Avantages du système de fichiers HDFS en termes de performances sur les entrepôts de données de type ROLAP

par

Mounir BOUSSAFSAF

essai présenté au CeFTI
en vue de l'obtention du grade de maître en génie logiciel
(maîtrise en génie logiciel incluant un cheminement de type cours en génie logiciel)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, février 2016

Sommaire

De nos jours, les entreprises récoltent de plus en plus de données de sources différentes pour construire une mine d'informations dans le but d'essayer de comprendre le comportement et le besoin de leurs clients et utilisateurs. D'autres utilisent ces données pour aligner la vision de l'entreprise et pour faciliter la prise de décisions.

Plusieurs solutions basés sur les entrepôts de données de type ROLAP existent sur le marché où les données sont stockées principalement en mode relationnel et sous forme de schéma étoile avec des tables de dimension et de fait. Ces entrepôts permettent une multitude d'options d'analyses comme : le traitement analytique en ligne *OLAP*, les requêtes SQL, les rapports ainsi que le forage des données.

Les problèmes de lenteur dans les entrepôts de données *ROLAP* commencent à manifester dès que la quantité de données dans les tables *faits et dimensions* commence à grandir.

Aujourd'hui dans le domaine informatique, on parle beaucoup du « B*ig Data* » et du fameux « *Framework* » de calcul « *Hadoop »*. Ce dernier est conçu principalement pour des calculs distribués de type « *batch* » en mode « *offline* » qui ne nécessite pas un résultat dans l'immédiat.

Le but de l'essai est de concevoir et d'implémenter une solution basée sur *Hadoop* afin d'interroger des données en mode SQL depuis un entrepôt de données *ROLAP* stocké sur un système de fichier de type « *HDFS* ». Le but est de voir si on peut tirer un avantage en termes de performances en utilisant l'écosystème de *Hadoop*.

L'approche est de nature comparative entre *MySQL*, *Microsoft SQL Server* et *HDFS* avec le test de plusieurs outils (*Hive, Impala, Presto et Spark SQL*).

Les conditions matérielles de tests (un seul serveur avec l'usage de la virtualisation) sont identiques et trois entrepôts de données de taille petite (Microsoft adventureworksdw2012), moyenne (1000 fois la petite) et grande (10000 fois la petite) sont interrogés en mode interactif dans chaque système avec plusieurs types de requêtes *SQL*.

Les mesures et les résultats de cette comparaison ont démontré une performance intéressante lorsqu'il s'agit d'un grand entrepôt de données avec l'usage des outils tel qu'*Impala* et *Presto* comparativement à *MySQL* et *SQL Server*.

Ces résultats peuvent être utiles dans la conception des architectures de solutions d'entreposage de grands entrepôts de données. La technique avantageuse est le stockage de données de type *ROLAP* sur un système de fichier *HDFS* avec l'interrogation de ces données en temps réel avec *Impala* ou *Presto*.

Cet essai pourra être considéré comme un début de solution aux problèmes liés aux performances des grands entrepôts de données *ROLAP*. Cette analyse pourra être étendue dans d'autres travaux futurs avec d'autres conditions comme l'usage d'un « *Cluster* » de serveurs qui pourra être implémenté dans une infonuagique avec des tailles de données plus importantes (voir des téraoctets).

Remerciements

Je tiens à remercier chaleureusement Monsieur Évariste Valéry Bévo Wandji le directeur académique de l'essai, ainsi que Monsieur Joël Quimper le directeur professionnel pour leurs orientations et conseils pour aboutir à ce travail.

Je remercie également, Monsieur Claude Cardinal, directeur adjoint au CeFTI, pour le rôle bénéfique qu'il joue pour offrir une formation de qualité à l'Université de Sherbrooke ainsi que pour sa minutieuse coordination avec mes directeurs.

Finalement, j'aimerais adresser mes remerciements à ma famille, spécialement mes parents, ma femme et ma fille pour leurs encouragements et leurs soutiens durant ma formation.

Table des matières

Liste des tableaux

Liste des figures

Glossaire

Apache	Une organisation crée en 1999 à but non lucratif qui développe des logiciels « open source » sous la licence Apache
Big Data	Expression anglophone pour désigner un environnement de mégadonnées
BigTable	Un système de gestion de base de données compressée et orientée colonnes, propriétaire à Google
Bottom Up	Conception et réalisation d'un projet module par module avec une approche ascendante
Cluster	Un regroupement d'ordinateurs (nœuds)
Data Mart	Un mini-entrepôt de données destiné à un département spécifique
Data Warehouse	Désigne une base de données utilisée pour collecter, ordonner, journaliser et stocker des informations provenant de bases de données opérationnelles
GoogleFS	Google File System, un système de fichier distribué propriétaire à Google
Hadoop	Une implémentation « open source » en Java du MapReduce distribué par la fondation Apache
Hive	Une infrastructure d'entrepôt de données qui utilise SQL pour interroger des données sur HDFS
Impala	Un projet libre qui utilise SQL pour interroger des données sur HDFS en utilisant des traitements parallèles massifs MPP
MapReduce	Un modèle de programmation popularisé par Google pour des calculs parallèles
Parquet	Apache Parquet est un format de stockage de type colonne, disponible pour les projets qui se basent sur Hadoop

Scalability	Échelonnage
SCD type 1	Changement dans la table dimension avec remplacement de valeur (pas d'historisation)
SCD type 2	Ajout d'une nouvelle ligne dans la table de dimension
SCD type 3	Ajout d'une nouvelle colonne dans la table de dimension
Spark SQL	Un projet Apache libre basé sur Apache Spark qui utilise SQL pour interroger des données sur HDFS sans passer par MapReduce
Sqoop	Un projet libre Apache qui permet de transférer des données entre les SGBD relationnels et HDFS
Thrift	Un langage de définition d'interface crée par Facebook, conçu pour la création et la définition de services pour de nombreux langages
Top Down	Conception complète de tout le projet avant sa réalisation avec une approche descendante

Liste des sigles, des symboles et des acronymes

CRM	*Customer relationship management* : gestion de la relation client
DAGs	*Directed Acyclical Graphs* : graphe orienté acyclique
EDW	*Enterprise Data Warehouse* : base de données décisionnelle
ETL	*Extract, Transform, Load* : extraction, transformation, chargement
HDFS	*Hadoop Distributed File System* : système de fichiers distribué de Hadoop
MOLAP	*Multidimensional Online Analytical Processing* : traitement analytique en ligne multidimensionnelle
MPP	*Massively Parallel Processing* : traitement massivement parallèle
OLAP	*Online Analytical Processing* : traitement analytique en ligne
PDBMS	*Parallel Database Management System* : système de gestion de base de données parallèle
RDBMS	*Relational Database Management System* : système de gestion de base de données relationnelle
ROLAP	*Relational Online Analytical Processing* : traitement analytique en ligne relationnelle
SCD	*Slowly Changing Dimensions* : dimension à évolution lente
SPOF	*Single Point Of Failure* : point unique de défaillance
SQL	*Structured Query Language* : langage de requête structurée
YARN	*Yet Another Resource Negotiator* : un autre négociateur de ressources

Introduction

Le but de ce document est de présenter les résultats de l'essai qui s'intitule : « **Avantages du système de fichiers HDFS en termes de performances sur les entrepôts de données de type ROLAP** ».

Un des défis des entrepôts de données à l'heure actuelle, considérant l'importante quantité de données qu'il est possible d'obtenir aujourd'hui d'un système opérationnel ou d'autres sources, est certainement la capacité de ces derniers à répondre aux besoins analytiques qui ne cessent d'augmenter. Ce défi est souvent lié aux problèmes de performances, de coûts, de maintenance, de disponibilité et de capacité d'échelonnage [9].

Lorsqu'on parle de données, et plus exactement de la masse de données qui circule sur Internet, il est question du terme actuellement à la mode « *Big Data* ». Ce terme signifie un volume important de données stockées, une variété de types de données (structurées, semi-structurées, non-structurées) et une vélocité dans la capture et le traitement des données.

Un des problèmes actuels est que les systèmes de gestion de bases de données « *RDBMS* », avec leurs modèles relationnels, souffrent en générale du problème de dégradation de performances dès que la quantité de données commence à devenir importante [42]. Des lenteurs se manifestent quand il faut exécuter des requêtes SQL complexes ou bien effectuer une analyse sur un entrepôt de données de type « *ROLAP* » avec des tables de faits d'une granularité fine [9].

L'amélioration la plus basique utilisée face à ce problème est généralement un échelonnage vertical de la capacité matérielle en mémoire et en processeur ou la maintenance continue des index. Ces solutions temporaires coûtent cher, et sont généralement limitées [9].

La question qu'on se pose et qu'on essaye de répondre à travers cet essai est la suivante : **comment peut-on améliorer les performances dans les entrepôts de données dans un contexte de « *Big Data* » ?**

Une des techniques les plus puissantes pour gérer de grandes masses de données est l'usage du *HDFS*, un système de fichier conçu par la fondation *Apache* pour des applications distribuées. *HDFS* qui se base sur *Hadoop* a été créé en s'inspirant des publications de Google qui utilise les techniques *MapReduce*, *GoogleFS* et *BigTable*. Ces techniques sont puissantes et permettent d'améliorer la performance d'accès aux données dans des circonstances particulières.

MapReduce [37] est un modèle de programmation basé sur *Java*, popularisé par Google pour des calculs parallèles. *GoogleFS* [36], pour sa part, est un système de fichier distribué propriétaire à Google. Enfin, *BigTable* [35] est un système de gestion de base de données compressée et orientée colonnes, qui est propriétaire à Google.

Une architecture *HDFS* de fichiers distribués permet une meilleure disponibilité de données grâce aux nœuds installés autour du « *Cluster* » *HDFS.* Ce dernier est conçu afin de stocker de très gros volumes de données sur des machines à disque dur classique, pour un traitement d'une façon parallèle par la suite [40] [41].

Plusieurs gros joueurs dans le monde ont adopté *Hadoop*. Parmi ces joueurs on trouve Google, Yahoo, Microsoft ainsi que Facebook qui dispose du plus grand « *Cluster* » *Hadoop* au monde.

Un entrepôt de données, qui grandit rapidement en taille, fera sûrement partie de ce monde du « *Big Data* », d'où l'idée de l'essai, pour valider si des techniques de stockage comme *HDFS* peuvent être bénéfiques pour un entrepôt de données.

L'objectif principal de l'essai est d'évaluer la performance d'une solution d'entreposage de données sur un système de fichier *HDFS*. Les informations techniques de cette réalisation ainsi que les étapes d'installation des outils utilisés seront détaillées aux annexes.

Une analyse des avantages et des inconvénients de la solution sera effectuée par la suite, afin de tirer le meilleur de l'expérience d'un entrepôt de données *ROLAP* sur *HDFS*. Parmi ces avantages on peut citer :

- **Avantages quantitatifs** :
    - *Performances en temps d'exécution à des requêtes SQL simples / complexes;*
    - Faibles coûts et licences gratuites.
- **Avantages qualitatifs** :
    - Faible maintenance;
    - Possibilité de traitement parallèle;
    - Haute disponibilité (aucun « *SPOF* »);
    - Capacité facile à monter en charge « *Scalability* ».

Dans cet essai, on s'intéresse à valider les avantages quantitatifs, et plus spécifiquement les performances en temps d'exécution d'une solution basé sur *HDFS*.

Pour être plus précis, l'hypothèse de l'essai est la suivante :
Le stockage d'un entrepôt de données sur un système *HDFS* est avantageux en termes de performances d'interrogation de données en mode *ROLAP*.

Cet essai n'a aucun but promotionnel d'un produit spécifique. L'utilisation de certains outils pour les mesures de performances de *HDFS* servira uniquement à valider la théorie d'entreposage sur *HDFS* comme une solution avantageuse en termes de performance.

Les résultats de cet essai pourront aider les entreprises qui disposent ou disposeront d'une grande quantité de données à faire un choix technique d'entreposage : classique ou sur *HDFS*.

Le premier chapitre porte sur une mise en contexte et présente l'évolution des besoins en termes d'analytiques ainsi que les problèmes qui sont reliés en termes de performances, coûts, maintenance, traitement parallèle et haute disponibilité. Ce chapitre présentera également le problème principal à traiter dans cet essai qui est la performance des entrepôts de données de type *ROLAP*.

Le deuxième chapitre se veut une brève description de l'écosystème *Hadoop* ainsi que la présentation détaillée d'une approche de stockage d'un entrepôt de données sur un système *HDFS*.

Le troisième chapitre expose en détail des résultats comparatifs de performances obtenues à partir de l'exécution de plusieurs types de requêtes SQL en mode *ROLAP* sur plusieurs tailles d'entrepôts de données dans un système *HDFS* en comparaison avec d'autres *SGBD* classiques.

Le quatrième et dernier chapitre se consacre sur l'analyse des résultats obtenus, la validation de la théorie de performance sur *HDFS* ainsi que la proposition de suggestions et de recommandations.

Chapitre 1

Mise en contexte et problématique

Les entrepôts de données « *Data Warehouse* » sont la base des solutions analytiques dans les entreprises. En effet, avec un entrepôt de données alimenté d'une ou de plusieurs sources de données, une multitude de services est offerte aux analystes pour les aider à prendre des décisions. Parmi ces outils, on trouve les rapports, les tableaux de bord, les outils basés sur « *OLAP* » et l'historisation des données.

Utiliser un entrepôt de données, pour des raisons analytiques, vient souvent du problème que les systèmes opérationnels n'offrent pas la flexibilité d'analyser facilement les données à cause de leur structure qui est plus optimisée pour un bon fonctionnement des opérations et non pour faire des analyses.

Depuis plusieurs années, le passage aux entrepôts de données était inévitable [15]. Les outils basés sur des schémas orientés informationnels sont disponibles sur le marché et la création d'un *ET*C (programme qui automatise l'extraction des données d'une ou de plusieurs sources, la transformation et le chargement vers l'entrepôt de données) est devenue un élément populaire dans le domaine d'intelligence d'affaires.

L'analyse des données ainsi que les forages se font sur la base des données obtenues du système opérationnel. Avec l'arrivée des concepts *CRM*, les entreprises commencent à essayer de comprendre les comportements et besoins de leurs clients, car celles-ci doivent bien servir le client et toute information sur ce dernier doit être considérée.

Une nouvelle dimension en termes de données s'est installée, surtout avec l'apparition des réseaux sociaux tels que Facebook, Twitter et Google+. L'enregistrement d'une transaction d'achat ne se limite plus à l'information des produits achetés et du montant payé. D'autres informations sont maintenant récoltées, même si elles ne sont pas utilisées dans l'immédiat, telles que l'adresse *IP*, les informations sur le navigateur et le système de l'utilisateur, le temps passé sur une page web et les intérêts de l'utilisateur.

Des mines de données sont enregistrées chaque jour, et les entrepôts de données deviennent de plus en plus volumineux. Cette situation commence à générer quelques inquiétudes en termes de performance d'exécution des *ETC*, d'accès aux données des entrepôts, et aux coûts générés par les licences des applications utilisées ainsi qu'au matériel en relation. En parallèle, les demandes en termes de données ne cessent d'augmenter et les analystes sont de plus en plus gourmands pour tout ce qui est « *Data* » [9] [16] [44].

1.1 Les entrepôts de données

Un entrepôt de données, ou « *Data Warehouse* » est une base de données qui aide à la décision et qui contient des informations provenant d'une ou de plusieurs sources de données opérationnelles. Des rapports et des statistiques peuvent être générés sur la base d'un entrepôt de données. Un entrepôt de données peut être constitué de plusieurs sous entrepôts qu'on appelle « *Data Mart* », ce dernier est généralement destiné à un département spécifique.

1.1.1 Approches d'entreposage

On distingue deux grandes approches d'entreposage de données : l'approche *Kimball* et celle de *Inmon* qui se réfèrent respectivement à leurs créateurs (Ralph Kimball et Bill Inmon). Les concepts des deux philosophies et leurs utilisations sont différents.

Tableau 1.1 Comparaison des approches Kimball et Inmon

Approche Kimball [15]	**Approche Inmon [13]**
« *Bottom up* »	« *Top Down* »
Data Mart en premier, il contient les données d'un seul département	*EDW* dès le début, il contient toutes les données de tous les départements
Les données dans l'entrepôt peuvent changer. Changement de type « *SCD* » 1, 2, 3	Les données dans l'entrepôt ne changent pas et sont en mode lecture seule
Besoin immédiat et individuel	Besoin non immédiat, mais organisationnel
Outils dimensionnels	Outils relationnels
Access *OLAP* à travers un schéma étoile	Access *OLAP* à travers un *Data Mart*

Source : [13] [15]

1.2 Problèmes des entrepôts de données

1.2.1 Performances ROLAP

Les entrepôts de données de type *ROLAP* sont populaires [43], mais ont un problème de performance quand il s'agit des grandes tables de faits et de dimensions [42]. Le modèle relationnel est excellent pour maintenir l'intégrité des données, mais il n'est pas performant quand il s'agit d'analyser des données d'un entrepôt avec des millions d'enregistrements [42].

L'usage des index ou le partitionnement peut augmenter temporairement la performance, mais pour un entrepôt qui ne cesse de croître, les index en mémoire seront de plus en plus volumineux et leur taille peut parfois dépasser la taille des données de l'entrepôt. Les index sont partagés par les utilisateurs de l'entrepôt, en effet, le besoin d'indexer une table pour un analyste spécifique n'est pas forcément le même que pour les autres analystes [22].
Le nombre de partitions est limité dans les *SGBD* également et l'efficacité du partitionnement est que temporaire lorsqu'il s'agit d'un entrepôt « *Big Data* ».

1.2.2 Coûts

Un entrepôt de données qui croît exponentiellement en termes de volume aura besoin de ressources matérielles plus performantes telle que le processeur et la mémoire, sans oublier les licences annuelles qui sont généralement par nombre de processeurs [9]. L'échelonnage, dans la majorité des cas, est souvent plus vertical que horizontal, c'est-à-dire une augmentation de la capacité de la ressource actuelle. Cette pratique est limitée si on la compare avec l'ajout horizontal des nœuds pour des traitements distribués.

Tableau 1.2 Liste des prix des SGBD ROLAP

SGBD	**Prix d'une licence**
MySQL	• Licence gratuite, sans support • 600 $ à 6 000 $ pour le support • Aucune limite de processeurs
Oracle	• Licence commerciale : 15 000 $ par processeur
SQL Server	• Licence commerciale : entre 600 $ et 10 000 $ • Version express : gratuite, mais limitée à un seul processeur
IBM DB2	• Licence commerciale : pour 6 000 $ par processeur

Source : Oracle, Microsoft, IBM

1.2.3 Maintenances

Il est très fréquent d'ajouter, de mettre à jour, ou de supprimer des index (s'ils sont obsolètes) dans un entrepôt de données afin d'optimiser le temps d'exécution à des requêtes SQL spécifiques. Cette opération sur de grandes tables de mesures ou dimensions peut être gourmande en temps d'exécution et peut affecter la disponibilité de l'entrepôt pour plusieurs heures de maintenance.

La maintenance est une opération complexe lorsqu'il s'agit des grands entrepôts, et toute manipulation sur les index doit être planifiée des jours à l'avance [14]. Le partitionnement est également un élément qui a des coûts dans les maintenances.

1.2.4 Traitement parallèle

Un des problèmes dans les *PDBMS* est la complexité de distribution des données dans le « Cluster ». En effet, les mesures d'un entrepôt de données sont partitionnées à travers tous les nœuds, alors que les dimensions qui sont censées d'être petit sont répliquées dans chaque nœud du cluster (Figure 1.1). Le problème avec cette solution est qu'elle est limitée et peu efficace lorsqu'il s'agit des dimensions avec un grand nombre d'enregistrements, sans oublier les problèmes engendrés par la complexité des jointures en parallèle [9] [38].

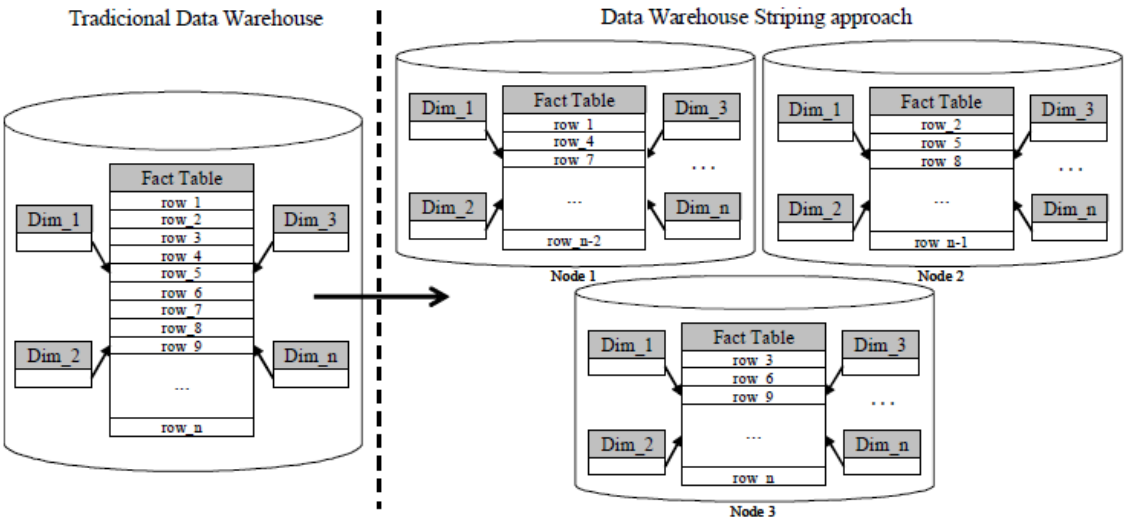


Figure 1.1 Partitionnement de données dans les PDBMS
Source : [9]

Les *PDBMS* performent sur les entrepôts de taille petite et moyenne, car ils chargent et indexent les données avant d'exécuter une requête SQL. Ce processus peut être lent et lourd quand il s'agit d'une grande quantité de données [12] [24] [39].

1.2.5 Haute disponibilité

La haute disponibilité d'un entrepôt de données peut être assurée par l'élimination du problème du *SPOF*. Pour cela, il est nécessaire d'avoir une réplication avec plus d'un serveur *SGBD* afin de détenir au moins deux copies des données. Cette tâche implique une maintenance pour la réplication et nécessite généralement une surveillance de la duplication des données.

La haute disponibilité nécessite généralement des efforts importants [45], et en cas de perte totale du serveur *SGBD* hébergeant l'entrepôt de données, on sera dans l'obligation de recharger les données dans un nouveau serveur en utilisant l'*ETC* initial (en espérant que les données sont toujours disponibles dans les sources, ce qui n'est pas toujours le cas). Cette technique ne garantit pas un retour à la normale et ne prend pas en considération les utilisateurs de l'entrepôt qui restent en attente de la donnée lors de la mise en service du serveur de relève.

1.3 Problématique de l'essai

Le « *Big Data* » et les demandes gourmandes des analystes pour tout ce qui est données, non seulement en termes de quantité, mais de diversités également, ont rendu de plus en plus volumineux les entrepôts de données *ROLAP* qui sont stockées sur des *SGBD* relationnels classiques [17] [18]. Cette situation a généré un problème au niveau des performances d'extraction des données depuis ces entrepôts de données [9] [22].

Les extractions de données se font à travers des requêtes SQL, ou l'utilisateur interroge l'entrepôt de données basé sur un schéma étoile avec des tables relationnelles de faits et de dimensions.

Malgré l'utilisation des index, ces extractions de données sont de plus en plus lentes lorsqu'il s'agit de grandes tables de faits et de dimensions avec plusieurs jointures qui compliquent la requête SQL au niveau du *SGBD* relationnel [9] [14].

D’où la problématique de l’essai :
Les entrepôts de données *ROLAP* stockés sur un *SGBD* relationnel ont un problème de performances d’exécution des requêtes SQL quand la quantité de données est importante dans les tables faits et dimensions.

1.4 Conclusion

L’intelligence d’affaires ou le « *Business Intelligence* », est un domaine qui existe depuis plusieurs années et qui est en train de vivre des défis plus intéressants par rapport au début de son apparition dans les années 1990. Un de ces défis est la performance des entrepôts de données dans un contexte de mégadonnées.

La quantité de données qu’on traite aujourd’hui dans les modèles *ROLAP* est beaucoup plus importante que celle de la précédente décennie [17]. Les problèmes liés à la performance, aux coûts, à la lenteur de traitement, à la complexité d’échelonnage, et à la haute disponibilité ont poussé sans arrêt à chercher des solutions qui répondent aux besoins analytiques de nos jours avec des coûts acceptables.

L’essai, et dans le même contexte du problème des performances des entrepôts de données *ROLAP* avec ce monde du « *Big Data* », essayera d’apporter des suggestions en tenant compte des outils utilisés aujourd’hui pour le traitement de mégadonnées, tels que *Hadoop* et son écosystème qui seront abordés dans le prochain chapitre.

Chapitre 2

L'écosystème Hadoop et l'approche proposée

Une recherche bibliographique a été effectuée sur les éléments nécessaires qui seront utilisés dans l'essai. Une brève comparaison des deux architectures les plus utilisées dans les entrepôts de données (*ROLAP*, *MOLAP*) sera suivie d'une présentation du système de fichiers *Hadoop* ainsi que de son écosystème. À la fin du chapitre, on abordera la description de l'approche proposée pour répondre au but initial de l'essai.

2.1 ROLAP versus MOLAP

Tableau 2.1 Comparaison entre ROLAP et MOLAP

ROLAP « *Relational OLAP* »	**MOLAP « *Multi-dimensional OLAP* »**
Les données stockées en mode relationnel	Les données stockées en mode propriétaire dans des cubes multidimensionnels
Un usage du langage SQL	Un usage du langage MDX
Les données sont extraites à la demande	Pré-calcule du cube à l'avance
Les performances peuvent être limitées	Les performances sont meilleures
Les données sont dans une seule source relationnelle	Les données sont transférées d'une source relationnelle vers des cubes (redondances)
La quantité de données à gérer est illimitée	La quantité de données à gérer est limitée

Source : Olap.com, [21]

2.2 L'écosystème Hadoop

Le tableau suivant représente les composantes qui entourent le système de gestion de fichiers *HDFS*. Ce dernier constitue la base de tout l'écosystème de *Hadoop*.

Tableau 2.2 L'écosystème de Hadoop

<table>
<tr><td colspan="3">L'ÉCOSYSTÈME HADOOP</td><td colspan="4">Extraction, transformation, chargement / Outils « BI »</td></tr>
<tr><td rowspan="7">Ambari (Gestion « cluster », surveillance)
Hue (Accès applications par interface web)</td><td rowspan="7">Zookeeper
(Coordination)</td><td rowspan="7">Oozie
(Planification)</td><td>Impala</td><td>Spark SQL</td><td>Presto</td><td>Temps réel</td></tr>
<tr><td>Pig
(Script)</td><td>Hive
(SQL)</td><td>Sqoop
(Transfert de données)</td><td>Accès aux données</td></tr>
<tr><td colspan="3">HCatalog
(Meta Data)</td><td>Métadonnées</td></tr>
<tr><td colspan="3">HBase
(Stockage de type Colonne)</td><td>Stockage table</td></tr>
<tr><td colspan="3">MapReduce
(Programmation distribuée)</td><td>Distribution</td></tr>
<tr><td colspan="3">Yarn
(Gestion de ressources)</td><td>Ressources</td></tr>
<tr><td colspan="3">(Hadoop Distributed File System)</td><td>Stockage objet</td></tr>
</table>

Source: Apache

2.2.1 Structure de données sur HDFS

Les données sur *HDFS* (structurées, semi-structurées ou non structurées) sont enregistrées avec le concept : écrire une fois, lire plusieurs fois. Cela veut dire que les mises à jour et les suppressions de données ne font pas partie des bonnes pratiques du *HDFS* [34].

C’est similaire au principe de l’entreposage de données décrit précédemment avec la technique d’*Inmon* à l’inverse de celle de *Kimball* où les *SCD* de type 1, 2, 3 sont permises.

Les données dans un *HDFS* sont enregistrées en blocs de grande taille de 64 MB (par défaut) ou plus pour des raisons de performance (réduire le temps d'accès).

Dans un « *Cluster* », on dispose d’un nœud maître « *NameNode* » qui gère les nœuds de données « *DataNodes* ». La haute disponibilité est assurée par la couche de données, car les blocs sont répliqués (par défaut) en 3 copies ou plus.

La haute disponibilité du « *NameNode* » pourra être assurée par l’ajout d’un « *Secondary NameNode* » qui se met en synchronisation avec le nœud maitre [34].

Une table sur un système *HDFS* n’est que l’ensemble de plusieurs sous fichiers qui contiennent des données et qui sont reliés entre eux. Contrairement aux *SGBD*, le *HDFS* ne gère pas les index, ce qui fait de lui un système sans schéma avec un gain d’espace et de mémoire et avec une faible maintenance [34].

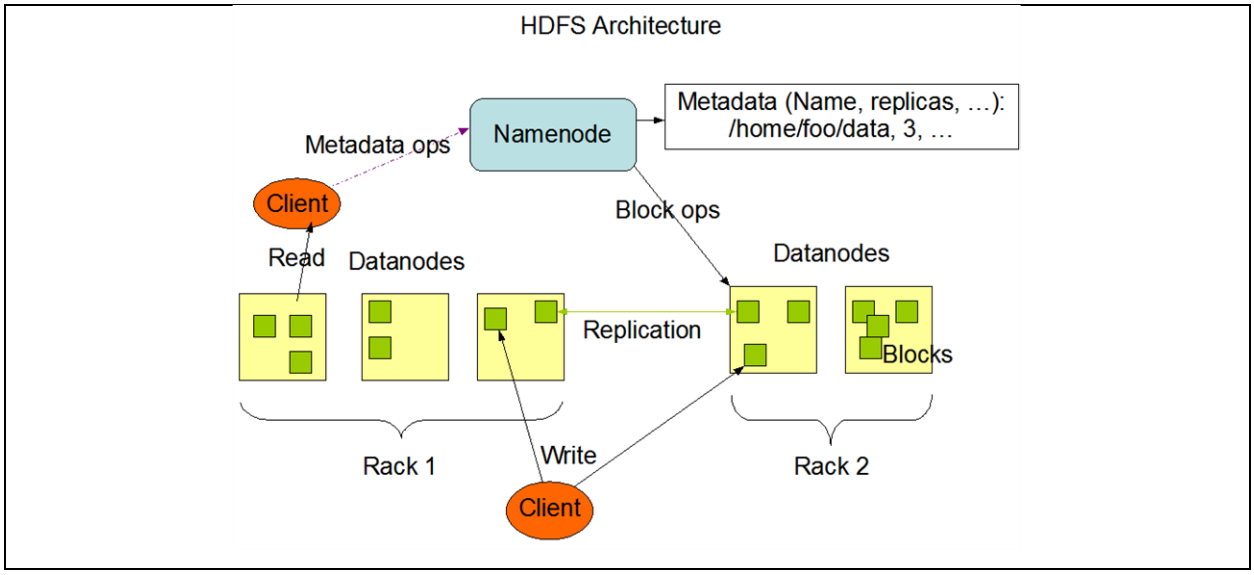


Figure 2.1 Architecture HDFS

Source : Apache

2.2.2 MapReduce

MapReduce est un modèle de programmation adapté au traitement massif et parallèle de grandes quantités de données, et qui est largement utilisé pour extraire des données du système de fichiers *HDFS* [12]. L'architecture MapReduce est composée d'un service « *JobTracker* » qui peut fonctionner sur le serveur « *NameNode* » et qui reçoit des travaux « *Jobs* » à partir des applications clientes, et envoie par la suite des tâches aux nœuds de données « *TaskTracker* » disponibles.

L'avantage de cette technique est qu'elle est simple, et composée deux fonctions qui doivent être écrite par le programmeur : la fonction « *Map* » et la fonction « *Reduce* », les deux se basant sur le concept clé/valeur. À l'inverse des *PDBMS* où on doit programmer en SQL, le *MapReduce* offre la possibilité d'utiliser un langage de programmation procédural ou orienté objet tel que C++ ou Java [12].

L'avantage du *MapReduce* est qu'il assure le traitement parallèle en utilisant la distribution des données, le balancement de charges ainsi que la tolérance aux pannes.

MapReduce est conçu pour fonctionner sur des machines à faible coût avec des disques durs simples, car il se base sur des opérations d'E/S sans que la mémoire soit une priorité.

Les inconvénients du *MapReduce* sont plus liés à la complexité de la programmation, même dans le cas d'une simple requête. Ainsi, en termes de performance, un *PDBMS* tel que « *Vertica* » par exemple, est plus rapide que *MapReduce* par un facteur de 3.2 à 7.4 pour un cluster de 100 nœuds [6] [12].

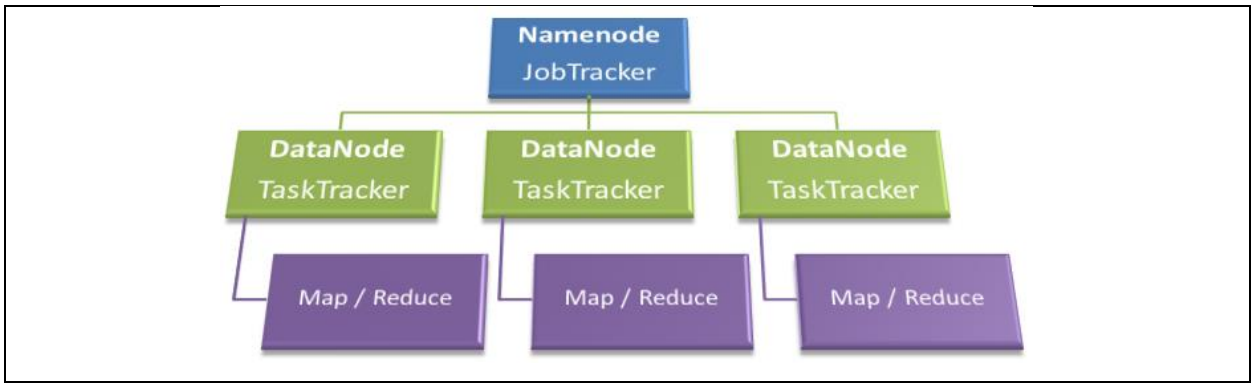


Figure 2.2 Architecture MapReduce

Source: Apache

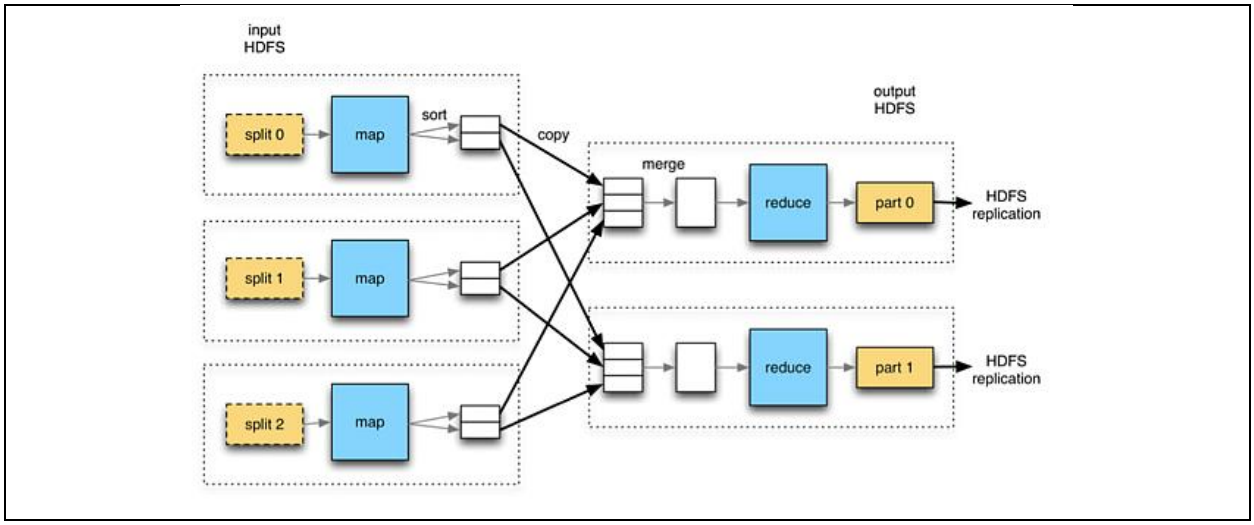


Figure 2.3 Algorithme MapReduce

Source: Hadoop – The Definitive Guide, by Tom White

2.2.3 Yarn

Yarn « *Yet Another Resource Negotiator* » est une technologie qui gère l'utilisation des ressources dans un « Cluster ». Depuis la version 2.0 de *Hadoop*, *Yarn* est plus générale que *MapReduce* et propose une nouvelle architecture de la fonction « *JobTracker* » qui consiste à séparer ses deux tâches de gestion de ressources et celle de planification et surveillance des

travaux. Cela permet d'exécuter des tâches qui ne se basent pas sur *MapReduce* comme *Spark* ainsi que des tâches *MapReduce* sur le même « *Cluster* ».

Le nouveau « *RessourceManager* » qui se compose de « *Scheduler* » et de « *ApplicationsManager* » gère l'assignement des ressources de calcul en communiquant avec chaque nœud « *NodeManager* » alors que « *ApplicationMaster* » assure la coordination et la planification des applications qui peuvent être une ou plusieurs « *Jobs* ». Ce dernier négocie avec le « *RessourceManager* » les besoins en ressources (processeur, mémoire, disque, réseau).

MapReduce, avec sa version MRv2, peut fonctionner à travers *Yarn* sans aucun changement du code *Map* et du code *Reduce* et en bénéficiant de la flexibilité et la meilleure gestion de *Yarn* en termes d'allocation de ressources [3].

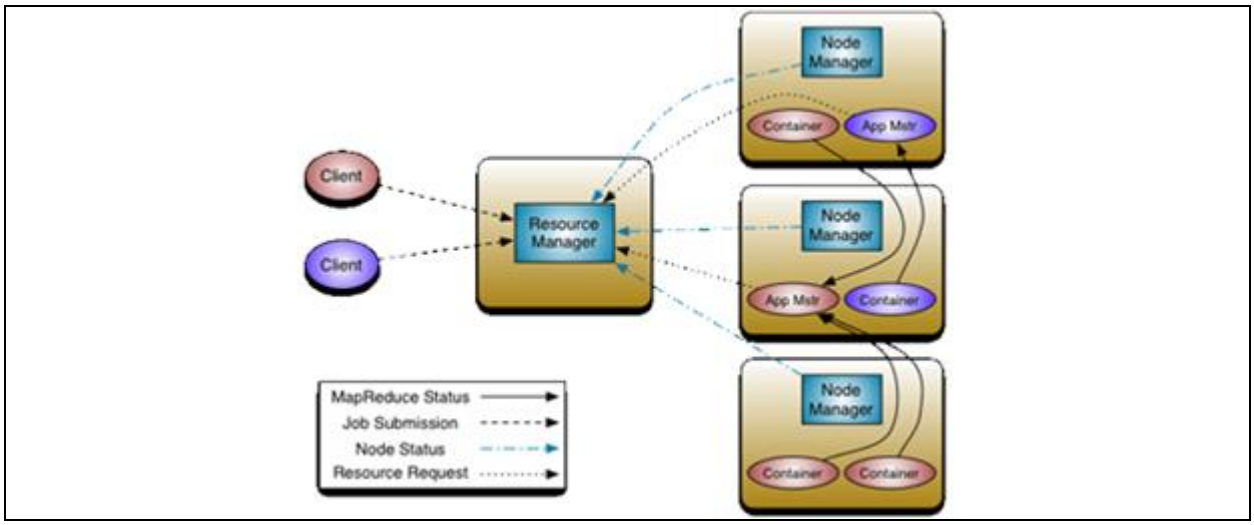


Figure 2.4 Yarn
Source : Apache

2.2.4 Sqoop

Apache Sqoop est un outil conçu pour transférer efficacement et rapidement des données en mode chargement rapide « *Bulk* » depuis des bases de données relationnelles vers *HDFS, Hive* ou *HBase*. Inversement, *Sqoop* peut aussi être utilisé pour extraire des données du *HDFS* et de les exporter vers des bases de données relationnelles ou des entrepôts de données.

Sqoop fournit un mécanisme de connecteur pour la connexion optimale aux systèmes externes. L'API *Sqoop* fournit un *Framework* pratique pour la construction de nouveaux connecteurs qui peuvent être déposés dans le répertoire d'installation de *Sqoop* pour fournir une connectivité à différents systèmes. *Sqoop* est livré avec les connecteurs les plus populaires comme *MySQL, Oracle, PostgreSQL, SQL Server* [30].

Exemples d'utilisation de *Sqoop* :

- Importation d'une table depuis *Microsoft SQL Server* (avec 6 processus *Map*)

```
Sqoop import --driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--table MYTABLE --connect
'jdbc:sqlserver://MSSQLSeverHost;database=DBNAME;username=USER;
password=PASSWORD' –m 6
```

- Importation d'une table depuis *DB2* (avec 1 seul processus *Map*)

```
sqoop import --driver com.ibm.db2.jcc.DB2Driver --connect
jdbc:db2://db2host:50000/DBNAME --username USER
--password PASSWORD --table MYTABLE -m 1
```

- Exportation d'une table de *Hive* vers *Teradata*

```
sqoop export --connect jdbc:teradata://terahost/DATABASE=DBNAME
--username USER --password PASSWORD --input-fields-terminated-by ','
--export-dir /user/hive/warehouse/mydb.db/data_stg --table MYTABLE
```

2.2.5 Zookeeper

Apache Zookeeper est un service centralisé libre qui a pour rôle de conserver les informations de configuration et de nommage, assurer la synchronisation distribuée des données. *ZooKeeper* est un projet important de l'écosystème de *Hadoop* avec une architecture qui supporte la haute disponibilité. Un client peut communiquer avec un autre « *Zookeeper master* » si le premier échoue.

Zookeeper garantie qu'une écriture de données d'un même client est procédée dans l'ordre envoyé par ce client, et permet aux processus distribués de se coordonner autour d'un espace de nom sous forme d'arborescence partagée ou les données sont stockées dans les nœuds de l'arbre. On appelle ces noeuds des « *Znodes* ». Chaque *Znode* est associé à des données spécifiques et est identifié par un chemin et un parent à l'exception de la racine '/' qui n'a aucun parent. La mise à jour des nœuds se fait d'une manière atomique et les clients peuvent être notifiés de cette mise à jour. *Zookeeper* peut fonctionner sur les mêmes machines qui hébergent d'autres services *Hadoop* [29].

2.2.6 Oozie

Apache Oozie est une application écrite en Java qui fait partir de l'écosystème de *Hadoop* et qui est utilisée pour planifier les « *Jobs* » de *Hadoop*. *Oozie* combine plusieurs travaux dans une seule unité logique de travail, il peut être utilisé avec *MapReduce* ou *Yarn* et supporte les outils *Pig, Hive et Sqoop*. Il peut également être utilisé pour planifier des travaux d'un système basé sur des programmes écrit en Java.

Il existe deux types de travaux *Oozie* :

- *Oozie Workflow* : les travaux représentent des séquences d'actions à exécuter qui sont regroupées dans ce qu'on appelle « *DAGs* »;
- *Oozie Coordinator* : les travaux sont des « *Oozie Workflow* » récurrents qui sont déclenchés par le temps et par la disponibilité des données.

Pour un travail de *MapReduce* contrôlé par *Oozie*, ce dernier planifie et déclenche les actions, mais c'est le *MapReduce* qui les exécute. Cela permet à *Oozie* de balancer la charge et de gérer les erreurs [27].

2.2.7 Hive / Pig / HCatalog

Les données sur un système *HDFS*, peuvent être interrogées de plusieurs façons. Programmer directement en *MapReduce* pour extraire des données de *HDFS* est une tâche compliquée qui nécessite beaucoup d'efforts. Dans ce contexte, des applications libres avec un niveau d'abstraction plus élevé que la programmation *MapReduce* sont disponibles pour répondre à ce besoin, surtout quand il s'agit des tâches avec des jointures.

- **Pig** : développé par Yahoo, utilise un langage de scripts appelé *Pig Latin* et transforme le programme en tâches *MapReduce* [28] [34].
- **Hive** : similaire à *Pig*, mais utilise un langage *HiveQL* semblable à SQL. *Hive* était créé principalement pour les analystes de Facebook pour leur permettre d'analyser des données dans *HDFS* sans le passage à la programmation *MapReduce* [26] [31].
- **HCatalog** : devenu une partie du projet *Hive* depuis mars 2013 [26], a pour rôle principal de centraliser les méta-données qui définissent l'endroit où les données sont stockées. *HCatalog* expose son service à d'autres applications de l'écosystème de *Hadoop* pour partager ces méta-données et leur permettre un accès rapide aux données correspondantes sans aucun besoin par l'application de connaitre l'endroit physique ou les données sont stockées sur *HDFS*.

  Par exemple, la création d'une table « clients » avec *Hive* à travers *HCatalog* où les données résident à l'emplacement *HDFS* « */user/hive/warehouse/clients* », rendra cette table accessible par *Pig* et *MapReduce* juste en utilisant l'alias « clients ».

  La définition des tables est maintenue dans le « *metastore* » de *Hive* et disponible pour *Hive, Pig, MapReduce* et les clients qui utilise une API *REST*.

Exemple (hortonworks.com) :

Une table qui représente la liste des clients est stockée sur *HDFS* en format délimité et créé avec *Apache HCatalog*, peut être accessible avec *Hive* et *Pig*. Pour extraire de cette table les informations du client « *IBM* » en utilisant *Hive* et *Pig* on procède comme suit :

Hive	SELECT * FROM clients WHERE name = 'IBM';

Pig	a = LOAD 'default.clients' USING
	org.apache.hcatalog.pig.HCatLoader();
	b = filter a by name == 'IBM';
	c = group b all;
	dump c;

2.2.8 Temps réel avec Hadoop

Des améliorations récentes de performances par rapport à Hive en termes d'extraction des données du *HDFS* avec une syntaxe SQL ont permis l'apparition de certains produits qui utilisent des techniques de « *MPP* » et contournent le *MapReduce*. Cette approche semble être un grand pas vers ce qu'on appelle « *real time on Hadoop* », ou parfois « *near real time on Hadoop* » si on la compare avec *Hadoop* qui était conçu pour des traitements distribués qui ne nécessitent pas une réponse rapide ou instantanée.

Ce progrès laisse penser que l'entreposage sur un *HDFS* peut être avantageux quand on voit ce qu'il peut apporter par rapport aux problèmes présenté précédemment. Parmi ces produits, on trouve « *Impala* », un projet libre crée par « *Cloudera* » un des plus grands distributeurs de solutions *Hadoop*. *Impala* utilise les métas-donnés de *Hive* pour extraire les données plus rapidement de *HDFS* avec des techniques de traitement parallèle massives [23]. On peut trouver également *Spark SQL*, un projet libre également et qui n'est pas directement lié au *MapReduce*. Ces outils « *real time / near real time* » qui utilisent un langage très proche du SQL sur *HDFS*, ont pour objectif d'améliorer les performances en termes de temps d'exécution par rapport au *MapReduce*.

2.2.8.1 Impala

Cloudera Impala est un moteur SQL basé sur des traitements parallèles massifs *MPP* et qui s'exécute nativement sur *Apache Hadoop*. *Impala* est un projet libre avec licence *Apache* qui combine la puissance de *Hadoop* avec les technologies modernes de bases de données échelonnées et qui permet à l'utilisateur d'interroger directement les données stockées sur *HDFS* ou *Apache HBase* sans passer par la complexité de *MapReduce* en utilisant les métadonnées de *Hive*. [7].

Les caractéristiques d'*Impala* sont :

- Licence *Apache* et 100 % libre;
- Architecture *MPP* de traitement parallèle massive pour la performance 10-100x plus rapide que *Hive;*

- Analyse interactive sur les données stockées dans *HDFS* et *HBase;*
- Construit avec la sécurité de *Hadoop;*
- Support de *SQL ANSI -92* avec des fonctions définies par l'utilisateur (*UDF*);
- Supporte les formats de *Hadoop* commun de fichiers : *texte, SequenceFiles, Avro, RCFile, LZO et Parquet* (stockage type colonne avec compression);
- Gains en coûts avec la réduction du mouvement de données (données déjà sur *HDFS*) et modèle de données avec un schéma flexible [7].

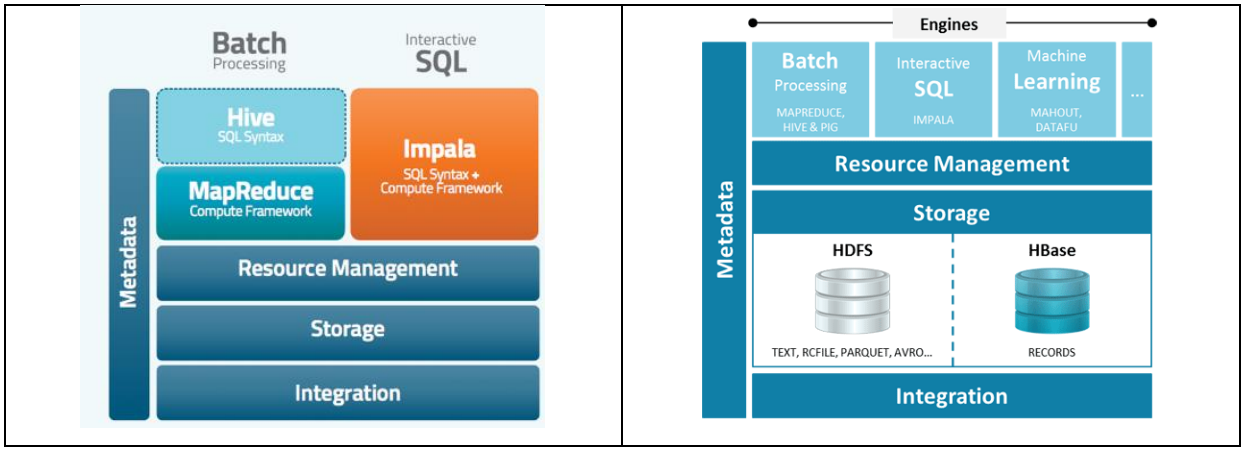


Figure 2.5 Moteur SQL Impala

Source : Cloudera

2.2.8.2 Spark SQL

Spark SQL (anciennement *Shark*) est un projet *Apache* basé sur *Apache Spark* qui est un engin qui exécute à travers *Yarn* des programmes en parallèle 100x plus rapide que *Hive* avec l'utilisation maximale de mémoire contrairement à *Hive* qui utilise un maximum d'E/S sur les disques [4].

Les caractéristiques de *Spark SQL* sont :

- Licence *Apache* et 100 % libre;
- Codage facile d'application en utilisant l'API *Spark* avec des langages haut niveau comme Java, Scala, Python et R;
- Chargement et interrogation des données depuis plusieurs sources : *HDFS, Cassandra, Hive, HBase, Amazon S3;*

- Connexion *ODBC, JDBC* [4].

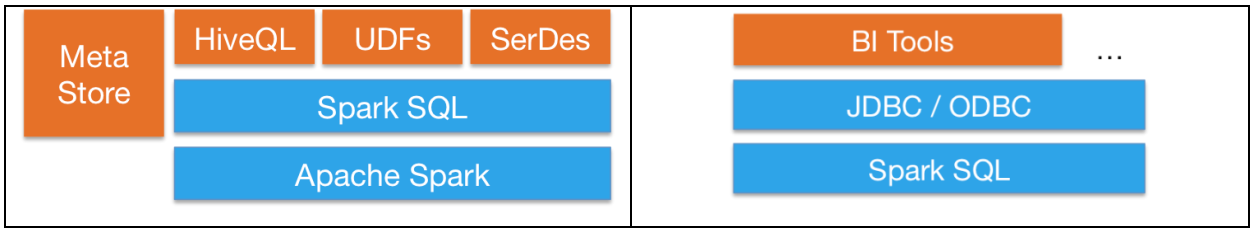


Figure 2.6 Moteur Spark SQL

Source : Cloudera

2.2.8.3 Presto

Presto est un outil conçu pour interroger efficacement des quantités énormes de données (des téraoctets ou pétaoctets) à travers des requêtes distribuées. *Presto* a été inventé par Facebook qui s'intéressait au temps réel avec *Hadoop* [5] comme une alternative aux outils qui interrogent *HDFS* en utilisant les pipelines de *MapReduce* tels que *Hive* ou *Pig*.

Presto a accès aux *HDFS* et peut être étendu pour fonctionner sur différents types de sources de données, y compris les bases de données relationnelles classiques et d'autres sources de données telles que *Cassandra*. On peut par exemple faire une jointure entre une table sur *HDFS* avec une autre sur *Cassandra* ou *MySQL*.

Presto a été conçu pour gérer l'entreposage, l'analyse des données, l'agrégation de grandes quantités de données et la production de rapports. Ces charges de travail sont souvent classées comme traitement analytique en ligne (*OLAP*). [11] [32]

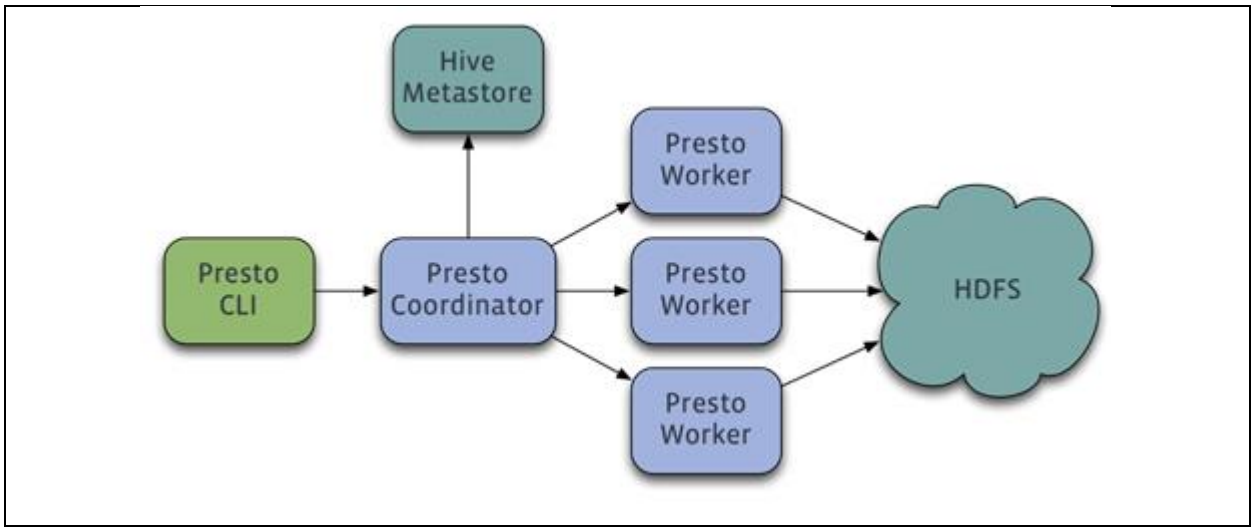


Figure 2.7 Facebook Presto

Source: presto.io

2.2.8.4 HBase

HBase est un projet *Apache* libre conçu pour gérer en temps réel des accès lecture/écriture aléatoires basé sur *HDFS*. Le but principal de ce projet est de pouvoir stocker de grandes tables à l'échelle de millions de colonnes groupées dans ce qu'on appelle « *Familiy group* » et billions de lignes de données. *HBase* est basé sur le concept clé-valeur.

```
put 'hbase_table_name', 'unique_row_id', 'family_group :field1', 'value1'
put 'hbase_table_name', 'unique_row_id', 'family_group :field2', 'value2'
get 'hbase_table_name', 'unique_row_id'
```

Il est aussi possible de l'interroger avec du SQL à travers *Hive* ou *Impala* avec la création d'une table externe stockée sur *HBase* :

```
create EXTERNAL table hbase_table_name(unique_row_id INT,field1
String, field2 INT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES('hbase.columns.mapping' =
```

```
:key,family_group:field1, family_group:field2')
TBLPROPERTIES('hbase.table.name' = 'hbase_table_name');
```

HBase a le même concept que *HDFS*, mais utilise *HRegion* au lieu des *DataNode* et *HRegionServer* au lieu du *NameNode*. Il est tout à fait possible d'utiliser les mêmes serveurs *HDFS* pour ajouter la couche *HBase*.

Les caractéristiques de *HBase* sont :

- Licence *Apache* et 100 % libre;
- Linéaire avec une évolutivité modulaire;
- Strictement conforme des lectures et écritures;
- Accès facile des clients à travers l'API Java;
- Passerelle *Thrift* et un service Web REST-full qui prend en charge *XML* et options binaires de codage de données;
- La possibilité de faire des « *INSERT / UPDATE / DELETE* » sur *HDFS* [2].

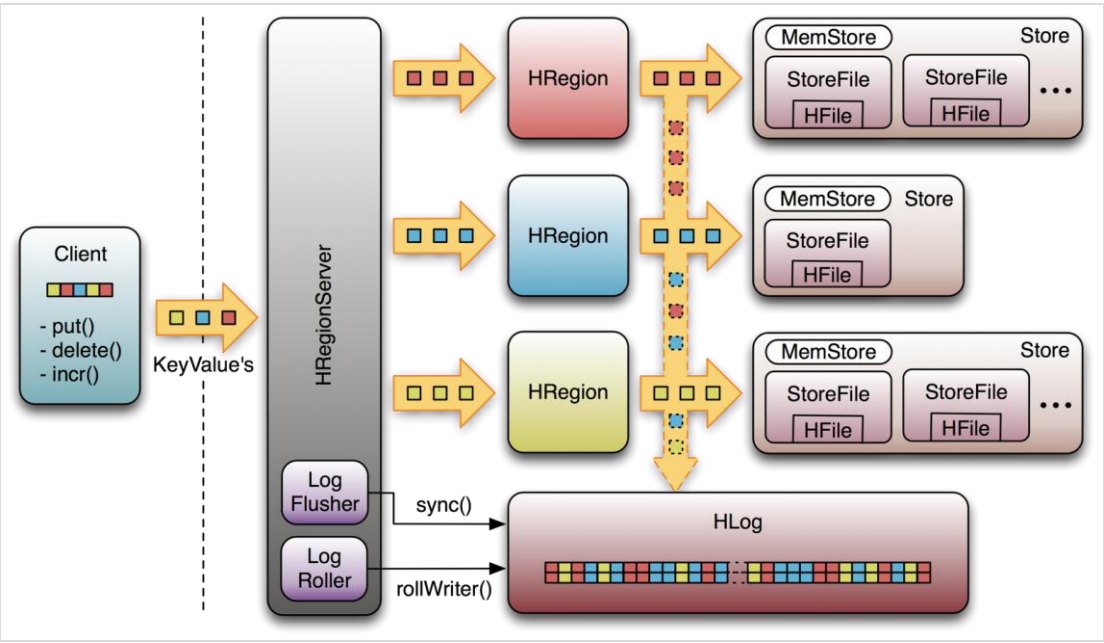


Figure 2.8 Apache HBase

Source : Datascience-Labs

2.2.9 Ambari

Apache Ambari est un projet libre destiné aux administrateurs systèmes qui est basé sur une *API RESTful* et qui a comme but de surveiller et de gérer un « *Cluster* » *Hadoop* avec une simple interface web.

Avec *Ambari* on peut facilement :

- installer et configurer facilement des services *Hadoop* dans plusieurs serveurs;
- gérer d'une façon centralisée des services *Hadoop* : arrêt, démarrage;
- surveiller l'état du « *Cluster* » *Hadoop* à travers un tableau de bord;
- recevoir des alertes en cas de problèmes sur un des serveurs (serveur inaccessible, espace disque insuffisant) [25].

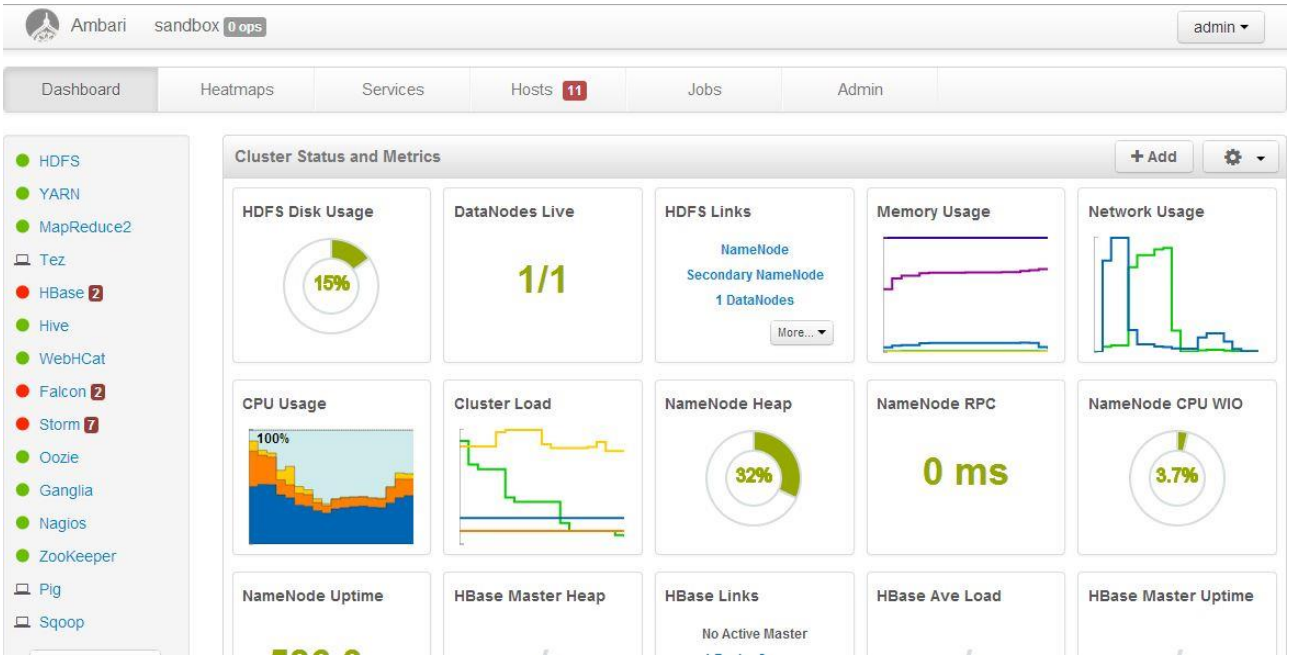


Figure 2.9 Ambari

Source: hortonworks.com

2.2.10 Hue

Hue est une interface web libre sous la licence d'*apache* V2.0 qui permet à un utilisateur de manipuler facilement des données sur *HDFS* et diminuer la complexité des lignes de

commandes.

Avec *Hue* on peut facilement :

- parcourir et manipuler des fichiers et répertoires sur *HDFS* (créer, supprimer);
- extraire des données avec des requêtes SQL en utilisant le client *Impala* ou *Hive;*
- écrire des scripts *Pig*, les exécuter et voir l'état de l'exécution;
- gérer les métadonnées *Hive* qui sont partagées avec *Impala;*
- créer et supprimer des tables et des bases de données *Hive* et *Impala*;
- utiliser plus facilement *Sqoop* pour transférer des données entre *HDFS* et les bases de données relationnelles avec une simple interface;
- créer des « *Job* », les exécuter et les surveiller;
- utiliser facilement *Oozie* et *HBase* [8].

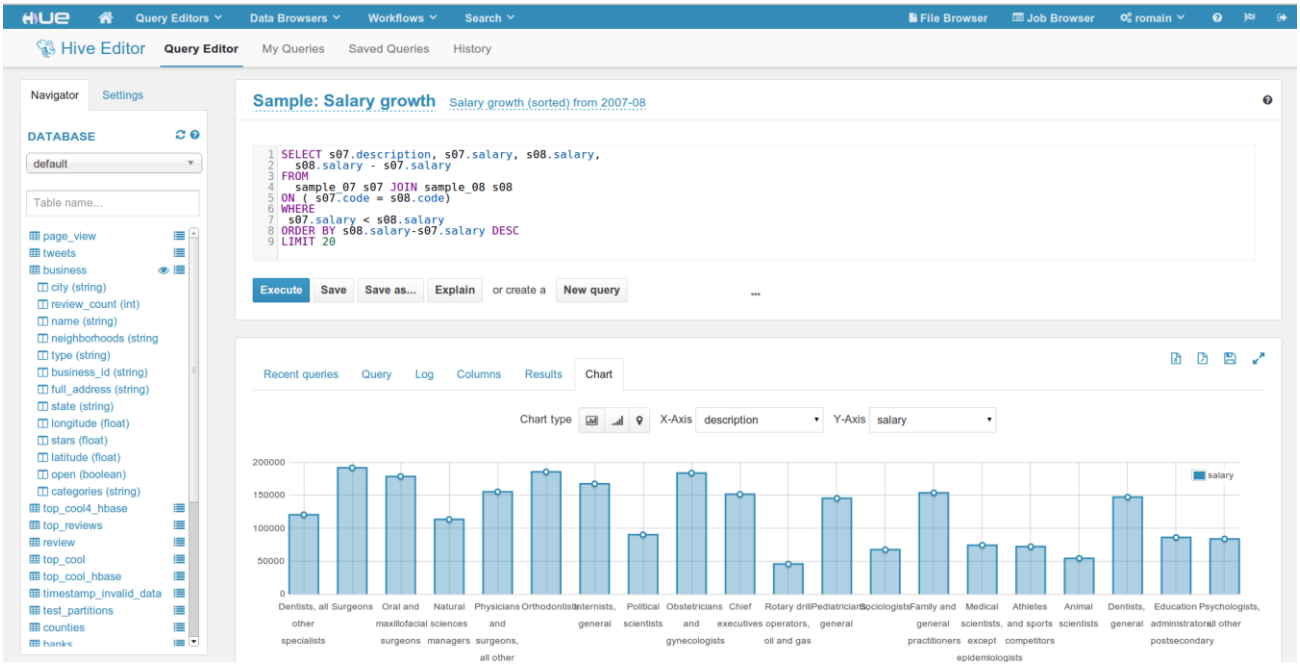


Figure 2.10 Hue

Source: gethue.com

2.3 Distributions Hadoop

Quelques distributions gratuites de *Hadoop* ont vu le jour avec l'évolution de l'écosystème de ce dernier. Ces distributions regroupent la majorité des services de l'écosystème de *Hadoop* et facilitent la gestion de tous les nœuds. En effet, il est devenu complexe de maintenir plusieurs nœuds dans un cluster et un petit changement de configuration dans un service doit être déployé et appliqué dans chaque nœud [7].

Le principe de cette gestion de nœuds consiste à l'installation de deux petites applications (agents) qui communiquent entre elles en mode client-serveur :

1. Agent serveur : on l'installe généralement sur le serveur « *NameNode* ».
2. Agent client : on l'installe dans chaque nœud pour communiquer avec l'agent serveur.

Les avantages d'une distribution *Hadoop* sont :

- Gestion centralisée et visuel de tout le « Cluster »;
- Ajout/suppression facile de nœuds dans le « Cluster »;
- Ajout/suppression/Configuration de services à travers tout le « Cluster »;
- Surveillance des nœuds, alarme en cas de problèmes [7].

Tableau 2.3 Services des distributions Cloudera et Hortonworks

Service	Distribution Cloudera	Distribution Hortonworks
HDFS / MapReduce / Yarn	✅	✅
Zookeeper / Oozie	✅	✅
Sqoop / Flume / Kafka	✅	✅
Pig / Hive / Solr / Storm	✅	✅
HBase / Spark	✅	✅
Impala	✅	
Tez		✅
Hue / Cloudera manager	✅	
Ambari		✅

Source : Cloudera, Hortonwors

2.4 Description de l'approche proposée

Le but principal de l'expérimentation est de mesurer les performances de l'interrogation des données avec des requêtes SQL en mode *ROLAP* sur des entrepôts de données de différentes tailles stockées sur *HDFS* et de comparer le temps d'exécution de chaque requête SQL avec d'autres *SGBD*.

Pour obtenir des résultats pertinents, une comparaison de performances entre une solution basée sur *HDFS* et d'autres solutions traditionnelles sera effectuée.

Les tests sont réalisés sur deux *SGBD* relationnels classiques (*MySQL* et *Microsoft SQL Server),* ainsi que sur des projets libres basés sur *HDFS* et qui utilisent SQL comme langage d'interrogation de données (*Hive, Impala, Spark SQL et Presto*).

Il s'agit d'interroger chaque système avec un nombre de requêtes SQL, puis de mesurer le temps d'exécution de chacune de ces requêtes.

- Les requêtes SQL sont exécutées dans des quantités de données différentes : petite, moyenne et grande;
- Les requêtes SQL sont exécutées à trois reprises et la moyenne est prise en compte;
- Les index nécessaires sont créés, les statistiques sont mises à jour et le cache du SGBD est vidé durant les tests;
- L'environnement matériel (mémoire, processeur, disque dur) utilisé est le même pour tous les tests;
- Le matériel informatique utilisé durant les tests de performance est local, en mode déconnecté d'Internet pour éviter toute sorte de mises à jour applicatives qui peut impacter les performances du serveur;
- Aucune utilisation de l'infonuagique n'est envisagée également pour éliminer toute falsification des résultats à cause des perturbations réseau non contrôlées.

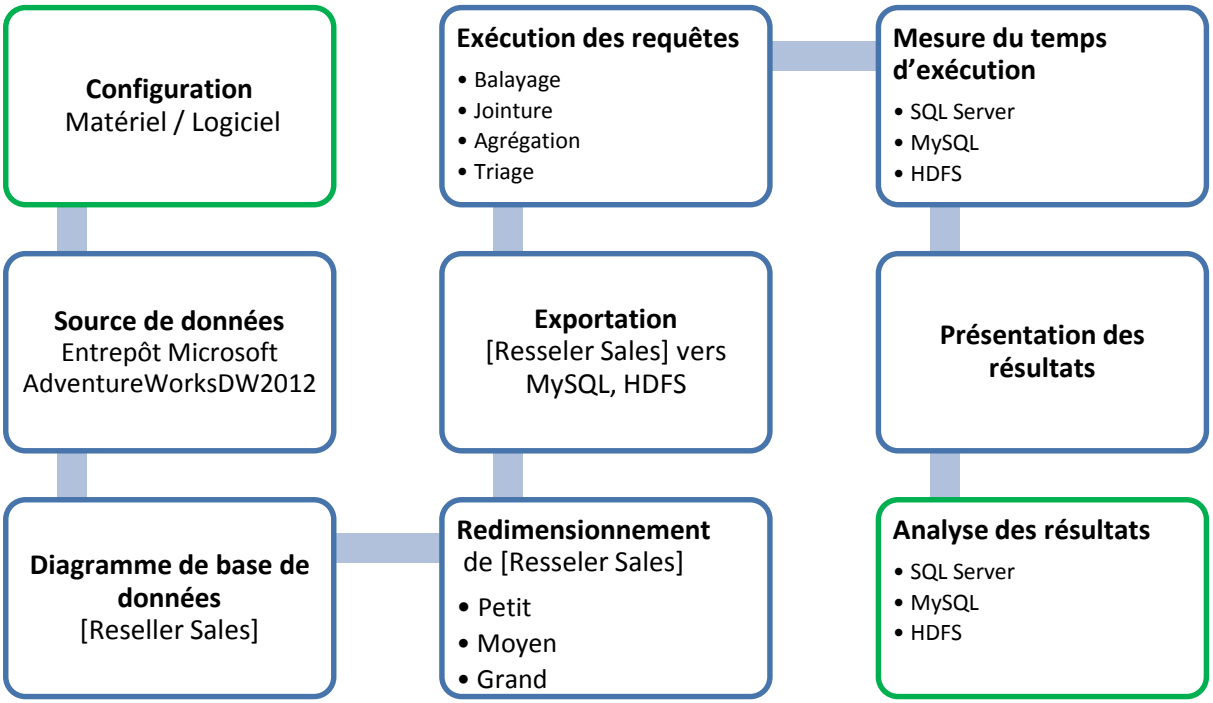


Figure 2.11 Processus de l'approche proposée

2.4.1 Configuration matérielle et logiciel

Les tests de performance sont effectués sur un serveur Workstation DELL PRECISION 490 (Annexe VI). Le Tableau 2.4 représente les moteurs SQL utilisés dans l'expérimentation.

Tableau 2.4 Moteurs SQL de l'expérimentation

SGBD relationnel	**HDFS**
1. MS SQL Server : 2012 2. MySQL : 5.6.24 avec le moteur MyISAM	1. Hive : 1.1.0-cdh5.4.0 (*Parquet*) 2. Impala : 2.2.0-cdh5 avec stockage colonne (*Parquet*) 3. Spark SQL : 1.3.0 (*Parquet*) 4. Presto 0.105 (*Parquet*)

Le choix de *MySQL* et de *MS SQL* Server comme *SBGD* traditionnelles est justifié par leur popularité ainsi que leur utilisation dans les entrepôts de données. Avant l'utilisation de chaque moteur, un arrêt des services de tous les autres moteurs est effectué pour permettre à un seul engin SQL d'être actif à la fois. *Hive, Impala, Spark SQL* et *Presto* sont configurés sur le même *HDFS* et interrogent les mêmes fichiers qui représentent les données de l'entrepôt.

2.4.2 Source de données

Amazon AWS héberge de nombreux ensembles de données publics auxquels tout le monde peut accéder gratuitement. Les sources les plus consultées de données sont [1] :

- NASA NEX : une collection d'ensembles de données portant sur les sciences de la terre, entretenue par la NASA.
- Common Crawl Corpus : un corpus de données d'indexation Web composé de plus de 5 milliards de pages Web.
- Projet 1 000 génomes : une cartographie détaillée des variations génétiques humaines.

- Données du recensement des Etats-Unis : données démographiques des États-Unis tirées des recensements de 1980, 1990 et 2000.

Malheureusement, même sur *AWS*, il est très difficile de trouver en téléchargement libre des données sous forme d'entrepôt de données *ROLAP*. Les entreprises sont plus confidentielles par rapport à la divulgation de ce type de données.

Le but de l'essai est de vérifier la performance de *HDFS* avec des entrepôts de données de plusieurs tailles. Microsoft, propose un entrepôt de données en téléchargement libre pour faciliter les essais sur *MS SQL Server*. ***AdventureWorksDW2012*** [19] est très populaire comme entrepôt de données, mais sa limitation est qu'il est de petite taille selon *Steven Wang* (Annexe I).

2.4.3 Diagramme de base de données

L'entrepôt de données gratuit *AdventureWorksDW2012* de *MS SQL Server* est muni de plusieurs diagrammes, où chaque diagramme contient au moins une table de mesure et une ou plusieurs tables de dimensions. Pour cet essai, on se limitera uniquement au diagramme « ***Reseller Sales*** » qui semble être suffisant avec la table centrale de mesure « ***FactResellersSales*** ». Le diagramme utilisé est présenté dans la figure 2.12.

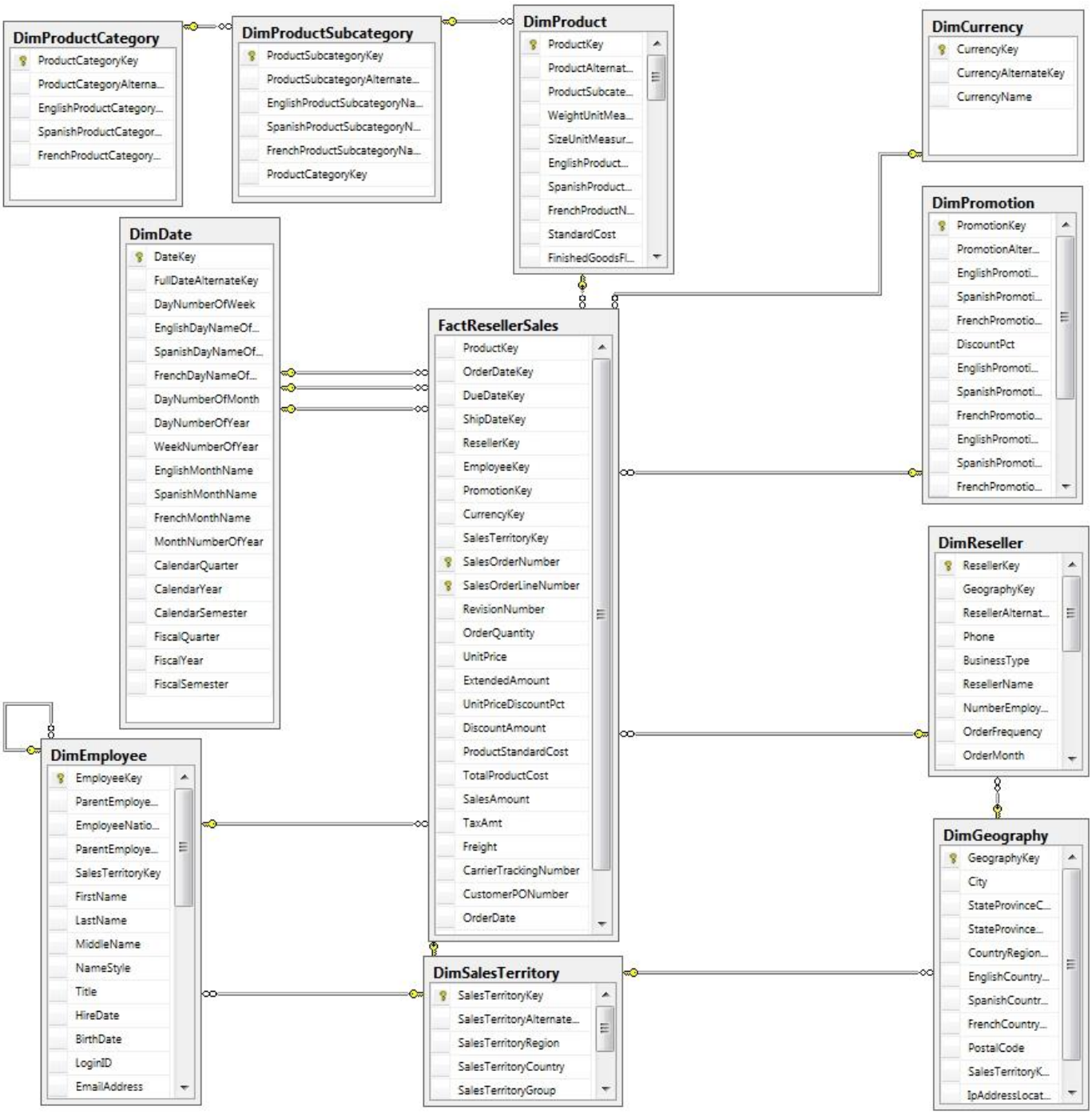


Figure 2.12 Description du diagramme « Reseller Sales »

Source : Microsoft AdventureWorksDW2012

2.4.4 Redimensionnement de la base de données

La base de données utilisé est limitée au diagramme « ***Reseller Sales*** **»** défini précédemment. Le nombre d'enregistrements initial des tables de ce diagramme proposé par Microsoft peut servir uniquement pour valider un entrepôt de petite taille. Le script SQL développé par « *Steven Wang* » (Annexe I) sera utilisé afin d'élargir l'entrepôt de données *AdventureWorksDW2012* en moyenne et grande taille. En utilisant ce script, il est possible de générer un entrepôt de données de plusieurs tailles. Il ne sera pas nécessaire de redimensionner toutes les tables mais uniquement :

- **FactResellerSales** vers :
    - Table de fait de taille moyenne : FactResellerSalesMedium;
    - Table de fait de taille grande : FactResellerSalesBig.
- **DimReseller** vers :
    - Table de dimension de taille moyenne : DimResellerMedium;
    - Table de dimension de taille grande : DimResellerBig.

On considère FactResellerSales et DimReseller comme des tables de petite taille (Annexe I).

Tableau 2.5 Redimensionnement de l'entrepôt de test

Table | Taille	**Petite (initial)**	**Moyenne**	**Grande**
FactResellerSales	FactResellerSales 60 855 lignes	FactResellerSalesMedium ~ 60 millions lignes	FactResellerSalesBig ~ 608 millions lignes
	Multiplicateur : 1	Multiplicateur : 1 000	Multiplicateur : 10 000
DimReseller	DimReseller 701 lignes	DimResellerMedium ~ 700 milles lignes	DimResellerBig ~ 7 millions lignes
	Multiplicateur : 1	Multiplicateur : 1 000	Multiplicateur : 10 000

2.4.5 Exportation de la base de données vers MySQL, HDFS

Les tables redimensionnées en plusieurs tailles seront copiées de *SQL Server* vers le serveur *HDFS* en utilisant l'utilitaire « *Apache Sqoop* », ensuite de *HDFS* vers MySQL en utilisant le même outil également avec la fonction « *Sqoop export* ».

Pour importer les tables de SQL Server vers HDFS :

```
sqoop import --connect
"jdbc:sqlserver://SQLServerHost;user=MyUser;password=MyPassword;
databaseName=AdventureWorksDW2012" --table "myTable" --hive-import
--hive-table myTable --create-hive-table --hive-overwrite
--hive-database inf796 -m 1
```

Pour exporter les tables de HDFS vers MySQL :

```
sqoop export --connect "jdbc:mysql://MySQLHost:3306/myDB"
--username MyUser --password MyPassword --direct --table myTable
--export-dir /user/hive/warehouse/inf796.db/myTable –m 1
```

Les données sont chargées dans des tables relationnelles standard sur le serveur *MySQL* et le moteur utilisé est *MyISAM,* car il est connu par ses performances en lecture par rapport à d'autres moteurs tels qu'InnoDB [20].

Les index nécessaires à la jointure entre les tables dimensions et fait, ainsi que les colonnes à chercher sont ajoutés également sur *MySQL* et *MS SQL Server*. Le serveur *Hadoop* contient un système *HDFS* pour le stockage des données, ainsi que les moteurs *Hive*, *Impala*, *Spark SQL* et *Presto*. Aucune notion d'index n'est disponible sur le système *HDFS*.

2.4.6 Exécution des requêtes SQL

Les types de requêtes SQL utilisés durant les tests d'interrogation de données sont :

- Requêtes de type balayage : utilisation de « *Where* »;
- Requêtes de type jointure : utilisation de « *Join* »;
- Requêtes de type agrégation : utilisation de « *Group By* »;
- Requêtes de type triage : utilisation de « *Order By* ».

Ces types de requêtes sont les plus utilisées dans le domaine des entrepôts de données. Les requêtes SQL de type insertion, mise à jour et suppression ne seront pas évaluées puisque le but de l'expérimentation est de valider les performances d'interrogation de données (lecture).

2.4.7 Mesure du temps d'exécution

Les mesures du temps d'exécution ne semblent pas être une tâche complexe, car les outils *MS SQL Server, MySQL, Hive, Impala, Spark SQL* et *Presto* affichent à la fin de chaque exécution SQL le temps entre le déclenchement du calcul et la fin de traitement.

2.4.8 Présentation des résultats

Voici le gabarit utilisé pour la présentation des résultats :

Tableau 2.6 Gabarit de collecte des mesures

Moteur SQL	MS SQL Server 2012
Type de requête	Balayage / Jointure / Agrégation / Triage
Requête SQL	Select <Content> From <Fact> Join <Dim> Where <Condition> Group by <Group> Order by <Order>
Taille des tables	FAIT : 50 millions d'enregistrements DIMENSION : 500 milles d'enregistrements
Temps d'exécution	3.58 secondes

2.4.9 Analyse des résultats

Les résultats obtenus à travers les exécutions des différentes requêtes SQL sur les moteurs *MS SQL Server 2012, MySQL, Hive, Impala, Spark SQL* et *Presto* vont permettre une analyse et valider la théorie d'avantages en performances sur *HDFS*.

On définit :

- ET_X_Y : le temps d'exécution d'une requête SQL sur un serveur configuré avec X GB de mémoire sur un entrepôt de donnée de taille Y.
- Dans cet essai, on utilisera :
    - ET_8_P, ET_24_P (P étant un entrepôt de taille petite);
    - ET_8_M, ET_24_M (M étant un entrepôt de taille moyenne);
    - ET_8_G, ET_24_G (G étant un entrepôt de taille grande).

On considère qu'un système A est plus performant qu'un système B avec une configuration de mémoire X sur un entrepôt de taille Y pour une requête SQL donnée si :

$$ET\_X\_Y_{(A)} < ET\_X\_Y_{(B)}$$

On validera que *HDFS* est avantageux en termes de performances dans une configuration mémoire X, dans un entrepôt de taille Y si :

$$ET\_X\_Y_{(HDFS)} < ET\_X\_Y_{(SQL\ SERVER)}$$

OU

$$ET\_X\_Y_{(HDFS)} < ET\_X\_Y_{(MySQL)}$$

2.5 Conclusion

À travers ce chapitre, on constate que la structure de données d'un système *HDFS* avec sa réplication entre les nœuds ainsi que la taille des blocs de données qui est supérieure à un système de fichier traditionnel donnent plus de flexibilité pour traiter des données plus rapidement d'une façon distribuée et parallèle.

Ce système de fichier est la base d'un grand nombre d'outils et d'applications qui utilisent un traitement distribué avec une gestion optimisée des ressources avec l'usage de *Yarn*.

Les outils qui permettent d'interroger les données sur *HDFS* dans un mode SQL ou par script (voir 2.2.7 et 2.2.8) sont là pour réduire la complexité d'accès aux données(*MapReduce*) sur *HDFS,* et constituent une couche de haut niveau plus accessible par des utilisateurs habitués à une syntaxe SQL. Les distributions de *Hadoop* facilitent également tout le mécanisme de gestion et de configuration des nœuds.

L'approche pour valider l'intérêt de l'usage de *HDFS* dans le contexte des entrepôts de données *ROLAP* va permettre de tester plusieurs tailles comme échantillon de données sur des *SGBG* relationnelles classiques (*MS SQL Server, MySQL*) ainsi que sur des outils qui se base sur *HDFS* comme *Hive, Impala, Presto,* et *Spark SQL*.

Une comparaison par la suite sera faite entre tous ces systèmes en mesurant le temps d'exécution dans des conditions de mémoires et de taille d'entrepôts différentes. Ces mesures permettront à travers une analyse de valider ou non l'avantage en termes de performances de l'utilisation de *HDFS* dans les entrepôts de données *ROLAP*.

Le chapitre suivant présentera les résultats comparatifs de l'expérience décrite dans l'approche précédente (2.4).

Chapitre 3

Résultats comparatifs

Cette section est dédiée à la présentation des résultats obtenus suite à l'exécution de la démarche décrite dans le chapitre précèdent.

À partir de l'entrepôt de données de Microsoft *AdventureWorksDW2012* on a appliqué un redimensionnement sur les données pour générer trois entrepôts de tailles différentes : Petit [P], Moyen [M] et Grand [G] (Annexe I).

Les trois entrepôts de données ont été transférés sur des serveurs identiques en termes de ressources matérielles et équipés des technologies :

- Microsoft SQL Server
- MySQL
- HDFS
    - Hive
    - Impala
    - Spark SQL
    - Presto

Différents types de requêtes SQL (balayage, jointure, agrégation et triage) qui sont largement utilisées dans le monde du ROLAP ont été exécutées sur chacune des technologies dans les mêmes environnements et dans des configurations de mémoire de taille petite et grande. Le choix de ces deux configurations de mémoire est justifié par le but d'essayer de détecter des changements importants en performance à travers un grand écart en taille de mémoire.

Les mesures du temps d'exécution en secondes ont été récoltées et groupées dans des tableaux afin de faciliter une comparaison et une analyse des résultats obtenus dans le but de la validation de la théorie concernant les avantages de *HDFS* en termes de performances.

3.1 Requête SQL de type balayage « *Where* »

Tableau 3.1 Mesure SQL de type balayage

Type de requête	**Balayage**					
Requête SQL	Select SalesOrderNumber, OrderQuantity, SalesAmount, TaxAmt From FactResellerSales[Medium/Big] Where ShipDateKey =20080608 and PromotionKey = 3 and SalesAmount < 5;					
Taille des tables (# lignes)	• Fact ○ FactResellerSales / Medium / Big : 60855 / 60855000 / 608550000 • Dim : -					
8 GB RAM ET_8 (sec)	**Microsoft SQL Server**	**MySQL MySQL**	**HDFS Hive**	**Impala**	**Spark SQL**	**Presto**
ET_8_P	1	1	1	1	1	1
ET_8_M	103	30	20	12	135	37
ET_8_G	953	745	1438	97	887	261
24 GB RAM ET_24 (sec)	**SQL Server**	**MySQL**	**Hive**	**Impala**	**Spark SQL**	**Presto**
ET_24_P	1	1	2	1	1	1
ET_24_M	98	12	15	10	89	31
ET_24_G	948	705	893	85	656	208

3.2 Requête SQL de type jointure « *Join* »

Tableau 3.2 Mesure SQL de type jointure

Type de requête	**Jointure**
Requête SQL	Select R.ResellerName, R.NumberEmployees, F.SalesAmount, P.FrenchPromotionName From FactResellerSales[Medium/Big] F Join DimReseller[Medium/Big] R on R.ResellerKey = F.ResellerKey Join DimDate D on D.DateKey = F.ShipDateKey Join DimPromotion P on P.PromotionKey = F.PromotionKey Where F.ShipDateKey = 20080608 and P.PromotionKey = 3 and SalesAmount < 5;

Taille des tables (# lignes)	• Fact
	  ○ FactResellerSales / Medium / Big : 60855 / 60855000 / 608550000
	• Dim :
	  ○ DimReseller / Medium / Big : 701 / 701000 / 7010000
	  ○ DimDate : 2191
	  ○ DimPromotion : 16

8 GB RAM ET_8 (sec)	**Microsoft SQL Server**	**MySQL MySQL**	**HDFS Hive**	**Impala**	**Spark SQL**	**Presto**
ET_8_P	1	1	106	1	1	1
ET_8_M	102	179	457	14	43	31
ET_8_G	988	3213	6038	113	408	152
24 GB RAM ET_24 (sec)	**SQL Server**	**MySQL**	**Hive**	**Impala**	**Spark SQL**	**Presto**
ET_24_P	1	1	222	1	1	1
ET_24_M	99	162	212	12	40	31
ET_24_G	946	2758	2540	105	318	151

3.3 Requête SQL de type agrégation « *Group By* »

Tableau 3.3 Mesure SQL de type agrégation

Type de requête	**Aggregation**					
Requête SQL	Select D.MonthNumberOfYear, R.BusinessType, P.FrenchPromotionName, SUM(F.SalesAmount) as TotalSalesAmount From FactResellerSales[Medium/Big] F Join DimDate D on D.DateKey = F.ShipDateKey Join DimReseller[Medium/Big] R on R.ResellerKey = F.ResellerKey Join DimPromotion P on P.PromotionKey = F.PromotionKey Where F.ShipDateKey = 20080608 and P.PromotionKey = 1 Group by D.MonthNumberOfYear, R.BusinessType, P.FrenchPromotionName;					

Taille des tables (# lignes)	• Fact
	  ○ FactResellerSales / Medium / Big : 60855 / 60855000 / 608550000
	• Dim :
	  ○ DimReseller / Medium / Big : 701 / 701000 / 7010000
	  ○ DimDate : 2191
	  ○ DimPromotion : 16					

8 GB RAM ET_8 (sec)	**Microsoft SQL Server**	**MySQL MySQL**	**HDFS Hive**	**Impala**	**Spark SQL**	**Presto**
ET_8_P	1	1	239	1	2	2
ET_8_M	375	104	389	13	340	26
ET_8_G	2689	1068	4064	128	504	218
24 GB RAM ET_24 (sec)	**SQL Server**	**MySQL**	**Hive**	**Impala**	**Spark SQL**	**Presto**
ET_24_P	1	1	131	1	1	1
ET_24_M	362	102	264	13	98	23
ET_24_G	2665	1059	2742	127	457	203

3.4 Requête SQL de type triage «*Order By* »

Tableau 3.4 Mesure SQL de type triage

Type de requête	**Triage**
Requête SQL	Select D.MonthNumberOfYear, R.BusinessType, P.FrenchPromotionName, SUM(F.SalesAmount) as TotalSalesAmount From FactResellerSales[Medium/Big] F Join DimDate D on D.DateKey = F.ShipDateKey Join DimReseller[Medium/Big] R on R.ResellerKey = F.ResellerKey Join DimPromotion P on P.PromotionKey = F.PromotionKey Where F.ShipDateKey = 20080608 and P.PromotionKey = 1 Group by D.MonthNumberOfYear, R.BusinessType, P.FrenchPromotionName Order by D.MonthNumberOfYear, R.BusinessType, TotalSalesAmount Desc;

Taille des tables (# lignes)	• Fact
	  ○ FactResellerSales / Medium / Big : 60855 / 60855000 / 608550000
	• Dim :
	  ○ DimReseller / Medium / Big : 701 / 701000 / 7010000
	  ○ DimDate : 2191
	  ○ DimPromotion : 16

8 GB RAM ET_8 (sec)	**Microsoft SQL Server**	**MySQL MySQL**	**HDFS Hive**	**Impala**	**Spark SQL**	**Presto**
ET_8_P	2	1	158	1	2	1
ET_8_M	402	102	286	13	131	26
ET_8_G	2848	1018	2925	127	503	265
24 GB RAM ET_24 (sec)	**SQL Server**	**MySQL**	**Hive**	**Impala**	**Spark SQL**	**Presto**
ET_24_P	1	1	156	1	1	1
ET_24_M	361	91	271	14	91	21
ET_24_G	2685	893	2735	125	471	198

3.5 Comparaison graphique

Le temps d'exécution sur HDFS est représenté par le meilleur temps d'exécution des outils utilisés dans l'essai : Hive, Spark SQL, Presto et Impala.

La figure 3.1 présente une comparaison de performance en temps d'exécution entre SQL Server, MySQL et HDFS pour les requêtes SQL de type balayage.

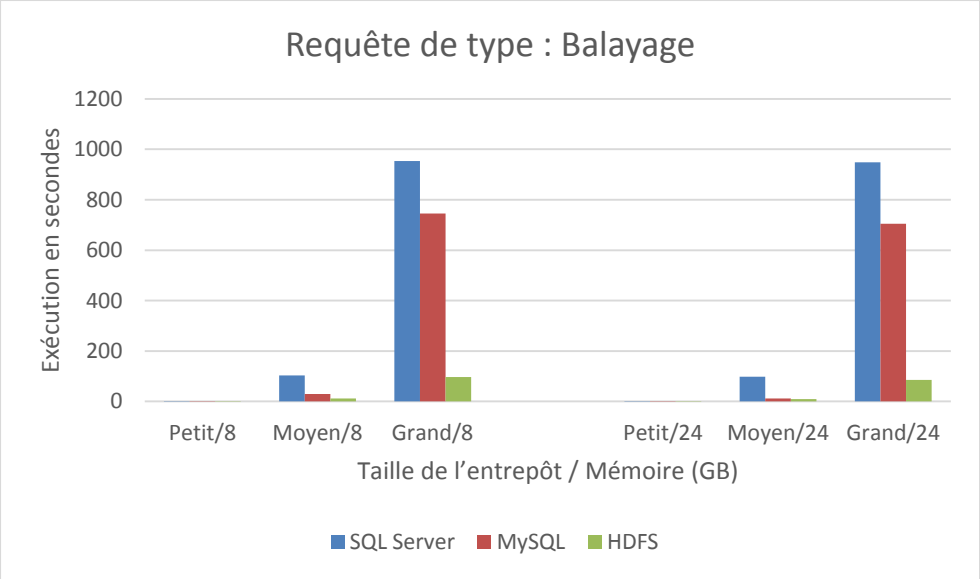


Figure 3.1 Comparaison des résultats pour la requête SQL de type balayage

La figure 3.2 présente une comparaison de performance en temps d'exécution entre SQL Server, MySQL et HDFS pour les requêtes SQL de type jointure.

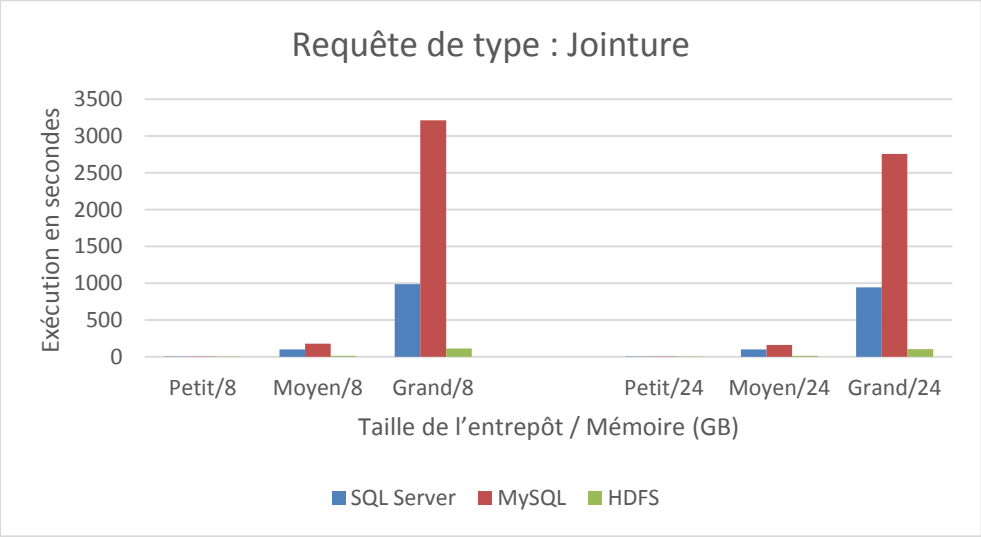


Figure 3.2 Comparaison des résultats pour la requête SQL de type jointure

La figure 3.3 présente une comparaison de performance en temps d'exécution entre SQL Server, MySQL et HDFS pour les requêtes SQL de type agrégation.

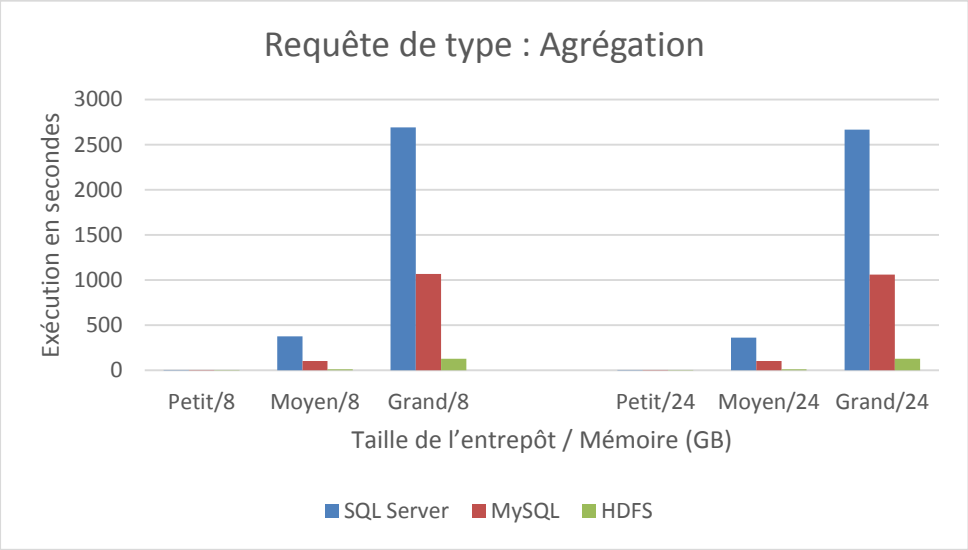


Figure 3.3 Comparaison des résultats pour la requête SQL de type agrégation

La figure 3.4 présente une comparaison de performance en temps d'exécution entre SQL Server, MySQL et HDFS pour les requêtes SQL de type triage.

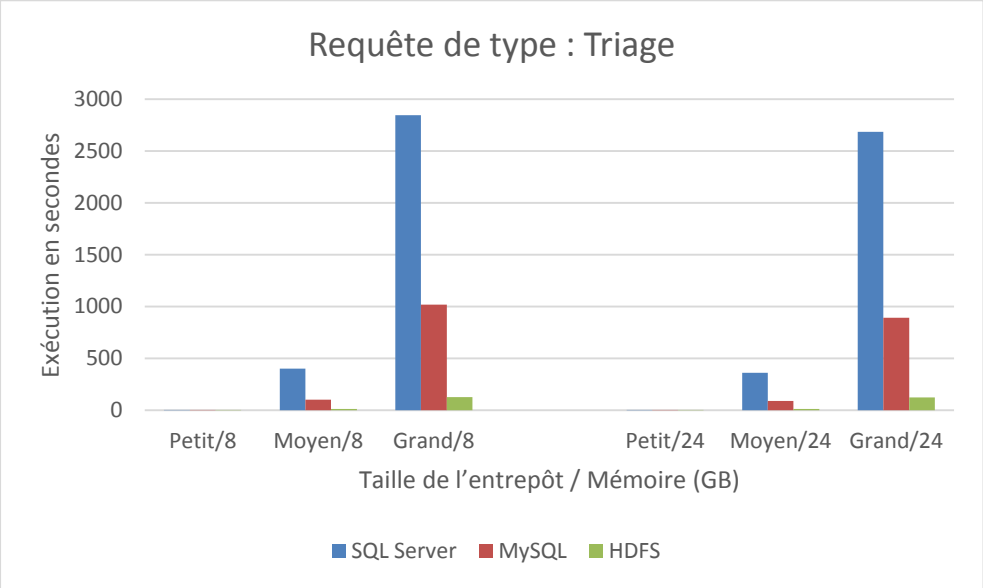


Figure 3.4 Comparaison des résultats pour la requête SQL de type triage

Chapitre 4

Analyse des résultats

4.1 Introduction

Un test de performance sur les requêtes SQL les plus utilisées dans le monde du *ROLAP* a été effectué dans des systèmes différents incluant *SQL Server*, *MySQL* ainsi que sur des technologies qui se basent sur *HDFS* comme système de fichiers.

Les résultats, présentés dans le chapitre précédant, couvrent les requêtes SQL de type balayage, jointure, agrégation et de triage dans deux environnements différents de mémoires vives sur le système (sans supplément de *RAM*, avec supplément de *RAM*).

Les requêtes SQL ont été exécutées sur 3 entrepôts de données redimensionnés (petit, moyen, grand), et le temps d'exécution a été mesuré par la suite.

Ces résultats permettront d'analyser l'efficacité de chaque système selon plusieurs axes, ce qui aidera également à valider la théorie quant aux avantages de *HDFS* en termes de performances sur les entrepôts de données de type *ROLAP* dans des contextes différents.
Les sections suivantes sont consacrées aux analyses des résultats obtenus selon la taille des données, le type de requête SQL ainsi que la taille de mémoire utilisée durant les tests.

La validation de la théorie pourra être établie après cette analyse si :

$$ET\_X\_Y_{(HDFS)} < ET\_X\_Y_{(SQL\ SERVER)}\ OU$$

$$ET\_X\_Y_{(HDFS)} < ET\_X\_Y_{(MySQL)}$$

ET : temps d'exécution de la requête SQL
X : taille de mémoire vive sur le système
Y : taille de l'entrepôt de données

4.2 Contexte des analyses

Le contexte des analyses se limite aux bases de données qui rentrent en jeu dans la comparaison avec *HDFS*. Il s'agit de deux joueurs importants dans le monde des bases de données relationnelles : *SQL Server* et *MySQL*.

Le contexte des analyses se limite également aux deux options de la taille de la mémoire vive (*RAM*) utilisée durant cet essai (8GB et 24 GB), le type de serveur choisi avec deux processeurs à double cœur chacun de type *Xeon* à trois GHZ, la vitesse des disques durs à 7200 *RPM* ainsi que la taille des entrepôts de données où le nombre d'enregistrements maximal pour une table de fait atteint 600 millions d'enregistrements et environ sept millions pour une table de dimension. Cette analyse est spécifique à la distribution de *Hadoop* utilisée dans cette étude qui est *Cloudera Express 5.4.0* (Annexe II) ainsi que le type d'installation de *Hadoop* qui est le mode « *Standalone* » (un seul nœud).

4.3 Facteur de la virtualisation

Les résultats représentent une vue virtuelle du monde réel puisque les tests ont été effectués sur des systèmes virtuels (virtualisation avec *VMware*). Théoriquement, le rendement est meilleur sans la virtualisation, avec un usage exclusif des ressources systèmes. Les résultats peuvent présenter quand même une idée assez proche de la réalité.

4.4 Facteur des versions des applications

Les performances peuvent changer d'une version à l'autre. Le facteur de versions concerne *Hadoop*, son écosystème, les systèmes de base de données *SQL Server* et *MySQL* ainsi que

celles des systèmes d'exploitation. Généralement une nouvelle version est dotée de plus de correctifs et de performances qu'une ancienne version.

4.5 Facteur de la taille des données

Une information importante peut être remarquée depuis les tableaux et graphiques du chapitre précédent. Cette information, concernant le temps d'exécution, semble être valide dans toutes les technologies utilisées, avec n'importe qu'elle taille de mémoire dans le système.

En effet, le temps d'exécution est fortement relié à la taille de l'entrepôt de données :

$$ET\_X\_P_{(SQL\ SERVER)} < ET\_X\_M_{(SQL\ SERVER)} < ET\_X\_G_{(SQL\ SERVER)}$$

$$ET\_X\_P_{(MySQL)} < ET\_X\_M_{(MySQL)} < ET\_X\_G_{(MySQL)}$$

$$ET\_X\_P_{(HDFS)} < ET\_X\_M_{(HDFS)} < ET\_X\_G_{(HDFS)}$$

Il est difficile de faire une comparaison en termes de performance sur un entrepôt de petite taille, vu que le temps d'exécution est presque semblable et minime à l'exception de Hive qui semble être lent même avec un petit entrepôt de données. Un point important qu'on peut constater est que l'usage de *HDFS* en utilisant *Hive* n'est pas avantageux. La lenteur de *Hive* peut être expliquée par la nature du fonctionnement de ce dernier qui se base sur le *MapReduce* contrairement aux autres technologies comme *Impala*, *Spark SQL* et *Presto* [7] [4] [11] [26].

Cependant, les résultats montrent que *HDFS* avec Impala est plus performant que *MySQL* et *SQL Server* et ce, même si l'entrepôt de données augmente en taille.
Ce qui est intéressant c'est que le temps d'exécution du côté d'*Impala* n'augmente pas d'une façon considérable lorsqu'on passe d'un entrepôt de taille moyenne à un entrepôt de grande taille, ce qui n'est pas le cas pour les autres technologies. À titre d'exemple, pour la requête SQL de type triage (qui utilise également de la jointure et de l'agrégation) sur un entrepôt de taille moyenne puis sur un entrepôt de grande taille et avec une taille de mémoire de 24GB :

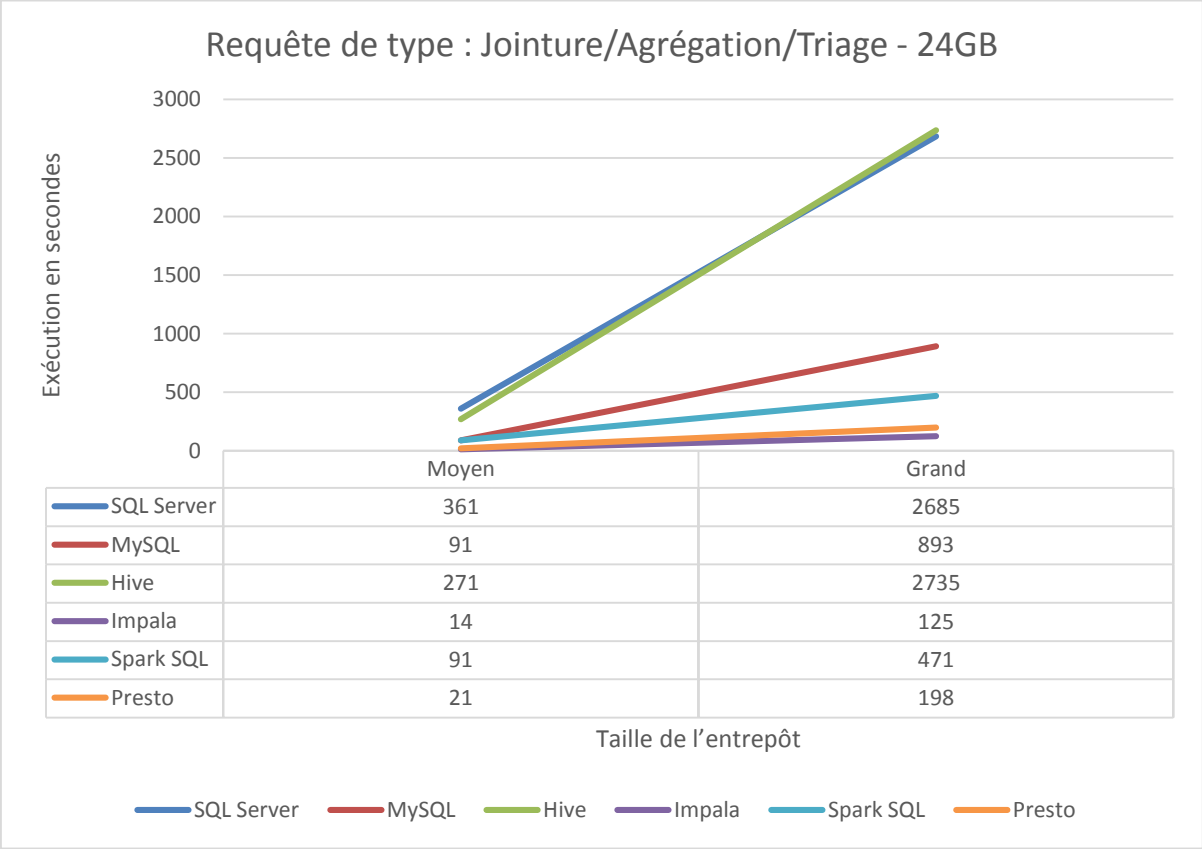


Figure 4.1 Performances avec le facteur de la taille de données

Impala passe de 14 secondes à 125 secondes avec un facteur d'augmentation de 1.85 minute, alors que *SQL Server* passe de 361 secondes à 2685 secondes avec un facteur d'augmentation de 38.73 minutes. Les mêmes conditions présentent un facteur d'augmentation de 13.26 minutes pour *MySQL*, de 6.33 minutes pour *Spark SQL* et de 2.95 minutes pour *Presto*.

On ne peut pas affirmer que *HDFS* en utilisant *Impala* peut préserver ce gain en temps d'exécution si on augmente encore la taille de l'entrepôt, mais les indicateurs dans le contexte de cette étude confirment que *HDFS* est un environnement intéressant pour stocker les entrepôts de données de type *ROLAP* avec *Impala* comme outil d'interrogation de l'entrepôt.

4.6 Facteur des ressources système

4.6.1 La quantité de mémoire disponible

Dans cet essai, les mesures du temps d'exécution de chaque requête SQL étaient influencées par le facteur de mémoire vive dans le système. En effet, la même requête SQL est exécutée dans un environnement à 8GB ainsi que dans un autre à 24GB qu'on considère théoriquement et respectivement « système à faible mémoire » et « système à haute mémoire ».

Ce qu'on peut remarquer est que théoriquement et même pratiquement avec les résultats obtenus, le temps d'exécution d'une requête SQL diminue considérablement dans la majorité des cas où on augmente la quantité de mémoire vive à l'exception de :

- *Presto* avec une requête SQL de type jointure;
- *Impala* et *SQL Server* avec une requête SQL de type agrégation ou triage.

Ces trois exceptions démontrent que le fait de passer de 8GB de mémoire à 24GB n'a pas amélioré les performances. Cela dit, cette analyse peut ne pas être valide dans un autre contexte (processeur plus lent/rapide, disques durs plus lents/rapides, taille de l'entrepôt plus grande, non-usage de la virtualisation). *Impala* à titre d'exemple, recommande des nœuds avec une taille de mémoire de 64GB à 128GB chacun dans des contextes de « *Big Data* » [7].
À titre d'exemple, pour la requête SQL de type triage (qui utilise également de la jointure et de l'agrégation) sur un grand entrepôt, avec une taille de mémoire de 8GB, puis sur un autre avec une taille de mémoire de 24GB :

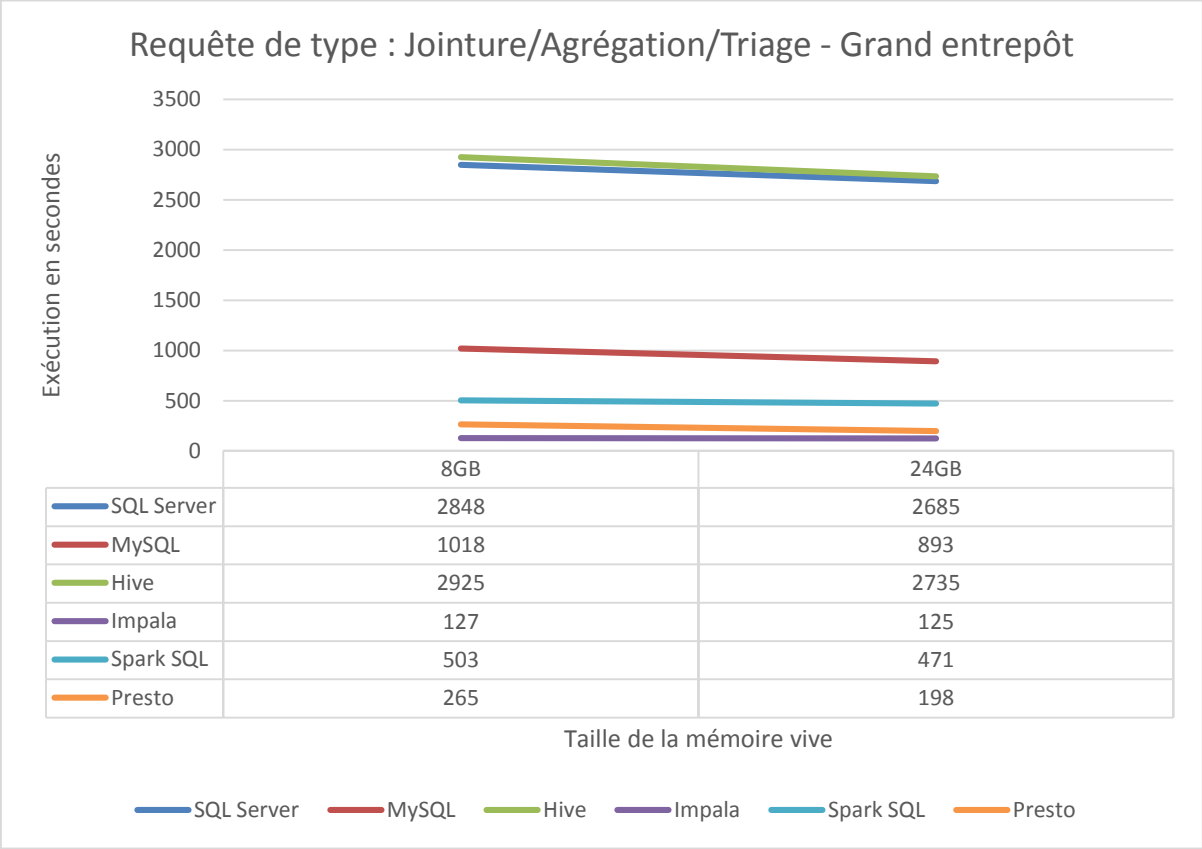


Figure 4.2 Performances avec le facteur de la taille de mémoire

L'effet de la mémoire vive dans cet exemple montre une certaine réduction dans le temps d'exécution, mais qui n'est pas assez important surtout pour les cas de *Spark SQL* et *Impala*. Cela dit, en aucun cas, le fait de rajouter de la mémoire vive n'a engendré une dégradation des performances dans les tests effectués.

On peut résumer l'effet de la mémoire vive pour cette expérience par :

$$ET\_8\_Y_{(SQL\ SERVER)} < ET\_24\_Y_{(SQL\ SERVER)} < ET\_24+\_Y_{(SQL\ SERVER)}$$

$$ET\_8\_Y_{(MySQL)} < ET\_24\_Y_{(MySQL)} < ET\_24+\_Y_{(MySQL)}$$

$$ET\_8\_Y_{(HDFS)} < ET\_24\_Y_{(HDFS)} < ET\_24+\_Y_{(HDFS)}$$

4.6.2 Autres ressources

Durant cet essai, deux quantités de la ressource mémoire vive ont été utilisées afin de mieux analyser les résultats selon l'axe de mémoire. D'autres ressources qui sont hors du contexte de cette analyse peuvent améliorer/dégrader les performances sur chaque système.
Parmi ces ressources, on peut citer :

- Le(s) processeur(s) utilisé(s) (type, vitesse, cache, nombre de cœurs, nombre de processeurs);
- La mémoire (taille, vitesse);
- Le(s) disque(s) dur(s) utilisé(s) (type *HD*/*SSD*/autre, vitesse de rotation, taille tampon, redondance *RAID* 0/1/5/autre, fragmentation sur le disque);
- Le serveur (qualité de carte mère, système de ventilation);
- Les nœuds (nombre, « *Standalone* », « *Cluster* »);
- Le réseau (vitesse du réseau, distance entre les nœuds).

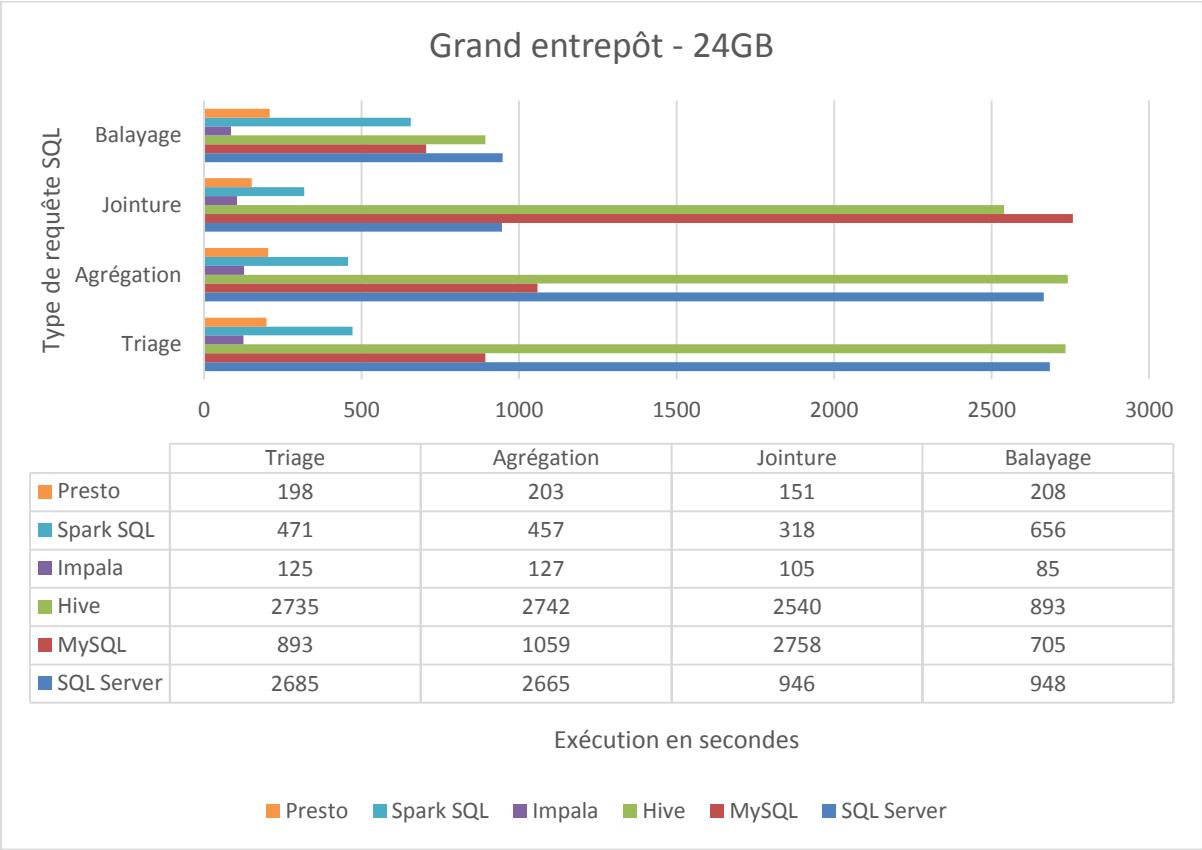
4.7 Facteur du type de requête SQL

Les requêtes SQL utilisées dans cette analyse sont :

- Type balayage : usage de la clause « *WHERE* »;
- Type jointure : usage de la clause « *JOIN* »;
- Type agrégation : usage de la clause « *GROUP BY* »;
- Type triage : usage de la clause « *ORDER BY* ».

À noter que la requête de type agrégation engendre un usage de jointure, et que la requête de type triage engendre un usage de jointure et d'agrégation. Le but était de compliquer au maximum la requête SQL avant de la soumettre aux engins SQL pour couvrir les cas extrêmes.

À titre d'exemple, pour un entrepôt de grande taille, avec une taille de mémoire de 24GB :



	Triage	Agrégation	Jointure	Balayage
Presto	198	203	151	208
Spark SQL	471	457	318	656
Impala	125	127	105	85
Hive	2735	2742	2540	893
MySQL	893	1059	2758	705
SQL Server	2685	2665	946	948

Figure 4.3 Performances avec le facteur de type de requête SQL

Ce qu'on peut remarquer est que la nature de la requête SQL joue un rôle important sur le temps d'exécution. En effet, dans la majorité des cas, ça prend moins de temps pour balayer un entrepôt de données avec une condition précise que de faire des opérations de jointures, d'agrégation ou de triage.

Ce qu'on peut constater également avec une grande quantité de données :

- la variation du temps d'exécution par changement de type de requête SQL est minime avec *Impala et Presto*, donc une meilleure performance pour ces deux outils;
- le balayage avec une condition est lent sur *Spark SQL* par rapport à d'autres types de requêtes SQL;

- *Hive* semble lent, peu importe le type de requête SQL;
- *MySQL* ne performe pas bien quand il s'agit de jointures;
- *SQL Server* performe mieux avec le balayage et la jointure (*même s'il est lent par rapport aux autres*) que l'agrégation et le triage;
- *Impala* performe mieux que *MySQL* et *SQL Server*.

On peut résumer l'effet du type de requête SQL pour cette expérience par :

$ET\_X\_Y_{(* \text{ -Balayage})} < ET\_X\_Y_{(* \text{ -Autres types})}$ exception : Spark SQL

$ET\_X\_Y_{(\text{HDFS -Balayage})} < ET\_X\_Y_{(\text{MySQL | SQL Server -Balayage})}$

$ET\_X\_Y_{(\text{HDFS -Jointure})} < ET\_X\_Y_{(\text{MySQL | SQL Server -Jointure})}$

$ET\_X\_Y_{(\text{HDFS -Agrégation})} < ET\_X\_Y_{(\text{MySQL | SQL Server -Agrégation})}$

$ET\_X\_Y_{(\text{HDFS -Triage})} < ET\_X\_Y_{(\text{MySQL | SQL Server -Triage})}$

4.8 Validation de la théorie

Rappelons le critère de validation de l'hypothèse de recherche :
HDFS est avantageux en termes de performance dans un environnement X, dans un entrepôt de taille Y si :

$ET\_X\_Y_{(\text{HDFS})} < ET\_X\_Y_{(\text{SQL SERVER})}$ OU

$ET\_X\_Y_{(\text{HDFS})} < ET\_X\_Y_{(\text{MySQL})}$

ET : temps d'exécution de la requête SQL
X : taille de mémoire vive sur le système
Y : taille de l'entrepôt de données

Les résultats obtenus ont démontré qu'au moins deux outils (*Presto* et *Impala*) se basant sur *HDFS* comme stockage de l'entrepôt de données *ROLAP* respectent la condition de la théorie dans la majorité des cas. Cela est valide pour les deux environnements de mémoires vives utilisées dans l'expérience ainsi que pour les 3 tailles de l'entrepôt (petit, moyen, grand).

Ce qui attire l'attention est que l'outil *Impala* contrairement à *Presto* a pu valider la théorie dans tous les cas. En effet le temps d'exécution de tout type de requêtes SQL (balayage, jointure, agrégation, triage) sur un entrepôt de taille (petit, moyen, grand) et dans un environnement à 8GB ou 24GB de mémoire est bien meilleur que celui affiché pour *MySQL* et *SQL Server*.

La théorie est validée avec l'outil *Impala*, avec un opérateur **ET** au lieu du **OU** :

$ET\_X\_Y_{(HDFS - Impala)} < ET\_X\_Y_{(SQL\ SERVER)}$ ~~OU~~ ET

$ET\_X\_Y_{(HDFS - Impala)} < ET\_X\_Y_{(MySQL)}$

On peut donc conclure que :

HDFS est avantageux en termes de performance sur les entrepôts de données *ROLAP*

4.9 Recommandations

Les résultats obtenus sont prometteurs et peuvent aider à prendre des décisions quant au choix d'un outil de stockage d'un grand entrepôt de données sur des conditions similaires. Les outils *Impala* de Cloudera et *Presto* de Facebook ont fait preuve de performance durant cet essai.

Ce qu'on peut recommander comme une continuité de ces travaux c'est d'élargir cette analyse en prenant compte d'autres axes et facteurs.
L'analyse pourra être étendue par rapport à l'usage des facteurs suivants :

- Usage de processeurs plus rapides pour les calculs;
- Usage de plus de mémoire vive dans les serveurs;
- Usage de disques durs plus rapides (disques *SSD*);
- Usage de l'infonuagique comme *Amazon* ou *RackSpace;*
- Usage d'un entrepôt de données plus grand (plusieurs téraoctets de données);

- Usage de récentes versions et distributions de *Hadoop*, de son écosystème ainsi que les outils qui se basent sur *HDFS* (*Impala, Presto, Spark*);
- Usage de plusieurs serveurs sous forme de « *Cluster* »;
- Usage de réseau comme infrastructure de communication entre les nœuds du *cluster;*
- Usage de serveurs dédiés au lieu de la virtualisation;
- Usage d'autres modèles de données et autre type de requêtes SQL;
- Usage d'autres modèles de *SGBD* dans la comparaison avec *HDFS;*
- Prise en compte de la concurrence avec plus d'une requête SQL en exécution en même temps;
- Prise en compte d'autres systèmes d'exploitation comme « *RedHat* » et « *CentOS* ».

Conclusion

Les entrepôts de données de type *ROLAP* avec le modèle relationnel, qui aide à maintenir facilement l'intégrité des données, sont de plus en plus populaires [43]. La section 2.1 décrit les avantages qu'apporte ce type d'entrepôt de données.

Avec l'intérêt qu'on porte aujourd'hui au « *Big Data* » et les quantités importantes des informations générées dans les sources de données [44], les entrepôts de type *ROLAP* ne cessent de grandir. Le volume des données dans les tables faits et dimensions de l'entrepôt commence à poser des problèmes de performances aux analystes de données (section 1.2.1).

Hadoop, avec son écosystème et son système de fichiers *HDFS* (section 2.2), propose des solutions de stockage de données de façon à garantir une haute disponibilité ainsi que des outils de manipulation de données de façon distribuée avec le *MapReduce*, *Hive* et autres.

Un des principaux progrès dans ce monde est l'apparition des outils dits « *near real time on Hadoop* » avec des moteurs de calculs plus rapides qui ne se basent plus sur le *MapReduce* et qui sont capables de traiter les données directement sur le système de fichier *HDFS* avec une meilleure gestion de ressources grâce à *Yarn.* On parle plus précisément des outils comme *Cloudera Impala, Facebook Presto et Spark SQL* (section 2.2.8).

L'idée de l'essai est de déterminer si *HDFS* avec ces outils temps réel peut être une solution au problème de performance des entrepôts de données de type *ROLAP*. Autrement dit, est-ce que le fait d'entreposer les tables faits et dimensions directement sur *HDFS* et d'interroger sous forme de SQL les données de ces entrepôts avec des outils à temps réel présente un avantage ou non?

Pour répondre à cette question, l'approche suggérée à la section 2.4 consiste en une étude comparative avec deux *SGBD* classiques *MySQL* et *Microsoft SQL Server*. L'entrepôt de données utilisé dans cet essai est Microsoft *AdventureWorks2012* disponible en téléchargement libre. Un Script (en annexe I) a été utilisé pour redimensionner cet entrepôt afin d'avoir les trois tailles (petit, moyen et grand). Ces trois entrepôts étaient stockés en utilisant la même infrastructure (section 2.4.1) sur Microsoft *SQL Server* puis exportés vers *MySQL* et *HDFS* avec l'outil *Sqoop* (section 2.2.4).

Concernant les tests, plusieurs types de requêtes SQL (balayage, jointure, agrégation et triage) ont été exécutés sur les systèmes *MySQL*, Microsoft *SQL Server* et *HDFS* (*Hive, Impala, Presto et Spark SQL*) dans les trois types d'entrepôts de données (de petite taille, de taille moyenne et de grande taille) et avec des configurations de mémoires vives de 8 GB, puis de 24 GB. Le temps d'exécution de chaque type de requête SQL sur chaque système dans un entrepôt de taille Y et une configuration de mémoire X été mesuré. Le Chapitre 3 présente les résultats comparatifs de notre expérience.

L'analyse de ces résultats (Chapitre 4), qui se limite au contexte de cet essai et des outils utilisés, considère que les résultats sont prometteurs. En effet, après l'entreposage sur *HDFS* puis l'interrogation des données avec *Impala et Presto*, il apparait que ces derniers présentent de meilleures performances que *MySQL* et Microsoft *SQL Server* surtout quand la quantité des données dans l'entrepôt devient importante (entrepôt de grand taille pour le contexte de l'essai).

Avec ces résultats, on a pu valider la théorie de l'essai qui était la suivante :
Le stockage d'un entrepôt de données sur un système *HDFS* est avantageux en termes de performances d'extraction de données en mode *ROLAP*.

Les avantages en utilisant le stockage de l'entrepôt sur *HDFS* sont clairement de type quantitatif vu qu'il s'agit de réduction de temps de réponses à des requêtes SQL.

D’autres avantages peuvent être intéressants également pour des travaux futurs et qui peuvent être étudiés, tel que :

- la réduction de coûts en distribuant les calculs sur plusieurs nœuds avec des ressources modestes;
- la diminution du temps de maintenance sur les tables de l’entrepôt (suppression d’indexation avec *HDFS*);
- la haute disponibilité grâce à la réplication des données sur le « *Cluster HDFS* ».

L’analyse de ce travail pourra être étendue avec les recommandations suggérées dans la (section 4.9), tel que l’usage de l’infonuagique avec un *Cluster* de plusieurs nœuds stockant plus de quantité de données et avec des requêtes SQL en mode concurrence.

Ce travail constitue une première brique pour un projet plus large d’usage d’une plateforme de technologie unifiée sur un système de fichier *HDFS* :

- stockage sur *HDFS* des données brutes (non structurées, semi- structurées, structurées) on provenance de différentes sources de données. « Zone de *Staging* »;
- stockage sur *HDFS* des données d’entrepôt de donnée (tables faits et dimensions);
- ETC : extraction des données *Staging* directement du *HDFS*, transformation sur *HDFS* en mode distribué et chargement de données vers *HDFS* également;
- usage des outils à temps réel pour l’analyse de données des entrepôts stockés directement sur *HDFS*;

Liste des références

[1] Amazon web services. (2015). *Ensembles de données publics AWS,* http://aws.amazon.com/fr/public-data-sets/, consulté le 28 décembre 2015.

[2] Apache HBase Team. (2015). *Apache HBase ™ reference guide* (Version 2.0.0-SNAPSHOT ed.) Apache.

[3] Apache Software Foundation. (2015). *Apache hadoop NextGen MapReduce (YARN),* https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, consulté le 28 décembre 2015.

[4] Apache Spark. (2015). *Spark, Lightning-fast cluster computing,* http://spark.apache.org/, consulté le 28 décembre 2015.

[5] Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., et al. (2011). Apache hadoop goes realtime at facebook. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data,* pp. 1071-1080.

[6] Chen, F., & Hsu, M. (2013). A performance comparison of parallel DBMSs and MapReduce on large-scale text analytics. *Proceedings of the 16th International Conference on Extending Database Technology,* pp. 613-624.

[7] Cloudera, I. (2014). *Cloudera impala, http://www.cloudera.com/content/www/en-us/documentation/archive/impala/2-x/2-1-x/PDF/cloudera-impala.pdf,* consulté le 28 décembre 2015.

[8] Cloudera, I. (2015). *Cloudera hue,* http://gethue.com/, consulté le 28 décembre 2015.

[9] Costa, M., & Madeira, H. (2004). Handling big dimensions in distributed data warehouses using the DWS technique. *Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP*, pp. 31-37.

[10] DELL. *Dell precision™ 490 workstation*, *http://www.dell.com/downloads/emea/products/precn/precn_490_uk.pdf*, consulté le 28 décembre 2015.

[11] Facebook. (2015). *Presto, distributed SQL query engine for big data*, https://prestodb.io/docs/0.105/overview/use-cases.html, consulté le 28 décembre 2015.

[12] Hobi, L. (2013). SQL versus MapReduce, A comparison between two approaches to large scale data analysis.

[13] Inmon, W. H., & Hackathorn, R. D. (1994). *Using the data warehouse* Wiley-QED Publishing.

[14] Javed, A., & Rafique, S. S. (2006). Data warehouse maintenance: Improving data warehouse perfomance through efficient maintenance.

[15] Kimball, R. (1996). The data warehouse toolkit: Practical techniques for building dimensional data warehouses.(ed) john wiley & sons. *Inc., New York, NY, USA.*

[16] Lin, B., Hong, Y., & Lee, Z. (2009). *Data Warehouse Performance.*

[17] Lin, J., & Ryaboy, D. (2013). Scaling big data mining infrastructure: The twitter experience. *ACM SIGKDD Explorations Newsletter, 14*(2), 6-19.

[18] Madden, S. (2012). From databases to big data. *IEEE Internet Computing*, (3), 4-6.

[19] Microsoft. (2013). *adventureworksdw2012,* http://msftdbprodsamples.codeplex.com/downloads/get/670502, consulté le 28 décembre 2015.

[20] Oracle Corporation. (2015). *Mysql MyISAM,* https://dev.mysql.com/doc/refman/5.6/en/myisam-storage-engine.html, consulté le 28 décembre 2015.

[21] Ordonez, C., Song, I., & Garcia-Alvarado, C. (2010). Relational versus non-relational database systems for data warehousing. *Proceedings of the ACM 13th International Workshop on Data Warehousing and OLAP,* pp. 67-68.

[22] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., et al. (2009). A comparison of approaches to large-scale data analysis. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data,* pp. 165-178.

[23] Russell, J. (2013). *Cloudera impala* "O'Reilly Media, Inc.", First Edition.

[24] Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., et al. (2010). MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM, 53*(1), 64-71.

[25] The Apache Software Foundation. (2015). *Apache ambari,* https://cwiki.apache.org/confluence/display/AMBARI/Ambari, consulté le 28 décembre 2015.

[26] The Apache Software Foundation. (2015). *Apache hive,* https://cwiki.apache.org/confluence/display/Hive/Home, consulté le 28 décembre 2015.

[27] The Apache Software Foundation. (2015). *Apache oozie workflow scheduler for Hadoop,* http://oozie.apache.org/docs/4.2.0/DG_Overview.html, consulté le 28 décembre 2015.

[28] The Apache Software Foundation. (2015). *Apache pig,* http://pig.apache.org/docs/r0.14.0/start.html, consulté le 28 décembre 2015.

[29] The Apache Software Foundation. (2015). *Apache ZooKeeper,* https://zookeeper.apache.org/doc/r3.4.7/zookeeperOver.html, consulté le 28 décembre 2015.

[30] The Apache Software Foundation. (2015). *Sqoop user guide,* http://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html, consulté le 28 décembre 2015.

[31] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., et al. (2010). Hive-a petabyte scale data warehouse using hadoop. *Data Engineering (ICDE), 2010 IEEE 26th International Conference on,* pp. 996-1005.

[32] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., et al. (2010). Data warehousing and analytics infrastructure at facebook. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data,* pp. 1013-1020.

[33] VMware.com. (2015). *VMware workstation 11.1.0,* https://my.vmware.com/web/vmware/details?downloadGroup=WKST-1110-WIN&productId=501, consulté le 28 décembre 2015.

[34] White, T. (2012). *Hadoop: The definitive guide* "O'Reilly Media, Inc.", 3rd Edition.

[35] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 4.

[36] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

[37] Apache Software Foundation. (2015). *MapReduce Tutorial*, http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html, consulté le 28 décembre 2015

[38] Wu, S., Vu, Q. H., Li, J., & Tan, K. (2009). Adaptive multi-join query processing in pdbms. *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pp. 1239-1242.

[39] Xu, Y., & Kostamaa, P. (2009). Efficient outer join data skew handling in parallel dbms. *Proceedings of the VLDB Endowment*, *2*(2), 1390-1396.

[40] Xu, Y., Kostamaa, P., & Gao, L. (2010). Integrating hadoop and parallel DBMs. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pp. 969-974.

[41] Xu, Y., Kostamaa, P., Qi, Y., Wen, J., & Zhao, K. K. (2011). A hadoop based distributed loading approach to parallel data warehouses. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pp. 1091-1100.

[42] Chen, Y., Dehne, F. K., Eavis, T., & Rau-Chaplin, A. (2006). Improved Data Partitioning for Building Large ROLAP Data Cubes in Parallel. IJDWM, 2(1), 1-26.

[43] BARC A CXP GROUP COMPANY. (2015). *The World's Largest Annual Survey of BI Users*, http://barc-research.com/bi-survey-15/, consulté le 24 janvier 2016.

[44] IDG Entreprise. (2015). *An IDG Communcations Company*, *http://www.idgenterprise.com/*, http://core0.staticworld.net/assets/2015/03/16/2015-data-and-analytics-survey.pdf consulté le 24 janvier 2016.

[45] Lau, Edmond, and Samuel Madden. "An integrated approach to recovery and high availability in an updatable, distributed data warehouse." *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006.

Bibliographie

Alex Holmes, *Hadoop in Practice,* « Manning Publications Co. », 2012, 536 p.

Arun C. Murthy & Vinod Vavilapalli, *Apache Hadoop YARN*, « Addison-Wesley Data & Analytics Series », 2014, 400 p.

Avkash Chauhan, *Learning Cloudera Impala*, « Packt Publishing Ltd. », 2013, 150 p.

Balmin, A., Kaldewey, T., & Tata, S. (2012). Clydesdale: Structured data processing on hadoop. *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data,* pp. 705-708.

Christopher Adamson, *Star Schema The Complete Reference*, 1st Edition, « Publisher Tata Mcgraw Hill Education Private Limited », 2010, 524 p.

Dayong Du, *Apache Hive Essentials*, « Packt Publishing Ltd. », 2015, 313 p.

Dean Wampler, *Programming Hive*, « O'Reilly Media, Inc », 2012, 352 p.

Ellen Friedman, *Real-World Hadoop*, «O'Reilly Media, Inc. », 2015, 104 p.

Erik Thomsen, *OLAP Solutions: Building Multidimensional Information Systems*, 2nd Edition, « Wiley Computer Publishing », 2002, 696 p.

Holden Karau, *Learning Spark: Lightning-Fast Big Data Analysis*, 1st Edition, « O'Reilly Media, Inc », 2015, 276 p.

Jarek Jarcec Cecho & Kathleen Ting, *Apache Sqoop Cookbook*, 1st Edition, « O'Reilly Media, Inc », 2013, 94 p.

John Russell, *Getting Started with Impala*, 1st Edition, « O'Reilly Media, Inc », 2014, 110 p.

Lars George, *HBase: The Definitive Guide*, « O'Reilly Media, Inc », 2011, 554 p.

Mohammad Kamrul Islam, *Apache Oozie*, « O'Reilly Media, Inc », 2015, 272 p.

Patrick LeBlanc, *Microsoft SQL Server 2012 Step by Step*, 1st Edition, « Step by Step Developer series », 2015, 432 p.

Paul DuBois, *MySQL Cookbook: Solutions for Database Developers and Administrators*, 3rd Edition, « O'Reilly Media, Inc », 2014, 866 p.

Ralph Kimball & Margy Ross, *The Data Warehouse Toolkit*, 3rd Edition, « Wiley Computer Publishing », 2013, 600 p.

Rohit Menon, *Cloudera Administration Handbook*, « Packt Publishing Ltd. », 2014, 255 p.

Saurav Haloi, *Apache ZooKeeper Essentials*, 1st Edition, « Packt Publishing Ltd. », 2015, 169 p.

Thilina Gunarathne, *Hadoop MapReduce v2 Cookbook*, 2nd Edition, « Packt Publishing Ltd. », 2015, 695 p.

Tom White, Hadoop: *The Definitive Guide*, 4th Edition, «O'Reilly Media, Inc. », 2015, 768 p.

W. H. Inmon, *Data Warehouse Performance*, 1st Edition, « Wiley », 1998, 444 p.

Annexe I

Script de redimensionnement de l'entrepôt AdventureWorksDW2012

Source: SQL Server 2012 Columnstore Index In Depth Part 1: Make a bigger AdventureworksDW database for Testing, avril 12, 2012 at 4:00

Steven Wang : Data scientist and a Microsoft Certified Solution Expert on Data Platform, MCSA on SQL server 2008/12 and MCITP on BI Developer, Database Developer and DBA

http://www.msbicoe.com/post/2012/04/12/xVelocity-Memory-Optimized-Columnstore-index-In-Depth-Part-1-Make-a-bigger-AdventureworksDW-database-for-Testing.aspx

DimResellerBig

```
USE [AdventureworksDW2012]
GO

--Create a bigger table for DimReseller
--INSERT INTO DimResellerBig – Use if the table already exists
SELECT
      Cast(Row_Number() Over(Order by (Select 0)) as int) As [ResellerKey]
      , A.[GeographyKey]
      , Cast('AW' + Format(Row_Number() Over(Order by (Select 0)), '0000000000') as
Nvarchar(15)) As [ResellerAlternateKey]
      , A.[Phone] , A.[BusinessType]
      , Cast(A.[ResellerName] +  Format(Row_Number() Over(Order by (Select 0)), '
0000000000') As nvarchar(50)) As ResellerName  , A.[NumberEmployees]
      , A.[OrderFrequency] , A.[OrderMonth]  , A.[FirstOrderYear]  , A.[LastOrderYear]
      , A.[ProductLine]  , A.[AddressLine1]  , A.[AddressLine2]
      , A.[AnnualSales]  , A.[BankName]  , A.[MinPaymentType]
      , A.[MinPaymentAmount]  , A.[AnnualRevenue]  , A.[YearOpened]
INTO DimResellerBig -- Use if the table does not exist
FROM    [dbo].[DimReseller] A
         Cross Join
      Master..spt_values B
Where   B.Type = 'P'
         And   B.Number Between 1 and 50
```

```
--50 times more rows than the original DimReseller Table
--you can make it bigger or smaller by change this number (max by batch 2000)
-- to avoid using batches replace Master..spt_values by a temporary table that contains
numbers from 0 to 10000 for example

GO

 --Add indexes and constraints for DimResellerBig
Alter Table dbo.DimResellerBig Alter Column [ResellerKey] Int Not Null;

ALTER TABLE [dbo].[DimResellerBig] ADD  CONSTRAINT
[PK_DimResellerBIG_ResellerKey] PRIMARY  KEY CLUSTERED
( [ResellerKey] ASC);

CREATE NONCLUSTERED INDEX [IX_DimResellerBig_BusinessType] ON
[dbo].[DimResellerBig]([BusinessType] ASC);

GO


-- Mise à jour des statistiques

use AdventureWorksDW2012;

GO

UPDATE STATISTICS DimResellerBig

GO
```

FactResellerSalesBig

```
 USE [AdventureWorksDW2012]
GO
--The script is used to make a bigger FactResellerSales table, called FactResellerSalesBig.
--When a RowMultiplier 180 is used the FactResellerSalesBig table will be 10,953,900 rows
--The query takes 25 minutes to run in my laptop with a 4-core processor, a 7200 rpm hard
disk and 8 GB RAM


Declare  @OrderCountTable Table (OrderDateKey INT, OrderCount Int)
Declare  @OrderDateKey Int = 2012
          , @OrderCount int
          , @RowMultiplier int = 180  --180 times more rows than the original
FactResellerSales Table
           --you can make it bigger or smaller by change this number

Insert Into @OrderCountTable
SELECT      Distinct [OrderDateKey],
Count(Distinct SalesOrderNumber) As OrderCount
FROM        [dbo].[FactResellerSales]
Group by    [OrderDateKey];


If Object_ID('[dbo].[FactResellerSalesBig]') Is Not Null
       Drop Table [dbo].[FactResellerSalesBig];

Select  Top(0) *
Into    FactResellerSalesBig
From    [dbo].[FactResellerSales];

While Exists (Select * From @OrderCountTable)
   Begin
      Select  Top(1)
            @OrderDateKey = OrderDateKey
            ,  @OrderCount =  OrderCount
      From    @OrderCountTable;

      Insert into FactResellerSalesBig with(Tablock)
      Select
            R.[ProductKey] , R.[OrderDateKey] , R.[DueDateKey] , R.[ShipDateKey]
            , Y.[ResellerKey] , R.[EmployeeKey] , R.[PromotionKey] , R.[CurrencyKey]
```

```
            , Y.[SalesTerritoryKey]
            , Cast(R.[SalesOrderNumber] + Format(Y.RowNum, '000') AS nvarchar(20)) As
SalesOrderNumber
            , R.[SalesOrderLineNumber]
            , R.[RevisionNumber]
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) As OrderQuantity
            , R.[UnitPrice]
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) * R.[UnitPrice] As
ExtendedAmount
            , R.[UnitPriceDiscountPct]
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) * R.[UnitPrice] *
R.[UnitPriceDiscountPct] As DiscountAmount
            , R.[ProductStandardCost]
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) *  R.[ProductStandardCost]
As TotalProductCost
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) * R.[UnitPrice] * (1 -
R.[UnitPriceDiscountPct]) As SalesAmount
            , Ceiling(R.[OrderQuantity] * Y.QuantityMultiplier) * R.[UnitPrice] * (1 -
R.[UnitPriceDiscountPct]) * 0.08 As TaxAmt
            , R.[Freight]
            , Cast(R.[CarrierTrackingNumber] +  Format(Y.RowNum, '-000') As nvarchar(25))
As CarrierTrackingNumber
            , Cast(R.[CustomerPONumber] + Format(Y.RowNum, '000') AS nvarchar(25)) As
CustomerPONumber
            , R.[OrderDate] , R.[DueDate] , R.[ShipDate]
      From
         (
            SELECT
                  [ProductKey] ,[OrderDateKey] ,[DueDateKey] ,[ShipDateKey]
                  ,[ResellerKey] ,[EmployeeKey] ,[PromotionKey] ,[CurrencyKey]
                  ,[SalesTerritoryKey] ,[SalesOrderNumber] ,[SalesOrderLineNumber]
                  ,[RevisionNumber] ,[OrderQuantity] ,[UnitPrice] ,[ExtendedAmount]
                  ,[UnitPriceDiscountPct] ,[DiscountAmount] ,[ProductStandardCost]
                  ,[TotalProductCost] ,[SalesAmount] ,[TaxAmt] ,[Freight]
                  ,[CarrierTrackingNumber] ,[CustomerPONumber] ,[OrderDate]
                  ,[DueDate] ,[ShipDate] , Dense_Rank()
            Over(Partition by [OrderDateKey] Order by SalesOrderNumber) As OrderNumber
            FROM    [dbo].[FactResellerSales]
            Where   OrderDateKey =  @OrderDateKey
         ) R
               Cross Apply
         (
```

```
            SELECT  TOP (@RowMultiplier)
                  A.[ResellerKey]
                  , B.SalesTerritoryKey
                  , Row_Number() Over(Order by Checksum(newid())) As RowNum
                  , RAND(CHECKSUM(NEWID())) * 2 As QuantityMultiplier
            FROM    [DimResellerBig] A
                     Inner join
                  [dbo].[DimGeography] B  on A.[GeographyKey] = B.GeographyKey
                     Cross Join  Master..spt_values C
-- for @RowMultiplier  > 2048 replace Master..spt_values by a temporary table that contains numbers from 0 to 10000 for example
            Where   C.Type = 'P' and
                  C.Number Between 1 and @OrderCount
                  and R.OrderNumber = C.number
         ) Y

      Print 'The records for the order date: ' + Cast(@OrderDateKey as nvarchar(8)) + ' has multiplied ' + Cast(@RowMultiplier as nvarchar(6)) + ' times';

      Delete Top(1) From  @OrderCountTable;
   End
Go

Create Clustered Index IX_FactResellerSalesBig_1 On [dbo].[FactResellerSalesBig] ([ResellerKey] Asc);

CREATE NONCLUSTERED INDEX [IX_FactResellerSalesBig_PromotionKey] ON [dbo].[FactResellerSalesBig]([PromotionKey] ASC);

CREATE NONCLUSTERED INDEX [IX_FactResellerSalesBig_ShipDateKey] ON [dbo].[FactResellerSalesBig]([ShipDateKey] ASC);
GO

--Mise à jour des statistiques

use AdventureWorksDW2012;

GO

UPDATE STATISTICS FactResellerSalesBig

GO
```

Annexe II

Installation de Cloudera Express 5.4.0 (single node)

sur Ubuntu server 12.04.5

Source (Cloudera)

Télécharger « *Cloudera Manager* » :

```
wget http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin
```

Changer les permissions sur le fichier :

```
chmod u+x cloudera-manager-installer.bin
```

Désactiver le « *firewall* »:

```
sudo ufw disable

sudo iptables -F
```

Ajouter au fichier /etc/hosts :

```
127.0.0.1 inf796.essai inf796
```

Changer le hostname:

```
sudo vi /etc/hostname

sudo hostname inf796.essai

sudo /etc/init.d/networking restart
```

Rendre votre utilisateur « *sudo* »:

```
sudo visudo

mon_utilisateur ALL=(ALL) NOPASSWD: ALL
```

Installer un serveur « *SSH* »:

```
sudo apt-get install openssh-server
```

Générer une clé « *SSH* »:

```
ssh-keygen
```

Rendre le serveur accessible par lui-même en utilisant ssh :

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

ssh inf796.essai
```

Désactiver « *IPV6* » sur le serveur :

```
sudo vi /etc/sysctl.conf
```

Ajouter à la fin :

```
net.ipv6.conf.all.disable_ipv6 = 1

net.ipv6.conf.default.disable_ipv6 = 1

net.ipv6.conf.lo.disable_ipv6 = 1
```

Exécuter ou redémarrer le serveur:

```
sudo sysctl -p
```

Corriger le « *swap* »:

```
sudo sysctl -w vm.swappiness=0

sudo vi /etc/sysctl.conf

et ajouter : vm.swappiness = 0
```

Lancer l'installation de « *Cloudera manager* » :

```
sudo ./cloudera-manager-installer.bin
```

Annexe III

Installation de MySQL 5.6.24 sur Ubuntu server 12.04.5

Source : MySQL

Télécharger « *mysql-apt* » pour la version 5.6.24:

```
wget http://dev.mysql.com/get/mysql-apt-config_0.3.3-2ubuntu12.04_all.deb
```

Lancer la configuration de MySQL 5.6.24

```
sudo dpkg -i mysql-apt-config_0.3.3-2ubuntu12.04_all.deb
```

Mettre à jour les références « *APT* »

```
sudo apt-get update
```

Installer MySQL 5.6.24

```
sudo apt-get install mysql-server
```

Annexe IV

Installation de Presto 0.105

Source : prestodb.io

Télécharger et décompresser la version 0.102 de Presto :

```
su - hdfs
wget https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.105/presto-
server-0.105.tar.gz
tar –xzvf presto-server-0.105.tar.gz
ln –s presto-server-0.105.tar.gz
mkdir presto/etc
mkdir presto/data
mkdir presto/catalog
```

Configuration du nœud Presto: créer /etc/node.properties

```
node.environment=test
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
node.data-dir=/var/lib/hadoop-hdfs/presto/data
```

Configuration des logs: créer /etc/log.properties

```
com.facebook.presto=ERROR
```

Configuration du catalogue: créer /etc/catalog/hive.properties

```
connector.name=hive-cdh5
hive.metastore.uri=thrift://inf796.essai:9083
```

Configuration de Presto (JVM): créer /etc/jvm.config

```
-server
-Xmx8G    // Xmx16G ou Xmx24G
-XX:+UseConcMarkSweepGC
-XX:+ExplicitGCInvokesConcurrent
-XX:+AggressiveOpts
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
```

Configuration de Presto (serveur): créer /etc/config.properties

```
coordinator=true
node-scheduler.include-coordinator=true
http-server.http.port=7777
task.max-memory=1GB
discovery-server.enabled=true
discovery.uri=http://inf796.essai:7777
```

Installation et configuration de Java8, 64bits

```
Télécharger: JRE 1.8.0_45 dans le répertoire presto/java
Ajouter au fichier bin/launcher avec la commande exec:
export JAVA_HOME=/var/lib/hadoop-hdfs/presto/java
export PATH=$JAVA_HOME/bin:$PATH
```

Démarrer le serveur Presto avec utilisateur hdfs

```
bin/launcher start   *puis vérifier avec la commande jps
```

Arrêter le serveur Presto avec utilisateur hdfs

```
bin/launcher stop   *puis vérifier avec la commande jps
```

Logs

```
data/var/log/launcher.log
```

```
data/var/log/server.log
data/var/log/http-request.log
```

Exécution du client Presto

```
Télécharger le client :
wget https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.102/presto-cli-0.102-executable.jar

mv presto-cli-0.102-executable.jar presto
chmod +x presto

créer un fichier client:
export JAVA_HOME=/var/lib/hadoop-hdfs/presto/java
export PATH=$JAVA_HOME/bin:$PATH./presto --server localhost:7777 --catalog hive --schema inf796

chmod +x client

Lancer le client avec utilisateur hdfs:
./client
```

Exécution des requêtes SQL depuis le client Presto

```
SELECT * FROM hive.inf796.MyTable
```

Annexe V

Informations techniques

Utilisation de MySQL 5.6 durant l'installation de « *Cloudera* »:

- Créer 2 utilisateurs hive/oozie sur MySQL
- Créer 2 bases de données hive/oozie sur MySQL
- GRANT ALL PRIVILEGES ON hive.* TO 'hive'@'%';
- GRANT ALL PRIVILEGES ON oozie.* TO 'oozie'@'%';
- FLUSH PRIVILEGES;

Corriger le problème de Yarn ou Sqoop reste bloqué durant l'import:

- Augmenter les valeurs : dans Yarn (node manager) :
    - yarn.nodemanager.resource.memory-mb
    - yarn.nodemanager.resource.cpu-vcores

« *Driver* » Microsoft SQL Server 2012 pour Sqoop:

- wget http://download.microsoft.com/download/0/2/A/02AAE597-3865-456C-AE7F-613F99F850A8/sqljdbc_4.1.5605.100_enu.tar.gzCopier les « driver .jar» dans :
- (supprimez les anciens « drivers » de SQL Server dans le cas échéant) /var/lib/sqoop

« *Driver* » MySQL 5.6 pour Sqoop et Cloudera management:

- wget http://mirrors.ibiblio.org/maven2/mysql/mysql-connector-java/5.1.26/mysql-connector-java-5.1.26.jar
- Copier le « driver » dans : (supprimez les anciens « drivers » de MySQL dans le cas échéant)

  /usr/share/cmf/lib

  /opt/cloudera/parcerls/CDH/lib/hive/lib

  /opt/cloudera/parcerls/CDH/lib/oozie/libext

  /var/lib/sqoop

Corriger le problème de mémoire limitée sur Impala daemon:

```
mem_limit = -1
```

Importation des tables de SQL Server vers Hive en utilisant Sqoop:

- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **DimReseller** --hive-import --hive-table DimReseller --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **DimResellerMedium** --hive-import --hive-table DimResellerMedium --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **DimResellerBig** --hive-import --hive-table DimResellerBig --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **FactResellerSales** --hive-import --hive-table FactResellerSales --create-hive-table --hive-

overwrite --hive-database inf796 -m 1

- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **FactResellerSalesMedium** --hive-import --hive-table FactResellerSalesMedium --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **FactResellerSalesBig** --hive-import --hive-table FactResellerSalesBig --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **DimPromotion** --hive-import --hive-table DimPromotion --create-hive-table --hive-overwrite --hive-database inf796 -m 1
- sqoop import --connect "jdbc:sqlserver://INF796-PC;databaseName=AdventureWorksDW2012;user=inf796;password= inf796" --table **DimDate** --hive-import --hive-table DimDate --create-hive-table --hive-overwrite --hive-database inf796 -m 1

Importation des tables Hive vers Impala:

```
INVALIDATE METADATA
```

Stockage sur Impala des tables en type colonnes « *Parquet* » :

- create table **P_DimReseller** STORED AS PARQUET AS
  SELECT * FROM DimReseller; compute stats P_DimReseller;
- create table **P_DimResellerMedium** STORED AS PARQUET AS
  SELECT * FROM DimResellerMedium; compute stats P_DimResellerMedium;
- create table **P_DimResellerBig** STORED AS PARQUET AS
  SELECT * FROM DimResellerBig;
  compute stats P_DimResellerBig;

- create table **P_FactResellerSales** STORED AS PARQUET AS SELECT * FROM FactResellerSales; compute stats P_FactResellerSales;
- create table **P_FactResellerSalesMedium** STORED AS PARQUET AS SELECT * FROM FactResellerSalesMedium; compute stats P_FactResellerSalesMedium;
- create table **P_FactResellerSalesBig** STORED AS PARQUET AS SELECT * FROM FactResellerSalesBig; compute stats P_FactResellerSalesBig;
- create table **P_DimPromotion** STORED AS PARQUET AS SELECT * FROM DimPromotion; compute stats P_DimPromotion;
- create table **P_DimDate** STORED AS PARQUET AS SELECT * FROM DimDate; compute stats P_DimDate;

Exportation des tables de « *Hive* »vers MySQL en utilisant Sqoop: (*La table doit être créée d'abord sur MySQL*)

- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **DimReseller** --export-dir /user/hive/warehouse/inf796.db/dimreseller -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **DimResellerMedium** --export-dir /user/hive/warehouse/inf796.db/dimresellermedium -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **DimResellerBig** --export-dir /user/hive/warehouse/inf796.db/dimresellerBig -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **FactReseller** --export-dir /user/hive/warehouse/inf796.db/factreseller -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **FactResellerMedium** --export-dir

/user/hive/warehouse/inf796.db/factresellermedium -m 1

- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **FactResellerBig** --export-dir /user/hive/warehouse/inf796.db/factresellerBig -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **DimPromotion** --export-dir /user/hive/warehouse/inf796.db/dimpromotion -m 1
- sqoop export --connect "jdbc:mysql://inf796.essai:3306/inf796" --username inf796 --password inf796 --direct --table **DimDate** --export-dir /user/hive/warehouse/inf796.db/dimdate -m 1

Création des index sur MySQL:

- CREATE INDEX idx_resellerkey ON DimReseller[Medium/Big] (ResellerKey);
- CREATE INDEX idx_businesstype ON DimReseller[Medium/Big] (BusinessType);
- CREATE INDEX idx_resellerkey ON FactResellerSales[Medium/Big] (ResellerKey);
- CREATE INDEX idx_datekey ON DimDate (DateKey);
- CREATE INDEX idx_monthnumberofyear ON DimDate (MonthNumberOfYear);
- CREATE INDEX idx_datekey ON FactResellerSales[Medium/Big] (ShipDateKey);
- CREATE INDEX idx_promotionkey ON DimPromotion (PromotionKey);
- CREATE INDEX idx_frenchpromotionname ON DimPromotion (FrenchPromotionName(35));
- CREATE INDEX idx_promotionkey ON FactResellerSales[Medium/Big] (PromotionKey);

Conversion d'une table InnoDB vers MyISAM sur MySQL:

```
ALTER TABLE MyTable ENGINE = MyISAM;
```

Exécution d'une requête SQL sans le cache sur MySQL:

```
SELECT SQL_NO_CACHE * FROM MyTable;
```

Mise à jour des statistiques des tables « *MyISAM* »sur MySQL:

- ANALYZE TABLE DimReseller;
- ANALYZE TABLE DimResellerMedium;
- ANALYZE TABLE DimResellerBig;
- ANALYZE TABLE FactResellerSales;
- ANALYZE TABLE FactResellerSalesMedium;
- ANALYZE TABLE FactResellerSalesBig;
- ANALYZE TABLE DimDate;
- ANALYZE TABLE DimPromotion;

Suppression du cache sur SQL Server:

- DBCC FREEPROCCACHE WITH NO_INFOMSGS;
- DBCC DROPCLEANBUFFERS;

Mise à jour des statistiques des tables sur SQL Server :

```
GO
UPDATE STATISTICS DimReseller
UPDATE STATISTICS DimResellerMedium
UPDATE STATISTICS DimResellerBig
UPDATE STATISTICS FactResellerSales
UPDATE STATISTICS FactResellerSalesMedium
UPDATE STATISTICS FactResellerSalesBig
UPDATE STATISTICS DimDate
UPDATE STATISTICS DimPromotion
GO
```

Usage de « *Hive* »par la ligne de commande avec utilisateur hdfs:

```
hive
```

Usage d'« *Impala* » par la ligne de commande: (méta-données Hive)

```
impala-shell
```

Usage de « *Spark SQL* » par la ligne de commande: (méta-données Hive)

Créer un lien symbolique pour la configuration de Hive :

```
cd /etc/spark/conf
ln -s /etc/hive/conf/hive-site.xml hive-site.xml
```

Lancer le client en mode local :

```
spark-shell --master local --executor-cores 2 --driver-memory Xg
```

X : la taille de mémoire du serveur en gigaoctet

Exécuter le SQL :

```
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc);
sqlContext.sql(″Show databases″).collect().foreach(println);
```

Usage de MySQL par la ligne de commande:

```
mysql –h MyHOST –u MyUSER –pMyPASSWORD
```

Ajout d'index sur SQL Server pour les tables DimDate et DimPromotion:

- CREATE NONCLUSTERED INDEX [IX_DimDate_MonthNumberOfYear] ON [dbo].[DimDate]([MonthNumberOfYear] ASC);
- CREATE NONCLUSTERED INDEX [IX_DimPromotion_FrenchPromotionName] ON [dbo].[DimPromotion]([FrenchPromotionName] ASC);

Annexe VI

Configuration matérielle et système

Configuration matérielle commune
a. Serveur Workstation: DELL PRECISION 490 [10]. b. 2X Intel® 5160 Dual Core Xeon™ CPU @ 3.00 GHz c. Mémoires 32 GB ECC d. 2X disque dur de 500 GB @ 7200 RPM
Virtualisation
a. VMware Workstation 11.1.0 [33]. b. Chaque machine virtuelle « *VM* »aura : **1.** 1X Intel® 5160 Dual Core Xeon™ CPU @ 3.00 GHz **2.** Mémoires 8/24 GB ECC **3.** 1X disque dur de 500 GB @ 7200 RPM
Configuration système pour la VM-MS SQL Server
c. Windows 7 Pro 64 bits avec service pack 1 d. Microsoft® SQL Server® 2012 version « *Business Intelligence* »
Configuration système pour la VM-MySQL
a. Système d'exploitation : Ubuntu 12.04.5, version 64 bits b. MySQL, Version communauté 5.6.24
Configuration système pour la VM-Hadoop
a. Système d'exploitation : Ubuntu 12.04.5, version 64 bits b. Cloudera 5.4.0 avec Hadoop/Yarn 2.6.0-cdh5.4.0