Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE SAAD DAHLAB DE BLIDA 1

Faculté des Sciences

Département d'informatique

Mémoire Master Ingénierie Logiciel

Outil Graphique de Génération de Benchmarks NoSQL (OGGB-NoSQL)

Réalisé par :

- ✓ Melle ZIANI Khouloud
- ✓ Melle SALMA Manel

Encadré par :
M. BALA Mahfoud
Co-encadré par:
M. DEHDOUH Khaled

Soutenu le 30/09/2019 devant le jury composé de :

Mme Imane CHERFA	Maitre Assistante	Dépt. d'Informatique, U. Blida 1	**Présidente**
Mme Yasmine MANCER	Maitre Assistante	Dépt. d'Informatique, U. Blida 1	**Examinatrice**
M. Mahfoud BALA	Maitre de Conférences	Dépt. d'Informatique, U. Blida 1	**Encadreur**
M. Khaled DEHDOUH	Maitre de Conférences	Dépt. M.I, AMC	**Co-Encadreur**

Promotion: 2018/2019
Session : Septembre

Résumé

Ce sujet est à la croisée des entrepôts de données, des modèles de stockage NoSQL, et du traitement de données massives (méga-données).

L'objectif principal étant de fournir un banc d'essai qui permet de simuler et d'évaluer un entrepôt de données multidimensionnelles NoSQL orienté documents, avec les différentes manipulations OLAP effectuées par un analyste.

Dans le domaine des entrepôts de données multidimensionnelles et l'analyse de données en ligne (OLAP), les bancs d'essai références sont TPC-DS, TPC-H et le SSB. Cependant, ces solutions ne sont pas définies dans un environnement BigData pour une utilisation dans un système distribué ou des bases de données NoSQL. Leur processus de génération de données nécessite beaucoup plus de temps lorsqu'il est question d'évaluer un large volume de données (Téraoctet voire 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.

Dans ce contexte, et en l'absence de benchmarks décisionnels conçus pour les systèmes NoSQL, nous proposons de réaliser un outil graphique de génération de données synthétiques .En effet, il offre l'opportunité de générer les données selon le modèle conceptuel de l'entrepôt de données en étoile « *SSB*» (*Star Schema Benchmark*). Lors de la génération des données, plusieurs aspects sont pris en charge par cet outil, tels que le format des données et le volume de données générées.

Mots clés : Entrepôt de données, NoSQL, banc d'essai, orienté documents, BigData, OLAP, OLAP, benchmark décisionnel, système distribué, SSB.

Abstract

Our study deals with issues that are at the cross between Data Warehouses, NoSQL data models, and BigData processing.

This topic aims to provide a test benchmark that can simulate and evaluate a multidimensional data warehouse NoSQL (document store), with different OLAP manipulations performed by an analyst.

In the Data Warehousing and OLAP fields, the well-known test benches used are TPC-DS, TPC-H and SSB. However, these environments are not defined for use in a distributed system or NoSQL databases. Their data generation process requires much more time when it comes to evaluating large volumes of data (Teraoctet and more). Comparing systems with large amounts of data has become crucial. The higher the volume, the more memory limitations we face when configuring with one machine.

In this context, and in the absence of decision-making benchmarks designed for NoSQL systems, we propose to create a graphical tool for generating synthetic data. Indeed, it offers the opportunity to generate data according to the conceptual model of the system. Star data warehouse "SSB" (Star Schema Benchmark). When generating data, several aspects are supported by this tool, such as the data format and the amount of data generated.

Keywords: Data Warehouses, NoSQL, BigData, document store, OLAP, well-known test benches, distributed system, decision-making benchmarks, SSB.

Remerciements

Nous remercions le bon Dieu le tout puissant de nous avoir accordé le

courage et la patience

pour mener à terme le présent mémoire.

Nous tenons, également, à exprimer notre sincère reconnaissance et notre profonde gratitude à notre promoteur Mr BALA Mahfoud, pour sa disponibilité, ses orientations, ses précieux conseils et ses encouragements qui nous ont permis de mener à bien ce travail.

Nous remercions chaleureusement notre encadreur Mr DEHDOUH Khaled pour sa disponibilité, ses orientations et ses conseils.

Nous tenons à exprimer notre gratitude aux membres de jury pour avoir accepté de juger ce travail.

Un immense merci à Mr YALAOUI Moussa pour l'accueil et la disponibilité que nous a donné.

Un énorme merci à nos parents Mr SALMA Hamid et Mr ZIANI Abdelkader pour leur amour, sacrifices et surtout leur patience, sans oublier nos familles et amis pour leurs éternel soutient et la confiance en nos capacités.

Abréviations

ACID	Atomicité, Cohérence, Isolation, Durabilité
AGPL	Affero General Public License
API	Application Program Interface
BD	Base de Données
BSON	Binary Structured Object Notation
CAP	Consistency, Availability, Partition tolerance
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
DSS	Decision Support System
ED	Entrepot de Données
ETL	Extract, Transform, Load
FN	Forme Normale
HADOOP	High-Availability Distributed Object Oriented Platform
HDFS	Hadoop Distributed File System
HOLAP	Hybrid OnLine Analytical Processing
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MLOD	Modèle Logique Orienté Documents
MOLAP	Multidimensional OnLine Analytical Processing
NoSQL	Not Only SQL

OGGB	Outil Graphique de Génération de Benchmark
OLAP	On Line Analytical Processing
OLTP	On-Line Transactional Processing
RAM	Random Access Memory
RDD	Resilient Distributed DataSet
RDF	Resource Description Framework
ROLAP	Relational OnLine Analytical Processing
SBT	Scala Build Tool
SGBD	System de Gestion de Base de Données
SGBDR	Système de Gestion de Base de Données Relationnelle
SID	Système d'Information Décisionnel
SPOF	Single Point Of Failure
SQL	Structured Query Language
SSB	Star Schema Benchmark
TD	Table de Dimension
TF	Table de Fais
TPC	Transaction Processinf Council
XML	eXtensible Markup Language

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Introduction générale

1. Contexte général

En quelques années, le volume des données numérique généré par les systèmes d'information des entreprises a considérablement augmenté. Émanant de sources diverses (transactions, comportements, réseaux sociaux, géolocalisation...), elles sont souvent structurées autour d'un seul point d'entrée, la clé, et susceptibles de croître très rapidement. Autant de caractéristiques qui les rendent très difficiles à traiter avec des outils classiques de gestion de données. Par ailleurs, l'analyse de grands volumes de données, appelés communément BigData, défie également les moteurs de bases de données traditionnels.

C'est pour répondre à ces différentes problématiques que sont nées les bases de données *NoSQL* (*Not Only SQL*), sous l'impulsion de grands acteurs du Web comme Facebook ou Google, qui les avaient développées à l'origine pour leurs besoins propres. Grâce à leur flexibilité et leur souplesse, ces bases non relationnelles permettent en effet de gérer de gros volumes de données hétérogènes sur un ensemble de serveurs de stockage distribués, avec une capacité de montée en charge très élevée. Et donc pour évaluer ces bases de données il nous faut des bancs d'essai (*benchmarks*) qui offrent aux chercheurs, industriels et étudiants un moyen incontournable en termes de données et de requêtes pour évaluer leurs produits et les comparer à d'autres produits tout en considérant les mêmes données et les mêmes requêtes. L'un des objectifs principaux d'un banc d'essai est d'estimer les performances d'un système logiciel, considéré comme prototype, tels que les SGBD ou de comparer les performances de plusieurs systèmes à travers une série de tests dont l'évaluation repose sur un ensemble de métriques tels que le temps de réponse, le débit, l'énergie consommée, le nombre d'entrées/sorties effectuées ainsi que les ressources consommées (RAM, cache, disque, bande passante...)

2. Problématique

Différents bancs d'essai 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 BigData, les systèmes distribués et les bases de données NoSQL se vulgarisent et connaissent aujourd'hui une large utilisation. Dans ce contexte, nous avons besoin de solutions de bancs d'essai compatibles avec ces nouveaux systèmes NoSQL distribués sur plusieurs machines.
Dans le cadre de ce travail, nous nous intéressons, particulièrement, aux bancs d'essai utilisés dans le contexte des entrepôts de données multidimensionnels et l'analyse de données en ligne (*OLAP : On Line Analytical Processing*) où plusieurs solutions existent telles que le TPC-DS (Poess et al, 2007), TPC-H et le SSB (O'Neil et al, 2009b). En revanche, 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 nécessite beaucoup plus de temps lorsqu'il est question d'évaluer un large volume de données (Teraoctet voire 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 la RAM d'espace disque lors d'une configuration avec une seule machine. Les nouvelles solutions BigData permettent le passage à l'échelle et pallient à ces problèmes de mémoire et d'espace disque. 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. Les bancs d'essais existants ne sont pas adaptés à cette approche distribuée et évolutive puisqu'ils génèrent les données sur une seule machine.

3. Motivations / Objectif

Nous nous proposons, dans le cadre de ce travail, de mettre en œuvre un outil graphique de génération de données synthétiques « OGGB » selon le modèle NoSQL orienté documents dans un environnement distribué et ce pour des fins de benchmarking. Cet outil a été conçu et développé pour permettre de simuler et d'évaluer un entrepôt de données multidimensionnelles orienté document. En effet, il offre l'opportunité de générer les données selon le modèle conceptuel de l'entrepôt de données en étoile « *SSB* » (*Star Schema Benchmark*) et selon le modèle logique NoSQL orienté documents. Quant au niveau physique, nous avons choisi comme formats intermédiaires JSON, TBL et CSV et comme environnement de stockage cible le SGBD MongoDB. Lors de la génération des données, plusieurs aspects sont pris en charge par cet outil, tels que le format des données et le volume de données générées.

En résumé, l'OGGB est défini selon les points suivants :

- ✓ Le modèle conceptuel SSB comme prototype d'ED.
- ✓ Le modèle logique NoSQL orienté document
- ✓ Le modèle physique du SGBD MongoDB
- ✓ Le mode de génération de données distribué ou mono machine.
- ✓ Le volume de données en octets à générer est paramétrable via le facteur d'échelle (*SF : Scale Factor*).
- ✓ Les formats de sortie de données fournis par OGGB sont les plus appropriés aux outils NoSQL de chargement de données à savoir TBL, CSV et JSON.

Organisation du mémoire

Nous commencerons dans ce mémoire par une introduction générale décrivant le contexte de travail, la problématique, la solution adoptée et l'objectif de ce sujet.

La deuxième partie s'articule autour de trois chapitres qui se présentent comme suit :

Dans le premier chapitre, on a introduit les concepts des systèmes décisionnels, les entrepôts de données, la modélisation dimensionnelle. On a également présenté le NoSQL, l'un des principales technologies du BigData, ainsi que des bancs d'essai décisionnels TPC et SSB. Nous avons terminé cette partie réservée aux fondamentaux par une étude des travaux connexes.

Le deuxième chapitre est consacré à la description de la solution proposée c'est-à-dire l'outil OGGB ainsi que leurs composants, nous avons aussi mentionnées les caractéristiques communes entre ces composants avec le cas d'utilisation de l'outil.

Dans le troisième et le quatrième chapitre, on a abordé la partie technique qui consiste à déployer, configurer et tester l'ensemble des éléments logiciels nécessaires pour la mise en œuvre de la solution.

Nous achèverons avec une conclusion, dans laquelle on va synthétiser tous les résultats obtenus et les enseignements acquis durant la réalisation de ce projet.

Chapitre I : Généralités et concepts de base

I. Généralités et Concepts de Base

Introduction

Toutes les entreprises du monde disposent d'une masse de données plus ou moins considérable. Ces informations proviennent principalement des sources internes (générées par leurs systèmes opérationnels au fil des activités journalières), mais éventuellement de sources externes (web, partenaires, institutions gouvernementales et banques…). Cette surabondance de données, et l'impossibilité des systèmes opérationnels traditionnels de les exploiter pour des fins d'analyse conduit, inévitablement, l'entreprise à se tourner vers une nouvelle informatique dite décisionnelle qui met l'accent sur la compréhension de l'environnement de l'entreprise et l'exploitation de ces données à bon escient. En effet, les décideurs ont besoin d'avoir une meilleure vision de l'environnement et de l'évolution de leur entreprise, ainsi, que des informations auxquelles ils peuvent se fier. Cela ne peut se faire qu'en mettant en place des indicateurs « business » clairs et pertinents. Ces indicateurs permettant la sauvegarde, l'utilisation de la mémoire de l'entreprise et offrant la possibilité de se reporter à ces indicateurs pour une bonne prise de décision.

L'entrepôt de données (*DW : Data Warehouse*), constitue une technologie de stockage de données des plus incontournables pour la mise en place d'applications décisionnelles. Les systèmes d'aide à la décision (*DSS : Decision Support System*) sont des systèmes d'information intelligents. Ces systèmes sont basés sur des modèles de décision qui peuvent être utilisés pour extraire de grandes quantités de données à partir des bases de données et ce afin de fournir des interfaces et des méthodes pour le traitement efficace, et pour obtenir des décisions significatives d'importance à la gestion / économique de celui-ci.

Les DSS ont été utilisés pour analyser une grande quantité d'informations sur la performance et compétitivité de l'entreprise et de fournir un support facilement accessible pour la distribution des connaissances générées à différentes catégories d'intervenants. L'évaluation de la performance des systèmes décisionnels est une activité non négligeable, en raison de différentes architectures et fonctionnalités pour des besoins spécifiques. Ce chapitre présente un aperçu sur les concepts de base de l'informatique décisionnelle, les méga-données, les systèmes distribués et se terminera par un exposé des trois modèles de benchmarks références à savoir TPC-DS, TPC-H et la SSB.

1. Système décisionnel

Les systèmes décisionnels sont nés d'un besoin exprimé par les entreprises qui n'étaient pas satisfaites par les systèmes traditionnels de bases de données. En intégrant la technologie des entrepôts de données (*DW : Data Warehouses*), le processus décisionnel apporte une réponse au problème de la croissance continuelle des données pouvant être issues de sources diverses et de formats différents. De plus, il supporte efficacement les processus d'analyse des données en ligne (*OLAP : On-Line Analytical Processing*)[1] .

1.1 Définition

"Le Décisionnel est le processus visant à transformer les données en informations et, par l'intermédiaire d'interrogations successives, transformer ces informations en connaissances"[2].
« Un Système d'Information Décisionnel SID est un ensemble de données organisé de façon spécifique, approprié à la prise de décision ».
Un SID est basé sur trois phases à savoir :

1. phase d'intégration des données (Extraction, Transformation et Chargement (*ETL : Extrat-Transform-Load),*
2. phase structuration
3. phase restitution ou bien la phase OLAP.

Le SID est basé sur la technologie d'entreposage de données DW.

1.2 Architecture décisionnelle

Comme le montre la figure I.1, un système d'information décisionnel SID comprend quatre composants principaux

i. Collecte de données (*Data Pumping*) : La collecte de données est une étape primordiale qui consiste à récolter les données à partir de plusieurs sources opérationnelles de l'entreprise. Ces données sont, souvent, hétérogènes, non standardisées. Les sources de données peuvent être des systèmes opérationnels de production, des archives, des données internes ou externes.

ii. Le processus *ETL* (*Extract, Transform* et *Load*) «est un système par lequel passent toutes les données des systèmes opérationnels avant d'aboutir à la forme souhaitée dans l'entrepôt de données. Les données en sortie sont nettoyées, purifiées, contextualisées et prêtes à être chargées dans l'entrepôt ».
Les ETL correspondent à une surface de travail et représentent l'intermédiaire entre le système opérationnel et l'interface du système décisionnel. Ce processus est constitué de trois étapes :
-Extraction : c'est la première étape qui consiste à identifier les données à partir des différentes sources. Elle consiste à lire, comprendre la source de données puis extraire les données pertinentes et les orienter vers le système décisionnel.
-Transformation : après l'extraction des données, ces dernières subissent un travail de transformation qui inclut le nettoyage, le formatage et la standardisation, la fusion ou l'éclatement et l'affectation des clés de substitution.
-Chargement : C'est la dernière étape qui permet de charger ces données vers la 'surface présentation' du système décisionnel. Il y a deux types de chargement : le chargement initial qui est le premier chargement de l'entrepôt et dans des cas spéciaux comme la perte des données, toutes les données seront chargées et le chargement incrémentiel qui est le fait d'ajouter les nouvelles données à l'entrepôt existant. C'est une opération qui se répète périodiquement.

iii. Stockage de données : Un entrepôt de données est un moyen pour centraliser un volume important de données dans un même endroit unifié et accessible par les applications d'analyse et d'aide à la décision.

iv. Analyse et Restitution : Pour faciliter l'accès à l'information pour tous les utilisateurs selon leurs profils métiers et afin d'extraire les éléments de décision pour dynamiser la réactivité globale dans l'entreprise, certains outils ont été mis à la disposition des décideurs et consistent à restituer des informations sous forme de rapports (*reporting*) et tableaux de bord (*dashboard*). Certains outils plus sophistiqués permettent de la fouille de données pour l'extraction des connaissances (*Data mining*).

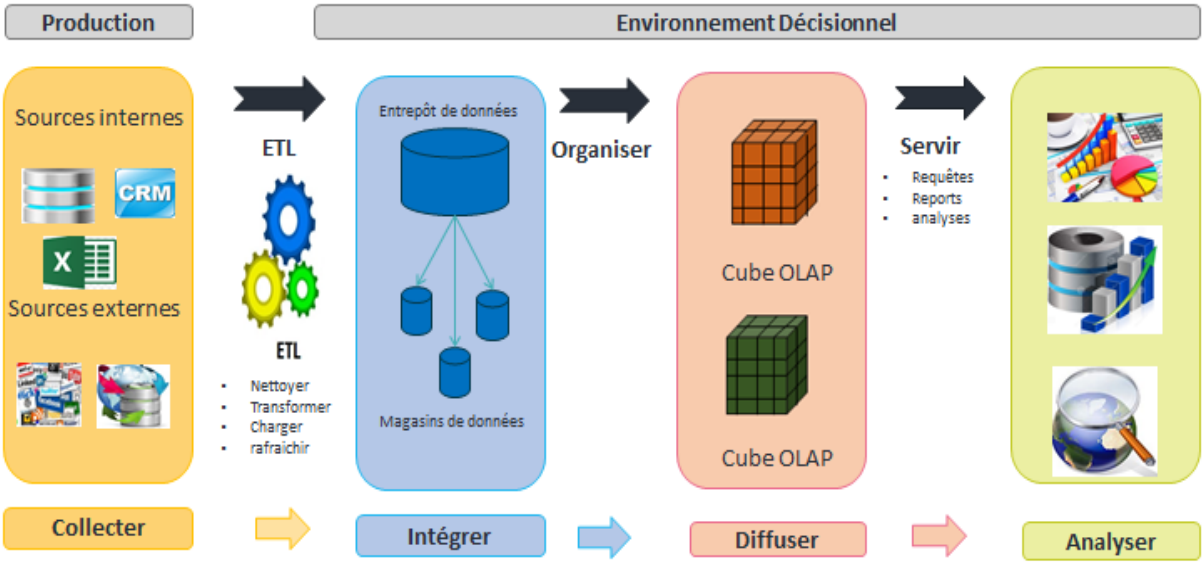

Figure I.1 : Architecture décisionnelle[3]

Dans le contexte des systèmes décisionnels, nous avons étudié les systèmes *OLAP* « *On-Line Analytical Processing* » (Codd et al, 1993) qui proposent de :
− Collecter les données pertinentes,
− Les organiser selon des structures adaptées à la prise de décision,
− Les interroger d'une manière interactive et dynamique.
Une plus ample description de ces systèmes est présentée dans la section suivante.

1.3 Les systèmes OLAP

Dans la littérature, plusieurs définitions sont proposées pour les systèmes OLAP (OLAP Report)[1] .Dans ces définitions, les caractéristiques de base sont la structure dimensionnelle des données, les données forment des points dans un espace à plusieurs dimensions, et l'interactivité de l'interrogation afin de s'approcher de la perception du décideur et de l'aider au mieux dans son processus de prise de décision. Nous proposons alors de définir les systèmes OLAP comme suit :
Définition : Un système OLAP est un système d'information décisionnel qui organise les données dans un espace dimensionnel. Il regroupe un ensemble d'outils en interaction qui réalisent la

[1] OLAP Report: http://www.olapreport.com/. C'est une ressource indépendante de recherche pour des organismes achetant et mettant en ligne des applications OLAP.

synthèse dynamique, l'analyse interactive et l'agrégation d'un grand volume de données afin d'améliorer le processus de prise de décision.

Ce système réunit un ensemble de nouvelles fonctionnalités décrites par 12 règles[4]. Les principales caractéristiques extraites de ces règles sont :

− La vision dimensionnelle des données, la transparence entre l'outil de visualisation et l'espace de stockage des données dimensionnelles,

− L'interopérabilité (l'outil rend invisible à l'utilisateur l'hétérogénéité des données),

− La manipulation intuitive des données et la flexibilité des restitutions, le décideur dispose d'une interface ergonomique de consultation.

Les systèmes OLAP visent à combler les lacunes des systèmes transactionnels. En effet, une des principales caractéristiques des systèmes transactionnels, est une activité de modification et d'interrogation fréquentes et répétitives[5]. L'accès au système est réalisé par de très courtes transactions. Enfin, la plupart de ces systèmes ne conservent pas les évolutions des données manipulées, seules les versions courantes sont conservées.

Dans la littérature, nous retrouvons souvent le mot OLAP associé à l'approche des entrepôts de données. Cette approche représente un axe de recherche dans le contexte des systèmes décisionnels. La section suivante présente les caractéristiques, les limites de cette approche.

1.4 Les entrepôts de données

Les entrepôts de données sont apparus en 1996, réponse au besoin de rassembler toutes les informations d'une entreprise en une base de données unique destinée aux analystes et aux gestionnaires. Cela en intégrant des informations provenant de différentes sources de données internes mais aussi externes à l'environnement de l'organisme et en offrant la possibilité de faire des analyses et des corrélations sur des agrégations créées dynamiquement à partir de plusieurs dimensions .

1.4.1 Définition et caractéristiques

Bill Inmon définit l'entrepôt de données dans son ouvrage "*Building Data warehouse*" de la façon suivante[6] : « L'entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et historiées, organisées pour support d'un processus d'aide à la décision ».

Cette définition d'ED a été conceptualisée en termes de caractéristiques du référentiel des données, qui seront détaillées dans les points suivants :

• Orientées sujet : Les données de l'entrepôt sont organisées par thème (autour des sujets majeurs et des métiers de l'entreprise). L'intérêt dans cette organisation est de disposer d'un ensemble d'informations utiles sur un sujet transversal aux structures fonctionnelles et organisationnelles de l'entreprise [6].

• Intégrées : Les données dans l'entrepôt proviennent de différentes sources éventuellement hétérogènes. L'intégration est un processus qui consiste à résoudre les problèmes d'hétérogénéité, où les données contenues dans ED sont divisées en grandes subdivisions appelées domaine[7].

• Non volatiles : Les données stockées au sien de l'entrepôt sont permanentes et ne peuvent être modifiées, et le rafraîchissement de l'entrepôt de données, consiste seulement à ajouter de nouvelles données sans modifier ou perdre celle qui existent. Ceci pour conserver la traçabilité des informations et des décisions prises [7].

• Historisées : Pour suivre dans le temps l'évolution des différentes valeurs des indicateurs à analyser, l'historisation est nécessaire. Un référentiel de temps est associé aux données, afin de permettre l'identification dans la durée de valeurs précise [6].

1.4.2 Concepts de base

La modélisation des données décisionnelles s'appelle modélisation multidimensionnelle ou modélisation dimensionnelle. Le formalisme utilisé est basé sur deux concepts clés : le fait et la dimension. Cette modélisation est valable que ce soit pour un entrepôt de données, un magasin de données ou pour un cube de données.

- Fait: est un point de vue d'analyse. Il s'agit d'une table appelée table de fait qui regroupe un ensemble de données observables appelés mesures que l'on souhaite analyser selon plusieurs axes d'analyse appelés dimensions.
- Mesure : est un attribut qui permet de mesurer un fait. Un fait est défini et caractérisé par un ensemble de mesures.
- Dimension : est un axe d'analyse selon lequel on analyse un fait. Les dimensions décrivent la sémantique des mesures et le contexte d'analyse du fait [8].
- Paramètre : Un paramètre est un attribut appartenant à une dimension. Il représente un niveau de détail selon lequel sont visualisées les mesures d'activité d'un sujet d'analyse[6].
- Hiérarchie : Une hiérarchie est une perspective d'analyse définie dans une dimension. Elle regroupe un ensemble de paramètres organisés de la granularité la plus fine vers la granularité la plus générale[9].

1.4.3 Les limites des entrepôts de données traditionnels

Les systèmes d'information décisionnels s'appuient sur l'entreposage de données et l'analyse en ligne qui sont deux piliers de l'informatique décisionnelle, aujourd'hui confrontés à de nouveaux défis scientifiques. De nouvelles sources d'informations hétérogènes, fortement évolutives, changeantes, dépendantes ou autonomes et distribuées apparaissent. Avec cette évolution, les entrepôts de données traditionnels sont confrontés à des limites qui sont au niveau[2] :

• Nature des données : la nécessité d'intégrer le Web à l'entreprise pour compléter les données de celle-ci avec celle du monde extérieur, a engendré le besoin de comprendre de nouvelles sources de données de nature semi-structurées, comme les fichiers HTML ou XML. L'intégration de ces données, revendique la prise en compte de modèles de données riches comme l'objet, le développement d'outils interactifs d'intégration de schémas hétérogènes, la prise en compte de requête complexes, et l'optimisation de requêtes distribuées[2].

• Disponibilité : représente un défi pour l'entrepôt, à la fois en raison du processus de chargement et de l'infrastructure dans son ensemble, les projections d'un ensemble croissant de données, cycles de conservation des données et le temps de réponses des requêtes associées. La non-disponibilité des données en temps opportun influence sur l'utilisation et l'adoption de la prise de décision au sein de l'entreprise.

• Stockage des données : le fait que les données, les index et les tables temporaires travaillent tous avec le même ensemble de disques et les contrôleurs au niveau de l'entrepôt de données, cela crée une énorme pénalité à la fois sur la disponibilité et la performance du système, et avec l'ajout de charge de travail des requêtes complexées sur l'architecture de stockage, à qui engendre un temps d'accès étendus des disques et des données, ce qui provoque un manque de performance.

• Performance des requêtes : C'est un autre domaine de performance qui remet en question l'entrepôt de données traditionnel. Les requêtes ad-hoc et des requêtes analytiques causent le plus d'impact sur la performance globale des requêtes, l'accès et le traitement des données, y compris le déplacer à travers le réseau en raison de leur nature non déterministe. Les requêtes peuvent nécessiter un large ensemble de données qui a besoin d'accéder à plusieurs zones de stockage ou d'un grand ensemble de données qui peut être consulté dans une zone de stockage plus petite.
• Transport de données : Problèmes au niveau de la surcharge lors du transport des données à partir d'une couche à l'autre et de la disponibilité des données ce qui ralentit le processus de traitement ultérieur.

2. Modélisation multidimensionnelle de l'entrepôt de données

Concevoir un système décisionnel nécessite une phase de modélisation des données multidimensionnelles. Plusieurs approches ont été proposées selon trois niveaux d'abstraction conceptuelle, logique et physique (voir la figure I.2).

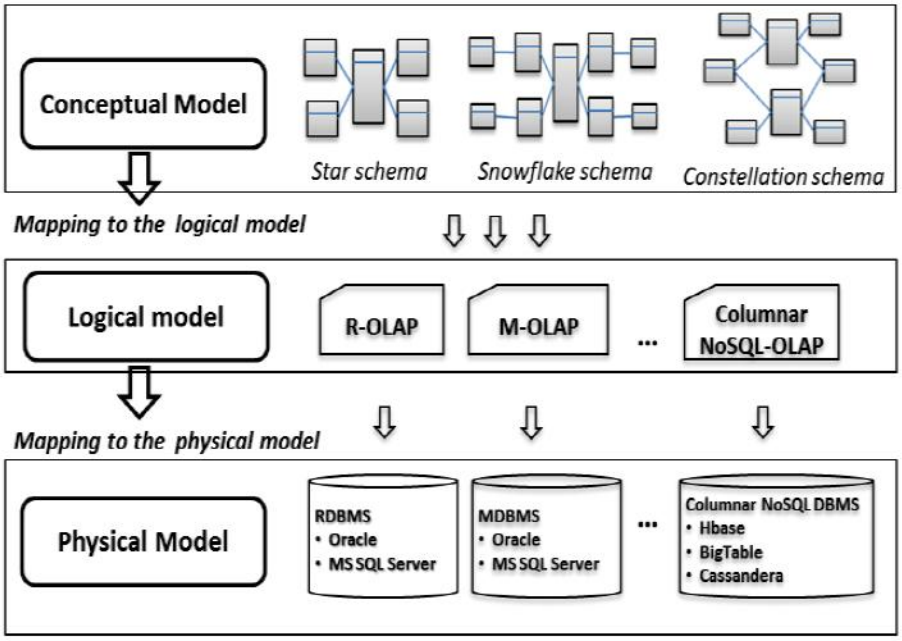

Figure I.2 : Processus de mise en œuvre pour les entrepôts de données[10]

Comme le montre la figure I.2, le modèle logique vise à réorganiser les données en fonction de l'architecture de stockage la plus appropriée pour une meilleure prise en charge par le SGBD. Le modèle logique est situé entre les modèles conceptuels et physiques de données. En d'autres termes, non seulement il donne plus de détails que le modèle conceptuel sur la structuration des données et leurs relations, mais il prépare le passage au niveau physique aussi bien ; ce qui rend le modèle le plus décisif dans le processus de modélisation.

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

2.1 Modélisation conceptuelle

Les données multidimensionnelles sont mises en œuvre sous forme de tables relationnelles, tables de faits et de dimensions. La table de fait est constituée d'attributs représentant les mesures d'activité et les attributs clés étrangères qui la relient vers chacune des tables de dimension associée. Les tables de dimension contiennent les paramètres et une clé primaire permettant de réaliser des jointures avec la table de fait [11]. Ces tables sont organisées dans des structures appelées : schéma en étoile, schéma en flocon et schéma en constellation qui peuvent être décrits de la manière suivante :

2.1.1 Schéma en étoile

Est un schéma relationnel dans lequel une table centrale contenant les faits analysés référence des tables de dimensions, chaque dimension étant décrite par une seule table dont les attributs représentent les divers niveaux de granularité [12]

Les avantages :

- Facilité de navigation
- Nombre de jointures limité

Les inconvénients :

- Redondance dans les dimensions (ne sont pas normalisées en 3 FN)
- Toutes les dimensions empaquetées dans une même table ayant un niveau d'agrégation supérieur ne correspondent pas sémantiquement aux mesures, qui, elles sont calculées à un niveau de granularité fin

2.1.2 Schéma en flocon de neige

Est un schéma relationnel dans lequel une table centrale contenant les faits analysés référence des dimensions du premier niveau, chaque dimension étant décrite par une succession de tables représentant les diverses granularités, chaînées par des contraintes référentielles [12], ce qui donne naissance à des hiérarchies au sein des dimensions.

Les avantages :

- Normalisation des dimensions
- Economie d'espace disque (réduction du volume)
- permettre des analyses par pallier (drill down et roll-up)

Les inconvénients :

- Modèle plus complexe (nombreuses jointures)
- Navigation difficile
- Requêtes moins performantes

2.1.3 Schéma en constellation

Ce schéma est une extension du schéma en étoile. Il consiste à fusionner plusieurs schémas en étoile qui utilisent des dimensions communes. Un schéma en constellation comprend donc plusieurs faits reliés à un ensemble de dimensions qui peuvent être partagées.

Ce schéma présente l'avantage de pouvoir corréler les sujets d'analyse tels que la comparaison des montants des locations réalisées dans les différentes agences par rapport aux chiffres d'affaires

réalisés par son personnel. En outre, le partage des dimensions par plusieurs faits permet d'éviter de les définir plusieurs fois [13].

2.2 Modélisation logique

Au niveau logique, plusieurs possibilités sont envisageables pour la modélisation multidimensionnelle. En effet, il est possible d'utiliser soit, un système de gestion de bases de données (SGBD) existant tel que les SGBD relationnels et dans ce cas l'environnement OLAP utilisé est le ROLAP ; soit un système de gestion de bases de données multidimensionnelles (MOLAP) qui stocke les données des faits et des dimensions de façon multidimensionnelle en natif. Enfin, un système qui utilise en même temps l'approche ROLAP (pour les dimension) et MOLAP (pour les faits) s'appelle modèle hybride (HOLAP) [14].

2.2.1 ROLAP (Relational OLAP)

Cette méthodologie repose sur la manipulation des données stockées dans la base de données relationnelle pour donner l'apparence des fonctionnalités de découpage et de découpage d'OLAP traditionnelles. En substance, chaque action de découpage en tranches équivaut à l'ajout d'une clause "*WHERE*" dans l'instruction SQL[15], voir la figure I.3.

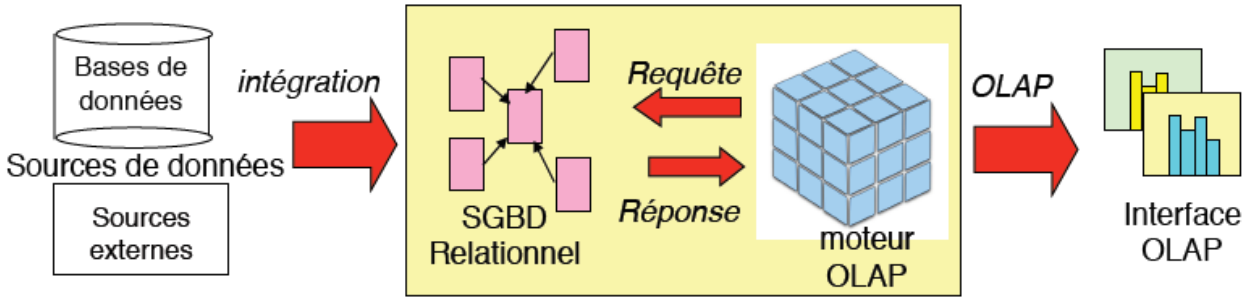

Figure I.3 : Représentation de modèle ROLAP[16]

2.2.2 MOLAP (Multidimensional OLAP)

C'est la méthode la plus performante pour l'analyse OLAP. Dans MOLAP, les données sont stockées, en natif, dans un cube multidimensionnel. Le stockage ne se trouve pas dans la base de données relationnelle, mais dans des formats propriétaires [15] ,voir la figure I.4.

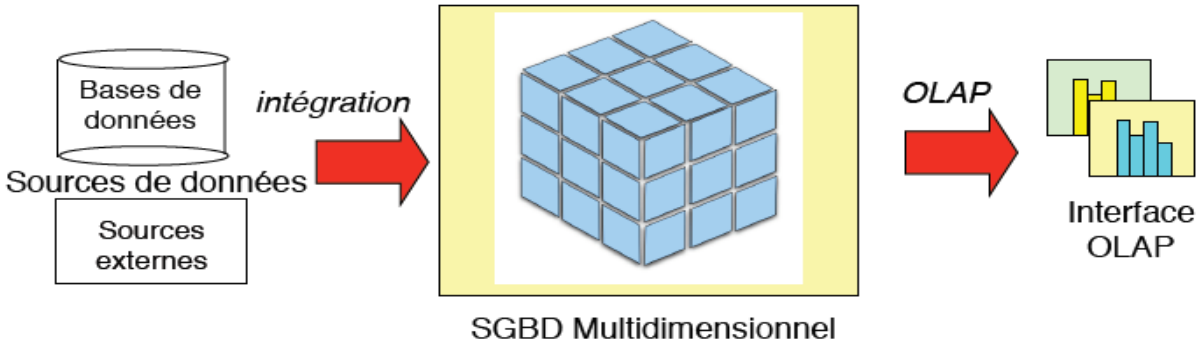

Figure I.4 : Représentation de modèle MOLAP[16]

2.2.3 HOLAP (Hybrid OLAP)

Les technologies HOLAP tentent de combiner les avantages de MOLAP et de ROLAP. Pour des informations de type récapitulatif, HOLAP utilise la technologie de cube pour des performances plus rapides. Lorsque des informations détaillées sont nécessaires, HOLAP peut "accéder au détail" du cube dans les données relationnelles sous-jacentes[15], voir la figure I.5.

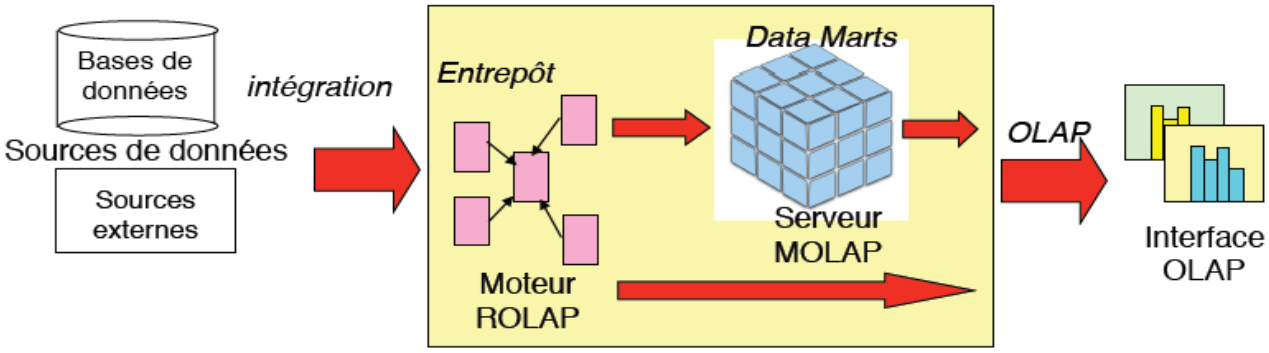

Figure I.5 : Représentation de modèle HOLAP [17]

3. Les méga données

On assiste aujourd'hui à une explosion du volume de données, conséquence de l'usage excessif de l'Internet et du Web, à travers des réseaux sociaux, des appareils mobiles et des objets connectés etc. Ce déluge de données s'est accompagné d'une diversité de formats allant parfois à l'absence de structures ce qui a rendu inutilisables les méthodes traditionnelles de stockage nécessitant un schéma unifié comme les SGBDR.

3.1 Définition et caractéristiques

Littéralement, BigData signifie données massives ou méga données. C'est un ensemble d'entités de données hétérogènes en extensibilité permanente qui ne peuvent pas être pris en charge par les systèmes de gestion de données classiques. BigData est aussi une architecture distribuée et scalable pour le traitement et le stockage de grands volumes de données. En effet, on crée environ

2,5 milliards de Giga octets de données tous les jours, émanant des différents domaines créés par les divers outils numériques : vidéos publiés, messages envoyés, signaux GPS, enregistrements transactionnels d'achats en ligne et bien d'autres encore. Ces volumes massifs de données sont baptisés BigData. Les géants du Web, au premier rang comme Yahoo, Facebook, Amazon et Google, ont été les tous premiers à déployer ce type de technologie pour permettre à tout le monde d'accéder en temps réel à leurs bases de données géantes[18].

Depuis l'apparition de cette terminologie, beaucoup de bruit a été rouspété autour du BigData, mais selon le Gartner, ce concept regroupe une famille d'outils qui a une triple problématique dite règle des « 3V » : Volume, Vélocité et Variété ensuite étendue pour couvrir la « Véracité » et la « Valeur » devenant ainsi les « 5V » du BigData [19]. Nous présenterons dans ce qui suit ces caractéristiques plus en détail.

-Volume : Le BigData est associé à un volume de données vertigineux, se situant actuellement entre quelques dizaines de téraoctets (1 To=2 12 octets) et plusieurs pétaoctets (1 Po=2 15 octets) en un seul jeu de données. Le volume correspond à la masse d'informations produite chaque seconde. Les entreprises issues de tous les secteurs d'activité gérant des données massives, se voient assujetties à trouver des techniques nécessaires et moyens capables pour gérer les volumes de données collectés chaque jour et d'une importance vitale pour leur survie[18].

-Vélocité : La vélocité ou la vitesse d'échanges décrit la fréquence à laquelle les informations sont générées, capturées, stockées et partagées. Elle est aussi le traitement des flux continus de données. Les entreprises doivent appréhender la vitesse non seulement en termes de création de données, mais aussi sur le plan de leur traitement, de leur analyse et de leur restitution à l'utilisateur en respectant les exigences des applications en temps réel[18].

-Variété : De plus en plus, le taux des données structurées manipulées dans des tables de bases de données relationnelles est en décroissance par rapport à l'expansion des types de données non structurées. Cela peut être des images, des vidéos, des messages, des voix, et bien d'autres encore. Aujourd'hui, on trouve plusieurs milliers de sources hétérogènes comme les capteurs d'informations aussi bien dans les trains, les automobiles, les avions ou les équipements électroménagers qui émettent une variété d'informations de tout genre. Les technologies BigData, permettent de faire de la création, l'intégration, l'analyse, la reconnaissance, le classement des données de différents types comme des photos sur différents sites ou les messages échangés sur les réseaux sociaux, etc. Ce sont les différents éléments qui constituent la variété supportée par le BigData[18].

-Véracité : L'aptitude à juger la crédibilité et la fiabilité du nombre indéfini de données collectées qualifie la Véracité du BigData. Il est difficile de justifier l'authenticité et l'exactitude des contenus des différents volumes et variétés de données manipulées comme dans les conversations dans les réseaux sociaux avec les abréviations, le langage familier, les coquilles, les hashtags. La vérité est que le BigData, a bien trois significations et que les éditeurs n'en abordent qu'une à la fois. Il est important de connaître leur positionnement pour leur poser les bonnes questions[18].

- Valeur Toutes ces données massives ressemblent à une mine d'or potentielle, mais comme dans une mine d'or, vous avez seulement peu d'or et beaucoup plus de tout le reste. Parmi les défis les très

sensibles de cette technologie est comment donner du sens à ces données pour tirer celles qui sont les plus pertinentes pour d'éventuelles prises de décisions dans une entreprise ? Comment une organisation traite-t-elle des quantités massives de manière significative ? La notion de Valeur correspond à l'intérêt qu'on puisse tirer de l'utilisation de cette technologie. Selon les experts du domaine, les entreprises qui ne s'intéressent pas sérieusement au contenu de leurs volumes de données hébergées risquent d'être pénalisées et dépassées. BigData désigne à la fois les grands volumes de données et la difficulté à extraire de cette masse de données celles ayant suffisamment de valeur pour justifier leur analyse. BigData offre un ensemble d'outils d'analyse de données qui peuvent servir à préserver un privilège concurrentiel[18].

3.2 Les enjeux technologiques

Pour beaucoup d'entreprises, le BigData représente de nouveaux enjeux qu'il faut envisager, mais il faut aussi étudier les risques induits.
D'après Michael Stonebraker « " Il y a beaucoup de bruit au tour du BigData .Ce concept a plusieurs significations en fonction du type de personnes. Nous pensons que la meilleure façon de considérer le BigData est de penser au concept de trois V. BigData peut être synonyme de gros volume, du téraoctet au pétaoctet. Il peut également signifier la rapidité (Velocity) de traitement de flux continus de données. Enfin la troisième signification : vous avez à manipuler une grande variété de données, de sources hétérogène. Vous avez à intégrer entre mille et deux mille sources de données différentes et l'opération est un calvaire. La vérité est que le BigData a bien trois significations et que les éditeurs n'en abordent qu'une à la fois. Il est important de connaitre leur positionnement pour leur poser les bonnes questions. "»[11]

3.3 Les limites du modèle relationnel face aux méga-données

Le modèle relationnel a été développé comme une technologie de stockage des données structurées et organisées sous forme de tableaux. Aux cours des années il est devenu l'élément essentiel des organisations et le modèle de référence pour la gestion des données des systèmes d'informations, cependant, avec l'augmentation continue des données stockées et analysées, les bases de données relationnelles commencent à présenter une variété de limitations.

Les bases de données relationnelles, étroitement associées au langage SQL, comportent un certain nombre de règles d'organisation. Ces règles largement utilisés garantissent le maintien des propriétés *ACID* (Atomicité, Cohérence, Isolation et Durabilité) .le model relationnel est basé sur une forme normalisée des données, pour garantir la robustesse du schéma relationnel, minimiser la redondance des données et de sécurité incontournables. Cependant toutes ces règles et outils s'avèrent être un véritable obstacle pour le déploiement de bases de données de grande envergure et redondantes, comme le nécessite le BigData, stockage et analyse. En matière de stockage, il existe des nouvelles approches, nous retrouvons Teradata, SQL Server, Vertica, Greenplum, ParAccel et Netezza1. Seulement, elles sont difficiles à gérer, coûteuses, souvent incompatibles avec les données semi ou non structurées et n'assurent pas la tolérance aux pannes pour les requêtes à longue durée

(Sakr, 2016). D'un autre coté, leur conformité avec la règle ACID devient un handicap puisqu'elle inclut des traitements additionnels pour la maintenir.
Cela dit que les bases de données relationnelles ne peuvent pas gérer un très grand volume de données (de l'ordre du péta-octet) et impossible de gérer des débits extrêmes (plus que quelques milliers de requêtes par seconde). Aussi, elles sont parfois peu adaptées au stockage et à l'interrogation de certains types de données (données hiérarchiques, faiblement structurées, semi-structurées).Donc ces approches classiques ont des performances limitées peu adapte à la gestion du BigData[20].
C'est dans ce contexte que les bases de données NoSQL étaient développées pour fournir un ensemble de nouvelles fonctionnalités de gestion des données tout en surmontant certaines limites de bases de données relationnelles. Ces nouvelles techniques permettent le stockage selon des modèles de données différents (documents, colonnes, graphes), introduisant une plus grande flexibilité au niveau des schémas. En termes de besoins d'évaluation, nous faisons face à une diversité de solutions à supporter.

3.4 Paradigmes et technologies des BigData

3.4.1 Paradigme MapReduce

Le traitement de données de façon distribuée soulève certaines questions, comment distribuer le travail entre les serveurs ? Comment synchroniser les différents résultats ? Comment gérer une panne d'une unité de traitement ? MapReduce répond à ces problèmes.

Il ne s'agit pas d'un élément de base de données, mais d'un modèle de programmation s'inspirant des langages fonctionnels et plus précisément du langage Lisp. Il permet de traiter une grande quantité de données de manière parallèle, en les distribuant sur divers nœuds d'un cluster. Ce mécanisme a été mis en avant par Google en 2004 et a connu un très grand succès auprès des sociétés utilisant des *DataCenters* telles que Facebook ou Amazon[11][21].

3.4.1.1 Principe de MapReduce

Le principe de MapReduce est simple: il s'agit de découper une tâche manipulant un gros volume de données en plusieurs tâches traitant chacune un sous-ensemble de ces données. MapReduce est vu en deux étapes (voir la figure I.6). La première étape, appelé « map » consiste à dispatcher une tâche, ainsi que les données quelle traite en plusieurs sous-tâches traitant chacune d'elles un sous-ensemble de données. La deuxième étape se nomme « Reduce ». Dans cette partie il s'agit de récupérer les résultats des différents serveurs (à savoir les sous-tâches) et de les consolider. Nous reviendrons plus en détail sur le sujet plus bas. Bien que le principe soit relativement simple, il l'est surtout grâce à l'apport du modèle MapReduce simplifiant au maximum les complexités de l'informatique distribuée. Il permet ainsi aux développeurs de se focaliser sur le traitement proprement dit. Le fait que MapReduce soit développé en java permet de s'abstraire de l'architecture matérielle pour qu'un framework MapReduce puisse tourner sur un ensemble de machines hétérogènes[22][16].

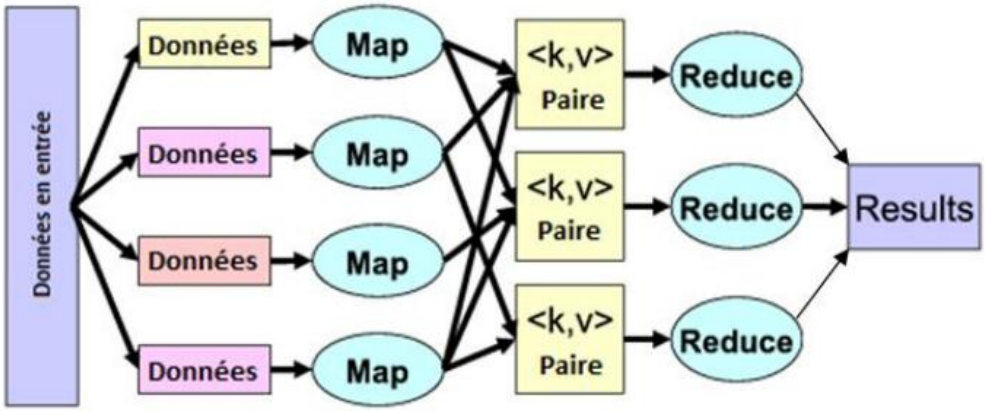

Figure I.6 : Schéma de MapReduce [11]

Comme dit précédemment, le modèle MapReduce consiste en deux étapes, représentées toutes deux par des fonctions. Dans la fonction Map on prend en entrée un ensemble de « clé-valeurs », le nœud fait une analyse du problème et il va le séparer en sous-tâches, pour pouvoir les redistribuer sur les autres nœuds du cluster. Dans le cas nécessaire, les nœuds recevant les sous-tâches refont le même processus de manière récursive. Les sous-tâches des différents nœuds sont traitées chacune d'entre elles dans leur fonction Map respective et vont retourner un résultat intermédiaire. La deuxième étape nommé « Reduce », consiste à faire remonter tous les résultats à leur nœud parents respectif. Les résultats se remonte donc du nœud le plus bas jusqu'à la racine. Avec la fonction Reduce, chaque nœud va calculer un résultat partiel en associant toutes les valeurs correspondant à la même clé en un unique couple (clé – valeur). Une fois obtenu ce couple (clé-valeur), le résultat est remonté au nœud parent, qui va refaire le même processus de manière récursive jusqu'au nœud racine. Quand un nœud termine le traitement d'une tâche, un nouveau bloc de données est attribué à une tâche Map.

3.4.2 L'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).

3.4.2.1 Hadoop Distributed File System (HDFS)

HDFS est le système de fichiers par défaut utilisé par Hadoop pour découper les données et répliquer ces pièces de données en triple sur plusieurs nœuds. Ce système dispose de deux modes d'implémentations [23]:

Distribué : permettant la triple réplication.

Pseudo-distribué : utilisé comme moyen de test, ce mode est implémenté sur une seule machine d'un seul nœud.

Dans Hadoop, les différents types de données, qu'elles soient structurées ou non, sont stockées à l'aide du HDFS. Le HDFS va prendre les données en entrée et va ensuite les partitionner en plusieurs blocs de données. Afin de garantir une disponibilité des données en cas de panne d'un nœud, le système fera un réplica des données. Par défaut les données sont répliquées sur trois nœuds différents (voir la figure I.7), deux sur le même support et un sur un support différent. Les différents nœuds de données peuvent communiquer entre eux pour rééquilibrer les données[24].

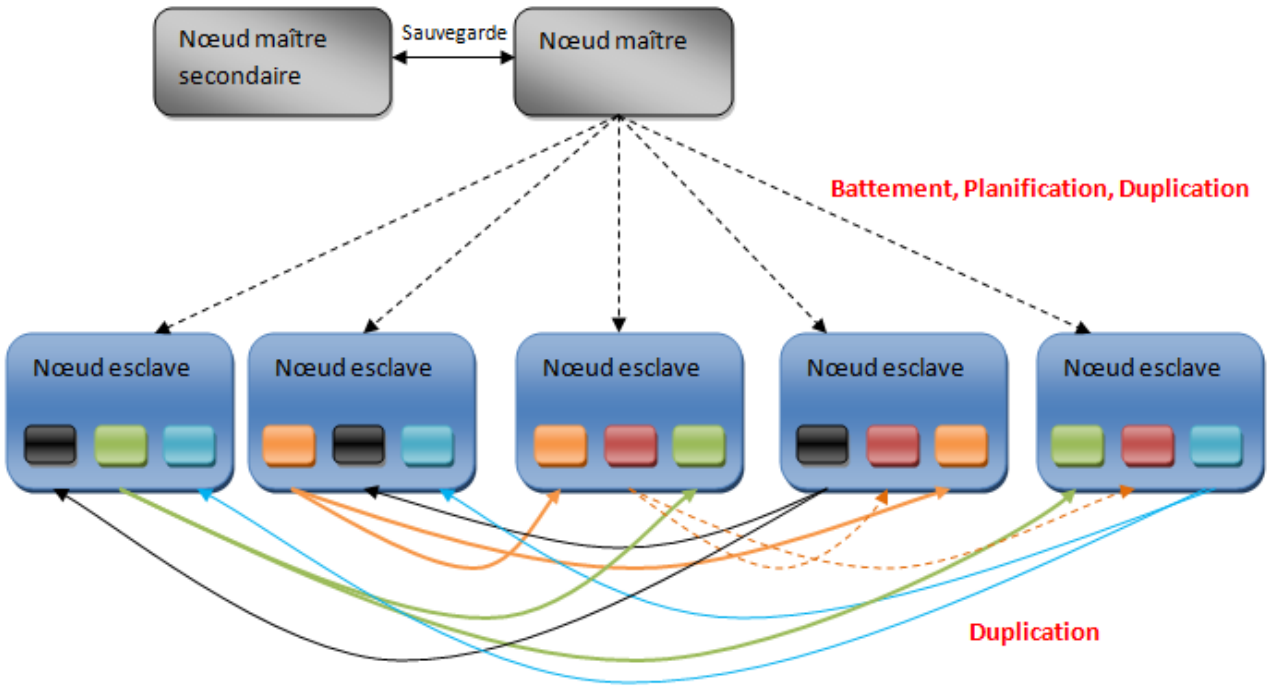

Figure I.7 : Schéma de principe du HDFS[2]

Hadoop repose sur un schéma dit « maître-esclave » (figure I.8) et peut être décomposé en cinq éléments[11].

Le nom du nœud (*Name Node*) : Le « *Name Node* » est la pièce centrale dans le HDFS, il maintient une arborescence de tous les fichiers du système et gère l'espace de nommage. Il centralise la localisation des blocs de données répartis sur le système. Sans le « *Name Node* », les données peuvent être considérées comme perdues car il s'occupe de reconstituer un fichier à partir des différents blocs répartis dans les différents « *Data Node* ». Il n'y a qu'un « *Name Node* » par cluster HDFS.

-Le gestionnaire de tâches (*Job Tracker*) : Il s'occupe de la coordination des tâches sur les différents clusters. Il attribue les fonctions de MapReduce aux différents « *Task Trackers* ». Le « *Job Tracker* » est un « Daemon » cohabitant avec le « *Name Node* » et ne possède donc qu'une instance par cluster.

[2] https://fr.wikipedia.org/wiki/Hadoop#/media/Fichier:HDFS.png

-Le moniteur de tâches (*Task tracker*): Il permet l'exécution des ordres de MapReduce, ainsi que la lecture des blocs de données en accédant aux différents « *Data Nodes* ». Par ailleurs, le « *Task Tracker* » notifie de façon périodique au « *Job Tracker* » le niveau de progression des tâches qu'il exécute, ou alors d'éventuelles erreurs pour que celui-ci puisse reprogrammer et assigner une nouvelle tâche. Un « *Task Tracker* » est un « Deamon » cohabitant avec un « *Data Node* », il y a un donc un « *Task Tracker* » par « *Data Node* ».

-Le nœud secondaire *(Secondary node)* : N'étant initialement pas présent dans l'architecture Hadoop, celui-ci a été ajouté par la suite afin de répondre au problème du point individuel de défaillance (*SPOF- Single point of failure*). Le « *Secondary Node* » va donc périodiquement faire une copie des données du « *Name Node* » afin de pouvoir prendre la relève en cas de panne de ce dernier.

-Le nœud de données (*Data Node*) : Il permet le stockage des blocs de données. Il communique périodiquement au « *Name Node* » une liste des blocs qu'il gère. Un HDFS contient plusieurs nœuds de données ainsi que des réplications d'entre eux. Ce sont les nœuds esclaves.

Figure I.8 : Architecture d'un cluster Hadoop[11]

Un cluster Hadoop peut-être constitué de machines hétérogènes, que ce soit au niveau du hardware comme au niveau software (système d'exploitation). Cependant il est bien plus simple d'administrer un cluster de type homogène.

3.4.3 Apache Spark

Apache Spark est un Framework Open Source de traitements BigData construit à la base de Hadoop MapReduce pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. Celui-ci a originellement été développé par l'Université UC Berkeley en 2009 et passé Open Source sous forme de projet Apache en 2010, Spark travaille avec plusieurs langages comme Java ou Python mais Spark devient vraiment intéressant avec son langage natif Scala[25][5].

Comme illustre la figure I.9 à côté des API principales de Spark, l'écosystème contient des librairies additionnelles qui permettent de travailler dans le domaine des analyses BigData et de l'apprentissage automatique. Parmi ces librairies, on trouve Streaming, SQL, MLlib et GraphX[9].

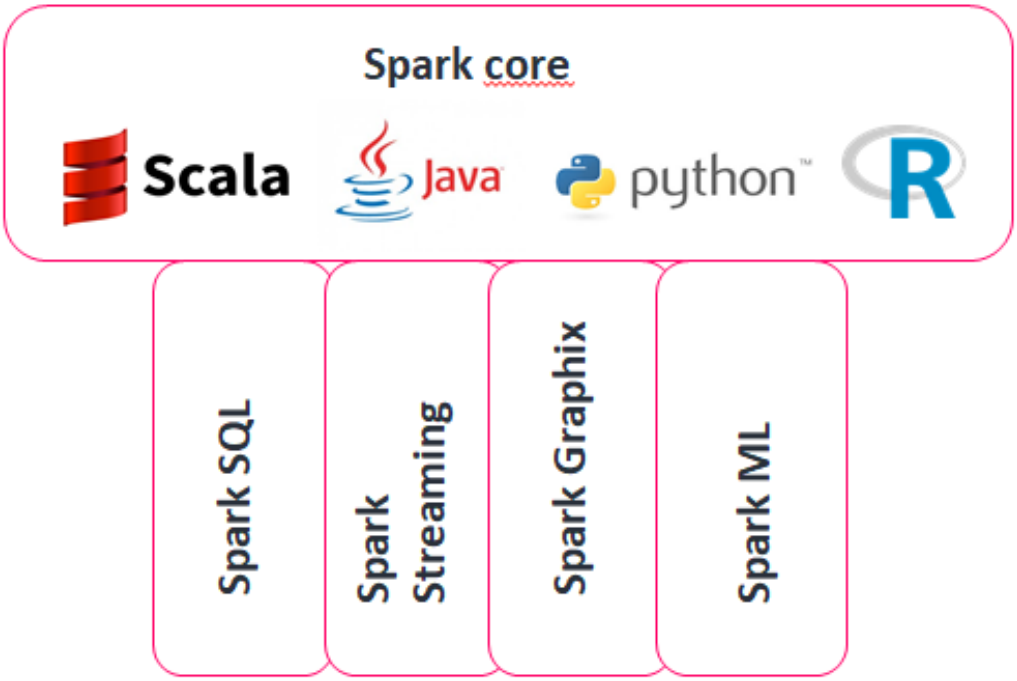

Figure I.9 : L'écosystème Spark[3]

3.4.3.1 Spark Streaming

Il peut être utilisé pour des traitements de flux en temps-réel. Il s'appuie sur un mode de traitement en micro-Batch et utilise une abstraction permettant de réutiliser efficacement les données dans une large famille d'applications appelée *RDD* (*Resilient Distributed Dataset*). Les RDD sont tolérants à la panne et proposent des structures de données parallèles qui permettent aux utilisateurs de :

- Persister explicitement les données intermédiaires en mémoire.
- Contrôler leur partitionnement afin d'optimiser l'emplacement des données.
- Manipuler les données, en utilisant un ensemble important d'opérateurs.

[3] https://spark.apache.org

3.4.3.2 Spark SQL

Elle permet d'exposer les jeux de données Spark, via une API de type JDBC et d'exécuter des requêtes de type SQL, en utilisant les outils BI et de visualisation traditionnels. Spark SQL permet d'extraire, de transformer et de charger des données sous différents formats et les exposer pour des requêtes ad hoc.

Lors de l'exécution de SQL à partir d'un langage de programmation, les résultats sont renvoyés sous la forme d'un ensemble de données / *DataFrame* qui est un jeu de données organisé en colonnes nommées. Il est conceptuellement équivalent à une table dans une base de données relationnelle ou à un bloc de données dans R / Python, mais avec des optimisations plus riches sous le capot. Les *DataFrames* peuvent être construits à partir d'un large éventail de sources telles que: des fichiers de données structurés, des tables dans Hive, des bases de données externes ou des RDD existants.

3.4.3.3 Spark MLlib

MLlib est une librairie d'apprentissage automatique, qui contient tous les algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le *Clustering*, le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous-jacentes.

3.4.3.4 Spark GraphX

GraphX est la nouvelle API encore en version alpha, pour les traitements de graphes et de parallélisations de graphes. GraphX étend les RDD de Spark en introduisant le *Resilient Distributed Dataset Graph*, un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arcs. Pour le support de ces traitements, GraphX expose un jeu d'opérateurs de base, tels que *Subgraph, JoinVertices et AggregateMessages*. De plus, GraphX inclut une collection toujours plus importante d'algorithmes permettant de simplifier les tâches d'analyse de graphes.

Un des points central de Spark est son utilisation des *RDDs* (*Resilient Distributed Datasets*).

Un RDD est une collection partitionnée d'enregistrements en lecture seule qui ne peut être créée que par des opérations déterministes; soit à partir de données présentes dans un stockage stable ou à partir d'autres RDDs.

Résumons. Les RDD possèdent deux types de méthodes :

- Les transformations qui donnent en sortie un autre RDD.
- Les actions qui donnent en sortie autre chose qu'un RDD.

Dans une application Spark, les transformations et les actions réalisées sur les RDD permettent de construire un graphe acyclique orienté *(DAG : "Directed Acyclic Graph")* (la figure I.10).

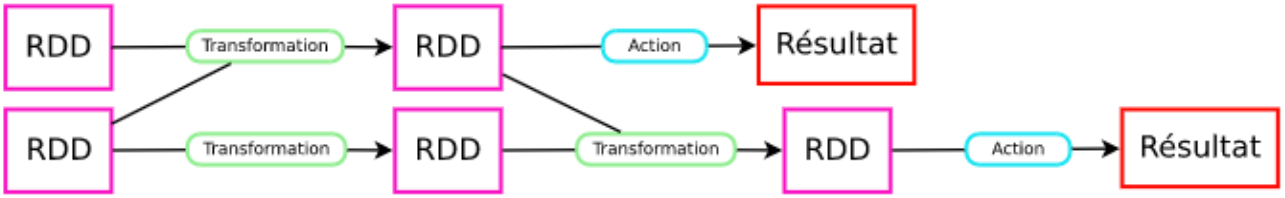

Figure I.10 : un graphe acyclique orienté[4]

Spark présente plusieurs avantages par rapport aux autres technologies comme Hadoop et Storm. D'abord, Spark propose un Framework complet et unifié pour répondre aux besoins de traitements BigData pour divers jeux de données, différents par leur nature (texte, graphe) aussi bien que par le type de source (en mode Batch ou en temps-réel). Ensuite, Spark permet à des applications sur des Clusters Hadoop d'être exécutées jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur le disque. Il permet d'écrire rapidement des applications en Java, Scala ou Python et inclut un jeu de plus de 80 opérateurs haut-niveau. De plus, il est possible de l'utiliser de façon interactive pour requêter les données depuis une fenêtre de commande Shell[9].

Dans cette partie on a vu les caractéristiques d'Apache Spark, Spark se positionne par rapport aux implémentations MapReduce traditionnelles comme Apache Hadoop. Il s'appuie sur le même système de stockage de fichiers qu'Hadoop, il est donc possible d'utiliser Spark et Hadoop ensemble dans le cas où des investissements significatifs ont déjà été faits avec Hadoop.

Il est possible également de combiner les types de traitements Spark avec Spark SQL, Spark Machine Learning et Spark Streaming, grâce à différents modes d'intégration et adaptateurs Spark. Cependant, Spark n'est pas un écosystème encore complètement mature. Il nécessite encore des améliorations dans certains domaines comme la sécurité ou l'intégration avec des outils d'analyse décisionnelle[5].

3.5 Les modèles de données NoSQL

Le stockage des méga données nécessite un partitionnement. Or selon le théorème de Brewer (ou théorème de *CAP* pour *Consistency*, *Availability*, *Partition tolerance*), la gestion de ces méga données partitionnées, est nécessairement supportée par un système distribué ne pouvant assurer simultanément la cohérence et la disponibilité des données, propriétés qu'assurent les systèmes relationnels (propriétés ACID). Aussi les systèmes de gestion de méga données devront faire le choix entre cohérence et disponibilité. La disponibilité est alors généralement privilégiée dans l'exploitation de ces méga données. De plus, les systèmes relationnels imposent une structuration des données selon des schémas spécifiques, or les méga données sont en grande partie peu ou pas structurées [26]. En conséquence, de nouveaux modèles de stockage de données, mieux adaptés aux méga données que le modèle relationnel, ont vu le jour, et ont conduit à l'émergence des bases de données NoSQL. Notons que le terme NoSQL, proposé par Carl Strozzi, ne signifie pas le rejet du modèle relationnel (SQL), mais doit être interprété comme « *Not Only SQL* ». Ces modèles ne remplacent pas les BD relationnelles mais sont une alternative, un complément apportant des solutions plus intéressantes dans certains contextes[27].

[4] https://openclassrooms.com/fr/courses/4297166-realisez-des-calculs-distribues-sur-des-donnees-massives/4308666-prenez-spark-en-main

Ces systèmes NoSQL permettent une gestion d'objets complexes et hétérogènes sans avoir à déclarer au préalable l'ensemble des champs représentant un objet. Ils adoptent une représentation de données non relationnelles, sans schéma pour les données, ou avec des schémas dynamiques, et concernent des données de structures complexes ou imbriquées[28].

Les bases de données NoSQL sont donc une catégorie de base de données qui n'est plus fondée sur l'architecture classique des bases relationnelles, et non pas un type à part entière. Il existe différentes manières de structurer les données mais l'ensemble des solutions développées peut être devisé en quatre model que nous décrivons dans ce qui suit :

3.5.1 Modèle orienté « clé-valeur »

Dans ce modèle (figure I.11), les données sont stockées sous la forme de grandes tables de hashage distribuées, permettant de facilement passer à l'échelle. Les données sont simplement représentées par un couple clé-valeur. La valeur peut être une simple chaîne de caractères, un objet sérialisé… Cette absence de structure et de typage a un impact important sur le requêtage.

Ainsi la complexité d'une requête qui sera dans un système relationnel prise en charge par le langage SQL, sera dans ce modèle NoSQL prise en charge par l'applicatif interrogeant la base de données, la communication avec la base se limitant généralement à des ordres *PUT, GET et DELETE*[3].

Les systèmes NoSQL orientés clé-valeur les plus connus sont Memcached, Amazon's Dynamo, Redis, Riak et Voldemort créé par LinkedIn.

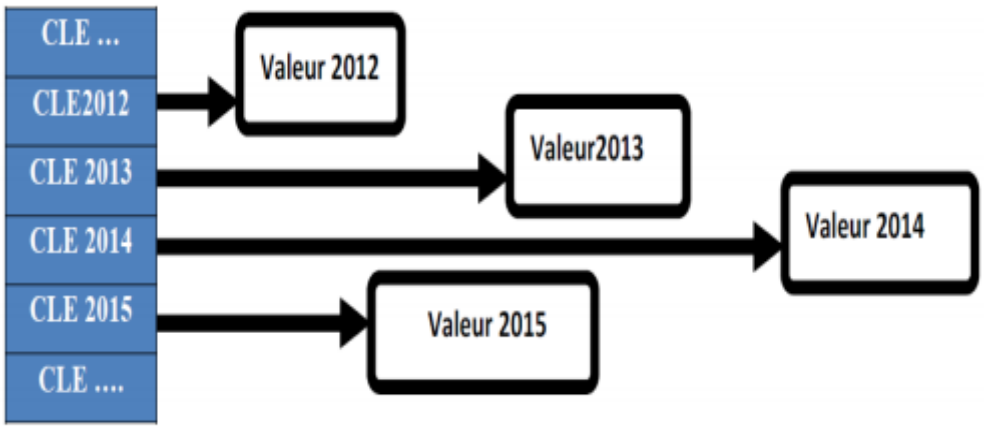

Figure I.11 : Modèle NoSQL[29]

3.5.2 Modèle orienté « documents »

Ce modèle (figure I.12) se base sur le paradigme clé-valeur précédent ; cependant dans ce nouveau modèle la valeur est un document de type JSON ou XML. Ainsi, dans ce modèle, les données sont stockées à l'intérieur de documents. Un document peut être vu comme un n-uplet d'une table dans le monde relationnel, à la différence toutefois que les documents peuvent avoir une structure complètement différente les uns des autres. L'avantage est de pouvoir récupérer, via une seule clé, un ensemble d'informations structurées de manière hiérarchique. La même opération dans le monde relationnel impliquerait plusieurs jointures. Les systèmes NoSQL orientés documents les plus connus sont CouchDB d'Apache, RavenDB (destiné aux plateformes .NET/Windows avec la possibilité d'interrogation via LINQ) et MongoDB[30].

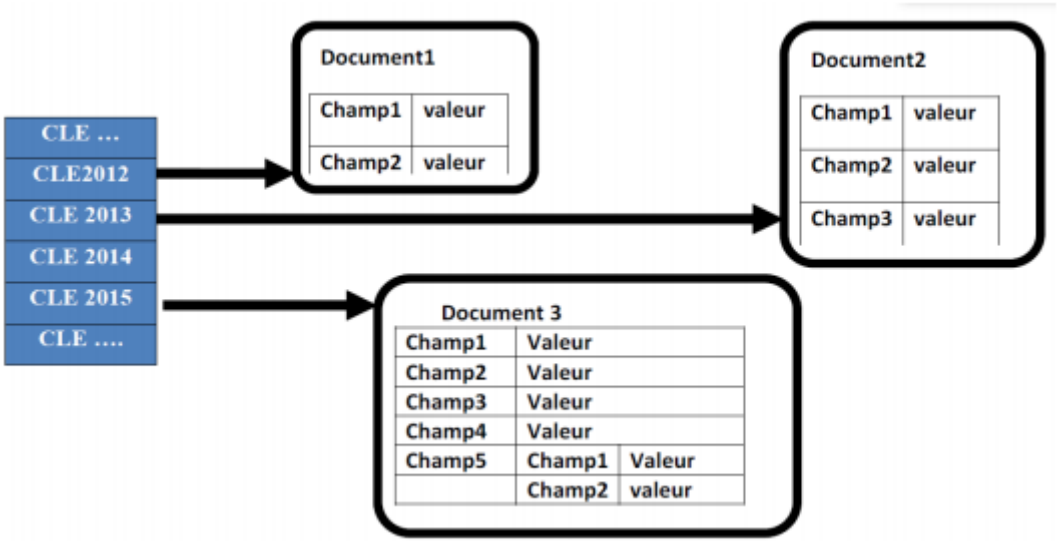

Figure I.12 : Modèle NoSQL « document »[29]

3.5.3 Modèle orienté « colonnes »

Ce modèle (figure I.13) ressemble à première vue à une table du modèle relationnel, du fait que les attributs sont regroupés en famille de colonnes. Ainsi, deux attributs qui sont fréquemment interrogés ensemble seront stockés au sein d'une même famille de colonnes. Cependant la différence est que, dans cette base NoSQL orientée colonnes, le nombre de colonnes est dynamique, alors que dans une table relationnelle, le nombre de colonnes est fixé dès la création du schéma de la table. De plus, dans ce modèle, contrairement au modèle relationnel, le nombre de colonnes peut varier d'un enregistrement à un autre, ce qui évite de retrouver des colonnes ayant des valeurs inconnues (*Null Value*).Les systèmes NoSQL orientés colonnes les plus connus sont principalement HBase, implémentation Open Source du modèle BigTable développé par Google, et Cassandra, projet Apache qui respecte l'architecture distribuée de Dynamo d'Amazon, et le modèle BigTable de Google.

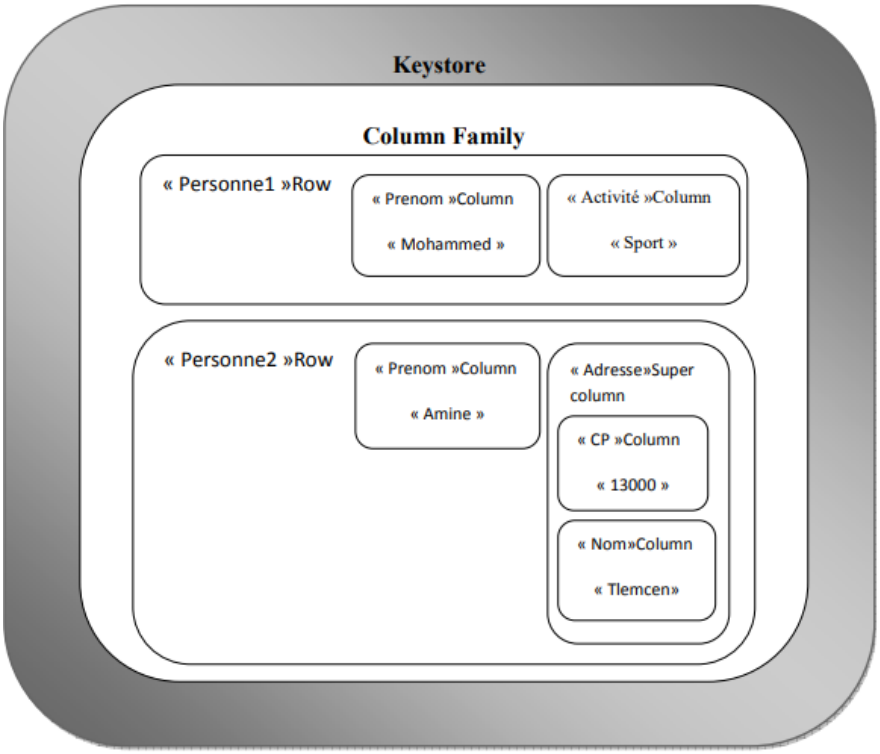

Figure I.13 : Modèle NoSQL « colonnes »[29]

3.5.4 Modèle orienté « graphe »

Ce modèle (figure I.14) qui repose sur la théorie des graphes, permet de représenter les données sous la forme de graphes. Le modèle s'appuie sur la notion de nœuds, de relations et de propriétés qui leur sont rattachées. Les entités sont alors les nœuds du graphe et les relations que partagent les entités sont alors des arcs qui relient ces entités. Ce modèle est notamment adapté au traitement des données des réseaux sociaux. Notons que les systèmes NoSQL orientés graphe trouvent un certain intérêt pour des applications dans le domaine du Web Sémantique, dans la gestion de bases de données de triplets RDF (*triple-stores*), permettant de stocker des connaissances ou ontologies, un triplet étant une arête d'un graphe.

Les systèmes NoSQL orientés graphe les plus connus sont Neo4J, Infinite Graph, OrientDB.

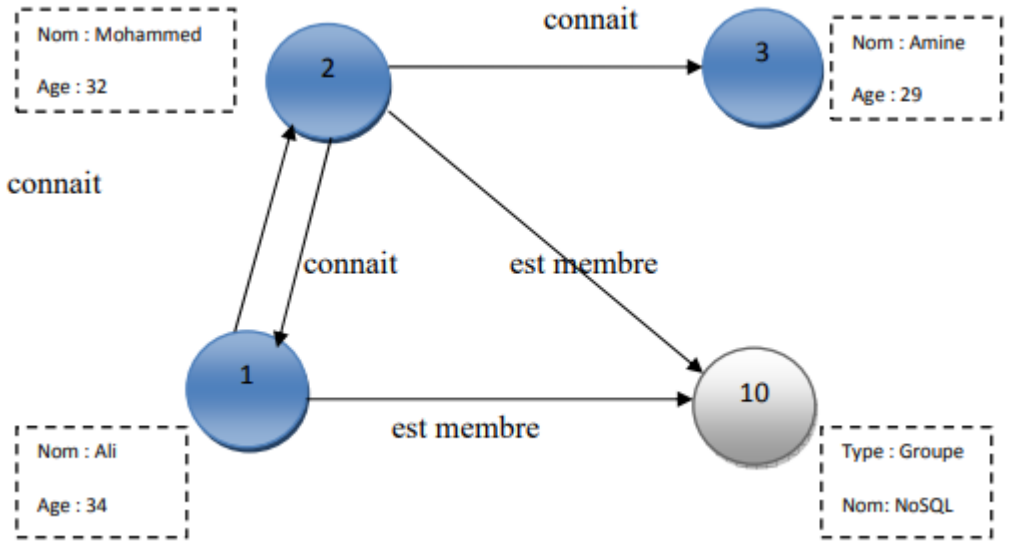

Figure I.14 : Modèle NoSQL « graphe »[29]

3.6 SQL vs NoSQL

SQL et NoSQL ont chacun son lot d'avantages et de désavantages, aucune des deux solutions n'est donc meilleure que l'autre. En fonction des types de besoins que peut avoir une entreprise par exemple, une solution SQL pourrait être plus avantageuse qu'une solution NoSQL, et vice-versa[11].

Dans cette partie on va citer quelques points forts de NoSQL qui n'existent pas dans SQL

- Plus évolutif : NoSQL est plus évolutif. C'est en effet l'élasticité de ses bases de données NoSQL qui le rend si bien adapté au traitement de gros volumes de données. Au contraire, les bases de données relationnelles ont souvent tendance à utiliser la scalabilté verticale, qui consiste à augmenter les capacités du serveur (plus d'espace, ajout de RAM, augmentation de la puissance du processeur) quand celui-ci atteint ses limites
- Plus flexible : N'étant pas enfermée dans un seul et unique modèle de données, une base de données NoSQL est beaucoup moins restreinte qu'une base SQL.
- Plus économique : Les serveurs destinés aux bases de données NoSQL sont généralement bon marché et de faible qualité, contrairement à ceux qui sont utilisés par les bases relationnelles. De plus, la très grande majorité des solutions NoSQL sont open–source, ce qui reflète d'une part une économie importante sur le prix des licences, mais aussi une vitesse de développement du produit beaucoup plus grande.
- Plus simple : Les bases de données NoSQL ne sont pas forcément moins complexes que les bases relationnelles, mais elles sont beaucoup plus simples à déployer. La façon dont elles ont été conçues (réparation automatique, données distribuées, simplicité des modèles de données)

3.7 Les SGBD NoSQL

Les bases de données relationnelles sont une technologie bien établie dans la plupart des organisations, qui sous-tendent les applications existantes qui répondent aux besoins actuels de l'entreprise. Cependant, les équipes informatiques envisagent de plus en plus des solutions de

rechange à l'infrastructure relationnelle existante pour créer des applications modernes, motivées par le besoin de:

- Accélérez le délai de mise sur le marché, grâce à un développement agile et à des modèles de données flexibles.
- Traitez des volumes croissants de données structurées, semi-structurées et non structurées.
- Échelle au-delà des contraintes de capacité des systèmes existants.
- Libérez-vous des coûteux logiciels et matériels propriétaires de bases de données.

Pour répondre à ces exigences, les nouvelles solutions NoSQL désigne une catégorie de systèmes de gestion de bases de données (SGBD) qui n'est plus fondée sur l'architecture classique des bases relationnelles. L'unité logique n'y est plus la table, et les données ne sont en général pas manipulées avec SQL. Dans cette section nous intéresserons à la technologie MongoDB.

3.7.1 MongoDB

MongoDB est un système de gestion de base de données orientée document, écrit en C++ et distribuée sous licence AGPL (licence libre) et très adaptée aux applications web. MongoDB a été adoptée par plusieurs grands noms de l'informatique, tels que Foursquare, ou bien même GitHub. Elle manipule des documents au format *BSON* (*Binary JSON*), qui est un dérivé de JSON plus axé sur la performance, fonctionnant comme une architecture distribuée centralisée. MongoDB réplique les données sur plusieurs serveurs avec le principe de maître-esclave, permettant ainsi une plus grande tolérance aux pannes. La répartition et la duplication des documents sont faites de telle sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué en un nombre de fois suffisant[31].

4. Les systèmes distribués

Les moteurs NoSQL sont majoritairement conçus pour être utilisés sur une architecture distribuée, afin de pouvoir gérer la montée de charge et de volumétrie de données. Les moteurs NoSQL utilisent deux manières différentes pour distribuer les traitements et les données sur leurs divers nœuds. La distribution de données avec maître et celle sans.

4.1 Distribution sur un schéma maître esclave

La distribution de données sur un schéma dit maître esclave (figure I.15) consiste à avoir un seul serveur dit maître dans un cluster, les autres serveurs étant esclave. Les requêtes des clients arrivent directement sur le serveur maître, pour ensuite être redirigées sur les serveurs « esclaves » qui contiennent l'information de la requête.

L'une des vulnérabilités de cette configuration consiste dans l'exposition à un point unique de défaillance (*SPOF – Single Point of Failure*). Ainsi, une mal fonction du serveur « maître » entraîne la panne du cluster. En effet, si le serveur tombe en panne, il ne pourra plus réceptionner les requêtes clientes et les redistribuer aux serveurs « esclave » concernés.

Un point important ici est la façon dont sont répliquées les données dans un schéma maître-esclave. Une réplication de données consiste à faire une copie des données entre serveurs afin de garantir non seulement la disponibilité de celles-ci mais aussi de se protéger de la perte de données.

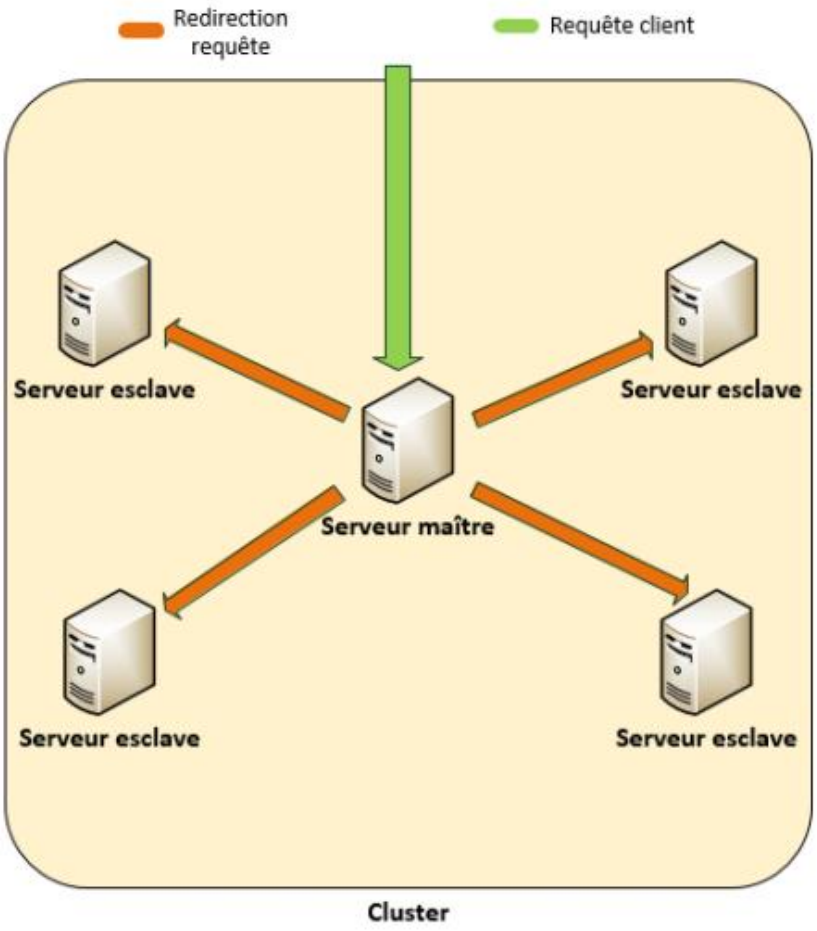

Figure I.15 : Distribution sur schéma maître-esclave[11]

Cela fonctionne ainsi : Les écritures se font uniquement sur le serveur « maître », le serveur lui va se charger de faire les réplications de données sur les serveurs « esclaves ». Le processus de réplication est géré automatiquement par le serveur et sa fréquence peut être configurable par l'administrateur. La lecture de données peut être faite aussi bien sur le maître que sur l'esclave, avec néanmoins un risque de produire des problèmes de cohérence de données si une lecture est faite sur un esclave n'ayant pas reçu la dernière version des données du maître[23].

4.2 Distribution sur un schéma multi-maitre

La distribution de données sur une architecture multi-maître part du principe que la panne est une normalité et que chaque serveur du cluster a exactement la même importance et que chaque serveur peut fournir un service complet en écriture ou en lecture. Il est important d'aborder différents point cruciaux afin qu'une architecture de ce type puisse fonctionner correctement, tel que la redirection de requête aux bons serveurs, savoir quels serveurs composent le cluster, ainsi que les techniques pour distribuer au mieux les données sur les différentes machines du cluster[11].

5. Les bancs d'essai décisionnels

Les transactions de détail des ventes se produisent partout, en quantités énormes. Chercher des clients, des achats ou des produits, le volume de données est énorme et ne cesse de croître. Tandis que la plupart des analystes utilisaient divers outils de veille stratégique pour recueillir des informations importantes sur leurs clients grâce à ces transactions, cette tâche est devenue de plus en plus difficile car de plus en plus de transactions ont lieu en magasin et en ligne. Extraire des informations pertinentes à partir de toutes ces données est essentiel pour la survie à l'ère numérique acharnée.

Les analystes ont la capacité de créer des promotions personnalisées et même d'améliorer la présentation de leurs magasins et faire des analyses détaillées de leurs transactions[5].
Il s'agit d'un test bien connu des performances des requêtes de base de données, modélisé d'après les problèmes classiques d'entreposage de données, avec une variante de transaction de vente en détail.
Star Schema Benchmark (SSB) est une référence conçue pour mesurer la performance des transactions dans les applications d'entrepôt de données. Il est basé sur le benchmark TPC-H , avec quelques modifications bien, notamment la suppression de tables et de champs non pertinents et la dénormalisation .En particulier, les données sont réorganisées dans un schéma en étoile . Les requêtes sont également basées sur quelques-unes des requêtes TPC-H, mais le nombre de requêtes est plutôt petit pour permettre aux utilisateurs d'exécuter facilement la SSB sur différentes plates-formes[32].

5.1 Les modèles de TPC

Le *TPC* (*Transaction Processing Performance Council*) est un organisme à but non lucratif créé pour définir des bancs d'essais et diffuser les résultats d'évaluation de performance. Les bancs d'essais proposés par le TPC sont répartis en quatre catégories : TPC-C et TPC-E sont des bancs d'essai pour le traitement des transactions en ligne, TPC-H, TPC-DS et TPC-DI sont destinés pour les systèmes d'aide à la décision, TPC-VMS et TPCx-V pour les BD virtualisées, TPCx-HS et TPCx-BB pour les données massives (BigData) et TPC-Energy et TPC-Pricing pour des spécifications communes[33].

5.1.1 TPC-DS

TPC-DS (figure I.16) est le standard de référence de l'industrie pour mesurer la performance des solutions d'aide à la décision, y compris, sans toutefois s'y limiter, les systèmes BigData. Il modélise plusieurs aspects généralement applicables d'un système d'aide à la décision, notamment les requêtes et la maintenance des données. Bien que le modèle commercial sous-jacent de TPC-DS soit un fournisseur de produits de vente au détail, le schéma de base de données, le remplissage de données, les requêtes, le modèle de maintenance des données et les règles de mise en œuvre ont été conçus pour être largement représentatifs des systèmes modernes d'aide à la décision[34]. Ce repère illustre les systèmes d'aide à la décision qui:

- Examiner de gros volumes de données

[5] Retail Analytics et le référentiel Star Schema-Pilosa.html

- Donner des réponses à des questions commerciales réelles
- Exécuter des requêtes sur diverses exigences opérationnelles et complexités (par exemple, ad-hoc, reporting, OLAP itératif, exploration de données)
- Fonctionne sur des solutions «BigData», telles que le SGBDR et les systèmes basés sur Hadoop / Spark

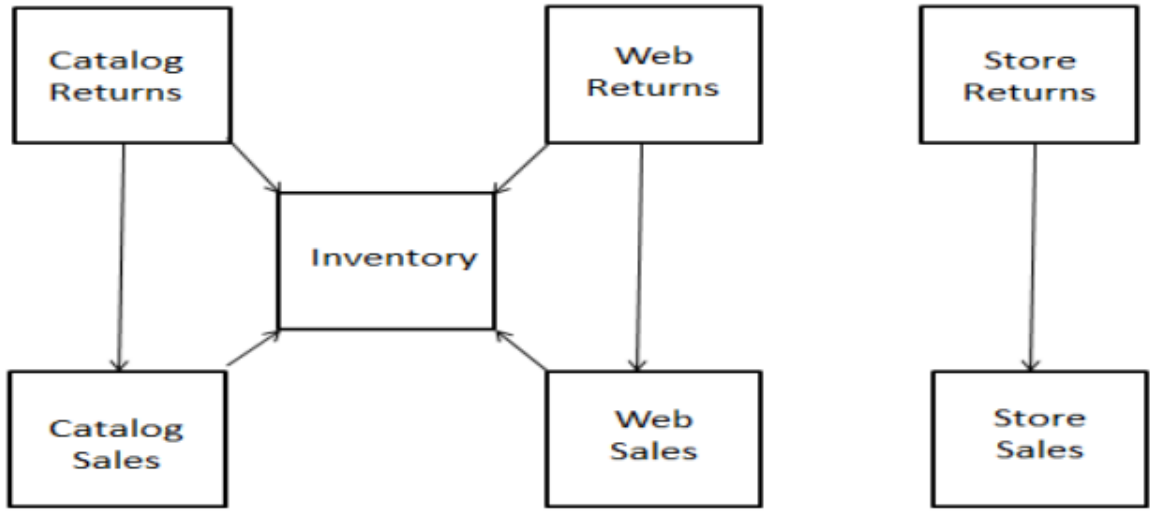

Figure I.16 : Schéma de base de données de TPC-DS[35]

5.1.2 TPC-H

TPC-H est une référence qui simule un environnement de base de données du système d'aide à la décision. Il se compose d'une suite de requêtes ad hoc axées sur les affaires et les modifications de données simultanées. Les requêtes et les données peuplant la base de données ont été choisis pour une grande pertinence ensemble de l'industrie tout en maintenant un degré suffisant de facilité de mise en œuvre. La performance d'un système ne peut être mesuré quand il fournit des moyens pour une analyse d'affaires sur un ensemble de données[36].
TPC-H évalue la performance des différents systèmes d'aide à la décision par l'exécution d'ensembles de requêtes sur une base de données standard dans des conditions contrôlées. Les requêtes TPC-H [20]:
• Donner des réponses aux questions d'affaires dans le monde réel;
• Simuler générés requêtes ad hoc;
• Sont plus complexes que la plupart des transactions OLTP;
• Générer une activité intense de la part du composant serveur de base de données du système en cours de test;
• Sont mises en œuvre avec des contraintes dérivées de rester synchronisées avec les bases de données de production en ligne.
La figure I.17 représente le schéma de base de données de TPC-H alors que la figure I.18 représente le schéma détaillé de TPC-H

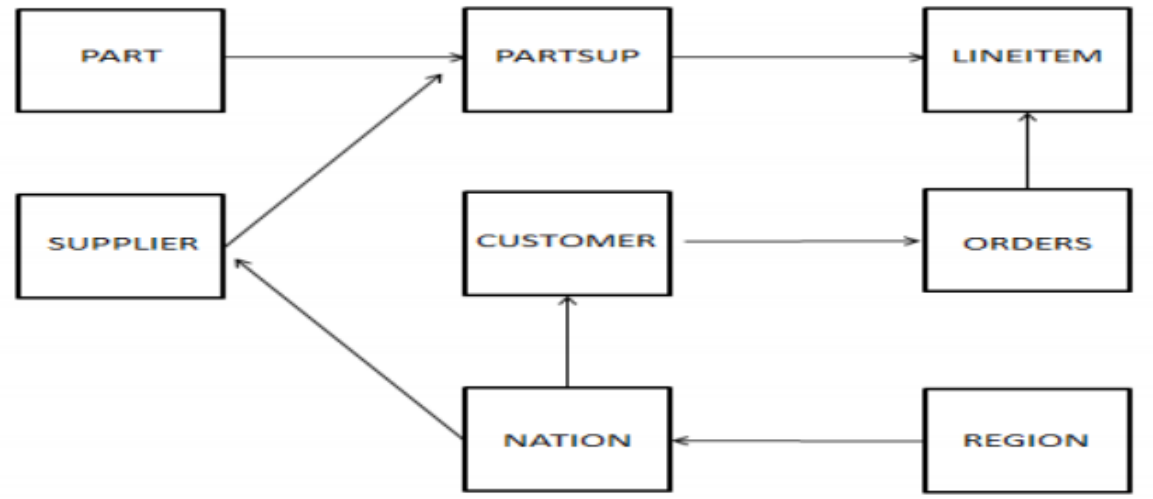

Figure I.17 : Schéma de base de données de TPC-H[35]

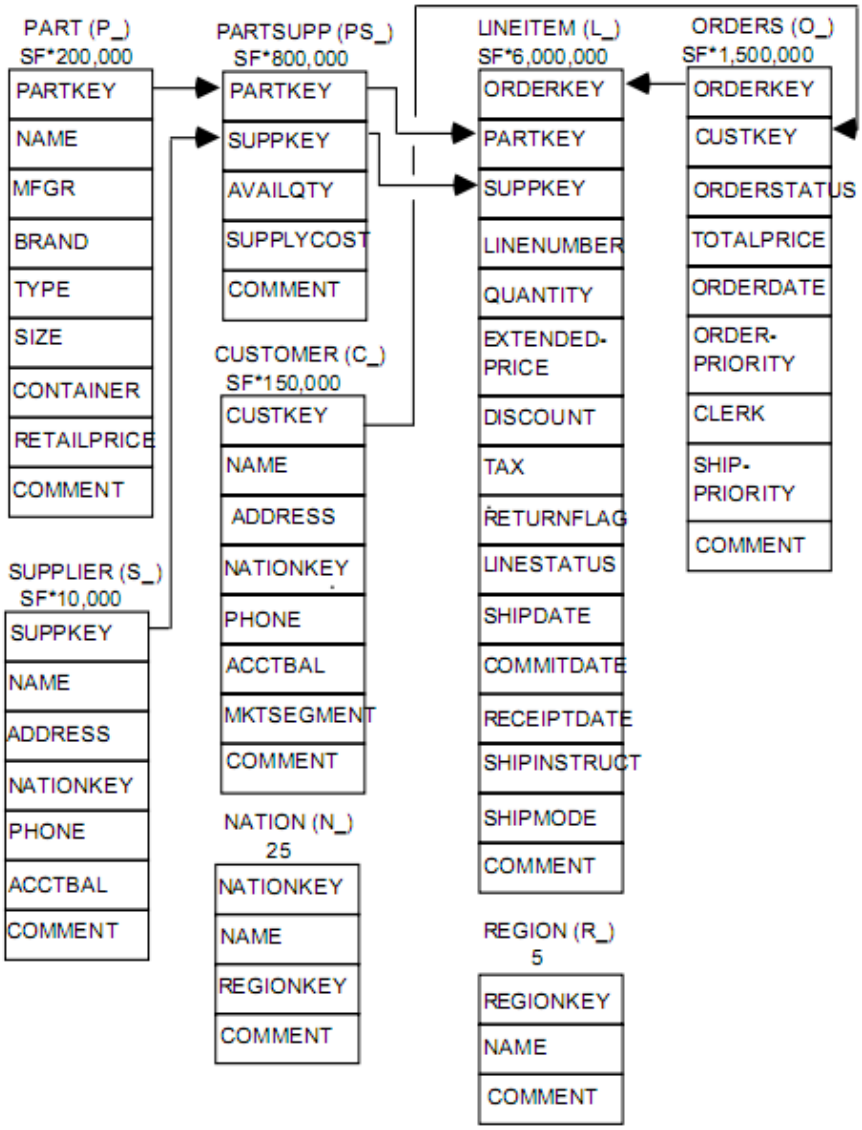

Figure I.18 : Schéma détaillé de TPC-H[32]

5.2 Star Schéma Benchmark(SSB)

Le SSB est un banc d'essai des entrepôts de données dérivé de TPC-H. Contrairement au TPC-H, il utilise un manuel de schéma en étoile des entrepôts de données. Il contient moins de requêtes que TPC-H et à des exigences moins strictes en terme de formes de calibrage autorisées et interdites[6].

Dans la terminologie du schéma en étoile, la table de faits contient des commandes en ligne, avec des clés pour quatre tables de dimension, décrivant les dates, les clients, les fournisseurs et les pièces. Les champs et les relations sont indiqués dans le diagramme représenté dans la figure I.19 [32] .

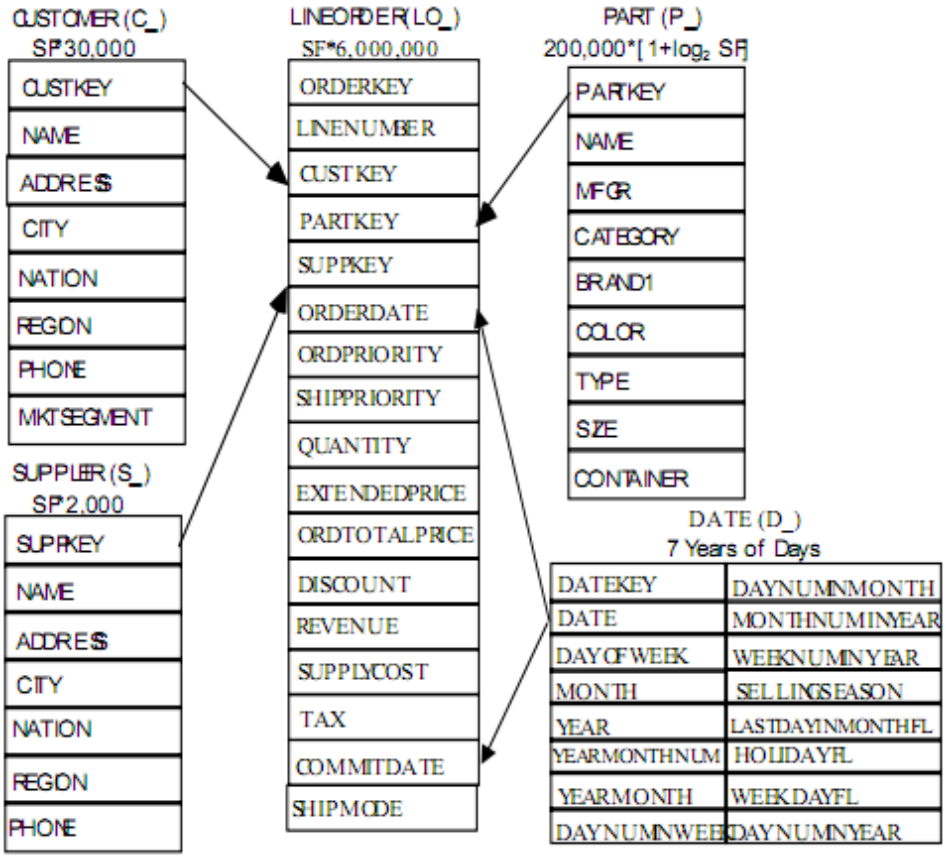

Figure I.19 : Schéma détaillé de SSB[32]

Le schéma de référence en étoile (SSB) est conçu pour mesurer la performance des produits de base de données à l'appui des applications classiques d'entreposage de données. SSB implémente les mêmes données logiques dans un schéma en étoile traditionnel tandis que TPC-H modélise les données dans la 3ème forme normale [33].

Nous expliquons brièvement, dans cette section, les modifications à opérer sur le schéma TPC-H afin d'obtenir SSB.

5.2.1.1 Modifications apportées à la table de fait

Nous exposons, dans ce qui suit, les modifications à opérer sur la table de fait du schéma TPC-H.

- **Création d'une table de fait *LINEORDER*:** La combinaison des tables *LINEITEM* et *ORDER* dans SSB donne naissance à une table de fait pour l'analyse des ventes appelée *LINEORDER*. Cette dé normalisation est standard dans l'entreposage de données, et rend de nombreuses jointures inutiles dans les requêtes courantes[32].
- **Suppression des quelques colonnes de *LINEITEM* et *ORDER* et ajoutez-en à *LINEORDER*.**

a) La suppression des attributs *COMMENT* et *SHIPINSTRUCT* de *LINEITEM* (27 caractères et 25 caractères respectivement), et *COMMENT* de *ORDER* (49 caractères). En effet, un entrepôt ne stocke pas d'informations textuelles n'ayant pas de valeur pour l'analyse dans une table de faits, sachant qu'il ne peut pas être agrégé et nécessite un stockage important.
b) De même, la suppression *LO_CLERK*, qui ne semble utile que dans les données opérationnelles, bien que certaines abstractions les informations puissent être incluses dans un entrepôt de données sous une forme dans laquelle une requête peut renvoyer des résultats quantitatifs.
c) L'ajout de *LO_SUPPLYCOST* pour *PART*, *LO_ORDSUPPLYCOST* additionnant pour *ORDERS* et apportons *O_TOTALPRICE* comme *LO_ORDTOTALPRICE*.
d) On supprime les attributs TPC-H pour les événements suivant *ORDERDATE* d'un élément. C'est-à-dire : *SHIPDATE*, *RECEIPTDATE* et *RETURNFLAG*. Il est clair que les informations relatives à la commande doivent pouvoir être interrogées avant l'expédition, réception du client et retour des événements plusieurs jours plus tard. La manière dont une telle séquence de dates est normalement traitée dans un entrepôt de données est une séquence de tables mais elles sont trop complexes pour notre objectif de référence. On garde l'attribut *COMMITDATE* (engagement d'expédition) dans *SSB*, car il fait partie de la négociation de vente[37].

5.2.1.2 Modifications apportées aux tables de dimension TPC-H

- **Suppression de la table *PARTSUPP* :** La suppression de la table *PARTSUPP* de TPC-H en raison d'une incompatibilité des niveaux de granularité. Alors que dans TPC-H, les tables *LINEITEM* et *ORDER* (combinées dans SSB en tant que table *LINEORDER*) ont le niveau de granularité de transaction le plus fin, la table *PARTSUPP* contient ce qu'on appelle une granularité d'instantané périodique. En gros, cela signifie que l'actualisation ajoutant de nouvelles lignes au fil du temps dans *LINEORDER* n'ajoute pas de lignes à *PARTSUPP*, qui est figé dans le temps[32].
- **Suppression, l'ajout et le changement des colonnes.**
  - ❖ Raccourcir les colonnes[37].
    - Réduire la taille de la colonne *P_NAME* de TPC-H considérée trop longue (55 octets, cinq "couleurs" concaténées); vraisemblablement, cette longueur avait pour but

de rendre la table *PART* plus grande et plus difficile à interroger; *P_NAME* est 22 octets en SSB (deux "couleurs" concaténées).

- Bien que *P_MFGR* est de 25 octets dans TPC-H, on modifie les valeurs en ["MFGR", M], où M est une valeur aléatoire [1, 5], un total de 6 caractères, par exemple:"MFGR #2. On effectue cette modification uniquement pour simplifier la SSB. Les utilisateurs n'ont donc pas besoin d'apprendre les noms P_MFGR complexes.

- ❖ La suppression des colonnes. Outre *P_RETAILPRICE* et *C_ACCTBAL*, on supprime également *P_COMMENT*; comme avec *O_COMMENT*, on n'a aucune utilité pour un commentaire non analysé dans une requête *Data Warehouse*.
- ❖ L'ajout des colonnes et modification des noms de colonnes. Les colonnes de schéma TPC-H ont toutes de petites cardinalités (nombres de valeurs), avec *O_ORDERDATE* comme seule exception. ainsi, la plupart des prédicats de colonne possibles génèrent de grands facteurs de filtrage qui rendent inefficaces les restrictions d'index. Comme pour *O_ORDERDATE*, les prédicats de TPC-H limitent ce délai à un an au minimum, et la plupart des restrictions à cinq ans sur sept. Pour cette raison, les principaux produits de base de données exécutant TPC-H ne créent des index que sur les clés primaires, afin de faciliter les jointures. On considère ces petites cardinalités comme irréalistes et apportons un certain nombre de modifications raisonnables pour autoriser les prédicats avec des facteurs de filtrage plus petits.

- **Suppression des tables *NATION* et *REGION* :** Qui font tourner le schéma en étoile dans un schéma en flocon de neige lorsqu'il est associé aux dimensions SSB *CUSTOMER*, *PART* et *SUPPLIER*. De telles tables, qui ajoutent des jointures aux requêtes TPC-H, peuvent être appropriées dans un système OLTP pour appliquer l'intégrité, mais pas dans un système d'entreposage de données, où les données sont nettoyées à l'arrivée. Il est peu probable qu'un utilisateur doive parcourir la dimension *NATION* ou *REGION* pour déterminer la requête à exécuter, mais dans ce cas, les tables peuvent exister pendant des fins de navigation sans être impliquées dans des requêtes[36].
- **Création d'une table *DATE* :** Contrairement aux tables supprimées *NATION* et *REGION*, la table *DATE* est pertinente dans un entrepôt relatif aux ventes et comporte un grand nombre d'attributs utiles pour les requêtes, telles que *DAYOFWEEK*, *MONTH*, *SELLINGSEASON*, etc [32].

Le résultat obtenu après cette série de ces modifications sur le schéma TPC-H constitue un schéma en étoile d'un magasin de données (*Data Mart*), dont la table centrale *LINEORDER* étant la table de fait ,et les tables *CUSTOMER, PART, SUPPLIER* et *DATE* étant les tables de dimensions.

5.2.1.3 Définition des requêtes SSB

Le schéma SSB modélise l'entrepôt un entrepôt de données d'un fournisseur en gros et ses requêtes sont des versions simplifiées des requêtes TPC-H. Ils Elles sont organisées en quatre vols de trois à quatre requêtes chacun [37].Chaque vol consiste en une séquence de requêtes qu'une personne travaillant avec le système d'entrepôt de données pourrait demander, par exemple, à une analyse approfondie.

•**Vol de requête Q1.** Cette requête mesure l'augmentation des revenus résultant de l'élimination de diverses gammes de remises dans des intervalles de quantité de commande de produit donnés expédiées au cours d'une année donnée Il effectue une jointure unique entre la table *LINEORDER* et la petite table de dimension *DATE*[35].

•**Vol de requête Q2.** La requête compare les revenus de certaines classes de produits et les fournisseurs d'une région donnée, regroupés par catégories de produits plus restrictives et toutes les années de commandes. Les requêtes du vol 2 rejoignent les tables *LINEORDER, DATE, PART* et *SUPPLIER*. Ils agrègent et trient les données par année et par marque[35].

•**Vol de requête Q3.** La requête récupère le total des revenus pour les transactions en ligne dans une région donnée pendant une certaine période, regroupés par pays client, pays fournisseur et année. A partir d'un client, fournisseur, région spécifique et date, vol 3 pénètre dans des villes et des années spécifiques [35].

•**Vol de requête Q4.** La requête fournit une séquence des requêtes "*What-If*" qui peuvent être générées dans un style d'exploration OLAP. Commencer avec une requête qui a des contraintes plutôt faibles sur les colonnes tridimensionnelles, nous récupérons les bénéfices agrégés groupés par année et par pays du client. Le vol 4 est le plus complexe, car il joint toutes les tables [35].

6. Les travaux connexes

La performance est un problème crucial pour les utilisateurs et les concepteurs de bases de données et avec le développement des données massives, l'évaluation des technologies centrées sur la décision telles que les entrepôts de données n'est pas une tâche facile. Dans ce contexte et plus précisément dans le domaine de développement des bancs d'essais il existe de nombreux travaux qui proposent des bancs d'essais permettent de comparer des systèmes de base de données. Ce sont des modèles synthétiques de bases de données et de charges (opérations à effectuer sur la base), ainsi que des ensembles de mesures de performance. Il existe des bancs d'essai dédié au contexte d'aide à la décision et autre dédiée au contexte BigData[36]. Dans ce qui suit, nous détaillons ces bancs d'essais.

6.1 Les bancs d'essai décisionnels

Les bancs d'essais édités par *TPC (Transaction Processing Council)* sont les plus utilisés pour évaluer les systèmes décisionnels. TPC-D a fait son apparition dans le milieu des années 90 et forme la base de TPC-H et TPC-R, qui l'ont désormais remplacé. TPC-H et TPC-R sont, en fait, identiques, seul leur mode d'utilisation les différencie. TPC-H est conçu pour le requêtage ad-hoc (les requêtes ne sont pas connues à l'avance et toute optimisation préalable est interdite), tandis que TPC-R a une vocation de *reporting* (les requêtes sont connues à l'avance et des optimisations adéquates sont possibles). TPC-H et TPC-R exploitent le même schéma de base de données que TPC-D (Les données sont structurées conformément à un schéma fixe en flocon)[36].

En 2009, un autre banc d'essai a fait son apparition [32]. Il s'agit de *SSB (Start Schema Benchmark).* Il introduit un certain niveau de dénormalisation des données pour des raisons de simplicité. Il implémente un schéma en étoile pur composé d'une table de faits et de tables à 4 dimensions.
Dans ce contexte, il existe des travaux pour adapter le benchmark SSB aux systèmes NoSQL. Le banc d'essai CNSSB [10] est proposé pour prendre en charge un seul modèle NoSQL qui est le modèle orienté colonnes, et il propose seulement deux manières d'organiser les données. Le banc d'essai SBB+ [38]. Il est adapté pour les modèles orientés colonnes et orientés documents et offre 4 modèles logiques pour chacun des systèmes NoSQL supportés. Ces deux dernières solutions reposent sur un jeu de requêtes simples contrairement à TPC qui offre un jeu de requêtes plus riche avec un degré de complexité plus important. De plus, elles se limitent à la génération de données fortement structurées, conformes à un schéma fixe, sans support de données variables au niveau du fait et des dimensions [38].

Un autre banc d'essai a été proposé pour répondre aux besoins des systèmes décisionnels basés sur des entrepôts de données multidimensionnelles massives appelé KoalaBench[39] .Il est dérivé du banc d'essai TPC-H, le banc d'essai de référence, pour évaluer les systèmes décisionnels, Il supporte des modèles logiques différents (plat, étoile, flocon et plat flexible). 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.

6.2 Les bancs d'essai BigData

Les bancs d'essai dédiés aux BigData visent à comparer les nouveaux systèmes qui stockent des données massivement distribuées et supportent des calculs parallèles [1].

Le service Yahoo Cloud est un outil très populaire permettant d'évaluer les capacités de récupération et de maintenance des programmes informatiques[40] .Il est souvent utilisé pour comparer les performances relatives aux systèmes de gestion de base de données NoSQL et offre de bonnes performances en termes de chargement de données.
BigFrame est un générateur de banc d'essai. Il se concentre principalement sur les problèmes de volume, de variété et de vélocité pour les environnements de BigData.
BigBench propose un cadre d'évaluation plus riche que les deux premiers [41].Ce dernier modélise des lignes de commandes. Il comprend 3 types de données : structurées (issues du TPC-DS), semi-structurées (flux de clics dans les sites web), et non structurées (commentaires de clients).

Contrairement aux bancs d'essai traditionnels, les bancs d'essai BigData 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.

Il existe d'autres efforts dans le contexte BigData. HadoopToSQL [42], 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, dans les travaux de [43], les auteurs proposent 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. Dans les travaux de

[44], l'auteur définit un ensemble de règles pour traduire le jeu de requêtes TPC-H de SQL en Pig Latin. Il propose une approche de construction des requêtes en Pig pour maximiser les performances. Le tableau décrit les différents travaux principaux avec leurs critiques.

Nom	Description	Critiques
(Noeil & al., 2009)	Il introduit un certain niveau de dénormalisation des donnéespour des raisons de simplicité. Il implémente un schéma en étoile *SSB* composé d'une table de faits et 4 tables dimensions.	-Cette solution n'est pas définie pour une utilisation dans un système distribué ou des bases de données NoSQL. -Elle fournit un seul format (tbl)
SBB+ (Chevalier & al, 2015)	Il est adapté pour les modèles orientés colonnes et orientés documents et offre 4 modèles logiques pour chacun des systèmes NoSQL supportés.	-Les données sont générées en format tbl puis transformer ensuite via une moulinette en format csv et Json. -La génération des donnés ne tient pas en compte l'environnement des big data.
(KoalaBench & al., 2015).	Cette solution supporte des modèles logiques relatifs au TPC-H.	-Ne tient pas en compte le cas du SSB.

Tableau I-1: Comparaison entre les travaux.

6.3 Synthèses des travaux ultérieurs

- Toutes les contributions proposent d'utiliser le moteur de génération fourni par le SSB (non distribué et qui fournit des données en format tbl) et de créer ensuite une moulinette (un programme de conversion de données) pour transformer les données en format JSON ou autres.
- L'écosystème des BigData n'est pris en charge au moment de la génération des données mais uniquement le chargement des données synthétique dans l'entrepôt cible.
- Ces solutions sont définies uniquement pour une utilisation dans un système distribué ou des bases de données NoSQL (chargement des données générées dans un système distribué)

Aucune contribution n'avait intervenu au niveau du générateur de données SSB pour prendre en charge l''écosystème des BigData.

Notre contribution Consiste à intervenir au niveau du générateur de données pour générer des données directement en formats tbl, csv et Json (sans recourir à une moulinette) et de prendre en compte l'écosystème des Big Data qui est caractérisé par le stockage distribué et le traitement parallèle.

Conclusion

Nous avons défini, dans le présent chapitre, les notions fondamentales les plus importantes dans le domaine des systèmes décisionnels, les méga données, les modèles de données *NoSQL* et nous avons terminé par les bancs d'essai décisionnels. Les concepts de base que nous avons présenté constituent des prérequis sur lesquelles se basent l'état de l'art et les contributions présentés dans la dernière section dans ce chapitre.

Bien que le nombre des travaux faire croire à un sujet fermé, mais l'absence de benchmarks décisionnels conçus pour les systèmes NoSQL reste un problème. C'est parmi les causes qui nous a conduit à proposer la réalisation d'un outil graphique de génération de données synthétiques qui compte permet de simuler et d'évaluer un entrepôt de données multidimensionnelles NoSQL orienté documents, avec les différentes manipulations OLAP effectuées par un analyste.

Chapitre II : Approches

II. Approches

Introduction

Grâce à l'ensemble des concepts exposés, nous avons présenté, dans ce chapitre, notre outil nommé OGGB. OGGB est un outil graphique de génération de données_synthétiques. Il offre l'opportunité de générer les données selon le modèle conceptuel de l'entrepôt de données en étoile « *SSB*» (*Star Schema Benchmark*). Lors de la génération des données, plusieurs aspects sont pris en charge par cet outil, tels que le format des données et le volume de données générées. Les données générées doivent être stockées dans un SGBD NoSQL orienté documents à savoir MongoDB. Pour ce faire, nous avons utilisé des règles de transformation permettant de passer depuis un modèle conceptuel multidimensionnel (modèle en étoile MDE) vers un modèle logique NoSQL (orienté documents MLOD*)*. L'outil graphique de génération de benchmark « OGGB » englobe quatre composants à savoir DBGen, DBLoad, DBGenLoad et QGen sachant que ces composants permettent à l'utilisateur de générer, charger et interroger les données ainsi générées.

1. Description de l'outil graphique de génération de benchmark « OGGB »

OGGB s'inscrit comme un outil de génération d'un benchmark NoSQL selon le modèle orienté document. Cet outil a été conçu et développé pour permettre de simuler et d'évaluer un entrepôt de données multidimensionnelles en étoile. En effet, il offre l'opportunité de générer les données selon le modèle conceptuel de l'entrepôt de données en étoile « *SSB* » (*Star Schema Benchmark*). Lors de la génération des données, plusieurs aspects sont pris en charge par cet outil, tels que le format des données et le volume de données générées.

En résumé, l'OGGB est défini selon les points suivants :

- ✓ Le modèle conceptuel SSB comme prototype d'ED.
- ✓ Le modèle logique NoSQL orienté document
- ✓ Le modèle physique du SGBD MongoDB
- ✓ Le mode de génération de données distribué ou mono machine.
- ✓ Le volume de données en octets à générer est paramétrable via le facteur d'échelle (*SF : Scale Factor*).
- ✓ Les formats de sortie de données fournis par OGGB sont les plus appropriés aux outils NoSQL de chargement de données à savoir TBL, CSV et JSON.

La figure II.1 représente le processus de génération de données synthétiques pour l'alimentation de l'entrepôt SSB.

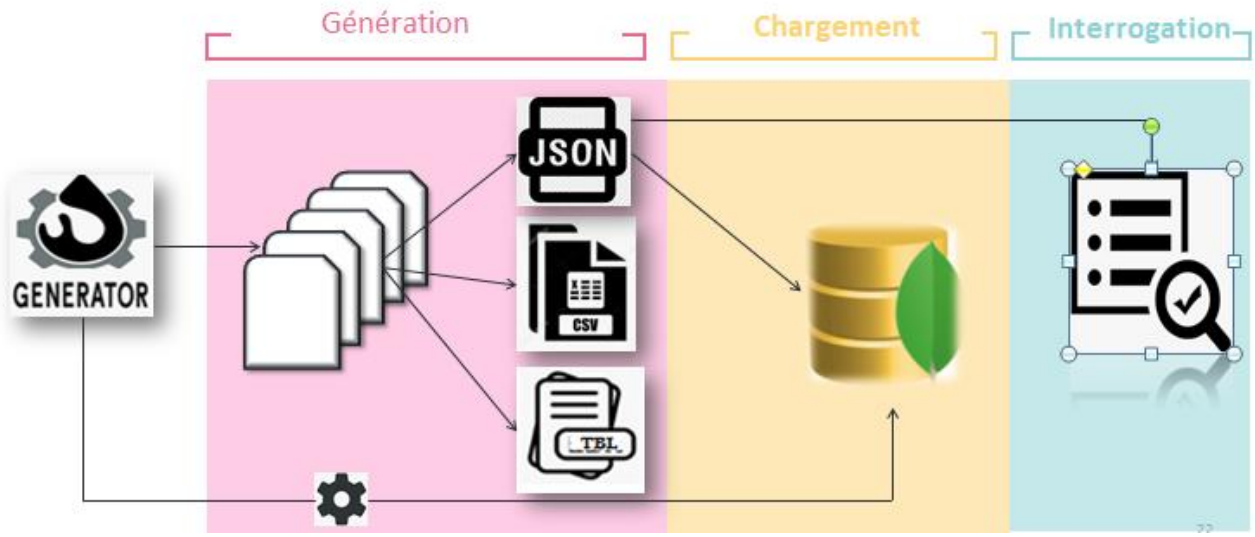

Figure II.1 : processus de génération de données synthétiques pour l'alimentation de l'entrepôt SSB

2. Les composants de l'outil graphique de génération de benchmark « OGGB »

L'outil graphique de génération de benchmark « OGGB » englobe quatre composants à savoir :

2.1 DBGen

2.1.1 Description

Ce composant est utilisé pour la génération des données. Il permet de générer cinq (05) fichiers dont quatre correspondent aux dimensions *SUPPLIER*, *CUSTOMER*, *PART*, *DATE*, en plus d'un fichier correspondant à la table de faits *LINEORDER*. Ces fichiers sont générés selon trois formats à savoir CSV, TBL ou JSON.

DBGen suit le schéma de la figure II.2.

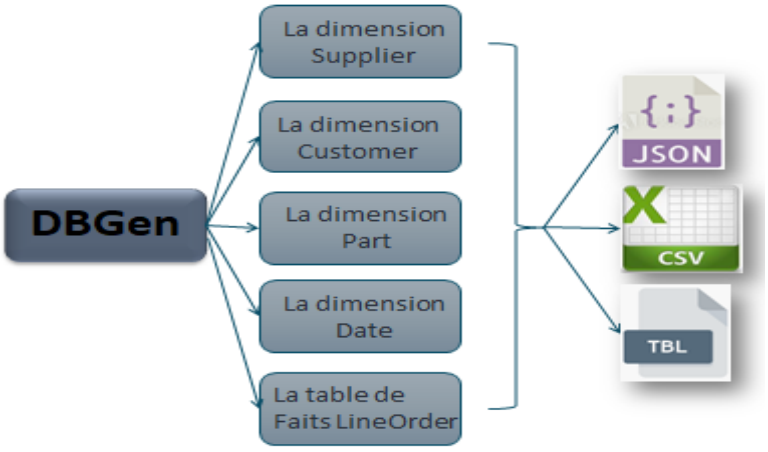

Figure II.2 : Processus de génération de DBGen

2.1.2 L'algorithme

Pour la génération de données nous proposons un programme DBGen. L'algorithme DBGen prend en entrée le format de sortie (typefichier) le nombre de map (nbrmap), le mode d'exécution (modexecution) ainsi que l'indicateur d'échelle (scalfactor). Le résultat est cinq fichiers (*SUPPLIER, CUSTOMER, PART, DATE, LINEORDER)* de type TBL, CSV ou JSON. Avant de commencer la génération, concernant le calcul de nombre de Map nécessaire pour chaque table, il faut calculer le coefficient de chaque table en devisant la taille de table sur la somme des taille, puis multiplier le résultat par le nombre de Map, le résultat de ce calcule peut ne pas être un entier donc nous avons arrondis le nombre calculé .pour générer les donnees il y a l'approche qui a que la fonction Map et l'approche qui a la fonction Map et Reduce, dans les deux approches la première étape est définir le mode d'exécution, deuxième étape est l'appel des méthodes qui permettre de générer les données chaque table, et pour les exécuter il faut tout d'abord crée les RDD en utilisant la méthode parallelize( ), cette méthode prend en paramètre le coefficient calculé pour pouvoir partitionné et parallélisé le traitement pour générer les fichiers

Algorithme : DBGen (algorithme de générateur de données l'approche 1)

Entrée :

scalfactor, nbrmap : entier

typefichier, modexection : chaine de caractère

Sortie : // le résultat de ce algorithme est les cinq fichiers (tbl, csv, json)

début

// calculer le nombre de map pour chaque table

suppmap ← round (2000**Scalfactor/smtaille*)**nbrmap*;

custmap ← round (30000**Scalfactor*/*smtaille)***nbrmap*;

partmap ← round ((200000*floor (1+ log2 (*Scalfactor*)))/*smtaille*)**nbrmap*;

datemap ← round (3000**Scalfactor*/*smtaille*)**nbrmap*;

liordmap ← round (6000000**Scalfactor*/*smtaille*)**nbrmap*;

// définir le mode d'exécution

//appel à la fonction map pour générer les données de la table Supplier (ce traitement est répété pour chaque table)

RDD ← *SC*.parallelize *(suppmap*) .collect;

Supplier ← *RDD*.map (mapsupplier (*typefichier, scalfactor*));

fin

Algorithme : DBGen (algorithme de générateur de données l'approche 2)

Entrée :

1 *scalfactor*, *nbrmap* : entier

2 *typefichier*, *modexection* : chaine de caractère

Sortie : // le résultat de ce algorithme est les cinq fichiers (tbl, csv, json)

3 début

4 **// calculer le nombre de map pour chaque table**

5 *suppmap* ← round (2000**Scalfactor/smtaille*)**nbrmap*;

6 *custmap* ← round (30000**Scalfactor*/*smtaille)***nbrmap*;

7 *partmap* ← round ((200000*floor (1+ log2 (*Scalfactor*)))/*smtaille*)**nbrmap*;

8 *datemap* ← round (3000**Scalfactor*/*smtaille*)**nbrmap*;

9 **// définir le mode d'exécution**

10 **//appel à la fonction Map pour générer les données de la table Supplier (ce traitement est répété pour les 4 tables dimension)**

11 *RDD1*← *SC*.parallelize *(suppmap*) .collect;

12 *Supplier* ← *RDD*.map (mapsupplier (*typefichier*, *scalfactor*));

13 **//appel à la fonction Reduce pour générer les données de la table Lineorder**

14 *RDD5* ← *RDD1*.unoin(les RDD des tables de dimension) *;*

15 *Lineorder*← *RDD2*.reduce (lineordf *(typefichier*, *scalfactor*)) ;

16 fin

2.2 DBLoad

2.2.1 Description

DBLoad est un composant qui offre la possibilité de charger les données déjà générées dans le SGBD cible. En effet, ce composant permet d'extraire les données à partir des fichiers « JSON » générés par DBGen et les chargés dans le SGBD NoSQL orienté documents « MongoDB » comme la montre la figure II.3.

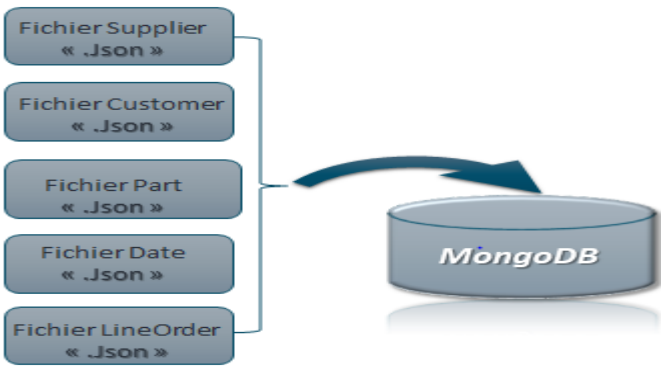

Figure II.3 : Processus de chargement "DBLoad"

2.2.2 L'algorithme

L'algorithme DBLoad permet de charger les données générées dans des fichiers de format JSON (par DBGen) dans le SGBD MongoDB, en effet il prend en entrée les fichiers JSON supplier, customer, date, part et lineorder .La sortie est un entrepôt de données dans MongoDB pour stocker les données dans MongoDB. La première étape est la connexion avec elle, ensuite la création de la base de données sous le nom « *StarSchemaBenchmark* » avec la méthode « *CreateDatabase* », aussi la création des collections avec la méthode « *CreateCollection* » qui prend en paramètre un nom pour la collection crée .Après les étapes de création chaque ligne de fichier sera inséré comme un document JSON dans la collection.

Algorithme 2 : DBLoad (algorithme de chargement)

```
Entrée :
1  Supplier, Customer, Part, Date, Lineorder : fichier Json
2  Sortie : // le résultat de ce algorithme est un entrepôt dans MongoDB
3  début
4  |  // Connexion à MongoDB
5  |  // Création de la base de données
6  |  db ← CreateDatabase ("StarSchemaBenchmark") ;
7  |  // Création des collections, cette partie est répétée 5 fois (le même traitement pour les autres fichiers)
8  |  db.CreateCollection ("supplier") ;
9  |  pour (chaque ligne dans le fichier Supplier) faire
10 |  |  // Insérer la ligne dans un document json de la collection supplier
11 |  fin pour
12 fin
```

2.3 DBGenLoad

2.3.1 Description

C'est un composant qui déclenche la génération des données et ensuite le chargement de manière automatique et simultané.

2.3.2 L'algorithme

DBGenLoad est la combinaison entre DBGen et DBLoad, il permet de générer les données et les stockées au même temps dans MongoDB, il prend en entrée le mode d'exécution (modexecution) soit local, standalone ou cluster, aussi d'autre paramètre on a le nombre de map (nbrmap), le format de sortie de fichier (typefichier) et en dernier l'indicateur d'échelle (scalfactor) un entier pour

préciser la taille des données générés. La sortie de cet algorithme est un entrepôt de données dans le SGBD MongoDB.

Le mode d'exécution la première étape à faire pour définir le mode de génération de donnée, ensuit établir la connexion (MongoDB), après créer les collections des tables et ensuite l'appel à la méthode qui permet de générer les données selon un indicateur d'échelle est fait. La dernière étape est d'insérer chaque ligne comme un document json dans la collection crée.

Algorithme : DBGenLoad (algorithme de génération et chargement)

Entrée :

```
1  scalfactor, nbrmap : entier
2  typefichier, modexection : chaine de caractère
3  Sortie : // le résultat de ce algorithme est un entrepôt de données dans MongoDB
4  début
5  | // mode d'exécution
6  | // Connexion à MongoDB
7  | // Création de la base de données
8  | db ← createDatabase ("StarSchemaBenchmark");
9  | //Création des collections et stockage des données générées, cette partie est répétée 5 fois (le même traitement
   |     pour les autres tables)
10 | db.createCollection ("supplier");
11 | //pour chaque table faire le même traitement que la table supplier
12 | //Appel à la méthode mapsupplier (scalfactor, typefichier) pour générer les données
13 | // Insérer chaque ligne généré comme un document json de la collection supplier
14 fin
```

2.4 QGen

2.4.1 Description

Ce composant est un générateur de requête (figure II.4). Il permet de générer une charge de requête composée de treize requêtes répartie en quatre groupes.

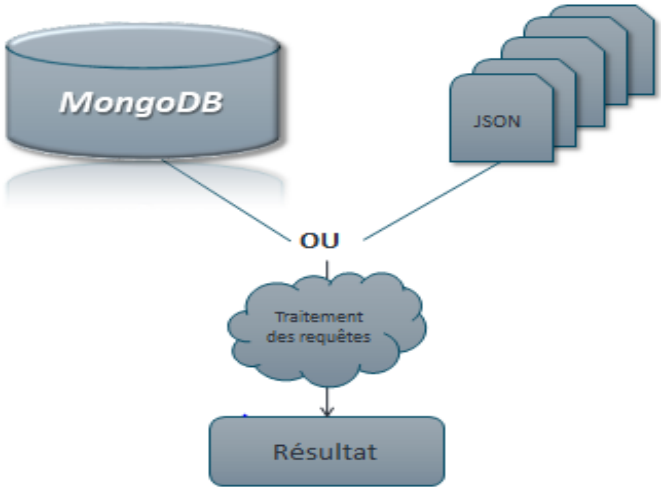

Figure II.4 : Processus d'interrogation de l'entrepôt SSB

2.4.2 L'algorithme

Cet algorithme est de générateur de requête. Il prend en entrée la source qui est soit un entrepôt de donnée ou des fichiers JSON, les entrées des requêtes qui seront utilisées pour l'interrogation. La sortie est le résultat obtenue à travers les requêtes.

Au début de cet algorithme la vérification de la source, si est égale à « MongoDB » la lecture des données se fait à partir de MongoDB sinon à partir des fichiers JSON, ensuite l'exécution des quatre requêtes en utilisant les données en entrée avec différentes valeurs.

Algorithme 4 : QGen **(**Générateur des requêtes)

Entrées : *source* : chaine de caractère

```
1   //les entrées des requêtes
2   regionCustomerQ4, regionSupplierQ4, category1Q4, category2Q4, regionSupplierQ3,
    regionCustomerQ3,                3   regionSupplierQ2, categoryQ2: chaine de caractère
4   discountValMax, discountValMin, quantity, yearQ1, yearDebut, yearFin: entier

Sortie :
5   Q1, Q2, Q3, Q4 : les résultats des requêtes

6   Début :
7     Variables :  supDB : n-tuples de table de MongoDB
8      supFL  : fichier Json

9     si ( source = "MongoDB" ) alors
10      //lire les donnees à partir de MongoDB
11      lire  (Supplier, supDB) // ce traitement est répété pour le reste des tables (Customer, Part, Date,
          Lineorder)

12    sinon
13      //lire les données à partir des fichiers json
14      lire (FSupplier, supFL) // ce traitement est répété pour le reste des tables (Customer, Part, Date,
        Lineorder)

15    fin

16    //Requete1
17    Q1← select sum (LO.EXTENDEDPRICE*LO.DISCOUNT) as revenue
18              from LineOrder LO, Date D
19              where LO.ORDERDATE D.DateKey and D.YEAR = yearQ1 and LO.DISCOUNT between
20              discountValMin  and "+discountValMax and LO.QUANTITY < quantity;

21    //Requete2
22    Q2 ← select sum (LO.REVENUE) as revenue, D.YEAR, P.BRAND1
23              from LineOrder LO, Date D, Part P, Supplier S
24              where LO.ORDERDATE = D.DATEKEY and LO.PARTKEY = P.PARTKEY
25              and LO.SUPPKEY = S.SUPPID and P.CATEGORY = categoryQ2 and S.REGION =regionSupplierQ2
26              group by D.YEAR, P.BRAND1 order by D.YEAR, P.BRAND1;

27    //Requete3
28    Q3← select C.NATION as C_NATION, S.NATION AS S_NATION, D.YEAR, sum(LO.REVENUE)
29              as revenue  from Customer C, LineOrder LO, Supplier S, Date D where LO.CUSTKEY =
30              C.CUSTID and LO.SUPPKEY = S.SUPPID and LO.ORDERDATE = D.DATEKEY and C.REGION
31              = regionCustomerQ3 and S.REGION = regionSupplierQ3 and D.YEAR >= yearDebut and
32              D.YEAR <= yearFin  group by C.NATION, S.NATION, D.YEAR
33              order by D.YEAR asc, REVENUE desc;

34    //Requete4
35    Q4 ← select D.YEAR, C.NATION, sum (LO.REVENUE - LO.SUPPLYCOST) as profit
36              from Date D, Customer C, Supplier S, Part P, LineOrder LO
37              where LO.CUSTKEY = C.CUSTID and LO.SUPPKEY = S.SUPPID and LO.PARTKEY =
38              P.PARTKEY and LO.ORDERDATE = D.DATEKEY and C.REGION = regionCustomerQ4 and
39              S.REGION = regionSupplierQ4and (P.MFGR =  category1Q4 or P.MFGR = category2Q4)
40              group by D.YEAR, C.NATION order by D.YEAR, C.NATION;
41  fin
```

3. Les formats de sortie de données

Afin d'optimiser la phase de chargement des données dans le SGBD NoSQL « MongoDB », 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, le format pivot dans MongoDB étant le format JSON.

La nécessité de spécifier le format de sortie concerne uniquement le modèle orienté documents [38]. Plusieurs formats de fichiers sont donc possibles : TBL, CSV et JSON ,ces trois formats sont des fichiers composés d'une suite de caractères où l'on distingue deux types d'information, les données et les caractères permettant de structurer ces données(voir la figure II.5) [45]

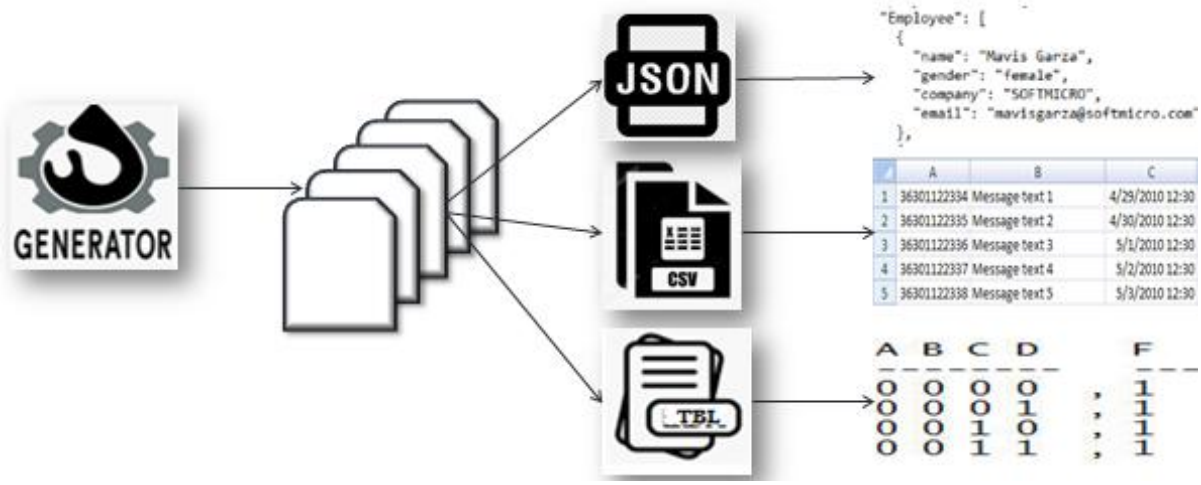

Figure II.5: Représentation détaillée des formats adoptés pour la génération des données

4. .L'indicateur d'échelle

OGGB génère les données avec différents volumes. L'indicateur d'échelle (*SF : Scale Factor*) 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 [6].

Pour connaitre le nombre de lignes à générer dans chaque table, nous avons utilisés les formules mentionnées dans le tableau II-1

Table	Lignes
Supplier	*SF**2,000
Customer	*SF**30,000
Part	200,000**floor* (1+log2SF)
Date	7 *years of days*
LineOrder	SF*6000,000

Tableau II-1 : Nombre de lignes par table(Source: Neil et al., 2009)

5. Les règles de passage d'un modèle conceptuel vers un modèle logique NoSQL

Comme l'illustre la figure II.6, nous avons utilisé des règles de transformation permettant de passer directement depuis un modèle conceptuel multidimensionnel (modèle en étoile MDE) vers un modèle logique NoSQL orienté documents MLOD.

(Les règles de mappage conceptuel-logique ont été proposées dans le cadre d'un autre PFE faisant partie d'un projet global initié par la même équipe d'encadrement).

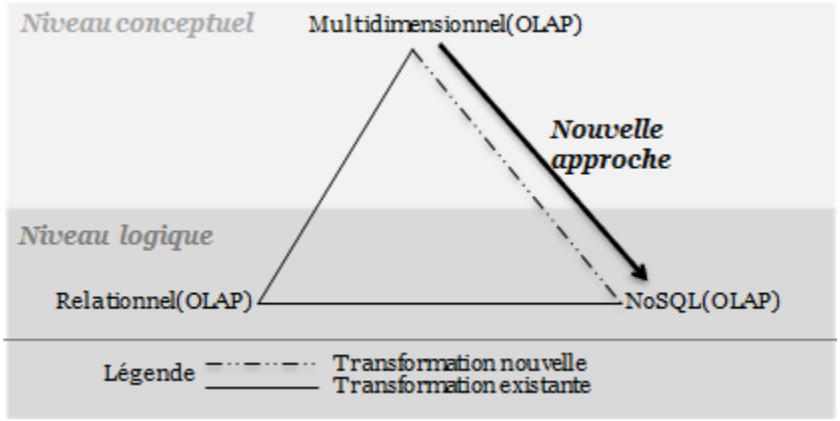

Figure II.6: Règles de transformations d'un modèle conceptuel en un modèle logique existantes[6].

Dans le modèle orienté documents, les données sont stockées en collections des documents. La structure des documents est définie par l'intermédiaire d'attributs (pouvant s'apparenter à des couples clé/valeur où 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, appelées documents imbriqués [6].
Les règles de transformation que n ous avons utilisé sont :

- Règle 1: Une table de faits TF, dans le modèle en étoile MDE, devient une collectionC^f, dans le modèle logique NoSQL orienté documents MLOD, dont les documents prennent leurs attributs à partir du schéma de la *TF*.
- Règle 2: Une ligne de la *TF*, dans le *MDE*, devient un document D^f inclus dans la collection C^f dans le MLOD.
- Règle 3: La clé primaire de la *TF*, dans le MDE, devient un attribut simple dans les documents D^f de la collection C^f dans le MLOD.
- Règle 4: Une mesure dans une *TF*, dans le MDE, devient un attribut simple dans les documents D^f de la collection C^f dans le MLOD.
- Règle 5: La clé étrangère de la *TF*, dans le MDE, devient un attribut simple dans les documents D^f de la collection C^f dans le MLOD. Cet attribut permettra de relier un document de type 'Fait' avec un document de type 'Dimension' correspondant dans la même collection, afin d'assurer le lien implicite entre ces documents.
- Règle 6: Une TD, dans le MDE, devient une collectionC^d, dans le MLOD, dont les documents prennent leurs attributs à partir du schéma de la TD.

- Règle 7: Une Ligne de la TD dans le MDE devient un documentD^d, qui est inclut dans la collectionC^d, dans le MLOD.
- Règle 8: La clé primaire de la TD, dans le MDE, devient un attribut simple dans les documents D^d de la collection C^d dans le MLOD.
- Règle 9: Un attribut d'une TF, dans le MDE, devient un attribut simple dans les documents Dd de la collectionC^ddans le MLOD.
- Règle 10: La clé naturelle de la TD, dans le MDE, devient un attribut simple dans les documents D^d de la collection C^ddans le MLOD.

6. Les approches proposées

Concernant l'utilisation de MapReduce, nous avons proposé deux approches dont le but de trouver une solution qui répond à nos besoins d'une part, et qui minimise le temps de génération d'autre part.

- Approche 1 : La phase Map crée et génère les données synthétiques en vue de construire les 5 fichiers (tables de dimensions et table de fait), la phase Reduce n'est pas nécessaire.
- Approche 2 : la phase Map crée et génère seulement les données des tables de dimension, et la phase Reduce pour la génération des données de la table de faits.

La figure II.7 représente les deux approches

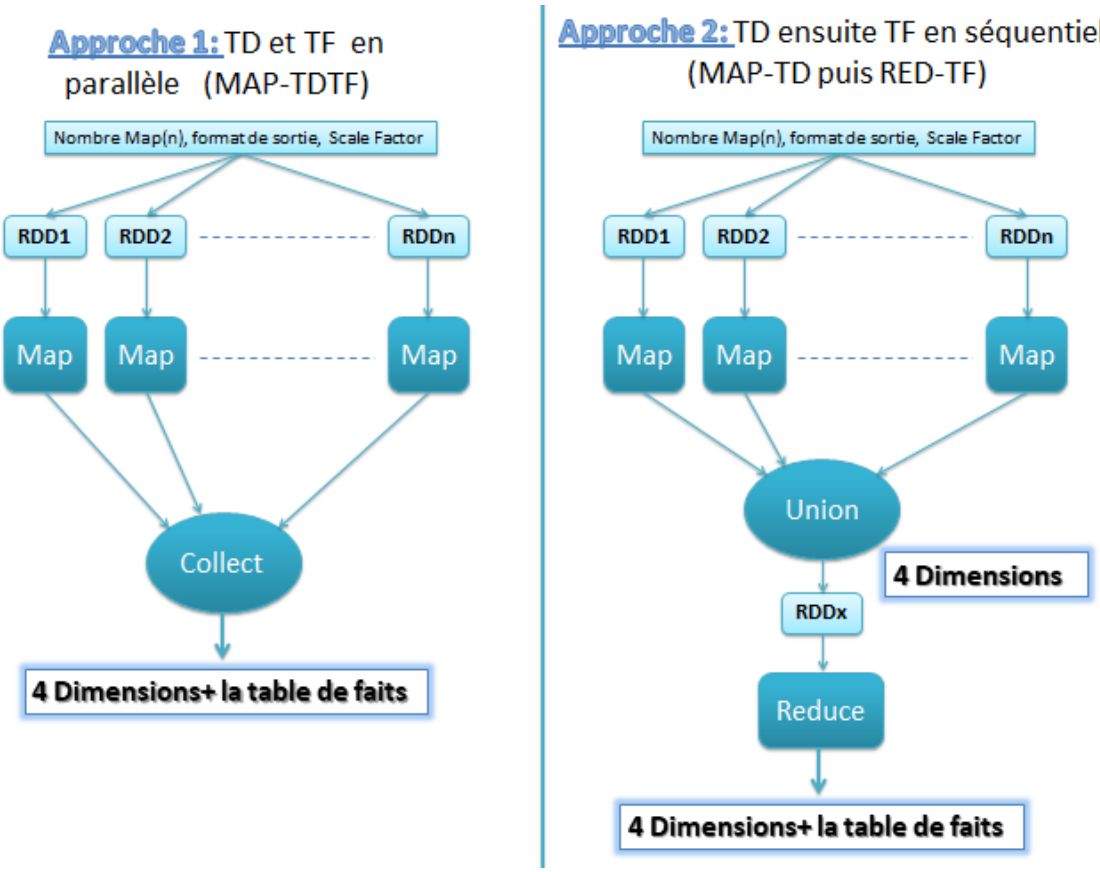

Figure II.7: les approches proposées.

7. La distribution des données

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 deux approches détaillées dans la section V.2 qui se basent sur le paradigme MapReduce.

Concernant le calcul de nombre de Map, il faut d'abord calculer la taille de chaque table(voir la figure II.8) .Apres le calcul de la taille de chaque table, on passe au calcul de nombre de Map nécessaire pour chaqu'une des tables, il faut calculer le coefficient de chaque table en devisant la taille de table sur la somme des taille, puis multiplier le résultat par le nombre de Map (dimension, fait).Sachant que le Nombre_Total_Map est entré par l'utilisateur (le calcul dépend de l'approche).

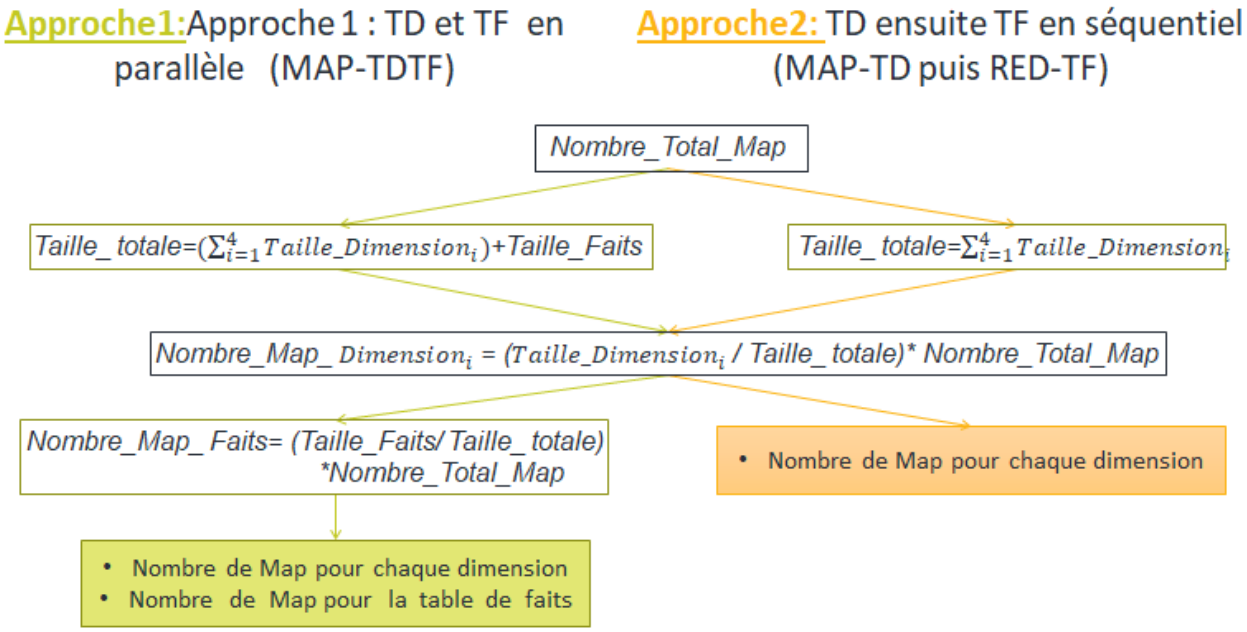

Figure II.8 : calcul de nombre de Map dans les deux approches

8. Le diagramme de cas d'utilisation de l'outil OGGB

Le diagramme de cas d'utilisation représenté dans la figure III.7 illustre un modèle de départ très simpliste auquel nous définissons les fonctionnalités considérant un seul acteur qui est l'analyste.

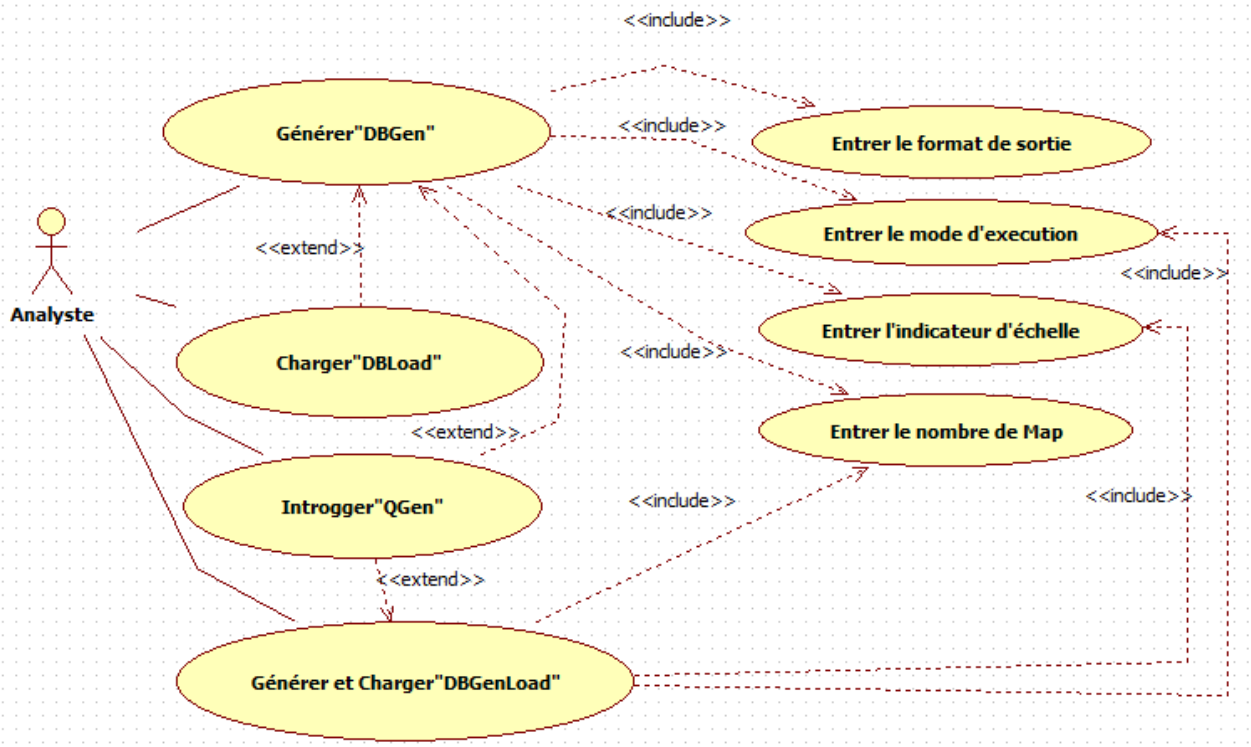

Figure II.9: Diagramme de cas d'utilisation modélisant l'outil OGGB

L'analyste peut utiliser les fonctionnalités suivantes :

- Générer les données soit en utilisant l'outil DBGen ou DBGenLoad qui offrent à l'utilisateur le choix entre les différents modes d'exécutions (local, standalone, cluster) et les différents formats de sortie CSV, TBL, JSON (formats de sortie sauf pour « DBGen »), ainsi la possibilité de prendre en entrée un entier quelconque pour le facteur d'échelle, ils permettent aussi de rentrer le nombre total de Map sachant qu'il doit être inférieur ou égale le nombre des cœurs de la machine utilisée, l'analyste dois rentrer toutes les informations nécessaires mentionnées ci-dessus.
- Charger les données dans un SGBD NoSQL orienté documents « MongoDB ».Cette tache nécessite l'existence des fichiers JSON générés avant par « DBGen »
- Interroger l'entrepôt de données ou les fichiers générés, cette tache nécessite le choix de source de données

Conclusion

A travers ce chapitre, nous avons fait une description de l'outil OGGB ainsi que leurs composants, nous avons aussi mentionnées les caractéristiques communes entre ces composants avec le cas d'utilisation de l'outil. Dans le prochain chapitre, on va entamer la partie implémentation pour mieux comprendre comment nous avons travaillé avec quelques environnements.

Chapitre III: Implémentation

III. Implémentation

Introduction

Après avoir établi une description de notre système, nous passons à l'implémentation de l'application tout en présentant les outils utilisés et l'interface de notre logiciel. Cette implémentation est basée sur une architecture distribuée et réalisée avec le langage de programmation Scala en utilisant l'*IDE* (*Integrated Development Environment)* IntelliJ IDEA et le SGBD NoSQL orienté documents MongoDB.

1. Environnement et outils de développement

Pour implémenter l'outil de génération des données OGGB, nous avons utilisé un environnement de développement intégré (IDE) « IntelliJ IDEA », connu par sa facilité d'utilisation, sa souplesse et sa conception solide[6] comparé à ses homologues. Vu que le choix du langage de programmation s'est porté sur « scala » qui est plus rapide et modérément facile à utiliser. Scala donne accès aux dernières fonctionnalités de Spark. Apache Spark étant écrit en Scala il devient vraiment intéressant avec son langage natif Scala. , nous avons utilisé Scala Plugin et Mongo Plugin ainsi que le SBT comme outil de gestion des dépendances pour gérer la cross-compilation native (capable de compiler facilement vers différentes versions de Scala (2.10, 2.11, 2.12, …).

1.1 Scala

Scala est un langage de programmation multi-paradigme conçu à l'école polytechnique fédérale de lausanne (EPFL) pour exprimer les modèles de programmation courants dans une forme concise et élégante. Son nom vient de l'anglais *Scalable language* qui signifie à peu près « langage adaptable » ou « langage qui peut être mis à l'échelle ». Il peut en effet être vu comme un métalangage[46].

Scala rassemble dans un même langage les paradigmes de programmation orientée objet et de programmation fonctionnelle. Il a été conçu pour exprimer les motifs de programmation courants avec un système de typage sûr. Plusieurs nouvelles constructions innovantes sont désormais à la disposition du programmeur[47] :

- Les types abstraits et la composition par *mixin unifient* les concepts d'objet et de module.
- Le *pattern matching* peut s'appliquer à des hiérarchies de classes et unifie ainsi l'accès aux données, qu'il s'effectue de façon fonctionnelle ou orientée objet. Il simplifie aussi énormément le traitement des arborescences XML.
- Une syntaxe et un système de typage souples permettent de créer des bibliothèques élaborées et des langages spécifiques au domaine (DSL).

En même temps, Scala est compatible avec Java, il y a des spécialistes comme James Gosling, l'inventeur de Java, prédisent d'ailleurs un bel avenir à ce langage. Scala, le langage qui va remplacer Java.

[6] https://www.techopedia.com/definition/7755/intellij-idea

1.2 L'Environnement de Développement Intégré « IntelliJ IDEA »

IntelliJ IDEA est un environnement de développement intégré Java (EDI) pour le développement de logiciels. Il est développé par JetBrains, et il est disponible en tant qu'édition communautaire Apache 2 sous licence et dans une édition commerciale propriétaire. Les deux peuvent être utilisés pour le développement commercial.

L'IDE prévoit l'intégration avec des outils de compilation tels que Grunt, Bower, Gradle et SBT. Il supporte le contrôle de version des systèmes tels que GIT, Mercurial, Perforce et SVN. Les bases de données telles que Microsoft SQL Server, ORACLE, MongoDB et MySQL peuvent être accessible directement depuis l'EDI[48].

1.2.1 SBT

SBT est un outil de build pour la JVM, à l'instar de Maven ou Gradle. À partir d'un projet, il permet, entre autres, de gérer ses dépendances, de compiler, d'exécuter des tests et de publier les artefacts sur des repositories. Son nom voulait à l'origine dire *Simple Build Tool* mais il a été changé à *Scala Build Tool*[46].

La figure III.1 représente le fichier de configuration build.sbt que nous avons utilisé dans notre projet

```
name := "GénérateurSSB_NOSQL"
version := "0.1"
scalaVersion := "2.11.8"

// https://mvnrepository.com/artifact/org.scala-lang/scala-library
libraryDependencies += "org.scala-lang" % "scala-library" % "2.11.8"

// https://mvnrepository.com/artifact/org.apache.spark/spark-core
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.0"
libraryDependencies += "org.apache.spark" %% "spark-yarn" % "2.4.0"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.0"

// https://mvnrepository.com/artifact/org.mongodb.scala/mongo-scala-driver
libraryDependencies += "org.mongodb.scala" %% "mongo-scala-driver" % "2.6.0"

// https://mvnrepository.com/artifact/org.mongodb.spark/mongo-spark-connector
libraryDependencies += "org.mongodb.spark" %% "mongo-spark-connector" % "2.4.0"
```

Figure III.1 : Fichier build.sbt

1.2.2 Scala Plugin

Le plugin Scala est utilisé pour transformer un IDE IntelliJ normale en un environnement de développement Scala très pratique.

Le plugin Scala étend le plugin Java pour ajouter un support pour les projets Scala. Il peut traiter du code Scala, du code mixte Scala et Java, et même du code Java pur. Le plugin prend en charge la

compilation conjointe, ce qui vous permet de mélanger et d'associer librement du code Scala et Java, avec des dépendances dans les deux sens[7].

1.2.3 Mongo Plugin

Cet outil permet d'accéder aux bases de données MongoDB et fournit des opérations *CRUD (Create, Read, Update, Delete)* sur les collections de MongoDB. Ce plugin a été construit pour IDEA en 2016, il intègre les serveurs MongoDB avec l'arborescence base de données / collections, Query Runner et la console Shell[8].

La figure III.2 représente l'explorateur Mongo après l'intégration de Mongo Plugin dans notre projet IntelliJ.

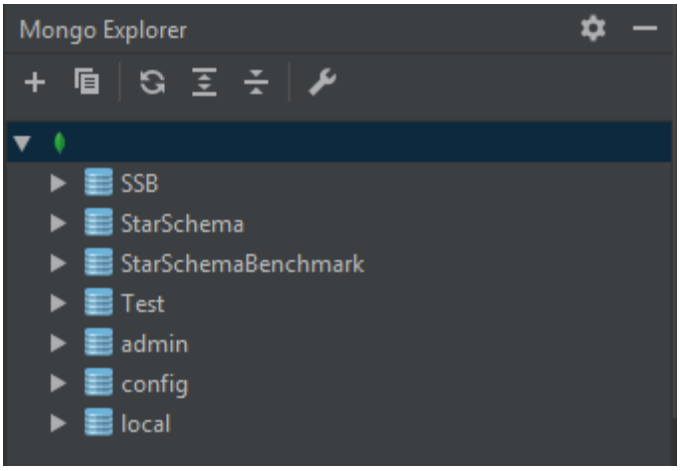

Figure III.2 : Explorateur Mongo

2. Architecture globale de l'outil OGGB

La figure III.3 montre que l'architecture globale de l'outil graphique de génération de benchmark OGGB englobe quatre composants développés dans l'Environnement de Développement Intégré IntelliJ IDEA en utilisant le langage de programmation Scala.

- DBGen prend en entrée le format de sortie, le nombre de Map, le mode d'exécution ainsi que l'indicateur d'échelle. Ce composant crée une instance de SparkContext pour pouvoir se connecter à plusieurs types de gestionnaires de cluster et pour créer un *RDD (Resilient Distributed Dataset)* qui dont le but d'utiliser les fonctions Map et Reduce. Apres la création de RDD DBGen suit le chemin et l'approche 1 ou l'approche 2 pour la génération des données qui seront stocké dans des fichiers de format choisi par l'utilisateur.

- DBGenLoad suit le même processus que DBGen, la différence c'est que DBGenLoad stocke les résultats dans MongoDB au lieu dans des fichiers.

[7] https://docs.gradle.org/current/userguide/scala_plugin.html
[8] https://plugins.jetbrains.com/plugin/7141-mongo-plugin

- QGen prend en entrée soit un entrepôt de données ou des fichiers de format JSON (générés par DBGen); il s'agit donc d'instancier SparkSession qui permet de créer des dataframes qui font partie de l'exécution des requêtes SQL en utilisant SparkSQL.

- DBLoad permet de charger les données générées stockés dans des fichiers de format JSON par DBGen dans MongoDB.

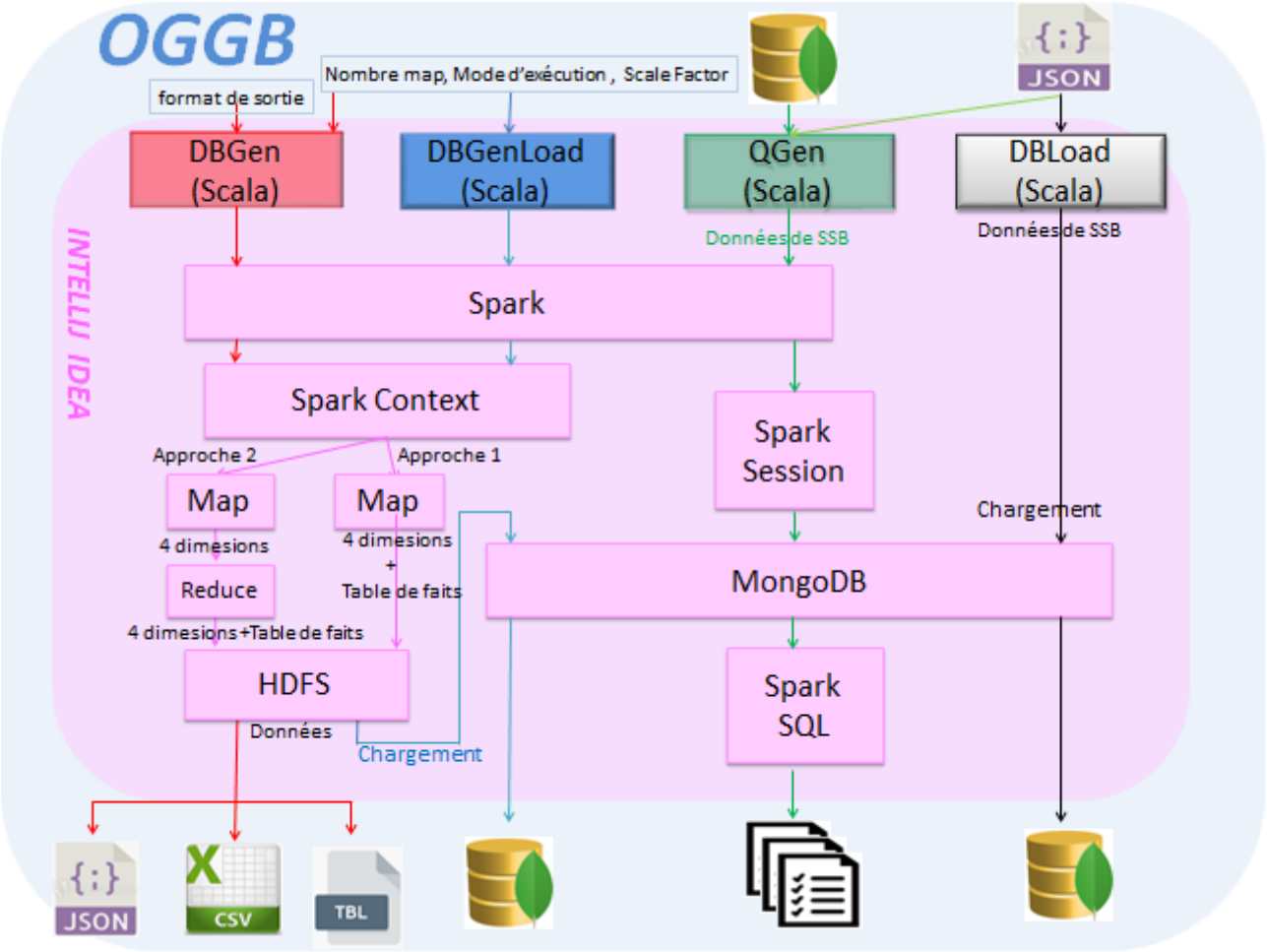

Figure III.3 : Architecture globale de l'outil OGGB

3. Les interfaces de l'outil OGGB

Pour mieux comprendre le fonctionnement de l'application, les figures qui suivent décrivent la chronologie des principales fonctionnalités de l'application.

La figure ci-dessous représente que l'interface contient trois onglets : SSB, Génération et Chargement, Interrogation

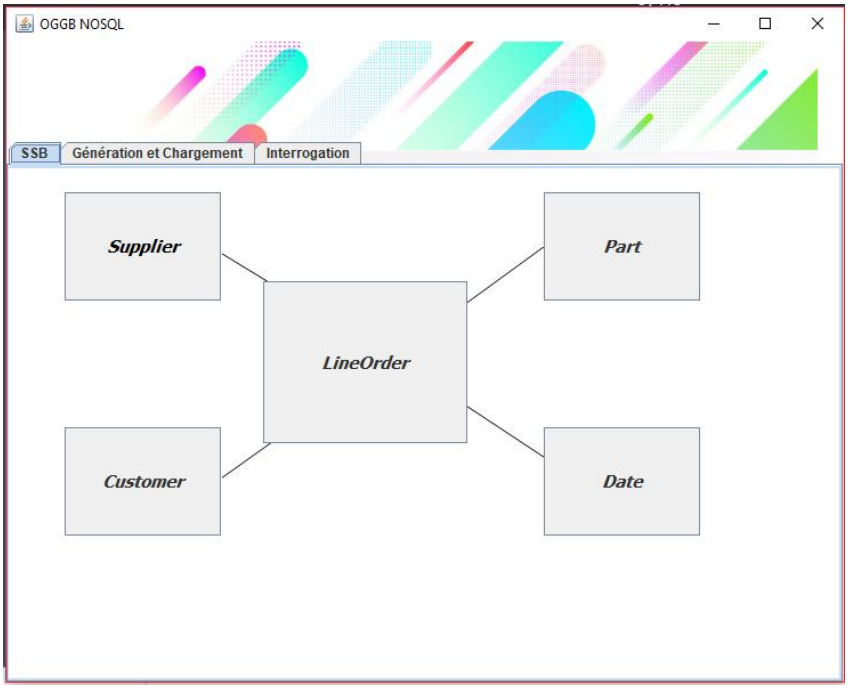

Figure III.4 :L'onglet SSB de l'interface d'OGGB

Le rôle de cet onglet est de fournir les informations nécessaires (les attributs, le type de chaque attribut,…) pour comprendre le modèle SSB. Les boutons sont schématisés comme le modèle *SSB (Star Schema Benchmark)*, il suffit seulement de cliquer sur la table souhaitée puis une autre interface sera affichée qui correspond au nom de la table.

La figure III.5 montre la nouvelle fenêtre ouverte après un clic dans le bouton *Supplier* (les autres tables seront affichées de même manière).

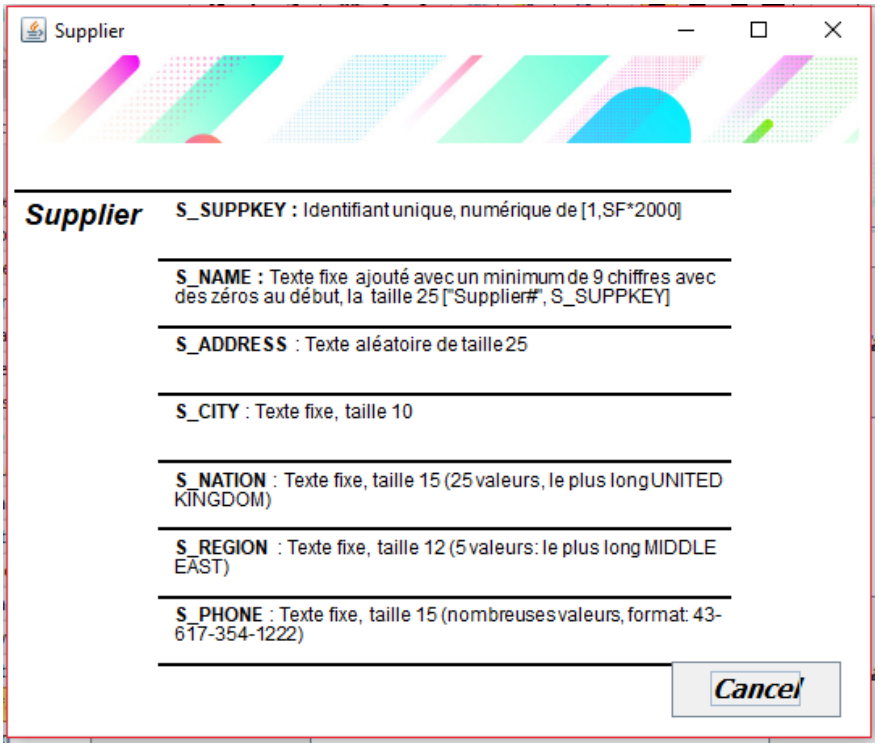

Figure III.5: Détails de la table *Supplier*

Dans l'onglet « Génération et Chargement » (figure III.6), le rectangle gris apparait vide .Si l'utilisateur choisit « Génération puis chargement » ; le bouton Générer sera affiché dans l'interface. Lorsque la génération se termine un message d'information sera affiché avec le bouton Charger. Si l'utilisateur a choisi l'option « Génération et chargement »; la génération des données et le chargement se font de manière automatique et simultané (dans les deux cas les champs doivent être remplis)

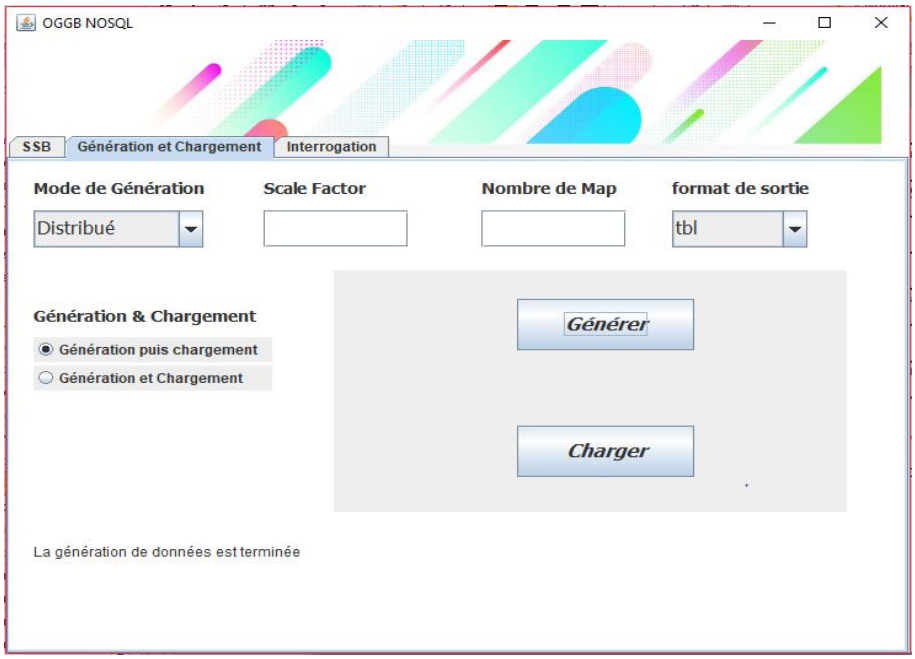

Figure III.6 : L'onglet Génération et Chargement

Concernant le dernier onglet « Interrogation » (figure III.7), il permet d'exécuter une série des requêtes sur les données en précisant la source de données et les valeurs appropriés pour chaque champs de texte (depuis des fichiers JSON ou entrepôt de données dans MongoDB).

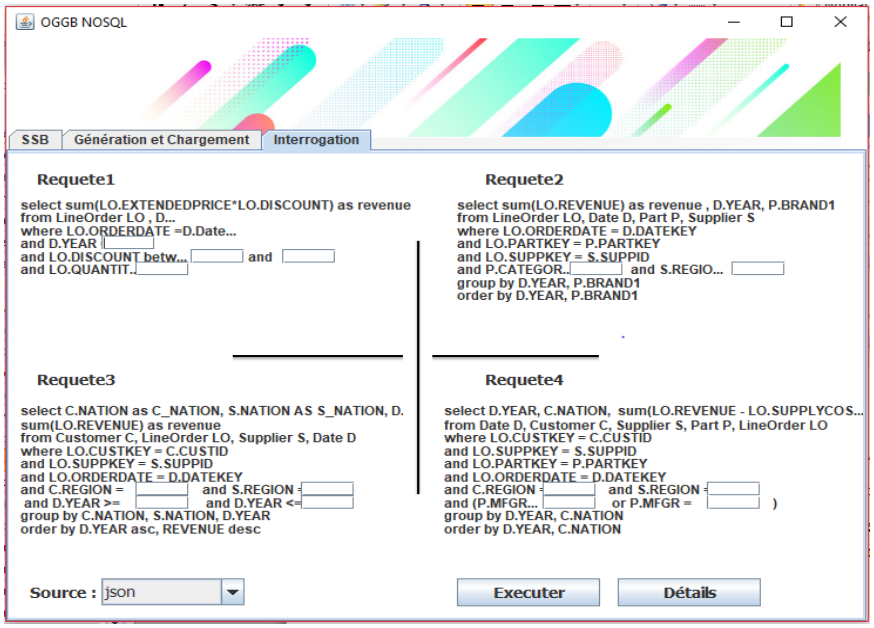

Figure III.7 : L'onglet Interrogation

Apres l'exécution, un nouveau onglet apparait dans l'interface contient les résultats des quatre requêtes (voir la figure III.8)

Figure III.8 : L'onglet ExecutionQGen

Conclusion

Ce chapitre décrit l'environnement distribué ainsi que les structures NoSQL adoptées pour la génération et le stockage des données. Cet environnement complexe intègre plus plateformes qui nous ont permis de mettre en œuvre l'outil OGGB destiné pour la génération de données synthétiques qui serviront pour des benchmarks décisionnels dans un environnement BigData. En effet, nous avons présenté une architecture globale d'OGGB qui intègre tous les outils nécessaires de l'environnement BigData ainsi que les composants OGGB développés. Afin de donner plus de détails sur le prototype développé, nous avons présenté des captures des OGGB.

Plusieurs phases ont été parcourues dans la mise en œuvre d'OGGB; nous résumons le processus de la manière suivante :

- Installation des outils dédiés à l'environnement BigData,
- Configuration de chaque outil parmi eux on mentionne IntelliJ IDEA. Mais puisque Scala donne accès aux fonctionnalités de Spark ; nous avons installé Scala Plugin ainsi que Mongo Plugin sachant que nous avons utilisé SBT pour la gestion des dépendances.
- Développement de l'outil OGGB comprenant cinq modules :
  - DataGenerator : module principal qui fait appel aux quatre autres modules
  - DBGen
  - DBLoad
  - DBGenLoad
  - QGen

Les quatre derniers modules font aussi appel à d'autres modules secondaires dont le but est de répondre au besoin de l'utilisateur (génération, chargement et interrogation).

L'exécution est faite dans les environnements « local », « standalone » et « cluster » avec une variation de l'indicateur d'échelle, du nombre de Map ainsi que le format de sortie.

Le prochain chapitre a est consacré pour une analyse expérimentale dont le but premier est de de disposer de données synthétiques pour des benchmarks décisionnels dans un environnement BigData mais en même temps d'assurer un processus de génération de données qui soit performant.

Chapitre IV: Partie expérimentale

IV. Partie Expérimentale

Introduction

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.

1. Expérimentations

Afin de réaliser nos expérimentations, nous avons mis en place un environnement distribué basé sur Spark et le système de gestion de données NoSQL orienté document MongoDB.

Pour évaluer notre outil de génération de données OGGB, nous avons mis en place trois environnements.

- **Le mode local :** est un environnement composé d'une seule machine. Ce mode n'utilise aucun type de gestionnaire de ressources (comme Yarn) et le job Spark est exécuter sont recourir au gestionnaire de Yarn et en utilisant le multithreading (simuler le traitement parallèle). Ce mode est utilisé pour tester l'exécution d'un programme en éliminant les contrainte liées à l'utilisation du gestionnaire de ressources de Spark (la répartition des traitements et la répartition des données dans les différents nœuds qui peuvent composés un cluster).
- **Le mode Standalone :** est un environnement composé d'une seule machine contenant un (01) master et six (06) esclaves et l'instance Yarn s'exécute en local. Ce mode est utilisé pour simuler un cluster en éliminant les contraintes liées à l'interconnexion des nœuds via un réseau.
- **Mode cluster :** est un environnement composé d'un cluster composé de trois machines dont une a le rôle d'un master et deux autres machines sont des nœuds. Chaque machine est une machine 8 Core avec une mémoire de 8Go et un espace de stockage de 1To.

1.1 Expérimentation1

L'objectif de cette expérimentation est de comparer le volume des données générées selon les trois formats de sorties (CSV, TBL et JSON). Les résultats sont résumés dans le tableau IV-1 et la figure IV.2

	1	**10**	**100**	**1000**
CSV	882Mo	8,85Go	91,2Go	912,3Go
TBL	882Mo	8,85Go	21,2Go	912,3Go
JSON	3,2Go	32,7Go	327,4Go	3To 274Go

Tableau IV-1 : Volumes générés selon les indicateurs d'échelle et les formats de sorties

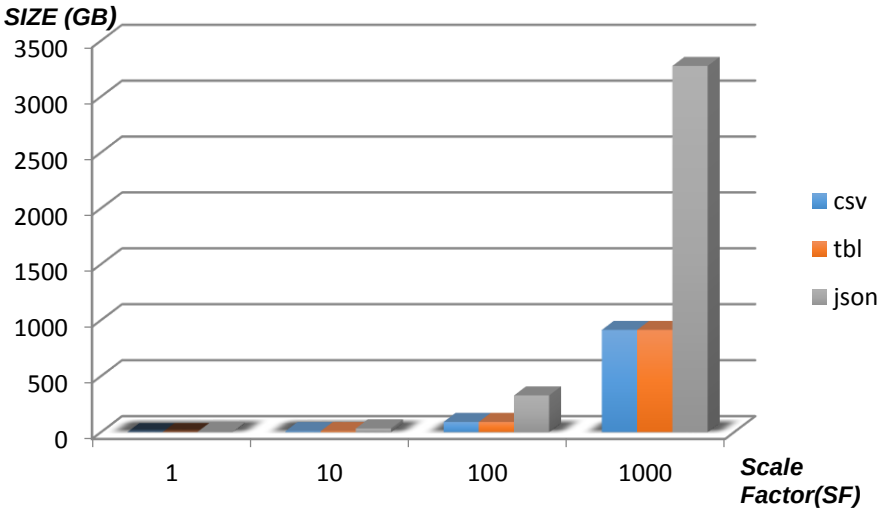

Figure IV.1 : Volumes générés en Gigaoctet

D'après les résultats, on remarque que la taille des fichiers de format TBL est égale à la taille des fichiers CSV, la taille de ces derniers est inférieure à SF. Par contre, la taille des fichiers de format JSON est supérieure à SF. Ces résultats nous permettent de conclure que les fichiers CSV et TBL consomment le même espace de stockage qui est inférieur 3 fois à l'espace alloué après la génération des fichiers JSON.

1.2 Expérimentation 2

Dans cette expérimentation, nous évaluons le temps nécessaire pour la génération des données synthétiques pour chacun des trois environnements dont le but est de faire une étude comparative entre les deux approches.

- **Mode cluster**

Dans le tableau IV-2, nous reportons le temps nécessaire pour générer les données dans le mode cluster, en utilisant différents indicateurs d'échelle pour les trois formats de sortie et les deux approches.

		Format De Sortie		
	SF	**CSV**	**TBL**	**JSON**
Approche 1	**1**	51s	51s	203s
	10	367s	367s	2497s
	100	3120s	3120s	5460s
	1000	13920s	13920s	18660s
Approche 2	**1**	51s	51s	184s
	10	358s	359s	2280s
	100	2820s	2823s	3498s
	1000	12420s	12360s	16680s

Tableau IV-2 : Temps de génération de données en mode cluster

Les résultats sont représentés dans la figure IV.3, la figure IV.4 et la figure IV.5

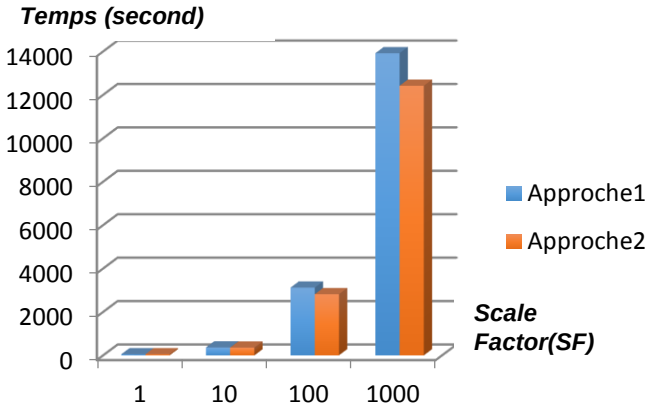

Figure IV.1 : Temps de génération des fichiers TBL en mode cluster

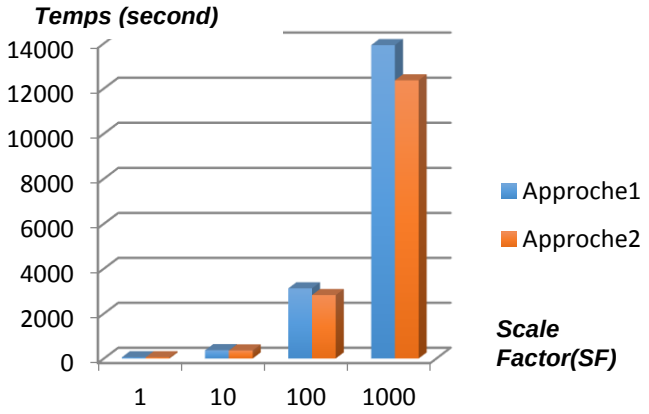

Figure IV.2 : Temps de génération des fichiers CSV en mode cluster

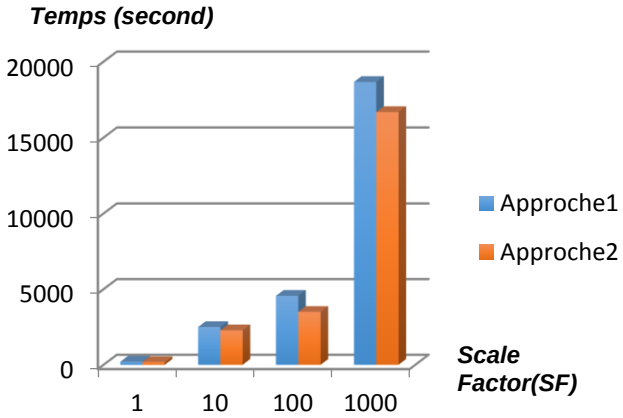

Figure IV.3: Temps de génération des fichiers JSON en mode cluster

On remarque que dans le mode cluster et avec les différents formats de sortie, la génération des données en utilisant l'approche 2 nécessite moins de temps que lorsqu'on génère les données avec l'approche 1. Ceci peut être expliqué par le fait que la répartition du nombre de Map sur les dimensions seulement permet de minimiser le temps car la table de faits qui a une grande taille va générer les données dans la phase Reduce donc la phase Map est dédiée pour les dimensions et la phase Reduce est dédiée pour la table de faits, ce qui assure une meilleure répartition de charge sur les différentes tâches du job OGGB. Par contre, dans l'approche 1 toutes les données des cinq tables seront générées dans la phase Map (divisée entre eux) qui rend le programme un peu lourd, qui ne donne pas une meilleure performance en termes de temps d'exécution et ceci est dû à une répartition de chargé non équitable entre les différentes tâches du job OGGB.

- **Mode standalone**

Dans le tableau IV-3, nous reportons le temps nécessaire pour générer les données dans le mode standalone en utilisant différents indicateurs d'échelle pour les trois formats de sortie et les deux approches.

		format De Sortie		
	SF	**CSV**	**TBL**	**JSON**
Approche 1	**1**	84s	66s	493s
	10	558s	546s	3120s
	100	4380s	4380s	6000s
	1000	16140s	15900s	22020s
Approche 2	**1**	66s	66s	378s
	10	546s	558s	2965s
	100	3840s	3840s	5640s
	1000	15120s	15000s	21480s

Tableau IV-3 : Temps de génération de données en le mode standalone

Les résultats sont représentés dans la figure IV.6, la figure IV.7 et la figure IV.8

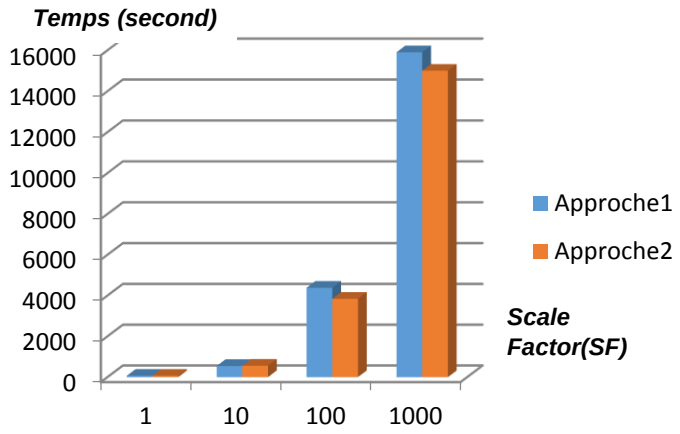

Figure IV.4 : Temps de génération des fichiers CSV en mode standalone

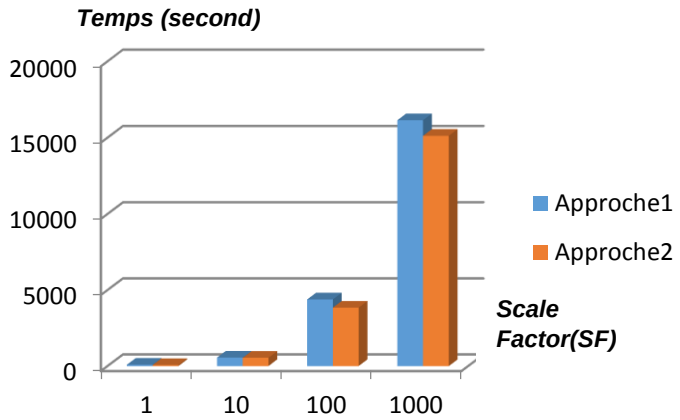

Figure IV.5 : Temps de génération des fichiers TBL en mode standalone.

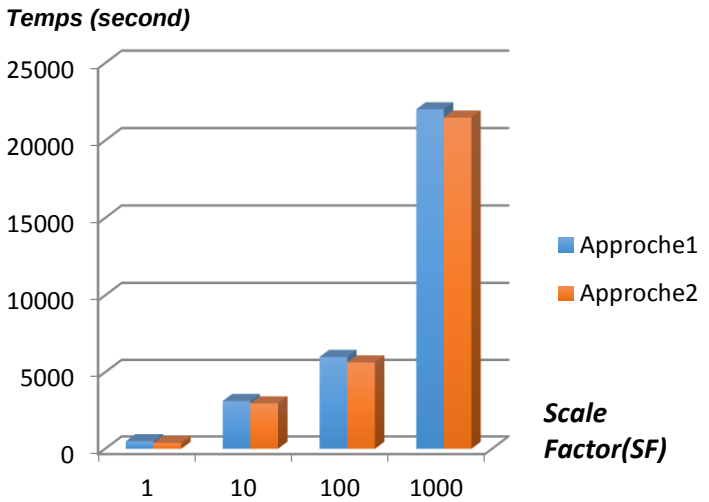

Figure IV.6 : Temps de génération des fichiers JSON en mode standalone.

Les figures montrent que la génération des données en utilisant l'approche 2 nécessite moins de temps que lorsqu'on génère avec l'approche 1 dans le mode standalone, et avec les différents formats de sortie. Ceci confirme, encore une fois, que l'approche qui sépare la génération des données des dimensions et les données de la table de fait est nettement plus efficace et performante par rapport à l'approche qui procède en une seule phase (Map) pour générer toutes les données des tables de l'entrepôt de données.

- **Mode local**

Le tableau IV-4 représente le temps nécessaire pour générer les données dans le mode local en utilisant différents indicateurs d'échelle pour les trois formats de sortie et les deux approches.

		format	De	Sortie
	SF	**CSV**	**TBL**	**JSON**
Approche 1	**1**	60s	60s	286s
	10	529s	529s	3000s
	100	5100s	5100s	6660s
	1000	16860s	16740s	26520s
Approche 2	**1**	60s	60s	378s
	10	533s	532s	3256s
	100	5100s	5100s	7020s
	1000	16920s	16920s	21480s

Tableau IV-4 : Temps de génération de données dans en mode local

Les résultats sont représentés dans la figure IV.9, la figure IV.10 et la figure IV.11

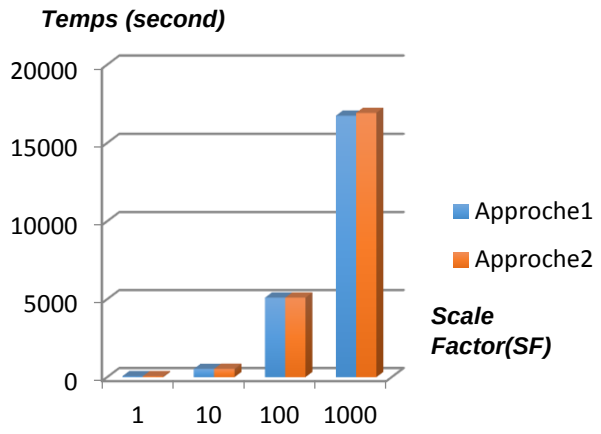

Figure IV.8 : Temps de génération des fichiers CSV en mode local

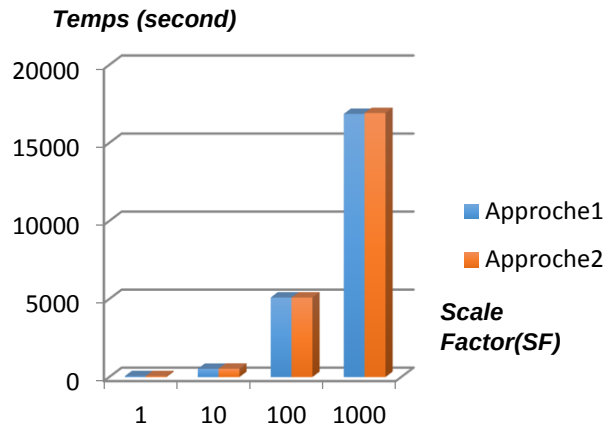

Figure IV.7 : Temps de génération des fichiers TBL en mode local

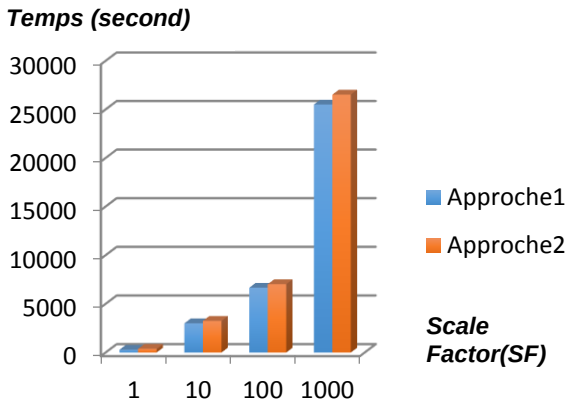

Figure IV.9 : Temps de génération des fichiers JSON en mode local

Les figures illustrent que la génération des données en utilisant l'approche 1 nécessite moins de temps que lorsqu'on génère avec l'approche 2 dans le mode local, et avec les différents formats de sortie revient au mode local, la taille de table de faits n'influe pas car dans le mode local est basé sur le multithreading qui permet le programme à gérer ses ressources.

1.3 Expérimentation 3

Dans cette expérimentation il est question d'évaluer le temps de génération pour les trois modes d'exécution. Notons que nous avons utilisé différents indicateurs d'échelle pour les différentes configurations.

- **Approche1**

Le Tableau IV-5 résume le temps de génération des données en utilisant l'approche 1 avec les trois formats de sortie et en variant le facteur d'échelle.

	Mode d'exécution	Scale Factor			
		1	10	100	1000
CSV	**local**	60s	529s	5100s	16860s
	standalone	84s	558s	4380s	16140s
	cluster	51s	367s	3120s	13920s
TBL	**local**	60s	529s	5100s	16740s
	standalone	66s	546s	4380s	15900s
	cluster	51s	367s	3120s	13920s
JSON	**local**	286s	3000s	6660s	25500s
	standalone	493s	3120s	6000s	22020s
	cluster	203s	2497s	4560s	18660s

Tableau IV-5 : Temps de génération des données avec l'approche 1

Les résultats sont aussi représentés dans la figure IV.12, la figure IV.13 et la figure IV.14.

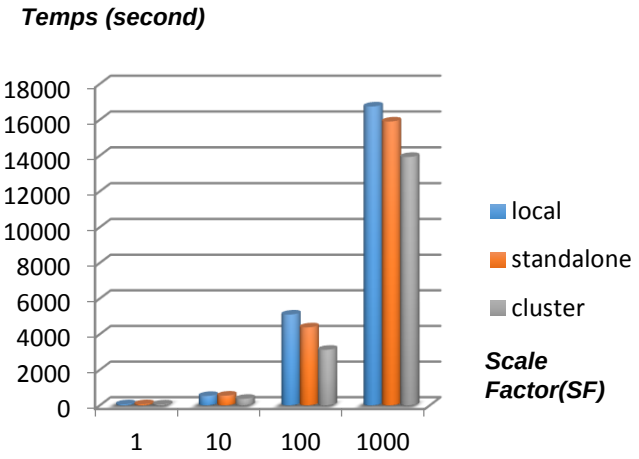

Figure IV.10 : Temps de génération des fichiers CSV en utilisant l'approche 1

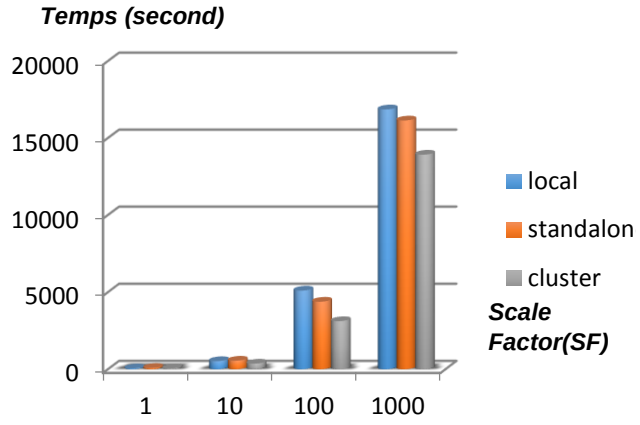

Figure IV.11 : Temps de génération des fichiers TBL en utilisant l'approche 1

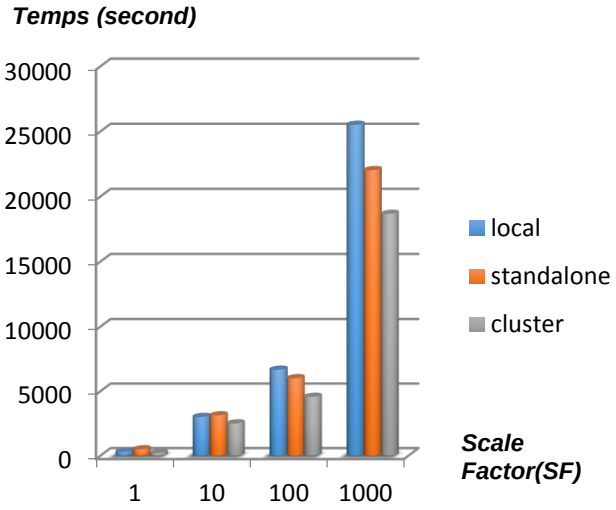

Figure IV.12: Temps de génération des fichiers JSON en utilisant l'approche 1

Dans les figures représentées ci-dessus on remarque que le temps de génération des fichiers CSV, TBL et JSON en utilisant l'approche 1 est meilleur dans le mode cluster que dans le mode local et standalone. Ainsi on remarque que le temps de génération dans le mode standalone est meilleur que le mode local.

- **Approche2**

Le Tableau IV-6 résume le temps de génération des données en utilisant l'approche 2 avec les trois formats de sortie et en variant le facteur d'échelle.

	Mode d'exécution	Scale Factor			
		1	10	100	1000
CSV	**Local**	60s	533s	5100s	16920s
	Standalone	66s	546s	3840s	15120s
	Cluster	51s	358s	2820s	12420s
TBL	**local**	60s	532s	5100s	16920s
	standalone	66s	558s	3840s	15000s
	cluster	51s	359s	2823s	12360s
JSON	**local**	378s	3256s	7020s	26520s
	standalone	378s	2965s	5640s	21480s
	cluster	184s	2280s	3498s	16680s

Tableau IV-6 : Temps de génération des données avec l'approche 2

Les résultats sont aussi représentés dans la figure V.12, la figure V.13 et la figure V.14.

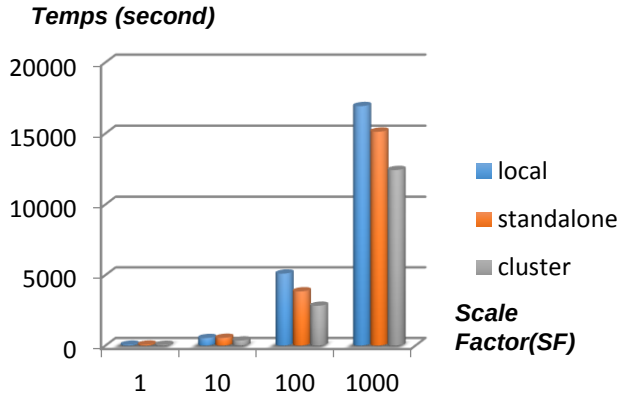

Figure IV.13:Temps de génération des fichiers CSV en utilisant l'approche 2

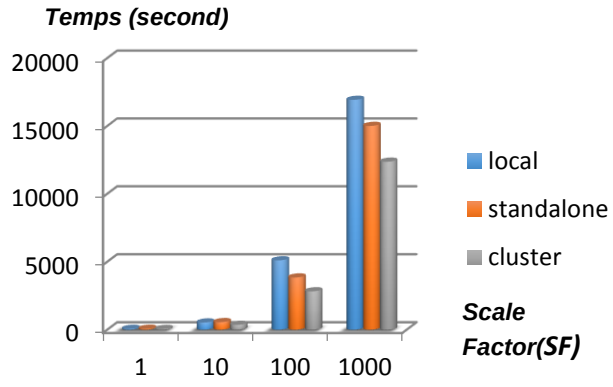

Figure IV.14 : Temps de génération des fichiers TBL en utilisant l'approche 2

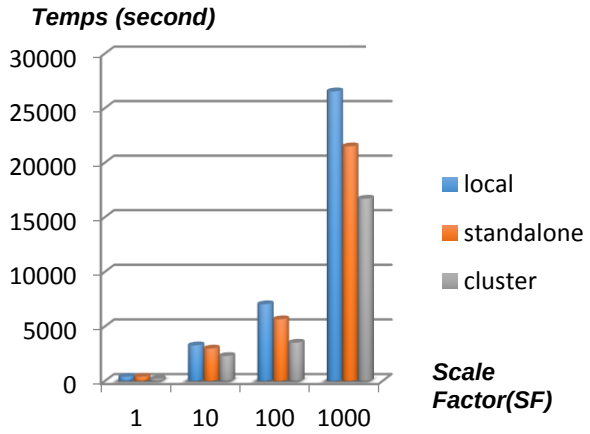

Figure IV.15 : Temps de génération des fichiers JSON en utilisant l'approche 2

D'après les figures ci-dessus on remarque que le temps de génération des fichiers CSV, TBL et JSON en utilisant l'approche 2 est meilleur dans le mode cluster que dans le mode local et standalone, ainsi on remarque que le temps de génération dans le mode standalone est meilleur que le mode local.

1.4 Expérimentation4

Cette expérimentation est pour but de montrer que le nombre des machines dans le mode distribué influe dans le temps de génération de données.

Dans cette expérimentation nous avons fait une comparaison entre deux cas dans le mode cluster sachant que le premier cas est composé de trois machines et le deuxièmes cas est composé de cinq machines, dans le deux cas une machine joue le rôle d'un master et deux autres machines sont des nœuds. Le test est fait en utilisant l'approche 2 qui est plus performante dans le mode distribué.

Les résultats sont résumés dans le tableau IV-7 et la figure IV.2

	1	10	100	1000
3 machines	184s	2280s	3498s	16680s
5 machines	159s	1562s	2207s	14580s

Tableau IV-7:Temps de génération de données avec une variation d'indicateur d'échelle

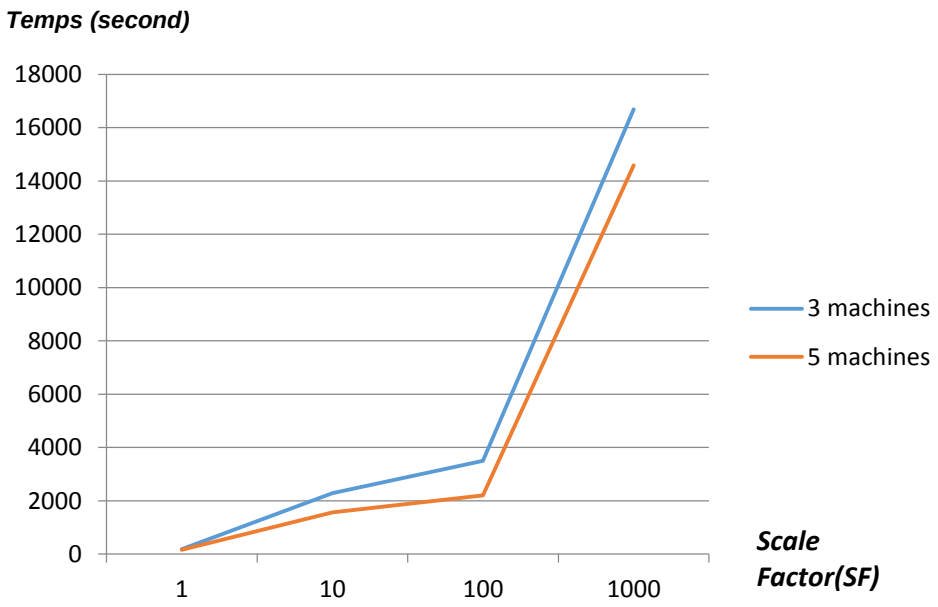

Figure IV.16 : Représentation de temps de génération avec différents valeurs d'indicateur d'échelle

D'après la figure IV.18 on remarque que temps de génération avec 5 machines est inférieur qui rend le premier cas (utilisation de 5 machine) un cas plus performant et efficace par rapport au temps de

génération avec 3 machines. On conclure que plus le nombre des machines augmente, plus le temps diminue.

2. Résultats

Les résultats obtenus après une série d'expérimentations, nous permettent d'aboutir à la conclusion suivante :

- Les résultats de l'expérimentation 2 montre que l'approche 2 (génération de table de fait dans la phase Reduce) nécessite moins de temps que l'approche1 dans l'architecture distribuée (mode standalone et cluster). La répartition du nombre de Map sur les dimensions seulement donne une meilleure performance en termes de temps d'exécution. En revanche, en mode local, l'approche 1 montre de meilleures performances.
- Le mode distribué (3 nœuds) nécessite moins de temps que les deux autres modes (expérimentation 3). Cela revient à la l'architecture distribuée qui permet à chaque nœud de fonctionner plus efficacement, ce qui augmente les performances
- La génération des fichiers TBL et CSV affirme que les deux consomment le même espace de stockage et nécessitent, également, le même temps d'exécution qui est inférieur à celui que nécessite la génération des fichiers JSON. Ceci est dû au fait que les documents JSON nécessitent l'écriture des noms des champs (*key*) qui doivent précéder les données (*value*)
- L'augmentation de nombre des machines provoque une diminution dans le temps de génération, cela montre que la distribution des données et des traitements améliore les performances d'un processus traitant des volumes importants de données.

Conclusion générale et perspective

Conclusion Générale et Perspective

Conclusion

Le présent mémoire traite sur la réalisation d'un outil graphique de génération de benchmark NoSQL. L'émergence des données massives et leur arrivée dans les systèmes d'information des organisations constituent la motivation première de nos travaux sachant que les solutions existantes sont développées et optimisées pour les bases de données relationnelles et pour une utilisation sur une seule machine.

La première partie de cette mémoire a été consacrée à la définition des notions fondamentales ayant rapport avec nos travaux de recherche. Ainsi, nous avons fait une synthèse sur les concepts clefs des systèmes décisionnels, les entrepôts de données, la modélisation dimensionnelle. Comme nos travaux se situent dans des environnements caractérisés par des données massives et un traitement distribué, nous avons donc exposé les notions élémentaires telles que les BigData, avec toutes leurs caractéristiques et leurs enjeux technologiques. Nous avons, également, présenté le paradigme MapReduce avec son processus de traitement et son architecture ainsi que l'environnement Hadoop d'Apache, le Framework Spark construit à la base de Hadoop MapReduce. Par ailleurs, nous avons décrit les nouvelles solutions NoSQL pour les systèmes distribués. Nous avons terminé cette partie réservée aux fondamentaux de notre travail, par un exposé des bancs d'essai décisionnels TPC et SSB constituant le vif du sujet. Enfin, nous avons passé en revue la littérature dans ce domaine où nous avons étudié les contributions présentées dans les deux domaines à savoir les systèmes décisionnels et le BigData.

Grâce à l'ensemble des concepts exposés, nous avons présenté, dans la deuxième partie, notre outil baptisé OGGB. OGGB est un outil graphique de génération de données_synthétiques. Il offre l'opportunité de générer les données selon le modèle conceptuel de l'entrepôt de données en étoile « *SSB*» (*Star Schema Benchmark*). Lors de la génération des données, plusieurs aspects sont pris en charge par cet outil, tels que le format des données et le volume de données générées. Les données générées doivent être stockées dans un SGBD NoSQL orienté documents à savoir MongoDB. Pour ce faire, nous avons utilisé des règles de transformation permettant de passer depuis un modèle conceptuel multidimensionnel (modèle en étoile MDE) vers un modèle logique NoSQL (orienté documents MLOD*)*. L'outil graphique de génération de benchmark « OGGB » englobe quatre composants à savoir DBGen, DBLoad, DBGenLoad et QGen sachant que ces composants permettent à l'utilisateur de générer, charger et interroger les données ainsi générées.

Le chapitre trois concerne le développement de l'outil OGGB, nous avons réalisé un programme qui contient cinq modules principaux à savoir le module « DataGenerator » qui fait appels aux quatre autres modules « DBGen », « DBLoad », « DBGenLoad » et « QGen ». Ces derniers font aussi des appels à

d'autres modules. L'exécution a été faite dans les environnements « local, standalone et cluster » avec une variation de valeur de l'indicateur d'échelle, le nombre de Map et le format de sortie.

En collaboration avec le centre de calcul, à travers le chapitre quatre, nous avons fait une étude profonde en utilisant un environnement distribué basé sur Spark afin de trouver une approche qui réponds à nos besoins d'une part, et minimise le temps de génération de données d'autre part.

La réalisation de ce projet de fin de cycle nous a permis d'acquérir une bonne expérience dans le domaine des systèmes décisionnels, plus précisément les systèmes d'entreposage de données.
L'apport du projet est considérable sur le plan pratique, car il nous a permis d'acquérir d'avantage de connaissances sur les techniques de configuration et programmation.

Toutes les tâches de notre projet nous ont permis d'enrichir nos expériences dans l'organisation, la démarche et la cohérence entre les déférentes parties du projet et aussi, de nous familiariser avec plusieurs outils tels que Spark et MongoDB.

Perspectives

Comme perspectives, nous avons étudié plusieurs idées pour la continuité de ce travail que nous résumons de la manière suivante :

1) Les modèles TPC-H et TPC-DS. Les deux modèles sont parmi les modèles conceptuels des entrepôts de données les plus utilisés, cela rend l'outil universel, enrichit par plusieurs modèles conceptuels.

2) Les modèles NoSQL orienté colonnes et orienté graphes.

3) Les conversions intra-modèle en orienté documents, orienté colonnes et orienté graphes. 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.

4) les conversions inter-modèles de d'un modèle NoSQL vers un autre modèle 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.

5) Augmentation de nombre des nœuds et de volume de données à générées.

Références Bibliographiques

[1] C. Favre *et al.*, "Les entrepôts de données pour les nuls... ou pas!," *2ème Atelier aIde à la Décision à tous les Etages (EGC/AIDE 13)*, p. 18, 2013.

[2] M. L. Chouder, S. Rizzi, and R. Chalal, "EXODUS: Exploratory OLAP over Document Stores," *Inf. Syst.*, vol. 79, pp. 44–57, 2019.

[3] L. Notes, I. Manolescu, P. Rigaux, M. Rousset, and P. Senellart, "Introduction aux systèmes NoSQL (Not Only SQL)," 2013.

[4] F. De Marchi, "Conception des bases de données relationnelles," *Sci. York*, pp. 1–29, 2009.

[5] D. Leonard, J. F. Rayport, H. Courtney, J. Kirkland, and C. C. Markides, "HarvardBusinessReview TO DIVERSIFY OR NOT TO DIVERSIFY WHEN CONSULTANTS AND CLIENTS CLASH."

[6] M. Chevalier *et al.*, "Entrepôts de données multidimensionnelles NoSQL To cite this version : HAL Id : hal-01360873," 2016.

[7] M. Delurey, "HOW THE DATA LAKE."

[8] "Introduction au domaine du décisionnel et aux Table des," 2016.

[9] H. Hashem and B. Modélisation, "Modélisation intégratrice du traitement BigData Hadi Hashem To cite this version : HAL Id : tel-01378609," 2016.

[10] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," pp. 469–475.

[11] A. P. Girolamo, "NoSQL Etat de l'art et benchmark," *2013*, 2013.

[12] U. A. M. De Béjaïa, "Mémoire de Fin de Cycle Mise en Oeuvre d ' un Entrepôt de Données sous Hadoop Remerciements."

[13] G. Faiza, "Ghozzi Faiza To cite this version : HAL Id : tel-00549421," 2010.

[14] P. Nous and L. Sid, "Systèmes d ' Information Décisionnels."

[15] N. Elsa, "Entrepôts de données," 2017.

[16] B. Espinasse and P. Bellot, "Introduction au Big Data - Opportunités , stockage et analyse des mégadonnées Introduction au Big Data Opportunités , stockage et analyse des mégadonnées," vol. 33, no. 0, 2017.

[17] B. Espinasse, “1 – I n t r o d u c t i o n a u x s y s t è m e s,” no. 5, 2015.

[18] D. E. N. Sciences, “Vers un nouveau modèle de stockage et d‘accès aux données dans les Big Data et les Cloud Computing,” 2018.

[19] E. Fromm, “Phase I : la composition urbaine,” no. 1.

[20] K. Emmanuel, “Master Ii Recherche,” 2012.

[21] M. Bala and Z. Alimazighi, “Modélisation de processus ETL dans un modèle MapReduce,” *Conférence Maghrébine sur les Avancées des Systèmes Décisionnels*, no. October 2014, pp. 1–12, 2013.

[22] S. Genaud, “Map-Reduce : un cadre de programmation parallèle pour l ’ analyse de grandes données Traitement de données distribuées.”

[23] P. Nerzic, “Outils Hadoop pour le BigData Table des matières,” 2018.

[24] T. White, *Hadoop : The Definitive Guide*. .

[25] T. D. Guide, “The Definitive Guide.”

[26] B. Alattar and N. M. Norwawi, “A personalized search engine based on correlation clustering method,” *J. Theor. Appl. Inf. Technol.*, vol. 93, no. 2, pp. 345–352, 2016.

[27] E. Malki, “Open Archive TOULOUSE Archive Ouverte (OATAO) Document-oriented Models for Data Warehouses NoSQL Document-oriented for Data Warehouses,” vol. 2016, no. April, 2016.

[28] F. Abdelhedi, “Open Archive TOULOUSE Archive Ouverte (OATAO) Processus de transformation MDA d ’ un schéma conceptuel de données en un schéma logique NoSQL,” vol. 2016, no. May, 2016.

[29] Z. BENALLAL and H. TAHRAOUI, “Etude comparative des bases de donnéesNoSQL :MongoDb, CouchBase, Cassandra, HBase, Redis, OrientDB,” pp. 2015–2016, 2016.

[30] W. Paper, “Les quatre piliers d ’ une solution de gestion des Big Data Table des Matières.”

[31] E. Stattner, “Bases de Données : Nouveaux paradigmes.”

[32] P. O. Neil, B. O. Neil, and X. Chen, “Star Schema Benchmark,” 2009.

[33] D. E. M. E. T. D. Aerotechnique, “Données de tests non fonctionnels de l ’ ombre à la déployer une base de données,” 2017.

[34] Á. Rocha, A. M. Correia, S. Costanzo, and L. P. Reis, “New contributions in information systems and technologies,” *Adv. Intell. Syst. Comput.*, vol. 354, pp. 619–628, 2015.

[35] M. Barata, J. Bernardino, and P. Furtado, “An Overview of Decision Support Benchmarks : TPC-DS , TPC-H and SSB,” pp. 619–620, 2015.

[36] J. Darmont, F. Bentayeb, and O. Boussaïd, “Conception d’un banc d’essais d\’ecisionnel,” 2007.

[37] P. O. Neil, B. O. Neil, and X. Chen, “The Star Schema Benchmark (SSB),” no. January, pp. 1–10, 2007.

[38] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, “Entrepôts de données multidimensionnelles NoSQL,” *Eda*, vol. d, no. 1996, 2015.

[39] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, “Un benchmark enrichi pour l ’ évaluation des entrepôts de données noSQL volumineuses et variables 1 Introduction,” 2004.

[40] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking Cloud Serving Systems with YCSB.”

[41] A. Ghazal *et al.*, “BigBench : Towards an Industry Standard Benchmark for Big Data Analytics,” 2013.

[42] M. Iu and W. Zwaenepoel, “HadoopToSQL a MapReduce Query Optimizer,” pp. 251–264, 2010.

[43] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang, “YSmart : Yet Another SQL-to-MapReduce Translator,” no. May 2014, 2011.

[44] R. Moussa, “TPC-H benchmarking of Pig Latin on a Hadoop cluster Massive Data Analytics in the Cloud : TPC-H Experience on Hadoop Clusters,” no. June 2012, 2017.

[45] T. P. Csv, “Tp1 : csv, xml, json,” pp. 1–6, 2018.

[46] N. Grosshans and S. Schwoon, “Projet programmation 2 Introduction à Scala,” pp. 1–17, 2017.

[47] M. Odersky, “Scala par l ’ exemple.”

[48] I. I. Handbook, “IntelliJ IDEA Handbook.”