Sommaire général

CHAPITRE I : Contexte et travaux ***1***

1.1 Introduction **1**

1.2 Les systèmes d'aide à la décision **2**

1.2.1 Entrepôts et magasins de données 2

1.2.2 Niveaux d'abstraction 3

1.2.2.1 Niveau conceptuel 3

1.2.2.2 Niveau logique 4

1.3 L'OLAP et le NoSQL **5**

1.3.1 Architecture distribuées 5

1.3.2 Les modèles NoSQL 6

1.3.3 Problématique de la thèse 6

1.4 Organisation de la thèse **7**

2. Chapitre II : Etat de l'art ***9***

2.1 Introduction **9**

2.2 Les entrepôts de données avec le système HDFS **10**

2.2.1 Environnement Hadoop 10

2.2.1.1 Présentation de HDFS 10

2.2.1.2 Le paradigme MapReduce 10

2.2.1.3 Exécution d'un traitement MapReduce avec Hadoop 11

2.2.2 Entrepôts de données sous HADOOP 11

2.3 Entrepôts de données avec les systèmes NoSQL **13**

2.3.1 Modèles NoSQL 13

2.3.1.1 Modèles orienté agrégats d'information 14

2.3.1.2 Modèles orienté graphes 17

2.3.1.3 Synthèse 18

2.3.2 Entrepôts de données sous NoSQL 18

2.3.2.1 Processus de traduction indirecte 20

2.3.2.2 Processus de traduction direct 21

2.3.3 Bilan 24

2.4 Panorama des solutions industrielles **24**

2.4.1 Les solutions clé-valeur 25

2.4.1.1 Voldemort 25

2.4.1.2 Riak 25

2.4.1.3 Redis 25

2.4.1.4 Memcahedb 25

2.4.1.5 Synthèse 26

2.4.2 Solutions orientées colonnes 26

2.4.2.1 Cassandra 26

2.4.2.2 HBase 27

2.4.2.3 Hypertable 27

2.4.2.4 Synthèse 27

2.4.3 Solutions orientées documents 28

2.4.3.1 MongoDB 28

2.4.3.2 CouchDB 28

2.4.3.3 SimpleDB 29
2.4.3.4 Terrastore 29
2.4.3.5 Synthèse 29
2.4.4 Solutions orientées graphes 30
2.4.5 Systèmes relationnels extensibles 30
2.4.5.1 MySQL Cluster 30
2.4.5.2 VoltDB 31
2.4.5.3 NuoDB 31
2.4.6 Synthèse des solutions industrielles : 31
2.5 Bilan 32
3. CHAPITRE III : Modélisation multidimensionnelle « Not only SQL » 33
3.1 Introduction 33
3.2 Modélisation conceptuelle multidimensionnelle 34
3.3 Modélisation logique Not-only-SQL 36
3.3.1 Modélisation multidimensionnelle orientée documents 36
3.3.1.1 Modèle NoSQL orienté documents 36
3.3.1.2 Processus de traduction plate en orienté documents 37
3.3.1.3 Processus de traduction par imbrication en orienté documents 39
3.3.1.4 Processus de traduction hybride en orienté documents 41
3.3.1.5 Processus de traduction éclatée en orienté documents 43
3.3.2 Modélisation multidimensionnelle orientée colonnes 46
3.3.2.1 Modèle NoSQL orienté colonnes 46
3.3.2.2 Processus de traduction plate en orienté colonnes 47
3.3.2.3 Processus de traduction par imbrication en orienté colonnes 49
3.3.2.4 Processus de traduction hybride en orienté colonnes 50
3.3.2.5 Processus de traduction éclatée en orienté colonnes 52
3.4 Optimisation par cube olap 55
3.4.1 Définition d'un cube OLAP 56
3.4.2 Processus de traduction en orienté documents 59
3.4.3 Processus de traduction en orienté colonnes 60
3.5 Bilan 60
4. CHAPITRE IV : Processus de Conversion Intra-modèles et Inter-modèles 64
4.1 Introduction 64
4.2 Processus de conversion intra-modèles 65
4.2.1 Conversions intra-modèles orientés documents 65
4.2.2 Conversions intra-modèles orientés colonnes 70
4.3 Processus de conversion inter-modèles 73
4.3.1 Conversions inter-modèles orientés documents et colonnes 73
4.3.2 Conversions inter-modèles des cubes OLAP 77
4.4 Bilan 79
5. Chapitre V : Environnement d'expérimentations 80
5.1 Introduction 80
5.2 Panorama Des bancs d'essai décisionnels 81
5.3 Le banc d'essai SSB 82

5.4 Le banc d'essai SSB+ 84
5.5 Amélioration de l'indicateur d'échelle 87
5.6 La distribution des données 88
5.7 Le jeu de requêtes 88
5.8 Commandes DBGenk 90
5.9 DBLoad : outil de chargement de données 90
5.10 Expérimentations 91
5.11 Bilan 95
6. CHAPITRE VI : Expérimentations et validations » 96
6.1 Introduction 96
6.2 Protocole expérimental 96
6.3 Instanciation des entrepôts de données orienté documents 98
6.3.1 Temps de chargement et espace de stockage 99
6.3.2 Mise à jour des données 100
6.3.2.1 Exemple de mise à jour dans MongoDB 101
6.3.3 Interrogations 101
6.3.3.1 Exemple d'agrégation pipeline dans MongoDB 103
6.3.4 Calcul du treillis 104
6.3.4.1 Calcul du cuboïde classique 105
6.3.4.2 Calcul des cuboïdes étendus 106
6.3.5 Conversion intra-modèles 112
6.3.6 Discussion 113
6.4 Comparaison entre modèle relationnel et modèle orienté documents 114
6.4.1 Modèle orienté document : Comparaison avec le modèle relationnel 115
6.4.2 Construction du treillis d'agrégats 117
6.4.3 Discussion 118
6.5 Instanciation des entrepôts de données orienté Colonnes 118
6.5.1 Chargement des données 119
6.5.2 Calcul du treillis d'agrégats 119
6.5.3 Conversion intra-modèles 121
6.5.4 Discussion 122
6.6 Bilan 122
CHAPITRE VII : Conclusion 124
7.1 Conclusion générale 124
7.2 Perspectives 125

Sommaire des figures

Figure 1 Architecture classique des systèmes d'aide à la décision 1
Figure 2 Exemple de schéma en étoile 4
Figure 3 Exemple d'entrepots de donnees multidimentionnelles R-OLAP concernant des tweets 5
Figure 4 Nouvelle architecture des Système d'aide à la décision 11
Figure 5 Processus d'interrogation dans l'approche de [D'Orazio and Bimonte, 2010] 12
Figure 6 Exemple de dimension codée par des clés dans l'approche de [Yan et al 2015] 13
Figure 7 Principe du modèle orienté clé-valeur 15
Figure 8 Principe du modèle orienté documents 16
Figure 9 Principe du modèle orienté colonnes 17
Figure 10 Principe du modèle orienté graphes 17
Figure 11 Nouvelle architecture des systèmes d'aide à la décision intégrant le NoSQL 19
Figure 12 Processus de transformation des entrepôts de données multidimensionnelles du niveau conceptuel vers le niveau logique 19
Figure 13 Architecture Hive 21
Figure 14 Représentation logique orientée graphes selon [Castelltort and Laurent 2014] 23
Figure 15 Processus de conception 34
Figure 16 Exemple de schéma conceptuel en étoile 36
Figure 17 Exemple de document par traduction plate 39
Figure 18 Exemple de document par traduction imbriquée 41
Figure 19 Exemple de document par traduction hybride 43
Figure 20 Exemple de document par traduction éclatée 46
Figure 21 Exemple de ligne par traduction plate 48
Figure 22 Exemple de ligne par traduction imbriquée 50
Figure 23 Exemple de ligne par traduction hybride 52
Figure 24 Exemple de document par traduction éclatée 55
Figure 25 Exemple de cube OLAP par treillis de cuboïdes (ou pré-agrégats) 58
Figure 26 Exemple de matérialisation partielle du cube OLAP 58
Figure 27 Exemple de matérialisation partielle du cube OLAP 59
Figure 28 Processus de transformations conceptuelle - logique 64
Figure 29 Processus de conversions logique orienté documents 65
Figure 30 Exemple de conversion intra-modèle en orienté documents, du modèle imbriqué vers le modèle hybride 70
Figure 31 Processus de conversions logique orienté colonnes 71
Figure 32 Exemple de conversion intra-modèle en orienté colonnes, du modèle hybride vers le modèle plat 73
Figure 33 Exemple de conversion inter-modèles de l'orienté documents vers l'orienté colonnes, en implantation plate 75
Figure 34 Exemple de conversion inter-modèles de l'orienté colonnes vers l'orienté documents, d'une implantation hybride vers une implantation imbriquée 77
Figure 35 Processus d'utilisation du SSB dans le NoSQL 84
Figure 36 Le modèle conceptuel en étoile du SSB 84
Figure 37 Schéma de chargement des bases de données avec le canc d'essai SSB+ 85
Figure 38 Schéma de données pour une génération normalisée des données 86
Figure 39 Génération des données en mode distribué 88
Figure 40 Espace de stockage utilisé 93
Figure 41 Temps de génération par configuration 94
Figure 42 Temps de chargement par configuration 94
Figure 43 Architecture cluster 97
Figure 45 Temps d'exécutions moyen par implantation 102
Figure 46 Treillis d'agrégats 105

Figure 47 Treillis d'agrégats avec le temps de calcul (en secondes) et la taille (en enregistrement/ documents). Le nom des dimensions est abrégé (D : Date, P : Part, S : Supplier, C : Customer) 106
Figure 48 Exemple de cube OLAP par treillis de cuboïdes (ou pré-agrégats) 107
Figure 49 Comparaison des temps d'exécution des cuboïdes classique et imbriqué 111
Figure 50 Comparaison des temps de réponse des cuboïdes classique et détaillé 112
Figure 51 temps de conversion intra-modèle orienté colonnes avec SF1 113
Figure 52 Espace de stockage par implantation et par indicateur d'échelle entre le modèle orienté documents et le modèle relationnel 116
Figure 53 Temps de chargement par modèle 116
Figure 54 Temps de calcul et nombre de lignes pour chaque cuboïde (les lettres correspondant au nom de chaque de dimension : C=Customer, S=Supplier, D=Date, P=Part/Product) 120
Figure 55 Temps de conversion intra-modèle orienté colonnes (sf=1) 121

Sommaire des tableaux

Tableau 1 Comparatif des modèles NoSQL 18
Tableau 2 Comparatif des travaux de transformation directe des schémas conceptuels en NoSQL 24
Tableau 3 Comparatif des systèmes NoSQL clé-valeur 26
Tableau 4 Comparatif des systèmes NoSQL orientés colonnes 28
Tableau 5 Comparatif des systèmes NoSQL orientés documents 29
Tableau 6 : Synthèse des règles de traduction du modèle conceptuel multidimensionnel vers les modèles logiques NoSQL orientés documents et colonnes. 62
Tableau 7 Synthèse des règles de conversion intra-modèles orientés documents 66
Tableau 8 Synthèse des règles de conversion intra-modèles orientés colonnes 71
Tableau 9 Synthèse des règles de conversion inter-modèles 74
Tableau 10 Synthèse des règles de conversion inter-modèles des cubes OLAP 78
Tableau 11 Mémoire disque par ligne par table 87
Tableau 12 L'impact de l'indicateur d'échelle sur le nombre de lignes du SSB 88
Tableau 13 Filtres de requêtes 89
Tableau 14 Mémoire disque par configuration 92
Tableau 15 Temps d'exécution par configuration 93
Tableau 16 Temps de chargement et espace mémoire par implantation 100
Tableau 18 Temps d'exécution par requête et par implantation 103
Tableau 19 Temps d'exécution moyen par cuboïde 106
Tableau 20 Temps d'exécution et mémoire utilisés par dimensions pour chaque cuboïde 109
Tableau 21 Temps d'exécution par modèle 117
Tableau 22 Temps d'exécution de cuboïdes à trois dimensions 118
Tableau 23 Temps de chargement de données par implantation 119
Tableau 24 Temps d'exécution par cuboïde et par modèle 121

CHAPITRE I : CONTEXTE ET TRAVAUX

1.1 INTRODUCTION

Les systèmes d'aide à la décision occupent une place prépondérante au sein des entreprises et des grandes organisations, pour permettre des analyses dédiées à la prise de décisions [Kimball and Ross 2011]. Ces systèmes sont généralement constitués en trois couches [Teste 2009] [Bimonte and Pinet 2012] comme l'illustre la Figure 1.

- Une première couche correspond à l'extraction, la transformation et le chargement des données provenant de sources de données ; on parle de couche ETL (*extract, transform, load*) [Vassiliadis et al. 2002]. Le plus souvent ces sources de données sont issues des applications de production de l'organisation.
- Une deuxième couche est constituée d'espaces de stockage pour entreposer (*data warehouses*) et préparer (*data marts*) les données collectées afin de supporter efficacement les analyses.
- Enfin, la troisième couche est dédiée à la restitution (*reporting*) et l'analyse de ces données. Différents outils d'interrogation, et de visualisation sont possiblement utilisés.

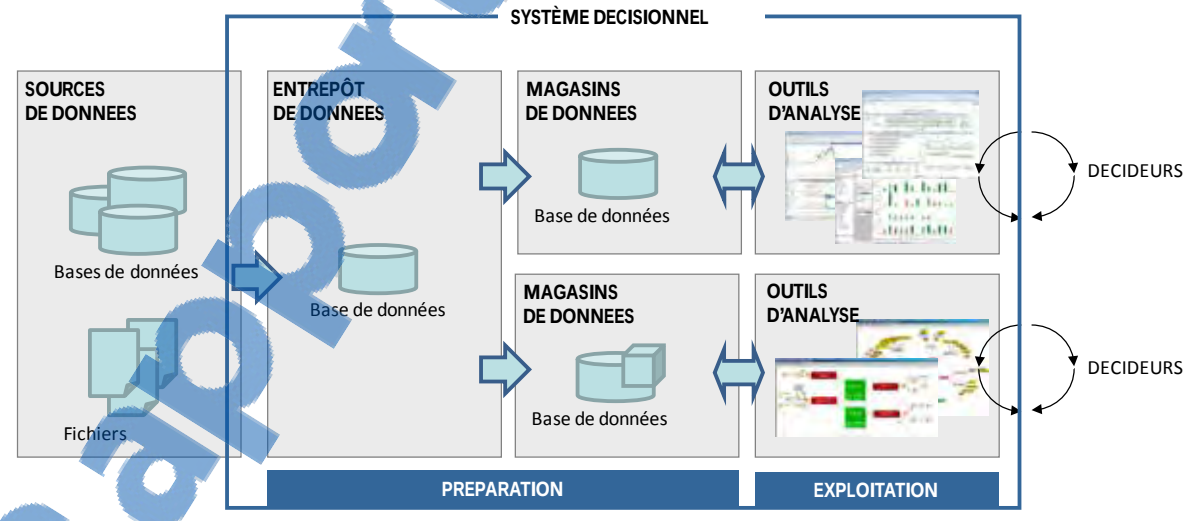


Figure 1 Architecture classique des systèmes d'aide à la décision

Avec la généralisation des technologies de l'information au travers de réseaux mondialisés, la diffusion massive de moyens de communications mobiles, et le développement d'objets autonomes connectés, l'humanité produit des quantités de données

numérisées dans des proportions et avec un rythme sans commune mesure avec le passé. Ce nouvel environnement, connu sous le nom de *big data* (ou mégadonnées) remet en cause les approches traditionnelles des systèmes d'aide à la décision [Stonebraker et al. 2007] [Abadi et al. 2016]. Les masses de données disponibles aujourd'hui représentent des volumes disparates difficilement accessibles aux méthodes et outils traditionnels de collecte, de stockage, d'analyse et de restitution.

L'objet d'étude de cette thèse concerne la modélisation des entrepôts de données avec des systèmes *not-only-SQL* (NoSQL) [Moniruzzaman and Hossain 2013]. Ces systèmes sont développés actuellement pour faire face aux importantes volumétries que les systèmes de gestion de données doivent prendre en charge. Nous étudions les problèmes de modélisation et de transformations des données décisionnelles avec ces nouveaux systèmes [Teste 2000].

1.2 LES SYSTEMES D'AIDE A LA DECISION

Généralement, un système d'aide à la décision est constitué de données entreposées. Les architectures classiques reposent sur deux catégories d'espaces de stockage :

- L'entrepôt de données qui héberge les données de manière centralisée et uniforme. Il constitue un premier niveau de stockage favorisant la collecte et la gestion historisée (conservation de l'évolution des données collectées) des données.
- Les magasins de données constituent un second niveau du stockage utilisé à des fins d'analyse. Généralement, un magasin de données est dédié à un domaine métier ou une catégorie d'analyses. Les données sont organisées selon une modélisation multidimensionnelle [Kimball and Ross 2011] afin de supporter efficacement les processus d'analyses en ligne (*on-line analytic processing*, OLAP).

1.2.1 Entrepôts et magasins de données

La notion d'entrepôts de données a été introduite pour la première fois par Bill Inmon [Inmon 1995] [Inmon 2005]. Il constitue une base de données intégrées, contenant des informations historisées, non volatiles et destinées à la prise de décision.

Définition : Un **entrepôt de données** est une collection de données intégrées, variant selon le temps et non volatiles, qui sert de support au processus de prise de décision.

Les informations d'un entrepôt de données sont :

- **Intégrées** : les différentes données concernant les métiers et les services de l'entreprise sont centralisées et uniformisées de manière cohérente.
- **Variant dans le temps** (ou historisées). Les informations contenues dans un entrepôt de données sont identifiées par des périodes temporelles. Les évolutions de ces données peuvent être conservées au cours du temps.
- **Non volatiles**. Un état de stabilité est obligatoire pour permettre une traçabilité des décisions prises. Les données matérialisées dans un entrepôt ne sont généralement ni modifiées, ni supprimées.

Les données d'un entrepôt de données sont structurées en fonction des besoins analytiques ; par exemple les besoins d'analyse du service financier ne concernent que les données liées à ce dernier, d'où l'utilisation de magasins de données **orientés sujets**. Un magasin de données est un sous-ensemble de l'entrepôt de données.

Définition. Un **magasin de données** est un extrait orienté sujet de l'entrepôt, organisé selon un modelé adapté (multidimensionnel) aux outils d'analyse et d'interrogation décisionnelle.

1.2.2 Niveaux d'abstraction

Concevoir un système décisionnel nécessite une phase de modélisation des données multidimensionnelles. Plusieurs approches ont été proposées selon trois niveaux d'abstraction :

- **Conceptuel**. Ce niveau d'abstraction consiste à représenter l'espace de données multidimensionnelles indépendamment des techniques informatiques.
- **Logique**. Ce niveau d'abstraction désigne une technique de modélisation (relationnel, objet, NoSQL, etc).
- **Physique**. Ce niveau d'abstraction correspond à un logiciel particulier choisi dans la technologie logique adoptée (Oracle, PostgresSQL, MongoDB, HBase…).

Nous détaillons les deux niveaux d'abstraction conceptuel et logique sur lesquels nos travaux de thèse sont focalisés.

1.2.2.1 Niveau conceptuel

Différents concepts sont définis pour représenter les données multidimensionnelles. Les sujets d'analyse (appelés faits), regroupent un ensemble d'indicateurs (appelés mesures). Les valeurs de ces indicateurs sont observables selon des axes d'analyse (appelés dimensions). Ces dimensions sont constituées de différents niveaux de détail, eux-mêmes organisés en hiérarchies ; par exemple, nous pourrions analyser le fait *ventes* au travers d'une mesure *montant*, ces montants pouvant être observés en fonction d'une dimension *temps* constituée de trois niveaux de détails (*jour*, *mois*, *année*) organisés au sein d'une hiérarchie définissant le jour comme un niveau de détail inférieur au mois, lui-même inférieur à l'année.

Ces différents concepts permettent de concevoir des schémas multidimensionnels, appelés constellation. Les dimensions peuvent ainsi être partagées entre les faits. Un cas particulier consiste à ramener la constellation à un seul fait, on parle alors de schéma en étoile (*star schema*) [Kimball and Ross 2011]. La Figure 2 présente un exemple de schéma en étoile en utilisant le formalisme graphique développé par notre équipe [Ravat et al. 2007] [Ravat et al. 2009].

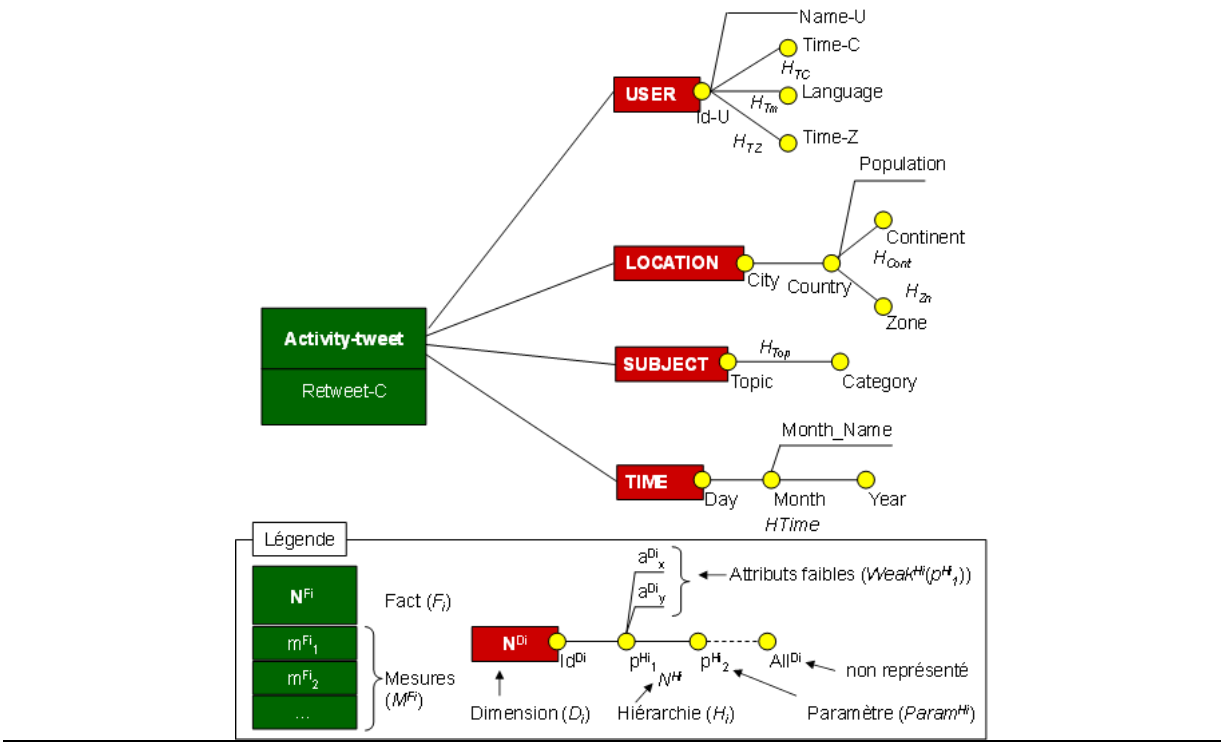


Figure 2 Exemple de schéma en étoile

1.2.2.2 Niveau logique

Plusieurs modèles logiques ont été proposés pour convertir les schémas en constellation.

- **L'approche R-OLAP** consiste à utiliser le modèle relationnel pour représenter un schéma en constellation [Vassiliadis and Sellis 1999] [Morfonios et al. 2007]. Elle est de loin l'approche la plus utilisée. Ce modèle traduit chaque fait dans une table appelée ***table de fait.*** Chaque dimension est traduite dans une table appelée ***table de dimension***. Dans la table de fait on retrouve les attributs représentants les mesures d'activités et les attributs les clés étrangères permettant la relation avec chacune des tables de dimensions. La table de dimension est constituée des paramètres et de la clé primaire (il est possible de normaliser les tables de dimensions constituant ainsi un schéma en flocon).
- **L'approche M-OLAP** consiste à utiliser un système dédié où les données sont organisées sous forme de tableaux multidimensionnels formant des hypercubes de données. Chaque arrête de l'hypercube correspond à une dimension et les cellules correspondent au fait.
- **L'approche H-OLAP** vise à cumuler les avantages des deux approches précédentes. Les données agrégées sont stockées sous formes multidimensionnelles tandis que les données détaillées sont stockées dans des structures relationnelles.

Exemple. Nous illustrons ci-dessous un exemple basé sur l'approche la plus connue, R-OLAP. Dans cet exemple, on observe le fait *Tweet* décrit selon quatre dimensions (*Time, Subject, Location* et *User*). La table *Tweet* contient des mesures et les clés étrangères pour référencer les tables des dimensions.

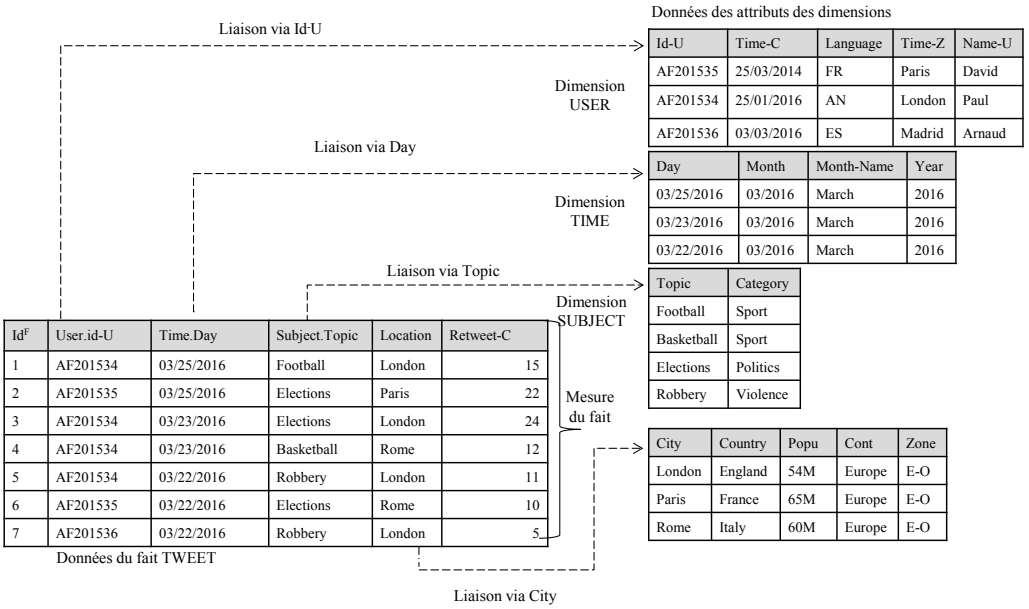


Figure 3 Exemple d'entrepôts de données multidimentionnelles R-OLAP concernant des tweets

1.3 L'OLAP ET LE NOSQL

De nos jours, l'augmentation du volume des données atteint des proportions critiques. Ces données massives remettent en cause les approches traditionnelles des entrepôts de données. En effet, les solutions actuelles de construction des entrepôts de données, reposent essentiellement sur ces Systèmes de Gestion de Bases de Données Relationnelles (SGBDR) [Codd 1970]. Ces approches s'avèrent difficilement extensibles aux mégadonnées.

1.3.1 Architectures distribuées

Avec l'essor des grandes plateformes Web (e.g. Google, Facebook, Twitter, Amazon), ont été développées ces dernières années des solutions de gestion des mégadonnées 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 à la volumétrie des données collectées. Fonctionnellement, cette solution se traduit par :

- Une distribution du stockage : Une répartition des données sur un ensemble de machines appelés nœuds, l'ensemble des nœuds constituant un cluster. La distribution doit aussi assurer un mécanisme de disponibilité continue des services.
- Une distribution du traitement : Une distribution du traitement sur l'ensemble des machines de sorte à réduire le temps de réponse.

De ce fait, il est envisageable de construire des entrepôts de données massives reposant sur ce principe de scalabilité de l'espace de stockage [Cuzzocrea et al. 2013] [Bellatreche et al. 2015]. La chute du coût de stockage depuis les années 2000 (moins d'un euro le gigaoctet) en fait une solution capable d'absorber de très gros volumes de données à moindre coût. Ce type d'architecture distribuée a connu récemment le développement de systèmes de gestion de fichiers massivement distribués (le plus connu est probablement Hadoop [Anderson et al. 2010]) et de nouvelles techniques de parallélisation massive des traitements (Map/Reduce [Dewitt and Stonebraker 2008]).

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.

1.3.2 Les modèles NoSQL

Les modèles NoSQL sont de nouveaux moteurs de stockage qui emploient une architecture distribuée capable de traiter de gros volume de données. Ils présentent une nouvelle alternative à l'approche classique R-OLAP pour l'entreposage des données multidimensionnelles.

Les systèmes NoSQL peuvent être classés en quatre catégories, correspondant à différents paradigmes de modélisation [Morfonios et al. 2007a].

- Le modèle clé-valeur [Dey et al. 2013] consiste à modéliser les données de manière très déstructurée par une clé servant à identifier les données et une valeur. Les structures soujacentes de la valeur ne sont pas gérées par le système ; elles sont renvoyées à la charge de l'application cliente.

- Les trois autres modèles introduisent des éléments complémentaires pour mieux structurer la valeur, au niveau du système de gestion de données, et permettre la mise en place d'index. Le modèle orienté documents [Chodorow 2013] repose sur une structuration horizontale, par imbrication de données, appelées documents. De manière orthogonale, le modèle orienté colonnes [Cattell 2011] [Moniruzzaman and Hossain 2013] structure les données verticalement, par regroupement de colonnes en familles de colonnes. Enfin, le modèle orienté graphes [Holzschuher and Peinl 2013] est spécialisé dans la structuration de données en noeuds et relations formant un graphe.

Ces systèmes ont pour caractéristique commune le principe de. Ce principe consiste à ne plus avoir un schéma commun à un ensemble de données au sein d'une même structure [Scherzinger et al. 2013] [Störl et al. 2015].Chaque donnée à son propre schéma, indépendamment des autres. Par conséquent la notion des contraintes d'intégrité sur les structures de données n'est plus ou peu assurée par ces systèmes.

1.3.3 Problématique de la thèse

Ces nouvelles approches constituent une voie intéressante pour la construction des entrepôts de données multidimensionnelles capables de supporter des grandes masses de données [Stonebraker et al. 2007] [Stonebraker 2012]. Toutefois la remise en cause de l'approche R-OLAP nécessite de revisiter les principes de la modélisation des entrepôts de données multidimensionnelles.

1. **Processus d'implantation NoSQL des schemas multidimentionnelles**. Pour pouvoir bénéficier des avantages des systèmes NoSQL, il est nécessaire de définir un processus d'implantation des entrepôts de données multidimensionnelles avec les modèles NoSQL. Ces systèmes réclament de nouvelles règles pour traduire de manière pertinente les faits, les dimensions et les hiérarchies dans les modèles NoSQL.
   En particulier, la distribution soujascente des données nécessite de revisiter les règles de traduction élaborées dans le contexte centralisé des entrepôts de

données.
Le contexte NoSQL rend également plus complexe le calcul efficace de pré-agrégats qui sont habituellement mis en place dans le contexte ROLAP (treillis) [Gray et al. 1997].

2. **Uniformisation de la modélisation NoSQL des entrepôts**. Les modèles NoSQL sont des solutions récentes qui sont destinées à répondre à des besoins spécifiques. Les modèles de représentation de ces systèmes ne sont pas standardisés, et reposent sur des concepts non uniformes à l'exception du simple principe clé/valeur. Un enjeu est donc de proposer une uniformisation des modèles de représentation pour permettre la définition homogène de processus de transformation entre modèles.
La grande flexibilité introduite au niveau des schémas dans les modèles NoSQL (*schemaless*) complexifie la définition de règles génériques pour la transformation d'une masse de données entre les niveaux conceptuel et logique.

L'étude menée dans cette thèse se focalise sur les approches NoSQL orientées documents et colonnes. Ces deux approches organisent les données de manière orthogonale, respectivement horizontalement et verticalement.

1.4 ORGANISATION DE LA THESE

Ce mémoire s'articule de la manière suivante.

Le **chapitre II** est consacré à l'état de l'art. Il commence par présenter l'architecture distribuée les modèles NoSQL. Ensuite la seconde partie du chapitre étudie les travaux qui se sont intéressés à l'entreposage de données dans un environnement NoSQL enfin, le chapitre finit par une présentation des solutions NoSQL industrielles.

Le **chapitre III**, s'intéresse à l'implantation des entrepôts de données dans le modèles orienté colonnes et orienté documents. La première partie de de chapitre définit des règles d'implantation pour transformer le modèle conceptuel multidimensionnel en constellation dans les deux modèles logiques (orienté colonnes et orienté documents). La seconde partie traite le processus de construction du treillis d'agréation dans ces deux modèles logiques NoSQL orienté colonnes et orienté documents.

Le **chapitre IV** étudie des processus de transformations des différentes implantations du chapitre 3. Deux processus sont discutés, le premier concerne des processus intra-modèles, c'est-à-dire des règles de passage d'une implantation à une autre implantation du même modèle logique NoSQL, tandis que le second processus détaille les règles de transformation d'une implantation d'un modèle logique vers une autre implantation d'un autre modèle logique.

Le **chapitre V** décrit l'environnement d'expérimentation, il s'agit d'un nouveau banc d'essai adapté et enrichi pour évaluer, en plus d'entrepôts de données relationnel, des entrepôts de données NoSQL orienté colonnes et orienté documents.

Le **Chapitre VI** valide nos travaux. IL présente l'ensemble des expérimentations pour valider la faisabilité des approches proposées.

Le **Chapitre VII** dresse un bilan général sur les contributions faites avec un regard sur les voies de recherches que nous envisageons d'emprunter par la suite.

2. CHAPITRE II : ETAT DE L'ART

Ce chapitre présente une étude de l'existant dans le domaine de l'implantation d'entrepôts de données multidimensionnelles avec les systèmes NoSQL.

2.1 INTRODUCTION

L'avènement des mégadonnées[1] (ou « big data ») a permis de faire émerger de nouvelles approches pour la gestion des grandes masses de données. Les grandes plateformes Web (e.g. Google, Facebook, Twitter, Amazon), ont développé ces dernières années des solutions de gestion des mégadonnées 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* [IBM et al. 2011], c'est à dire l'ajustement d'une manière progressive et continue de l'outil de stockage et des traitements à la volumétrie des données collectées.

Ces données massives remettent en cause les approches traditionnelles des entrepôts de données. En effet, les solutions actuelles de construction des entrepôts de données, reposent essentiellement sur des SGBDR ; on parle de l'approche R-OLAP. Ces approches s'avèrent difficilement extensibles aux mégadonnées.

Les bases de données relationnelles ont été mises en place pour répondre à des requêtes transactionnelles (OLTP) tandis que les entrepôts de données sont élaborés avec ces systèmes relationnels pour supporter des requêtes OLAP [Chaudhuri and Dayal 1997] [Colliat 1996]. Le modèle relationnel est basé sur des principes dont en particulier la gestion ACID des transactions, qui ont fait de lui l'approche dominante pour la gestion de données, mais l'handicape pour un passage à l'échelle nécessitant une distribution massive des données. Selon les principes édictés par Codd, le modèle relationnel se base sur une forme normalisée des données. L'avantage réside dans le fait de minimiser la redondance des données. Cependant l'éclatement des informations dans des structures normalisées induit l'utilisation de jointures pour les recomposer lors des interrogations, et peut nécessiter la mise en place de contraintes d'intégrité référentielles pour en assurer la cohérence. Ces mécanismes sont adaptés dans un contexte centralisé, mais peuvent limiter les performances du système en environnement massivement distribué. Les systèmes NoSQL renoncent à ces propriétés afin d'améliorer leur capacité de gestion et de traitement des données massives.

C'est dans ce contexte que nous assistons à l'apparition de nouveaux systèmes de gestion de fichiers distribués, tel que *Hadoop Distributed File System* (HDFS) [Lee et al. 2012a] [Pavlo et al. 2009] [Stonebraker and Cattell 2011] [Floratou et al. 2012], adaptés aux besoins de stockage massif de données. Dans le sillage de ces innovations, de nouveaux paradigmes de modélisation des bases de données apparaissent. On parle de modélisation NoSQL [Sadalage and Fowler 2012] [Cattell 2011a].

Ces nouveaux paradigmes permettent d'entrevoir le développement d'entrepôts de données multidimensionnelles capables de gérer des grandes masses de données par

[1] Mégadonnées : données massives ou « Big Data », Journal Officiel de la République Française, JORF du 22.08.2014

l'adoption d'approches NoSQL. Les entrepôts de données massives peuvent ainsi opter pour une architecture reposant sur ces principes de grande scalabilité.

Dans la suite de ce chapitre, nous étudions et analysons dans un premier temps les grands principes des systèmes de fichiers massivement distribués, puis dans un second temps les systèmes NoSQL, appelés aussi *data stores* [Stonebraker and Cattell 2011]. Nous étudierons en particulier les travaux concernant l'entreposage des données dans ces deux contextes.

2.2 LES ENTREPOTS DE DONNEES AVEC LE SYSTEME HDFS

2.2.1 Environnement Hadoop

Hadoop est un *framework open source* développé en java, faisant partie des projets de la fondation *Apache* depuis 2009. Il a été conçu pour :

- Stocker des volumes de données très importants ;
- Supporter des données de formats variés, structurées, semi- structurées ou non structurées.

Hadoop est basé sur un ensemble de machines formant un *cluster Hadoop*. Chaque machine est appelée *nœud*. C'est l'addition des capacités de stockage et traitement de ses nœuds qui lui assure un important système de stockage et une puissance de calcul. Le système de stockage est appelé HDFS (*Hadoop Distributed File System*) [Borthakur 2008]. La puissance de calcul repose sur le paradigme de programmation parallèle *MapReduce* [Dean and Ghemawat 2010].

2.2.1.1 Présentation de HDFS

HDFS est le composant chargé de stocker les données dans un cluster Hadoop. Basé sur une architecture « maître-esclave », le système HDFS repose dans son fonctionnement sur deux types de nœuds :

- Le *namenode* : le nœud maître est l'orchestrateur du système HDFS, au moyen de métadonnées. Il maintient l'arborescence de tous les fichiers du système et gère l'espace de nommage, autrement dit, il gère la correspondance entre un fichier et les blocs qui le constitue. Il prend en charge également la localisation des blocs des données dans le cluster. Il n'y a qu'un namenode par cluster HDFS. Le *secondary namenode* : il sert à effectuer à intervalle réguliers des sauvegardes du *namenode*. Il est utilisé pour prendre la relève en cas de panne du namenode.
- Les *datanode* : ils servent d'espace de stockage et de calcul des blocs de données.

Dans un cluster hadoop il y a donc une machine jouant le rôle de namenode, une autre machine sert de secondary namenode tandis que le reste des machines sont utilisées comme des datanodes.

2.2.1.2 Le paradigme MapReduce

Introduit en 2004 par Google, le paradigme MapReduce est un modèle de programmation parallèle développé spécifiquement pour lire, traiter et écrire des volumes de données très importants dans un environnement distribué. Google l'utilise pour gérer les gigantesques tâches portant sur des téraoctet- ou pétaoctet de données [O'Malley 2008].

Son principe de fonctionnent est très simple : il s'agit de décomposer une tâche en tâches plus petites, autrement dit, découper une tâche portant sur de très gros volumes de données en tâches identiques portant sur des sous-ensembles de ces données. La décomposition consiste à découper le volume de données initial en N volumes plus petits, qui seront manipulés séparément.

Le programme MapReduce est réalisé à partir de deux fonctions, Map et Reduce. La fonction Map découpe la tâche en un ensemble de petites tâches et les repartit sur l'ensemble des nœuds de sorte à ce que chaque petite tâche s'exécute sur le nœud qui héberge les données concernées. La fonction Reduce rassemble et agrège les résultats issus des fonctions Map parallélisées.

2.2.1.3 Exécution d'un traitement MapReduce avec Hadoop

Pour qu'un traitement MapReduce soit exécuté, Hadoop dispose d'un composant du coté *namenode* appelé *jobtracker,* chargé de superviser l'exécution complète du traitement sur les différents *datanodes*. Sur chaque *datanodes* autre composant appelé *tasktracker* supervise l'exécution de la tâche qui lui a été confiée.

Au démarrage le *jobtracker* reçoit une requête cliente Map, consulte le *namenode* pour avoir la liste des nœuds stockant les données répondant au traitement. Une fois la réponse reçue, le *jobtracker* assigne la fonction Map aux différents *tasktracker*. Les *tasktracker* cherchent les données correspondantes au traitement sur leur *datanode*. Une fois que les fonctions Map terminées, les résultats de celles-ci sont stockés au niveau de chaque *datanode*. Les résultats des Map sont agrégés par la fonction Reduce prise en charge par un ou plusieurs *datanodes*. Le résultat final est renvoyé au *datanode*.

2.2.2 Entrepôts de données sous HADOOP

La communauté scientifique des entrepôts de données s'est orientée vers l'utilisation de cette plateforme [Chaudhuri et al. 2011]. La Figure 4 montre comment l'architecture des systèmes décisionnels est étendue à l'utilisation de ce système distribué.

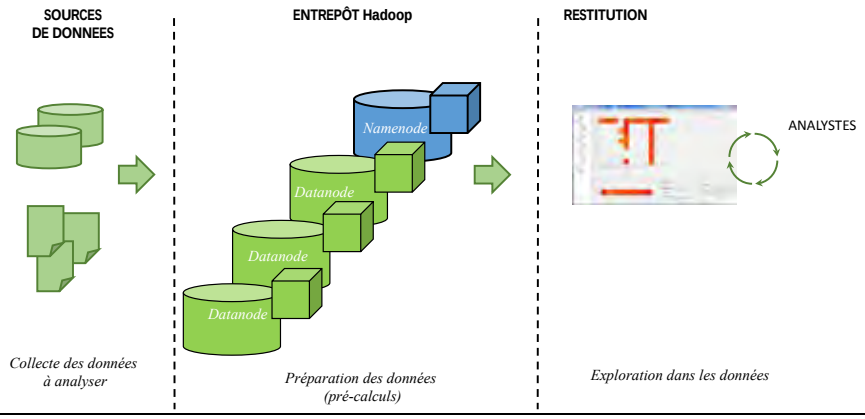


Figure 4 Nouvelle architecture des systèmes d'aide à la décision

Dans cette section nous étudions les travaux ayant exploité la plateforme Hadoop pour la construction de cube de données [Cao et al. 2011] [D'Orazio and Bimonte 2010] [Song et al. 2015]

[Cao et al. 2011] introduisent ES^2, une nouvelle plateforme distribuée basée sur le cloud pour supporter des analyses sur de gros volumes de données, en particulier les

transactions OLTP et les processus OLAP. Les données sont issues de sources différentes et des traitements OLAP et OLTP peuvent s'effectuer simultanément sur le même espace de stockage. L'accès aux données se fait à travers deux interfaces, une dédiée aux opérations OLAP et une autre dédiée aux opérations OLTP. Les opérations sont isolées grâce à la politique mise en place appelée « *locking mechanism* ». Par exemple, lorsqu'une requête est soumise, elle se voit affectée un certain temps *ts* d'accès aux données, durant lequel l'opération ne sera pas interrompue jusqu'à la fin de son exécution ; les autres requêtes attendront l'écoulement du temps *ts*. Les données sont stockées dans une base de données orientée colonnes.

Le but de cette approche est de bénéficier des avantages de MapReduce en termes de distribution des traitements et de la tolérance aux pannes, mais ne s'intéresse pas à la définition processus d'implantation des schémas multidimensionnels.

[D'Orazio and Bimonte 2010] proposent de stocker des mégadonnées dans un tableau multidimensionnel et d'utiliser le langage PIG [Olston et al. 2008] [Gates et al. 2009]. Dans cette approche, les données d'abord stockées dans un tableau multidimensionnel, sont traduites dans des fichiers pour une analyse via le langage PIG. La traduction est assurée par un algorithme qui produit un fichier temporaire PIG supprimé après analyse (cf. Figure 5). Le fichier PIG est en un fichier constitué de tuples dénormalisés. L'interrogation se fait aussi en utilisant le langage d'interrogation PIG qui repose sur un plan d'exécution (composé de trois instructions) permettant de bonnes performances. Par exemple pour une requête OLAP classique calculant '*le gain par région entre 1990 et 1991*', PIG effectue une sélection sur les dimensions membres (1990-1991), une autre instruction effectue un groupement par année et par région. La dernière instruction est utilisée pour effectuer l'agrégation, la somme dans cet exemple.

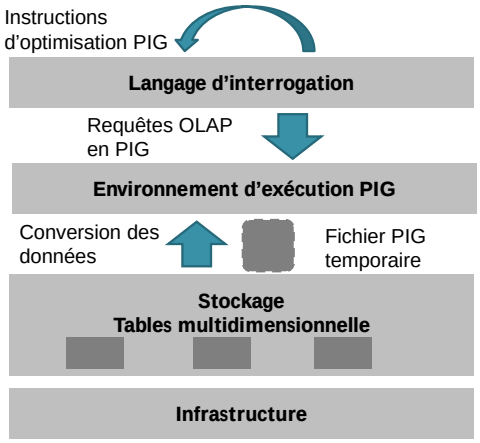


Figure 5 Processus d'interrogation dans l'approche de [D'Orazio and Bimonte, 2010]

Le but de cette approche n'est pas de définir un processus d'implantation mais de réduire le coût des infrastructures cloud [Ahirrao and Ingle 2015] [Agrawal et al. 2011] en utilisant un haut langage procédural dédié à la gestion des données massives.

Une autre approche proposée par [Song et al. 2015] consiste à construire un cube OLAP selon l'approche MOLAP et le stocker dans l'espace de stockage HDFS. Les dimensions sont représentées par des codes entiers dans des fichiers HDFS, chaque valeur est codée par une clé. Par exemple dans la dimension *temps*, la valeur 1990 de l'attribut *année* est codée par un zéro (*cf*. Figure 6), ce qui permet de former le couple <0-1990>, ainsi la position de la cellule est rapidement déterminée. Les clés des valeurs sont maintenues dans un fichier

de métadonnées, consulté, à la réception de la requête. Pour déterminer un niveau hiérarchique autre que la racine, un algorithme est mis en place, qui permet de calculer le niveau hiérarchique de la dimension. L'ensemble des valeurs des dimensions est représenté sous forme d'un arbre comme illustré dans la Figure 6.

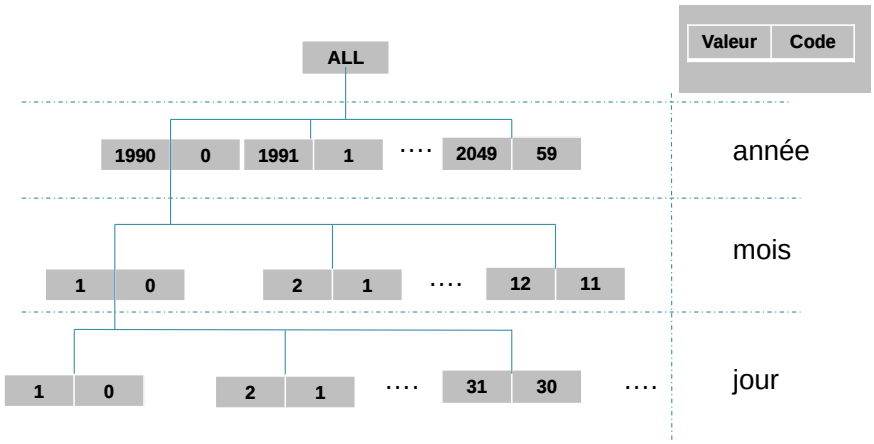


Figure 6 Exemple de dimension codée par des clés dans l'approche de [Yan et al 2015]

En outre, les auteurs exploitent la notion de partition (*chunk*) pour accélérer l'accès aux données : une partition est un ensemble de données agrégées. Un fichier de métadonnées est mis en place pour déterminer l'emplacement de l'agrégat.

L'avantage de cette approche est qu'elle exploite bien les propriétés offertes par le paradigme MapReduce, et l'espace de stockage HDFS, ce qui lui permet de renoncer à l'utilisation des techniques d'optimisations telles que l'utilisation des indexes et le treillis d'agrégats. En revanche, cette approche souffre de quelques inconvénients : elle nécessite un important espace de stockage et une phase couteuse pour bien déterminer la politique des clés à mettre en place. En outre, il faut maintenir à jour les fichiers de métadonnées des dimensions puisqu'une valeur codée avec une mauvaise clé fausse le calcul et détermine une cellule erronée. Par ailleurs l'utilisation du HDFS comme espace de stockage contraint l'utilisateur à coder lui-même ses requêtes.

Synthèse. Il existe donc des travaux qui permettent la construction d'entrepôts de données massives. Ces travaux sont basés sur le paradigme MapReduce et utilise le système HDFS pour stocker les données dans des fichiers. L'interrogation des données se fait via des fonctions codées par l'utilisateur faute de langage d'interrogation dédié. Par ailleurs, comme pointé par [Dewitt and Stonebraker 2008], HDFS est conçu pour des manipulations des données avec une seule passe, d'où par exemple, le recours à l'utilisation de chaînes MapReduce.

En outre, dans ces travaux, les auteurs ne s'intéressent pas à la définition de processus d'implantation du modèle conceptuel multidimensionnel dans le modèle logique NoSQL. Ils se limitent à un stockage dans HDFS.

2.3 ENTREPOTS DE DONNEES AVEC LES SYSTEMES NOSQL

2.3.1 Modèles NoSQL

Le terme NoSQL a été utilisé pour la première en 2009 lors d'une rencontre sur les bases de données distribuées [Cattell 2011a] [Leavitt 2010] [Han et al. 2011]. Ce terme désigne une

nouvelle approche des bases de données basée sur une performance capable de gérer une grande quantité d'informations, d'assurer une haute disponibilité de service et d'avoir une bonne scalabilité pour gérer la montée en charge. Le NoSQL adopte une approche non relationnelle, son développement a été marqué par l'abandon ou le compromis fait sur les propriétés ACID des bases de données relationnelles, qualifiées jusqu'ici comme un obstacle à la scalabilité horizontale.

Un nombre important de solutions NoSQL s'est développé. Il existe différentes manières de structurer les données mais l'ensemble des solutions développées peut être divisé en quatre modèles : le modèle orienté clé-valeur, modèle orienté colonnes, le modèle orienté documents et le modèle orienté graphes. A leur débuts, ces approches ont été développées en relâchant les principes du relationnel. Le NoSQL est considéré actuellement comme une solution complémentaire. Une différence importante concerne le modèle de données adopté par chacune de ces solutions [Sadalage and Fowler 2012]. Pour distinguer ces modèles, E. Evans dans [Evans 2004] définit la notion d'agrégat comme une unité d'information identifiée par une racine. Cet agrégat est l'objet de la modélisation. Les modèles orienté clé-valeur, orienté colonnes et orienté documents peuvent modéliser directement un agrégat d'informations, tandis que modèle orienté graphe ne le permet pas.

Dans ce qui suit nous décrivons ces différents modèles en distinguant les modèles supportant les agrégats d'information et le modèle orienté graphes.

2.3.1.1 Modèles orienté agrégats d'information

Modèle orienté clé-valeur

Le modèle clé-valeur [Cattell 2011] [Dey et al. 2013] considère chaque enregistrement comme une clé associée à une valeur. Cette approche s'assimile à un tableau associatif de deux colonnes où la clé identifie de manière unique la valeur associée. Contrairement au modèle relationnel, la taille et le type de la valeur ne sont pas déterminés au préalable. Le cas du modèle clé-valeur est plus flexible, la valeur de la clé peut changer d'un enregistrement à l'autre, elle peut à chaque ligne avoir un type et une taille différente.

Une seconde caractéristique du modèle clé-valeur réside dans la modularité des clés, qui peuvent être générées automatiquement ou construites selon une certaine logique. La manière dont la clé peut être construite est laissé libre ; le développeur peut imaginer diverses méthodes de génération de clés pour faciliter sa recherche. Si l'on reprend l'exemple de la base de données relatives aux Tweets présentée au chapitre précédent, nous pouvons construire des clés basées sur une concaténation *Id-U* et *Country*. Cette clé permet de rechercher plus rapidement tous les utilisateurs d'un pays donné ayant émis un Tweet. La clé peut être étendue. Si l'on poursuit l'exemple précédent, pour rechercher efficacement tous les utilisateurs d'un pays donné en fonction du temps, nous pouvons utiliser *Id-U*, *Country* et *Day*.

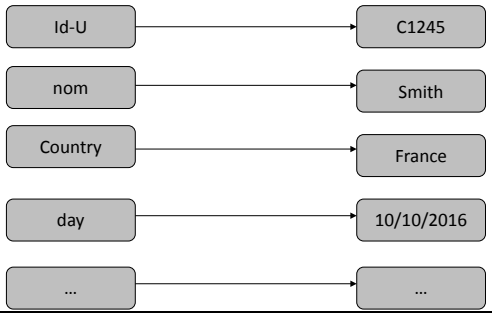


Figure 7 Principe du modèle orienté clé-valeur

L'organisation de données en paires clé-valeur permet de bonnes performances. Les clés sont nativement indexées. La communication avec la base de données est limitée aux opérations de bases, désignées par l'acronyme CRUD (*create, read, update, delete*). En revanche, pour des recherches à partir d'autres critères, notamment des parties de la valeur, les performances sont impactées. L'absence de structuration de la valeur oblige l'utilisateur à programmer l'accès aux valeurs dans un langage externe au système. On parle de « *user-defined function* (UDF) ».

En réponse à cette limite, d'autres modèles sont proposés. Il s'agit du modèle orienté colonnes et du modèle orienté documents. Ces deux modèles structurent la valeur afin de pouvoir mieux manipuler cette dernière.

Modèle orienté documents

Selon le même principe que le modèle orienté clé-valeur, le modèle orienté documents est constitué de paires clé-valeur. La clé identifie 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. Une valeur peut être également une référence vers un autre document. L'ensemble des documents est contenu dans une collection qui correspond à la notion de table en relationnel, et suit un stockage physique horizontal.

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) [Crockford 2006]. Il facilite la représentation de données structurées notamment d'une manière hiérarchique. D'autres formats de représentation sont possibles tels que le XML.

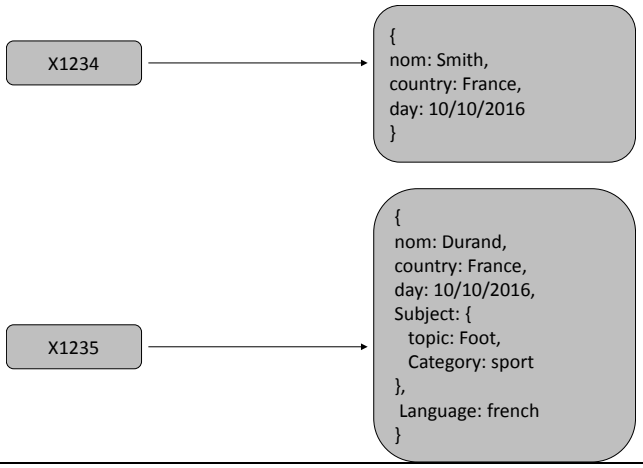


Figure 8 Principe du modèle orienté documents

Dans l'approche NoSQL orientée documents, le schéma n'est pas établi à l'avance, chaque document peut avoir sa propre structure, on parle de *dynamic schema* [Scherzinger et al. 2013]. Nativement, l'enregistrement des valeurs dans un document n'est soumis à aucun contrôle du système. Cependant, il est préférable de garder une structure minimale commune afin de faciliter la manipulation des données.

Contrairement au modèle clé-valeur, il est possible de manipuler directement les éléments constituant une valeur donnée. Par exemple, il devient possible de manipuler une sous-partie d'un document représentant un utilisateur comme son nom ou son adresse sans devoir extraire toute la valeur. De plus, le modèle orienté documents permet l'indexation élargie à d'autres attributs (autres que la clé) ce qui peut améliorer les performances de l'interrogation.

Modèle orienté colonnes

Le modèle orienté colonnes [Chang et al. 2008] [Aniceto et al. 2015] est la seconde forme évoluée du modèle clé-valeur. Dans les bases de données relationnelles, la structure des données est déterminée en avance par le schéma de la relation, avec un nombre limité de colonnes, chacune similaire pour tous les enregistrements (« n-uplets »). Le modèle orienté colonnes fournit un schéma flexible (colonnes non typées) pouvant varier entre chaque enregistrement (chaque ligne). La flexibilité d'une base NoSQL orientée colonnes permet de gérer l'absence de certaines colonnes entre les différentes lignes de la table.

Une base de données orientée colonnes est un ensemble de tables qui sont définies ligne par ligne, mais dont le stockage physique est organisé verticalement par groupe de colonnes, appelés « familles de colonnes ». Une famille de colonnes peut contenir un très grand nombre de colonnes [George 2011]. Le nombre de colonnes peut varier d'une ligne à l'autre ; chaque famille de colonnes s'apparente à un ensemble clé-valeur où la clé est vue comme une colonne associée à une valeur.

Un autre avantage des modèles orientés colonnes est la possibilité de stocker plusieurs valeurs « historisées » (versionning). Pour illustrer cet avantage, nous pouvons nous appuyer sur un exemple d'actualité en France ; la répartition géographique des régions de France a été modifiée et cela nécessite une évolution des données. Dans le modèle relationnel, la solution passe par la création d'une nouvelle table avec la nouvelle répartition et maintenir en parallèle les deux tables. Dans le modèle orienté colonnes il est possible d'insérer les nouvelles

données et maintenir un numéro de version par valeur [Ravat and Teste 2000] [Ravat et al. 2006]

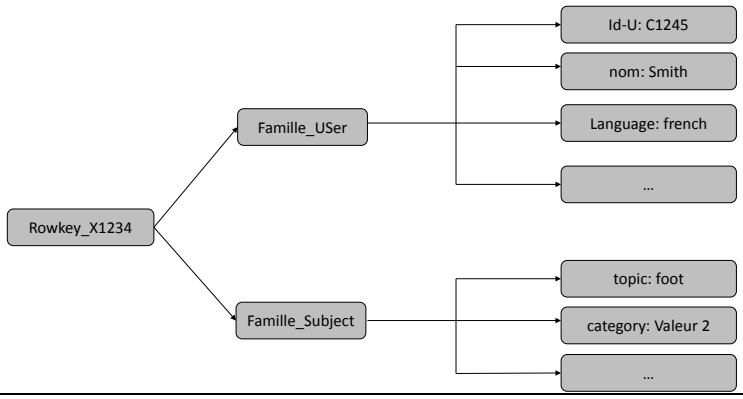


Figure 9 Principe du modèle orienté colonnes

2.3.1.2 Modèles orienté graphes

Le modèle orienté graphes se distingue des modèles précédents car il ne peut prendre en compte nativement la notion d'agrégats d'information proposée par [Evans 2004]. Le modèle orientés graphes est fondé sur la théorie des graphes. Le modèle orienté graphes repose sur trois notions ; nœud, relation et propriété. Chaque nœud possède des propriétés. Les relations relient les nœuds et possèdent éventuellement des propriétés. Ce type d'approche facilite les requêtes navigationnelles entre les nœuds en suivant les relations qui les relient : chaque nœud a un pointeur physique vers les nœuds voisins permettant une recherche locale rapide.

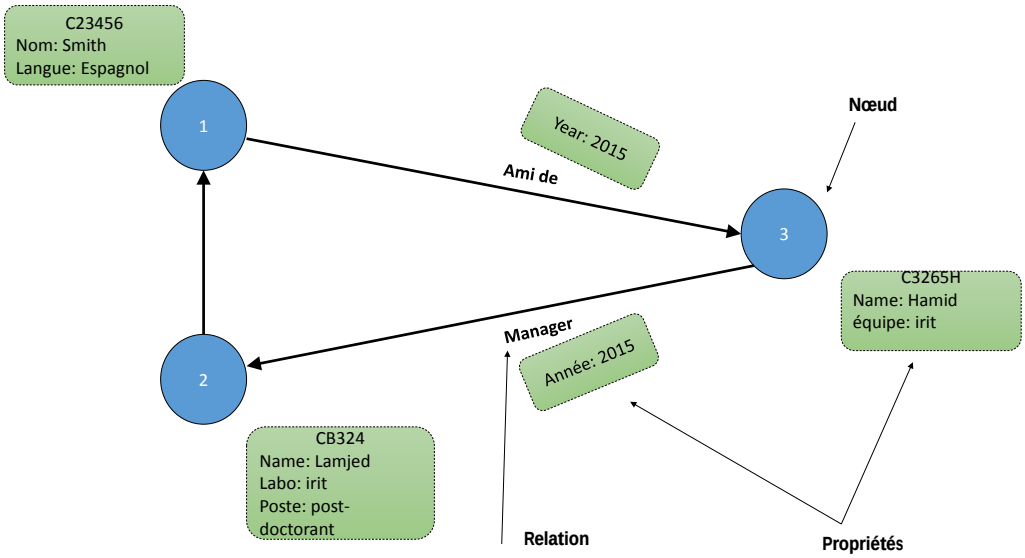


Figure 10 Principe du modèle orienté graphes

La structure du modèle orienté graphes est très adaptée pour répondre à des problématiques telles que la gestion d'un réseau social d'une entreprise ou tout autre stockage nécessitant le parcours de graphes.

Le modèle orienté graphe se caractérise lui aussi par une flexibilité de schéma ; il n'est pas nécessaire de créer un schéma au préalable pour les nœuds et les relations.

2.3.1.3 Synthèse

Le modèle clé-valeur est le modèle NoSQL fondamental. Il bénéficie d'une structure simple qui lui permet un gain de performances importantes. Cependant il ne permet pas une manipulation fine de la valeur. Cette limite a motivé le développent de nouveaux modèles orientés colonnes et orienté documents, qui peuvent être considérés comme une forme évoluée du modèle orienté clé-valeur. Ces deux modèles introduisent une structuration de la valeur, mais selon des principes orthogonaux. La valeur peut ainsi être soit atomique, soit composée. Ces modèles se distinguent par le stockage des données qui est effectué soit par document (horizontal), soit par ligne décomposée en familles de colonnes (vertical). Un quatrième modèle NoSQL repose sur le modèle orienté graphes. Il se caractérise par une structure basée sur la théorie des graphes étiquetés.

Dans le Tableau 1, nous synthétisons les caractéristiques de ces quatre modèles NoSQL ainsi que le modèle relationnel. Nous considérons 3 caractéristiques principales :

- La structure de la valeur peut être soit atomique (plus petite valeur manipulable) soit composée (valeur composée de plusieurs éléments manipulables).
- Le schéma peut être statique (schéma unique pour un ensemble d'enregistrements) ou dynamique (schéma différent pour chaque enregistrement).
- Le requêtage fournit des fonctions d'accès aux valeurs. L'acronyme CRUD représente les fonctions élémentaires de tous systèmes de gestion de données (écriture, lecture, modification, suppression). Il peut être complété par des fonctions avancées développées dans un langage propriétaire du système comme par exemple SQL, HQJ et CQL.

Tableau 1 Comparatif des modèles NoSQL

	Relationnel	**Clé-valeur**	**Colonnes**	**Documents**	**Graphes**
Structure	atomique	atomique	atomique & composée	atomique & composée	atomique & composée
Schéma	statique	dynamique	dynamique (colonnes)	dynamique	dynamique
Requêtage	SQL	CRUD	CRUD + langage	CRUD + langage	CRUD + langage
Exemple de systèmes	Oracle, PostgresSQL	Redis	HBase, Cassandra	MongoDB, CouchDB	Neo4j, OrientDB

2.3.2 Entrepôts de données sous NoSQL

Dans cette section, nous présentons les travaux de recherche portant sur le développement des entrepôts de données avec ces nouveaux systèmes (cf. Figure 11).

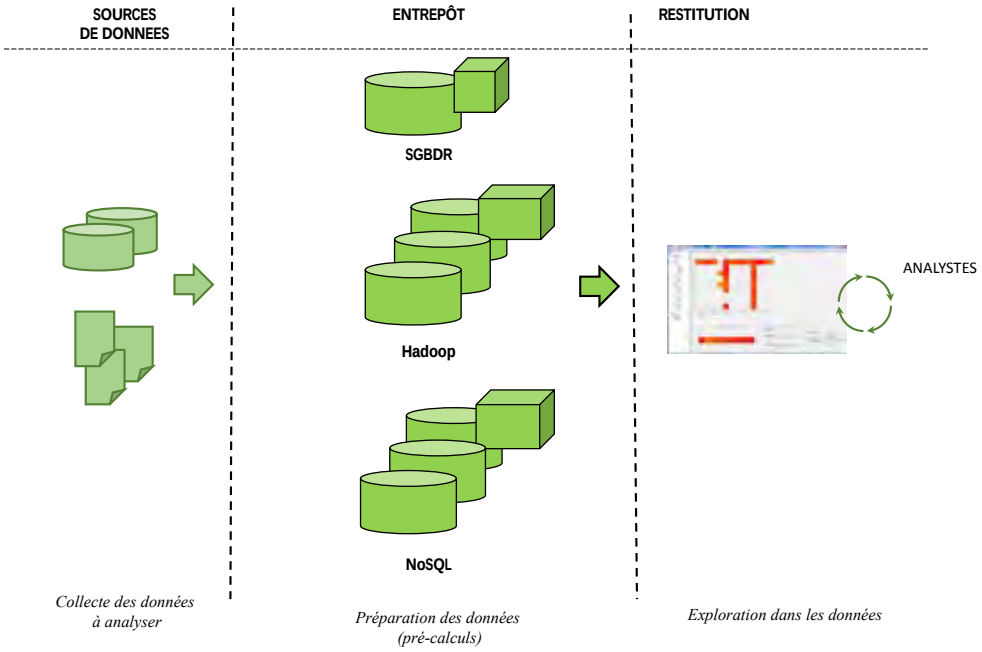


Figure 11 Nouvelle architecture des systèmes d'aide à la décision intégrant le NoSQL

Nous étudions tout particulièrement les travaux visant à implanter les modèles multidimensionnels du niveau conceptuel à l'aide les modèles NoSQL [Dehdouh et al., 2015] [Chevalier et al., 2015. Ils se résument en deux catégories :

- La première catégorie consiste à traduire de manière indirecte les schémas multidimensionnels. Le processus réutilise les règles de traduction du conceptuel vers le R-OLAP [Morfonios et al. 2007a] [Kimball and Ross 2010], puis est étendu par de nouvelles règles permettant un passage du modèle intermédiaire R-OLAP vers le modèle NoSQL cible.
- La seconde catégorie consiste à traduire directement les schémas multidimensionnels du niveau conceptuel vers le modèle NoSQL cible.

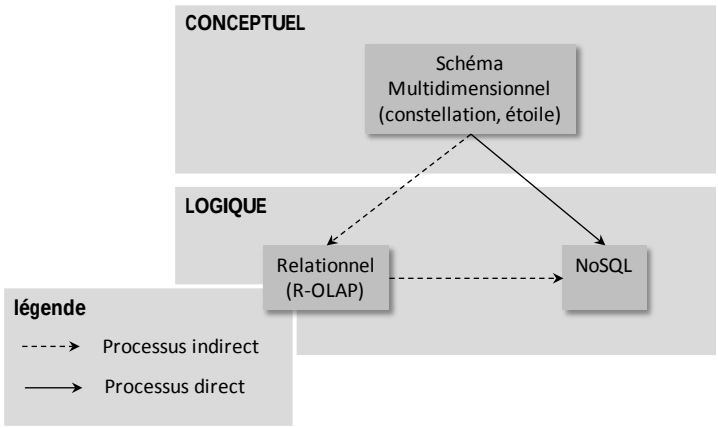


Figure 12 Processus de transformation des entrepôts de données multidimensionnelles du niveau conceptuel vers le niveau logique

2.3.2.1 Processus de traduction indirecte

De très nombreux travaux ont étudié la transformation des schémas conceptuels des entrepôts de données multidimensionnelles vers le modèle relationnel [Morfonios et al. 2007a] [Teste 2009]. Nous détaillons ici simplement les travaux qui se focalisent sur la transformation de schémas relationnels en NoSQL.

Les travaux proposant des processus de transformations des bases de données relationnelles en bases de données NoSQL, ciblent principalement les modèles orienté colonnes ou orienté documents.

[Steve Ataky et al. 2015] développent un framework utilisant le système HBase basé sur un modèle orienté colonnes. Le schéma relationnel est restructuré dans une unique table. Cette dernière comporte une famille de colonnes pour chaque table relationnelle. Une famille de colonnes additionnelle est utilisée pour conserver les clés des tuples des relations initiales. [Chongxin Li, 2010] propose également une méthode en trois étapes pour structurer automatiquement un schéma relationnel dans HBase. A chaque étape, un ensemble de règles est appliqué pour la correspondance entre le schéma relationnel et le modèle logique HBase. La gestion des dépendances se traduit par l'ajout d'une famille de colonnes comme dans l'approche précédente. [Scavuzzo et al. 2014] et [Lee and Zheng 2015] utilisent également le système HBase. Pour chaque élément du schéma relationnel, une règle désigne l'élément correspondant dans le modèle orienté colonnes. Chaque table est automatiquement transformée en une famille de colonnes placée dans une même table HBase. Les clés primaires sont concaténées et utilisées pour identifier la ligne. Chaque attribut du schéma relationnel est une colonne.

Des travaux similaires, consistent à traduire le schéma relationnel dans un modèle orienté documents [Lee and Zheng 2015] [Schram and Anderson 2012] [Hsu et al. 2014] [Abdullah and Zhuge 2015] [Fotache and Cogean 2013] [Batra and al 2016] [Zhao et al. 2013] [Mior 2014] [Banerjee et al. 2015] [Zhao et al. 2014]. Tous ces travaux proposent de convertir le schéma relationnel dans MongoDB. Toutes les relations du schéma sont regroupées dans une même collection. Chaque tuple de la relation universelle est traduit en un document imbriquant les valeurs issues de chaque relation. Les données sont ainsi transformées dans un schéma dénormalisé dans l'objectif d'éviter l'utilisation de jointure dans ces environnements distribués.

Outre les approches académiques, il existe des solutions industrielles de transformations de données relationnelles vers des bases de données NoSQL. Dans la solution Apache Sqoop, 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 à 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 revanche, et comme la solution Sqoop, les relations entre les tables ne sont pas assurées dans ces mécanismes d'échanges.

Synthèse. L'ensemble de ces travaux s'inscrivent dans le cadre de transformation de données dans le modèle NoSQL orienté colonnes ou orienté documents. Ils proposent tous une dénormalisation complète des données relationnelles dans la base de données NoSQL

cible. L'objectif de ces travaux est de proposer un schéma de stockage permettant d'éviter l'utilisation des jointures pour lesquelles les systèmes NoSQL sont peu adaptés en raison de l'architecture distribuée sous-jacente.

Ces travaux ne s'intéressent pas aux schémas multidimensionnels R-OLAP. Ils se situent uniquement au niveau logique, ne considérant pas le modèle conceptuel. Ces travaux peuvent être combinés aux approches d'implantation R-OLAP des entrepôts de données, mais la prise en compte du schéma multidimensionnel et des treillis d'agrégats associés ne sont pas abordés.

2.3.2.2 Processus de traduction direct

Nous relevons également un certain nombre de travaux, qui traitent de l'implantation directement de schémas multidimensionnels conceptuels dans les modèles NoSQL.

Hive

Le système Hive est un système spécialisé pour la construction d'entrepôts de données massives [Taylor 2010] [Vora 2011] [Cudré-Mauroux et al. 2013]. Ce dernier repose sur un espace de stockage, appelé *memstore* s'interfaçant avec d'autres bases de données NoSQL (HBase dans la majorité des cas) et un langage d'interrogation proche du SQL appelé HQL [Thusoo et al. 2009]. Des scripts HQL permettent d'insérer les données dans des tables externes (*external table*), par exemple des tables HBase tout en gardant les métadonnées dans des tables Hive. Les données sont manipulées grâce à la correspondance entre les attributs de Hive et les attributs de la table HBase. Il est aussi possible d'interroger les données directement depuis un terminal HBase. L'utilisation de Hive est motivée par la richesse de son langage d'interrogation qui permet d'utiliser des fonctionnalités relationnelles et l'utilisation de requêtes assez complexes.

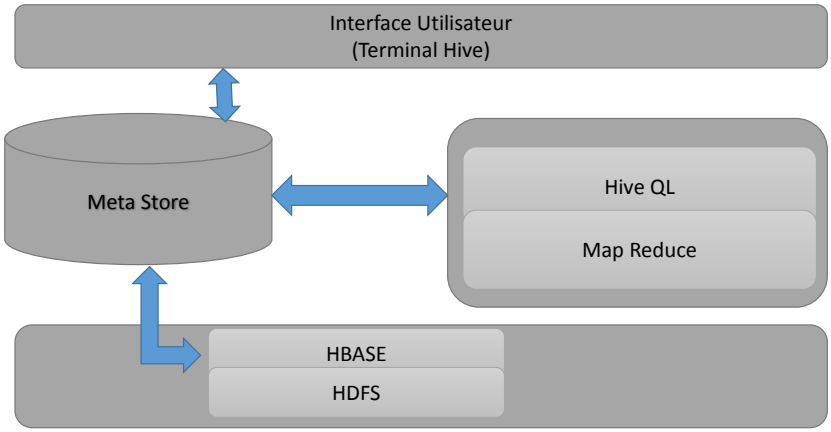


Figure 13 Architecture Hive

Travaux sur les entrepôts multidimensionnels

Dans l'approche [Abelló et al. 2011] les auteurs proposent trois méthodes pour construire un cube de données dans le modèle orienté colonnes. Dans la première méthode, les données sont dénormalisées dans une seule table. La seconde ajoute des index au niveau des valeurs afin de réduire la taille des données manipulées lors de l'interrogation. Dans cette dernière, les auteurs utilisent un index bitmap pour déterminer les positions des identifiants pour réduire encore le nombre de valeurs parcourues.

D'autres travaux visent à développer un entrepôt de données dans un système NoSQL orienté colonnes [Dehdouh et al. 2014a], ou orienté clé-valeur [Zhao and Ye 2013]. L'objectif de ces travaux est de proposer des benchmarks et n'est pas centré sur le processus de transformation de modèles. Dans [Dehdouh 2015] les auteurs proposent des travaux d'implantation de cubes OLAP dans le modèle orienté colonnes.

L'approche de [Scabora et al. 2016] consiste à dénormaliser les données du schéma conceptuel en étoile dans une table HBase composée de deux familles de colonnes. La première famille de colonnes reçoit le fait et les dimensions fréquemment interrogées tandis qu'une deuxième famille de colonnes regroupe les autres dimensions.

Dernièrement les travaux de [Freitas et al. 2016], proposent une approche complète *R2NoSQL* pour transformer un modèle conceptuel d'un entrepôt de données en étoile dans une base de données NoSQL pouvant être soit orientée colonnes, soit orientée documents. L'approche propose par retroconception du schéma relationnel existant, de reconstruire un schéma conceptuel E-R, puis à partir de ce modèle conceptuel, ils appliquent un processus de traduction dans le modèle NoSQL cible. Le schéma NoSQL est basé sur la dénormalisation des données. Dans le modèle orienté colonnes, l'ensemble des tables sont stockées dans une seule famille du modèle orienté colonnes et dans un seul document dans le modèle orienté documents. Cette approche a été expérimentée uniquement avec la base de données MongoDB.

Récemment les travaux de [Yangui et al. 2016] proposent deux approches automatique pour transformer le schéma multidimensionnel en étoile dans deux modèle orienté colonnes et orienté documents. La première approche propose des règles de transformations dites *simples* où la hiérarchie des attributs des dimensions n'est pas considérée. Dans les secondes approches les auteurs proposent une politique de nommage pour retrouver la relation hiérarchique des attributs. Les deux approches ont été évaluées aves des requêtes simples et nécessitent une expérimentation avec des requêtes OLAP plus complexes (drill down, roll-up). Ces deux approches ne prennent cependant pas en considération la construction du treillis d'agrégats.

Autres travaux

D'autres travaux étudient les processus de traduction de schémas conceptuels en NoSQL. Les travaux de [Abdelhédi et al. 2016] définissent des processus de traduction de diagramme de classes UML en NoSQL orienté colonnes avec HBase.

En 2014, une première approche a été proposée pour coupler le modèle orienté graphe et l'OLAP [Castelltort and Laurent 2014]. Dans cette approche les auteurs proposent de structurer les données dans le système NoSQL orienté graphes Neo4J et présentent deux formalismes pour représenter le fait et les dimensions au niveau du modèle logique orienté graphes. Le formalisme assure deux types de relations, celles liant le fait aux dimensions, et celles reliant les attributs des dimensions entre eux ; ces dernières permettent de préserver la relation hiérarchique (cf. Figure 14).

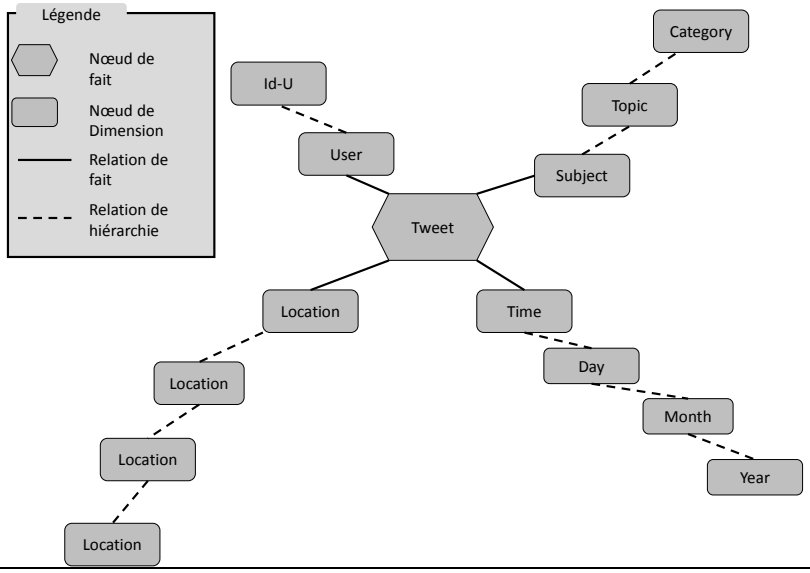


Figure 14 Représentation logique orientée graphes selon [Castelltort and Laurent 2014]

Synthèse. Les deux modèles orientés colonnes et documents sont utilisés comme une alternative au modèle relationnel pour implanter des entrepôts de données multidimensionnelles. Le modèle orienté colonnes est la première solution adoptée dans ce contexte. L'entreposage orienté colonnes repose généralement sur le SGBD HBase. HBase permet de gérer de gros volumes de données dans un environnement distribué. Il est implémenté dans une couche au-dessus de Hadoop (il stocke les données dans l'espace de stockage HDFS). La second approche, basée sur le modèle orienté documents, utilise souvent le système MongoDB. Ce choix se justifie par la richesse de son lagunage d'interrogation et son moteur d'agrégation.

Dans le Tableau 2, nous synthétisons les processus de transformation directe. Nous considérons les caractéristiques suivantes :

- Le schéma conceptuel multidimensionnel. Il s'agit de vérifier si le schéma multidimensionnel a été utilisé.
- Le type de schéma peut être en constellation ou en étoile.
- La catégorie du modèle logique NoSQL utilisé. Le modèle peut être orienté colonnes, orienté documents, orienté graphe ou orienté clé/valeur.
- Le système NoSQL utilisé (HBase, MongoDB...) pour implanter l'entrepôt de données.
- Le type de stockage utilisé pour implanter l'entrepôt de données, il s'agit du schéma normalisé ou dénormalisé.

Tableau 2 Comparatif des travaux de transformation directe des schémas conceptuels en NoSQL

	Conceptuel		Logique			
	Multi-dimensionnel	**Type de schéma**	**Catégorie**	**Système NoSQL**	**Type**	**Treillis**
[Dehdouh et al. 2015]	Oui	Constellation	Colonnes	HBase	Dénormalisé	Non
[Abelló et al. 2011]	Oui	Etoile	Colonnes	HBase	Dénormalisé	Non
[Scabora et al. 2016]	Oui	Etoile	Colonnes	HBase	Dénormalisé	Non
[Freitas et al. 2016]	Oui	Etoile	Colonnes, Documents	MongoDB	Dénormalisé	Non
[Castelltort and Laurent 2014]	Oui	-	Graphes	Neo4J	-	NON
[Abdelhédi et al. 2016]	Non	-	Colonnes	HBase	Dénormalisé	Non
[Yangui et al. 2016]	Oui	Etoile	Colonnes, Documents	Cassandra HBase	Dénormalisé	NoN
[Chevalier et al, 2015a-g] [Chevalier et al, 2016a-c]	Oui	Constellation	Colonnes, Documents	MongoDB HBase	Dénormalisé Normalisé	Oui

2.3.3 Bilan

A la lumière des travaux que nous avons décrits et qui portent sur l'utilisation du modèle NoSQL pour implanter les entrepôts de données multidimensionnelles, nous pouvons souligner l'utilisation prédominante des deux modèles orienté colonnes et le modèle orienté documents pour implanter des entrepôts de données NoSQL. Pour le modèle orienté colonnes, le système HBase est souvent adopté alors que l'outil MongoDB est utilisé pour valider les approches dans le modèle orienté documents.

En revanche si ces deux modèles sont souvent utilisés, il n'existe pas de processus d'implantation directe du schéma multidimensionnel en constellation considérant les deux modèles. En outre toutes les approches existantes proposent un processus unique d'implantation, le plus souvent dénormalisé. Nos travaux généralisent ces approches, en prenant en compte de manière simultanée les deux modèles NoSQL orienté colonnes et documents. Par ailleurs nous proposons plusieurs processus de transformations alternatifs pour chacun des deux modèles cibles ainsi que les processus des conversions intra et inter-modèles.

Dans la section suivante, nous présentons un panorama des solutions développées dans l'industrie.

2.4 PANORAMA DES SOLUTIONS INDUSTRIELLES

Nous dénombrons plus de 150 solutions à l'heure où nous écrivons ce manuscrit. Nombre de ces solutions sont très bien adoptées par la communauté industrielle, d'autres sont très récentes. A l'origine, la première solution de gestion de données Big Data a été développée par Google pour ses besoins. Il s'agit de la solution BigTable [Chang et al. 2008] dont le

stockage est orienté colonnes. Cette solution est à l'origine de ce que l'on peut trouver sur le marché aujourd'hui. D'autres approches de stockages s'en sont inspirées, le modèle orienté documents, le modèle orienté graphes et le modèle clé-valeur. Dans ce qui suit nous présentons les solutions principales pour chaque modèle NoSQL.

2.4.1 Les solutions clé-valeur

2.4.1.1 Voldemort

Développé en java, il est actuellement en usage interne chez *LinkedIn*, Voldemort [Sumbaly et al. 2012] [Abadi 2012] se base sur Amazon Dynamo [DeCandia et al. 2007] et l'enrichit par un mécanisme de stockage en mémoire cache. Il est sorti en tant qu'open source en 2009. Voldemort assure la « cohérence éventuelle » : les lectures et les écritures peuvent être effectuées directement sur les nœuds par les clients. Voldemort rend persistantes les données en utilisant BerkeleyDB [Olson et al. 1999] et permet au développeur de spécifier le facteur de réplication pour chaque partie de la table de hachage. Cela implique que le développeur partitionne manuellement l'espace des clés.

2.4.1.2 Riak

Riak est developpé en Erlang et proposent deux protocoles d'accès aux données [Moniruzzaman and Hossain 2013], Protobuf [Feng and Li 2013] et l'interface REST. Développé par la société *Basho* sur le modèle de dynamo d'Amazon, Riak est un modèle NoSQL orienté clé-valeur qui offre de bonnes performances et une simplicité d'administration. Il est adopté par de grosses entreprises telles que *the water company*, plus gros fournisseur météo dans le monde. Le partitionnement de la donnée se fait dans une architecture décentralisée (maitre-maitre) par hachage. Aucun serveur maitre n'est désigné et chaque nœud est indépendant. Le client peut adresser sa requête à n'importe quel nœud. Si ce dernier n'a pas la donnée, la requête est redirigée vers le nœud contenant la donnée.

2.4.1.3 Redis

Développé en langage C par Salavatore Sanfilippo, et soutenu par la suite par VMware, Redis (*Remonte Dictionnary Service*) est un système NoSQL orienté clé-valeur qui travaille principalement en mémoire. Il est en open source et utilisé actuellement par des sociétés telles que *The Garden* et *GitHub*. Redis est totalement distribué et gère intégralement ses données en mémoire. La distribution des données se fonde sur une architecture maître-esclave avec la particularité qu'un esclave peut être à son tour maître pour un autre esclave ; cela permet de réduire la charge du maitre principal. En cas de panne du maître, la récupération ou le choix d'un nouveau maitre ne se fait pas automatiquement. Redis remédie dans sa dernière version à ce problème par l'ajout d'un nouveau service de monitoring pour informer et alerter l'administrateur, sur l'état de chaque instance, mais encore le remplacement en cas de panne. Redis est un moteur très rapide grâce à son stockage en mémoire et qui via son architecture favorise l'intégrité des données.

2.4.1.4 Memcahedb

Memcachedb est écrit en C et utilise Berkeley DB pour la persistance des données et la réplication. *Memcachedb* est un moteur orienté colonnes qui emploie une approche maître-esclave pour l'accès aux données. *Memcachedb* fonctionne avec un nœud maître unique et de multiples nœuds de calcul. Les clients peuvent accéder en lecture à un nœud quelconque dans le système mais ne peuvent écrire sur le nœud maître. Cela assure la cohérence tout en

permettant de multiples points d'entrée de lecture. Memcachedb est conçu pour permettre de stocker les données en mémoire.

2.4.1.5 Synthèse

Dans le Tableau 3, nous synthétisons les caractéristiques de ces divers systèmes en considérant 6 caractéristiques principales :

- La technologie sur laquelle repose le système, il s'agit du langage avec lequel la solution a été développée.
- L'accès ou la politique utilisée pour la modification des données. Lors d'une opération d'un enregistrement les données sont bloquées (Loks). Un enregistrement peut ne pas être bloqué grâce à la politique MVCC (*Multiversion Concurrency control*).
- Le type de mémoire utilisé pour le stockage de données. Les données peuvent être stockées par exemple en RAM ou sur disque.
- La technique de réplication des données utilisées (synchrone, asynchrone…).
- Le type de licence. Il s'agit des conditions de mise à disposition de la solution.
- Les outils d'interrogation possibles pour la manipulation des données.

Tableau 3 Comparatif des systèmes NoSQL clé-valeur

	Technologies	Accès	Stockage	Réplication	Licence	Interrogation
Redis	C	Locks	RAM	Async	Apache	Ligne de commandes Librairies
Voldemort	Java	MVCC	RAM ou BDB	Async	None[2]	Thrift Avro Protobuf
Riak	Erlang	MVCC	Plug-in	Async	Apache	RESET (HTTP)
Memcachedb	Erlang	Locks	Disk	Sync	BSD[3]	API

2.4.2 Solutions orientées colonnes

2.4.2.1 Cassandra

Facebook a mis en place une solution pour traiter des données grandissantes notamment pour les besoins de stockage et de recherche dans sa messagerie [Lakshman and Malik 2010] [Aniceto et al. 2015]. En 2009, une solution libre est publiée, appelée Cassandra. Il s'agit d'un moteur orienté colonnes totalement distribué.

Dans son modèle de données, Cassandra stocke les données dans une table regroupant des lignes. Chaque ligne est constituée d'une ou plusieurs familles de colonnes, chacune est constituée d'un ensemble de colonnes. Une colonne est apparentée à une paire clé-valeur. Le schéma n'est pas fixé à l'avance ; le nombre de colonnes contenues dans la famille de colonnes peut varier d'une ligne à l'autre. Les données d'une même famille de colonnes sont stockées sur le disque dans un fichier qui lui est propre. Cassandra, diffère légèrement de la

[2] Le site www.projet-voltdemort.com fournit plus d'informations sur la licence NONE
[3] Le site www.memcached.org fournit plus d'informations sur la licence BSD

définition du modèle orienté colonnes puisqu'il ajoute la notion de super colonnes, c'est-à-dire un regroupement de famille de colonnes.

Pour l'interrogation des données, Cassandra utilise le langage d'analyse CQL, proche du langage SQL. Toutefois CQL ne permet pas de faire des agrégations. Pour ce faire il faut définir ses propres fonctions en java par exemple. L'entreprise qui supporte Cassandra accorde un effort particulier à cette problématique, ce qui laisse entrevoir des évolutions possibles lors de prochaines versions.

2.4.2.2 HBase

HBase est une solution supportée par Apache depuis 2008, initiée en 2006 par l'entreprise *Powerset*. HBase est un moteur orienté colonnes [George 2011] [Vora 2011]. Il est considéré comme une contribution de Hadoop puisqu'il est interfacé avec Hadoop et ne peut fonctionner sans. Il utilise HDFS comme espace de stockage et MapReduce pour les traitements. Son développement est prévu pour la gestion de gros volumes de données dans une architecture totalement distribuée. Plusieurs distributions proposent directement HBase dans leur solution telle que *Cloudera* et *Hortonworks*.

Dans son modèle de données HBase stocke les données dans des tables, chaque table est un ensemble de lignes regroupant un ensemble de familles de colonnes. Chaque famille de colonnes regroupe un ensemble de colonnes, variable d'une ligne à l'autre. Chaque valeur de colonnes pour être « versionnée » grâce à un horodatage. Chaque famille de colonnes est stockée séparément sur disque dans un fichier appelé *HFile*.

Pour l'interrogation des données, HBase offre peu de fonctions d'analyse (scan, get, put…). Il est nécessaire d'écrire ses propres fonctions MapReduce avec des langages d'interrogations externes tels que Hive, Phoenix, Java, Python...

2.4.2.3 Hypertable

Hypertable a été développé par la société *Zvents* en 2007 [Khetrapal and Ganesh 2006]. Cette solution open source est totalement distribuée et repose sur une architecture maître-esclave. Hypertable fonctionne sur HDFS pour tirer bénéfice de la réplication automatique des données et la tolérance aux pannes. Actuellement, le plus grand utilisateur d'Hypertable est le fournisseur de recherche chinois *Baidu.*

Hypertable est écrit en C++ pour faciliter un plus grand contrôle de la gestion de la mémoire (mise en cache, la réutilisation, la récupération, etc.). Hypertable fournit un shell, Hypertext Query Language (HQL), avec lequel les utilisateurs peuvent interroger les données stockées. Il offre la possibilité de créer et de modifier des tables d'une manière analogue à celle du langage SQL.

2.4.2.4 Synthèse

Les systèmes orientés colonnes que nous venons de présenter sont principalement inspirés de la solution *Bigtable*. Le Tableau 4 résume les caractéristiques principales (les mêmes que celles de la section 2.4.1.5)

Tableau 4 Comparatif des systèmes NoSQL orientés colonnes

	Technologies	Accès	Stockage	Réplication	Licence	Interrogation
HBase	Java	Locks	HDFS	Async	Apache	Hive Phoenix
Cassandra	C	MVCC	GFS Disk	Async	Apache [4]	CQL
Bigtable	C	Locks	GFS	Sync & Async	Prop[5]	Java
Hypertable	C++	Locks	Files	Sync	GPL[6]	HQL

2.4.3 Solutions orientées documents

Le modèle orienté documents est le concurrent direct du modèle orienté colonnes, il connait un succès considérable. Plusieurs solutions ont été développées.

2.4.3.1 MongoDB

La solution MogoDB a été créée en 2009 par la société *10gen* [Banker 2011] [Chodorow 2013]. Comme les autres solutions, MongoDB est développée pour répondre aux besoins internes de la société en termes de traitement et de stockage de gros volume de données. Elle intègre son propre protocole de distribution et un langage natif d'interrogation.

Dans son modèle de stockage**,** les données sont stockées dans des documents. Les documents sont regroupés dans une collection où chaque document est constitué d'un ensemble de paires clé-valeur pouvant être organisées de manière hiérarchique. Un document est identifié par une clé unique (de taille maximale de 96 octets) gérée automatiquement par le système (comme elle peut être gérée par le client). Chaque document est stocké dans une représentation comprise et optimisée par MongoDB, BJSON (*Binay JSON*) et peut posséder sa propre structure, on parle de *schemaless*. Dans la dernière version de MongoDB (la 3.2), la taille maximale conseillée d'un document est de 16 Mo.

Dans son mécanisme de stockage, MongoDB stocke l'ensemble des collections dans des bases de données regroupées dans des espaces de noms. Chaque collection est stockée séparément et indépendamment sur disque. Pour les objets de taille large, MongoDB les découpe en petits segments (considérés comme des documents). Ces segments sont stockés dans une collection appelée *Chunks* tandis que les métadonnées des segments sont stockées dans une autre appelée *Files,* permettant d'identifier chaque segment de la collection *Chunks.*

Pour l'interrogation des données, MongoDB offre un langage de manipulation de données très riche. De plus, il est possible de coder ses propres fonctions MapReduce. *MongoDB* se caractérise aussi par son propre moteur d'agrégation. Un autre avantage non négligeable est l'intégration d'un moteur de manipulation de données géospatiales.

2.4.3.2 CouchDB

Développée en 2005 par Damien Katz, ancien développeur chez *IBM* pour répondre à des besoins web, CouchDB est un moteur de stockage orienté documents, qui a été développé en langage Erlang [Anderson et al. 2010].

[4] Cassandra offre aussi une version gratuite sans support. Les informations sont accessibles à l'adresse : www.incubator.apache.org/cassand.

[5] Le site officiel labs.google.com/papers/bigtable.html fournit plus d'informations sur la licence Apache

[6] Pour plus d'informations : www. hypertable.org

CouchDB est caractérisé par une bonne cohérence des données grâce à sa stratégie de type MVCC (Mutiversion concurency controle) ; un document en cours de modification n'est jamais bloqué par contre à l'enregistrement des mises à jour, une vérification est menée pour voir si le document n'a pas été modifié entre temps. Il dispose également d'un système de version pour gérer les éventuels conflits. Chaque document contient un numéro de version qui correspond au nombre de fois où le document a été modifié.

Comme MongoDB les données sont stockées dans des documents de format JSON et son manipulables par des requêtes écrites en JavaScript. Le langage d'interprétation au niveau du serveur (javascript) peut être remplacé par d'autres langages tels que Python.

2.4.3.3 SimpleDB

SimpleDB est un système de stockage orienté documents développé depuis 2007 par Amazon et utilisé dans les offres de cloud, EC2 (*Elastic Computing Cloud*) et S3 (*Simple Storage Service*) [Murty 2008]. Comme son nom l'indique, son modèle est simple. Il se contente des opérations de base (telles que *Select*, *Delete*, *GetAttributes*, et *PutAttributes*). SimpleDB est considéré comme le plus simple des systèmes de stockage orientés documents, car il n'utilise pas l'imbrication des données. Comme la plupart des systèmes que nous discutons, SimpleDB supporte la cohérence éventuelle, et non pas la cohérence transactionnelle. Comme la plupart des autres systèmes, il utilise la réplication asynchrone.

Les documents sont regroupés dans des domaines (équivalent de la notion de collection dans MongoDB). Les index du domaine sont automatiquement mis à jour lorsque les attributs d'un document donné sont modifiés. Le partitionnement des données n'est pas automatique. La réplication se fait de manière asynchrone.

2.4.3.4 Terrastore

Un autre système récent, orienté documents, est *Terrastore*, qui est bâti sur le produit distribué Terracotta [Cattell 2011]. L'accès client à Terrastore est basé sur les opérations HTTP pour récupérer et stocker des données. Les API clients Java et Python ont également été implémentées.

Terrastore distribue automatiquement des données sur les nœuds du serveur, et peut automatiquement redistribuer les données lorsque des nœuds sont ajoutés ou supprimés. Il peut effectuer des requêtes basées sur des prédicats, il supporte le mécanisme MapReduce et des agrégations des données.

Comme les autres bases de données documentaires, Terrastore est sans schéma défini au préalable, et ne supporte pas de transactions ACID. Il assure la cohérence pour un seul document (pour chaque document).

2.4.3.5 Synthèse

Les systèmes orientés documents sont tous caractérisés par la flexibilité des schémas. Le Tableau 5 résume les caractéristiques principales (les mêmes que celles de la section 2.4.1.5)

Tableau 5 Comparatif des systèmes NoSQL orientés documents

	Technologies	Accès	Stockage	Réplication	Licence	Interrogation

Terrastore	JAVA	Locks	RAM+	Sync	Apache[7]	HTTP API Python & java
SimpleDB	Erlang	-	S3	Async	Prop[8]	API
MongoDB	C++	Locks	Disk	Async	GPL[9]	Javascript (Schell) API
CouchDB	Erlang	MVCC	Disk	Async	Apache	REST

2.4.4 Solutions orientées graphes

Nous limitons notre étude au système le plus connu, Neo4J [Holzschuher and Peinl 2013] [Vukotic et al. 2015]. La solution Neo4j conserve certaines caractéristiques des systèmes relationnels (transactions) mais avec une nouvelle philosophie de structuration des donnée, basée sur la théorie des graphes. Il bénéficie également d'un espace de stockage extensible et tolérant aux pannes. Neo4j est très populaire grâce à sa structure reposant sur deux objets fondamentaux : les nœuds et les relations. Le nœud définit un concept réel ou abstrait. Il peut posséder des propriétés et peut avoir des relations.

Neo4j peut être installé sur un seul serveur comme il peut être déployé en mode cluster. La communication entre nœuds suit une architecture maître-esclave où le maître peut être désigné au démarrage ou en cas de panne.

Une limite notable est que Neo4j ne repose pas sur des mécanismes de sharding. Il ne permet pas de découper les données et en faire des partitions mais en revanche il respecte les propriétés ACID. De ce fait, il permet le respect de l'intégrité des données via des fonctions de verrou lors des modifications des données. Il assure la disponibilité des données grâce à la réplication des données et sa capacité à agir rapidement pour désigner un nouveau maître en cas de panne.

2.4.5 Systèmes relationnels extensibles

Contrairement aux autres bases de données NoSQL, ce type de bases de données dispose d'un schéma relationnel prédéfini avec une interface SQL et supporte les transactions ACID. Traditionnellement conçus pour des environnements centralisés, ces systèmes peuvent etre étendus aux environnements massivement distribués.

2.4.5.1 MySQL Cluster

MySQL Cluster fait partie de la version de MySQL depuis 2004, et le code est développé à partir d'un projet antérieur d'*Ericsson* [Cattell 2011]. La solution MySQL Cluster est devenue possible en remplaçant le moteur InnoDB avec une couche distribuée appelé NDB.

Les données sont distribuées et répliquées sur l'ensemble des nœuds pour palier à un éventuel problème d'indisponibilité d'un ou plusieurs nœuds. Depuis la version 4.1, MySQL est capable de gérer une grappe de serveurs MySQL, chaque nœud d'une grappe stocke une copie de la base de données. Le cluster MySQL prend en charge le stockage en mémoire (In-memory) ainsi que le stockage des données sur disque. Le stockage en mémoire (In-memory) permet des réponses en temps réel.

[7] Pour plus d'informations : www.code.google.com/terrast
[8] Pour plus d'informations : amazon.com/simpledb
[9] Pour plus d'informations : www.mongodb.org

Bien que le cluster MySQL semble évoluer vers un nombre important de nœuds, l'architecture est limitée lorsque l'infrastructure repose sur une très grande quantité de nœuds.

2.4.5.2 VoltDB

C'est un nouveau SGBDR open-source conçu pour la haute performance (par nœud), ainsi que l'évolutivité ; il est conçu spécialement pour répondre à des besoins OLTP comme l'expliquent clairement le fondateur (M. Stonebarker) : « NoSQL a abandonné SQL, a abandonné des atouts pour gagner en performance et scalabilité mais ne convient pas à l'OLTP » [Stonebraker and Weisberg 2013].

Les fonctions d'évolutivité et de disponibilité sont compétitifs avec le cluster MySQL et les systèmes NoSQL :

- Les tables sont partitionnées sur plusieurs nœuds, et les clients peuvent appeler chaque nœud du serveur. La distribution est transparente pour les utilisateurs de SQL, mais il peut être pris en charge par le client.
- Les tables sélectionnées peuvent être répliquées sur les serveurs, par exemple pour un accès rapide aux données.
- Dans tous les cas, les données sont répliquées, de sorte que les données soient toujours disponibles en cas de panne d'un nœud. Des sauvegardes (snapshots) de la base de données sont également prises en charge, en continu ou planifiées.

VoltDB fait valoir que ces optimisations réduisent considérablement le nombre de nœuds nécessaires pour supporter une charge d'application donnée. Elle se différencie des autres SGBD car elle a été pensée pour retirer tous les goulots d'étranglement d'une base de données (logging, latching, locking et la gestion du buffer). Sur le site officiel les représentants rapportent des résultats impressionnants, 100 fois plus rapide que MySQL et 13 fois que Cassandra et 45 plus qu'Oracle. Des résultats quasi-linaires annoncés par le constructeur jusqu'à 50 nœuds, mais sans indications lorsque les données atteignent des volumes gigantesques (Petaoctet, Exaoctet).

2.4.5.3 NuoDB

Crée en 2008 sous le nom de NimbusDB, NuoDB est une base de données relationnelles (qui porte le nom de l'entreprise qui l'a développée) [Brynko 2012] [Proctor 2013]. Elle est conçue pour fonctionner sur une architecture distribuée avec le respect des propriétés ACID. Elle supporte l'ensemble du SQL'99. Son architecture distribuée repose sur le modèle P2P et une communication asynchrone. Chaque élément de la base de données est appelé atome, si un atome est modifié, les autres atomes, stockant la même copie, sont avisés de manière transactionnelle des modifications, via des messages.

Parmi les avantages principaux :

- Partitionnement horizontal ou vertical à la volée.
- Haute disponibilité du SGBD qui grâce à son système de réplication qui continue à fonctionner même s'il y a plusieurs pannes.
- Plusieurs API supportées : node.js, Java EE, Python…

2.4.6 Synthèse des solutions industrielles :

Dans cette dernière section, nous avons décrit quelques solutions industrielles. En 10 ans, les solutions NoSQL se sont imposées dans le paysage de la gestion des données. Au sein du

même modèle NoSQL, les solutions sont très comparables avec des différences au niveau des modèles de représentation des données.

L'approche récente, NewSQL, sont des bases de données relationnelles extensibles tout en respectant les propriétés ACID. Cette technologie annonce des performances intéressantes. Toutefois le NoSQL continue à se développer, et connaît régulièrement des améliorations.

2.5 BILAN

Ces dernières années, les systèmes NoSQL ont connu un important développement, et ont atteint aujourd'hui un certain niveau de maturité, laissant entrevoir la possibilité d'utiliser ces systèmes pour le développement d'entrepôts de données multidimensionnelles. Plusieurs travaux de recherche ont exploré cette voie en se focalisant sur un seul type de système NoSQL : orienté colonnes, orienté documents et également orienté graphes.

Les travaux menés dans ce mémoire de thèse visent à étudier de manière conjointe plusieurs modèles NoSQL, afin de développer des processus uniformes. Nos travaux portent sur les modèles NoSQL orientés documents et colonnes.

Contrairement aux approches existantes, nous proposons plusieurs solutions d'implantation des entrepôts de données multidimensionnelles dans les systèmes NoSQL. Ces alternatives offrent chacune des avantages qui tiennent compte de spécificités de ces nouveaux paradigmes : dénormalisation, distribution massives, structures de données complexes.

Nos travaux prennent en compte l'ensemble des concepts de la modélisation multidimensionnelle. Les processus que nous définissons permettent l'implantation des schémas en constellation, et leur optimisation par pré-agrégats.

3. CHAPITRE III : MODELISATION MULTIDIMENSIONNELLE « *NOT ONLY SQL* »

3.1 INTRODUCTION

Nous proposons d'adopter une approche de modélisation des entrepôts de données basée sur les trois niveaux d'abstraction conceptuel, logique et physique.

Au niveau conceptuel, la modélisation multidimensionnelle repose sur les concepts de fait, de dimension et de hiérarchie. Cette modélisation consiste à décrire les données analysées au travers d'un schéma en étoile ou en constellation [Kimball and Ross 2013]. Le schéma conceptuel est ensuite traduit en un modèle logique lui-même traduit en modèle physique. Traditionnellement, au niveau logique, trois approches d'implantation sont possibles. Il s'agit des approches ROLAP [Morfonios et al. 2007b], MOLAP [Chaudhuri et al. 2001] et l'approche hybride HOLAP [Kaser and Lemire 2003]. L'approche la plus répandue est l'approche ROLAP consistant à implanter les bases de données multidimensionnelles en utilisant des SGBDR. Des règles de passage sont utilisées pour convertir les structures multidimensionnelles du niveau conceptuel en un modèle logique basé sur des relations.

L'émergence des systèmes « Not-Only SQL » (*NoSQL*) permettent d'envisager de nouvelles approches pour implanter le schéma d'un entrepôt de données. Ces systèmes offrent l'avantage de gérer de grandes masses de données (mégadonnées ou « big data ») et sont facilement extensibles. Contrairement aux systèmes relationnels, les systèmes NoSQL reposent sur des approches de dénormalisation des données, afin d'éviter les calculs de jointure. En contrepartie, une certaine redondance dans les données est tolérée.

Ce chapitre présente nos propositions pour modéliser un entrepôt de données structurées de manière multidimensionnelle dans les systèmes NoSQL. Comme l'illustre la Figure 15, nos propositions se situent aux niveaux conceptuel et logique. Nous retenons deux types de modèles *NoSQL* qui modélisent les données de façon orthogonale :

- le modèle orienté documents organise les données horizontalement au sein d'un ensemble de documents ;
- le modèle orienté colonnes organise les données verticalement au sein d'un ensemble de colonnes.

Notre processus convertit les mécanismes conceptuels dans le modèle logique NoSQL cible pour obtenir un schéma en étoile. Ce schéma est ensuite optimisé par pré-calculs au sein d'un cube OLAP.

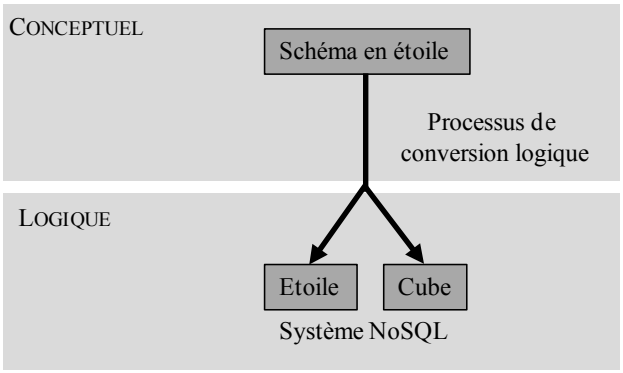


Figure 15 Processus de conception

La section 3.2 définit les concepts et formalismes permettant de représenter un schéma en constellation au niveau conceptuel. La section 3.3 décrit la traduction des concepts dans les deux contextes NoSQL orienté documents et orienté colonnes. Enfin la section 3.4 présente les mécanismes d'optimisation par cube OLAP dans ces deux systèmes NoSQL.

3.2 MODELISATION CONCEPTUELLE MULTIDIMENSIONNELLE

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

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

- $F^S = \{F_1, ..., F_n\}$ est un ensemble fini de n faits ;
- $D^S = \{D_1, ..., D_m\}$ 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** $f_i \in \{SUM, MAX, MIN, COUNT, AVG...\}$.

Une dimension est constituée d'un ensemble d'attributs représentant différents niveaux de granularité sur les données à analyser (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 ;
- $Param^{H_j} = \langle id^D, p_1^{H_j}, ..., p_w^{H_j}, all^D \rangle$ est un ensemble ordonné de w+2 attributs (avec $w \leq u$) appelés **paramètres** fournissant une échelle graduée de la hiérarchie, $\forall k \in [1..w], p_k^{H_j} \in A^D$;
- $Weak^{Hj}$: $Param^{Hj} \rightarrow 2^{A^D - param^{Hi}}$ 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 correspondant au paramètre extrémité.

Exemple : Nous introduisons un exemple d'analyse OLAP portant sur les tweets. Pour ce faire, nous définissons un schéma en étoile graphiquement représenté à la Figure 16. Cette figure permet d'introduire les formalismes graphiques associés aux concepts du modèle.

Le fait concerne la popularité des tweets : elle correspond au nombre de fois qu'un tweet est retweeté, considérant que plus un tweet est retweeté, plus il est populaire. Le fait est ainsi défini par $F_{Tweet} = (N^{FTweet}, count(M^{FTweet}))$. Le fait est associé à quatre dimensions : $Star^S(F_{Tweet}) = \{D_{User}, D_{Location}, D_{Subject}, D_{Time}\}$.

La dimension D_{User} décrit l'auteur du tweet. Elle comporte six attributs $A^{User} = \{id^{User}, name\_u, time\_c, language, time\_z, all^{User}\}$ organisés selon trois hiérarchies :

- $(H_{TC}, \langle id^{User}, time\_c, all^{User} \rangle, \{(id^{User}, \{name\_u\})\})$
- $(H_{TM}, \langle id^{User}, language, all^{User} \rangle, \{(id^{User}, \{name\_u\})\})$
- $(H_{TZ}, \langle id^{User}, time\_z, all^{User} \rangle, \{(id^{User}, \{name\_u\})\})$

La dimension $D_{Location}$ décrit le lieu d'où est émis le tweet. Elle comporte six attributs $A^{Location} = \{city, country, population, continent, zone, all^{Location}\}$ organisés selon deux hiérarchies :

- $(H_{Cont}, \langle city, country, continent, all^{Location} \rangle, \{(country, \{population\})\})$
- $(H_{Zn}, \langle city, country, zone, all^{Location} \rangle, \{(country, \{population\})\})$

La dimension $D_{Subject}$ décrit le sujet du tweet (classiquement, le sujet est déterminé en fonction de la fréquence d'apparition de termes dans le tweet). Elle comporte trois attributs $A^{Subject} = \{topic, category, all^{Subject}\}$ organisés selon une hiérarchie :

- $(H_{Top}, \langle topic, category, all^{Subject} \rangle, \{\})$

La dimension D_{Time} décrit la date à laquelle le tweet est émis. Elle comporte cinq attributs $A^{Time} = \{day, month, month\_name, year, all^{Time}\}$ organisés selon une hiérarchie :

- $(H_{Time}, \langle day, month, year, all^{Time} \rangle, \{(month, \{month\_name\})\})$

Notons que les attributs racines des dimensions $D_{Location}$, $D_{Subject}$ et D_{Time} correspondent respectivement aux paramètres *city*, *topic* et *day* (au lieu de id^D).

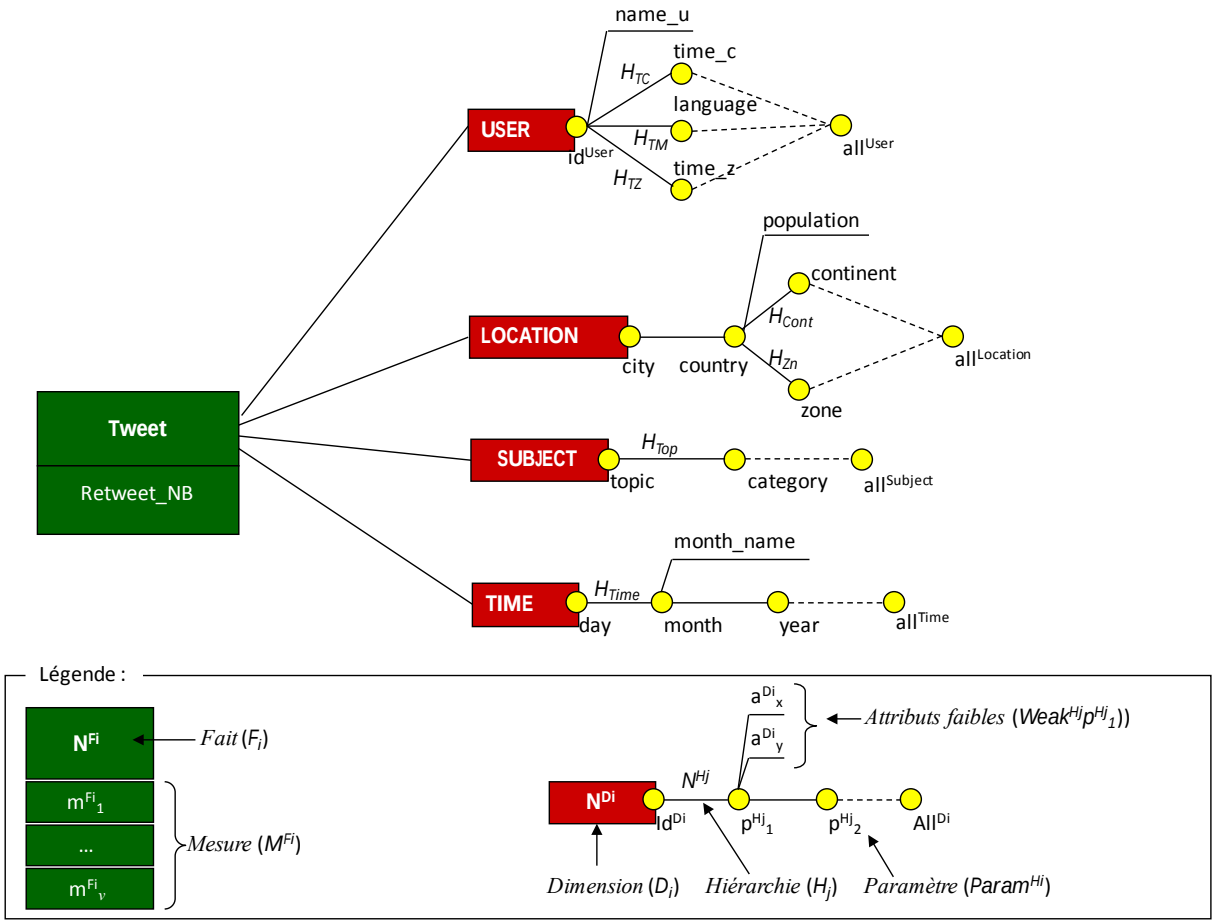


Figure 16 Exemple de schéma conceptuel en étoile

3.3 MODELISATION LOGIQUE NOT-ONLY-SQL

Nos propositions portent sur deux modèles NoSQL : orienté documents et orienté colonnes. Dans la section 3.3.1 nous présentons le modèle orienté documents et quatre processus de traduction du schéma en constellation. Dans la section 3.3.2 nous présentons le modèle orienté colonnes ainsi que les processus de traduction.

3.3.1 Modélisation multidimensionnelle orientée documents

3.3.1.1 Modèle NoSQL orienté documents

Dans le modèle orienté documents, les données sont organisées en **collections** de **documents**. Nous utilisons deux constructeurs :

- [] pour représenter une structure formée d'un ou plusieurs attributs, et
- {} pour représenter un ensemble.

Définition : Une **collection** C est définie par (N^C, E^C) où

- N^C est le nom de la collection,
- $E^C = \{d_1, \ldots, d_t\}$ est un ensemble de documents.

La structure des documents est définie par l'intermédiaire d'**attributs** (pouvant s'apparenter à des couples clé/valeur où le nom de l'attribut est la clé). Nous distinguons les **attributs simples** dont les valeurs sont atomiques, des **attributs composés** dont les valeurs sont elles-mêmes des documents appelés **documents imbriqués**.

Définition : Un **document** d_i est défini par (S_i, V_i) où

- $S_i = [a_1,..., a_{nai}]$ est le schéma du document formé par un ensemble d'attributs,
- $V_i = [a_1 : v_1,..., a_{nai} : v_{\text{nai}}]$ est la valeur du document.

Il est important de noter que le schéma est inhérent à chaque document. Dans les systèmes NoSQL orientés documents, chaque document à son propre schéma, pouvant varier d'un document à l'autre, même au sein d'une même collection.

Un attribut simple a_k peut correspondre à une seule valeur atomique v_k ou un ensemble de valeurs atmoiques $[v_{k_1}, v_{k_2}, ...]$.

Un attribut composé a_k peut être composé, c'est-à-dire constitué par une structure imbriquée S_k. Celle-ci peut être mono-valuée (structure plate) $a_k[S_k]$, soit multi-valuée $a_k\{[S_k]\}$.

Exemple : Ci-dessous nous présentons un document d_i décrivant une personne.

```
di = (
      [id,nom,prenom,daten[jour,mois,annee],telephone{[numero]}],
      [id:12345,
       nom:"el malki",
       prenom:"mohammed",
       daten: [jour:"24",mois:"08",annee:"1984"],
       telephone : {[numero:"06"],[numero:"04"]}]
      )
```

L'attribut composé *telephone* pourrait être un attribut simple dont la valeur est constituée d'un ensemble de valeurs, c'est-à-dire *telephone* : {"06", "04"}.

Les sous-sections suivantes présentent 4 processus de traduction, en NoSQL orienté documents, d'un entrepôt de données multidimensionnelles en constellation.

3.3.1.2 Processus de traduction plate en orienté documents

Ce premier processus repose sur une dénormalisation des données du schéma en constellation.

A chaque fait $F\hat{I}\ F^S$ et ses dimensions associées $Star^S(F)$ correspond 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 plat sans imbrication.

Un document d_i est défini par un schéma S_i 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 $\bigcup_{D_j \in Star^S(F)} A^{D_j}$ forme un attribut a_k simple.

A l'instar des implantations R-OLAP, nous ne matérialisons pas au niveau logique les attributs extrémités all^D des dimensions.

Exemple : Considérons le schéma conceptuel de la Figure 16. Chaque instance du fait F_{Tweet} est traduite par un document. L'expression ci-dessous présente la collection obtenue et détaille un document :

```
{
…,
d_i = (
      [id,
       idUser,name,language,time_c,time_z,
       city,country,population,zone,
       day,month,month_name,year,
       topic,category,
       Retweet_NB],
      [id:12345,
       idUser:"C02265",name:"Smith",language:"french",time_c:"Paris",
time_z:"France",
       city:"Paris",country:"France",population:66000000,
zone:"Europe-Ouest",
       day:"03-31-2015",month:"03-2015",month_name:"march",year:2015,
       topic:"foot",category:"sport",
       Retweet_NB:200],
      ),
...
}
```

La Figure 17 présente sous une forme graphique le document d_i constitué par un ensemble d'attributs simples issus des mesures et des attributs des dimensions liées.

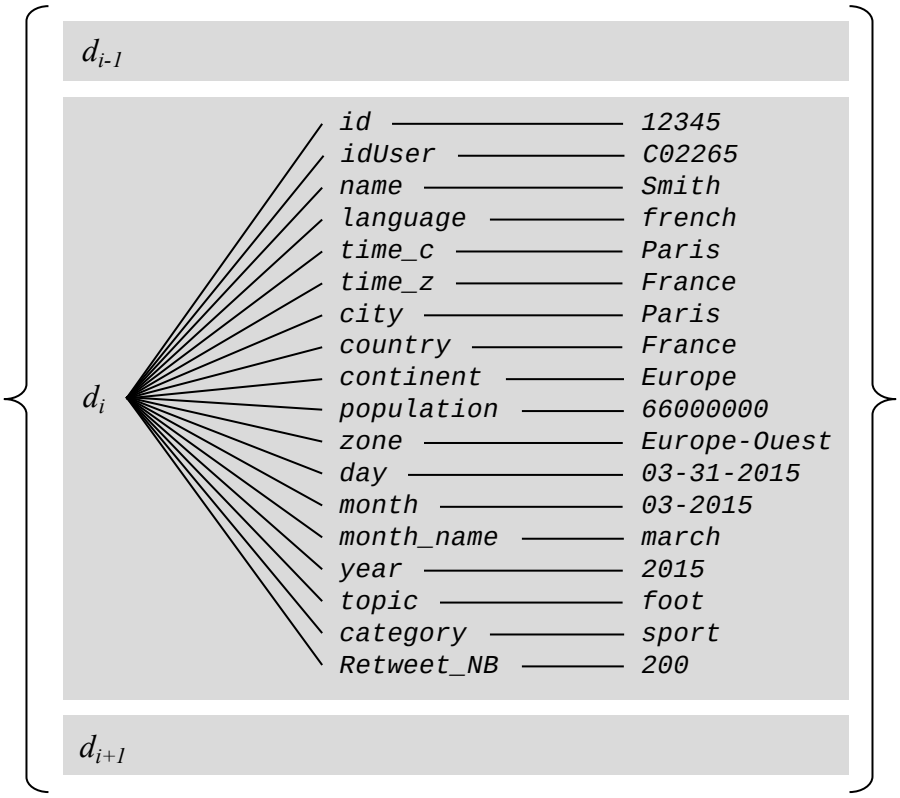


Figure 17 Exemple de document par traduction plate

Cette implantation évite l'utilisation des jointures entre le fait et les dimensions, mais génère un important niveau de redondance par duplication des données des dimensions.

3.3.1.3 Processus de traduction par imbrication en orienté documents

Ce second modèle vise à tirer profit des possibilités d'imbrication offertes par le modèle orienté documents. Il combine l'utilisation d'attributs simples et d'attributs composés. Comme précédemment, à chaque fait $F\hat{I}\ F^S$ et ses dimensions associées $Star^S(F)$ correspond une collection de même nom (N^F, E^F). Chaque instance du fait est traduite par un document d_i.

L'imbrication est utilisée pour rassembler les attributs représentant les mesures du fait. De même, les attributs de chaque dimension associée $Star^S(F)$ sont rassemblés dans un même attribut composé.

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

- *id* est l'identifiant du document,
- un attribut composé est défini pour le fait, et chaque mesure de M^F forme un attribut m_k^F simple imbriqué,
- un attribut composé est défini pour chaque dimension, chaque attribut $a_k^{D_j}$ de la dimension D_j forme un attribut simple imbriqué.

Exemple : Considérons l'instance utilisée dans l'exemple précédent, que nous restructurons ci-dessous en fonction de l'approche par imbrication :

{

...,

d_i = (

[id,

User[idUser,name,language,time_c,time_z],

Location[city,country,population,zone],

Time[day,month,month_name,year],

Subject[topic,category],

Tweet[Retweet_NB]],

[id:12345,

User:[idUser:”C02265”,name:”Smith”,language:”french”,

time_c:”Paris”, time_z:”France”],

Location:[city:”Paris”,country:”France”,population:66000000,

zone:”Europe-Ouest”],

Time:[day:”03-31-2015”,month:”03-2015”,month_name:”march”,

year:2015],

Subject:[topic:”foot”,category:”sport”],

Tweet:[Retweet_NB:200]],

),

...

}

La Figure 18 présente sous une forme graphique le document d_i constitué par un ensemble d’attributs composés issus du fait (imbrication des mesures) et de chaque dimension associée (imbrication des attributs de la dimension).

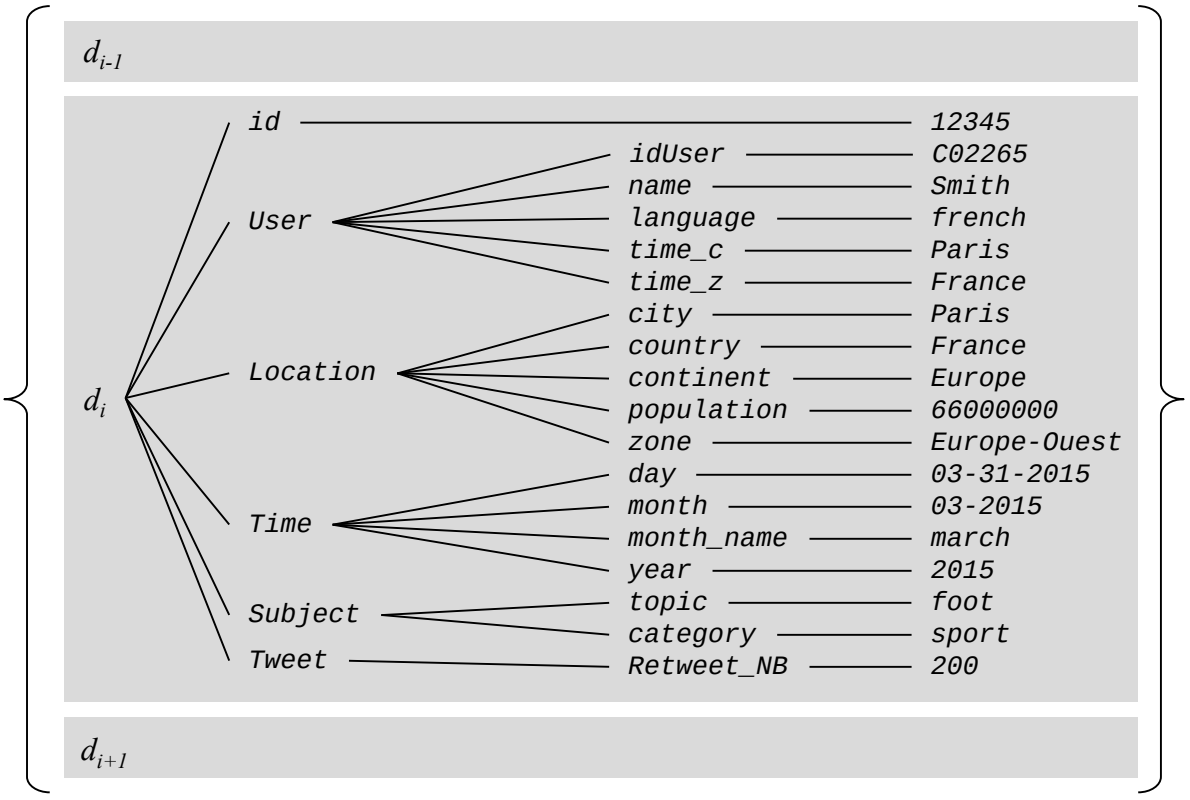


Figure 18 Exemple de document par traduction imbriquée

3.3.1.4 Processus de traduction hybride en orienté documents

Comme les approches précédentes, une traduction hybride rassemble au sein d'une même collection les données issues d'un fait $F \in F^S$ et des dimensions associées $Star^S(F)$.

Les attributs des dimensions sont convertis en attributs simples, à plat sans imbrication, et stockés dans un document de la collection, à raison d'un document par dimension. Les mesures du fait sont converties en attributs simples, à plat sans imbrication, et stockés dans un document de la même collection. Les documents des faits contiennent également les références aux documents représentant les dimensions associées.

Pour chaque instance de dimension, un document d_i est défini par un schéma S_i 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 est défini par un schéma S_i 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.

Exemple : Considérons l'exemple de document précédent. Dans l'approche hybride, un document est constitué pour chaque instance des dimensions ainsi qu'un autre document pour l'instance du fait. Nous obtenons les documents définis ci-dessous :

{

...,

d_i^{Tweet} = (

[id,

idUser,city,day, topic,

Retweet_NB],

[id:12345,

idUser:”C02265”, city:”Paris”, day:”03-31-2015”, topic:”foot”,

Retweet_NB:200],

),

d_i^{User} = (

[idUser,name,language,time_c,time_z],

[idUser:”C02265”,name:”Smith”,language:”french”,time_c:”Paris”,

time_z:”France”],

),

$d_i^{Location}$ = (

[city,country,population,zone],

[city:”Paris”, country:”France”, population:66000000,

zone:”Europe-Ouest”],

),

d_i^{Time} = (

[day,month,month_name,year],

[day:”03-31-2015”,month:”03-2015”,month_name:”march”,

year:2015],

),

$d_i^{Subject}$= (

[topic,category],

[topic:”foot”,category:”sport”],

),

...

}

La Figure 19 présente sous une forme graphique les documents d_i^{Tweet}, d_i^{User}, $d_i^{Location}$, $d_i^{Subject}$, $d_i^{Subject}$constitués par un ensemble d’attributs simples issus respectivement du fait et de chaque dimension.

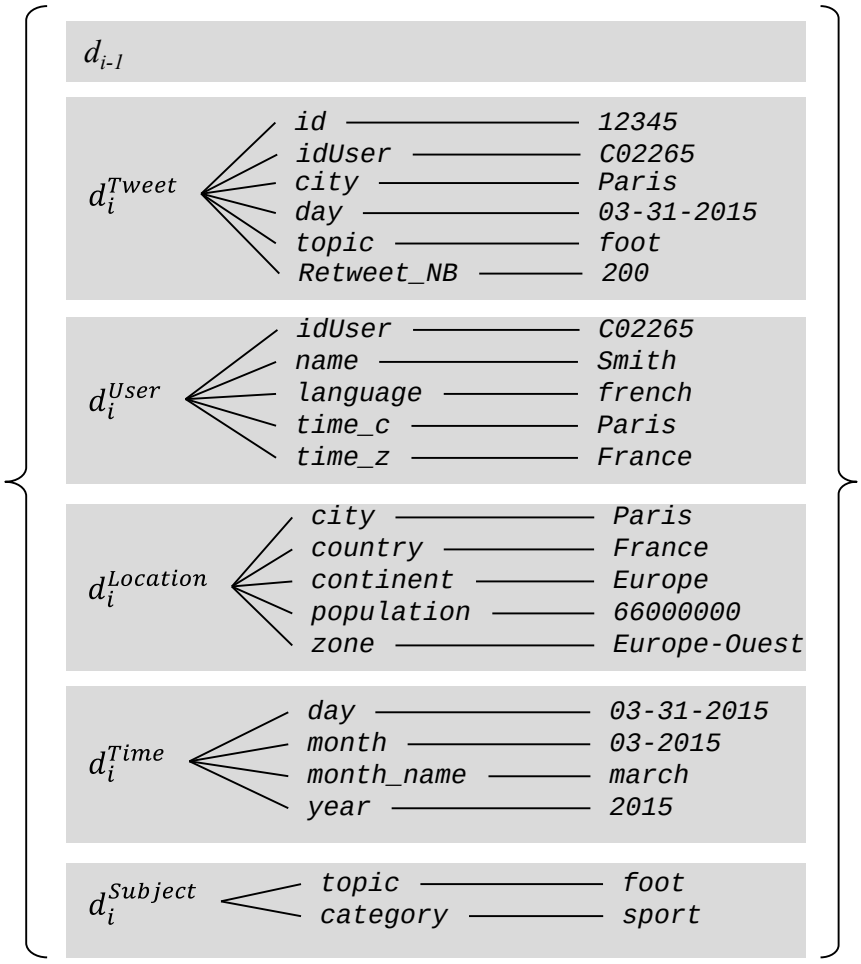


Figure 19 Exemple de document par traduction hybride

Cette implantation évite la redondance des données des dimensions des deux modèles précédents, mais introduit une hétérogénéité dans les structures des documents placés dans la même collection. Contrairement aux approches précédentes, l'approche hybride nécessite l'utilisation d'auto-jointures pour retrouver les documents de dimensions liés aux documents des faits au sein de la collection.

3.3.1.5 Processus de traduction éclatée en orienté documents

Contrairement aux approches précédentes, une traduction éclatée distribue les données au sein de plusieurs collections : les données issues d'un fait $F\hat{I}\ F^S$ sont placées dans une première collection, et les données de chaque dimension associée $Star^S(F)$ 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 $d_i\hat{I}\ C^{D_j}$ est défini par un schéma S_i 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 C^F est défini par un schéma S_i 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 *Star*^{S}*(F)* associée, un attribut simple id^{D_j} est ajouté référençant un document lié.

Exemple : Considérons l'exemple de document précédent. Dans l'approche éclatée, un document est constitué pour chaque instance des dimensions ainsi qu'un autre document pour l'instance du fait. Ces documents sont placés dans des collections distinctes. Nous obtenons les documents définis ci-dessous :

```
{
...,
d_i^Tweet = (
      [id,
       idUser,city,day, topic,
       Retweet_NB],
      [id:12345,
       idUser:"C02265", city:"Paris", day:"03-31-2015", topic:"foot",
       Retweet_NB:200],
      ),
...
}
{
...,
d_i^User = (
      [idUser,name,language,time_c,time_z],
      [idUser:"C02265",name:"Smith",language:"french",time_c:"Paris",
time_z:"France"],
      ),
...
}
{
...,
d_i^Location = (
      [city,country,population,zone],
```

[city:”Paris”,country:”France”,population:66000000,
zone:”Europe-Ouest”],
),
...
}
{
...,
d_i^{Time} = (
[day,month,month_name,year],
[day:”03-31-2015”,month:”03-2015”,month_name:”march”,
year:2015],
),
d_{i+1},
...
}
{
...,
d_{i-1},
$d_i^{Subject}$= (
[topic,category],
[topic:”foot”,category:”sport”],
),
...
}

La Figure 20 présente sous une forme graphique les documents d_i^{Tweet},d_i^{User}, $d_i^{Location}$, $d_i^{Subject}$, $d_i^{Subject}$ appartenant respectivement aux collections C^{Tweet}, C^{User}, $C^{Location}$, C^{Time} $C^{Subject}$. Les documents sont constitués par un ensemble d’attributs simples issus respectivement du fait et de chaque dimension.

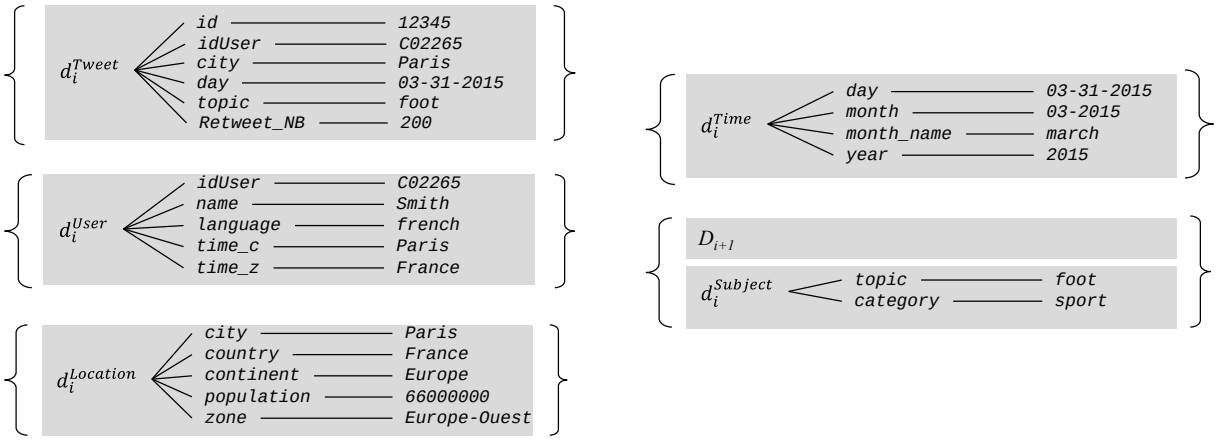


Figure 20 Exemple de documents par traduction éclatée

Comme l'approche précédente, cette implantation évite la redondance des données des dimensions des deux modèles précédents et place les documents dans différentes collections structurellement homogènes. Contrairement aux deux premières approches, l'approche éclatée nécessite également l'utilisation de jointures pour retrouver les documents de dimensions des différentes collections liés aux documents des faits.

Le modèle orienté documents est parmi les modèles *NoSQL* les plus populaires et il se prête à de nombreux cas d'utilisation. Le modèle orienté colonnes est un autre modèle *NoSQL* qui est également très utilisé et il se caractérise par un stockage de données orthogonal par rapport au modèle orienté documents. Dans la suite, nous étudions l'implantation des entrepôts de données multidimensionnelles dans le modèle orienté colonnes.

3.3.2 Modélisation multidimensionnelle orientée colonnes

3.3.2.1 Modèle NoSQL orienté colonnes

Dans le modèle orienté colonnes, les données sont organisées en **tables** contenant un ensemble de lignes, chacune organisée en **colonnes**.

Nous adoptons le même formalisme que précédemment, basé sur deux constructeurs :

- [] pour représenter une structure formée d'un ou plusieurs colonnes, et
- {} pour représenter un ensemble.

Définition : Une **table** T est définie par (N^T, E^T) où

- N^T est le nom de la table,
- $E^T = \{r_1, ..., r_t\}$ est un ensemble de lignes.

La structure des lignes est définie par l'intermédiaire de **colonnes** (pouvant s'apparenter à des paires clé-valeur). Nous distinguons les **colonnes simples** dont les valeurs sont atomiques, des **colonnes composées** (appelées **familles de colonnes**) dont les valeurs sont elles-mêmes des regroupements de colonnes.

Définition : Une **ligne** r_i est définie par (S_i, V_i) où

- $S_i = [c_1,..., c_{nci}]$ est le schéma de la ligne, formé par un ensemble de colonnes,
- $V_i = [c_1 : v_1,..., c_{nci} : v_{nci}]$ est la valeur de la ligne.

Le schéma est inhérent à chaque ligne. Dans les systèmes NoSQL orientés colonnes, chaque ligne à son propre schéma, pouvant varier d'une ligne à l'autre, même au sein d'une même table.

Il est important de noter qu'un seul niveau d'imbrication n'est possible : une famille de colonnes est composée uniquement de colonnes simples. Une colonne c_k appelée famille de colonnes, peut être composée de colonnes, c'est-à-dire constituée par une structure imbriquée S_k. Celle-ci suit une structure mono-valuée c'est-à-dire une structure plate à un seul niveau d'imbrication.

Exemple : Ci-dessous nous présentons une ligne r_i décrivant une personne.

```
r_i = (
      [id, personne[nom,prenom], daten[jour, mois, annee],
telephone[fixe,portable]],
      [id:12345,
       personne [nom:"el malki",prenom:"mohammed"],
       daten[jour:"24",mois:"08",annee:"1984"],
       telephone[fixe: "04", portable:"06"]
      )
```

Remarquons que dans l'approche orientée colonnes, il n'est pas possible de construire des colonnes multi-valuées. Dans l'exemple, nous ne pouvons modéliser la colonne telephone par un ensemble de colonnes {[numero]}.

Les sous-sections suivantes présentent 4 processus de traduction, en NoSQL orienté colonnes, d'un entrepôt de données multidimensionnelles en constellation.

3.3.2.2 Processus de traduction plate en orienté colonnes

A chaque fait $F\hat{I}\ F^S$ et ses dimensions associées $Star^S(F)$ correspond une table de même nom (N^F, E^F). Ce premier processus repose sur une dénormalisation des données du schéma en constellation. Chaque instance du fait est traduite par une ligne r_i. Les mesures du fait et les attributs de dimensions sont traduits en colonnes simples au sein d'une unique famille de colonnes.

Une ligne r_i est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de la ligne,
- chaque mesure de M^F forme un attribut m_k^F simple de la famille de colonnes,
- chaque attribut de $\bigcup_{D_j \in Star^S(F)} A^{D_j}$ forme un attribut a_k simple.

A l'instar des implantations ROLAP, nous ne matérialisons pas au niveau logique les attributs extrémités all^D des dimensions.

Exemple : Considérons le schéma conceptuel de la Figure 16. Chaque instance du fait F_{Tweet} est traduite par une ligne. L'expression ci-dessous présente la table obtenue et détaille une ligne :

```
{
...,
```

```
ri = (
      [id,
       Tweet[idUser,name,language,time_c,time_z,
             city,country,population,zone,
             day,month,month_name,year,
             topic,category,
             Retweet_NB]
      ],
      [id:12345,
       Tweet:[
          idUser:”C02265”,name:”Smith”,language:”french”,
       time_c:”Paris”,time_z:”France”,
          city:”Paris”,country:”France”,population:66000000,
       zone:”Europe-Ouest”,
          day:”03-31-2015”,month:”03-2015”,month_name:”march”,
       year:2015,
          topic:”foot”,category:”sport”,
          Retweet_NB:200]],
      ),
...
}
```

La Figure 21 présente sous une forme graphique la ligne r_i. Elle est constituée d'une identifiant (*row key*) et d'une famille de colonnes.

Figure 21 Exemple de ligne par traduction plate

Cette implantation évite l'utilisation des jointures entre le fait et les dimensions, mais génère un important niveau de redondance par duplication des données des dimensions entre les différentes lignes.

3.3.2.3 Processus de traduction par imbrication en orienté colonnes

Comme précédemment, à chaque fait FÎ F^S et ses dimensions associées $Star^S(F)$ correspond une table de même nom (N^F, E^F). Chaque instance du fait est traduite par une ligne r_i. Ce processus distribue les colonnes issues des mesures, et les colonnes issues de chaque dimension dans différentes familles de colonnes.

Une ligne r_i est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de la ligne,
- une famille de colonnes est définie pour le fait, et chaque mesure de M^F forme une colonne m_k^F imbriquée dans la famille,
- une famille de colonnes est définie pour chaque dimension, chaque attribut $a_k^{D_j}$ de la dimension D_j forme une colonne imbriquée dans la famille.

Exemple : Considérons la ligne décrite dans l'exemple précédent, que nous restructurons ci-dessous en fonction de l'approche par imbrication :

```
{
...,
r_i = (
      [id,
       User[idUser,name,language,time_c,time_z],
       Location[city,country,population,zone],
       Time[day,month,month_name,year],
       Subject[topic,category],
       Tweet[Retweet_NB]],
      [id:12345,
       User:[idUser:"C02265",name:"Smith",language:"french",
time_c:"Paris", time_z:"France"],
       Location:[city:"Paris",country:"France",population:66000000,
zone:"Europe-Ouest"],
       Time:[day:"03-31-2015",month:"03-2015",month_name:"march",
year:2015],
       Subject:[topic:"foot",category:"sport"],
       Tweet:[Retweet_NB:200]],
      ),
...
}
```

La Figure 22 présente sous une forme graphique la ligne r_i constituée par un ensemble de familles de colonnes issues du fait et de chaque dimension associée.

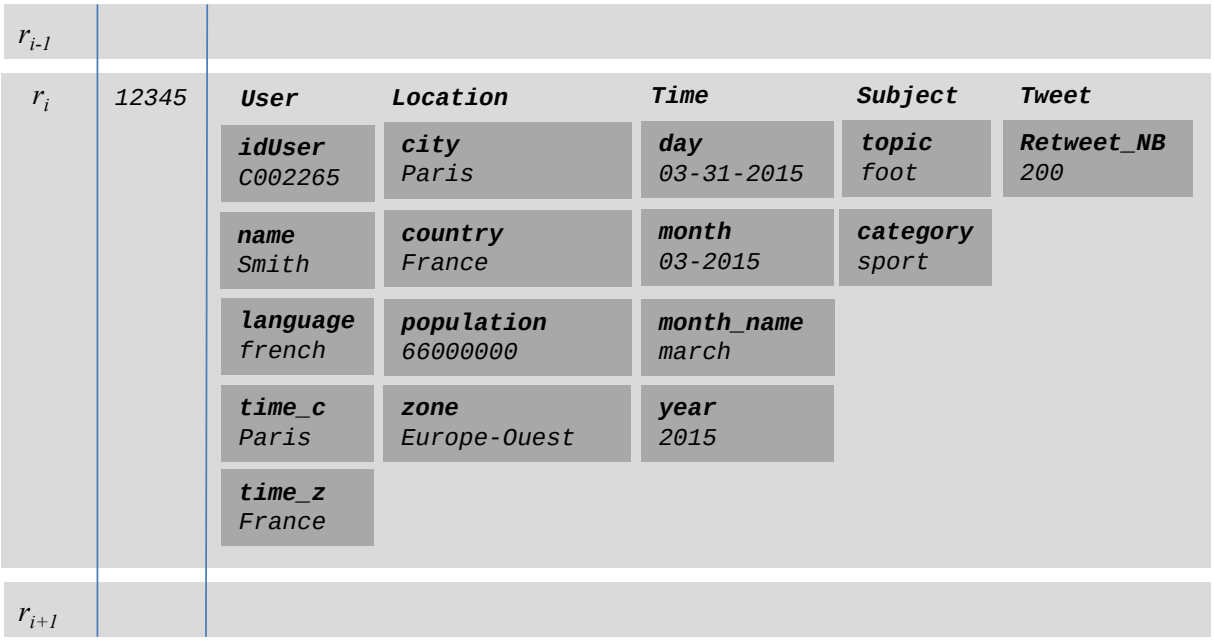


Figure 22 Exemple de ligne par traduction imbriquée

3.3.2.4 Processus de traduction hybride en orienté colonnes

L'approche hybride consiste à décomposer au sein de la table les données issues d'un fait Fï F^S et de chaque dimension associée $Star^S(F)$.

Les attributs des dimensions sont convertis en colonnes, sous une même famille de colonnes, et stockés comme une ligne de la table, à raison d'une ligne par dimension. Les mesures du fait sont converties en colonnes, sous une même famille de colonnes, et stockés sous forme de lignes de la même table. Les lignes des faits contiennent également les références aux lignes représentant les dimensions associées.

Pour chaque instance de dimension, une ligne r_i est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de ligne, correspondant à la racine id^{D_j} de la dimension,
- chaque attribut $a_k^{D_j}$ de la dimension D_j forme une colonne imbriquée dans une unique famille de colonnes de même nom que la dimension.

Pour chaque instance du fait, une ligne r_i est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de ligne,
- chaque mesure de M^F forme une colonne m_k^F imbriquée dans une unique famille de colonnes de même nom que le fait,
- pour chaque dimension D_jÎ $Star^S(F)$ associée, une colonne id^{D_j} est ajoutée référençant une ligne liée issue d'une dimension.

Exemple : Considérons l'exemple de document précédent. Dans l'approche hybride, une ligne est constituée pour chaque instance des dimensions ainsi qu'une autre pour l'instance du fait. Nous obtenons les lignes définies ci-dessous :

```
{
...,
r_i^Tweet = (
      [id,
       idUser,city,day, topic,
       Retweet_NB],
      [id:12345,
       idUser:"C02265", city:"Paris", day:"03-31-2015", topic:"foot",
       Retweet_NB:200],
      ),
r_i^User = (
      [idUser,name,language,time_c,time_z],
      [idUser:"C02265",name:"Smith",language:"french",time_c:"Paris",
time_z:"France"],
      ),
r_i^Location = (
      [city,country,population,zone],
      [city:"Paris",country:"France",population:66000000,
zone:"Europe-Ouest"],
      ),
r_i^Time = (
      [day,month,month_name,year],
      [day:"03-31-2015",month:"03-2015",month_name:"march",
year:2015],
      ),
r_i^Subject= (
      [topic,category],
      [topic:"foot",category:"sport"],
      ),
...
}
```

La Figure 23 présente sous une forme graphique les lignes r_i^{Tweet}, d_i^{User}, $r_i^{Location}$, $r_i^{Subject}$, $r_i^{Subject}$ constituées par une famille de colonnes issues d'attributs respectivement du fait et de chaque dimension.

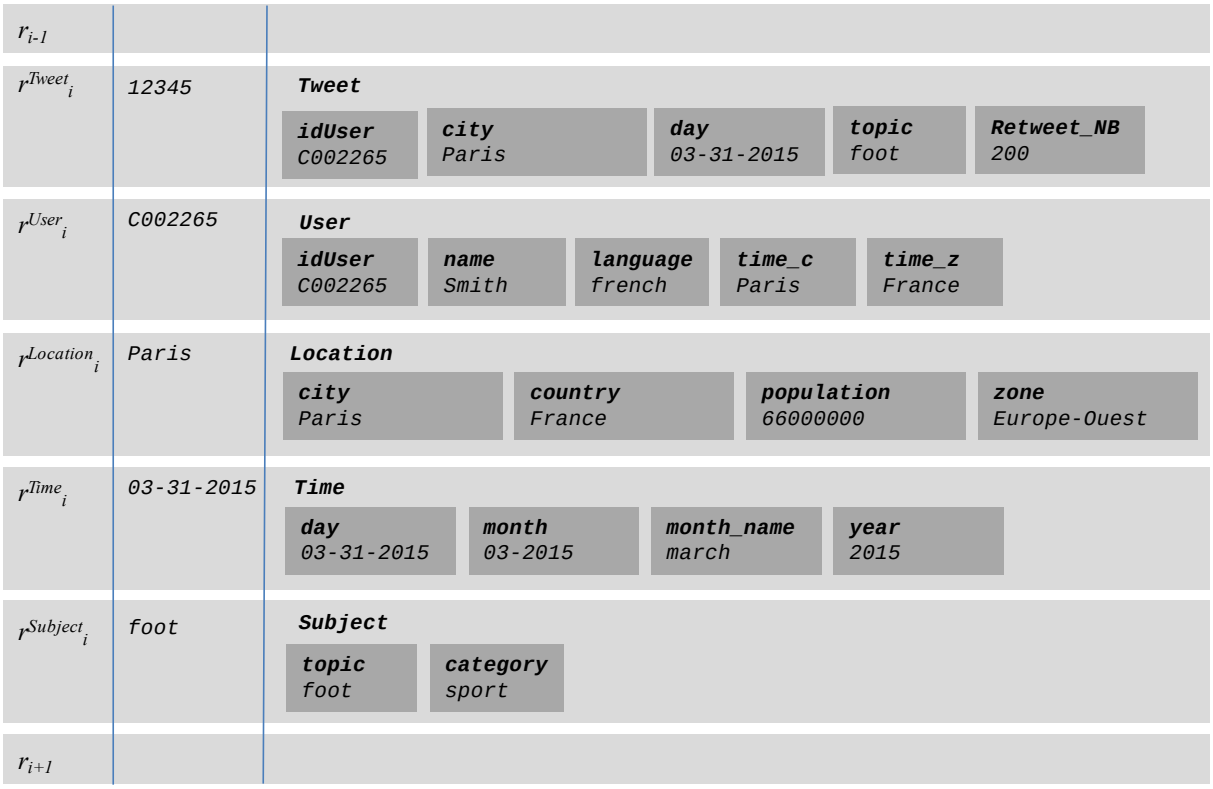


Figure 23 Exemple de ligne par traduction hybride

Cette implantation limite la redondance des données des dimensions des deux modèles précédents, mais introduit une hétérogénéité des structures des lignes placées dans la même table. L'approche hybride nécessite l'utilisation d'auto-jointures pour retrouver les documents de lignes liées aux lignes du fait.

3.3.2.5 Processus de traduction éclatée en orienté colonnes

La traduction éclatée distribue dans plusieurs tables les données : les données issues d'un fait FÎ F^S sont placées dans une première table, et les données de chaque dimension associée $Star^S(F)$ sont placées dans des tables distinctes (une table par dimension).

Les attributs des dimensions sont convertis en colonnes, sous une même famille de colonnes, et stockés dans une ligne de la table dédiée, T^{Dj}. Les mesures du fait sont converties en colonnes, sous une même famille de colonnes, et stockés dans une ligne d'une autre table T^F. Les lignes des faits contiennent également les références aux lignes représentant les dimensions associées, stockées dans les tables des dimensions.

Pour chaque instance de dimension, une ligne r_iÎ T^{Dj} est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de la ligne, correspondant à la racine id^{Dj} de la dimension,
- chaque attribut a_k^{Dj} de la dimension D_j forme une colonne imbriquée dans une unique famille de colonnes de même nom que la dimension.

Pour chaque instance du fait, une ligne r_iÎ T^F est définie par un schéma S_i constitué de la manière suivante :

- *id* est l'identifiant de la ligne,

- chaque mesure de M^F forme une colonne m_k^F imbriquée dans une unique famille de colonnes de même nom que le fait,
- pour chaque dimension $D_j \hat{I}\ Star^S(F)$ associée, une colonne id^{D_j} est ajoutée référençant une ligne liée issue de la table de la dimension.

Exemple : Considérons l'exemple de ligne précédent. Dans l'approche éclatée, une ligne est constituée pour chaque instance des dimensions ainsi qu'une autre ligne pour l'instance du fait. Ces lignes sont placées dans des tables distinctes. Nous obtenons les lignes définies ci-dessous :

```
{
...,
r_i^Tweet = (
      [id,
       idUser,city,day, topic,
       Retweet_NB],
      [id:12345,
       idUser:"C02265", city:"Paris", day:"03-31-2015", topic:"foot",
       Retweet_NB:200],
      ),
...
}
{
...,
r_i^User = (
      [idUser,name,language,time_c,time_z],
      [idUser:"C02265",name:"Smith",language:"french",time_c:"Paris",
time_z:"France"],
      ),
...
}
{
...,
r_i^Location = (
      [city,country,population,zone],
      [city:"Paris",country:"France",population:66000000,
zone:"Europe-Ouest"],
      ),
...
```

```
}
{
...,
r_i^Time = (
      [day,month,month_name,year],
      [day:"03-31-2015",month:"03-2015",month_name:"march",
year:2015],
      ),
...
}
{
...,
r_i^Subject = (
      [topic,category],
      [topic:"foot",category:"sport"],
      ),
...
}
```

La Figure 24 présente sous une forme graphique les lignes r_i^{Tweet},d_i^{User}, $r_i^{Location}$, $r_i^{Subject}$, $r_i^{Subject}$ appartenant respectivement aux tables T^{Tweet}, T^{User}, $T^{Location}$, T^{Time} $T^{Subject}$ dont la construction est homogène pour toutes ses lignes. Les lignes sont constituées par une famille colonnes issues respectivement du fait et de chaque dimension.

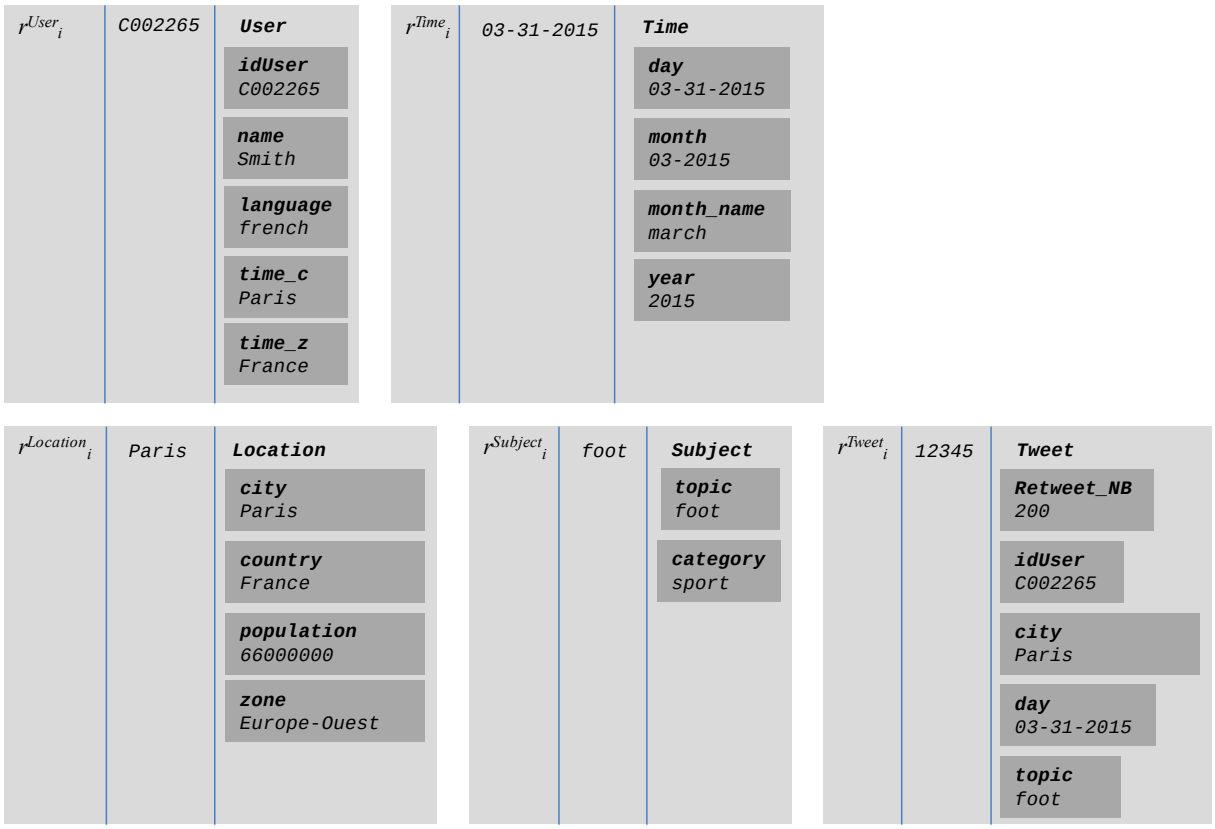


Figure 24 Exemple de documents par traduction éclatée

Comme l'approche précédente, cette implantation réduit la redondance des données des dimensions et place les lignes dans différentes tables structurellement homogènes. L'approche éclatée nécessite l'utilisation de jointures pour retrouver les lignes de dimensions des différentes tables liées aux lignes du fait.

3.4 OPTIMISATION PAR CUBE OLAP

Les analyses nécessitent parfois des requêtes complexes pouvant provoquer des temps de réponses importants. Un temps d'exécution acceptable varie de quelques secondes à quelques minutes au maximum [Harinarayan et al. 1996]. En bases de données, deux approches sont souvent utilisées : la création d'index (accélérant les accès aux données) et la matérialisation de vues (pré-calculs d'agrégats).

- **Création des index.** Les index sont des structures de données du niveau physique permettant d'accélérer les recherches, les tris, les jointures et les regroupements des données. Ils sont utilisés dans les bases de données relationnelles de manière générale et dans les systèmes décisionnels en particuliers. Leur rôle est de réduire le temps de réponse [Chaudhuri and Shim 1994]. Il existe plusieurs stratégies d'indexation dont à titre d'exemples : l'index bitmap, l'index de jointure [O'Neil and Graefe 1995] et index de jointure en étoile [O'Neil et al. 2009a]. Ces travaux ne se situent pas dans le cadre de nos études.

- **Matérialisation de vues.** La technique couramment utilisée en entrepôt de données consiste à matérialiser (pré-calculer et stocker) des requêtes fréquemment demandées. Ces requêtes (ou pré-agrégats) sont organisées dans un treillis [Harinarayan et al. 1996] où les nœuds représentent les agrégats et les arcs représentent les relations de

dépendance entre eux (nœuds calculables à partir d'autres nœuds). Sélectionner le bon ensemble de pré-agrégats à stocker est une question essentielle dans la conception de l'entrepôt de données [Teste 2000], car matérialiser un pré-agrégat améliore les temps de réponses aux requêtes, mais au détriment de l'espace de stockage et des temps de mises à jour.

Dans le modèle relationnel, l'espace de stockage est un paramètre plus contraint que dans un contexte NoSQL. Il existe trois stratégies pour déterminer l'ensemble des pré-agrégats à matérialiser [Ullman 1996] organisé hiérarchiquement en treillis :

- Ne matérialiser aucun pré-agrégat. Cette stratégie n'apporte aucune optimisation pour améliorer les performances de calculs des requêtes.
- Matérialiser tous les pré-agrégats. Cette approche coûteuse en espace de stockage, offre les meilleures performances lors de l'interrogation. A l'inverse, elle impacte fortement le temps des mises à jour.
- Matérialiser un sous-ensemble optimal de pré-agrégats du treillis. Cette stratégie est un compromis entre l'amélioration du temps de réponse aux requêtes des utilisateurs, et le surcoût induit en terme de stockage et de maintenance des mises à jours [Teste, 2000] [Zhuge and Garcia-Molina 1998].

Dans cette section, nous étudions la construction du treillis de pré-agrégats dans les modèles NoSQL orientés documents et orientés colonnes. L'architecture distribuée et extensible sur laquelle repose ces systèmes NoSQL rend plus envisageable le choix de matérialiser tous les pré-agrégats du treillis.

3.4.1 Définition d'un cube OLAP

Le treillis d'agrégats pré-calculés correspond à un ensemble de cuboïdes, chacun étant un sous ensemble de mesures d'un fait $F \hat{I} F^S$, 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 (par exemple, *obtenir le nombre total de tweets par catégorie, par utilisateur et par mois*). Les cuboïdes sont pré-calculés et stockés à l'avance pour accélérer le temps de réponse des requêtes en anticipant le calcul du résultat. Typiquement, les données des mesures sont regroupées selon un sous-ensemble de dimensions (par exemple, *grouper par catégorie et par mois*) et des fonctions d'agrégation sont utilisées pour agréger les données des mesures (par exemple, *somme du nombre de retweets*) selon les groupes créés.

Définition : Un treillis d'agrégats, appelé **cube OLAP**, est construit à partir d'un fait $F \hat{I} F^S$. 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.

Définition : Un **cuboïde** v_i dans un treillis L est défini par (P_i, M_i^F) où :

- $P_i = \left\{ \forall D_j \in Star^S(F), p_k^{D_j} \right\}$ est un ensemble de paramètres issus de chaque dimension $Star^S(F)$ à raison d'un paramètre par dimension,

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

Exemple : Considérons le cube OLAP L^{Tweet} à partir du fait Tweet. La Figure 25 représente le cube L^{Tweet} limité aux paramètres racines des quatre dimensions $Star^S(Tweet)$, c'est-à-dire id^{User}, *city*, *topic*, *day* respectivement issus de *User*, *Location*, *Subject* et *Time*.

La mesure du fait dans chaque cuboïde est calculée à l'aide de deux fonctions d'agrégation, *sum* et *max*. Nous n'abordons pas dans cette thèse les problèmes liés à l'additivité des fonctions d'agrégation [Hassan et al. 2012] [Hassan 2014].

Ainsi dans cet exemple, les cuboïdes obtenus sont définis comme suit :

($\{id^{User}$,*city*,*topic*,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$,*city*,*topic*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$,*city*, *day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$, *topic*,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*city*,*topic*,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$,*city*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}$,*topic*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*city*,*topic*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*city*,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*topic*,*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{id^{User}\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*city*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*topic*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{$*day*$\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

($\{\}$,$\{$*sum*(*Retweet_NB*),*max*(*Retweet_NB*)$\}$)

Le cuboïde noté *all* dans la Figure 25 correspond à l'agrégation de la mesure sans dimension. Chaque strate du treillis correspond à une analyse multidimensionnelle de zéro à quatre dimensions de haut en bas.

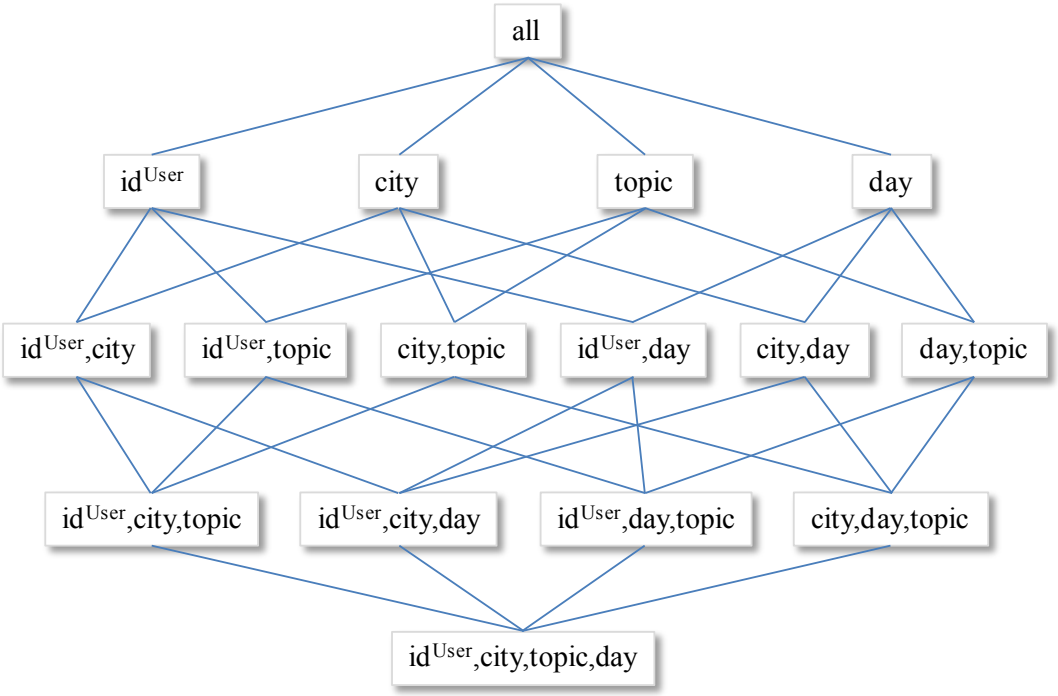


Figure 25 Exemple de cube OLAP par treillis de cuboïdes (ou pré-agrégats)

Il est possible d'utiliser d'autres paramètres que le paramètre racine. Pour simplifier, dans notre contexte nous limitons les cuboïdes à au plus un paramètre par dimension liée.

Dans les approches classiques, les cubes OLAP matérialisés sont généralement que partiels, c'est-à-dire que seulement un sous-ensemble des cuboïdes est pré-agrégé. Par exemple, la Figure 26 représente une matérialisation partielle du cube OLAP limitée aux trois dimensions *User*, *Subject* et *Time*.

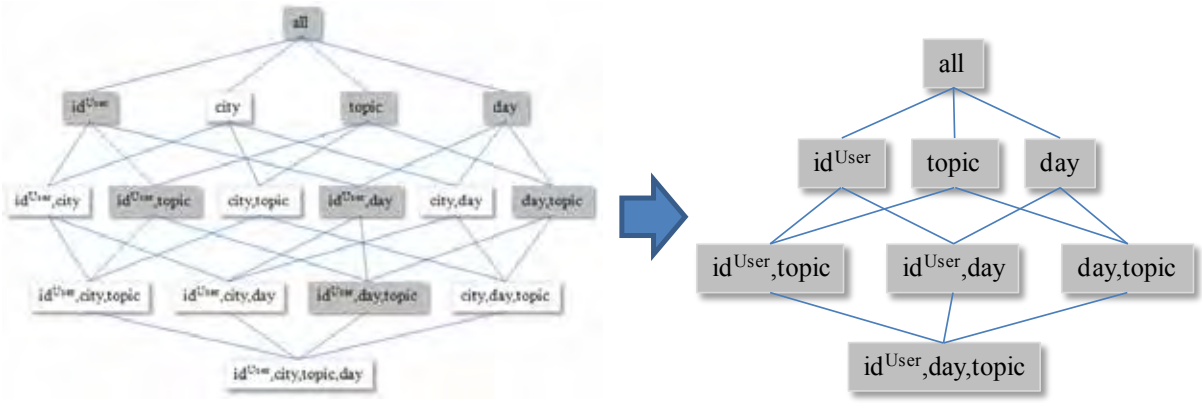


Figure 26 Exemple de matérialisation partielle du cube OLAP

Exemple : Dans la figure suivante, nous présentons un exemple d'instances pré-calculées pour chaque cuboïde constituant le cube OLAP partiel présenté à la Figure 26, et limité à une seule mesure *sum*(*NB_Retweet*).

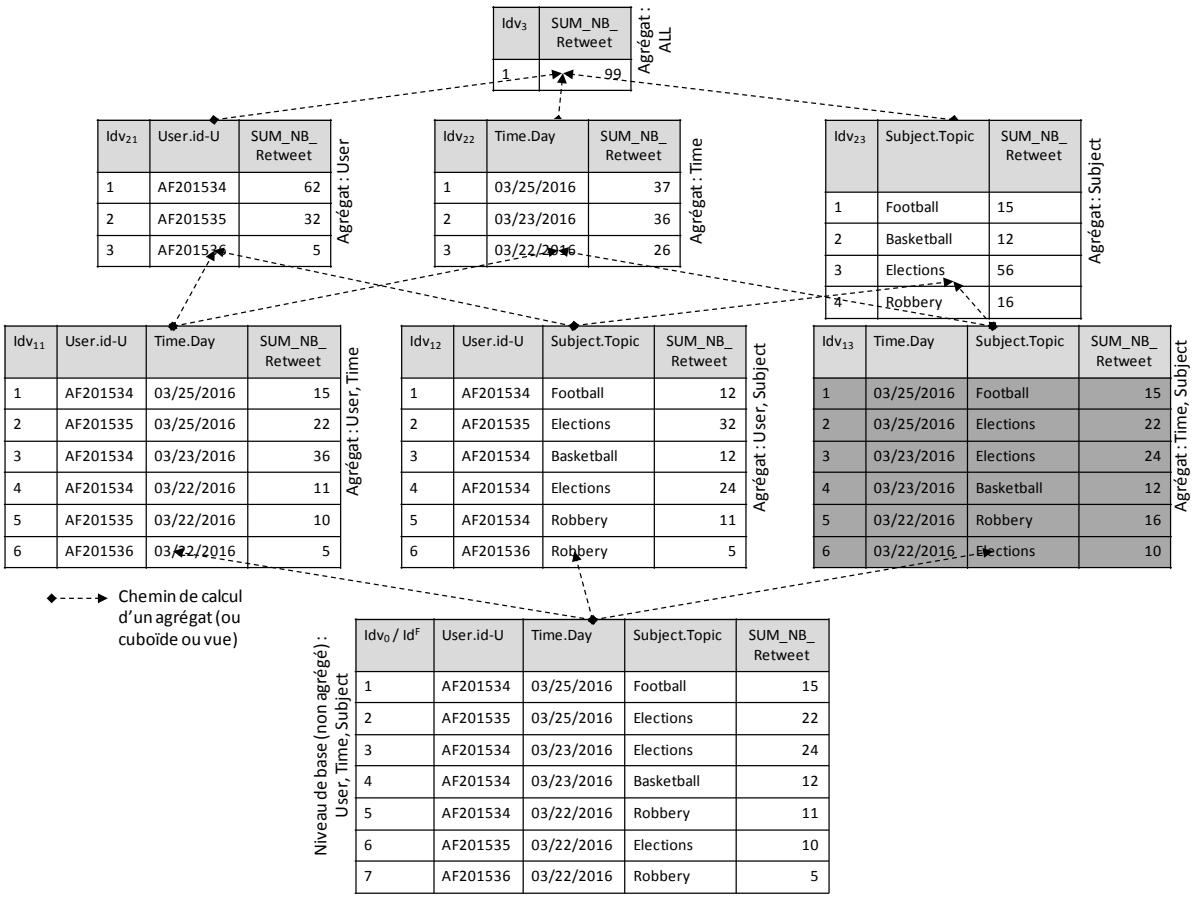


Figure 27 Exemple de matérialisation partielle du cube OLAP

Dans la suite, nous présentons les processus de traduction NoSQL du cube OLAP dans les deux systèmes cibles de nos travaux, orienté documents et orienté colonnes.

3.4.2 Processus de traduction en orienté documents

Nous définissons dans ce qui suit des règles de transformation du cube OLAP dans le modèle orienté documents. Chaque cuboïde v_i dans un treillis L est stocké dans une collection spécifique. Pour chaque instance du cuboïde, un document est créé dans la collection.

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

- *id* est un identifiant du document,
- chaque paramètre $p_k^{D_j} \hat{\imath} P_i$ est traduit par un attribut composé $D_j[p_k^{D_j}]$,
- chaque mesure $f_{k1}(m_{k2}) \in M_i^F$ est traduite par un attribut composé $F[f_{k1}\_m_{k2}]$.

Exemple : Considérons le cuboïde ({*topic*,*day*},{*sum*(*Retweet*) }) présenté en gris dans la Figure 26. Ce dernier est traduit par une collection de documents. Les 6 instances donnent lieu à 6 documents dans la collection.

Par exemple la première instance est implantée dans un document d_k défini comme suit :

d_k = (

[id,

```
 Time[day],
 Subject[topic],
 Tweet[sum_Retweet_NB]],
[id:1,
 Time:[day:03/25/2016],
 Subject:[topic:football],
 Tweet:[sum_Retweet_NB:15]]
)
```

3.4.3 Processus de traduction en orienté colonnes

Nous définissons dans ce qui suit des règles de transformation du cube OLAP dans le modèle orienté colonnes. Chaque cuboïde v_i dans un treillis L est stocké dans une table spécifique. Pour chaque instance du cuboïde, une ligne est créée dans la table.

Une ligne r_k est définie par un schéma S_k constitué de la manière suivante :

- *id* est un identifiant de la ligne,
- chaque paramètre $p_k^{D_j} \hat{\mathrm{I}}\ P_i$ est traduit par une famille D_j de colonnes $p_k^{D_j}$,
- chaque mesure $f_{k1}(m_{k2}) \in M_i^F$ est traduite par une famille F de colonnes $f_{k1}\_m_{k2}$.

Exemple : Considérons le même cuboïde ({*topic*,*day*},{*sum*(*Retweet*) }) présenté en gris dans la Figure 26. Ce dernier est traduit de manière analogue, par une table. Les 6 instances donnent lieu à 6 lignes. Par exemple la première instance est implantée dans une ligne r_k définie comme suit :

r_k =
```
(
 [id,
  Time[day],
  Subject[topic],
  Tweet[sum_Retweet_NB]],
 [id:1,
  Time:[day:03/25/2016],
  Subject:[topic:football],
  Tweet:[sum_Retweet_NB:15]]
)
```

3.5 BILAN

Ce chapitre décrit un processus pour implanter automatiquement un entrepôt de données multidimensionnelles dans un système NoSQL. Nous avons défini un processus qui traduit un schéma multidimensionnel conceptuel (schéma en constellation) dans deux modèles logiques NoSQL orthogonaux : orienté documents (stockage horizontal des données) ou orienté colonnes (stockage vertical des données). Nous avons proposé quatre processus de traduction pour chaque modèle logique :

- le processus de traduction plate qui permet un stockage totalement dénormalisé et évitant toute jointure entre les données des faits et des dimensions. Les données sont redondantes et de structure homogène.
- le processus de traduction par imbrication qui permet une structuration des mesures et des attributs. Les données sont redondantes et de structure homogène.
- le processus de traduction hybride normalise les données des dimensions, mais nécessite l'usage d'auto-jointures. Les données sont non redondantes mais de structure hétérogène.
- le processus de traduction éclaté normalise également les données des dimensions. Il nécessite l'utilisation de jointures entre les structures homogènes et sans redondance engendrées.

Le Tableau 6 présente une synthèse des processus de traduction d'un schéma conceptuel vers les schémas logiques orientés documents et colonnes. Nous utilisons les notations suivantes pour les règles vers le modèle orienté documents :

- C^F, C^D désignent une collection issue de F, respectivement D,
- m, a désignent un attribut simple issu de la mesure m, respectivement de l'attribut de dimension a,
- N^F, N^D désignent un attribut composé de même nom que le fait F, respectivement la dimension D.

Nous utilisons les notations suivantes pour les règles vers le modèle orienté colonnes :

- T^F, T^D désignent une table,
- m, a désignent une colonne,
- N^F, N^D désignent une famille de colonnes.

Nous désignons par d_i les documents et r_i les lignes obtenus à partir des instances du fait et des dimensions. Chaque chemin $C.d_i.S_i$ exprime la structure commune S_i à l'ensemble des documents d_i de la collection C ; de même chaque chemin $T.r_i.S_i$ exprime la structure commune aux lignes r_i de la table T.

Tableau 6 : Synthèse des règles de traduction du modèle conceptuel multidimensionnel vers les modèles logiques NoSQL orientés documents et colonnes.

Modèle conceptuel	Modèle logique NoSQL							
	Plat		Imbriqué		Hybride		Eclaté	
	Document	Colonne	Document	Colonne	Document	Colonne	Document	Colonne
$"F \hat{I} F^S$, $"m \hat{I} M^F$	$C^F.d_i.m$	$T^F.r_i.N^F.m$	$C^F.d_i.N^F.m$	$T^F.r_i.N^F.m$	$C^F.d_i.m$	$T^F.r_i.N^F.m$	$C^F.d_i.m$	$T^F.r_i.N^F.m$
$"D \hat{I} Star^S(F)$, $"a \hat{I} A^D$	$C^F.d_i.a$	$T^F.r_i.N^F.a$	$C^F d_i.N^D.a$	$T^F.r_i.N^D.a$	$C^F.d_j.a$ $a^o id^D \triangleright C^F.d_i.a$	$T^F.r_j.N^D.a$ $a^o id^D \triangleright T^F.r_i.N^F.a$	$C^D.d_j.a$ $a^o id^D \triangleright C^F.d_i.a$	$T^D.r_j.N^D.a$ $a^o id^D \triangleright T^F.r_i.N^F.a$

Nous avons étendu les processus de traduction aux cubes OLAP. Les cubes OLAP sont une technique très développée dans les entrepôts de données ROLAP afin d'optimiser ces derniers en pré-calculant les requêtes les plus fréquentes ou coûteuses. Ainsi nous avons défini des processus de traduction complète ou partielle d'un ensemble de cuboïdes constituant un cube OLAP, soit dans le modèle orienté documents, soit dans le modèle orienté colonnes.

Ces travaux ont fait l'objet de plusieurs publications [Chevalier, Malki, et al. 2015b] [Chevalier, Malki, et al. 2015c] [Chevalier, Malki, et al. 2015f] [Chevalier et al. 2016b].

Les modèles NoSQL se caractérisent par une forte dépendance au traitement : l'efficacité d'une structure de données dépendant du type de requêtes appliquées (contrairement à la modélisation relationnelle dont les formes normales sont définies indépendamment des traitements). Ainsi, suivant le type de requêtes, le choix de structuration peut être différent ; dans le pire cas. Il est même parfois nécessaire de concevoir un entrepôt de données combinant deux implantations logiques ou plus, chacune favorisant un sous-ensemble de requêtes liées aux traitements. Ces approches ont été proposées sous le nom de *multi-store* [Bondiombouy and Valduriez 2016] [LeFevre et al. 2014a], [Hacígümüş et al. 2013], [Atzeni et al. 2012], [Atzeni et al. 2009]. Ces systèmes *multi-store* sont des systèmes de stockage combinant différents modèles NoSQL (et même relationnel) de stockage. Ils nécessitent l'utilisation mutuelle de plusieurs bases de données avec leurs langages d'interrogation et techniques de traitement de requêtes [Bondiombouy and Valduriez 2016].

Dans le chapitre suivant, nous étudions un des axes de l'approche *multi-store* qui consiste à migrer les données selon des processus de conversion d'une implantation logique NoSQL vers une autre implantation logique NoSQL.

4. CHAPITRE IV : PROCESSUS DE CONVERSION INTRA-MODELES ET INTER-MODELES

4.1 INTRODUCTION

Il est difficile de choisir une seule implantation NoSQL supportant efficacement tous les traitements applicables [Hacígümüş et al. 2013] car les modèles NoSQL remettent en cause la séparation entre données et traitements, qui caractérise le modèle relationnel.

Une solution possible pour répondre à l'évolution des traitements, qui peuvent rendre un choix de modélisation inapproprié, consiste à convertir les modèles les uns vers les autres afin de migrer les données. Ces conversions peuvent être utilisées pour combiner deux systèmes [Hacígümüş et al. 2013] [LeFevre et al. 2014b] [Kolev et al. 2016]. Cette approche s'apparente à une approche multi-stores, capable d'offrir, d'une manière transparente, l'accès et l'interrogation des données partagées entre plusieurs systèmes de stockages.

Cela nécessite plusieurs processus de migration de données entre les différents systèmes de stockage. La Figure 28 résume les processus de transformation qui ont été introduits au chapitre III. Les travaux de ce chapitre se focalisent sur des processus de conversion entres les modèles logiques définis dans le chapitre III.

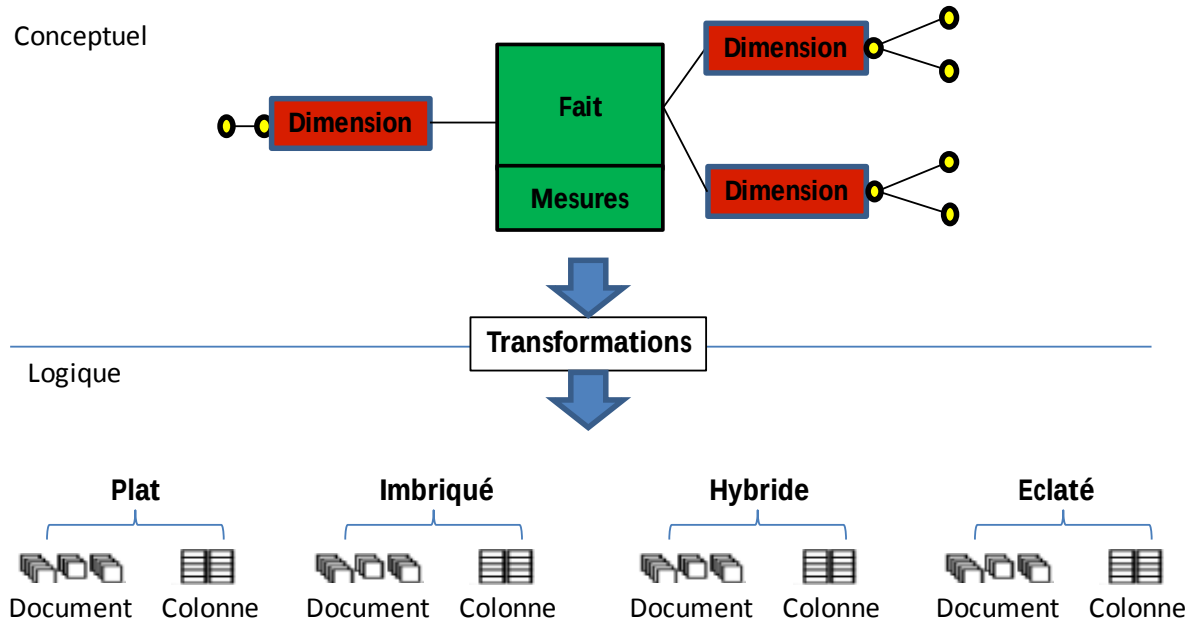


Figure 28 Processus de transformations conceptuelle - logique

Deux types de processus sont proposés :

- **les processus intra-modèles** reposent sur des règles de passage d'une implantation vers une autre implantation au sein d'un même modèle *NoSQL,* orienté documents ou orienté colonnes ;

- **les processus inter-modèles** reposent sur des règles de passage du modèle orienté documents vers le modèle orienté colonnes, et inversement.

4.2 PROCESSUS DE CONVERSION INTRA-MODELES

Dans cette partie, nous définissons les processus intra-modèles dans le système orienté documents. Nous proposons un ensemble de règles pour la conversion de données entre implantations logiques orientées documents. L'ensemble de ces règles opérant au niveau logique permettent de convertir directement les structures logiques, sans passer par une retro-conception.

4.2.1 Conversions intra-modèles orientés documents

La Figure 29 illustre les conversions directes entre les quatre modèles d'implantation orientée documents. Nous allons définir dans cette section les différentes règles de conversion.

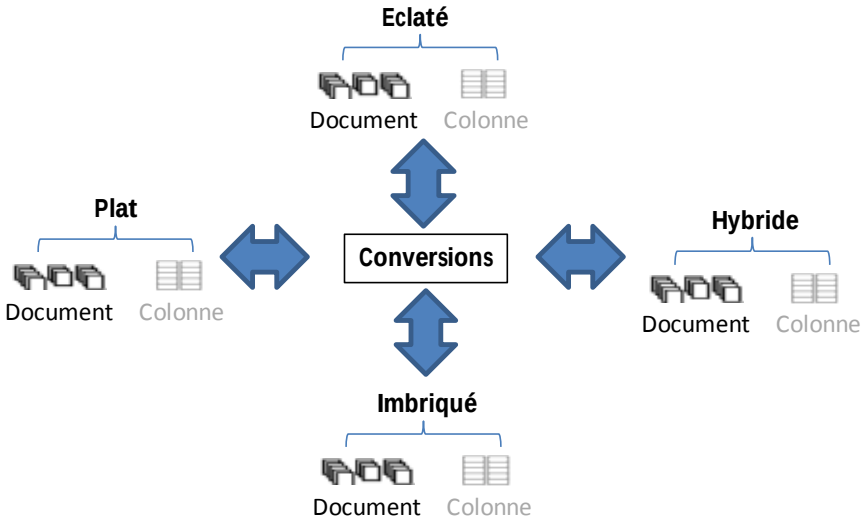


Figure 29 Processus de conversions logique orienté documents

Nous définissons douze règles de conversion intra-modèles orientés documents. Le Tableau 7 définit les règles de conversion. Chaque ligne désigne le modèle origine et chaque colonne le modèle cible vers lequel la conversion est effectuée. Nous spécifions la conversion pour chaque mesure ($"F \hat{I} F^S$, $"m \hat{I} M^F$) ainsi que pour chaque attribut de dimension ($D \hat{I} Star^S(F)$, $"a \hat{I} A^D$).

Tableau 7 Synthèse des règles de conversion intra-modèles orientés documents

			Schéma cible orienté documents			
			Plat	**Imbriqué**	**Hybride**	**Eclaté**
Schéma origine orienté documents	**Plat**	$C^F.d_i.m$ $C^F.d_i.a$		$C^F.d_i.\mathbf{N^F.m}$ $C^F\ d_i.\mathbf{N^D.a}$	$C^F.d_i.\mathbf{m}$ $C^F.\mathbf{d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$	$C^F.d_i.\mathbf{m}$ $\mathbf{C^D.d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$
	Imbriqué	$C^F.d_i.N^F.m$ $C^F\ d_i.N^D.a$	$C^F.d_i.\mathbf{m}$ $C^F.d_i.\mathbf{a}$		$C^F.d_i.\mathbf{m}$ $C^F.\mathbf{d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$	$C^F.d_i.\mathbf{m}$ $\mathbf{C^D.d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$
	Hybride	$C^F.d_i.m$ $C^F.d_j.a$ $a^o id^D \triangleright C^F.d_i.a$	$C^F.d_i.\mathbf{m}$ $C^F.d_i.\mathbf{a}$	$C^F.d_i.\mathbf{N^F.m}$ $C^F\ d_i.\mathbf{N^D.a}$		$C^F.d_i.\mathbf{m}$ $\mathbf{C^D.d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$
	Eclaté	$C^F.d_i.m$ $C^D.d_j.a$ $a^o id^D \triangleright C^F.d_i.a$	$C^F.d_i.\mathbf{m}$ $C^F.d_i.\mathbf{a}$	$C^F.d_i.\mathbf{N^F.m}$ $C^F\ d_i.\mathbf{N^D.a}$	$C^F.d_i.\mathbf{m}$ $C^F.\mathbf{d_j.a}$ $a^o id^D \triangleright C^F.d_i.\mathbf{a}$	

Nous détaillons les différents processus de conversion du tablau précédent pour chaque modèle origine.

Règle de conversion du modèle plat : une collection C^F implantée suivant le modèle à plat, est constituée de documents dont la structure est comme suit :

- $\forall F \in F^S, \forall m \in M^F, C^F.d_i.m$
- $\forall D \in Star^S(F), \forall a \in A^D, C^F.d_i.a$

Leur conversion vers les trois modèles cibles repose sur les règles ci-dessous.

Imbriqué :

- Chaque attribut issu d'une mesure *m* est placé dans une structure imbriquée N^F.
- Chaque attribut issu d'une dimension *a* est placé dans une structure imbriquée N^D.

Hybride :

- Le document d_i est converti en plusieurs documents placés dans la même collection : un document d_i et un document d_j pour chaque dimension liée.
- Chaque attribut issu d'une mesure *m* est placé dans une structure plate
- Chaque attribut issu d'une dimension *a* est placé dans une structure plate, mais dans un document différent. Lorsque l'attribut est racine $a^o id^D$, il est également ajouté dans la structure du document d_i.

Eclaté :

- Le document d_i est converti de manière analogue, en plusieurs documents, mais ces derniers sont placés dans des collections distinctes, une pour le fait C^F, et une pour chaque dimension liée C^D.
- Chaque attribut issu d'une mesure *m* est placé dans une structure plate.

- Chaque attribut issu d'une dimension a est placé dans une structure plate, mais dans un document différent de la collection. Lorsque l'attribut est racine $a^{o}id^{D}$, il est également ajouté dans la structure du document d_i.

Règle de conversion du modèle imbriqué : une collection C^F implantée suivant le modèle imbriqué, est constituée de documents dont la structure est comme suit :

- $\forall F \in F^S, \forall m \in M^F, C^F.d_i.N^F.m$
- $\forall D \in Star^S(F), \forall a \in A^D, C^F\ d_i.N^D.a$

Leur conversion vers les trois modèles cibles repose sur les règles ci-dessous.

Plat :

- Chaque mesure de N^F est « désimbriquée ».
- Chaque attribut de N^D est « désimbriqué ».

Hybride :

- Le document d_i est converti en plusieurs documents placés dans la même collection : un document d_i et un document d_j pour chaque dimension liée.
- Chaque attribut issu d'une mesure m est « désimbriquée ».
- Chaque attribut issu d'une dimension a est « désimbriquée », mais dans un document différent. Lorsque l'attribut est racine $a^{o}id^{D}$, il est également ajouté dans la structure du document d_i.

Eclaté :

- Le document d_i est converti de manière analogue, en plusieurs documents, mais ces derniers sont placés dans des collections distinctes, une pour le fait C^F, et une pour chaque dimension liée C^D.
- Chaque attribut issu d'une mesure m est « désimbriquée ».
- Chaque attribut issu d'une dimension a est « désimbriquée », mais dans un document différent d_j d'une collection différente C^D. Lorsque l'attribut est racine $a^{o}id^{D}$, il est également ajouté dans la structure du document d_i.

Règle de conversion du modèle hybride : une collection C^F implantée suivant le modèle imbriqué, est constituée de documents dont la structure est comme suit :

- $\forall F \in F^S, \forall m \in M^F, C^F.d_i.m$
- $\forall D \in Star^S(F), C^F.d_j.a$ et $a^{o}id^{D} \rhd C^F.d_i.a$

Leur conversion vers les trois modèles cibles repose sur les règles ci-dessous.

Plat :

- Les documents d_i et d_j sont regroupés en un seul document d_i.
- Chaque mesure m est maintenue à plat dans le document d_i. On pourra noter que les attributs dupliqués issus des racines ne sont pas convertis (supprimés).
- Chaque attribut a d'un document d_j est replacé dans le document d_i.

Imbriqué :

- Les documents d_i et d_j sont regroupés en un seul document d_i.

- Chaque attribut issu d'une mesure *m* est placé dans une structure imbriquée N^F. On pourra noter que les attributs dupliqués issus des racines ne sont pas convertis (supprimés).
- Chaque attribut *a* d'un document d_j est replacé dans le document d_i, dans une structure imbriquée N^D.

Eclaté :

- Les documents d_i et d_j sont distribués dans des collections différentes C^F et C^D.

Règle de conversion du modèle éclaté : les collections C^F et C^D implantées suivant le modèle éclaté, sont constituées de documents dont la structure est comme suit :

- $\forall F \in F^S, \forall m \in M^F, C^F.d_i.m$
- $\forall D \in Star^S(F), C^D.d_j.a$ et $a \circ id^D \rightarrow C^F.d_i.a$

Leur conversion vers les trois modèles cibles repose sur les règles ci-dessous.

Plat :

- Les documents d_i et d_j sont regroupés en un seul document d_i placé dans une même collection C^F.
- Chaque mesure *m* est maintenue à plat dans le document d_i. On pourra noter que les attributs dupliqués issus des racines ne sont pas convertis (supprimés).
- Chaque attribut *a* d'un document d_j est replacé dans le document d_i.

Imbriqué :

- Les documents d_i et d_j sont regroupés en un seul document d_i placé dans une même collection C^F.
- Chaque attribut issu d'une mesure *m* est placé dans une structure imbriquée N^F. On pourra noter que les attributs dupliqués issus des racines ne sont pas convertis (supprimés).
- Chaque attribut *a* d'un document d_j est replacé dans le document d_i, dans une structure imbriquée N^D.

Hybride :

- Les documents d_i et d_j sont regroupés dans une collection C^F.

Exemple : Nous n'illustrons qu'un cas de conversion d'un modèle imbriqué vers le modèle hybride. Nous reprenons l'exemple du document d_i du chapitre précédent.

La structure S_i initiale du document est la suivante :

```
[id,
     User[idUser,name,language,time_c,time_z],
     Location[city,country,population,zone],
     Time[day,month,month_name,year],
     Subject[topic,category],
     Tweet[Retweet_NB]
```

]

Cette structure S_i est convertie suivant les règles énoncées précédemment. Plusieurs documents sont construits :

- un document d_i^{Tweet} pour les attributs issus de la structure Tweet[Retweet_NB] (correspondante au fait), et
- quatre documents $d_j^{User}, d_j^{Location}, d_j^{Time}, d_j^{Subject}$ pour chaque structure User[idUser, name, language, time_c, time_z], Location[city, country, population, zone], Time[day, month, month_name, year], Subject[topic, category] (correspondantes aux quatre dimensions liées).

Les structures de chaque document sont définies comme suit.

S_i^{Tweet} = [id,idUser,city,day,topic,Retweet_NB]

S_j^{User} = [idUser,name,language,time_c,time_z]

$S_j^{Location}$ = [city,country,population,zone]

S_j^{Time} = [day,month,month_name,year]

$S_j^{Subject}$ = [topic,category]

La Figure 30 présente un exemple de conversion du document complet, c'est-à-dire, de son schéma et des valeurs.

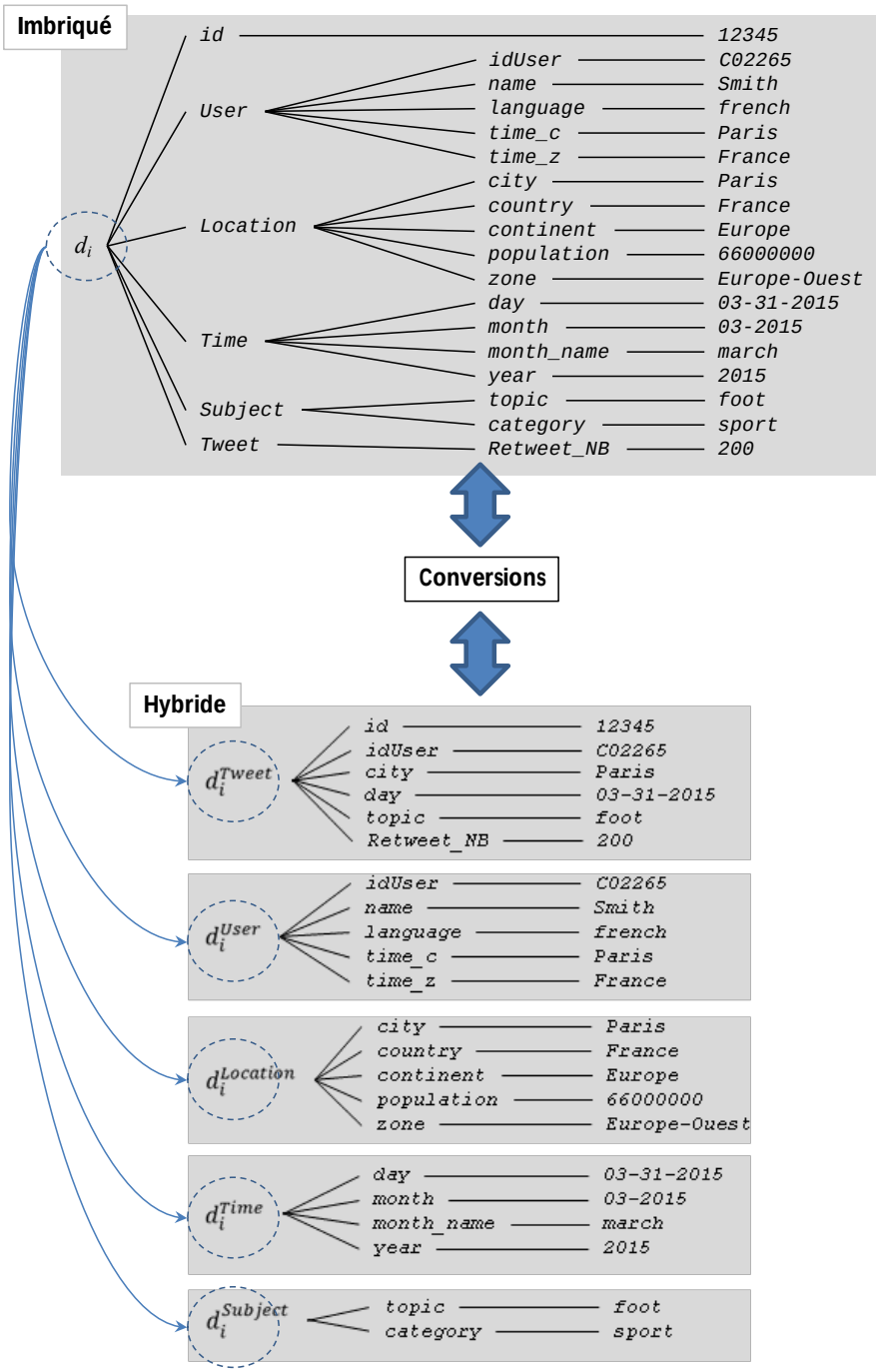


Figure 30 Exemple de conversion intra-modèle en orienté documents, du modèle imbriqué vers le modèle hybride

4.2.2 Conversions intra-modèles orientés colonnes

Dans cette partie, nous définissons les processus intra-modèles dans le système orienté colonnes. Comme précédemment, nous proposons un ensemble de règles pour la conversion de données entre implantations logiques orientées colonnes. La Figure 31 illustre les conversions directes entre les quatre modèles d'implantation orientée colonnes.

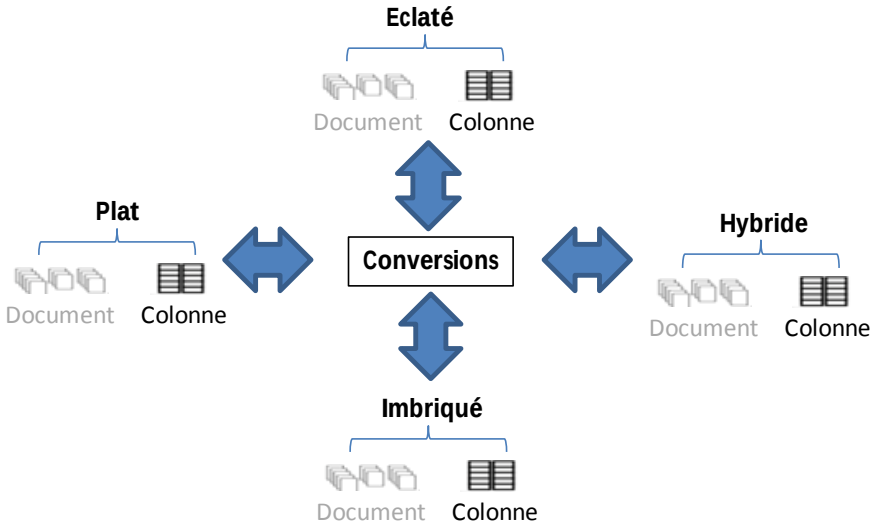


Figure 31 Processus de conversions logique orienté colonnes

Comme pour les conversions intra-modèle orienté documents, nous définissons douze règles de conversion intra-modèle orienté colonnes. Le Tableau 8 définit ces règles de conversion.

Tableau 8 Synthèse des règles de conversion intra-modèles orientés colonnes

			Schéma cible orienté colonnes			
			Plat	**Imbriqué**	**Hybride**	**Eclaté**
Schéma origine orienté colonnes	**Plat**	$T^F.r_i.N^F.m$ $T^F.r_i.N^F.a$		$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^D.a}$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.\mathbf{r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.N^F.a$	$T^F.r_i.\mathbf{N^F.m}$ $\mathbf{T^D.r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.\mathbf{N^F.a}$
	Imbriqué	$T^F.r_i.N^F.m$ $T^F.r_i.N^D.a$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^F.a}$		$T^F.r_i.\mathbf{N^F.m}$ $T^F.\mathbf{r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.N^F.a$	$T^F.r_i.\mathbf{N^F.m}$ $\mathbf{T^D.r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.\mathbf{N^F.a}$
	Hybride	$T^F.r_i.N^F.m$ $T^F.r_j.N^D.a$ $a\,^o\!id^D$ Þ $T^F.r_i.N^F.a$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^F.a}$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^D.a}$		$T^F.r_i.\mathbf{N^F.m}$ $\mathbf{T^D.r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.\mathbf{N^F.a}$
	Eclaté	$T^F.r_i.N^F.m$ $T^D.r_j.N^D.a$ $a\,^o\!id^D$ Þ $T^F.r_i.N^F.a$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^F.a}$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.r_i.\mathbf{N^D.a}$	$T^F.r_i.\mathbf{N^F.m}$ $T^F.\mathbf{r_j.N^D.a}$ $a\,^o\!id^D$ Þ $T^F.r_i.N^F.a$	

Pour ne pas alourdir la lecture de ce mémoire, nous ne décrivons pas ici le détail des conversions. Toutefois nous présentons un exemple pour illustrer une des conversions possibles.

Exemple : Nous présentons la conversion d'un modèle hybride vers le modèle plat en orienté colonnes. La table origine comporte 5 lignes de structure hétérogène conformes aux schémas suivants :

S_i^{Tweet} = [id,idUser,city,day, topic,Retweet_NB]

S_j^{User} = [idUser,name,language,time_c,time_z]

$S_j^{Location}$ = [city,country,population,zone]

S_j^{Time} = [day,month,month_name,year]

$S_j^{Subject}$= [topic,category]

Les règles convertissent les 5 lignes en une seule ligne dont le schéma est défini comme suit.

S_i = [id,
Tweet[idUser,name,language,time_c,time_z,
city,country,population,zone,
day,month,month_name,year,
topic,category,
Retweet_NB]]

La Figure 32 présente un exemple de conversion complète des cinq lignes avec les valeurs associées.

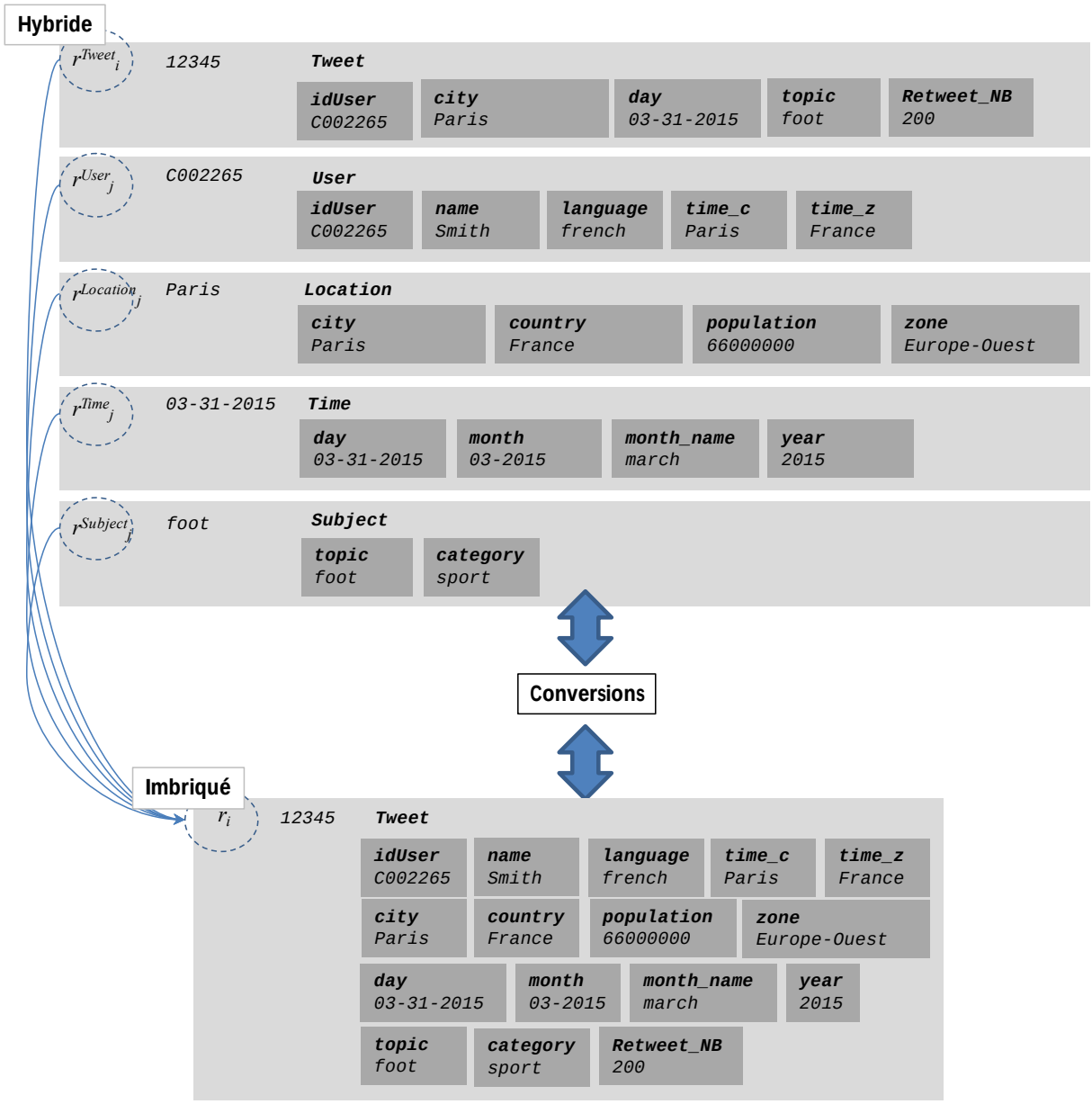


Figure 32 Exemple de conversion intra-modèle en orienté colonnes, du modèle hybride vers le modèle plat

4.3 PROCESSUS DE CONVERSION INTER-MODELES

Cette dernière section se focalise sur les conversions inter-modèles, en l'occurrence les conversions des implantations NoSQL orienté colonnes vers les implantations du modèle NoSQL orienté documents et inversement. Nous définissons des processus de conversion pour les implantations plate, imbriquée, hybride et éclatée.

4.3.1 Conversions inter-modèles orientés documents et colonnes

Il existe 16 conversions inter-modèles. Nous présentons dans le Tableau 9 l'ensemble des règles de conversion.

Tableau 9 Synthèse des règles de conversion inter-modèles

			Schéma orienté documents			
			Plat	**Imbriqué**	**Hybride**	**Eclaté**
Schéma orienté colonnes	**Plat**	$T^F.r_i.N^F.m$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{N^F.m}$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{m}$
		$T^F.r_i.N^F.a$	$C^F.d_i.\mathbf{a}$	$C^F\ d_i.\mathbf{N^D.a}$	$C^F.\mathbf{d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$	$\mathbf{C^D.d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$
	Imbriqué	$T^F.r_i.N^F.m$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{N^F.m}$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{m}$
		$T^F.r_i.N^D.a$	$C^F.d_i.\mathbf{a}$	$C^F\ d_i.\mathbf{N^D.a}$	$C^F.\mathbf{d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$	$\mathbf{C^D.d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$
	Hybride	$T^F.r_i.N^F.m$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{N^F.m}$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{m}$
		$T^F.r_j.N^D.a$ $a^o id^D$ ⊳ $T^F.r_i.N^F.a$	$C^F.d_i.\mathbf{a}$	$C^F\ d_i.\mathbf{N^D.a}$	$C^F.\mathbf{d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$	$\mathbf{C^D.d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$
	Eclaté	$T^F.r_i.N^F.m$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{N^F.m}$	$C^F.d_i.\mathbf{m}$	$C^F.d_i.\mathbf{m}$
		$T^D.r_j.N^D.a$ $a^o id^D$ ⊳ $T^F.r_i.N^F.a$	$C^F.d_i.\mathbf{a}$	$C^F\ d_i.\mathbf{N^D.a}$	$C^F.\mathbf{d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$	$\mathbf{C^D.d_j.a}$ $a^o id^D$ ⊳ $C^F.d_i.\mathbf{a}$

Nous présentons deux exemples pour illustrer les conversions inter-modèles.

- Le premier exemple, présente une conversion du modèle orienté documents vers le modèle orienté colonnes, avec le même type d'implantation plate.
- Le second exemple, présente en sens inverse une conversion du modèle orienté colonnes vers le modèles orienté documents, d'une implantation hybride vers une implantation imbriquée.

Exemple 1 : Nous effectuons une conversion d'un document d_i en une ligne r_i, les deux étant structurés de manière plate.

Le schéma du document noté $d_i.S_i$, est constitué d'un ensemble d'attributs à plat comme le montre l'expression ci-dessous.

```
d_i.S_i = [id,
      idUser,name,language,time_c,time_z,
      city,country,population,zone,
      day,month,month_name,year,
      topic,category,
      Retweet_NB]
```

Le schéma de la ligne noté $r_i.S_i$, est structuré par une famille de colonnes nommée *Tweet*. Les attributs du document sont convertis en colonnes.

```
r_i.S_i = [id,
        Tweet[idUser,name,language,time_c,time_z,
              city,country,population,zone,
```

```
            day,month,month_name,year,
            topic,category,
            Retweet_NB]
    ],
```

La Figure 33 illustre cette conversion en faisant apparaître les valeurs du document origine et de la ligne obtenue par conversion inter-modèles.

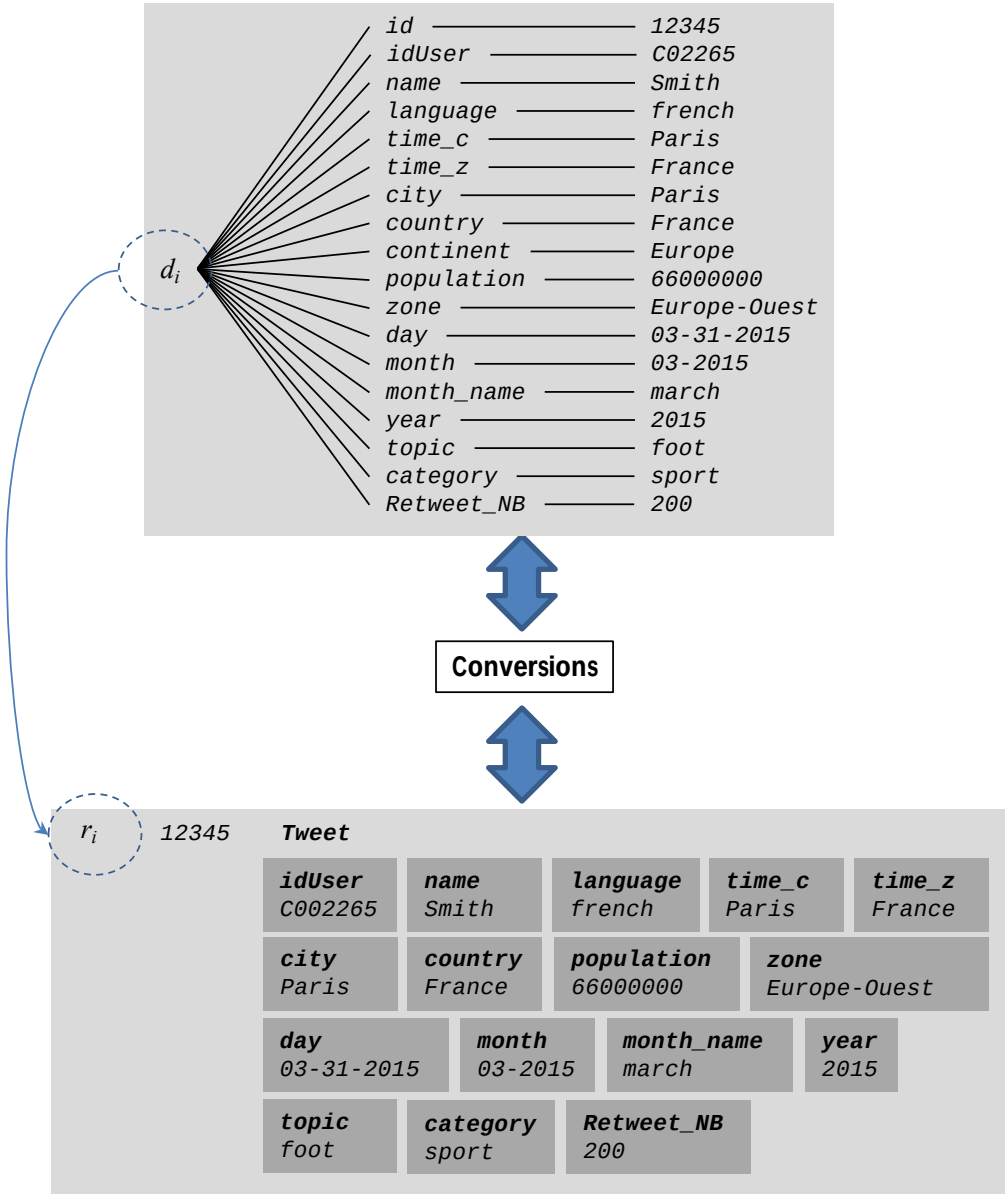


Figure 33 Exemple de conversion inter-modèles de l'orienté documents vers l'orienté colonnes, en implantation plate

Exemple 2 : Ce deuxième exemple illustre inversement une conversion du modèle orienté colonne vers le modèle orienté documents. Le processus convertit 5 lignes d'une même table (implantation hybride) en un document imbriqué.

La table origine comporte 5 lignes de structure hétérogène conformes aux schémas suivants :

S_i^{Tweet} = [id,idUser,city,day, topic,Retweet_NB]

S_j^{User} = [idUser,name,language,time_c,time_z]

$S_j^{Location}$ = [city,country,population,zone]

S_j^{Time} = [day,month,month_name,year]

$S_j^{Subject}$= [topic,category]

Le document obtenu est constitué d'un schéma S_i comme suit.

```
[id,
 User[idUser,name,language,time_c,time_z],
 Location[city,country,population,zone],
 Time[day,month,month_name,year],
 Subject[topic,category],
 Tweet[Retweet_NB]
]
```

La Figure 34 illustre cette conversion en faisant apparaître les valeurs des lignes d'origines placées dans la table, et les valeurs du document obtenu par conversion inter-modèles.

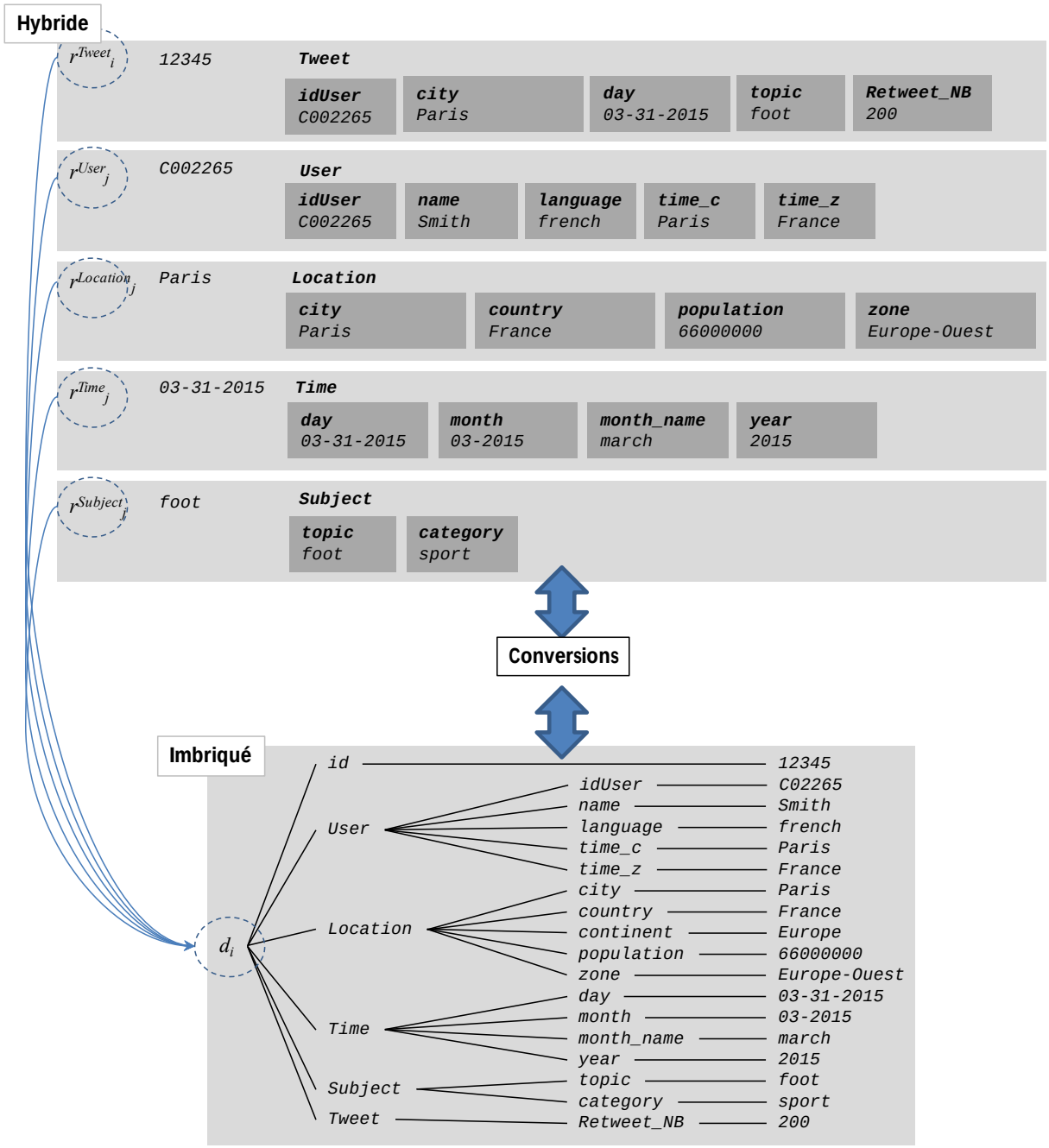


Figure 34 Exemple de conversion inter-modèles de l'orienté colonnes vers l'orienté documents, d'une implantation hybride vers une implantation imbriquée

4.3.2 Conversions inter-modèles des cubes OLAP

Nous définissons ici des règles inter-modèles, permettant de convertir des cubes OLAP, du modèle orienté documents vers celui orienté colonnes, et inversement.

Nous définissons deux règles de conversion inter-modèles. Le Tableau 10 définit ces règles de conversion.

Tableau 10 Synthèse des règles de conversion inter-modèles des cubes OLAP

			Cube OLAP cible	
			Orienté colonnes	Orienté documents
Cube OLAP d'origine	Orienté colonnes	$D_j[p_k^{D_j}]$		$D_j[p_k^{D_j}]$
		$F[f_{k1}\_m_{k1},.., f_v\_m_v]$		$F[f_{k1}\_m_{k1},.., f_v\_m_v]$
	Orienté documents	$D_j[p_k^{D_j}]$	$D_j[p_k^{D_j}]$	
		$F[f_{k1}\_m_{k1},.., f_v\_m_v]$	$F[f_{k1}\_m_{k1},.., f_v\_m_v]$	

Nous détaillons le processus de conversion du tableau précédent pour le cube OLAP orienté documents vers le cube OLAP orienté colonnes.

Règle de conversion du cube OLAP orienté documents : une collection est implantée pour chaque cuboïde suivant le cube OLAP orienté documents, est constitué de documents dont la structure est comme suit :

- $\forall p_k^{D_j} \in P_i, D_j[p_k^{D_j}]$
- $\forall f_v(m_v) \in M_i^F, F[f_{k1}\_m_{k1},.., f_v\_m_v]$

Leur conversion vers le cube OLAP orienté colonnes repose sur les règles ci-dessous.

- Chaque attribut simple issu d'une mesure forme une colonne placée dans une famille de colonne N^F.
- Chaque attribut composé issu des dimensions est placé dans famille de colonnes N^D.

Exemple : Nous illustrons un cas de conversion d'un cube OLAP orienté documents vers un cube OLAP orienté colonnes. Nous reprenons le cuboïde ({*topic*,*day*},{*sum*(*Retweet*) }) considéré dans l'exemple du chapitre III.

La structure initiale du document est la suivante :

```
d_i = (
       [id,
        Time[day],
        Subject[topic],
        Tweet[sum_Retweet_NB]],
      )
```

Les règles convertissent le document en une ligne dont le schéma est défini comme suit.

```
r_i = (
       [id,
        Time[day],
```

```
 Subject[topic],
 Tweet[sum_Retweet_NB]]
)
```

4.4 BILAN

Ce chapitre présente les processus de conversion au niveau logique. Nous avons présenté deux types de conversions :

- les conversions intra-modèle en orienté documents et en orienté colonnes. Elles consistent à convertir les données (documents ou lignes) entre les différentes structures d'implantation (plate, imbriquée, hybride et éclatée). Ces conversions restent au sein d'un même modèle logique NoSQL.
  Nous avons défini 24 règles de conversion.

- les conversions inter-modèles de l'orienté documents vers l'orienté colonnes et inversement. Ces conversions consistent à migrer les données entre deux modèles logiques NoSQL différents, et ceci en fonction des quatre structures d'implantation plate, imbriquée, hybride et éclatée.
  Nous avons défini 16 règles de conversion.

L'intérêt de ces conversions est notamment de permettre la cohabitation de plusieurs systèmes NoSQL pour essayer de bénéficier des avantages de chacun, on parle d'approche *multi-store* [Hacígümüş et al. 2013] [Bondiombouy and Valduriez 2016].

Dans notre contexte d'entrepôts de données multidimensionnelles, nous avons également définis des règles inter-modèles additionnelles, permettant de convertir les cubes OLAP, du modèle orienté documents vers celui orienté colonnes, et inversement.

Ces travaux ont fait l'objet de plusieurs publications [Chevalier et al. 2015] [Chevalier et al. 2015] [Chevalier, Malki, et al. 2015e]

Dans le chapitre suivant, présentons l'environnement d'expérimentations que nous avons mise en place pour valider nos propositions. Nous présentons la plateforme expérimentale, le cluster de machines, et le benchmark SSB+ que nous avons défini.

5. CHAPITRE V : ENVIRONNEMENT D'EXPERIMENTATIONS

Ce chapitre a pour objectif de présenter le prototype que nous avons développé, afin de montrer la faisabilité de nos approches et d'expérimenter nos propositions. L'objectif est de fournir un banc d'essai qui permet de simuler et d'évaluer un entrepôt de données multidimensionnelles, avec les différentes manipulations OLAP effectuées par un analyste.

5.1 INTRODUCTION

Différents bancs d'essai (*benchmarks*) ont été proposés pour comparer des systèmes de base de données. Ils fournissent des jeux de données et des scenarii d'utilisation permettant d'évaluer le comportement du système, dans des conditions équivalentes. Cependant les solutions existantes sont développées et optimisées pour les bases de données relationnelles et pour une utilisation sur une seule machine. Avec le développement des solutions Big Data, les systèmes distribués et les bases de données NoSQL sont rapidement devenus populaires. Dans ce contexte, nous avons besoin de solutions de bancs d'essai compatibles avec ces nouveaux systèmes NoSQL distribués sur plusieurs machines.

Nous focalisons notre attention sur les bancs d'essai utilisés dans le contexte des entrepôts de données multidimensionnelles et l'analyse OLAP où plusieurs solutions existent telles que le TPC-DS [Poess et al. 2007], TPC-H et le SSB [O'Neil et al. 2009b]. Ces solutions ne sont pas définies pour une utilisation dans un système distribué ou des bases de données NoSQL. Leur processus de génération de données est relativement sophistiqué et intéressant mais il nécessite beaucoup plus de temps lorsqu'il est question d'évaluer un large volume de données (Teraoctet et plus). Comparer les systèmes avec un volume de données important est devenu crucial. Plus le volume est important plus nous sommes confrontés aux limites de mémoire lors d'une configuration avec une seule machine. Les nouvelles solutions Big Data permettent le passage à l'échelle et pallient à ces problèmes de mémoire. Les données sont réparties sur plusieurs machines et lorsque la limite de stockage est atteinte, de nouvelles machines peuvent être facilement ajoutées. Cette technique est moins couteuse qu'augmenter la capacité de stockage d'une seule machine dont le coût devient important. Cette solution pratique n'est pas prise en charge par les bancs d'essai existants. Ils génèrent les données sur une seule machine.

Dans ce contexte, et en l'absence de benchmarks décisionnels conçus pour les systèmes NoSQL, nous proposons une extension du Star Schema Benchmark (SSB) qui

supporte les systèmes NoSQL. Le SSB est un banc d'essai très populaire dans le contexte d'aide à la décision (DSS). Nous étendons les fonctionnalités du système à la génération distribuées des données au niveau du système de fichiers distribués HDFS [Lee et al. 2012b]. Les données peuvent être générées en mode normalisé (adapté aux bases de données relationnelles) mais également en mode dénormalisé (adapté aux systèmes *NoSQL*). Les données peuvent être générées en différents formats (non exclusivement le csv). Générer des données directement dans un format spécifique est intéressant pour les solutions NoSQL dont le processus de chargement peut être facilité par un format approprié ; par exemple *MongoDB* supporte les fichiers JSON.

Les nouvelles fonctionnalités se résument ainsi :

- Permettre une génération de données pour différents systèmes de stockage relationnel et NoSQL ;
- Permettre la distribution et la génération parallèle des données ;
- Améliorer la précision de l'indicateur d'échelle (Scale Factor—sf) ;

La section suivante est consacrée à un rappel des bancs d'essai existants, la section 5.3 décrit le banc d'essai Star Schema Benchmark (SSB). Dans la section 5.4, nous proposons notre benchmark SSB+. Dans la section 5.5, nous présentons les différentes expérimentations menées avec ce nouveau banc d'essai et les comparaisons avec la version d'origine.

5.2 PANORAMA DES BANCS D'ESSAI DECISIONNELS

Il existe un nombre considérable de travaux dans le domaine des systèmes d'information. L'évolution et le développement des systèmes de stockage suscitent une évolution continue des bancs d'essai.

Nous distinguons deux familles de bancs d'essai, ceux utilisés dans le contexte d'aide à la décision et ceux dédiés au contexte Big Data. En ce qui concerne la première famille, nous détaillons les méthodes basées sur les bancs d'essai TPC-D qui s'intéressent aux systèmes d'aide à la décision. Quant à la deuxième famille d'approches, nous détaillons des bancs d'essai qui supportent les systèmes distribués.

Les **Bancs d'essai DSS (***Decision Support System***)** sont édités par TPC (*Transaction Processing Council*). Ils représentent les systèmes les plus utilisés pour évaluer les performances des bancs d'essai DSS. Le premier banc d'essai est l'APB-1 largement exploité dans les années 90, mais il est rapidement devenu obsolète et inadapté pour la plupart des cas d'évaluation expérimentales [Stevenson et al. 2006] [Darmont et al. 2007]. Par la suite, le banc d'essai TPC-D a été proposé, et est la première référence pour les systèmes d'aide à la décision, c'est le système à partir duquel ont été dérivés les deux systèmes TPC-H [Council 2008] et TPC-R. En plus d'être plus riche, le modèle de TPC-H est normalisé et permet de supporter une centaine de requêtes, qui peuvent être classées en 4 catégories : requêtes OLAP interactives, requêtes d'aide à la décision ad-hoc, requêtes d'extraction et requêtes de rapports. Le modèle de données est un schéma de constellation composé de 7 faits et de 17 dimensions partagées par les 7 faits. En 2009, un autre banc d'essai avec un schéma étoile a été proposé. Il s'agit de *SSB* (Start Schema Benchmark). Contrairement à [Poess et al. 2002] [Darmont 2007] [Nambiar and Poess 2006], SSB introduit un certain niveau de dénormalisation des données pour des raisons de simplicité. Il met en œuvre un schéma en étoile composé d'un fait et quatre dimensions. Nous rencontrons dans ce contexte, un des rares efforts pour adapter le benchmark SSB aux systèmes NoSQL, il s'agit du banc d'essai *CNSSB* [Dehdouh et al. 2014a] qui est proposé pour supporter les modèles de données

orientés colonnes. Il permet uniquement les évaluations dans un seul modèle NoSQL, le modèle orienté colonnes, et il propose seulement deux manières d'organiser les données.

Les bancs d'essai TPC demeurent la référence la plus utilisée pour les systèmes d'aide à la décision. Toutefois, ils sont basés sur le système relationnel et ne peuvent pas être facilement mis en œuvre dans des bases de données NoSQL.

Les **Bancs d'essai Big Data.** Ces bancs d'essai visent à comparer les nouveaux systèmes qui stockent des données dans des systèmes distribués massivement et supportent des calculs parallèles.

Le service Yahoo Cloud [Cooper et al. 2010] est l'un des outils les plus populaires. Il est utilisé pour comparer les opérations CRUD élémentaires (Create, Read, Update et Delete) sur le système NoSQL. Il a déjà été exploité pour la plupart des systèmes NoSQL et offre de bonnes performances en termes de chargement de données de mises à jour, etc [Cooper et al. 2010]. De la même manière, *Bigframe*[10] [Ivanov et al. 2015] est un banc d'essai de données qui se focalise principalement sur les problèmes de volume, la variété et la vélocité dans le contexte Big Data. Avec plus de fonctionnalités que les deux premiers, les auteurs de [Ghazal et al. 2013] proposent *BigBench*. Ce dernier modélise des lignes de commandes. Il comprend 3 types de données, des données structurées (issues du TPC-DS), semi-structurées (flux de clics dans les sites web), des données non structurées (commentaires client). Ce banc d'essai est destiné à être la référence la plus complète qui considère des défis importants en entrepôt de données.

Contrairement aux bancs d'essai traditionnels, les bancs d'essai Big Data sont orientés sur la flexibilité de l'information, des données massives et évolutives et n'évaluent pas les mêmes critères que les bancs d'essai DSS.

D'autres bancs d'essai ont été aussi proposés. HadoopToSQL [Iu and Zwaenepoel 2010] évalue les performances de MapReduce pour la charge de travail. Les requêtes MapReduce sont transformées pour utiliser l'indexation, le regroupement et l'agrégation des attributs fournis par les bases de données SQL. De la même façon, [Li et al. 2016] ont proposé la solution *YSmart*, un banc d'essai construit au-dessus de la plate-forme Hadoop qui permet de traduire des requêtes SQL en requêtes MapReduce. La traduction est assurée via un ensemble de fonctions MapReduce. De même, Moussa [Moussa 2012] propose de traduire le jeu de requêtes TPC-H de SQL en Pig Latin. Il propose une approche de construction des requêtes Pig pour maximiser les performances.

Dans ce chapitre, nous proposons un nouveau banc d'essai, par extension de SSB. Notre solution prend en charge les modèles NoSQL orienté colonnes et le modèle orientés documents.

5.3 LE BANC D'ESSAI SSB

Le banc d'essai Star schéma benchmark (SSB) est l'un des bancs d'essai les plus utilisés dans le contexte d'aide à la décision. Il est dérivé du banc d'essai TPC-H. Il 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 décrites en fonction de ses quatre axes d'analyses.

Plus précisément, on y observe :

[10] Pour plus d'informations sur BigFrame : https://github.com/bigframeteam/BigFrame/wiki/

- le fait, *Lineorder*;
- *Date*, *Supplier*, *Customer* et *Part* sont les dimensions associées au fait.

Le SSB est constitué de deux composantes logicielles :

- ***DBGEN* :** qui assure la génération des données avec différents facteurs d'échelle appelés *Scale Factor* (*SF*).
- ***QGEN* :** qui assure la génération d'un jeu de requêtes qui dépend des données générées.

Les données sont générées avec différents facteurs d'échelle (2GB, 10GB, 100GB, 1TB...), dans un format tabulaire tel que le format CSV, à raison d'un fichier par table. Ce format de données est adapté pour les bases de données relationnelles et les solutions NoSQL ne peuvent pas tirer directement avantage de ce format. Des données dénormalisées sont requises. De plus les approches NoSQL ne supportent pas nécessairement le format csv. Il existe aussi quelques problèmes liés à l'indicateur d'échelle [Dehdouh et al. 2014b] : la taille des données générées ne correspond pas à l'indicateur précisé, par exemple 580 Go sont réellement génèrées quand 1To de données est demandé.

La génération de données suit le schéma de la Figure 35. Si nous avons besoin de charger les données dans une base NoSQL, nous devons suivre un processus complexe comme le montre la Figure 35. Egalement les données sont générées dans des fichiers plats (un fichier par élément conceptuel) dans un format tabulaire csv. Les données doivent être dénormalisées. Nous devons traiter les données pour obtenir un seul fichier fusionnant les données provenant de ces différents fichiers. Cette tâche est complexe car elle nécessite le chargement des données dans une base de données relationnelles avant d'effectuer une projection et obtenir un fichier dénormalisé. Le résultat de cette projection doit être transformé dans un autre format adapté à l'outil de stockage utilisé. Par exemple, dans le cas de MongoDB, le fichier doit être transformé d'un format csv vers un format JSON. De plus, chaque outil NoSQL nécessite un script ou un outil de chargement spécifique comme *Mongoimport* pour MongoDB.

L'ensemble de ces étapes se résument comme suit (Figure 35) :

- Génération des fichiers plats ;
- Chargement des données dans les tables de la base de données relationnelles ;
- Faire une projection pour récupérer un seul fichier plat dénormalisé ;
- Convertir le fichier dénormalisé dans le format approprié de l'outil NoSQL utilisé ;
- Générer le script de chargement du système NoSQL utilisé.

Comme nous pouvons le voir à la Figure 35, la génération des données nécessite une phase d'extraction et de transformation s'accompagnant de limites importantes.

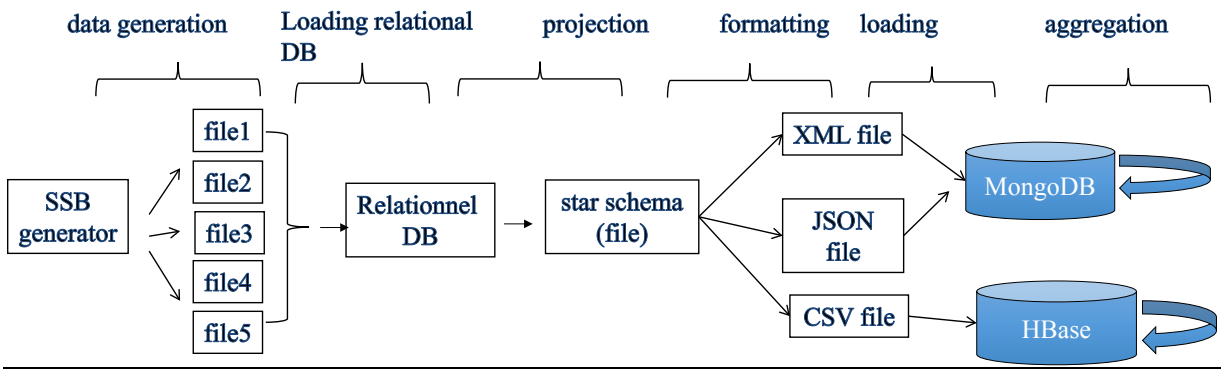


Figure 35 Processus d'utilisation du SSB dans le NoSQL

5.4 LE BANC D'ESSAI SSB+

Le Star Schema Benchmark Plus (SSB+) [] est le nom que nous attribuons à la version SSB adaptée aux systèmes NoSQL. Il modélise le même jeu de données que SSB (Figure 36). Ce nouveau banc d'essai considère plus de modèles de données. Il supporte les systèmes NoSQL cibles.

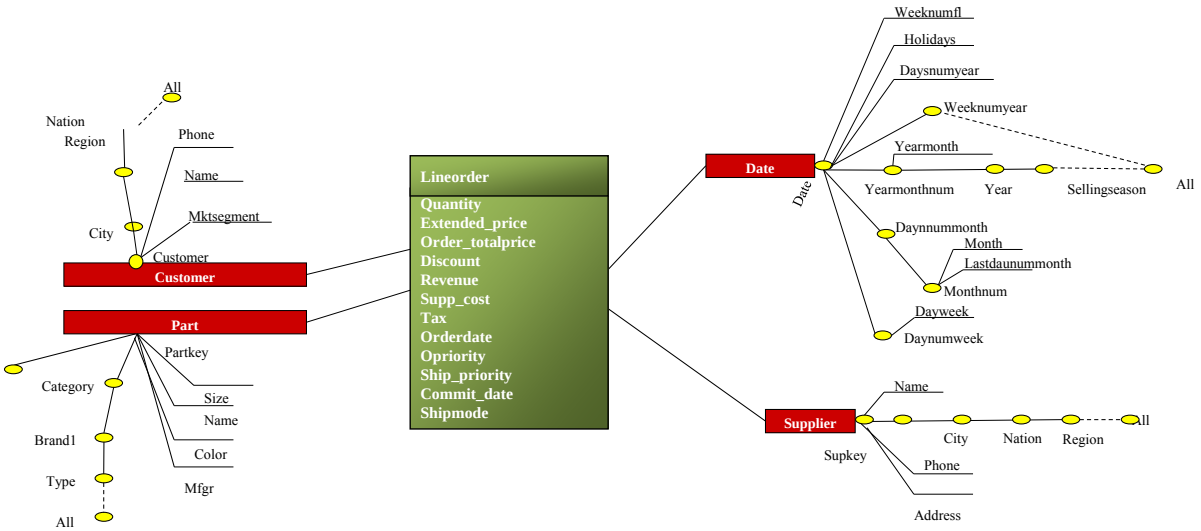


Figure 36 Le modèle conceptuel en étoile du SSB

Il inclut trois composantes logicielles :

- *DBGEN* : version étendue du générateur de données SSB ;
- *DBload* : outil de chargement ;
- *QGen* le générateur de requêtes.

DBGEN est utilisé pour la génération des données, il couvre les principales améliorations apportées dans la version SSB+. *DBLoad* est un outil utilisé pour la migration des données. Il est dépendant du système de stockage utilisé, il propose des scripts de chargement de données pour les systèmes NoSQL tels que HBase et MongoDB. Cette liste de scripts ne vise pas à être exhaustive et peut être progressivement enrichie par de nouveaux systèmes. Les requêtes sont générées en SQL via le générateur QGen. Les requêtes ne sont donc pas générées en langage spécifique à MongoDB ou à HBase. Toutefois, SQL est un langage déclaratif standard et il est toujours possible de traduire les requêtes dans le langage spécifique de l'outil utilisé [Moussa 2012].

Pour alimenter un système de stockage relationnel ou NoSQL nous suivons le processus de la génération des données illustré dans le schéma de la Figure 37. Ce dernier a été généralisé et simplifié. Les principales propriétés sont :

- les données peuvent être générées en mode normalisé (un fichier plat par élément conceptuel avec un total de 5 fichiers) ou en mode dénormalisé (un seul fichier dénormalisant toutes les données) ;
- les données peuvent être générées sur une seule machine ou sur le système distribué d'Hadoop, HDFS via MapReduce ;
- Les modèles NoSQL orienté colonnes et orienté documents sont supportés ;
- Trois formats de données sont supportés : CSV, XML et JSON.

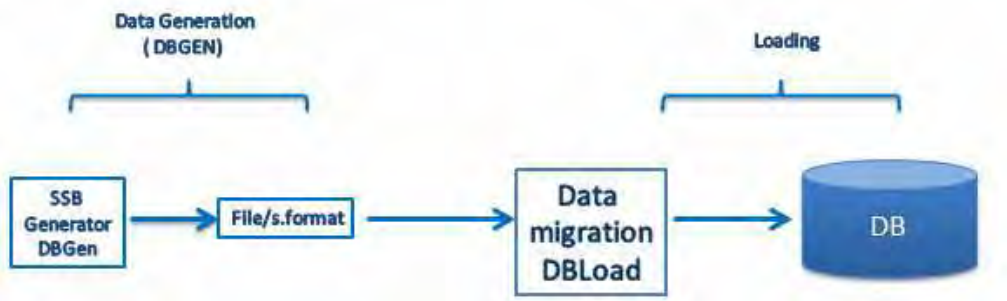


Figure 37 Schéma de chargement des bases de données avec le banc d'essai SSB+

Le nouveau banc d'essai supporte désormais les bases de données NoSQL. Il peut générer les données dans un système de fichiers distribués en l'occurrence le système HDFS. Le système HDFS est la plateforme la plus utilisée dans ces contextes et elle permet la parallélisation de la génération des données sur plusieurs machines.

C'est une amélioration qui simplifie le processus de génération des données. Avec la version précédente du SSB, nous étions confrontés à la limitation de l'espace de stockage lors d'une configuration avec une seule machine. Avec cette nouvelle version, nous pouvons générer les données dans une architecture distribuée permettant de générer de plus grandes quantités de données/

Nous décrivons dans ce qui suit les détails de la version étendue.

Normalisation vs dénormalisation des données. Le schéma de données supporté par le SSB est donné dans la Figure 36. Nous élargissons la génération à l'approche dénormalisée supportée par les systèmes NoSQL. Dans la version normalisée, les données sont générées dans cinq fichiers plats qui serviront à alimenter chacun une structure : *LINEORDER* (fait), *PART*, *SUPLLIER*, *DATE* ET *CUSTOMER* (dimensions). Dans l'autre cas d'utilisation, les données sont générées en un seul fichier appelé *Global.* Cette dénormalisation des données correspond à la dénormalistaion standard utilisée dans les entrepôts de données [Kimball and Ross 2013]. Le processus de dénormalisation requiert l'abandon des clés étrangères et l'ajout des données des autres tables. Le résultat obtenu est constitué de lignes plus importantes composée de 49 attributs : 11 mesures du fait et 38 attributs provenant des dimensions.

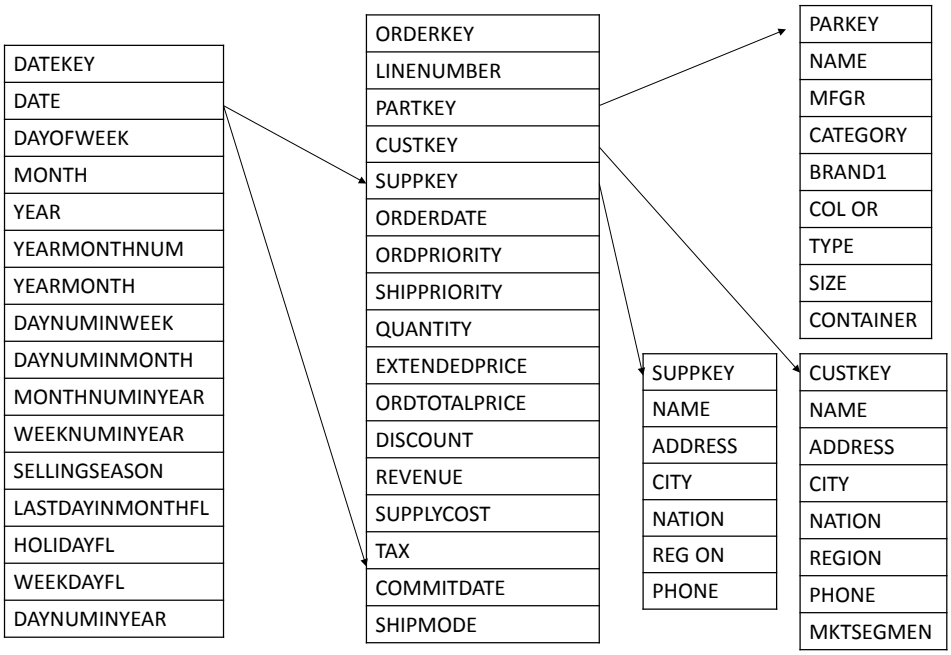


Figure 38 Schéma de données pour une génération normalisée des données

Le format des fichiers. Afin d'optimiser la phase de chargements des données dans les outils NoSQL, le générateur offre à l'utilisateur la possibilité de spécifier le format approprié par l'outil NoSQL utilisé. Par exemple, dans le modèle orienté documents, MongoDB est une base de données qui stocke les données en BJSON (binary JSON), elle est optimisée pour le format JSON. Charger un fichier de format CSV dans MongoDB est possible mais passera par une conversion en JSON ce qui augmentera considérablement le temps de chargement. Plusieurs formats de fichiers sont donc possibles : TBL, CSV, JSON et XML. La nécessité de spécifier le format de sortie concerne uniquement le modèle orienté documents ; le modèle orienté colonnes supporte très bien les fichiers tabulaires (.tbl ou .csv).

Parallélisation : le processus de génération est étendu pour fonctionner sur un système distribué, Hadoop.

Modèles : la génération supporte deux modèles NoSQL en plus du modèle relationnel, qui sont le modèle orienté colonnes et le modèle orienté documents étudiés dans les chapitres précédents.

- **Le modèle orienté documents** : La génération des données dans le modèle orienté documents constitue une nouvelle partie majeure dans le nouveau générateur DBGEN. Plusieurs implantations logiques orientées documents sont proposés. On retrouve les implantations logiques orientées documents décrites dans le chapitre III, l'implantation plate, l'implantation imbriquée et l'implantation éclatée. Pour chaque modèle un seul fichier dénormalisé est généré.
- **Le modèle orienté colonnes.** La génération des données dans le modèle orienté colonnes est la seconde partie principale du générateur. La donnée est générée selon les implantations orientés colonnes présentées dans le chapitre III.

5.5 AMELIORATION DE L'INDICATEUR D'ECHELLE

Le SSB génère les données avec différents volumes. L'indicateur d'échelle permet de spécifier le volume de données à générer. Par exemple pour générer un volume de données de 10GB il faudrait utiliser un indicateur d'échelle *sf=10*.

Comme évoqué précédemment, pour le modèle relationnel, 5 fichiers sont générés proportionnellement à l'indicateur d'échelle. Par exemple pour le fichier Customer, 30000×sf lignes sont générées. L'indicateur d'échelle impacte le nombre de lignes générées pour une table donnée (cf. Tableau 12). Seule la table *PART* n'est pas impactée car sa génération ne suit pas un *sf* linéaire ; le nombre de lignes est déterminé selon une formule logarithmique : $200000 \times (1+\log(sf))$.

Tableau 11 Mémoire disque par ligne par table

Table	Lignes	Mémoire en Bytes (*sf=10*)	Moyenne Mémoire par ligne (Byte)
Customer	$300000 \times sf$	29360128	97,87
Part	$800000 \times (1+\log_2(sf))$	69206016	86,51
Lineorder	$6 \times 10^6 \times sf$	6227702579	103,82
Supplier	$20000 \times sf$	1782579	89,13
Date	$2556 \times sf$	233472	91,34
Total		6328284774	-

Etant donné que les coefficients (sf) sont dérivés du banc d'essai TPC-H, les données générées ne correspondent pas à la spécification initiale. Le DBGEN génère entre 0.56 et 0.58 du volume demandé autrement dit, pour un indicateur d'échelle de 100 (*sf=100*) un volume de données de 56GB est obtenu.

Ce problème n'est pas simple à gérer quand on veut utiliser un benchmark générique. La taille des données est plus importante (3 à 4 fois plus volumineuses) quand elle est générée selon une approche dénormalisée. Pour pallier à ce problème deux solutions étaient envisageables :

- multiplier linéairement le coefficient par 1.69
- augmenter le nombre de lignes générer pour le *sf=1*. Le nombre de lignes passe de $6x10^6$ à 10^7.

Nous avons opté pour la seconde solution simple mais approximative. Un résultat de 9.8GB est obtenu pour un *sf=10*.

Tableau 12 L'impact de l'indicateur d'échelle sur le nombre de lignes du SSB

Fichier	Lignes	Meoire en Bytes (*sf=10*)	Moy.memoire par ligne (Byte)
Customer	$300000 \times sf$	29360128	97,87
Part	$800000 \times (1+\log_2(sf))$	69206016	86,51
Lineorder	$6 \times 10^6 \times sf$	6227702579	103,82
Supplier	$20000 \times sf$	1782579	89,13
Date	$2556 \times sf$	233472	91,34
Total	*6322560*	6328284774	-

5.6 LA DISTRIBUTION DES DONNEES

Le volume de données à générer est très important et la capacité d'une seule machine est rapidement dépassée, d'où la nécessité de distribuer les données sur plusieurs machines. La distribution se base sur les deux principaux composants du système de fichiers distribués Hadoop:

- **MapReduce** : assure la distribution du processus de génération des données
- **HDFS** : est utilisé pour stocker le résultat de chaque tache Map

Notons que nous avons uniquement besoin de la phase Map, la phase Reduce n'est pas nécessaire, nous n'assurons que la distribution de la génération des données. Comme présenté dans le schéma du processus de la génération de la donnée en mode distribué (Figure 39), le namenode assigne aux nœuds de calculs une tâche de type *map* qui génère les données.

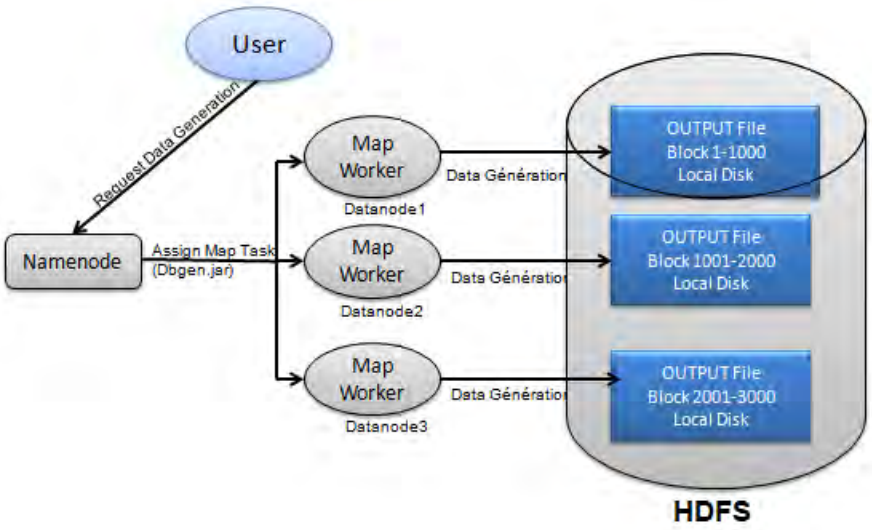


Figure 39 Génération des données en mode distribué

5.7 LE JEU DE REQUETES

QGEN : Le générateur de requêtes de SSB a un nombre important de requêtes qui ne peuvent être traduites dans un langage d'interrogation NoSQL du fait que certains critères n'ont pas à être évalués tels que les jointures. Nous avons donc opté pour la réalisation d'un générateur de requêtes composé de 12 requêtes. Les requêtes sont organisées selon deux critères :

1. La dimensionnalité qui affecte le nombre de dimensions dans les clauses de regroupement (équivalent du Group By dans le SQL) : 1D : 1 dimension, 2D, deux dimensions et 3D : 3 dimensions.
2. La sélectivité qui affecte le degré de filtrage les données quand des conditions sont appliquées. Le degré de sélectivité est divisé en 4 niveaux selon les formules suivantes :
   - very high (VH): sélection de 0 à e^k enregistrements ;
   - high (H): sélection de e^k à e^{2k} enregistrements ;
   - average (A): sélection de e^{2k} à e^{3k} enregistrements ;
   - low (L): sélection de e^{3k} à e^{4k} enregistrements.

Où k est défini selon la taille de la collection$|C|$. Nous utiliserons $k = \frac{\ln |C|}{4}$: cette formule nous permet de diviser la charge des requêtes en quatre ensembles en fonction du nombre de lignes sélectionnées.

Le premier ensemble est composé de trois requêtes ayant des sélectivités différentes et effectuant une agrégation (sum(Extendedprice*Discount)) en appliquant des restrictions sur les attributs *Discount*, *Quantity* et la famille de colonnes *FC_DATE*, pour une année dans la première requête, pour un mois dans la deuxième et pour une semaine pour la troisième.

Le deuxième ensemble est composé de trois requêtes qui calculent des agrégats sur le revenu (sum (Revenue)), et trient les résultats en fonction de l'année (Year) et du fabricant (*Brand1*) des structures respectives *FC_DATE* et *FC_PART*. Ces requêtes appliquent des restrictions sur la catégorie et le fabricant des produits (*Category*, *Brand1*) ainsi que la région des fournisseurs (*Region*). Ainsi, elles sollicitent la hiérarchie *Brand1 → Category → Mfgr*, définie dans la famille de colonnes des produits (*FC_PART*).

Le troisième groupe est composé de quatre requêtes qui calculent des agrégats sur le revenu (sum (revenue)) en fonction des attributs appartenant aux structures clients, fournisseurs et date. Ceci en appliquant des restrictions sur la hiérarchie *City → Nation → Region* de *FC_CUSTOMER et FC_SUPPLIER*, et *Day → month → Year* de *FC_DATE*.

Le quatrième ensemble est composé de trois requêtes qui effectuent en fonction des structures produit, client et fournisseur, un calcul d'agrégats sur le profit (sum (Revenue – Supplycost)). Ces requêtes appliquent des restrictions sur la hiérarchie *Brand1 → Category → Mfgr* de *FC_PART* et *City → Nation → Region* de *FC_CUSTOMER* et *FC_SUPPLIER.*

Tableau 13 Filtres de requêtes

Filtres/restriction	**Nombre de lignes gardées**	**Classe de sélectivité**
store.c_city='Midway', date.d_year='2012'	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

Notons que le jeu de requêtes est traduit dans le langage de MongoDB et le langage Hive (pour HBase).

Dans ce qui suit nous décrivons comment utiliser le banc d'essai SSB +.

5.8 COMMANDES DBGEN

La nouvelle génération des données a été enrichie. Nous décrivons ici les principales fonctionnalités. L'argument T permet de choisir le fichier à générer. Nous listons ci-dessous les valeurs possibles:

- **c:** générer les données Customer seulement (1 fichier),
- **p:** générer les données Part seulement (1 fichier),
- **s:** générer les données Supplier seulement (1 fichier),
- **l:** générer les données Lineorder seulement (1 fichier),
- **d**: générer les données Date seulement (1 fichier),
- **a**: générer les données pour tous les fichiers (5 files),
- **g**: générer le fichier Global (1 fichier).

Exemple. Pour générer tous les fichiers avec un facteur d'échelle *sf=1* dans une approche normalisée, nous devons utiliser la commande suivante.

- `dbgen -s 1 -T a`

Pour générer un fichier global avec un facteur d'échelle *sf=1*, nous devons suivre la commande :

- `dbgen -s 1 -T g`

Nous introduisons un nouvel argument *F,* permettant de spécifier le format de sortie et qui peut prendre les valeurs suivantes :

- **j**: génère les données dans un fichier de format JSON;
- **x**: génère les données dans un fichier de format XML ;
- **c**: génère les données dans un fichier de format CSV.

Les données générées doivent être chargées dans les bases de données. Cette phase de chargement est assuré par l'outil *DBLoad*.

5.9 DBLOAD : OUTIL DE CHARGEMENT DE DONNEES

A l'origine, le SSB génère seulement un script SQL pour le chargement de données dans les bases de données relationnelles. Dans le nouveau benchmark, *DBLoad* joue le rôle d'un outil ETL (Extracting, Transforming and Loading) réduit à la simple fonction de chargement. DBLoad supporte trois configurations de chargement, chacune correspondant à un modèle spécifique :

- Chargement du fichier Global JSON dans MongoDB,
- Chargement du fichier Global XML dans MongoDB ;
- Chargement le fichier Global CSV dans HBASE ou MongoDB.

En guise d'exemple voici le script de chargement du fichier Global CSV dans la base de données *HBase*. Le script est écrit avec le langage *Hive*.

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
'Dimporttsv.separator=;'
-Dimporttsv.columns=HBASE_ROW_KEY,
Customer:C_NAME
Customer:C_ADDRESS ,
Customer:C_CITY ,
Customer:C_NATION,
Customer:C_REGION,
Customer:C_PHONE ,
Customer:C_MKTSEGMENT,
Supplier:S_NAME ,
Supplier:S_ADDRESS,
Supplier:S_CITY,
Supplier:S_NATION,
Supplier:S_REGION,
Supplier:S_PHONE,
Part:P_NAME,
Part:P_MFGR,
Part:P_CATEGORY,
Part:P_BRAND1,
Part:P_COLOR,
Part:P_TYPE,
Part:P_SIZE,
Part:P_CONTAINER,
Date:D_ORDERDATE,
Date:D_DATE,
Date:D_DAYOFWEEK,
Date:D_MONTH,
Date:D_YEAR ,
Date:D_YEARMONTHNUM,
Date:D_YEARMONTH ,
Date:D_DAYNUMINWEEK,
Date:D_DAYNUMINMONTH,
Date:D_DAYNUMINYEAR ,
Date:D_MONTHNUMINYEAR,
Date:D_WEEKNUMINYEAR ,
Date:D_SELLINGSEASON,
Date:D_LASTDAYINWEEKFL,
Date:D_LASTDAYINMONTHFL,
Date:D_HOLIDAYFL ,
Date:D_WEEKDAYFL,
Lineorder:ORDERPOPRIORITY,
Lineorder:SHIPPRIORITY ,
Lineorder:QUANTITY,
Lineorder:EXTENDEDPRICE,
Lineorder:TAX,
Lineorder:REVENUE,
Lineorder:DISCOUNT,
Lineorder:SUPPLYCOST,
Lineorder:ORDERTOTALPRICE,
HBase_Global , hdfs:/Global.CSV
```

5.10 EXPERIMENTATIONS

L'objectif de cette section n'est pas d'expérimenter les approches de nos contributions du chapitre III et IV.

Dans cette section, nous présentons les résultats des expérimentations menées dans le nouveau benchmark. Nous comparons ces résultats avec sa version d'origine, SSB. Plus précisément, nous présentons les résultats expérimentaux suivant :

- Analyse et comparaison de l'espace mémoire disque ;
- Analyse et comparaison du temps de chargement des données;
- Comparaison du temps de chargement dans les deux systèmes HBase et MongoDB.

Les résultats concernent trois types de configurations :

- Génération des données avec SSB (Données normalisées, CSV) ;
- Génération des données avec SSB+ (Données normalisées, CSV) ;
- Génération des données avec SSB+ (Données dénormalisées, CSV) ;

Pour chaque configuration nous faisons varier l'indicateur d'échelle pour différentes volumétries.

Environnement de test. Pour réaliser ces expérimentations nous avons mis en place un environnement distribué basé sur la plateforme Hadoop 2.6.0 et le système de gestion de données NoSQL orienté colonnes *HBase* 0-98.8 et orienté document *MongoDB*.

HBase est dotée des 4 opérations pour manipuler les données. Pour des requêtes plus avancées, nous avons installé au-dessus de HBase un langage d'interrogation, Hive qui

permet l'application des clauses de sélection et de filtres. Il est également utilisé pour le chargement de données dans HBase.

Hive n'est pas juste un simple langage d'interrogation mais c'est une plateforme distribuée basée sur l'outil Hadoop qui permet le stockage, l'analyse et la restitution des données. Dans son architecture, Hive dispose d'un *metastore* lui permettant de stocker ses bases de données. Il peut être interfacé avec plusieurs autres bases de données telles que la base de données HBase. Pour le chargement de données, dans HBase, Hive crée des tables externes. Ces tables ne sont pas stockées dans le *metastore* et servent uniquement pour le chargement et l'interrogation.

L'architecture distribuée est un cluster composé de trois nœuds. Chaque nœud est une machine Unix (Centos 7) avec 4 Core-i5 et une mémoire RAM de 8Go. Chaque nœud dispose d'un espace de stockage de 2x2To et une connexion réseau de 1Gb/s. Chaque nœud est utilisé pour jouer le rôle de *datanode* et un joue aussi le rôle de *namenode*. Dans la terminologie de HBase, chaque machine est utilisée pour jouer le rôle de RegionServer dont un joue le rôle de *HMaster*. La gestion des ressources est assurée par l'outil Zookepper que nous avons installé en parallèle à HBase et que nous gérons indépendamment de HBase pour une meilleure maitrise des logs et des erreurs techniques.

Pour le système de gestion orienté documents, nous avons installé MongoDB au niveau du cluster. Ce dernier est composé de trois nœuds, chacun est un *datanode* (nœud de calcul et de stockage) dont un qui joue aussi le rôle de *namenode*. Nous avons installé *MongoDB* V3.2 dans chaque nœud. 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*).

Les données sont chargées en utilisant l'outil de MongoDB, *Mongoimport*.

Expérimentation 1 : **Utilisation de la mémoire**.

Premièrement, nous comparons la génération des données dans les deux versions SSB et SSB+. Les résultats sont résumés dans le Tableau 14 et la Figure 40 Espace de stockage utilisé Comme mentionné précédemment, le SSB ne génère pas le volume de données attendu. Il génère entre 0.56 et 0.58 fois volume de données attendu ; par exemple 0.56 GB pour un *sf = 1*, alors qu'un volume de 1GB est attendu. Pour un indicateur d'échelle *sf = 100*, nous obtenons un volume de données égal à 59 GB. Nous avons un ratio entre l'indicateur d'échelle et la taille de données d'environ 0.58.

Tableau 14 Mémoire disque par configuration

Configuration	***sf=1***	***sf=10***	***sf=100***	***sf=1000***
SSB, normalisé	*987M*	*5.6G*	*59G*	*589G*
SSB+, normalisé	*978M*	*9,7G*	*97G*	*976G*
SSB+, denormalisé	*968M*	*9.6G*	*96G*	*967G*

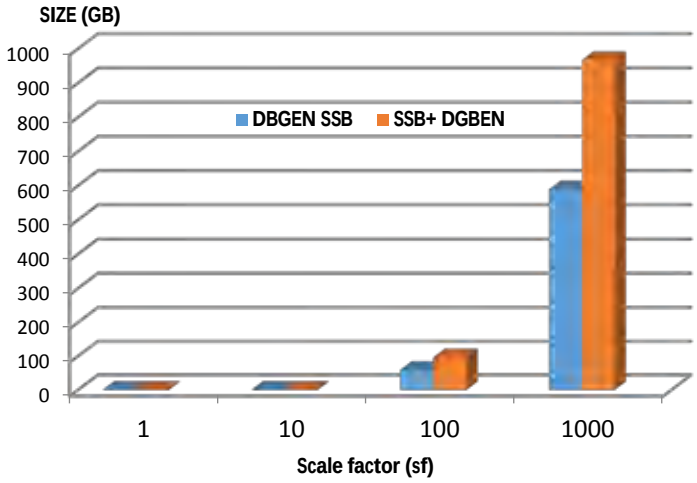


Figure 40 Espace de stockage utilisé

SSB+ prend en considération ce problème. Nous observons qu'il est plus proche du volume des données attendu dans le cas de données normalisées. Il génère 97 Go de données pour un *sf=100* et 976 Go pour un *sf = 1000*. Le ratio est un plus grand que 0.96. Pour résumer le DBGEN du SSB+ améliore d'environ 0.5 la taille de données attendue.

Expérimentation 2 : **le temps d'exécution dans les différentes configurations**

Dans le tableau 15, nous reportons le temps nécessaire pour générer la donnée en utilisant différents indicateurs d'échelle pour les différentes configurations.

Tableau 15 Temps d'exécution par configuration

Configuration	***sf=1***	***sf=10***	***sf=100***	***sf=1000***
SSB, normalized	0.19min (11.42s)	1.38min (90.8s)	23,3min (1383s)	278min (16715s)
SSB+, normalized	0.35min (21.05s)	3,61min (217s)	35.58min (2135s)	47.37min (2864s)
SSB+, denormalized	0.37min (20.82s)	3.47min (208.2s)	34,53min (2072s)	347min (20820s)

Nous observons dans la Figure 41 que le générateur SSB nécessite moins de temps que le générateur du SSB+ pour la génération de données. Ceci s'explique par le fait que l'indicateur d'échelle du SSB+ génère plus de données.

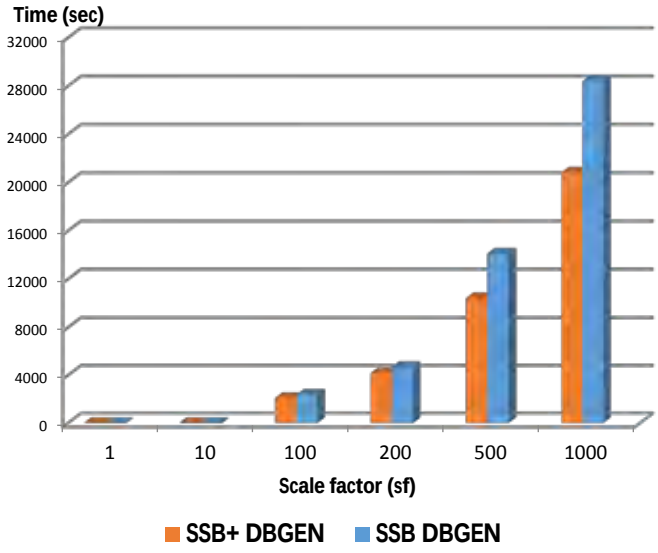


Figure 41 Temps de génération par configuration

Expérimentation 3 : Chargement de données dans *HBase* et *MongoDB*.

Nous utilisons l'outil *DBLoad* pour charger les données générées dans HBase et MongoDB. Nous reportons dans la Figure 42 nos mesures concernant le processus de chargement des données.

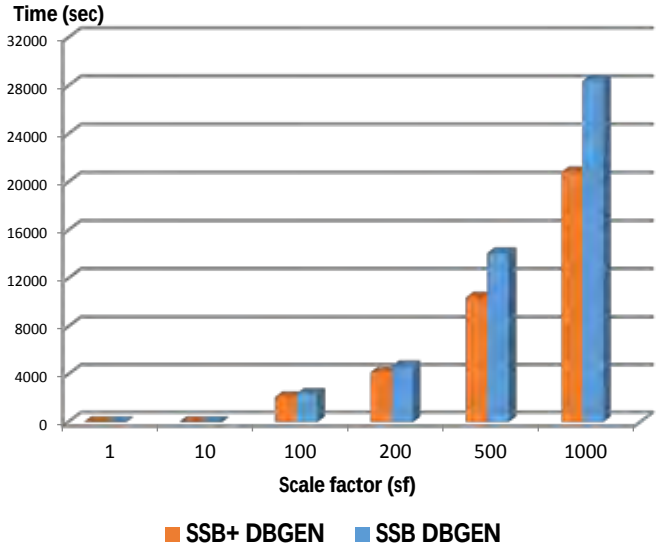


Figure 42 Temps de chargement par configuration

Nous considérons différents indicateurs d'échelle, *sf =1*, *sf =100* et un *sf =1000* et des données générées en mode dénormalisé. Les résultats confirment que le chargement se fait plus facilement dans *HBase*. Les données brutes pour *HBase* sont dans un fichier de format CSV, pendant que les données brutes pour *MongoDB* sont dans un fichier de format JSON. Il est important de noter qu'un format JSON est entre 3 et 3.5 fois plus volumineux que le format csv, différence due au balisage dans le fichier csv. Rappelons aussi que les données sont déjà distribuées dans le système de gestion de fichiers Hadoop et que HBase est interfacée avec ce dernier ce qui accélère le processus de chargement.

5.11 BILAN

Nous avons proposé dans ce chapitre une version étendue du benchmark de référence SSB adaptée aux systèmes distribués, en l'occurrence les bases de données NoSQL orientées colonnes et orientées documents. Les données peuvent être générées dans différents formats (CSV, XML, JSON) et dans différents modèles de données. Ce nouveau benchmark n'est pas restreint à une utilisation dans le modèle relationnel, il peut générer les données dans plusieurs utilisations incluant différents systèmes NoSQL. Les données peuvent être générées dans une architecture distribuées utilisant la plateforme Hadoop pour la distribution. Ce nouveau benchmark enrichi le jeu de requêtes et propose également un script de chargement spécifique à chaque système évalué.

Nos expérimentations montrent que le SSB+ apporte de nombreux avantages par rapport à la version d'origine SSB. Il améliore la précision de l'indicateur d'échelle et facilite la phase de chargement en permettant également le chargement des données dans un environnement distribué en utilisant Hadoop. Nous avons testé *DBload* dans les deux bases de données NoSQL, HBase et MongoDB.

Nous avons fourni un cadre plus simple pour traiter la technologie NoSQL et des technologies de type SQL. Cette proposition a fait l'objet d'une publication dans [Chevalier, Malki, et al. 2015a]

Dans le chapitre suivant, nous utilisons SSB+ comme plateforme d'expérimentations pour valider nos approches proposées dans les chapitres III et IV.

6. CHAPITRE VI : EXPERIMENTATIONS ET VALIDATIONS »

Ce chapitre a pour objectif d'expérimenter notre approche, de valider expérimentalement nos propositions ; les différentes implantations du schéma en étoiles dans les modèles logiques NoSQL orienté colonnes et orienté documents sont expérimentées. Nous évaluons aussi les différentes implantations des treillis d'agrégations.

6.1 INTRODUCTION

Dans le chapitre précèdent nous avons présenté la plateforme de génération des entrepôts de données multidimensionnelles SSB+. Cette dernière permet, en plus d'entrepôts de données ROLAP, de générer des entrepôts de données selon les deux modèle NoSQL orienté colonnes et orienté documents. La plateforme supporte plusieurs implantations logiques. Nous allons nous appuyer sur cette dernière pour montrer la faisabilité de nos propositions pour les deux modèles NoSQL. Il s'agit par la suite d'évaluer les performances relatives au temps de générations des treillis d'agrégats mais aussi le comportement de chacun de ces deux modèles face à un jeu de requêtes utilisateur.

Nous pouvons résumer les expérimentations à mener dans les points suivants :

- l'instanciation d'entrepôts de données dans le modèle orienté documents ;
- Comparaisons des approches d'implantations orienté documents avec les approches d'implantions du modèle relationnel ;
- l'étude des treillis étendus dans le modèle orienté documents ;
- l'instanciation d'entrepôts de données dans le modèle orienté colonnes.

6.2 PROTOCOLE EXPERIMENTAL

Architecture matérielle et logicielle : Nous utilisons deux architectures décrites dans le chapitre précèdent : la première est composée d'une seule machine (Standalone) tandis que la seconde est basée sur une architecture en cluster de trois machines [Edward and Sabharwal 2015].

Sur chaque machine, une instance MongoDB version 3.2 a été installée dans chaque nœud (appelé *shard* dans le langage de MongoDB : un shard est un nœud de stockage et de calcul). Un des trois shards jouera aussi le rôle de config-server, pour gérer la communication avec les autres shards à l'image du namenode.

Nous avons aussi installé sur chacun de ces nœuds, une instance HBase (version 0.98). Chaque nœud joue le rôle de *region-server* (nœud de stockage et de calcul HBase) ; l'un des trois nœuds joue aussi le rôle de serveur de configuration (*HMaster*). Comme décrit dans le chapitre précèdent la gestion des ressources de HBase est assurée par un outil externe appelé *Zookeeper* [Hunt et al. 2010].

Pour pouvoir mener des comparaisons avec le modèle relationnel, nous avons aussi installé une instance PostgresSQL sur un des trois nœuds (cf. Figure 43).

Notons que nous démarrons un serveur à la fois selon l'expérimentation à mener.

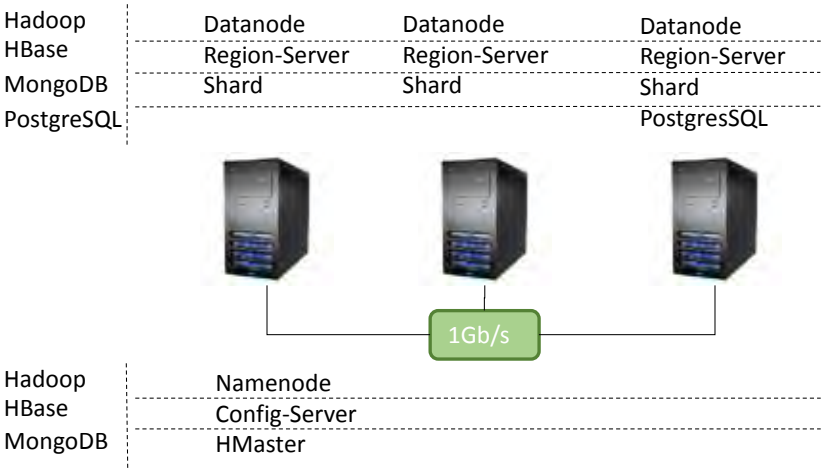


Figure 43 Architecture cluster

Les données : Pour la génération des données, nous utilisons le benchmark SSB+ [Chevalier et al. 2015a].

Les jeux de requêtes. Nous avons deux jeux de requêtes**.** Pour l'interrogation, nous utiliserons un jeu composé de 3 ensembles de requêtes, chacun comporte 3 requêtes. Une quatrième requête avec un niveau de sélectivité diffèrent est utilisée à des fins de comparaisons. Le second jeu de requêtes est dédié à la construction du treillis d'agrégats.

Dans le premier jeu, les requêtes sont fournies par le générateur SSB+. Nous donnons ici les requêtes du premier ensemble.

Requête 1.1.
```
select sum(l_extendedprice*l_discount) as revenue
from lineorder, date
where l_orderdate = d_datekey  and d_year = '1993'  and
l_discount between 1 and 3 and l_quantity < 25;
```

Requête 1.2.
```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and p_category = 'MFGR#12' and
s_region_name = 'AMERICA'   group by d_year, p_brand1
order by d_year, p_brand1;
```

Requête 1.3.
```
select c_nation_name, s_nation_name, d_year,
sum(lo_revnue)
as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey and c_region = 'ASIA' and
s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997
group by c_nation_name, s_nation_name, d_year
order by d_year asc, revenue desc;
```

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. A des fins d'analyses, nous avons créé une requête Q4 similaire à la requête Q1 mais avec moins de sélectivité. 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. Pour les requêtes Q2 et Q3, il n'est pas possible de réduire la sélectivité car les clauses 'WHERE' et 'GROUP BY' réduisent déjà significativement le nombre d'enregistrements retournés.

Le second ensemble contient des requêtes pour la construction du treillis d'agrégats, permettant de faire évaluer graduellement le nombre de dimensions. Nous fournissons ici un exemple d'une requête avec trois dimensions (*Customer, Part* et *Supplier*).

Requête.
```
select c_city, c_nation_name, c_region_name, s_city,
s_nation_name, s_region_name, p_type, p_category,
min(quantity) as min_quantity,
max(quantity) as max_quantity,
sum(quantity) as sum_quantity,
count(quantity) as count_quantity,
avg(quantity)as avg_quantity,
min(totalprice)as min_totalprice,
max(totalprice) as max_totalprice,
sum(totalprice) as sum_totalprice,
count(totalprice)as  count_totalprice,
avg(totalprice) as avg_totalprice,
min(revenue) as min_revenue,
max(revenue)as max_revenue,
sum(revenue)as sum_revenue,
count(revenue)as count_revenue,
avg(revenue) as avg_revenue
from       globalr
where orderdate = datekey into cuboid_CPS

group by c_city, c_nation_name, c_region_name, s_city,
s_nation_name, s_region_name, p_type, p_category;
```

6.3 INSTANCIATION DES ENTREPOTS DE DONNEES ORIENTE DOCUMENTS

Le but des expérimentations qui suivent est de valider l'instanciation d'un entrepôt de données multidimensionnelles avec les approches définies dans le chapitre III. Nous prenons également en compte la conversion d'une implantation logique à une autre au sein du modèle orienté documents, car ces conversions peuvent être nécessaires en fonction des traitements à

privilégier sur les données. Enfin, sur la base de ces modèles, nous générons des treillis d'agrégats et comparons le coût de chaque modèle.

Pour permettre de suivre l'interprétation de nos résultats, nous proposons un rappel des différentes implantations logiques orientés documents à savoir :

- L'implantation plate (*DFL—Document Flat Logic*) : une implantation dénormalisée où le fait et ses dimensions sont confondus dans un même document plat;
- L'implantation imbriquée (*DNL—Document Nested Logic*) : semblable au modèle plat mais plus structuré ; les dimensions et les faits sont stockés dans un même document mais dans lequel nous séparons les mesures du fait des attributs des dimensions (paramètres et attributs faibles) dans des documents imbriqués;
- L'implantation hybride (*DHL—Document Hybrid Logic*) : les concepts fait et dimensions sont stockés dans des documents séparés mais contenus dans la même collection;
- L'implantation éclatée (*DSL—Document Shattered Logic*) : les concepts fait et dimensions sont stockés dans des documents séparés contenus dans des collections distinctes.

6.3.1 Temps de chargement et espace de stockage

Le but de cette expérimentation est d'évaluer le temps de chargement et l'espace disque utilisé par chaque approche d'implantation. Les données sont générées directement (selon les règles de transformations du chapitre III) et dans le format JSON qui est le format le plus optimisé pour MongoDB. Nous faisons évoluer la taille des données, en générant trois volumes de données avec les indicateurs d'échelle *sf =1*, *sf=10* et *sf=25*. Avec l'indicateur d'échelle *sf =1* nous générons approximativement 10^7 documents pour le fichier *LineOrder*, pour l'indicateur d'échelle *sf=10* nous avons environ 10^8 et ainsi de suite. Pour l'implantation éclatée, pour un *sf =1* nous aurons 10^7 documents pour le fichier *LineOrder* et beaucoup moins pour les fichiers de dimensions (voir chapitre V).

Résultats : Le Tableau 16 résume le temps de chargement et l'espace de stockage nécessaire pour chaque implantation pour les trois indicateurs d'échelle. Nous observons que les deux implantations hybride (DHL) et éclatée (DSL) nécessitent moins d'espace de stockage et moins de temps pour le chargement. Par exemple, pour un indicateur d'échelle *sf=1* (10^7 enregistrements), nous avons besoin de 4,2 Go pour l'implantation éclatée et hybride (DHL) tandis qu'il faut un espace de 15 Go pour les deux implantations plate (DFL) et imbriquée (DNL).

Ces premiers résultats attendus s'expliquent par le fait que les deux implantations éclatée et hybride sont moins redondantes. Dans les autres implantations, plate (DFL) et imbriquée (DNL), les données des dimensions sont dupliquées dans chaque document de la collection *LineOrder*. Dans l'implantation éclatée (DSL), les données des dimensions sont stockées dans des collections dédiées : *Customer*, *Supplier*, *Part* et *Date.* Elles contiennent respectivement 5 000 documents, 3 333 documents, 333 333 documents et 2 556 documents.

Tableau 16 Temps de chargement et espace mémoire par implantation

Implantations / Configuration	**Plate** (DFL)	**Imbriquée** (DNL)	**Eclatée** (DSL)	**Hybride** (DHL)
sf=1, une seule machine	1306s/15Go	1235s/15Go	1005s/4.2Go	501s/4.2Go
sf=10, une seule machine	16680s/150Go	16080s/150Go	4320s/42Go	4407s/42Go
sf=25, une seule machine	46704s/375Go	44220s/375Go	10980s/105Go	11020s/105Go
sf=1, cluster	4246s/15Go	4304s/15Go	3767s/4.2Go	3737s/4.2Go

Nous pouvons constater que le temps de chargement est meilleur dans une configuration avec une seule machine que dans la configuration composée du cluster. Par exemple, l'implantation plate nécessite 1306 secondes pour le chargement dans la configuration avec une seule machine contre 4246 secondes pour le chargement des données dans le cluster. Nous pouvons expliquer cela par le transfert de données entre les shards. En effet, MongoDB possède un outil qui équilibre les données (*Balancer*) entre les différents shards. Les données sont déplacées vers les nœuds ayant le moins de données ce qui ralentit le processus de chargement. Nous pensons que ce résultat s'inverserait en augmentant le volume des données.

Notez qu'à des fins de comparaisons significatives entre le modèle NoSQL orienté documents et celui orienté colonnes, nous n'avons pas optimisé les configurations de MongoDB. Par exemple, le temps de chargement aurait pu être amélioré en utilisant la désactivation des index lors du chargement [Huang et al. 2013] [Kaur and Singh n.d.]. Les index sont le point fort de mongoDB mais représente une taille importante puisque la taille de ces dernièrs représente jusqu'à 1/5 de la taille réelle des données, par exemple pour le *sf=10*, la taille réelle est de 150 (DNL) Go et elle est égale à 176 Go après chargement dans MongoDB. En comparaison les systèmes HBase (orienté colonnes) ne génèrent pas d'indexes lors des chargements de données.

6.3.2 Mise à jour des données

Dans le Tableau 17, nous rapportons le temps d'exécution moyens pour 10 mises à jour choisies aléatoirement. Nous observons que le temps d'exécution est généralement significatif pour les modèles éclaté et hybride. Les temps de mises à jour sont intéressants pour l'implantation éclatée lorsqu'il s'agit d'actualiser les attributs des dimensions car nous avons besoin d'actualiser qu'une seule collection de taille relativement moyenne. Dans les autres implantations, plate et imbriquée, le nombre de documents est comparable. La mise à jour est plus rapide lorsqu'il s'agit d'attributs non imbriqués.

Tableau 17 temps d'exécution des mises à jour

1 seule machine, **sf=1**	**M0**	**M1**	**M2**	**M3**
update orderdate	1806s	1850s	93s	97s
update name (customer)	1853s	1893s	0.1s	56s
update name (supplier)	1816s	1824s	0.05s	214s
update brand (part)	1810s	1814s	0.1s	204s
update yearmonthnum (date)	1826s	1893s	0.1s	208s
cluster, sf=1	M0	M1	M2	M3
update orderdate	107s	120s	30s	34s
update name	110s	125s	0.2s	32s
update name (supplier)	110s	112s	0.2s	86s
update brand (part)	108s	111s	0.2s	85s
update yearmonthnum (date)	108s	114s	0.2s	87s

6.3.2.1 Exemple de mise à jour dans MongoDB

La mise à jour des données dans MongoDB n'est pas une tâche difficile. Nous fournissons ici un exemple. Considérons la collection de documents "M0_SF1" pour laquelle nous souhaitons actualiser tous les documents avec le critère "*c_name=Customer#000014993*".

Nous actualisons "*c_name*" avec une nouvelle valeur "NewCustomer#000014993", la requête pour la mise à jour est la suivante :

```
db.M0_SF1.update
({c_name: 'Customer#000014993'},
{$set:{c_name: 'NewCustomer#000014993'}});
```

Dans notre expérimentation, la sélection de la nouvelle valeur lors d'une mise à jour est aléatoire. Le processus est simple puisque nous prenons des valeurs entre *'Customer#000000001'* et *'Customer#x'* avec x la valeur maximale possible.

6.3.3 Interrogations

Nous évaluons pour chaque implantation le temps d'exécution pour le jeu de requêtes que nous avons défini précédemment. Nous montrons le temps d'exécution dans les deux configurations, avec une seule machine et avec un cluster. 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.

Résultats : Dans le Tableau 18 nous rapportons tous les résultats, tandis que dans la , nous ne rapportons que les résultats moyens (Q1 AVG, Q2 AVG, Q3 AVG, Q4 AVG).

Les temps d'exécution sont meilleurs dans les implantations plate et imbriquée, puisque les données y sont dénormalisées (pas de jointures). Comme indiqué précédemment, les requêtes du benchmark ont été générées en SQL. Ces requêtes ont été traduites et surtout optimisées en tirant profit d'une riche palette qu'offre MongoDB : agrégation pipeline,

fonctions MapReduce, requêtes simples et procédures [Parker et al. 2013]. Selon le modèle utilisé, les requêtes ne sont pas simples à écrire. Elles sont assez complexes pour le modèle hybride et le modèle éclaté par exemple. Pour le modèle éclaté, nous avons besoin de matérialiser les données qui ne sont pas forcément stockées dans les mêmes documents. Si nous n'écrivions pas les requêtes efficacement et/ou si les requêtes nécessitent de joindre toutes les données avant l'application des filtres, le temps d'exécution serait impacté négativement. Nous appliquons donc des filtres avant les jointures des données du fait et celles des dimensions et nous utilisons les agrégations pipeline, les fonctions mapreduce, les requêtes et les procédures.

Les variantes des requêtes Q1, Q2 et Q3 ont des temps d'interrogations comparables dans les implantations hybride et éclatée. En revanche, nous pouvons voir que dans l'implantation plate et imbriquée, la requête Q4 retourne des temps d'exécutions plus importants, cela est dû à un niveau de sélectivité plus faible et par conséquent un nombre de documents plus important à retourner (500000 documents environ pour un *sf=1*— ce chiffre ne ressort pas dans les tableaux des résultats mais il est possible de l'obtenir grâce à l'utilisation du plan d'exécution (*explain*()). Ces temps sont significativement importants dans les implantations nécessitant des jointures : les implantations éclatée et hybride.

Matérialisation totale des données : de nombreux systèmes NoSQL déconseillent l'utilisation de jointures et/ou ne les supportent pas nativement. Nous avons mené ces expérimentations pour évaluer le coup de la normalisation des données dans ces architectures distribuées par rapport à des implantations qui dénormalisent les données (l'équivalent de l'implantation plate). Dans les meilleurs des cas, nous obtenons un temps d'exécution de 16.33 heures pour l'implantation éclatée et 3.24 heures pour l'implantation hybride pour un *sf=1*. Un temps d'exécution qui n'est pas acceptable surtout si toutes les requêtes nécessitent une matérialisation totale c'est-à-dire combinant toutes les données. Dans ce cas de figure, il est préférable d'abandonner les modèles avec des jointures totales. Il est toujours possible de filtrer les données au préalable puis jouer avec les données matérialisées au niveau de la RAM pour réduit le temps d'exécutions mais les limitations restent importantes.

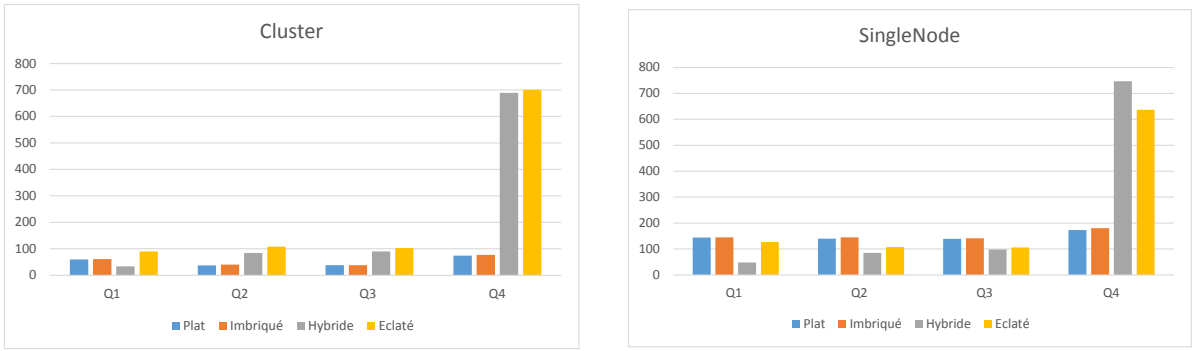


Figure 44 Temps d'exécutions moyen par implantation

Tableau 18 Temps d'exécution par requête et par implantation

sf=1 single node	Plat	Imbriqué	Hybride	Eclaté
Q1.1	150s	152s	50s	129s
Q1.2	141s	142s	47s	125s
Q1.3	141s	141s	47s	127s
Q1 avg	**144s**	**145s**	**48s✓**	**127s**
Q2.1	140s	140s	85s	107s
Q2.2	140s	142s	84s	103s
Q2.3	140s	138s	86s	111s
Q2 avg	**140s**	**145s**	**85s✓**	**107s**
Q3.1	137s	138s	97s	105s
Q3.2	140s	143s	99s	107s
Q3.3	142s	143s	98s	108s
Q3 avg	139s	141s	98s	106s
Q4	173s ✓	180s	747s	637s
sf=1 cluster	Plat	Imbriqué	Hybride	Eclaté
Q1.1	62s	62s	37s	94s
Q1.2	59s	61s	33s	91s
Q1.3	58s	58s	33s	86s
Q1 avg	**60s**	**61s**	**34s✓**	**90s**
Q2.1	36s	39s	85s	105s
Q2.2	37s	41s	83s	109s
Q2.3	37s	40s	83s	109s
Q2 avg	**37s✓**	**40s**	**84s**	**108s**
Q3.1	36s	36s	89s	100s
Q3.2	40s	40s	89s	104s
Q3.3	38s	38s	92s	104s
Q3 avg	**38s✓**	**38s**	**90s**	**103s**
Q4	**74s✓**	**77s**	**689s**	**701s**
✓ : meilleur temps				

6.3.3.1 Exemple d'agrégation pipeline dans MongoDB

Nous avons mentionné que nous utilisions les agrégations pipeline et le paradigme MapReduce de Mongodb. Nous allons détailler brièvement ces outils.

MongoDB recommande l'utilisation des agrégations pipeline lorsqu'il s'agit de requêtes complexes. Les données sont modélisées selon le processus des pipelines, les documents entre dans un processus (pipeline) de plusieurs phases qui transforme les documents en un résultat agrégé.

Une requête de ce type s'écrit de la façon suivante :

```
db.M0_SF1.aggregate([
            {$match: {s_region_name: 'ASIA', c_region_name: 'ASIA',
            d_year: {$gt: '1992', $lt: '1997'}}},
            {$group: {_id: {c_nation_name: "$c_nation_name",
            s_nation_name: "$s_nation_name", d_year: "$d_year"},
            revenue: {$sum: "$l_revenue"} } },
```

La requête suivante est son équivalent en SQL:

```
select c_nation_name, s_nation_name, d_year, sum(l_revenue)
from M0_SF1
where s_region_name='ASIA' and s_region_name='ASIA'
group by c_nation_name, s_nation_name
order by d_year ASC, l_revenue DESC;
```

MongoDB permet aussi l'utilisation du paradigme *MapReduce*. Les données et le calcul sont distribués en combinant les deux fonctions *Map* er *Reduce*. La fonction *Map* permet de filter et trier les données tandis que la fonction *Reduce* se charge de les agréger.

Dans ce qui suit, nous fournissons un exemple de fonction MapReduce dans le langage javascript pour MongoDB.

```
var mapFunction1 = function() {
      emit(this.c_custkey, this.l_revenue);
      };
var reduceFunction1 = function(keyCustId, revenues) {
      return Array.sum(revenues);
      };

db.orders.mapReduce(
      mapFunction1,
      reduceFunction1,
{ out: "sum_revenue_example" }
      )
```

Les agrégations pipeline constituent une bonne alternative au paradigme MapReduce car souvent la complexité du MapReduce est notable.

6.3.4 Calcul du treillis

Comme vu dans les chapitres précédents, dans les applications OLAP, il est courant de stocker durablement les résultats des requêtes souvent utilisées. Le résultat de chaque requête est stocké dans un nœud (appelé aussi cuboïde ou agrégat) et l'ensemble des cuboïdes constitue le cube OLAP. Dans le premier modèle, que nous appellerons cuboïde classique par la suite, le cuboïde est constitué d'un ensemble de mesures et d'un paramètre au maximum par dimension.

Dans cette section, nous menons plusieurs expérimentations classées en deux parties :

- **Calcul du cuboïde classique**. Évaluer le temps nécessaire pour le calcul des cuboïdes dans les différents modèles proposés ;
- **Calcul des cuboïdes étendus.** Comparer le cuboïde classique avec deux nouveaux types de cuboïdes que nous baptisons *cubes étendus imbriqué* et *détaillé* dont nous décrivons la construction un peu plus loin dans cette section.

6.3.4.1 Calcul du cuboïde classique

Premièrement, les données sont agrégées via les dimensions *supplier*, *part*, *date* et *customer* (cf. Figure 45). Quatre niveaux d'agrégats sont calculés : toutes les combinaisons des quatre dimensions ; 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 appliquées sont : *Max*, *Min*, *Sum* et *Count*.

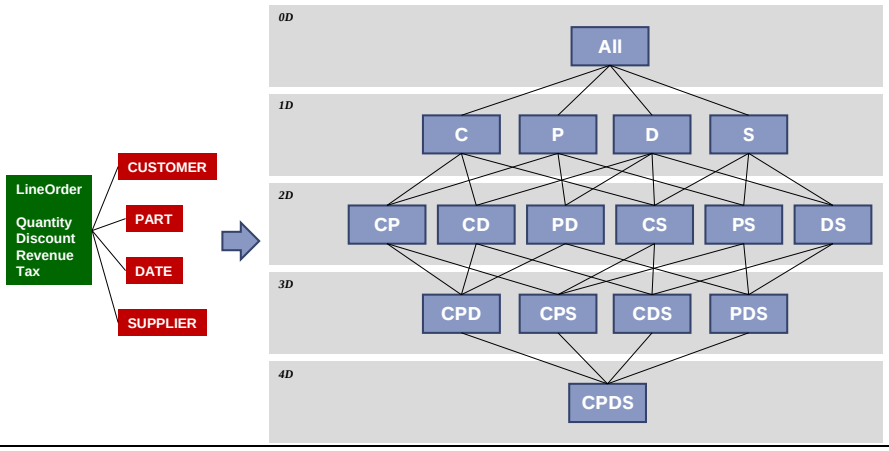


Figure 45 Treillis d'agrégats

L'expérimentation est réalisée dans les deux configurations comme vues précédemment, avec un seul nœud puis dans le cluster composé de trois nœuds.

Résultats. Dans la Figure 46, nous rapportons le temps d'exécution du treillis. Le niveau supérieur correspond à CSPD (les données détaillées *: Customer, Supplier, Part, Date*). Sur le second niveau, nous conservons toutes les combinaisons de trois dimensions, puis de deux et ainsi de suite. Pour chaque nœud nous montrons le temps d'exécutions en seconde pour un *sf=1*.

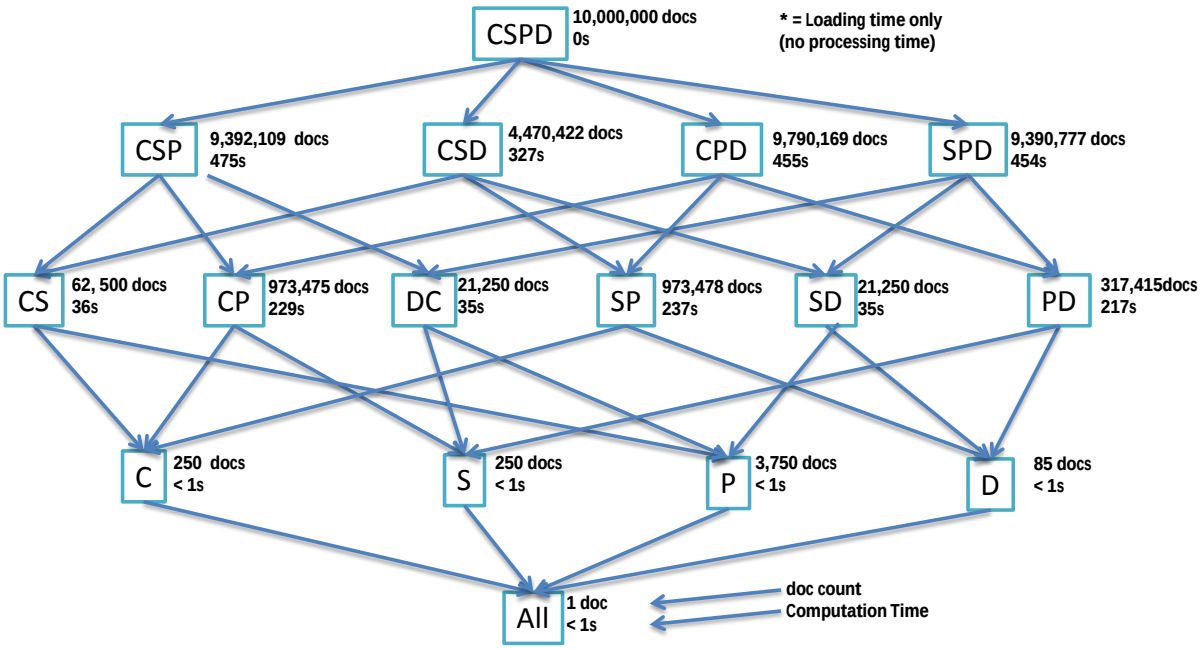


Figure 46 Treillis d'agrégats avec le temps de calcul (en secondes) et la taille (en enregistrement/ documents). Le nom des dimensions est abrégé (*D : Date, P : Part, S : Supplier, C : Customer*)

Dans le Tableau 19, nous rapportons le temps d'exécution nécessaire pour le calcul d'un cuboïde OLAP de *x* dimensions (*x* variant de 1 à 4) et pour chaque implantation logique étudiée. En général nous pouvons observer que les deux modèles éclaté (DSL) et hybride (DHL) sont moins coûteux puisque les données sont regroupées uniquement sur les clés de références. En revanche, si d'autres attributs de dimensions sont nécessaires, le temps d'exécution augmentera significativement.

Tableau 19 Temps d'exécution moyen par cuboïde

	Plat	**Imbriqué**	**Hybride**	**Eclaté**
3 dimensions	423s	460s	303s	308s
2 dimensions	271s	292s	157s	244s
1 dimension	196s	201s	37s	44s
0 dimension (all)	185s	191s	37s	27s

6.3.4.2 Calcul des cuboïdes étendus

Dans cette partie d'expérimentations, nous comparons le cuboïde classique avec les cuboïdes étendus. Il est donc question de trois types de cuboïdes : classique, détaillé et imbriqué. Les deux derniers sont construits pour bénéficier des spécificités techniques du modèle orienté documents. Ils utilisent également la notion d'ordre partiel.

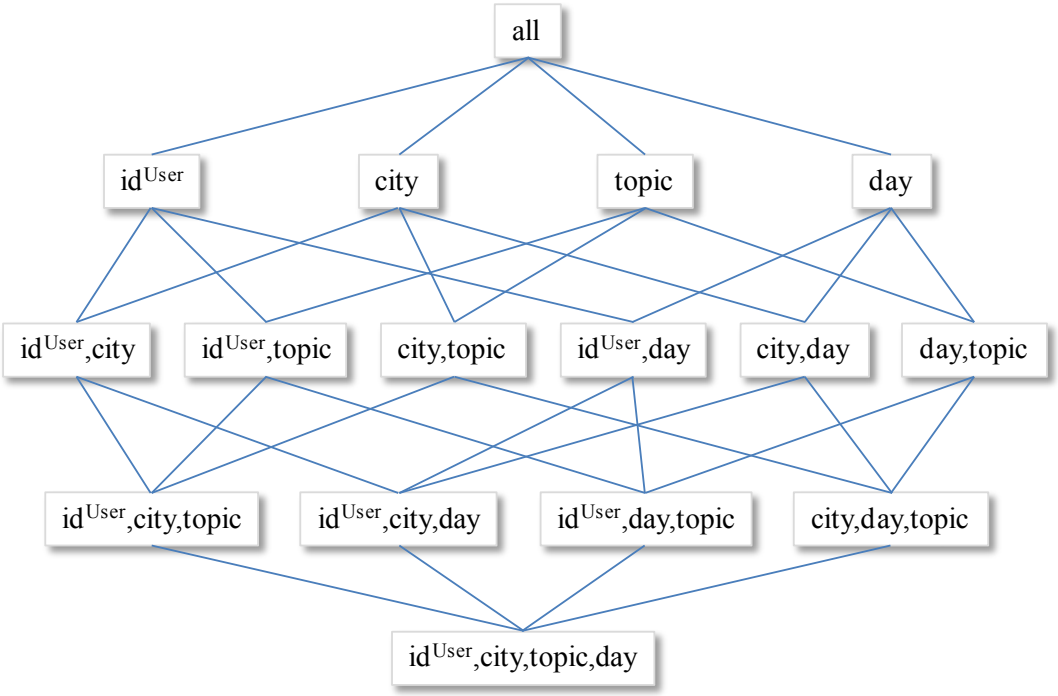


Figure 47 Exemple de cube OLAP par treillis de cuboïdes (ou pré-agrégats)

Nous détaillons dans un premier temps la construction de ces deux nouveaux cuboïdes.

Le cuboïde 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). Cela est possible en utilisant l'imbrication : le modèle imbriqué stocke la valeur agrégée et surtout imbrique, dans un tableau, toutes les mesures agrégées des agrégats du niveau inférieur (s'il existe) qui ont permis l'obtention de cet agrégat.

Exemple. Considérant l'exemple décrit dans la Figure 47 Exemple de cube OLAP par treillis de cuboïdes (ou pré-agrégats). Considérons un nœud faisant intervenir des niveaux hiérarchiques : les paramètres *Month* et *Category* Le document stockant l'agrégat de la combinaison de valeurs *Month=March* et *Category=Sport* est le suivant :

```
{ id: 1,
  Category: "Sport",
  Month: "March",
  sum_Retweet-C: 27,
  by_Topic:   [ { Topic: "Football",  sum_Retweet-C: 15},
                { Topic: "Basketball",sum_Retweet-C: 12}
              ],
  By_Day:     [ { Day: "03/25/2016",  sum_Retweet-C: 15},
                { Day: "03/23/2016",  sum_Retweet-C: 12},
                { Day: "03/22/2016",  sum_Retweet-C: 0}
              ]
}
```

Nous pouvons observer dans cet exemple que le cuboïde imbriqué groupe les données sur la combinaison des attributs *Category* et *Month*, dont la combinaison de valeurs est : (*Sport*,

March). Puis le modèle imbrique dans un tableau les données agrégées pour les niveaux hiérarchiques directement inférieurs aux paramètres *Category* et *Month*, soit les paramètres *Topic*, pour la hiérarchie H_{Sub} de la dimension *Subject* et *Day*, pour la hiérarchie H_{Time} de la dimension *Time*.

Le cuboïde détaillé

Le cube OLAP détaillé est une deuxième extension du cube OLAP classique qui, comme la précédente adaptation, ne peut être facilement implantée dans un système relationnel dû à l'absence de principe d'imbrication natif. Dans cette extension, le cuboïde contient à la foi, la valeur agrégée pour les données du niveau courant et un tableau imbriqué contenant tous les détails du calcul (les valeurs du niveau le plus fin).

Exemple. Considérons le même treillis dans la Figure 47. Considérons le même nœud que précédemment (non représenté dans la figure) faisant intervenir des nivaeux hiérarchiques. *Month* et *Category*. Le document stockant l'agrégat de la combinaison de valeurs *Month=March* et *Category=Politics* sera le suivant :

```
{ id: 1,
  Category: "Politics",
  Month: "March",
  sum_Retweet-C: 56,
  details: [ {id: 2, Retweet-C: 22},
             {id: 3, Retweet-C: 24},
             {id: 6, Retweet-C: 10}
           ]
}
```

Dans le tableau au sein de l'attribut composé *details*, les valeurs présentent concernent les retweets de catégorie « Politics », c'est-à-dire les retweets du Topic « Elections », et ces retweets datant de mars (« March »). Les identifiants au sein du tableau sont les identifiants de chaque valeur du fait.

Noter que, dans cet exemple, le modèle agrège les données groupées selon les paramètres *Category*, *Month* et All^{User} des dimensions *Subject*, *Time* et *User*. Toute dimension non présente est en fait « remplacée » par son paramètre englobant *All*.

Expérience. Calcul des treillis étendus

Dans ces expérimentations nous comparons le temps de calcul de chacun des trois treillis avec une comparaison de la mémoire de stockage requise. Pour ce calcul du treillis, quatre niveaux d'agrégats sont calculés à partir du niveau de base fourni par le générateur du banc d'essai SSB+. Ces agrégats sont :

- toutes les combinaisons de trois dimensions (*CSP, CSD, CPD, SPD*),
- toutes les combinaisons de deux dimensions (*CS, CP, CD, SP, SD, PD*),
- toutes celles d'une dimension (*C, S, P, D*) et enfin
- toutes les données (*All*).

A chaque niveau, les fonctions d'agrégation *Max*, *Min*, *Sum* et *Count* sont utilisées pour agréger chaque mesure.

L'expérimentation est menée sur un volume de données de *sf=10* (*150 Go*).

Tableau 20 Temps d'exécution et mémoire utilisés par dimensions pour chaque cuboïde

Nb. de dimensions Type de Cuboïde	exemple de cuboïde OLAP	Temps de calcul	Mémoire requise
3D, classique	*c(supplier,date,part,T)*	428s	4065 Mo
2D, classique	*c(supplier,date,T)*	197s	86Mo
1D, classique	*c(date,T)*	1s	0.3Mo
Moyenne		207s	1198Mo
3D, imbriqué	*c(supplier,date,part, T,[customer,T])*	476s	4950Mo
2D, imbriqué	*c(supplier,date,T[part,T])*	227s	1897Mo
1D, imbriqué	*c(date,T,[supplier,T])*	10s	512Mo
Moyenne		189s	2373Mo
3D, détaillé	*c(supplier,date,part, T,[M])*	521s	5917Mo
2D, détaillé	*c(supplier,date,T, [M])*	251s	2600Mo
1D, détaillé	*c(date,T,[M])*	59s	2110Mo
Moyenne		273s	3407Mo
1D : une dimension, 2D : deux dimensions, 3D : trois dimensions			

Résultats. Dans le Tableau 20, nous rapportons les résultats de cette expérimentation. Nous comparons le temps de calcul et l'espace disque nécessaire pour chacun des trois cuboïdes étudiés. 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.

```
{ 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}
           ]
}
```

Le même constat peut être fait pour le temps de calcul du treillis : les deux cuboïdes étendus nécessitent également plus de temps de calcul. Le calcul de chaque agrégat nécessite une requête supplémentaire pour extraire aussi les données détaillées à imbriquer.

L'objectif des cuboïdes n'est pas de gagner en temps de calcul du treillis mais plutôt de répondre efficacement à des requêtes de type drill down (forage). Nous montrons ce point dans les deux expérimentations suivantes.

Expérience. Comportement des treillis face à un jeu de requêtes

La seconde partie des expérimentations se focalise sur l'évaluation du temps d'interrogation et particulièrement le coût de requêtes nécessitant une navigation vers les données des niveaux inférieurs.

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

Dans le premier ensemble, 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.

$QS_1 = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}, \ldots Q_{40}\}$ 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.

Dans le second ensemble, nous constituons un jeu de 40 requêtes, sur le même principe que l'ensemble QS_1. Autrement dit, les requêtes sont les mêmes mais nous modifions l'enchainement et le nombre de requêtes de type drill down. Cette modification vise à ne pas pénaliser un des deux cuboïdes. Dans ce deuxième ensemble QS, nous appliquons quatre requêtes de l'ensemble, puis une requête de type drill down tel que.

- $QS = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}, \ldots Q_{40}\}$ est l'ensemble des requêtes utilisées et
- $QD = \{Q_5, Q_{10}, Q_{15}, Q_{20}, Q_{25}, \ldots\}$, est l'ensemble des requêtes avec le niveau de granularité le plus fin.
- $Queries = \{Q_1, Q_2, Q_3, Q_4, Q_6, Q_7, Q_8, Q_9, Q_{11}, Q_{12} \ldots\}$ est l'ensemble de requêtes OLAP classiques.

L'expérimentation est menée sur un volume de données *sf=10,* soit *150 Go.*

Résultats. les résultats de l'expérimentation sont observables dans les Figure 48 et Figure 49. Dans ces deux figures nous montrons le temps d'exécution en fonction de nos requêtes de test.

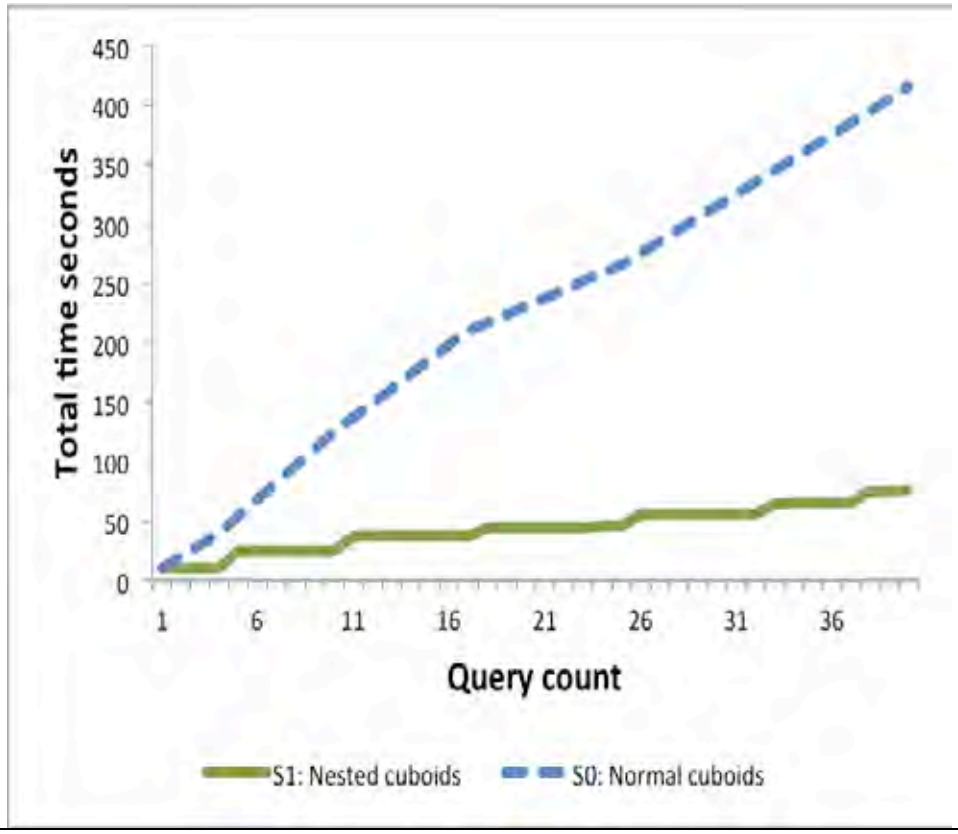


Figure 48 Comparaison des temps d'exécution des cuboïdes classique et imbriqué

Dans la Figure 48, nous comparons le comportement des deux types de cuboïdes classique et imbriqué 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.

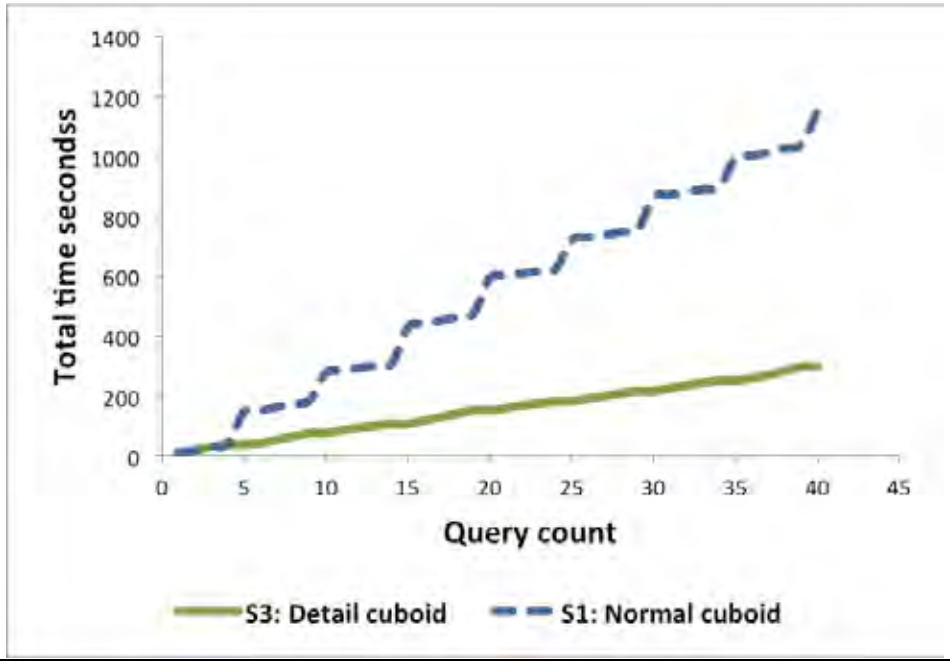


Figure 49 Comparaison des temps de réponse des cuboïdes classique et détaillé

Dans la Figure 49, 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 (une requête drill down après 4 requêtes classiques).

Discussion. 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 les données détaillés.

Ces deux cuboïdes, imbriqué et detaillé, permettent aux décideurs d'avoir une vision plus détaillée sur les données du calcul sans devoir consulter les niveaux inférieurs. En revanche, en terme 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 cela reste une option intéressante.

De ce constant, 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 pré-calculés afin de supporter des analyses OLAP faisant intervenir des requêtes de forages.

6.3.5 Conversion intra-modèles

Il s'agit dans cette expérimentation d'évaluer le temps d'exécution pour les processus intra-modèles orientés documents, soit la transformation des données d'une implantation vers une autre au sein du modèle NoSQL orienté documents. Dans les implantations logiques proposées, la hiérarchie des attributs n'est pas préservée et selon l'implantation plate où les

attributs des mesures et des dimensions sont confondues, il est nécessaire de tenir des métadonnées pour permettre la conversion entre implantations.

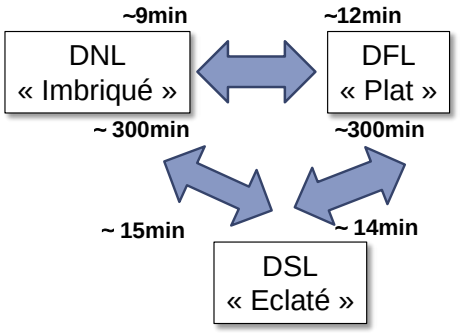


Figure 50 temps de conversion intra-modèle orienté colonnes avec SF1

Résultats. Dans la Figure 50, nous montrons le temps des conversions intra-modèles dans MongoDB. La transformation des données de l'implantation plate (DFL) vers l'implantation imbriquée et inversement, sont comparables. Pour effectuer la transformation de l'implantation plate vers l'implantation imbriquée (DNL), nous regroupons les attributs dans plusieurs sous-documents, c'est-à-dire nous regroupons les attributs représentants les mesures dans un sous-document et ceux représentants les attributs de dimensions dans un sous-document à raison d'un sous-document par dimension. Dans la transformation inverse, on regroupe les attributs provenant d'un document composé de plusieurs sous-documents dans un seul document plat.

La conversion est plus compliquée lorsqu'il s'agit de transformer les données de l'implantation plate vers l'implantation éclatée (DSL). Pour cela, nous avons besoin de distribuer les données dans 5 collections (*Date, Part, Supplier, Customer* et *Lineorder*) : nous avons besoin de 5 projections sur les données brutes puis une sélection des attributs des dimensions. Les conversions depuis le modèle éclaté nécessitent des temps très important dû au fait que MongoDB ne supporte pas nativement les jointures malgré une première tentative d'intégration dans la dernière version de Mongodb.

Remarque. La derniere version de MongoDB (MongoDB-3.2) supporte les jointures.

Notons que par manque de temps, nous n'avons pas évalué les transformations depuis et vers l'implantation hybride ainsi que les processus inter-modèles. Ces expérimentations sont prévues dans nos perspectives.

6.3.6 Discussion

Dans cette section nous avons évalué l'instanciation des entrepôts de données orientées documents selon quatre approches logiques. Les deux implantations plate et imbriquée offrent des résultats similaires et ont l'avantage de regrouper au sein d'une même collection aussi bien les données du fait que celles des dimensions. Cela permet d'éviter des jointures, mais a pour conséquence une importante redondance des 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 une distribution 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.

Les deux implantations hybride et éclatée offrent des performances moins bonnes que les deux implantations précédentes (éclatée et imbriquée), la redondance des données est évitée au détriment du temps de réponses aux requêtes. Par contre nous nous attendions à de meilleures performances pour l'implantation hybride dont les jointures se font au niveau de la même collection. Nous pensons que ces résultats peuvent être améliorés si on optimise, au niveau physique, le processus de distribution des collections sur les différents nœuds du cluster.

La seconde partie d'expérimentations concerne les processus de transformations entre les implantations logiques (excepté l'approche hybride suite à un manque de temps). Nous confirmons que les jointures engendrent un coût important à l'image des transformations depuis l'implantation éclatée. Transformer les données depuis une implantation éclatée nécessite un temps d'exécution peu acceptable tandis que dans le sens inverse nous avons obtenu des temps bien meilleurs.

Dans la section suivante nous comparons ces implantations orientées documents avec les implantations du modèle relationnel notamment l'approche ROLAP.

6.4 COMPARAISON ENTRE MODELE RELATIONNEL ET MODELE ORIENTE DOCUMENTS

Dans la section précédente, nous avons montré la faisabilité de nos approches dans le modèle orienté documents. Nous avons instancié quatre systèmes d'entrepôts de données, orientés documents. Nous avons expérimenté la construction du treillis d'agrégats.

Nous menons des comparaisons entre le modèle relationnel et le modèle orienté documents. Nous considérons quatre implantations logiques pour transposer le schéma multidimensionnel en étoile. Pour une comparaison significative entre le modèle relationnel et le modèle NoSQL orienté documents, nous considérons deux implantations pour chaque modèle. Pour le modèle orienté documents, nous utilisons les deux modèles plat et éclaté décrit dans le chapitre III. Pour le modèle relationnel, nous utilisons le modèle logique standard que nous appelons modèles RFL (*relationnal flat*) et le modèle dénormalisé appelé RSH (*relationnal shattered*). Nous avons fait le choix de ces deux implantations car le modèle relationnel ne supporte pas l'imbrication.

L'implantation normalisée est composé de cinq relations :

Schéma Supplier :(S_SUPPKEY, S_NAME, S_ADDRESS, S_CITY, S_NATION, S_NATION, S_REGION, S_PHONE) ;

Schéma Customer : (CUSTOMER, C_CUSTKEY, C_NAME, C_ADDRESS, C_CITY C_NATION, C_REGION, C_PHONE, C_MKTSEGMENT);

Schéma Date : (D_DATEKEY, D_DATE, D_DAYOFWEEK, D_MONTH, D_YEAR, D_YEARMONTHNUM, D_YEARMONTH, D_DAYNUMINWEEK, D_DAYNUMINMONTH, D_DAYNUMINYEAR, D_MONTHNUMINYEAR, D_WEEKNUMINYEAR, D_SELLINGSEASON, D_LASTDAYINWEEKFL, D_LASTDAYINMONTHFL, D_HOLIDAYFL, D_WEEKDAYFL);

Schéma Lineorder : (LO_ORDERKEY, LO_LINENUMBER, LO_CUSTKEY#, LO_PARTKEY#, LO_SUPPKEY#, LO_ORDERDATE#, LO_ORDERPRIORITY, LO_SHIPPRIORITY, LO_QUANTITY, LO_EXTENDEDPRICE, LO_ORDTOTALPRICE, LO_DISCOUNT, LO_REVENUE, LO_SUPPLYCOST, LO_TAX, LO_COMMITDATE# LO_SHIPMODE) ;

Schéma Part : (P_PARTKEY, P_NAME, P_MFGR, P_CATEGORY, P_BRAND1 P_COLOR, P_TYPE, P_SIZE, P_CONTAINER).

L'implantation relationnelle dénormalisée est composée d'une seule relation regroupant tous les attributs des cinq relations ci-dessus.

Les comparaisons que nous réalisons sont les suivantes :

- volume requis pour chaque implantation ;
- espace de stockage ;
- temps d'exécution du jeu de requêtes décrit dans l'environnement d'expérimentation du chapitre V;
- le temps nécessaire pour la construction du treillis.

Nous suivons la même logique d'expérimentation que précédemment et utilisons deux configurations matérielles : une seule machine (*singlenode*) et un cluster de 3 shards. Pour le système relationnel nous utiliserons la solution open source PostgresSQL 8.1 installée sur un des nœuds. Pour ne pas parasiter, lors des expérimentations dans PostgresSQL, nous arrêtons les instances MongoDB et inversement.

6.4.1 Modèle orienté document : Comparaison avec le modèle relationnel

Dans cette expérimentation nous évaluons deux points :

- l'espace de stockage utilisé pour chaque modèle et pour indicateurs d'échelle, *sf=1*, *sf=10* et *sf=25*.
- Le comportement de chaque modèle face aux jeux de requêtes que nous avons décrit dans l'expérimentation précédente.

Nous utilisons MongoDB de la plateforme d'expérimentation pour le stockage des deux implantations DFL et DSL.

Chaque modèle est testé avec trois ensembles de requêtes OLAP que nous avons définis.

Résultats : Dans la Figure 51, nous montrons l'espace de stockage nécessaire pour chaque approche pour les trois volumes de données générés (*sf=1, sf=10, sf=25*). Nous pouvons constater que PostgresSQL nécessite moins d'espace de stockage par rapport à MongoDB (entre 3 et 5 fois moins). Cela peut s'expliquer par le fait que dans le modèle orienté documents et spécialement dans MongoDB le type des données est aussi stocké

explicitement. Pour stocker les données selon une implantation plate orientée documents, nous avons besoin de quatre fois plus d'espace dû à la redondance. Par exemple, pour un *sf=10* (10^8 documents) nous avons besoin respectivement de 150 Go et 42 Go pour DFL (documents plats) et DSL (implantation éclatée orienté documents) et de 45 Go et de 12 Go pour le RFL (relationnel plat) et RSH (relationnel dénormalisé).

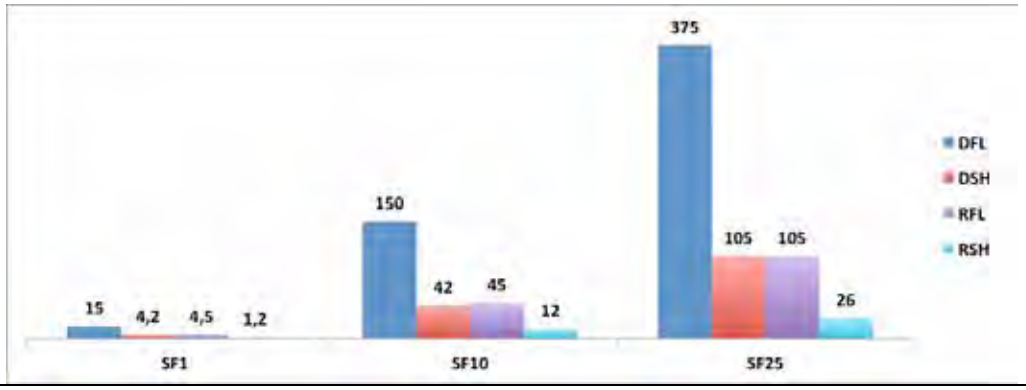
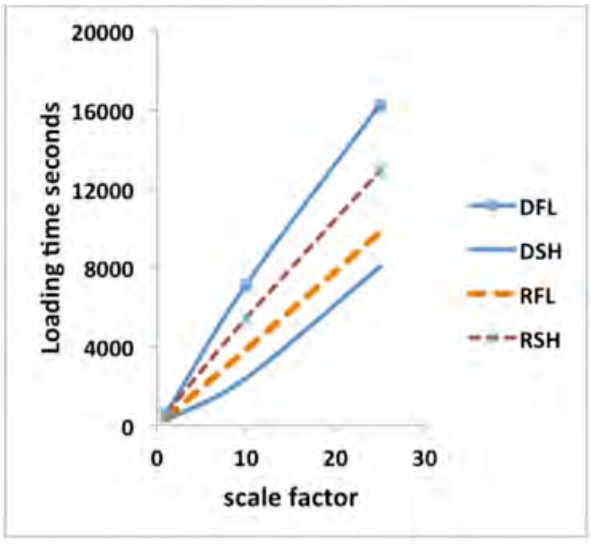


Figure 51 Espace de stockage par implantation et par indicateur d'échelle entre le modèle orienté documents et le modèle relationnel

Dans la Figure 52, nous rapportons le temps de chargement par modèle. Le modèle denormalisé est chargé environ 23% à 26% plus vite avec PostgresSQL (implantation RFL) qu'avec MongoDB (DFL). En revanche, avec l'approche normalisée les données sont chargées environ 35-40% plus vite dans MongoDB (document éclaté) que dans PostgresSQL (relationnel plat). Cette dernière observation est expliquée par le fait que dans PostgresSQL, la contrainte des clés étrangères ralentit significativement le processus de chargement. Nous pouvons observer que MongoDB à un rythme environ de 18MB/s à 21MB/s tandis que PosgresSQL charge environ 3MB/s à 8MB/s.



x

Figure 52 Temps de chargement par modèle

Dans le Tableau 21, nous rapportons le temps d'exécution pour le jeu de requêtes. Un premier constat confirme le résultat obtenu auparavant, le modèle DFL est plus performant que le modèle DSL (document éclaté) dû au faible support des jointures dans MongoDB. Les requêtes tournent de 15 à 22 fois plus vite dans PostgresSQL avec le modèle RSH (relationnel dénormalisé) que dans MongoDB avec le modèle DSL (document éclaté). PostgresSQL est

donc performant avec ce jeu de requêtes puisque ces requêtes sont particulièrement sélectives : les données à traiter peuvent tenir dans la RAM après filtrage.

Avec MongoDB, nous observons que les temps d'exécution sont généralement meilleurs dans les configurations distribuées. Pour de nombreuses requêtes, les temps d'exécutions sont améliorés jusqu'à 2 à 3 fois selon les cas. Si l'architecture distribuée est pénalisée par le transfert réseau, elle est favorisée par la distribution du calcul.

6.4.2 Construction du treillis d'agrégats

Dans cette expérimentation, nous considérons les requêtes OLAP correspondant au calcul des cuboïdes OLAP. Ces requêtes sont plus expressives que celles utilisées précédemment. Nous évaluons ici des combinaisons de trois dimensions.

Tableau 21 Temps d'exécution par modèle

Configuration	**Mongo, singlenode**		**PostgreSQL, singlenode**		**Mongo, cluster**	
Requêtes	**DFL**	**DSH**	**RFL**	**RSH**	**DFL**	**DSH**
QS1.1	1317s	1403s	*720s*	*143s*	*625s*	*400s*
QS1.2	1448s	1587s	*577s*	*60s*	*585s*	*392s*
QS1.3	1001s	1315s	*364s*	*60s*	*2091s*	*388s*
QS1 avg	*1255s*	*1433s*	*553s*	*88s*	*1100s*	*393s*
QS2.1	1284s	1384s	*399s*	*67s*	*364s*	*922s*
QS2.2	1002s	1202s	*397s*	*70s*	*373s*	*884s*
QS2.3	1411s	1611s	*443s*	*60s*	*361s*	*885s*
QS2 avg	*1232s*	*1399s*	*413s*	*66s*	*361s*	*897s*
QS3.1	1438s	1645s	*738s*	*61s*	*363s*	*942s*
QS3.2	1442s	1701s	*740s*	*61s*	*404s*	*955s*
QS3.3	1127s	1821s	*657s*	*61s*	*374s*	*963s*
QS3 avg	*1335s*	*1722s*	*711s*	*61s*	*380s*	*953s*
avg	*1274s*	*1518s*	*559s*	*71s*	*610s*	*747s*

Résultats. Dans les résultats du Tableau 22, 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 singlenode; les requêtes sont moins rapides sur PostgresSQL, cela est expliqué par la selectivité de ces requêtes ; le nombre de requêtes retournées est important dépassant la RAM du système relationnel PostgresSQL. Nous pouvons aussi noter une amélioration du temps d'exécution dans une configuration distribuée. 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 êtres toutes chargées dans la RAM ni dans MongoDB ni dans PostgresSQL mais MongoDB semble mieux supporter cette charge que PostgresSQL.

6.4.3 Discussion

Dans cette section nous avons mené une comparaison entre le modèle orienté documents (MongoDB) et le modèle relationnel (PostgresSQL). 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 ROLAP (éclaté relationnel) est moins coûteuse en terme 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 dépassent 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 ROLAP pour un volume de données raisonnables. L'implantation orienté documents doit être motivée par un gros volume de données.

Tableau 22 Temps d'exécution de cuboïdes à trois dimensions

Cuboïde	**DFL, singlenode**	**RSH, singlenode**	**DFL cluster**
c_city, s_city, p_brand	7466s	9897s	6480s
c_city, s_city, d_date	3540s	6742s	2701s
c_city, p_brand, d_date	4624s	9302s	3358s
s_city, p_brand, d_date	4133s	8509s	3301s

Nous pouvons conclure que MongoDB offre de meilleures performances en termes d'exécution lorsque le volume de données à traiter augmente significativement et lors de l'utilisation d'une architecture distribuée. A l'instar de MongoDB, PostgresSQL enregistre de nettes performances lorsqu'il s'agit de traiter des données entrant en RAM.

Dans la section, nous évaluons nos approches dans le modèle orienté colonnes.

6.5 INSTANCIATION DES ENTREPOTS DE DONNEES ORIENTE COLONNES

L'objectif des expérimentations suivantes est de montrer la faisabilité des approches que nous avons définies pour le modèle orienté colonnes avec trois implantations logiques : plate, imbriquée et éclatée. Dans un premier temps, nous évaluons le coût d'implantation pour chacune des trois approches (plate, imbriquée et éclatée), en second lieu, nous nous

intéressons au calcul du treillis, et en dernière étape, nous évaluons le temps de conversion d'une approche à l'autre.

Ces expérimentations sont menées avec les deux configurations (cf. chapitre V), un seul nœud puis un cluster de trois nœuds. Les données sont générées avec le banc d'essai SSB+ et sont chargées en utilisant Hive installé au-dessus de HBase (cf. chapitre V).

Comme pour le modèle orienté documents, nous rappelons les différentes implantations que nous expérimentons ici, à savoir :

- l'implantation plate (*CFL—Column Flat Logic*) : les paramètres du fait et des dimensions sont confondus dans une table composée d'une seule famille de colonnes ;
- l'implantation imbriquée (*CNL—Column Nested Logic*) : plusieurs familles de colonnes sont utilisées au sein d'une même table pour les différents concepts du schéma multidimensionnel ;
- l'implantation éclatée (*CSL—Column Split Logic*) : inspirée du modèle relationnel où plusieurs tables contenant chacune une seule famille de colonnes sont utilisées.

6.5.1 Chargement des données

Dans cette expérimentation, nous comparons le temps chargement pour chaque implantation. Nous commençons par un indicateur d'échelle de *sf=1* soit 10^7 lignes (3,9 Go pour les deux implantations plate et imbriquée et environ 1 Go pour l'implantation éclatée). Ensuite nous augmentons le volume et générons des données avec les indicateurs d'échelle *sf=10* et *sf=100* (cf. chapitre V). Les fichiers sont de format CSV (supporté par HBase). La génération est faite d'une manière distribuée dans Hadoop sur lequel repose HBase.

Résultats. Dans le Tableau 23, nous résumons le temps de chargement par modèle et par indicateur d'échelle. Nous pouvons observer que l'implantation éclatée requière moins d'espace de stockage car les données ne sont pas redondantes comme c'est le cas pour les deux implantations plate et imbriquée.

Tableau 23 Temps de chargement de données par implantation

	plat	imbriqué	éclaté
sf=1 (10^7 lignes)	380s / 3.9Go	402s / 3.9Go	264s / 0.997Go
sf=10 (10^8 lignes)	3458s / 39Go	3562s / 39Go	2765s / 9.97Go
sf=100 (10^9 lignes)	39075s / 390Go	39716s / 390Go	33097s / 99.7Go

6.5.2 Calcul du treillis d'agrégats

Dans cette expérimentation, nous évaluons le temps nécessaires pour la construction du treillis pour chacune des trois implantations orientées colonnes.

Résultats. Les résultats sont présentés dans la Figure 53. Le niveau supérieur correspond à CSPD (les données détaillées : *Customer, Supplier, Part, Date*). Sur le second niveau, nous conservons toutes les combinaisons de trois dimensions, puis de deux et ainsi de suite. Pour chaque nœud, nous montrons le nombre d'enregistrements qu'il contient ainsi que le temps nécessaire au calcul (en secondes).

Nous observons comme attendu que le nombre de lignes est proportionnel au nombre de dimensions, il diminue avec la diminution du nombre de dimensions. La même analyse est à noter concernant le temps du calcul (le temps de calcul est plus important quand le nombre de lignes est plus important). Nous avons besoin entre 740 et 642 secondes pour calculer les cuboïdes du premier niveau (avec trois dimensions). Pour le niveau en dessous impliquant deux dimensions, nous avons besoin entre 78 et 481 secondes, et nous avons un temps de calcul variant entre 1 et 23 secondes pour les cuboïdes avec 0 et 1 dimension.

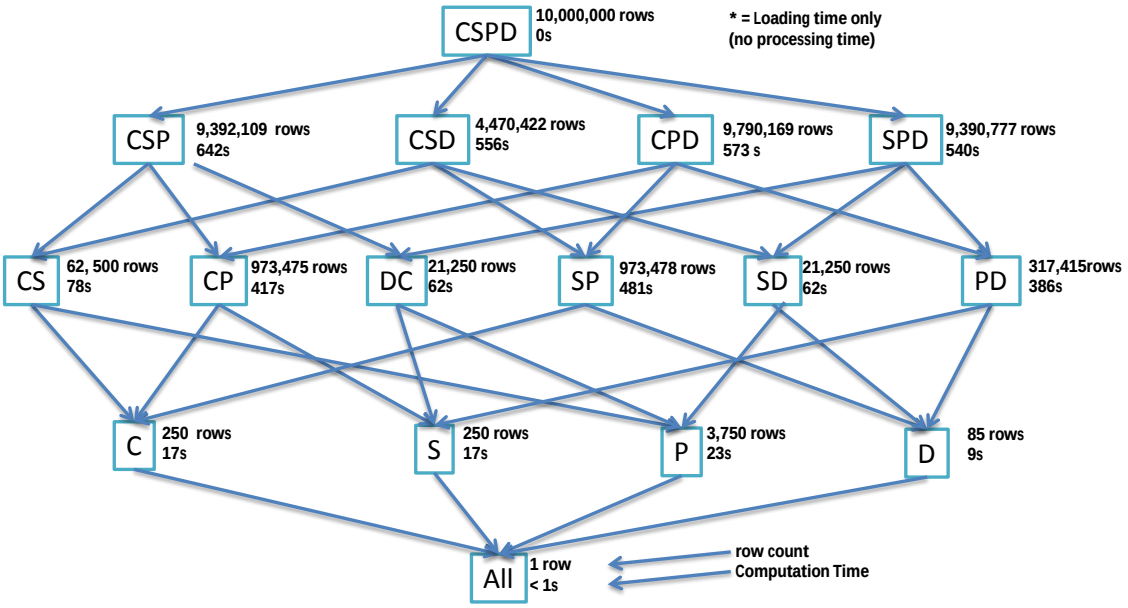


Figure 53 Temps de calcul et nombre de lignes pour chaque cuboïde (les lettres correspondant au nom de chaque de dimension : C=Customer, S=Supplier, D=Date, P=Part/Product)

Le calcul du treillis avec le modèle imbriqué donne des résultats similaires au treillis avec le modèle plat. En revanche avec l'approche éclatée, nous avons des temps d'exécution relativement élevés. Ces résultats sont dûs au mécanisme de jointure. Les résultats peuvent être améliorés si les requêtes de chargement sont écrites directement dans MapReduce, c'est-à-dire sans devoir traduire les requêtes d'insertion de Hive en langage MapReduce. Si Hive offre une syntaxe proche du SQL il impacte le temps d'exécution par cette phase de traduction dans le paradigme MapReduce. Le temps de traduction devient négligeable avec un volume relativement important. En effet, Hive utilise une interface relationnelle [Capriolo 2012] ce qui nécessite la reconstitution des enregistrements pour pouvoir effectuer des traitements. Ensuite, il adopte la stratégie verticale pour le calcul des agrégats.

Tableau 24 Temps d'exécution par cuboïde et par modèle

		Implantations logiques orientées colonnes		
		imbriquée	**éclatée**	**plate**
Cuboïdes	CSD (c_city, s_city, d_date)	556s	4892s	564s
	CSP (*c_city, s_city, p_brand*)	642s	5487s	664s
	CPD (*c_city, p_brand, d_date*)	573s	4992s	576s
	SPD (*s_city, p_brand, d_date*)	540s	4471s	561s
	Dimensions: C = Customer, S = Supplier, D = Date, P = Part (i.e. Product)			

La différence entre les deux modèles dénormalisant les données et le modèle éclaté impacte uniquement le calcul du premier niveau (le calcul des cuboïdes des autres niveaux se faisant depuis les cuboïdes du premier niveau, les mêmes pour les trois modèles). Nous reportons dans le Tableau 24, le temps nécessaire pour chaque cuboïde du premier niveau (trois dimensions) pour chacune des trois implantations de cuboïdes.

6.5.3 Conversion intra-modèles

Dans cette expérimentation il est question d'évaluer le temps d'exécution pour les processus de conversions intra-modèles orienté colonnes. Notons que dans les implantations logiques proposées, la hiérarchie des données n'est pas préservée cela implique la mise en place d'un fichier de métadonnées pour l'implantation plate où les attributs des mesures et des dimensions sont confondues pour faciliter les conversions.

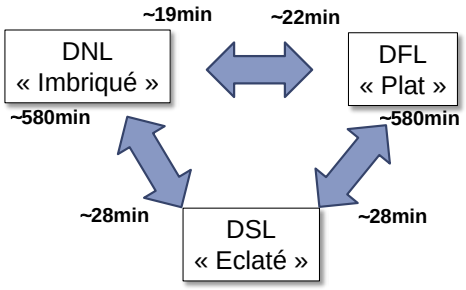


Figure 54 Temps de conversion intra-modèle orienté colonnes (*sf=1*)

Résultats. Dans la Figure 54, nous présentons les résultats des processus de conversions intra-modèles dans HBase. Nous relevons que :

- la transformation des données de l'implantation plate vers l'implantation imbriquée et inversement, sont comparables et présentent les temps de conversion les moins élevés ;
- La conversion de l'implantation plate vers l'implantation imbriquée offre un temps de conversion plus élevé, dû à l'opération de distribution des données dans les structurées imbriquées ;

- La conversion depuis l'implantation éclatée est très coûteuse à cause de l'utilisation des jointures.

6.5.4 Discussion

Dans cette section nous avons évalué les différentes implantations logiques dans le modèle orienté colonnes mais également les transformations des données entre ces différentes implantations.

Concernant les approches d'implantation logiques, l'implantation plate et l'implantation imbriquée offrent des réponses très proches en termes de stockage mais aussi en termes de temps de réponses à des requêtes analytiques. En terme d'espace de stockage, les résultats obtenus nous apparaissent logiques puisque le volume est pratiquement le même. En revanche, pour le temps de réponse à des requêtes analytiques, nous nous attendions à de meilleures performances pour le modèle imbriqué, or les résultats sont aussi très proches. La séparation des attributs dans plusieurs familles de colonnes devrait diminuer le temps de réponses. Nous pensons que cela est dû à la volumétrie manipulée et au nombre de colonnes contenues dans chaque famille de colonnes ; le fait que le nombre d'attributs ne soit pas très grand (55 attributs constituant mesures et paramètres de dimensions), le temps mis pour balayer les 55 attributs qui sont contenues dans une famille de colonnes (pour l'implantation imbriquée) et le temps mis pour accéder aux attributs de chaque famille de colonnes (pour l'implantation imbriquée) n'est pas différents ce qui produit des résultats très proche.

La troisième approche (l'implantation éclatée) nécessitant des jointures offre des performances moins intéressantes que les deux premières (plate et imbriquée). Le coût des jointures dans ces architectures distribuées impacte significativement les performances. L'utilisation d'une telle implantation doit être faite conjointement avec un contrôle de la distribution des données dans les différents nœuds de stockage.

Le troisième objectif était d'évaluer le coût des transformations des données d'une implantation logique vers une autre implantation au sein du modèle orienté colonnes. Le coût est important lorsqu'il s'agit de transformer les données d'une implantation éclatée vers les deux autres implantations plate et imbriquée. Dans le sens inverse, transformer les données imbriquées ou plate vers l'implantation éclatée est moins coûteux. Ces résultats confirment l'idée que matérialiser les données dans une architecture distribuée engendre des coûts parfois peu raisonnables. Dans le cas d'une implantation éclatée il serait plus raisonnable de reconsidérer le niveau conceptuel.

6.6 BILAN

Dans ce chapitre nous avons présenté les différentes expérimentations que nous avons menées pour valider nos propositions en nous basant sur l'environnement de test présenté lors du chapitre précèdent. En premier lieu, une série d'expérimentations s'est axée sur l'instanciation des entrepôts de données dans le modèle orienté documents. Nous avons instancié un entrepôt de données pour chaque implantation et avons évalué le calcul et le temps d'exécution face à un jeu de requêtes d'analyses.

En second lieu, nous avons mené une comparaison entre les approches du modèle orienté documents et le modèle relationnel. Nous avons considéré deux approches équivalentes dans chaque modèle, les approches plate et éclatée pour le modèle documents et les approches normalisées et dénormalisée pour le modèle relationnel. Les comparaisons sont allées de la génération à la construction du treillis d'agrégats. Le modèle relationnel est

efficace quand il s'agit de données supportées par la taille de la RAM tandis que la modèle orienté document s'avère efficace dès lors qu'il s'agit de requêtes retournant un nombre de documents important.

Ensuite, nous avons également évalué et validé l'instanciation d'entrepôts de données dans le modèle orienté colonnes. Dans cette partie nous avons instancié trois entrepôts de données, avec les implantations plate et imbriquée et nous avons évalué la génération de chacun de ces entrepôts de données. Nous avons considéré le calcul des treillis d'agrégats comme nous avons aussi évalué le processus de transformation d'une implantation vers une autre.

En dernier lieu, nous avons évalué les cuboïdes étendus qui ont fait l'objet d'une publication dans [Chevalier et al. 2016a]. Nous avons validé et comparé l'implantation des deux cuboïdes étendus imbriqué et détaillé avec le cuboïde classique. Malgré un coût de stockage coûteux, les deux cuboïdes donnent de bonnes performances selon le cas d'utilisation. Le principal inconvénient du cuboïdes imbriqué et detaillé par rapport au cuboïde classique est donc un volume accru du fait que les documents stockent des données supplémentaires. Toutefois, cet inconvénient est compensé par plusieurs avantages :

- 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.

CHAPITRE VII : CONCLUSION ET PERSPECTIVES

7.1 CONCLUSION GENERALE

Les travaux de recherche présentés dans ce mémoire s'inscrivent dans le cadre des systèmes d'aide à la décision. Ces systèmes reposent sur un processus d'analyse en ligne (OLAP) facilitant l'analyse interactive et la synthèse des données. Les systèmes d'aide à la décision reposent généralement sur une modélisation multidimensionnelle des données pour faciliter les analyses et sont étudiés selon trois niveaux d'abstraction. Au niveau conceptuel, les données sont modélisées selon un schéma multidimensionnel en étoile ou en constellation. Jusqu'ici ce schéma est transposée dans un modèle logique reposant sur le model relationnel, on parle de l'approche ROLAP. L'avènement du Big Data a souligné la difficulté du modèle relationnel à traiter de gros volumes de données et a permis le développement de nouvelles technologies reposant sur une architecture décentralisée et extensible, il s'agit des systémes NoSQL.

Notre premier objectif a été de proposer de nouvelles approches pour implanter le modèle conceptuel multidimensionnel en constellation dans les modèles NoSQL. Dans cette étude, nous avons retenu les deux modèles orienté documents et orienté colonnes qui sont une spécialisation du modèle clé-valeur, et qui présentent deux approches de stockage orthogonales.

Processus d'implantations. Nous avons proposé quatre processus de traduction pour chaque modèle logique :

- le processus de traduction plate qui permet un stockage totalement dénormalisé évitant toute jointure entre les données des faits et des dimensions. Les données sont redondantes et de structure homogène.
- le processus de traduction par imbrication qui permet une structuration des mesures et des paramètres. Les données sont redondantes et de structure homogène concernant l'organisation.
- le processus de traduction hybride normalise les données des faits et les dimensions liées et, mais nécessite l'usage d'auto-jointures. Les données sont non redondantes mais de structures hétérogènes.

- le processus de traduction éclaté normalise également les données des faits et des dimensions. Il nécessite l'utilisation de jointures entre les structures homogènes sans redondance engendrées.

Nous avons étendu les processus de traduction NoSQL aux cubes OLAP. Les cubes OLAP sont une technique très développée dans les entrepôts de données ROLAP afin d'optimiser ces derniers en pré-calculant les requêtes les plus fréquentes ou coûteuses. Ainsi nous avons défini des processus de traduction complète ou partielle d'un l'ensemble de cuboïdes constituant un cube OLAP, soit dans le modèle orienté documents, soit dans le modèle orienté colonnes. Nous avons également introduit des cubes étendus en orientés documents, *treillis détaillé* et *treuillé imbriqué* pour mieux supporter les requêtes OLAP de forages.

Les modèles NoSQL se caractérisent par une forte dépendance de données au traitement : l'efficacité d'une structure de données dépendant du type de requêtes appliquées (contrairement à la modélisation relationnelle dont les formes normales sont définies indépendamment des requêtes). Ainsi, suivant le type de requêtes, le choix de structuration peut être différent. Il est même parfois nécessaire de concevoir un entrepôt de données combinant deux implantations logiques ou plus, chacune favorisant un sous-ensemble de requêtes (approche multistore). Afin de répondre à cette évolution de traitement, nous avons défini des processus de conversions.

Processus de conversions. Nous avons défini des processus de conversion au niveau logique. Nous avons présenté deux types de conversions :

- les conversions intra-modèle en orienté documents et en orienté colonnes. Elles consistent à convertir les données (documents ou lignes) entre les différentes structures d'implantation (plate, imbriquée, hybride et éclatée). Ces conversions restent au sein d'un même modèle logique NoSQL.

- les conversions inter-modèles de l'orienté documents vers l'orienté colonnes et inversement. Ces conversions consistent à migrer les données entre deux modèles logiques NoSQL différents, et ceci en fonction des quatre structures d'implantation plate, imbriquée, hybride et éclatée.

Dans notre contexte d'entrepôts de données multidimensionnelles, nous avons également définis des règles inter-modèles additionnelles, permettant de convertir les cubes OLAP, du modèle orienté documents vers celui orienté colonnes, et inversement.

Validation. Afin de valider nos contributions nos avons développé un benchmark SSB+ [Chevalier, Malki, et al. 2015a] dérivé du benchmark SSB. SBB+ permet de générer des entrepôts de données NoSQL orientés colonnes et orientés documents. Il nous a permis de montrer la faisabilité de nos approches mais aussi de mener une étude comparative entre les différentes implantations d'un côté et entre les deux modèles colonnes et documents de l'autre. Nous l'avons aussi utilisé pour la construction des treillis d'agrégats et l'évaluation des processus de traduction au niveau des modèles logiques.

7.2 PERSPECTIVES

A **court terme,** nous envisageons de poursuivre les expérimentations des processus de conversions notamment les conversions inter-modèles. Nous envisageons également d'étudier l'implantation des entrepôts de données massives dans les modèles orienté graphes notamment dans le système Neo4J.

Nous envisageons également d'étudier la dépendance entre le volume des données et le nombre de nœuds. En théorie les performances en terme de calcul sont étroitement liés à la puissance de calcul : plus le nombre de nœuds est important plus la puissance de calcul accroit, En revanche dans une architecture distribuée où le transfert de données est un paramètre influençant les performances, l'utilisation d'un nombre important de nœuds doit être fonction de la volumétrie des données à traiter. Notre objectif est de déterminer le seuil à partir duquel l'ajout de nœuds ne sera pas significatif.

Par ailleurs, les différentes implantations que nous avons proposées se limitent à des hiérarchies strictes, or dans le monde réel il est aussi question des hiérarchies non strictes et des hiérarchies manquantes. Les solutions actuelles reposent sur des séparations des données strictes dans une table et les non strictes dans une autre table ou bien, sur l'intégration de niveaux de calcul intermédiaires [Pedersen and Jensen 1999] [Malinowski and Zimányi 2006] [Hachicha and Darmont 2013] avec un impact sur la sémantique des données. L'imbrication de données est une des solutions pour pallier les erreurs engendrées par ce type de hiérarchies. Une première contribution a fait l'objet d'une publication [Chevalier et al. 2016a], nous souhaitons mener plus d'investigations pour améliorer la solution proposée.

D'autres perspectives à **plus long terme** sont envisageables.

Nous avons proposé des processus de transformations entre les différentes implantations logiques étudiées. Ces travaux s'inscrivent dans le cadre des entrepôts de données multi-stores. Nous envisageons de proposer une plateforme complète combinant plusieurs outils de stockages. Ce travail est très complexe puisqu'il interpelle plusieurs niveaux :

- Le langage d'interrogation uniforme qui permet de communiquer avec les différents moteurs de stockage. L'objectif est de bénéficier des avantages de MapReduce. Actuellement peu de travaux s'inscrivent dans ce cadre.
- Une répartition des données du schéma sur plusieurs outils de stockage. Un travail aussi complexe puisqu'il nécessite une connaissance de la localité des attributs : en fonction des données. la requête est automatiquement divisée en sous-requêtes. Les différents résultats sont agrégés dans un résultat final avant d'être retournés au décideur.

Cette plateforme offrirait de nombreux avantages. Elle permettrait de tester plusieurs implantations mais aussi d'évoluer vers une autre implantation. Une interface d'interrogation uniforme s'implifiant l'acées aux données.

Implantations logiques basée sur la localité des données. Nous avons jusqu'ici proposé des implantations qui bénéficient des caractéristiques des bases de données NoSQL et les architectures distribuées. La distribution des données est gérée par le système de stockage, or il est possible de contrôler la distribution des données de sorte à piloter la localisation des données et ainsi réduire les transferts des données entre les nœuds. Des travaux sont à souligner dans ce contexte mais ne considèrent pas les entrepôts de données.

Pour finir, il serait également pertinent de s'intéresser à une démarche ETL permettant d'automatiser le plus possible l'entreposage des données ouvertes tabulaires dans le contexte du Big Bata [Berro et al. 2013] [Megdiche Bousarsar 2015] en élargissent l'entreposage à de l'analyse textuelle [Tournier 2007].

Références

A

Daniel Abadi, Samuel Madden, and Miguel Ferreira. 2006. Integrating compression and execution in column-oriented database systems. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data. ACM, 671-682.

D. J. Abadi, D. S. Myers, D. J. DeWitt and S. R. Madden, "Materialization Strategies in a Column-Oriented DBMS," 2007 IEEE 23rd International Conference on Data Engineering, Istanbul, 2007, pp. 466-475.

Daniel J. Abadi, Samuel R. Madden, and Nabil Hachem. 2008. Column-stores vs. row-stores: how different are they really? In Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 967-980.

Daniel J. Abadi. 2012. Consistency tradeoffs in modern distributed database system design. Comput.-IEEE Comput. Mag. 45, 2 (2012), 37.

Fatma Abdelhédi, Amal Ait Brahim, Faten Atigui, and Gilles Zurfluh. 2016. Processus de transformation MDA d'un schéma conceptuel de données en un schéma logique NoSQL. In Actes du XXXIVème Congrès INFORSID, 15–30.

Alberto Abelló, Jaume Ferrarons, and Oscar Romero. 2011. Building Cubes with MapReduce. In Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP. DOLAP '11. ACM, 17–24.

Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. 2011. Big Data and Cloud Computing: Current State and Future Opportunities. In Proceedings of the 14th International Conference on Extending Database Technology. EDBT/ICDT '11. ACM, 530–533.

Swati Ahirrao and Rajesh Ingle. 2015. Scalable transactions in cloud data stores. J. Cloud Comput. 4, 1 (December 2015). DOI:https://doi.org/10.1186/s13677-015-0047-3

J. Chris Anderson, Jan Lehnardt, and Noah Slater. 2010. CouchDB: the definitive guide, O'Reilly Media, Inc.

Rodrigo Aniceto et al. 2015. Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency, Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency. Int. J. Genomics Int. J. Genomics.

E Annoni, F Ravat, O Teste, G Zurfluh 2006 Towards multidimensional requirement design International Conference on Data Warehousing and Knowledge Discovery, 75-84.

Faten Atigui 2013. Approche dirigée par les modèles pour l'implantation et la réduction d'entrepôts de données. Thèse de doctorat, Université Toulouse 1 Capitole.

Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme. 2009. A runtime approach to model-independent schema and data translation. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 275–286.

B

Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. 2012. Uniform Access to Non-relational Database Systems: The SOS Platform. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, & Stanislaw Wrycza, eds. Advanced Information Systems Engineering. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 160–174.

Kyle Banker. 2011. MongoDB in action, Manning Publications Co.

Ladjel Bellatreche, Alfredo Cuzzocrea, and Il-Yeol Song. 2015. Advances in Data Warehousing and OLAP in the Big Data Era. Inf Syst 53, C (October 2015), 39–40.

Alain Berro, Imen Megdiche, and Olivier Teste. 2013. Vers l'intégration multidimensionnelle d'Open Data dans les entrepôts de données. In EDA. 95–104.

Sandro Bimonte and François Pinet. 2012. Conception des entrepôts de données: de l'implémentation à la restitution. J. Decis. Syst. 21, 1–2.

Carlyna Bondiombouy and Patrick Valduriez. 2016. Query Processing in Multistore Systems: an overview. INRIA Sophia Antipolis-Méditerranée.

Dhruba Borthakur. 2008. HDFS architecture guide. HADOOP APACHE Proj. Httphadoop Apache Orgcommondocscurrenthdfs.

Barbara Brynko. 2012. Nuodb: Reinventing the database. Inf. Today 29, 9 (2012), 9–9.

C

G Cabanac, M Chevalier, F Ravat, O Teste 2007. An annotation management system for multidimensional databases International Conference on Data Warehousing and Knowledge Discovery, 89-98,

Yu Cao et al. 2011. ES2: A Cloud Data Storage System for Supporting Both OLTP and OLAP. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering. ICDE '11. Washington, DC, IEEE Computer Society, 291–302.

Arnaud Castelltort and Anne Laurent. 2014. NoSQL Graph-based OLAP Analysis: In SCITEPRESS - Science and and Technology Publications, 217–224.

Rick Cattell. 2011a. Scalable SQL and NoSQL Data Stores. SIGMOD Rec 39, 4 (May 2011), 12–27.

Rick Cattell. 2011b. Scalable SQL and NoSQL data stores. Acm Sigmod Rec. 39, 4 (2011), 12–27.

Fay Chang et al. 2008. Bigtable: A Distributed Storage System for Structured Data. ACM Trans Comput Syst 26, 2 (June 2008), 4:1–4:26.

S. Chaudhuri, U. Dayal, and V. Ganti. 2001. Database technology for decision support systems. Computer 34, 12 (December 2001), 48–55.

Surajit Chaudhuri and Umeshwar Dayal. 1997. An Overview of Data Warehousing and OLAP Technology. SIGMOD Rec 26, 1 (March 1997), 65–74.

Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. 2011. An overview of business intelligence technology. Commun. ACM 54, 8 (August 2011), 88.

Surajit Chaudhuri and Kyuseok Shim. 1994. Including group-by in query optimization. In VLDB. 354–366.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2016a. Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document. In Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, IEEE, 1–11.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2016b. Document-oriented Models for Data Warehouses - NoSQL Document-oriented for Data Warehouses.- 18th International Conference on Enterprise Information Systems, Volume 1 (ICEIS), SciTePress, 142–149.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, Ronan Tournier 2016c.Document-oriented data warehouses : complex hierarchies and summarizability. Dans : International Symposium on Ubiquitous Networking (UNet).

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, Ronan Tournier 2016d: Document-oriented data warehouses : models and extended cuboids. INFORSID 2016: 13-14

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015a. Benchmark for OLAP on NoSQL technologies comparing NoSQL multidimensional data warehousing solutions. In 9th IEEE International Conference on Research Challenges in Information Science, RCIS 2015, IEEE, 480–485.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015b. Entrepôts de données multidimensionnelles NoSQL. In Esteban Zimányi, Stijn Vansummeren, & Toon Calders, eds. Actes des 11es journées francophones sur les Entrepôts de Données et l'Analyse en Ligne, EDA 2015, RNTI. Hermann-Éditions, 161–176.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015c. How Can We Implement a Multidimensional Data Warehouse Using NoSQL. Lecture Notes in Business Information Processing. Springer International Publishing, 108–130.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015d. Implementation of Multidimensional Databases in Column-Oriented NoSQL Systems. ADBIS 2015-Springer International Publishing, 79–91.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015e. Implementation of Multidimensional Databases with Document-Oriented NoSQL. 17th International Conference, DaWaK 2015, 1-4, 2015, Proceedings. Lecture Notes in Computer Science. Springer, 379–390.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. 2015f. Implementing Multidimensional Data Warehouses into NoSQL. ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 1, SciTePress, 172–183.

Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, Ronan Tournier 2015g.Implantation Not Only SQL des bases de données multidimensionnelles. Dans : Colloque Veille Stratégique Scientifique et Technologique.

Kristina Chodorow. 2013. MongoDB: the definitive guide, O'Reilly Media, Inc.

George Colliat. 1996. OLAP, Relational, and Multidimensional Database Systems. SIGMOD Rec 25, 3 (September 1996), 64–69.

Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing. SoCC '10. New York, NY, USA: ACM, 143–154.

Transaction Processing Performance Council. 2008. TPC-H benchmark specification. Publ. Httpwww Tcp Orghspec Html (2008).

Douglas Crockford. 2006. The application/json media type for javascript object notation (json). (2006).

Philippe Cudré-Mauroux et al. 2013. Nosql databases for rdf: an empirical evaluation. In International Semantic Web Conference. Springer, 310–325.

Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. 2013. Data warehousing and OLAP over big data: current challenges and future research directions. In Proceedings of the sixteenth international workshop on Data warehousing and OLAP. ACM, 67–70.

D

Jérôme Darmont. 2007. Expérimentations avec le banc d'essais pour entrepôts de données DWEB. (2007).

Jerome Darmont, Fadila Bentayeb, and Omar Boussaid. 2007. Benchmarking data warehouses. Int. J. Bus. Intell. Data Min. 2, 1 (January 2007), 79–104.

Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: a flexible data processing tool. Commun. ACM 53, 1 (2010), 72–77.

Giuseppe DeCandia et al. 2007. Dynamo: Amazon's Highly Available Key-value Store. In Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles. SOSP '07. New York, NY, USA: ACM, 205–220.

Khaled Dehdouh, Omar Boussaid, and Fadila Bentayeb. 2014a. Columnar NoSQL Star Schema Benchmark. In Yamine Ait Ameur, Ladjel Bellatreche, & George A. Papadopoulos, eds. Model and Data Engineering. Lecture Notes in Computer Science. Springer International Publishing, 281–288.

Khaled Dehdouh, Omar Boussaid, and Fadila Bentayeb. 2014b. Columnar NoSQL Star Schema Benchmark. In Yamine Ait Ameur, Ladjel Bellatreche, & George A. Papadopoulos, eds. Model and Data Engineering. Lecture Notes in Computer Science. Springer International Publishing, 281–288. DOI:https://doi.org/10.1007/978-3-319-11587-0_26

Dewitt and Stonebraker. 2008. DeWitt and Stonebraker's "MapReduce: A major step backwards." (2008). Retrieved August 28, 2016 from http://craig-henderson.blogspot.fr/2009/11/dewitt-and-stonebrakers-mapreduce-major.html

Akon Dey, Alan Fekete, and Uwe Röhm. 2013. Scalable Transactions Across Heterogeneous NoSQL Key-value Data Stores. Proc VLDB Endow 6, 12 (August 2013), 1434–1439. DOI:https://doi.org/10.14778/2536274.2536331

Laurent D'Orazio and Sandro Bimonte. 2010. Multidimensional Arrays for Warehousing Data on Clouds. In Proceedings of the Third International Conference on Data Management in Grid and Peer-to-peer Systems. Globe'10. Berlin, Heidelberg: Springer-Verlag, 26–37.

E

Shakuntala Gupta Edward and Navin Sabharwal. 2015. MongoDB-Installation and Configuration. In Practical MongoDB. Springer, 35–52.

Eric Evans. 2004. Domain-driven design: tackling complexity in the heart of software, Addison-Wesley Professional.

F

Jianhua Feng and Jinhong Li. 2013. Google protocol buffers research and application in online game. In IEEE Conference Anthology. 1–4. DOI:https://doi.org/10.1109/ANTHOLOGY.2013.6784954

Avrilia Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel, and Donghui Zhang. 2012. Can the Elephants Handle the NoSQL Onslaught? Proc VLDB Endow 5, 12 (August 2012), 1712–1723. DOI:https://doi.org/10.14778/2367502.2367511

Myller Claudino de Freitas, Damires Yluska Souza, and Ana Carolina Salgado. 2016. Conceptual Mappings to Convert Relational into NoSQL Databases: In SCITEPRESS - Science and and Technology Publications, 174–181. DOI:https://doi.org/10.5220/0005836301740181

G

Alan F. Gates et al. 2009. Building a High-level Dataflow System on Top of Map-Reduce: The Pig Experience. Proc VLDB Endow 2, 2 (August 2009), 1414–1425.

Lars George. 2011. HBase: The Definitive Guide, O'Reilly Media, Inc.

Ahmad Ghazal et al. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD '13. New York, NY, USA: ACM, 1197–1208.

F Ghozzi, F Ravat, O Teste, G Zurfluh 2005 Méthode de conception d'une base multidimensionnelle contrainte. EDA, 51-70.

F. Ghozzi, F Ravat, O Teste, G Zurfluh 2003. Constraints and Multidimensional Databases. ICEIS (1), 104-111.

H

Marouane Hachicha and Jérôme Darmont. 2013. Problèmes d'additivité dus à la présence de hiérarchies complexes dans les modèles multidimensionnels: définitions, solutions et travaux futurs. In 9èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2013). Hermann, 7–16.

Hakan Hacígümüş, Jagan Sankaranarayanan, Junichi Tatemura, Jeff LeFevre, and Neoklis Polyzotis. 2013. Odyssey: A Multistore System for Evolutionary Analytics. Proc VLDB Endow 6, 11 (August 2013), 1180–1181.

Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. 1996. Implementing Data Cubes Efficiently. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. SIGMOD '96. ACM, 205–216.

Ali Hassan. 2014. Modélisation des bases de données multidimensionnelles: analyse par fonctions d'agrégation multiples. Toulouse 1.

Ali Hassan, Franck Ravat, Olivier Teste, Ronan Tournier, and Gilles Zurfluh. 2012. Differentiated multiple aggregations in multidimensional databases. In International Conference on Data Warehousing and Knowledge Discovery. Springer, 93–104.

Florian Holzschuher and René Peinl. 2013. Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops. ACM, 195–204.

Chao-Wen Huang, Wan-Hsun Hu, Chia Chun Shih, Bo-Ting Lin, and Chien-Wei Cheng. 2013. The improvement of auto-scaling mechanism for distributed database-A case study for MongoDB. In APNOMS. 1–3.

Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX Annual Technical Conference. 9.

I

IBM, Paul Zikopoulos, and Chris Eaton. 2011. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data 1st ed, McGraw-Hill Osborne Media.

W. H. Inmon, Claudia Imhoff, and Greg Battas. 1995. Building the Operational Data Store. John Wiley & Sons, Inc., New York, NY, USA.

Ming-Yee Iu and Willy Zwaenepoel. 2010. HadoopToSQL: A mapReduce Query Optimizer. In Proceedings of the 5th European Conference on Computer Systems. EuroSys '10. New York, NY, USA: ACM, 251–264. DOI:https://doi.org/10.1145/1755913.1755939

Todor Ivanov et al. 2015. Big Data Benchmark Compendium. In Raghunath Nambiar & Meikel Poess, eds. Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things. Lecture Notes in Computer Science. Springer International Publishing, 135–155.

J-K

H Jerbi, G Pujolle, F Ravat, O Teste 2011. Recommandation de requêtes dans les bases de données multidimensionnelles annotées Ingénierie des systèmes d'information 16 (1), 113-138

H Jerbi, F Ravat, O Teste, G Zurfluh Preference-based recommendations for OLAP analysis International Conference on Data Warehousing and Knowledge Discovery, 467-478 2009

H Jerbi, F Ravat, O Teste, G Zurfluh Applying recommendation technology in OLAP systems International Conference on Enterprise Information Systems, 220-233 2009

H Jerbi, F Ravat, O Teste, G Zurfluh Management of context-aware preferences in multidimensional databases Digital Information Management, 2008. ICDIM 2008. 2008

Owen Kaser and Daniel Lemire. 2003. Attribute Value Reordering for Efficient Hybrid OLAP. In Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP. DOLAP '03. New York, NY, USA: ACM, 1–8.

Harpinder Kaur and Janpreet Singh. Improvement in Load Balancing Technique for MongoDB Clusters.

Ankur Khetrapal and Vinay Ganesh. 2006. HBase and Hypertable for large scale distributed storage systems. Dept Comput. Sci. Purdue Univ. (2006), 22–28.

Ralph Kimball and Margy Ross. 2013. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, John Wiley & Sons.

Ralph Kimball and Margy Ross. 2010. The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence, John Wiley & Sons.

Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jimenez-Peris, Raquel Pau, and José Pereira. 2016. The CloudMdsQL Multistore System. In Proceedings of the 2016 International Conference on Management of Data. SIGMOD '16. ACM, 2113–2116.

L

Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. SIGOPS Oper Syst Rev 44, 2, 35–40.

N. Leavitt. 2010. Will NoSQL Databases Live Up to Their Promise? Computer 43, 2 (February 2010), 12–14. DOI:https://doi.org/10.1109/MC.2010.58

Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. 2012a. Parallel Data Processing with MapReduce: A Survey. SIGMOD Rec 40, 4.

Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. 2012b. Parallel Data Processing with MapReduce: A Survey. SIGMOD Rec 40, 4 (January 2012), 11–20. DOI:https://doi.org/10.1145/2094114.2094118

Jeff LeFevre, Jagan Sankaranarayanan, Hakan Hacigumus, Junichi Tatemura, Neoklis Polyzotis, and Michael J. Carey. 2014a. MISO: Souping Up Big Data Query Processing with a Multistore System. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. SIGMOD '14. ACM, 1591–1602.

Jeff LeFevre, Jagan Sankaranarayanan, Hakan Hacigumus, Junichi Tatemura, Neoklis Polyzotis, and Michael J. Carey. 2014b. MISO: Souping Up Big Data Query Processing with a Multistore System. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. SIGMOD '14. New York, NY, USA: ACM, 1591–1602. DOI:https://doi.org/10.1145/2588555.2588568

Bing Li, Junbo Zhang, Ning Yu, and Yi Pan. 2016. J2M: a Java to MapReduce translator for cloud computing. J. Supercomput. 72, 5 (May 2016), 1928–1945.

M

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

Imen Megdiche Bousarsar. 2015. Intégration holistique et entreposage automatique des données ouvertes. Université de Toulouse, Université Toulouse III-Paul Sabatier.

Mongoimport. Réinvention de la gestion des informations | MongoDB. Retrieved August 27, 2016 from https://www.mongodb.com/fr

A.B.M. Moniruzzaman and Syed Akhter Hossain. 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison.

Konstantinos Morfonios, Stratis Konakas, Yannis Ioannidis, and Nikolaos Kotsis. 2007a. ROLAP Implementations of the Data Cube. ACM Comput Surv 39, 4.

Konstantinos Morfonios, Stratis Konakas, Yannis Ioannidis, and Nikolaos Kotsis. 2007b. ROLAP Implementations of the Data Cube. ACM Comput Surv 39, 4.

R. Moussa. 2012. TPC-H benchmarking of Pig Latin on a Hadoop cluster. In 2012 International Conference on Communications and Information Technology (ICCIT). 85–90.

James Murty. 2008. Programming amazon web services: S3, EC2, SQS, FPS, and SimpleDB, O'Reilly Media, Inc.

N

Raghunath Othayoth Nambiar and Meikel Poess. 2006. The making of TPC-DS. In Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 1049–1058.

N

Michael A. Olson, Keith Bostic, and Margo I. Seltzer. 1999. Berkeley DB. In USENIX Annual Technical Conference, FREENIX Track. 183–191.

Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig Latin: A Not-so-foreign Language for Data Processing. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. SIGMOD '08. ACM, 1099–1110.

O

Owen O'Malley. 2008. Terabyte sort on apache hadoop. Yahoo Available Online Httpsortbenchmark OrgYahoo-Hadoop PdfMay (2008), 1–3.

Patrick O'Neil and Goetz Graefe. 1995. Multi-table Joins Through Bitmapped Join Indices. SIGMOD Rec 24, 3 (September 1995), 8–11.

Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009a. The Star Schema Benchmark and Augmented Fact Table Indexing. In Raghunath Nambiar & Meikel Poess, eds. Performance Evaluation and Benchmarking. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 237–252.

Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009b. The Star Schema Benchmark and Augmented Fact Table Indexing. In Raghunath Nambiar & Meikel Poess, eds. Performance Evaluation and Benchmarking. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 237–252.

P-Q

Zachary Parker, Scott Poe, and Susan V. Vrbsky. 2013. Comparing nosql mongodb to an sql db. In Proceedings of the 51st ACM Southeast Conference. ACM, 5.

Andrew Pavlo et al. 2009. A Comparison of Approaches to Large-scale Data Analysis. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. SIGMOD '09. ACM, 165–178.

Torben Bach Pedersen and Christian S. Jensen. 1999. Multidimensional data modeling for complex data. In Data Engineering, 1999. Proceedings., 15th International Conference on. IEEE, 336–345.

Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. 2007. Why You Should Run TPC-DS: A Workload Analysis. In Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB '07. Vienna, Austria: VLDB Endowment, 1138–1149.

Meikel Poess, Bryan Smith, Lubor Kollar, and Paul Larson. 2002. Tpc-ds, taking decision support benchmarking to the next level. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM, 582–587.

Seth Proctor. 2013. Exploring the Architecture of the NuoDB Database, Part 1. Dosegljivo Na Httpwww Infoq Comarticlesnuodb-Archit.-1 Dostopano Sept. 2014 (2013).

G Pujolle, F Ravat, O Teste, R Tournier, G Zurfluh 2011. Multidimensional database design from document-centric XML documentsInternational Conference on Data Warehousing and Knowledge Discovery, 51-65.

R

F Ravat, O Teste, R Tournier, G Zurfluh 2010. Finding an application-appropriate model for XML data warehouses Information Systems 35 (6), 662-687.

F Ravat, O Teste 2009 Personalization and OLAP databases New Trends in Data Warehousing and Data Analysis, 1-22

F Ravat, O Teste, R Tournier, G Zurfluh 2008a Top_Keyword: an aggregation function for textual document OLAP International Conference on Data Warehousing and Knowledge Discovery, DAWAK'08, 55-64.

F Ravat, O Teste 2008b. Personalization and OLAP databases Annals of Information Systems, New Trends in Data Warehousing and Data.

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

Franck Ravat, Olivier Teste, and Ronan Tournier. 2008d. Analyse multidimensionnelle de documents via des dimensions OLAP. Doc. Numéro. 10, 2, 85–104.

F Ravat, O Teste, R Tournier 2007a. Olap aggregation function for textual data warehouse. ICEIS, 151-156

Franck Ravat, Olivier Teste, Ronan Tournier, Gilles Zurfluh 2007b. Querying Multidimensional Databases. Dans : East-European Conference on Advances in Databases and Information Systems (ADBIS), Varna, Bulgarie, Springer, LNCS 4690, p. 298-313.

F Ravat, O Teste, R Tournier, G Zurlfluh 2007c. A conceptual model for multidimensional analysis of documents International Conference on Conceptual Modeling, ER'07, 550-565

F Ravat, O Teste, R Tournier, G Zurfluh 2007d. Graphical querying of multidimensional databases East European Conference on Advances in Databases and Information Systems ADBIS'07.

Franck Ravat 2007e. Modèles et outils pour la conception et la manipulation de systèmes d'aide à la décision, F. Ravat, Habilitation à Diriger des Recherches en Informatique de l'Université Toulouse I.

Ravat, Olivier Teste, and Gilles Zurfluh. 2006. A multiversion-based multidimensional model. In International Conference on Data Warehousing and Knowledge Discovery. Springer, 65–74.

F Ravat, O Teste, G Zurfluh 2005.Constraint-Based Multi-Dimensional Databases Database Modeling for Industrial Data Management, 323-368

F Ravat, O Teste, G Zurfluh 2002 Langage pour bases multidimensionnelles: OLAP-SQL INGENIERIE DES SYSTEMS D INFORMATION 7 (3), 11-38.

F Ravat, O Teste, G Zurfluh 2001 Modélisation multidimensionnelle des systèmes décisionnels. EGC, 201-212,

F. Ravat and Olivier Teste. 2000. A temporal object-oriented data warehouse model. In International Conference on Database and Expert Systems Applications. Springer, 583–592

F Ravat, O Teste, G Zurfluh 1999. Towards data warehouse design CIKM'99.

S

Pramod J. Sadalage and Martin Fowler. 2012. NoSQL distilled: a brief guide to the emerging world of polyglot persistence, Pearson Education.

Lucas C. Scabora, Jaqueline J. Brito, Ricardo Rodrigues Ciferri, and Cristina Dutra de Aguiar Ciferri. 2016. Physical Data Warehouse Design on NoSQL Databases - OLAP Query Processing over HBase: In SCITEPRESS - Science and and Technology Publications, 111–118.

Stefanie Scherzinger, Meike Klettke, and Uta Störl. 2013. Managing Schema Evolution in NoSQL Data Stores. ArXiv13080514 Cs (August 2013).

Jie Song, Chaopeng Guo, Zhi Wang, Yichan Zhang, Ge Yu, and Jean-Marc Pierson. 2015. HaoLap: A Hadoop based OLAP system for big data. J. Syst. Softw. 102 (April 2015), 167–181.

D.S. Stevenson et al. 2006. Multimodel ensemble simulations of present-day and near-future tropospheric ozone. J. Geophys. Res. Atmospheres 111, D8 (2006).

Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2005. C-store: a column-oriented DBMS. In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment 553-564. Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. 2007. The end of an architectural era:(it's time for a complete rewrite). In Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 1150–1160.

Michael Stonebraker and Rick Cattell. 2011. 10 Rules for Scalable Performance in "Simple Operation" Datastores. Commun ACM 54, 6 (June 2011), 72–80.

Michael Stonebraker and Ariel Weisberg. 2013. The VoltDB Main Memory DBMS. IEEE Data Eng Bull 36, 2 (2013), 21–27.

Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. 2012. Serving Large-scale Batch Computed Data with Project Voldemort. In Proceedings of the 10th USENIX Conference on File and Storage Technologies. FAST'12. Berkeley, CA, USA: USENIX Association, 18–18.

T

Ronald C. Taylor. 2010. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinformatics 11, Suppl 12 (2010), S1.

Olivier Teste. 2000. Modélisation et manipulation d'entrepôts de données complexes et historisées. Université Paul Sabatier-Toulouse III.

O Teste Towards conceptual multidimensional design in decision support systems 2010

O. Teste 2009 Modélisation et manipulation des systèmes OLAP : de l'intégration des documents à l'usager. Habilitation à Diriger des recherches de l'Université Toulouse 3.

Ashish Thusoo et al. 2009. Hive: A Warehousing Solution over a Map-reduce Framework. Proc VLDB Endow 2, 2 (August 2009), 1626–1629.

Ronan Tournier. 2007. Analyse en ligne (OLAP) de documents. phd thesis. Université Paul Sabatier - Toulouse III.

U

Jeffrey D. Ullman. 1996. Efficient Implementation of Data Cubes Via Materialized Views. In KDD. 386–388.

V-W

Mehul Nalin Vora. 2011. Hadoop-HBase for large-scale data. In 2011 International Conference on Computer Science and Network Technology (ICCSNT). 601–605.

Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. 2015. Neo4j in Action, Manning.

Y-Z

Rania Yangui, Ahlem Nabli, and Faiez Gargouri. 2016. Automatic Transformation of Data Warehouse Schema to NoSQL Data Base: Comparative Study. Procedia Comput. Sci. 96 (2016), 255–264.

Hongwei Zhao and Xiaojun Ye. 2013. A Practice of TPC-DS Multidimensional Implementation on NoSQL Database Systems. In Raghunath Nambiar & Meikel Poess, eds. Performance Characterization and Benchmarking. Lecture Notes in Computer Science. Springer International Publishing, 93–108.

Yue Zhuge and Hector Garcia-Molina. 1998. Graph structured views and their incremental maintenance. In Data Engineering, 1998. Proceedings., 14th International Conference on. IEEE, 116–125.