Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab de Blida

Faculté des sciences

Département d'informatique

Mémoire Présenté par

ALLAMI LATIFA

MALKI AMINA

En vue d'obtenir le diplôme de Master

Domaine : Mathématique et Informatique.

Filière : Informatique.

Option : Ingénierie des logiciels.

Titre :

Mise en œuvre d'un processus ETL dans un environnement Hadoop Pig.

Soutenu le : 29 Juin 2013 , devant le jury composé de :

Mme OUAHRANI Laila,	Présidente
Mlle TOUBALINE Nesrine,	Examinatrice
Mlle AZZOUZ Mahdia,	Examinatrice
Mr BALA Mahfoud,	Promoteur

- promotion 2012/2013 -

Remerciements

Tout d'abord, nous tenons à rendre grâce à DIEU tout puissant pour nous avoir donné le courage et la détermination nécessaire pour finaliser ce travail.

Nous tenons à remercier avec gratitude notre promoteur Mahfoud BALA qui a endossé son rôle de la meilleure façon qui soit. Nous retiendrons sa disponibilité, son aide indéfectible, ses conseils avisés et ses idées riches ainsi que sa sympathie et ses encouragements.

Par ailleurs ; nous rendons un vibrant hommage à l'ensemble du corps professoral du département d'informatique de l'université Saâd DAHLAB de Blida qui ont contribué activement à notre formation pendant notre cursus universitaire.

Nous ne pouvons pas terminer sans remercier nos parents respectifs qui, par leur amour et leur soutien nous ont permis de mener à terme ce travail.

Merci.

Dédicaces

A ceux qui sont la cause de mon existence.

A ceux qui ce sont réjouis de mon sourire et qui vénèrent mon succès

A ceux qui sont toujours là pour moi, et qui continuent à être là

A ceux qui sont la lumière de mes yeux, la lueur de mon chemin et

l'éclat de ma réussite.

A ma très chère maman, qui est toute ma vie, que dieu la garde pour moi,

A mon père que dieu l'accueille dans son vaste paradis.

A mes frères Yacine et Fayçal, mes sœurs Fouzia et Karima, mes beau frères Rédha et Amine, mes belles sœurs Nacéra et Madina, mes nièces Siham et Cérine et mes neveux Ali, Abd el-Djalil et Wassim.

A toute ma famille (Allami et Yaïche achour), mon trésor, qui m'a toujours soutenue.

A tous mes oncles, tantes, cousins et cousines.

A mes amies d'enfance : Ryma, Selma, Khadidja, Sarah Chérifi, Sarah Nekab

qui on partagé tous mes souvenirs.

A mes très chères amis : Nassima, Khadidja, Amina, Amina Malki, , Fatima, Mohammed et Halim, sans eux, mes années passées à l'université auraient été très difficiles.

A tous mes amis qui sont si nombreux, qui m'ont envahi de bonheur.

A mon binôme Amina Malki et sa famille.

A mon promoteur Mr : BALA Mahfoud.

A tous mes professeurs, qui ont contribué à ma formation.

A tous ceux qui aiment faire le bien, et aider les gens.

A tous ceux qui souffrent et espèrent une meilleure vie, que dieu soit avec eux.

A toute personne qui va lire ce Mémoire.

A toutes ces personnes je dédie ce modeste ouvrage.

Latifa

Dédicaces

Je dédie ce modeste travail :

A mes très chers parents en témoignage de ma profonde

gratitude et mon incontestable reconnaissance, pour leurs

sacrifices, la confiance qu'ils m'accordent, leurs soutien

permanent et tout l'amour dont ils m'entourent.

A mes très chers frères et sœur, Abdelhakim, Mohammed, Hassiba, Assiaet khadidja.
à mes adorables nièces Issma ,salssabil et mayssam que dieu les protège
A tous mes oncles, tantes, cousins et cousines.

A mes amis Khadidja, Nassima, Amina, hafida, chrifa, hafsa et Mohammed ainsi que tous mes amis que je ne peux pas,
Malheureusement, les cités tous.
A mon binôme Latifa ainsi que toute sa famille.

A tous mes collègues de ma promotion.

Amina

Résumé

Extract-Transform-Load (ETL) est un processus de préparation de données qui se manifeste périodiquement pour alimenter les entrepôts de données (DW) avec des données provenant de différents systèmes sources. Le traitement d'énormes volumes de données en un temps raisonnable est l'un des défis majeurs de ce processus. Pour cela, nous nous intéressons dans le cadre de ce PFE au processus ETL et particulièrement à ses performances face aux données massives. Nous proposons une solution de traitement parallèle basée sur le paradigme MapReduce. Ce dernier s'impose comme le standard effectif pour les traitements de données à forte intensité connues aujourd'hui sous le nom '*Big Data*'. Vu la complexité introduite par ce type de données, nous avons opté pour la migration du processus ETL dans un environnement de calcul parallèle et distribué sur une infrastructure de type cluster. Pour la mise en œuvre, nous avons retenu l'environnement Apache Hadoop que nous exploitons à travers son outil associé Hadoop Pig.

Mots clés

Systèmes décisionnels, ETL, Entrepôt de données, Cubes de données, MapReduce, performance, Hadoop Pig, Big Data.

Abstract

Extract-Transform-Load (ETL) is the process of data preparation that occurs periodically to feed data warehouses (DW) with data from different source systems. Handling huge volumes of data in a reasonable time is one of the most challenging problems of this process. For this, we are interested in the context of our PFE to ETL process and particularly to its performance in the face of massive data. We propose a parallel processing solution based on the MapReduce paradigm. This later has emerged as the standard for effective processing of intensive data known as 'Big Data'. Given the complexity introduced by this type of data, we chose to migrate the ETL process in an environment of parallel and distributed computing on a cluster infrastructure. For the implementation, we chose the Apache Hadoop environment and its associated tool Hadoop Pig as programming language.

Keywords

Decision support systems, ETL, Data Warehousing, Data Cubes, MapReduce, performance, Hadoop Pig, Big data.

Liste des Acronymes

SI	*Système d'information*
ED	*Entrepôt de Données*
DWH	*DataWarehouse*
ETL	*Extracting Transforming Loading*
OLAP	*On-Line Analytical Processing*
OLTP	*On-Line Transaction Processing*
FASMI	*Fast Analysis of Shared Multidimensional Information*
SGBD	*Système de Gestion de Base de Données*
BD	*Base de Données*
SCD	*Slowly Changing Dimension*
ETLMR	*ETL dans un modèle MapReduce*
UDF	*User Defined Function*
ODOT	*One Dimension One Task*
ODAT	*One Dimension All Tasks*
DFS	*Distributed File System*
HDFS	*Hadoop Distributed File System*
RPC	*Remote Procedure Call*
JVM	*Java Virtual Machine*
CSV	*Comma Separated Values*
MySQL	*My Structured Query Language*
CPU	*Central Processing Unit*
HTTP	*Hyper Text Transfer Protocol*
SSH	*Secure Shell*
GFS	*Google File System*
API	*Application Programming Interface*
BI	*Business Intelligence*

Liste des Figures

Liste des tableaux

Table des matières

Chapitre 02 Processus ETL et volumétrie des données

Chapitre 03 MapReduce, un nouveau paradigme pour le traitement des données intensives et parallélisation des processus

Partie II Conception et Mise en œuvre

Annexes

Introduction Générale

Introduction générale

1. Contexte général

La prise de décision dans une organisation repose sur des informations pertinentes fournissant aux décideurs une vue synthétique et permettant de prendre les décisions les plus appropriées. Afin de faciliter le processus de prise de décision, les entreprises ont recours à des outils de l'informatique décisionnelle, basés sur l'approche des entrepôts de données (EDs) et des technologies OLAP. Un ED représente une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour supporter un processus d'aide à la décision.

La construction et la mise en œuvre d'un entrepôt de données représente une tâche cruciale dans l'entreposage de données, cette tâche passe généralement par trois étapes essentielles. La première consiste à sélectionner et préparer les données sources selon le besoin spécifique de l'utilisateur, la deuxième consiste à modéliser et structurer les données pour les stocker dans l'entrepôt de données et la troisième consiste à établir des outils de restitution et d'analyse.

Dans notre travail, nous nous intéressons au processus ETL par rapport à sa complexité et son importance dans la réussite d'un projet décisionnel. L'ETL est responsable de l'alimentation de l'entrepôt de données en trois étapes qui sont : l'extraction, la transformation et le chargement. Chaque étape possède ses propres fonctionnalités dont la finalité du processus est d'obtenir des données préparées et adaptées au schéma global de l'entrepôt de données. L'extraction permet de capturer les données à partir des systèmes sources, qui servent à la gestion quotidienne de l'entreprise et n'ayant aucune qualité d'analyse. Pour leur donner de la valeur, ces données passent par une deuxième étape qui est la transformation. Celle-ci est la plus complexe dans le processus ETL, elle demande beaucoup de réflexion, surtout au niveau des tâches de filtrage, nettoyage, homogénéisation et agrégation de données pour les convertir au format approprié.

Nous nous plaçons dans le cadre de ce PFE dans le contexte de grands projets décisionnels, caractérisés par des données sources massives (Téra-Bytes voire Péta-Bytes) et une complexité dans les tâches ETL nécessaires à leur traitement.

Pour faire face et garantir des applications performantes malgré la volumétrie des données sources, Google a introduit en 2004 un environnement de calcul parallèle et distribué basé sur le paradigme MapReduce. Ce paradigme exprime un calcul parallèle qui s'articule en deux fonctions de base, Map et Reduce, ayant chacune en entrée une liste de paires clé/valeur et produisant en sortie une autre liste de paires clé/valeur.

MapReduce offre une flexibilité de programmation, une évolutivité rentable, une possibilité de communication interprocessus, une tolérance aux pannes et l'équilibrage de charges. Ce modèle est très populaire et se présente à nos jours comme la solution la plus adéquate pour les traitements des données intensives. D'où l'intérêt de mettre en œuvre un processus ETL dans un environnement MapReduce.

2. Problématique

Comme nous l'avons défini auparavant, l'ETL est chargé de collecter les données de différentes sources, de les transformer et de les nettoyer pour qu'elles soient conformes aux exigences définies par l'entreprise. De plus, ce processus ne s'exécute pas une seule fois mais il est planifié pour alimenter l'entrepôt de manière continue car les sources de données évoluent sans cesse suite aux transactions (*Insert, Update et Delete*). Ceci donne à l'ETL un rôle primordial et décisif dans la réussite de la chaine décisionnelle.

La continuité de l'ETL, sa complexité au niveau des tâches et la volumétrie de données sources constituent une problématique dans le monde décisionnel : comment améliorer les performances de l'ETL ?

3. Objectifs

L'objectif ciblé dans ce projet est de répondre à la problématique présentée ci-dessus en performant le processus ETL. Pour ce faire, nous proposons des méthodes qui

permettent la mise en œuvre de ce processus dans un environnement distribué et parallèle à savoir Hadoop Pig supportant le paradigme MapReduce. Cette solution nous permet de garantir la parallélisation d'exécution des tâches ETL, qui est la clé pour parvenir à une meilleure performance et évolutivité.

4. Organisation du mémoire

Ce mémoire est organisé en deux parties :

La première est composée de trois chapitres qui définissent les concepts de base de l'informatique décisionnelle ainsi que le domaine du traitement parallèle des données avec le modèle MapReduce. Nous présentons dans le chapitre 01 l'architecture et les fondamentaux des systèmes décisionnels. Le chapitre 02 présente les différentes tâches ETL de manière détaillée ainsi que l'impact de la volumétrie des données sur les performances du processus. Le troisième chapitre est consacré à la présentation d'un nouveau modèle de traitement parallèle des données intensives nommé MapReduce ainsi que l'étude de l'architecture du processus ETL dans ce modèle.

La partie II présente la conception de notre système, les différents outils de développement utilisés et l'implémentation du système. Elle est organisée en deux chapitres :
Le chapitre 04 présente la conception de notre système alors que le chapitre 05 présente les différents outils utilisés, l'implémentation du système.

Nous terminons ce manuscrit par une conclusion générale qui synthétisera notre travail et par quelques perspectives.

Partie I

Etat de l'art

Chapitre 01

Systèmes décisionnels, Concepts et Architecture

Avec la généralisation de l'informatique dans tous les secteurs d'activité, les entreprises produisent et manipulent un volume très important de données. Ces données sont stockées dans les systèmes opérationnels de l'entreprise au sein des bases de données, des fichiers, etc. L'exploitation de ces données dans un but d'analyse est difficile. Aussi, cette exploitation est réalisée par les décideurs le plus souvent d'une manière imparfaite en utilisant des moyens classiques (requête SQL, vues, outils graphiques d'interrogation, etc.).

Ces bases opérationnelles utilisent le modèle relationnel. Celui-ci convient bien aux applications gérant l'activité quotidienne de l'entreprise, mais s'avère inadapté au décisionnel. Face à cette inadéquation, les entreprises ont recours à des systèmes d'aide à la décision basés sur l'approche des entrepôts de données.

Dans ce chapitre nous allons présenter les concepts de base des systèmes décisionnels ainsi que leurs architectures. Nous discutons également sur la différence entre les SI décisionnels et SI classiques, nous terminerons ce chapitre par la présentation des concepts de base de la modélisation multidimensionnelle.

1. Systèmes d'information décisionnels

1.1. Définition

Un système décisionnel est un ensemble des moyens, des outils et de techniques qui permettent de **collecter**, **consolider**, **modéliser** et **restituer** les données, matérielles ou immatérielles (logiciel, humain), d'une entreprise en vue d'offrir une **aide à la décision** et de permettre aux décideurs et aux analystes d'entreprise d'avoir une vue d'ensemble de l'activité traitée [De Malleray 2008].

1.2. Architecture décisionnelle

La chaine décisionnelle passe par trois grandes étapes essentielles qui sont l'alimentation, la structuration, et l'analyse (voir figure 1.1). Pour ce faire, des ressources sont utilisées et des outils sont mis en place pour récupérer les données de système opérationnelle de l'entreprise afin de constituer l'entrepôt de données souhaité. Par la suite, l'accès aux données de l'entrepôt peut se faire par des outils de restitution et d'analyse.

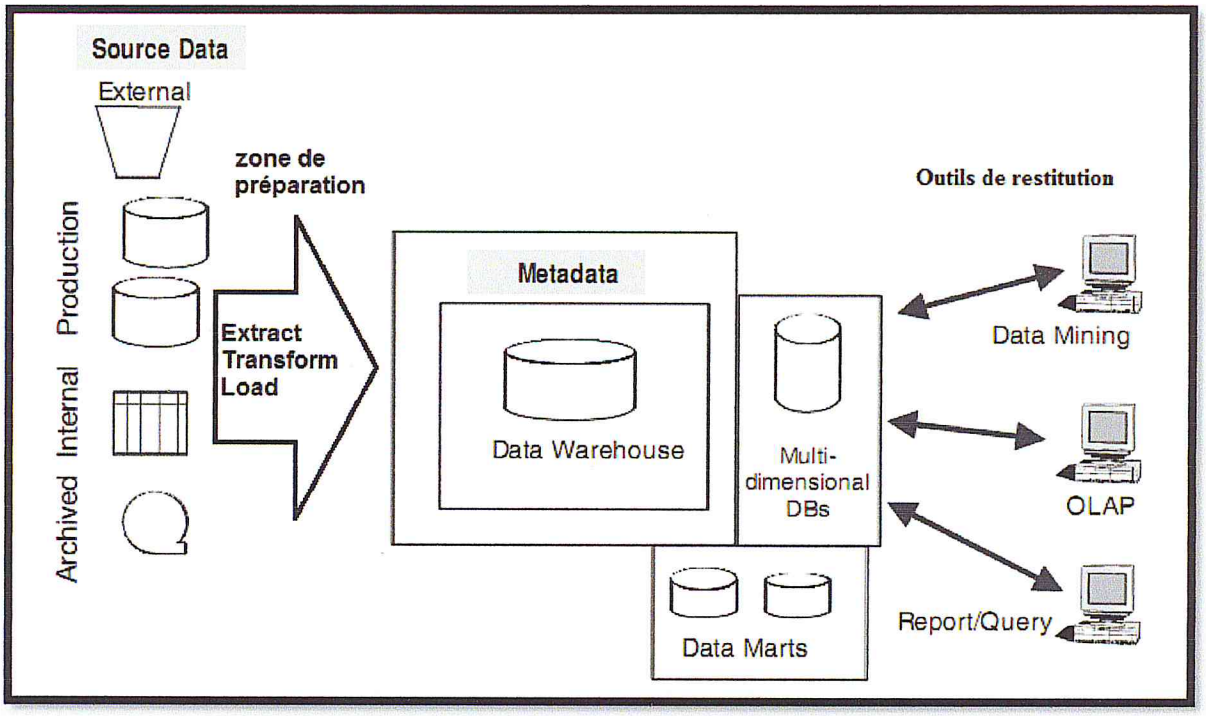


Figure 1.1. Architecture décisionnelle [Ponniah 2010].

Comme le montre la figure1.1. Le système décisionnel comprend plusieurs composants (outils et techniques) utilisés pour manipuler les données. L'architecture décisionnelle met en jeu ces éléments essentiels :

1.2.1. Sources de données

Les données sont nombreuses, variées, distribuées et peuvent se présenter sous différents formats. Il peut s'agir des fichiers "plats" (fichiers CSV, fichiers texte, etc.),ou encore des systèmes de gestion de bases de données (MySQL, PostgreSQL, DB2, ORACLE, etc.).

1.2.2. Zone de préparation

Les données extraites de différentes sources passent par la zone de préparation pour les traiter c.à.d. les nettoyer, les filtrer, les homogénéiser et les agréger. A la fin de cette étape, les données deviennent prêtes au chargement dans l'entrepôt de données. Ces fonctions sont réalisées grâce à une famille d'outils dénommée ETL (Extract-Transform-Load).

1.2.3. Entrepôts de données

Est un lieu de stockage centralisé des informations utiles pour les décideurs. Il met en commun les données provenant des différentes sources et il permet ainsi de conserver leurs évolutions. L'entrepôt de données offre une vue unique des données, ainsi qu'une information plus complète et plus précise qui facilite la prise de décisions [Serna Encinas 2005].

1.2.4. Métadonnées

Ce sont les informations nécessaires pour la gestion des données dans le système décisionnel et ce depuis les sources de données jusqu'à la restitution. Elles constituent une sorte de dictionnaire sur lequel le système s'appuie pour comprendre les données utilisées par les différents outils qui alimentent l'entrepôt de données [De Malleray 2008].

1.2.5. Outils de restitution

Une fois les données stockées et accessibles, elles seront utilisées selon les besoins, à travers des outils d'exploitation de différents types, tels que les tableaux de bord, datamining, reporting, OLAP, etc., qui peuvent répondre aux besoins de toutes les catégories d'utilisateurs.

1.3. Système d'information décisionnel vs Système d'information classique

Les systèmes d'information classiques (OLTP) sont dédiés pour la gestion des applications quotidiennes de l'entreprise, ce qui implique des activités constantes de mise à jour et d'interrogation fréquentes des bases de données par de nombreux utilisateurs. Les systèmes d'information décisionnels, quant à eux, n'ont aucun besoin de modification ou de suppression des données. En effet, le but principal de ces deniers est d'interroger le système d'information pour des fins d'analyse et d'aide à la décision.

Le tableau 1.1 [Benitez et al. 2001], [Teste 2000] compare les caractéristiques des systèmes transactionnels et décisionnels vis-à-vis des données et des utilisateurs.

Caractéristiques	SI classiques	SI décisionnels
Données	- Exhaustives - Orientées applications - Courtes - Dynamiques - La taille est en gigaoctet	- Résumés - Orientées sujets (d'analyse) - Historiques - Statiques - La taille est en téraoctet
Utilisateurs	- Varié (employés, directeurs,..) - Nombreux - Concurrents - Mises à jour et interrogations - Requêtes prédéfinies - Réponses immédiates - Accès à peu d'information	- Uniquement les décideurs - Peu nombreux - Non concurrents - Interrogations - Requête imprévisibles et complexe - Réponses moins rapides - Accès à beaucoup d'information

Tableau 1.1 Comparaison entre les SI classiques et les SI décisionnels.

2. Entrepôts de données

2.1. Définition

Un entrepôt de données (Datawarehouse) est une structure informatique qui donne une vision centralisée et universelle de toutes les informations d'une entreprise et qui a pour but de collecter les données de l'entreprise pour des fins d'analyse et d'aide à la décision [De Malleray 2008].

Le créateur du concept de Datawarehouse, *Bill Inmon*, le définit comme suit :

> *« Un ED est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour la prise de décision. »* [Inmon 2005]

Selon Ralph Kimball :

> *« Un entrepôt de données est une copie des données de transaction spécifiquement structurées pour l'interrogation et l'analyse. »* [Kimball 1996]

2.2. Caractéristiques des données entreposées

D'après Bill Inmon, les données d'un entrepôt de données possèdent les quatre caractéristiques suivantes :

2.2.1. Données orientées sujet

Les données des systèmes de production sont organisées par un processus fonctionnel (Application) de l'entreprise, contrairement aux systèmes décisionnels où les données sont structurées par thème ou sujets (production, Vente, Marketing …). Cette

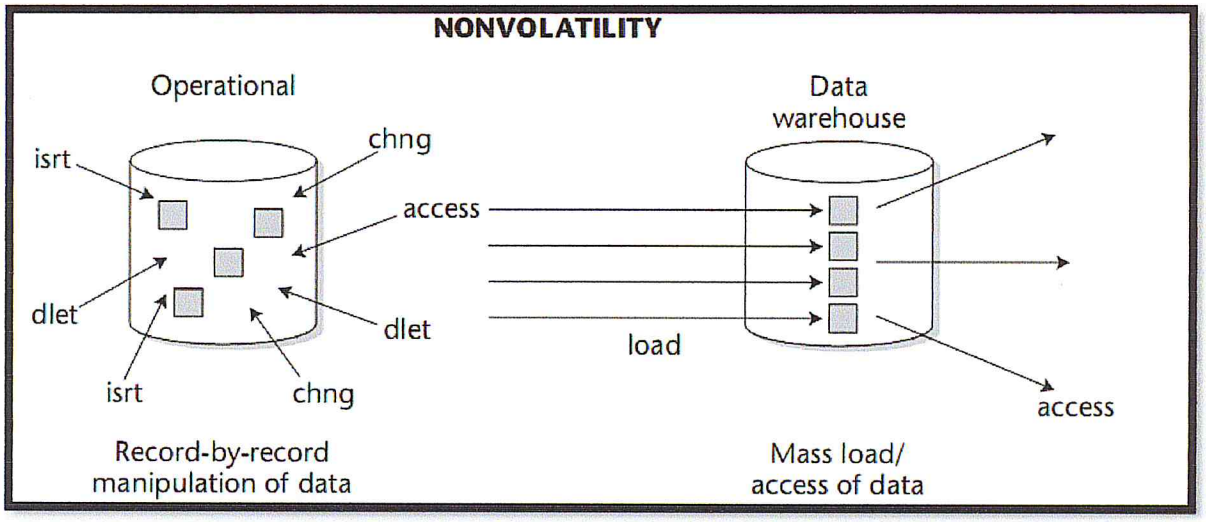


Figure 1.4. Données non volatiles [Inmon 2005].

2.2.4. Données historisées

Alors que dans un système de production les données sont mises à jour à chaque nouvelle transaction, les données d'un entrepôt ne doivent jamais être modifiées. Donc, un référentiel temporel doit être associé aux données afin de pouvoir identifier une valeur particulière dans le temps et permettre l'analyse de leur évolution.

2.2.5. Support d'un processus d'aide à la décision

Les données entreposées peuvent être consultées à travers divers outils (requêtes, outils OLAP, outils de datamining, outils de statistiques ...) permettant leur manipulation et leur analyse.

2.3. Magasin de données (datamarts)

C'est un entrepôt de données dédié à un métier particulier de l'entreprise et ce pour des raisons d'accessibilité, de facilité d'utilisation et de performances. Les données sont représentées d'une façon adaptée aux besoins spécifiques de la fonction désignée.

Selon Bill Inmon [Inmon 2005] :

> *« Un magasin de données est issu d'un flux de données provenant de l'entrepôt de données. Contrairement à ce dernier qui présente le détail des données pour toute l'entreprise, le magasin de données a pour vocation de présenter la donnée de manière spécialisée, agrégée et regroupée fonctionnellement. »*

2.4. Outils d'accès aux entrepôts

2.4.1. Outils de traitement analytique OLAP

OLAP est défini comme étant « le nom donné à l'analyse dynamique requise pour créer, manipuler, animer et synthétiser l'information par des modèles d'analyse de données exégétiques, contemplatifs et selon des formules. Nigel Pendse auteur de OLAP récapitule la définition de l'OLAP en cinq mots ***Fast Analysis of Shared Multidimensional Information*** **(FASMI)** en français « Analyse Rapide d'Information Multidimensionnelle Partagée » [Codd 1993].

Un serveur OLAP permet d'accéder à l'entrepôt de données, il convertit les requêtes des clients en requêtes d'accès à l'ED et fournit des vues multidimensionnelles des données à des outils d'aide à la décision [Khouri et all 2009].

2.4.2. Exploration des données (*Datamining*)

Datamining est un ensemble d'outils de restitutions qui permet l'exploration des vastes quantités de données de l'entrepôt par l'utilisation des techniques statistiques, mathématiques et même de l'intelligence artificielle (IA) [W12].

3. Métadonnées décisionnelles

Dans l'entrepôt de données, les métadonnées regroupent l'ensemble des informations concernées et les processus associés. Elles constituent un véritable support permettant de connaître toutes les informations nécessaires à l'accès, à la compréhension et à l'exploitation des données de cet entrepôt. Les métadonnées du Datawarehouse permettent de répondre aux questions suivantes : Que représente telle donnée ? (sémantique), D'où vient-elle ? Qui l'a créée (origine), Comment est-elle calculée ? (règle de calcul), Quel est son format ? (stockage), Avec quelle fréquence est-elle mise à jour ? (utilisation), Qui est responsable de cette donnée ? (administrateur). Donc, elles jouent un rôle clé pour l'administration des données de l'entrepôt [De Malleray 2008].

4. Modélisation multidimensionnelle

Les modèles de conception des systèmes transactionnels (OLTP) sont adaptés aux applications conçues pour les opérations quotidiennes dans les bases de données. D'autre

part, les systèmes décisionnels (OLAP) sont conçus particulièrement pour l'analyse et l'exploitation des données complexes des bases de données très volumineuses. OLAP est en effet une technique d'analyse, élaborée en 1993 par E. F. Codd, un des créateurs des bases de données relationnelles [Codd 1993]. L'objectif était de pouvoir sélectionner les données selon des critères multiples. L'OLAP permet une analyse et une visualisation des données plus fine.

Pour ce type d'environnement, une nouvelle approche de modélisation a été proposée : La modélisation multidimensionnelle. Popularisée par Ralph KIMBALL dans les années 90, la modélisation multidimensionnelle consiste à considérer un sujet d'analyse comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet et les différentes perspectives de l'analyse [Teste 2000]. Conceptuellement, cette modélisation multidimensionnelle a donné naissance aux concepts de « fait » et de « dimension » [Kimball 1996].

4.1. Concepts fondamentaux de la modélisation conceptuelle

Dans la conception, le modèle multidimensionnel est basé sur les deux concepts suivants :

- **Fait :** Représente un thème ou un sujet d'analyse qui se compose par un ensemble de mesures qui sont des attribues liés relativement au sujet traité. Les mesures d'un fait sont numériques et généralement valorisées de manière continue [Kimball 1996].
- **Dimension :** C'est l'axe d'analyse du fait. Une dimension se compose par un ensemble de paramètres organisés selon une hiérarchie. Une hiérarchie organise les paramètres d'une dimension selon une relation "est_plus_fin" conformément à leur niveau de détail [Teste 2000].

Exemple

Soit le fait (mesure) « total des ventes » à analyser par rapport aux trois dimensions (axes d'analyse) « produit », « localisation » et « temps ». Pour la dimension localisation, nous développons la hiérarchie « *région* → *ville* → *état* ». De manière similaire, pour l'aspect temporel, nous développons la hiérarchie « *année* → *semestre* → *mois* → *jour* ».

Les dimensions peuvent subir des changements de description de leurs paramètres après un certain temps. La mise à jour effectuée sur ces dimensions permet de suive leur évolution dans le temps ; dans cette modélisation nous distinguons deux types.

a) Dimension à évolution lente (Slowly changing dimension)

Dans ce type, une dimension peut subir des changements de ses membres mais de manière très lente telle qu'un changement d'adresse pour un client ou changement de désignation ou d'unité de mesure pour un produit. Pour gérer cette situation trois solutions sont proposées [Kimball & Caserta, 2004] :

- **Ecrasement de l'ancienne valeur (SCD type1) :** Consiste à modifier l'ancienne valeur par une nouvelle. Cette solution offre l'avantage de simplicité mais ne permet pas de garder la traçabilité des valeurs d'attributs.
- **Versionnement (SCD type2) :** C'est la création d'un nouvel enregistrement avec les nouvelles valeurs d'attributs. Cette solution permet de suivre l'évolution des attributs mais elle conduit à l'accroissement de la table de dimensions.
- **Valeur d'origine / valeur courante (SCD type3) :** C'est l'ajout d'un nouvel attribut pour chaque ancienne valeur d'un enregistrement. Cette solution offre une vision ancienne et nouvelle aux attributs. Cependant, elle ne permet pas le suivi de plusieurs valeurs d'attributs intermédiaires.

b) ***Dimension à évolution rapide (Rapid changing dimension)*** : Est une dimension qui subit des changements très fréquents (tous les mois par exemple) des attributs dont on veut préserver l'historique.

4.2. Schéma en étoile (*Star Schema*)

C'est une structure composée d'un fait central qu'est le sujet à analyser entouré des dimensions représentant les axes d'analyse. Ce modèle prend, visuellement, la forme d'une étoile, on parle alors d'un modèle en étoile [Kimball 1996]. Chaque dimension est représentée par une table de dimension et les mesures par une table de fait dont la clé primaire est la concaténation des clés étrangères reliant celle-ci aux tables de dimension.

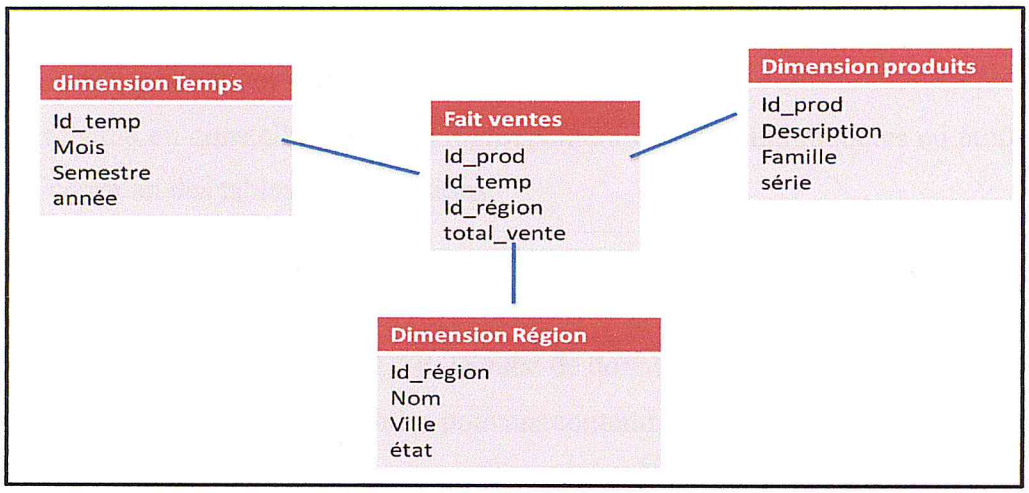


Figure .1.5. Exemple de schéma en étoile (*Star Schema*).

4.3. Schéma en flocon de neige (*Snowflake Schema*)

Un flocon de neige est une étoile où les hiérarchies sont explicites et les dimensions sont normalisées. L'avantage de ce schéma est de formaliser une hiérarchie au sein d'une dimension ce qui facilite l'analyse à différents niveaux de granularité. Aussi, la normalisation des dimensions réduit leur taille alors que la table de fait reste inchangée. Ce type de schéma offre une meilleure visualisation et compréhension des données. En effet, une requête nécessitera plusieurs jointures ce qui augmente son temps de réponse. La figure ci-dessous présente un exemple de schéma en flocons de neige.

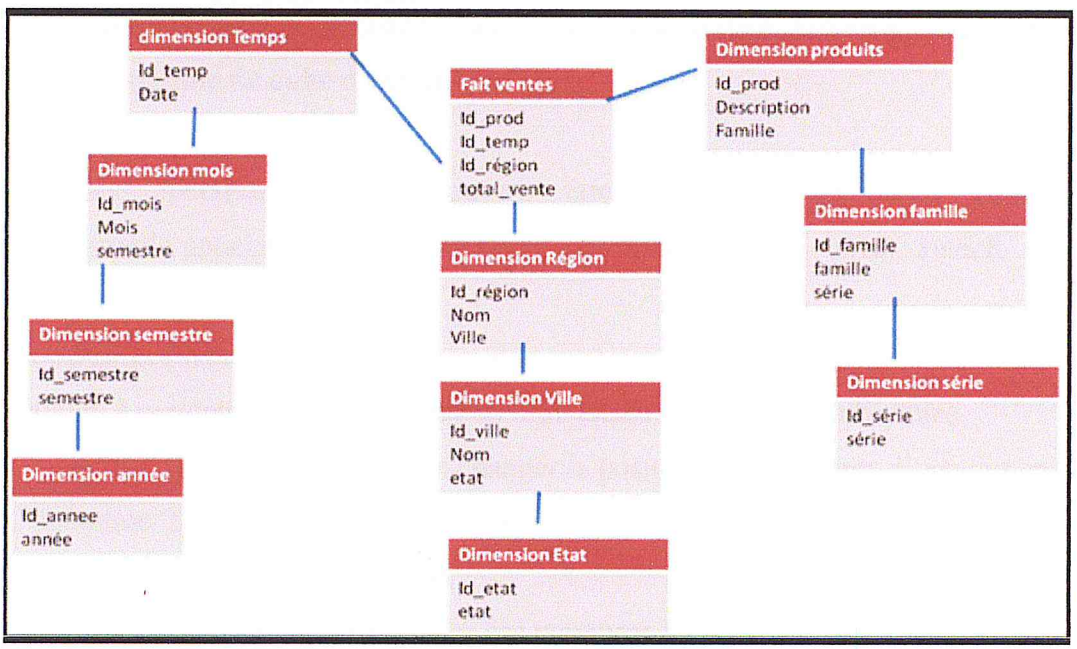


Figure .1.6. Exemple de schéma en flocons (*SnowFlake Schema*).

- L'opération *Push* permet de combiner les membres d'une dimension aux mesures du cube.

D'autres opérations sont liées à la granularité permettant ainsi la hiérarchisation des données. Ces opérations sont *Roll up* et *Drill down*.

- L'opération *Roll up* permet de visualiser les données de manière résumée (en allant d'un niveau fin de la hiérarchie vers un niveau plus général).
- L'opération *Drill down* permet de naviguer vers des données d'un niveau plus fin et donc plus détaillé.

5. Conclusion

Actuellement, les systèmes d'informations décisionnels donnent une souplesse considérable dans l'exploitation des données d'une entreprise grâce aux outils et techniques fournis. Dans ce chapitre, nous avons présenté les systèmes décisionnels, leur architecture et les différentes étapes de leur construction. Cependant, nous remarquons dans l'étape d'alimentation que l'outil ETL responsable d'extraire, de transformer et de charger les données dans l'entrepôt est très important dans la chaine décisionnelle surtout avec l'hétérogénéité et la volumétrie des sources de données.

Le chapitre suivant sera consacré à l'étude détaillée des différentes tâches de l'ETL ainsi que la présentation de l'impact de la volumétrie des données sur la performance de ce dernier.

Chapitre 02

Processus ETL et volumétrie des données

ETL «*Extract, Transform, Load* » est un système d'intégration de données issues de différentes sources hétérogènes d'information de l'entreprise pour l'alimentation de l'entrepôt de données. Ce système ne se contente pas de charger les données, il doit les faire passer par plusieurs moulinettes pour les normaliser, les nettoyer, les conceptualiser et enfin les charger dans l'entrepôt de données. Il est important de savoir que la réalisation de l'ETL constitue, en moyenne, 70% d'un projet décisionnel. Dans ce chapitre, nous allons présenter les différents aspects liés à l'ETL, en particulier les tâches d'extraction (E), de transformation (T) et de chargement (L).

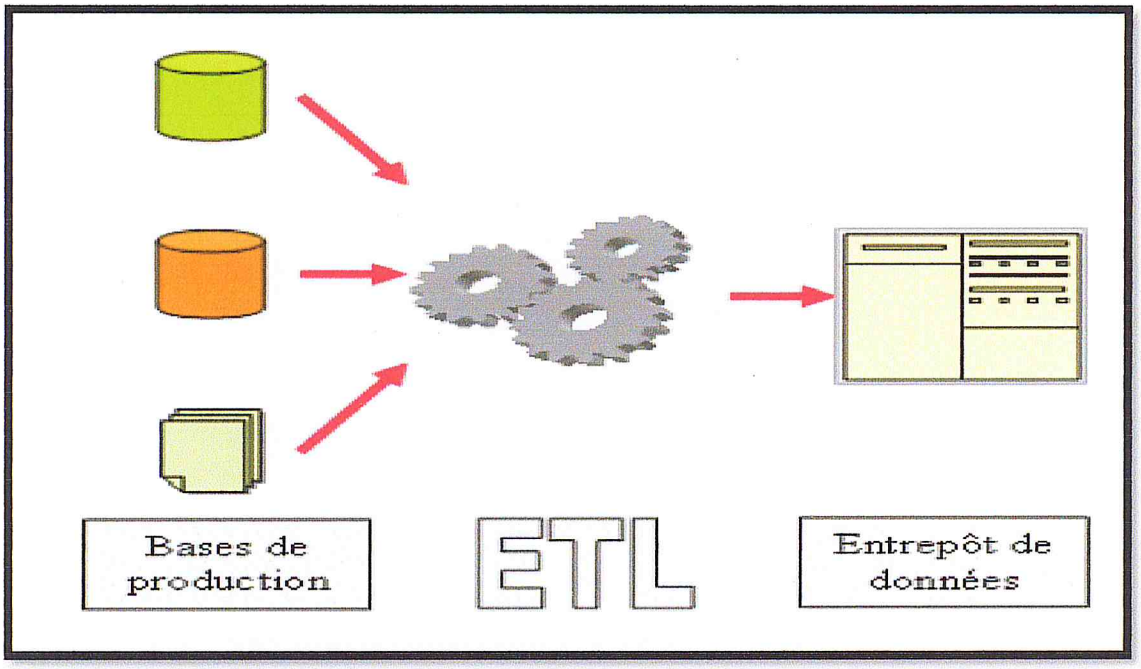


Figure 2.1. Processus ETL

1. Extraction de données

La première phase d'un processus ETL consiste à extraire les données de différents systèmes sources. Face à l'hétérogénéité des sources de données, la caractéristique principale de l'outil ETL est sa capacité de gérer de nombreux fournisseurs de données tout en opérant une sélection de ces dernières, afin de n'extraire que l'information pertinente.

1.1. Sources de données

Il s'agit de toutes les sources, quelque soit leur format, dont le contenu est jugé pertinent pour l'alimentation d'un entrepôt de données. Ces sources sont dans la grande

majorité internes à l'entreprise (capitalisées dans les bases de données opérationnelles), mais peuvent également être externes à l'entreprise (données du marché mondial, audits, documents diffusés sur le web, etc.). Les sources de données peuvent être de plusieurs types :

a) **Structurées :** Les données dans les bases de production structurées respectent un schéma prédéfini comme les bases de données relationnelles et les bases de données objet. L'extraction à partir des sources structurées est relativement simple grâce à l'utilisation des SGBDs.

b) **Semi-structurées :** Les données semi-structurées ne respectent pas un schéma strictement prédéfini comme par exemple les fichiers XML, HTML et les graphes.

c) **Non-structurées :** Les données non-structurées n'ont aucun format prédéfini. Les fichiers texte, les images, le son et les fichiers internet sont des exemples de sources de données non-structurées.

1.2. Identification des sources

Le processus d'identification des sources de données est essentiel à la réussite de l'entreposage de données et les projets du *business intelligence* (BI). Il est important de passer à travers cet effort rapidement et obtenir suffisamment de renseignements sur les sources de données sans s'enliser dans l'excès du détail tout en obtenant l'information nécessaire. Tel qu'à partir des systèmes sources, on doit définir la source de données adéquate pour chaque élément de données dans l'entrepôt.

Etapes pour l'identification des sources :

Nous présentons ici une démarche pour l'identification des sources de données [Ponniah 2010] :

1. Lister les sujets d'analyse (ou fait) dans les tables de faits,
2. Lister les attributs pour toutes les dimensions,
3. Pour chaque élément de donnée cible, trouver la source et l'élément de donnée correspondant dans cette source,
4. Si plusieurs sources sont candidates, choisir la plus pertinente,
5. Si l'élément cible exige des données de plusieurs sources, former des règles d'intégration,

6. Si l'élément source renferme plusieurs éléments cibles (ex: un seul champ pour le nom et l'adresse du client), définir des règles de découpage,
7. Inspecter les sources sur les valeurs creuses.

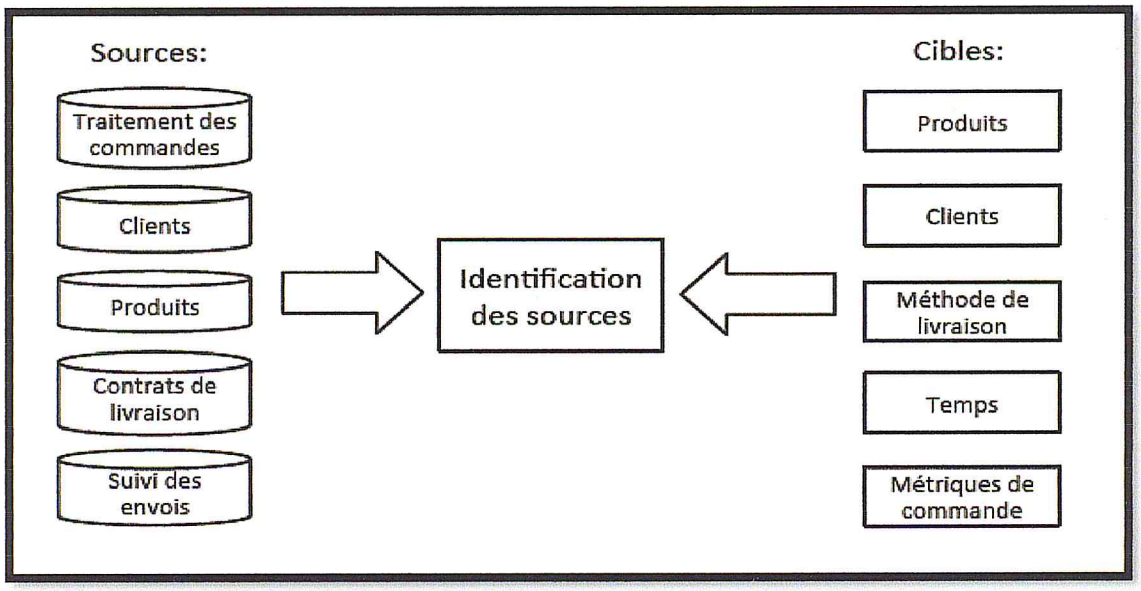


Figure 2.2. Identification de la source : approche par étapes [Desrosiers 2011]

1.3. Techniques d'extraction

Avant d'examiner les différentes techniques d'extraction de données, on doit bien comprendre la nature de la source de données à extraire. En outre, on a besoin d'obtenir un aperçu de la façon avec laquelle les données extraites seront utilisées et stockées.

Les données dans les systèmes sources sont dites temporelles parce que les données sources changent avec le temps. Pour comprendre l'évolution des données en fonction du temps, les systèmes décisionnels stockent l'historique dans l'entrepôt de données. Cela nous amène à la question : comment capturer l'historique à partir des systèmes sources ? La réponse dépend de la manière dont les données sont stockées dans les systèmes sources. Dans ce qui suit, nous examinons comment les données sont stockées dans les systèmes sources.

1.3.1. Stockage de données dans les systèmes sources

a) Valeurs courantes

Correspondant à la grande majorité des cas, la valeur stockée pour un attribut représente la valeur en cours ; les valeurs sont modifiées suite à des transactions d'affaire. Il n'y a aucun moyen pour prédire quand les données seront modifiées.

b) Etat périodique

Cette situation est rarement rencontrée ; ici le système opérationnel conserve tous les changements des valeurs avec la date effective et cela simplifie énormément le travail de capture pour l'entrepôt de données.

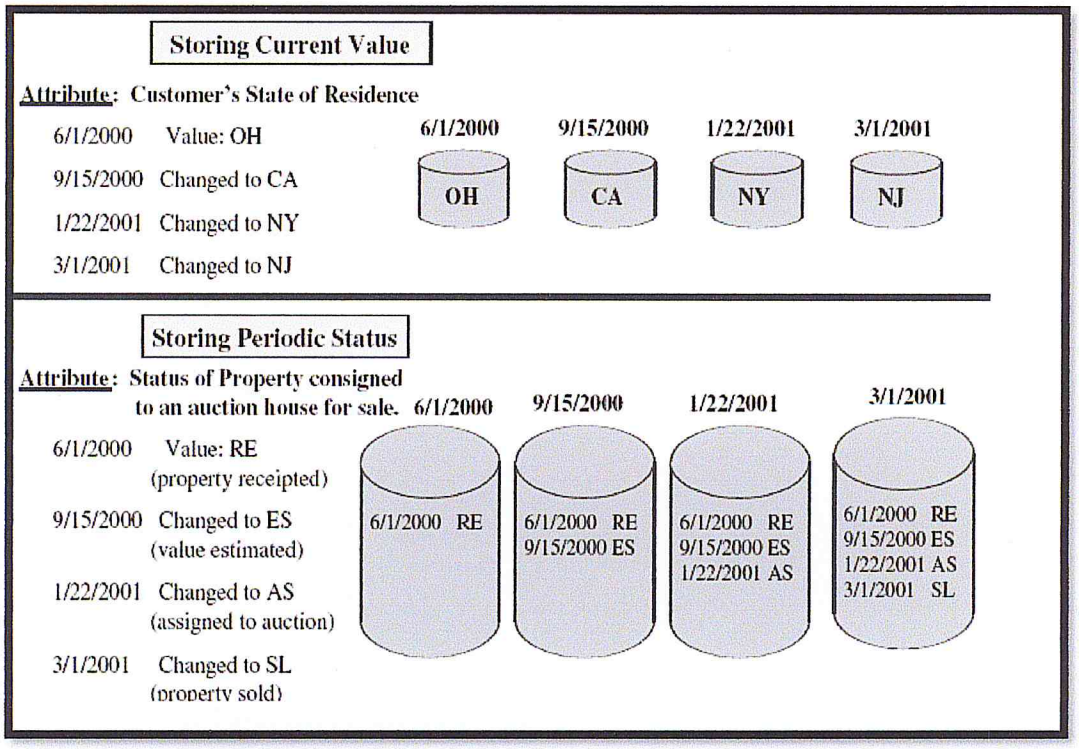


Figure 2.3. Stockage de données dans les systèmes opérationnels [Ponniah 2010]

Après avoir examiné les catégories en indiquant comment les données sont stockées dans les systèmes d'exploitation, nous allons voir les techniques communes pour l'extraction des données. Lors du déploiement de l'entrepôt de données, les données initiales d'une période donnée doivent être déplacées vers l'entrepôt pour une première alimentation, il s'agit du chargement initial. Après le chargement initial, l'entrepôt de données doit être mis à jour afin que l'historique des changements soit reflété dans l'entrepôt de données. Globalement, il existe deux grands types d'extractions de données des systèmes sources :

1.3.2. Extraction statique

On prend une image de toutes les données à un moment précis. Cette prise inclut toutes les données transitoires identifiées pour l'extraction. On utilise le chargement

statique pour le chargement initial de l'entrepôt de données ou parfois le rafraîchissement complet de l'entrepôt de données.

1.3.3. Extractions incrémentales

Dans cette technique, l'extraction des données est en temps réel. Elle survient lorsque les transactions se produisent à des bases de données sources et aux fichiers sources. L'extraction de données incrémentale peut être immédiate ou déférée.

a) Extractions immédiates

Les modifications sont capturées au fil des transactions et mises de côté pour le traitement dans l'entrepôt plus tard.

Capture à l'aide du journal des transactions

- Utilise les journaux des transactions du SGBD maintenus pour la récupération de défaillances possibles,
- Lit le journal des transactions et sélectionne toutes celles qui sont validées,
- Copie dans la zone de préparation des données des transactions confirmées dans le journal,
- Doit être fait avant le rafraîchissement périodique du journal.

Capture à l'aide de triggers

- Un trigger est une procédure stockée dans la BD déclenchée en réponse à un événement prédéfini comme par exemple un ajout ou une mise à jour d'une ligne dans une table,
- Des triggers sont définis pour recopier les données à extraire dans un fichier de sortie ;

Capture à l'aide des applications sources

- Les applications sources sont modifiées pour écrire chaque ajout et modification de données dans un fichier d'extraction, en plus de la BD ;

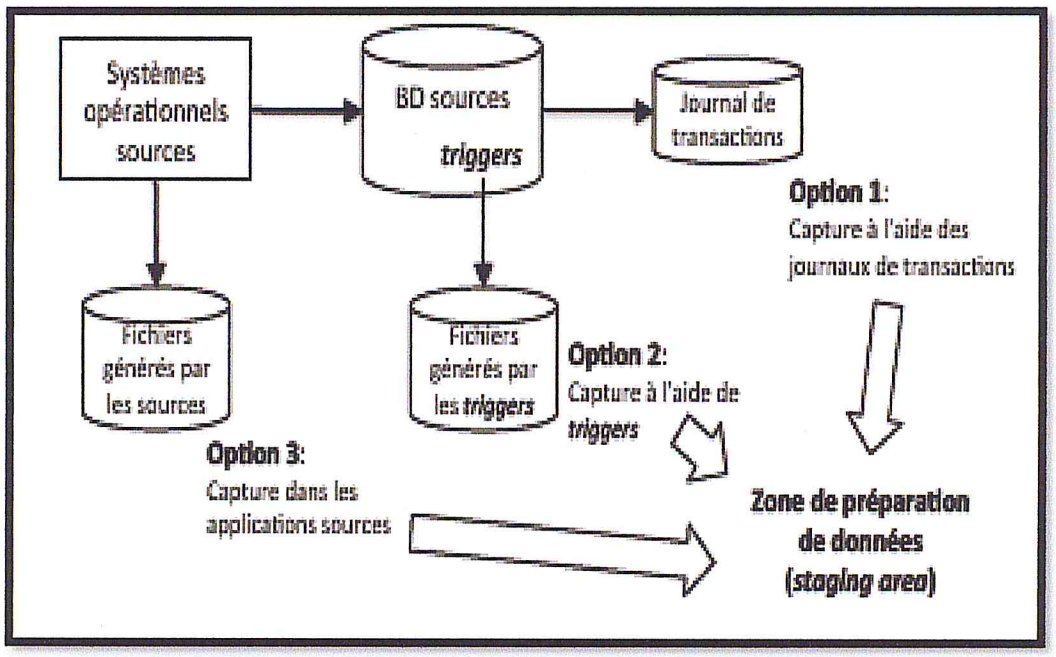


Figure 2.4. Extraction Immédiate de données [Desrosiers 2011].

b) Extractions différées

Un travail spécifique et planifié doit découvrir les modifications effectuées sur les données. La figure 2.5 présente deux options d'extraction immédiate de données qui se résument comme suit :

- **Capture basée sur l'horodatage (*timestamp*) :**

Une estampille (*timestamp*) est ajoutée à chaque transaction des systèmes sources, spécifiant l'instant et la date où la transaction a eu lieu. L'extraction se fait uniquement sur les données dont le *timestamp* est plus récent que la dernière extraction.

- **Capture basée sur la comparaison de fichiers :**

Compare deux captures instantanées *« snapshots »* successifs des données sources qui serviront à extraire les différences (ajouts, modifications, suppressions) entre les deux snapshots.

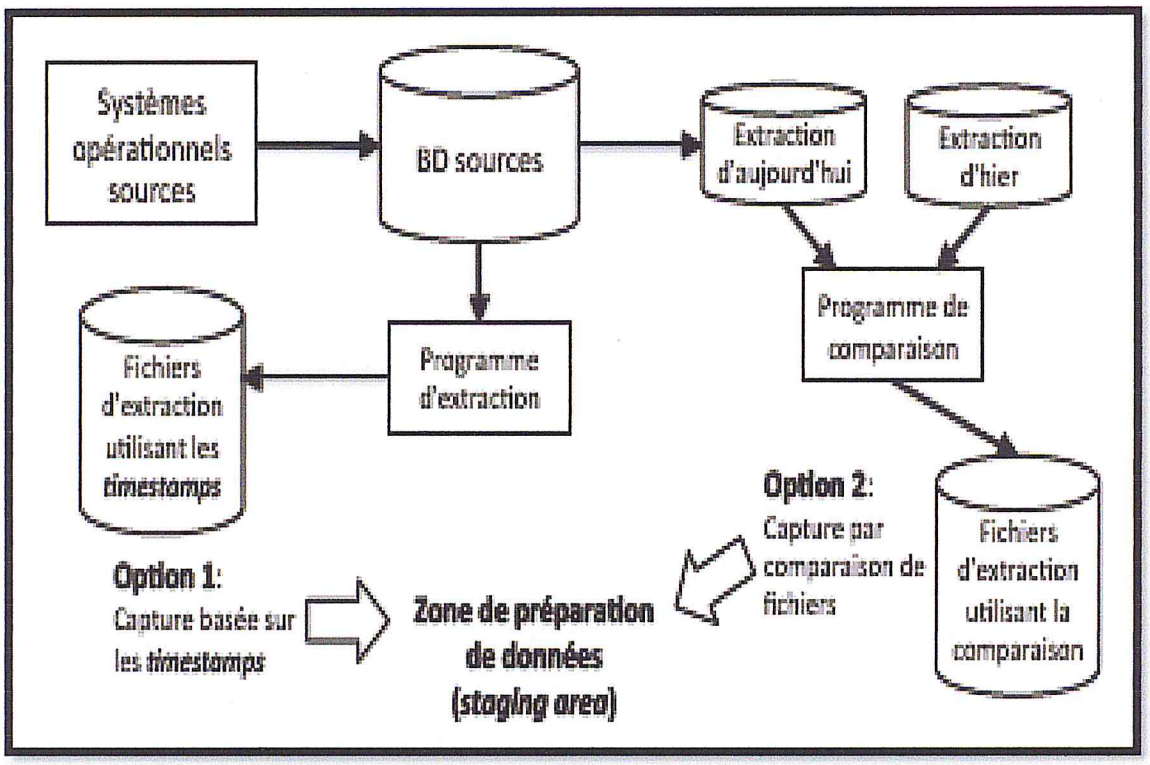


Figure 2.5. Extraction différée de données [Desrosiers 2011].

2. Transformation de données

Les données extraites dans la phase précédente avaient une vocation de gestion et sont par conséquent brutes, dans un format inadéquat et manquent éventuellement de pertinence. Avant de les charger dans l'entrepôt, ces données doivent être transformées et préparées pour qu'elles deviennent des données décisionnelles.

L'étape de transformation d'un processus ETL comporte l'exécution d'une série de règles ou de fonctions sur les données sources pour les convertir vers un format standard. Elle inclut la validation des données et éventuellement leur rejet si celles-ci ne remplissent les conditions minimales requises. La complexité du processus de transformation dépend des données sources. Les bonnes sources de données requièrent peu de transformations tandis que d'autres peuvent nécessiter plusieurs opérations de transformation. Les traitements les plus utilisés pour la transformation sont la conversion, la suppression des doublons, la normalisation, le filtrage, le tri, la fusion et l'agrégation.

2.1. Tâches de base de la phase transformation

Les tâches de base de la phase transformation peuvent se résumer comme suit [Ponniah 2010] :

a) Sélection

Elle s'effectue au début de l'ensemble du processus de transformation de données. La sélection s'applique soit sur des enregistrements entiers ou sur des parties de plusieurs enregistrements à partir des systèmes sources. La tâche de la sélection fait habituellement partie de la fonction d'extraction elle-même.

b) Jointure / Fractionnement

Cette tâche comprend les types de manipulation des données nécessaires pour l'exécution sur les parties sélectionnées de données sources. Parfois, il faut fractionner les éléments sélectionnés lors de la transformation des données, et parfois il faut faire la jointure d'éléments sélectionnés à partir de nombreux systèmes sources.

c) Conversion

Elle comprend une grande variété de conversions élémentaires de champs uniques pour deux raisons primaires. La première des raisons est de normaliser les extractions de données des systèmes sources hétérogènes, et l'autre pour maitre les champs utilisables et compréhensible aux utilisateurs.

d) Résume

L'entrepôt de données a une vocation de synthèse de l'activité de l'entreprise. Nous avons rarement besoin de données d'un niveau de granularité fin. C'est pour cela que dans la phase transformation, il est prévu des fonctions d'agrégation qui transforme les données à un niveau de granularité plus élevé.

e) Enrichissement

Cette tâche vise à simplifier et réarranger les champs individuels pour les rendre plus utiles pour l'environnement d'entrepôt de données. On peut utiliser un ou plusieurs champs à partir du même enregistrement d'entrée pour créer une meilleure vue des données pour le Datawarehouse. Ce principe est étendu lorsqu'un ou plusieurs champs proviennent de plusieurs enregistrements résultant dans un champ unique pour l'entrepôt de données.

2.2. Types de transformations majeurs

Les types de transformation les plus courants sont les suivants [Ponniah 2010] :

a) **Révision de format :** C'est par exemple, changer le type ou la longueur de champs dans le but de standardiser le format des valeurs.

b) **Décodage de champs :** Il s'agit d'unifier, dans le système cible, la codification pour un objet codifié différemment dans les différents systèmes sources. Exemple : ['homme', 'femme'] vs ['M', 'F'] vs [1,2].

c) **Pré-calcul des valeurs dérivées (ou agrégation):** C'est le calcul de nouvelles valeurs (agrégées) à partir des données sources (détaillées). Exemple : *profit* calculé à partir des *ventes* et *coûts*.

d) **Découpage de champs complexes :** Il s'agit de décomposer une donnée source complexe (composée) pour obtenir des composants individuels séparés à représenter dans l'entrepôt de données. Exemple : extraire les valeurs *prénom*, *secondPrénom* et *nomFamille* à partir d'une seule chaîne de caractères *nomComplet*.

e) **Déduplication :** La déduplication consiste à éliminer les doublons identifiés dans une source de données. Exemple : Plusieurs enregistrements représentant un même client. La plupart du temps, les doublons sont le résultat de la création des dossiers supplémentaires par erreur. Par contre, c'est bien de conserver un enregistrement unique pour un client dans l'entrepôt de données et d'établir un lien vers tous les doublons dans les systèmes sources.

f) **Fusion de l'information :** La fusion de l'information désigne la combinaison de plusieurs champs provenant de différentes sources de données pour produire une entité de données unique.

g) **Conversion des unités de mesures** : Exprimer dans une unité de mesure unique dans l'entrepôt de données des données sources exprimées dans des unités différentes.

h) **Conversion de la date :** Représentation en format standard de dates exprimées dans divers formats sources. Exemple : le 11 Octobre 2000 est écrit 10/11/2000 dans le format ISO (USA).

i) **Clés de substitution :** Les clés de substitution (Surrogate key) sont des clés générées de manière automatique (non intelligentes) utilisées afin de se substituer

aux clés provenant des systèmes sources présentant des risques de doublons. Les clés des systèmes sources ont un sens particulier (orientées métier) ce qui conduit à des problèmes lors de l'utilisation de ces clés dans l'entrepôt de données.

3. Chargement de données

Le chargement des données dans l'entrepôt est la dernière étape du processus ETL. Dans cette étape, les données extraites et transformées sont écrites dans des structures multidimensionnelles destinées pour l'analyse en ligne par les utilisateurs finaux.

3.1. Techniques de chargement

Les données préparées dans la phase précédente peuvent être chargées dans l'entrepôt par les quatre différentes techniques suivantes [Ponniah 2010] :

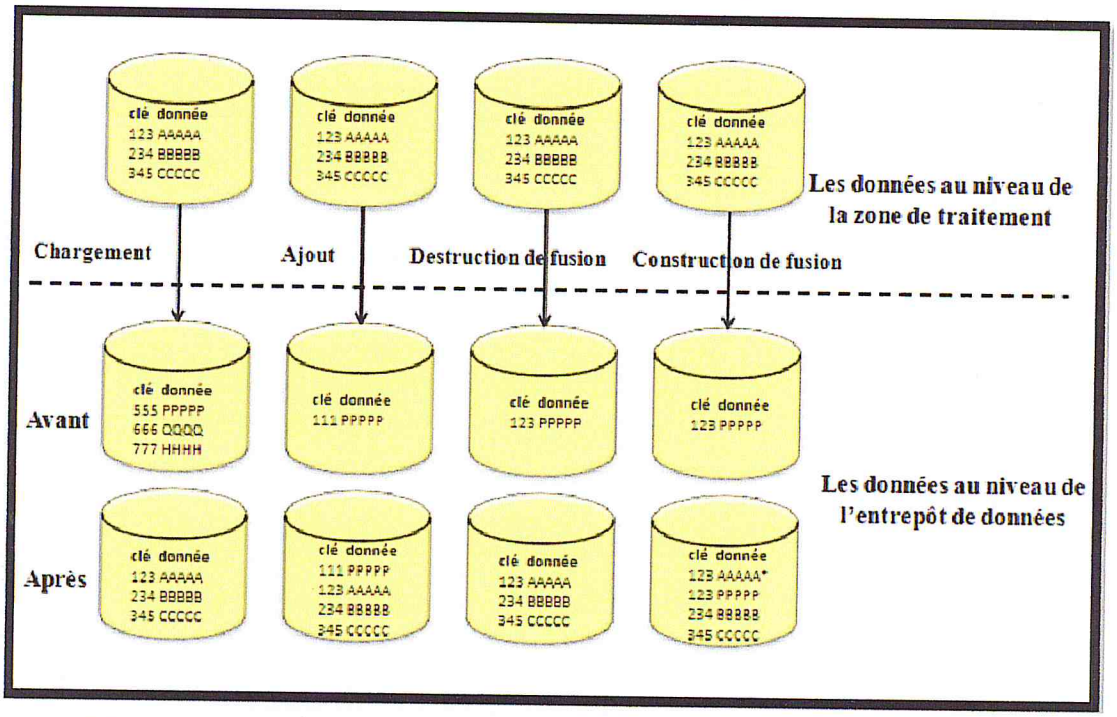


Figure 2.6. Techniques de chargement de données [Desrosiers 2011]

a) Chargement

Si la table cible à charger existe déjà et les données existent dans la table, le processus de chargement efface les données existantes et applique les données à partir du fichier entrant. Si la table cible est vide avant le chargement, le processus de chargement s'applique tout simplement sur les données du fichier entrant.

b) Ajout

Se fait une fois le chargement initial est complété. Si les données existent déjà dans la table, le processus d'ajout ajoute inconditionnellement les données entrantes, tout en préservant les données existantes dans la table cible. Lorsqu'un enregistrement entrant est un doublon d'un enregistrement déjà existant, on peut définir comment gérer la réception d'un doublon :

- L'enregistrement entrant peut être autorisé à être ajouté comme un doublon.
- En d'autre option, l'enregistrement entrant doublon peut être rejetée au cours du processus d'ajout.

c) Fusion destructive

Dans ce mode, les données sources sont transférées vers les données cibles correspondantes, i.e. si la clé primaire d'un enregistrement source correspond à la clé d'un enregistrement cible, ce dernier sera mis à jour. Dans le cas où l'enregistrement source est un nouvel enregistrement (sans correspondance avec un enregistrement existant), celui-ci sera inséré dans la table cible.

d) Fusion constructive

Cette technique est un peu différente par rapport à la fusion destructive. Dans la fusion constructive, si la clé primaire d'un enregistrement source correspond à la clé d'un enregistrement cible déjà existant, celui-ci sera marqué comme substitut de l'ancien enregistrement tout en préservant ce dernier.

3.2. Types de chargement

a) Chargement initial

Il s'agit de la première alimentation de l'entrepôt effectuée juste après la création de son schéma. Différentes situations sont possibles pour le chargement initial :

- Tâche unique : Chargement complet de toutes les tables.
- Plusieurs tâches : Répartition du chargement en plusieurs tâches s'exécutant chacune de manière individuelle et séparée. Dans ce mode, une tâche pourra assurer le chargement de plusieurs tables mais plusieurs tâches peuvent assurer le chargement d'une seule table.

b) Chargement incrémental

C'est le chargement des nouvelles valeurs ou valeurs modifiées en provenance des systèmes opérationnels. Le mode de chargement habituel est la fusion constructive qui

préserve les données historisées. Le mode par fusion destructive peut être appliqué si la préservation des données historiques n'est pas un objectif.

c) Rafraichissement complet

Périodiquement, on remet à neuf la totalité de l'entrepôt ou certaines de ses tables bien ciblées (rafraichissement partiel). Le deuxième cas est plus rare parce que chaque table de dimension est étroitement liée à la table de faits. Le rafraîchissement complet est similaire au chargement initial sauf que dans le premier les données existent dans les tables cibles avant que les données sources ne soient chargées. Les données existantes doivent être effacées avant le chargement. Tout comme dans le cas de chargement initial, les modes chargement et ajout sont applicables au rafraîchissement complet.

3.3. Procédures de chargement

a) Chargement de la table de dimension

Dans un entrepôt de données, les tables de dimension décrivent le contexte dans lequel les mesures sont analysées. La région, produit et temps décrivent par exemple le contexte d'analyse des ventes. La procédure qui sert à maintenir les tables de dimension comprend deux fonctions (1) le chargement initial des tables (2) les changements qui s'opèrent par la suite de manière continue. Deux questions importantes sont à examiner :

- La première concerne les clés des enregistrements dans les systèmes sources et les clés des enregistrements dans l'entrepôt de données. Nous n'utilisons pas éventuellement les clés du système source dans les tables de dimension. On utilise généralement des clés appelées clés de substitution (Surrogate key) générées par le système. Les enregistrements dans les systèmes sources ont leurs propres clés (business key).
- La deuxième question concerne l'application des techniques de chargement vu précédemment pour l'alimentation des dimensions. La figure suivante montre comment ces différents types sont manipulés.

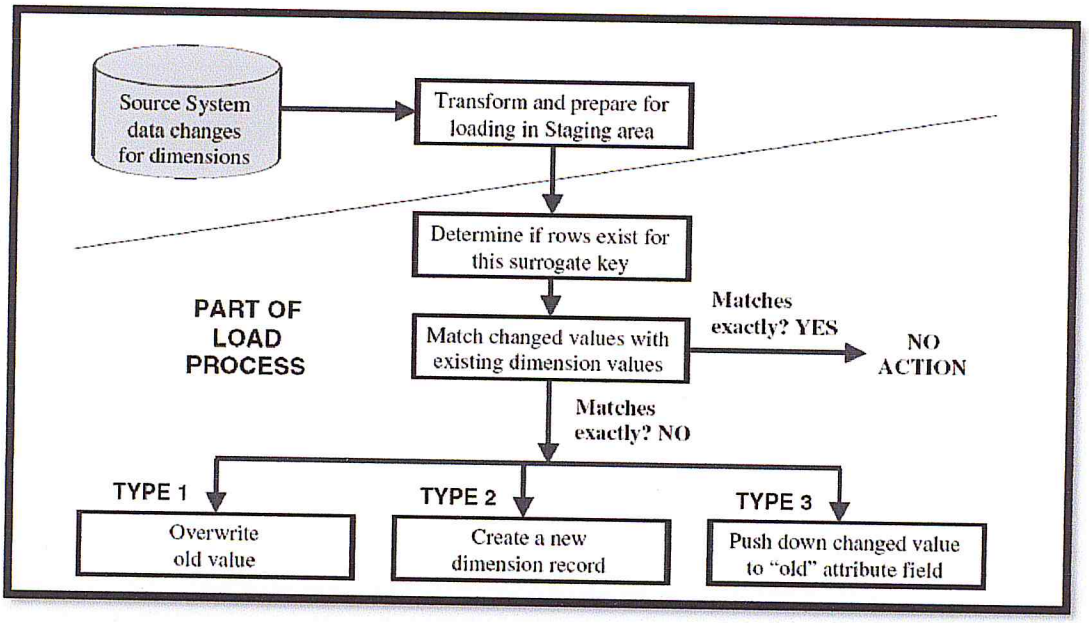


Figure 2.7. Procédure de chargement de dimensions [Ponniah 2010].

b) Chargement de la table de fait

La clé de la table de faits est la concaténation des clés des tables de dimension. Pour cette raison, les enregistrements de dimension sont chargés en premier. Aussi, il faut créer la clé concaténée pour l'enregistrement de la table de fait à partir des clés des enregistrements de dimensions correspondantes avant le chargement de chaque enregistrement de la table de faits.

4. Concept de « Big Data »

L'expression « Big Data » (ou grosse donnée, ou encore données massives) est apparue pour la première fois en 2008. Elle a émergé car la quantité de données traitée ces dernières années est équivalente à la quantité traitée en 10 années dans le passé et augmente de manière explosive. Selon Franck Cohen, président de SAP EMEA (une université partenaire en France), "on crée actuellement le même volume de données en deux heures qu'on en a créé depuis le début de l'humanité jusqu'en 2003". Sans que tous les chiffres avancés soient aussi spectaculaires, les observateurs s'accordent à constater une croissance exponentielle des volumes de données, liée à un besoin de numérisation à tout crin des documents en tous genres : les entreprises capturent désormais quotidiennement des milliards de milliards d'octets dans tous les domaines, depuis des données clients ou fournisseurs jusqu'aux données opérationnelles ou contractuelles, sans oublier les millions

de capteurs disséminés à travers tous les réseaux, dans des unités embarquées dans les véhicules ou les téléphones mobiles, qui eux-mêmes recueillent, transforment, créent et communiquent des données. Pour Jean-Michel Jurbert, responsable du marché pour les solutions BI chez SAP France, "le Big Data n'est pas nouveau : les problèmes de volumétrie existent depuis de nombreuses années. Ce qui est nouveau, c'est de parler de très gros volumes, de type pétaoctets (Po), une évolution accélérée du fait du développement de la digitalisation de tous les documents et du traçage des opérations sur le Net".

5. Conclusion

L'ETL est un outil très important dans un projet décisionnel. Nous remarquons dans ce chapitre que l'ETL est capable de traiter des données de plusieurs sources, de plusieurs formats ainsi que de grands volumes de données. Ces contraintes rendent ce processus énormément lourd et long en termes de temps. Donc, le traitement de ces données en utilisant des outils classiques s'avère difficile. Un modèle de traitement parallèle des données intensives nommé MapReduce est innové pour résoudre ces problèmes et surtout la volumétrie des données. Avec ce modèle nous pouvons fragmenter les tâches ETL en plusieurs sous tâches pour les traiter parallèlement afin d'améliorer la performance. L'étude de ce modèle sera le sujet du chapitre suivant.

Chapitre 03

MapReduce, un nouveau paradigme pour traitement des données intensives et parallélisation des processus

La volumétrie de données est un défi majeur pour les entreprises, surtout celles qui intègrent des contenus sur l'internet. Pour ce type de problème, le meilleur exemple est Google. Ce dernier est le premier organisme qui a confronté ce problème. En prenant les chiffres par l'exemple :
Google a environ 20 milliards de pages web multiplié à la taille moyenne d'une page (20KB) et ça vaut 400 TB, avec une vitesse de lecture disque 30-35 MB/sec implique 4 mois de lecture.

La seule approche possible pour lutter contre ce problème est de **diviser pour régner**, qui est un concept fondamental en informatique, introduit très tôt dans les programmes. Son concept est simple, l'idée de base consiste à partitionner un gros problème en petits sous-problèmes afin de garantir un équilibrage de charge de traitement de données.

Le paradigme MapReduce est basé sur l'approche diviser pour régner, son principe est de diviser les données à traiter en partitions indépendantes, traiter ces partitions en parallèle et finalement refusionner les partitions traitées pour produire le résultat final.

Dans ce chapitre nous allons présenter le modèle MapReduce ainsi leur principe de fonctionnement, nous discuterons également sur leur implémentation par Google et Hadoop et à la fin nous citons le travail de Liu et *al.* sur le processus ETL dans un **Framework** [1]MapReduce.

1. Modèle de programmation MapReduce

Google a introduit MapReduce comme une solution à la nécessité de traiter des données de taille de plusieurs téraoctets sur une base quotidienne. Le but du modèle de programmation MapReduce est de fournir une abstraction qui permet aux utilisateurs d'effectuer des calculs sur de grandes quantités de données.

Grâce à sa conception, le paradigme MapReduce fournit un support pour les aspects suivants [Moise 2011] :

[1] C'est un ensemble de classes qui se chargent de la réalisation de fonctionnalités communes à une catégorie d'application.

1.1. Parallélisation automatique de calculs

La parallélisation des calculs de l'utilisateur, ainsi que leur exécution en parallèle sont gérés automatiquement par ce modèle.

1.2. Grande distribution pour les données

MapReduce est conçu pour gérer efficacement le traitement d'un ensemble de données très volumineuses. Cela implique que la distribution des ensembles de données d'utilisateur est intrinsèquement gérée par ce modèle.

1.3. Matériel commode

Le paradigme est conçu pour fonctionner sur des **clusters**[2] peu coûteuses, et donc ne nécessite pas de matériel spécialisé pour fonctionner.

2. Principe de fonctionnement

Le modèle MapReduce exprime le calcul parallèle dans les deux primitives Map et Reduce, en tenant une liste des entrées de paires clé/valeur et en sortie une liste de paire clé/valeur, tel que :

- **Fonction Map :** lit à partir de la source de données qui est introduit dans la fonction Map en tant que paires clé/valeur et produit un ensemble intermédiaire des paires clé / valeur. Habituellement, la même clé intermédiaire apparaît dans plusieurs paires et associée à des valeurs différentes.
  - Map : (K1, V1) → list(K2, V2)

- **Fonction Reduce :** recueille et agrège toutes les données produites par la première phase. Elle gère également des données des paires de clé / valeur, telles que les valeurs intermédiaires qui ont la même clé sont regroupées et transmises à la phase reduce pour le traitement.
  - Reduce : (K2, list(V2)) → list(V3)

[2] Architecture composé de plusieurs machines (ordinateurs, serveurs, etc.).

3. Frameworks MapReduce

3.1. Framework de Google

De nombreuses implémentations différentes de l'interface MapReduce sont possibles. Le bon choix dépend de l'environnement. Parallèlement à cette abstraction, Google a également proposé un Framework qui permet le traitement distribué de calculs MapReduce.

3.1.1. Séquence d'exécution de MapReduce

Les fonctions Map et Reduce qui seront invoquées sur plusieurs machines, sont instanciées dans les tâches qui sont créées par la bibliothèque MapReduce comme suit : les tâches de Map sont créées une pour une partie d'entrée, tendis que le nombre « R » des tâches de Reduce est spécifié par l'utilisateur. Le mécanisme de distribution des appels de Reduce repose sur une fonction de partitionnement fournie par l'utilisateur. Cette fonction divise l'espace clé intermédiaire dont le même nombre de partitions que le nombre de tâches Reduce prévu par l'utilisateur.

La figure ci-dessus représente le flux global de l'opération MapReduce, tel que : lorsque le programme utilisateur appelle la fonction MapReduce, la séquence d'actions suivante se produit [Dean & Ghemawat, 2004] :

1. La bibliothèque MapReduce d'abord divise les données d'entrée en blocs de taille fixe, la taille de chaque partition est de 16 MB jusqu'à 64 MB (par défaut 64MB). Ensuite, de nombreuses copies du programme commencent sur un cluster de machines.
2. Un des exemplaires du programme est spécial pour le *Master*. Le reste pour les *Workers* qui sont affectés le travail par le Master. Il ya « M » tâches Map et « R » tâches Reduce à attribuer. Le Master choisit les Workers inactifs et attribue à chacun une tâche Map ou une tâche Reduce.
3. Un Worker qui est chargé d'une tâche Map (un Mapper) lit le contenu de la partition d'entrée correspondante. Il analyse les paires clé / valeur à partir des données d'entrée sur laquelle il applique la fonction Map, il produit un ensemble des paires intermédiaires clé / valeur qui sont stockées dans la mémoire.
4. Périodiquement, les données intermédiaires sont stockées sur le disque local du Mapper et elles sont partitionnées en R régions par la fonction de partitionnement.

Les emplacements de ces paires sur le disque local sont passés vers le Master, qui est chargé de transmettre ces endroits pour les Reducers.

5. Un Worker exécutant une tâche Reduce (un Reducer) reçoit a partir du Master des emplacements contenant les partitions qu'on lui a affectées pour le traitement. Le Worker utilise les appels de procédure à distance pour lire les données a partir des disques locaux des Mappers. Quand un Reducer a lu toutes les données intermédiaires, il trie par les clés intermédiaires de telle sorte que toutes les occurrences de la même clé sont regroupées. Le tri est nécessaire, car généralement plusieurs clés correspondre à la même tâche Reduce.
6. Le Reducer parcourt toutes les données intermédiaires triées et pour chaque clé intermédiaire unique rencontrée, il passe la clé et l'ensemble de valeurs intermédiaires qui correspondent à la fonction Reduce de l'utilisateur. Pour éviter la synchronisation, chacun des Workers ajoute les données qu'il produit à son propre fichier de sortie.
7. Lorsque toutes les tâches de Map et les tâches de Reduce ont été accomplies, le Master réveille le programme utilisateur. À ce stade, l'appel MapReduce dans le programme utilisateur retourne en arrière au code utilisateur.

Après avoir réussi, la sortie de l'exécution MapReduce est disponible dans R fichiers de sortie (un par tâche Reduce, avec les noms des fichiers spécifiés par l'utilisateur). En général, les utilisateurs n'ont pas besoin de combiner ces fichiers de sortie dans un seul fichier. Souvent, ces fichiers passent en entrée à un autre appel MapReduce, ou les utiliser depuis une autre application distribuée qui est capable de faire face à une entrée partitionné en plusieurs fichiers.

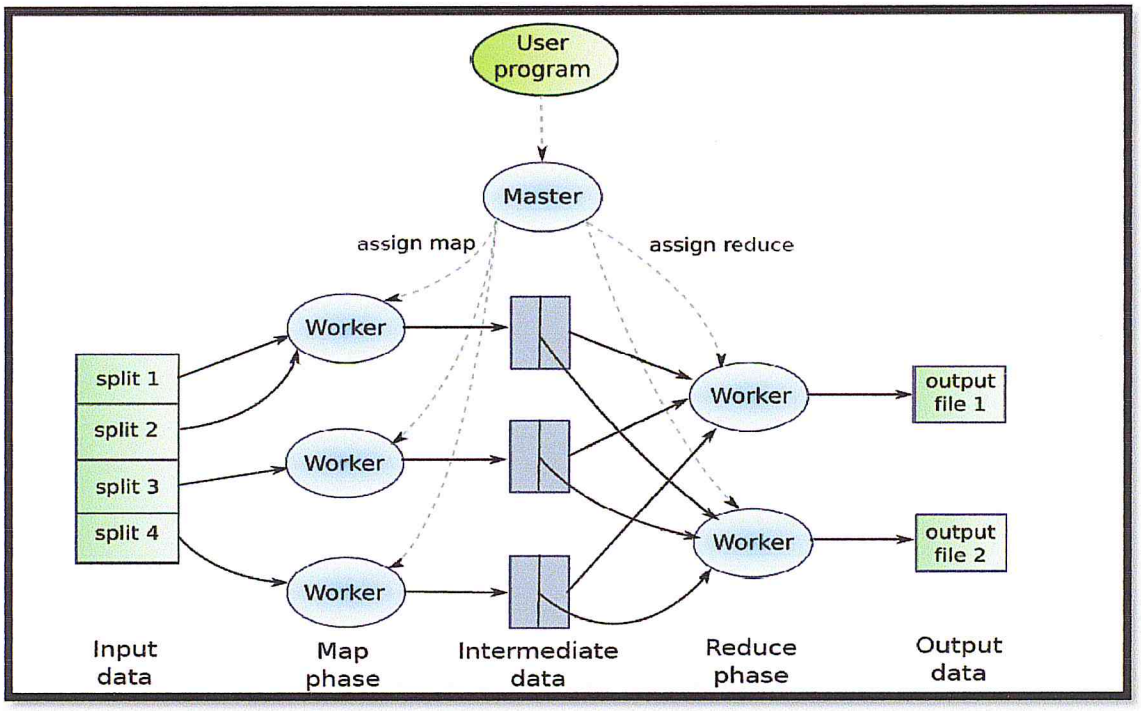


Figure 3.1. Séquence d'exécution de Google MapReduce [Dean & Ghemawat, 2004].

3.1.2. Structure de données du Master

Le Master conserve plusieurs structures de données. Pour chaque tâche Map et tâche Reduce, il mémorise l'état (inactif, en cours ou terminé), et l'identité de la machine Worker (pour les tâches non-inactives).

Le Master est le conduit par lequel l'emplacement des régions de fichiers intermédiaires est propagé à partir des tâches Map aux tâches Reduce. Par conséquent, pour chaque tâche Map terminée, le Master stocke les emplacements et les tailles de R régions de fichier intermédiaire produit par la tâche Map. Les mises à jour de cet emplacement et la taille des données sont reçues tant que les tâches Map sont terminées. L'information est poussée progressivement aux Workers qui ont en cours d'élaboration des tâches Reduce.

3.1.3. Tolérance aux pannes

Etant donné que la librairie MapReduce est destinée à traiter de très grosses quantités de données en utilisant des milliers de machines, ce procédé doit avoir une bonne tolérance aux pannes.

- **Panne de Worker** : Le master surveille tous les Workers périodiquement. S'il n'a pas de réponse d'un Worker au bout d'un temps t, ce dernier est considéré comme mort. La tâche qu'il exécutait est redistribuée à un autre Worker. L'échec d'un Worker nécessite la reprise au début de la tâche. Etant donné que les taches sont rapides à traiter, le temps d'exécution total n'est pas trop dégradé. On assure ainsi l'exécution de toutes les tâches indépendamment du travail des workers.
- **Panne du Master** : Le Master établit périodiquement des points de contrôle de sa structure de données. Si le Master est tué, une nouvelle copie est générée et elle reprend l'exécution au dernier point de contrôle.

3.1.4. Raffinements

Nous avons jusqu'ici présenté une vue simplifiée de MapReduce. Il ya d'autres éléments supplémentaires qui complètent ce modèle [Dean & Ghemawat, 2004] :

3.1.4.1.Fonction de partitionnement

L'utilisateur de MapReduce doit spécifier le nombre R des tâches Reduce. Les données sont partitionnées entre ces tâches en utilisant une fonction de partitionnement basé sur la clé intermédiaire. Une fonction de partitionnement par défaut est fournie qui utilise le hachage, par exemple « hash (key) **mod** R ».

3.1.4.2.Tri

Le tri garantie que dans une partition donnée, les paires intermédiaires clé / valeur sont traitées dans l'ordre croissant de la clé.

3.1.4.3. Fonction de combinaison

Dans certains cas, il ya une répétition significative des clés intermédiaires produites par chaque tâche Map. Cette fonction « Combine » est exécutée sur chaque machine qui exécute une tâche Map. En général, le même code est utilisé pour mettre en œuvre les deux fonctions Combine et Reduce. La seule différence entre ces deux fonctions est comment la bibliothèque MapReduce traite la sortie de la fonction. La sortie d'une fonction Reduce est écrite dans le fichier de sortie final et la sortie d'une fonction Combine est écrite dans un fichier intermédiaire qui sera envoyé à une tâche Reduce.

3.2. Framework de Hadoop de la fondation Apache

3.2.1. Qu'est ce que Hadoop

Hadoop est un Framework Java libre destiné aux applications distribuées et à la gestion intensive des données. Hadoop permet aux applications de travailler avec des milliers de nœuds et des péta-octets de données.

3.2.2. Architecture de Hadoop MapReduce

Le projet Hadoop fournit une implémentation open-source du paradigme MapReduce de Google à travers le framework Hadoop MapReduce. Le framework a été conçu suivant le modèle architectural de Google et il est devenu l'implémentation de référence du MapReduce.

L'architecture de Hadoop MapReduce est conçue de façon maître-esclave, consistant en un seul maître « ***JobTracker*** » et multiples esclaves « ***TaskTrackers*** ».
La figure 2.3 présente les principales entités du système Hadoop MapReduce et le flux des interactions.

Le client Hadoop soumet une tâche au JobTracker pour l'exécution. Le JobTracker divise le travail en un ensemble de tâches qui sont soit des Maps ou Reduces. Les données d'entrée sont également divisées en morceaux de taille fixe qui sont stockés dans le système de fichiers distribué de Hadoop « HDFS ». Chaque tâche Map est associée à un bloc de données. Cette étape consiste à enquérir l'espace de noms de fichiers système, de sorte que le JobTracker doit connaitre l'emplacement de chaque bloc d'entrée. Cette information est requise par l'étape suivante de l'exécution, ce qui est la planification des tâches. Le rôle principal du JobTracker est d'agir en tant que planificateur de tâches, en attribuant le travail aux TaskTrackers. Chaque TaskTracker dispose d'un certain nombre d'emplacements disponibles pour l'exécution des tâches. Chaque tâche Map ou Reduce active prend un emplacement, donc un TaskTracker exécute généralement plusieurs tâches simultanément [Moise 2011].

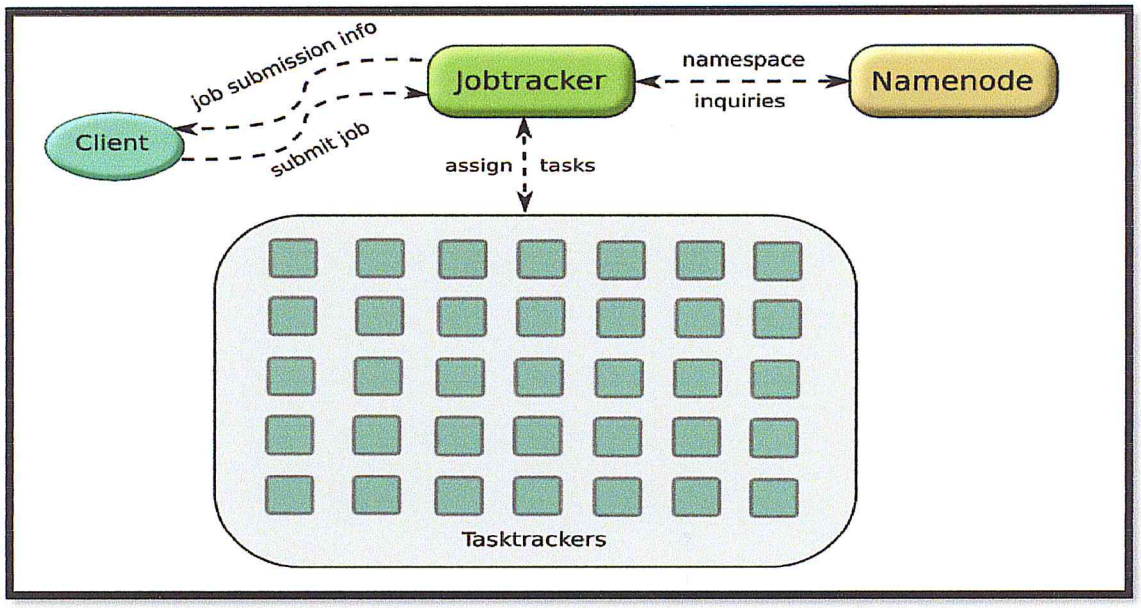


Figure 3.2.Framework Hadoop MapReduce [Moise 2011]

En dehors de la responsabilité de fractionnement de données et d'ordonnancement, le JobTracker est chargé aussi de suivre les tâches et de faire face aux défaillances.

3.2.3. Ordonnancement des tâches

La politique d'ordonnancement implémentée par le JobTracker de Hadoop traite des tâches par la politique FIFO (en français « PEPS » : premier entrée premier sortie).

3.2.4. Tolérance aux pannes

Hadoop a été conçu pour tolérer les défaillances de manière transparente pour l'utilisateur. L'un des avantages les plus encouragés de l'utilisation de Hadoop est sa capacité à traiter automatiquement avec différents types de défaillances causées par différentes sources.

- **Panne de la tâche :** Si un TaskTracker signale un code d'erreur lors de l'exécution d'une tâche donnée, le JobTracker reporte l'exécution de la tâche échouée et transmet la tâche à un autre TaskTracker. En outre, le JobTracker garde une trace du nombre de fois qu'une tâche a échoué. Si ce nombre atteint un seuil configurable, la tâche ne sera pas ré-exécuté
- **Panne du TaskTracker :** Le TaskTracker envoie un message au JobTracker chaque période de temps configurable. Si le JobTracker ne reçoit pas un message à

partir d'un TaskTracker pendant un intervalle du temps donné, ce dernier est considéré mort. Ce TaskTracker est retiré à partir des Workers de JobTracker. Par ailleurs, le JobTracker replanifie les tâches Map ou Reduce qui ont été exécutées sur le TaskTracker échoué.

- **Panne du JobTracker :** Le JobTracker représente le point de défaillance unique dans le système, ce qui rend les défaillances à ce niveau critiques et conduisent à l'échec de Job MapReduce.

4. Travail de Liu et *al.* (ETLMR)

Pour démontrer le fonctionnement de ce modèle, nous allons faire une étude détaillé sur le travail qui à été fait par Lui et al sur l'intégration d'un processus ETL dans le modèle MapReduce.

4.1. Principe

Liu, Thomson et Pederson (2011) ont présenté un prototype appelé ETLMR dans lequel ont été intégrés les concepts décisionnels et l'implémentation de stratégies de découpage et parallélisation des tâches ETL dans un environnement MapReduce.

ETLMR est le premier travail qui répond spécifiquement aux schémas dimensionnels d'ETL sur MapReduce, ils ont fait plusieurs contributions présentées par des nouvelles méthodes ou stratégies qui sont utilisées pour traiter les dimensions d'un schéma en étoile, en flocons, et SCD. De plus ils ont introduit le schéma hors ligne de la dimension (*offline dimension scheme*) qui est très bon échelle que le schéma en ligne (*online dimension scheme*) lors de la manipulation des charges de travaille massives [Liu et *al.* 2011].

4.2. Architecture d'ETL dans MapReduce

La construction de l'architecture d'ETL dans MapReduce passe par trois phases essentielles (Partition de données, Map et Reduce), avant que la première étape commence, les données provenant des sources hétérogènes sont préparées pour construire le fichier CSV (fichier d'entrée) qui est le résultat de la jointure des données Extraites. Dans la première étape, le map-reader lit ce fichier ligne par ligne et applique le rowing sur ces lignes. Les lignes résultantes (nommées *rows*) sont partagées aux Mappers par deux

la synchronisation des dimensions entre les nœuds [3](si aucun système de fichiers distribué (DFS) n'est installé) [Liu et *al.* 2011].

Algorithm 1 ETL process on MapReduce framework

1: Partition the input data sets;
2: Read the configuration parameters (Table 1) and initialize;
3: Read the input data and relay the data to the map function in the map readers;
4: Process dimension data and load it into online/offline dimension stores;
5: Synchronize the dimensions across the clustered computers, if applicable;
6: Prepare fact processing (connect to and cache dimensions);
7: Read the input data for fact processing and perform transformations in mappers;
8: Bulk-load fact data into the DW.

Algorithme 3.1. Traitement d'ETL dans MapReduce [Liu et *al.* 2011]

4.3.1. Partitionnement

ETLMR utilise deux méthodes de partitionnement **Round-robin** et **hachage** [Liu et al. 2011] **:**

4.3.1.1. Round-robin

Cette méthode permet de distribuer les lignes sur les tâches, tel que la ligne numéro **n** est assignée à la tache numéro (**n** mod **nr_map**) ou **nr_map** est le nombre de tâches Map.

4.3.1.2.Hachage (hash by field partitioning)

Cette méthode désigne un ou plusieurs attributs comme des attributs de partitionnement. Les *tuples* avec les mêmes valeurs de hachage sur les attributs de partitionnement sont affectés à la même tâche. S'il y a **nr_map** tâches, le tuple avec une valeur de hachage « **hash** » est affecté à la tâche numéro (**hash** mod **nr_map**).

4.3.2. Configuration et initialisation

En ETLMR, tous les paramètres d'exécution sont stockés dans un fichier de configuration unique, y compris les paramètres de sources de données, les méthodes de partitionnement, les dimensions, les faits, les données à forte intensité de dimensions, et le

[3] Dispositif électronique qui joint à un réseau et qui capable de recevoir et transmettre des informations sur un canal de communication.

nombre de Mappers et Reducers. Ces paramètres offrent aux utilisateurs la flexibilité de configurer les tâches [Liu et *al.* 2011].

L'initialisation est la première étape MapReduce, qui a pour rôle de lire les paramètres du fichier de configuration et de les initialiser pour commencer le job MapReduce.

4.3.3. Préparation des données

ETLMR fournit une implémentation de map-readers basée sur les deux méthodes de partitionnement (round-robin, hachage), les map-readers sont responsables de lire des lignes par itération et les converties en rows. Autrement dit, un dictionnaire de mappage est appliqué aux lignes de sorte d'attribuer un nom pour chaque valeur qui était dans la ligne (nom1 : valeur1, nom2 : valeur2…). Ensuite ces rows seront transmises au Mapper correspondant par l'une des méthodes de partitionnement.

4.3.4. Traitements des dimensions et chargement direct/indirect

4.3.4.1. Traitement de dimension

ETLMR emploie les primitives MapReduce : map, partition, combine, et reduce pour traiter les données. Chaque ligne d'une dimension est traitée par toutes ces primitives [Liu et *al.* 2011]. Par la suite, nous présenterons les différentes approches utilisées dans ETLMR pour traiter les données des dimensions.

4.3.4.1.1. Une dimension une tâche (ODOT)

Dans cette approche, la tâche map traite les données de toutes les dimensions en appliquant des transformations définies par l'utilisateur et en trouvant pour chaque dimension les parties pertinentes de la source de données. Par contre, chaque tâche reduce ne traite que les données d'une seule dimension [Liu et *al.* 2011].

4.3.4.1.2. Une dimension toutes les tâches (ODAT)

Dans cette approche, toutes les tâches reduce traitent les données de toutes les dimensions d'où le nom ODAT (one dimension all task). Cette approche est similaire que ODOT, mais avec des raffinements au niveau de deux places, à savoir la fonction de partition des sorties de map et la fonction reduce .Avec ODAT, les sorties de Mappers sont partitionnées par la méthode round-robin pour que les Reducers reçoivent les mêmes

méthodes de partitionnement (*round-robin*, *hachage*) implémentées par les maps-readers. Le résultat de chaque Mapper est combiné (regroupement par clé) pour optimiser le coût d'envoi des données au Reducer sur le réseau, les outputs combinés vont passer par le Reducer. Dans un Reducer, une row est d'abord traitée par les UDFs pour faire les transformations de données, puis cette row traitée est insérée dans la dimension. Finalement le résultat de ces Reducers sera stocké dans *DWH*

Nous présentons par la figure ci-dessous l'enchainement des étapes ETL dans MapReduce.

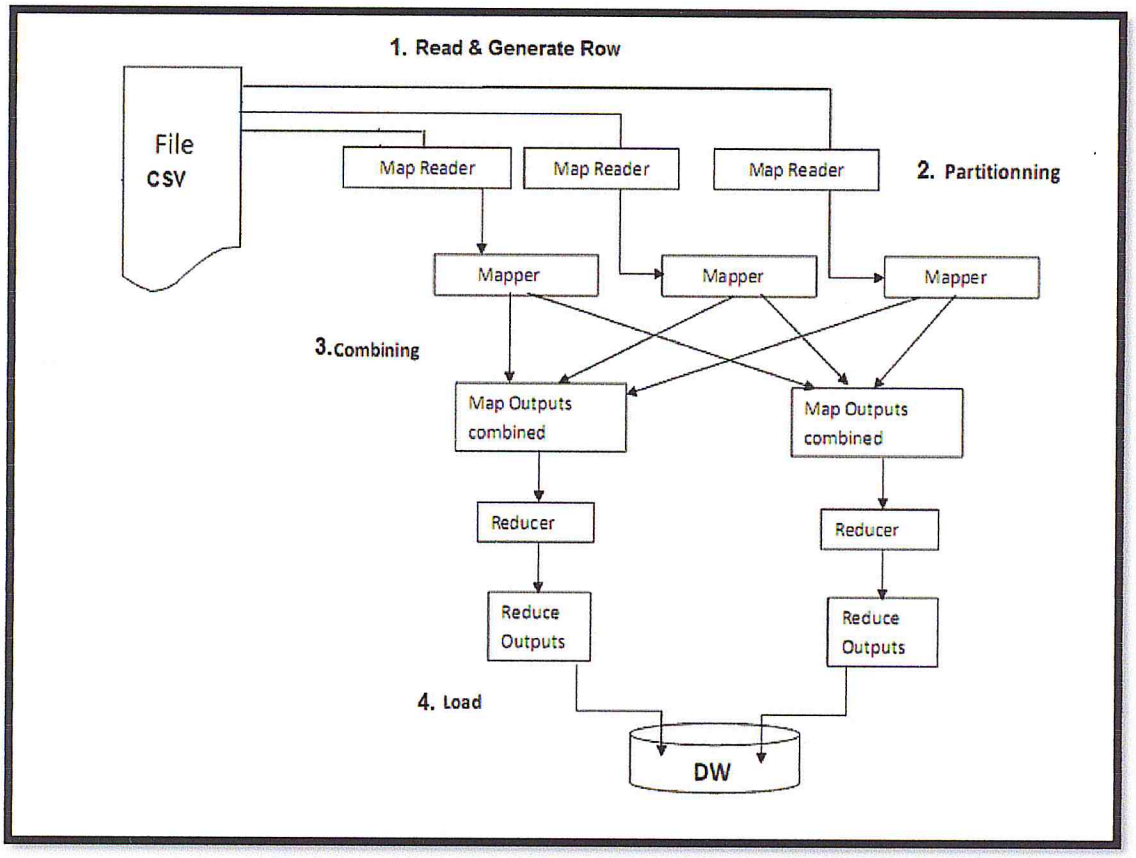


Figure 3.3. Architecture d'ETL dans MapReduce

4.3.Décomposition des tâches ETL dans MapReduce

L'algorithme ci-dessous présent le découpage des tâches ETL et leurs séquences dans MapReduce .Les opérations des lignes 2-4 et 6-7 sont des étapes MapReduce responsables de l'initialisation, invocation des jobs pour le traitement des dimensions et des faits, et le retour de traitement de l'information. Les opérations des Lignes 1 et 5 sont des étapes non MapReduce, utilisées pour la préparation d'ensemble des données entrantes et

nombres de rows (voir la figure 3.4). Dans la fonction reduce, deux problèmes sont examinés afin de traiter les données des dimensions [Liu et al. 2011].

1er problème (unicité) : Comment maintenir l'unicité de la valeur de clé d'une dimension traitée par toutes les tâches ?

2éme problème (concurrence) : Comment garantir le problème de concurrence dans la manipulation des données telle que insertion, mise à jour, suppression dans la table de dimension.

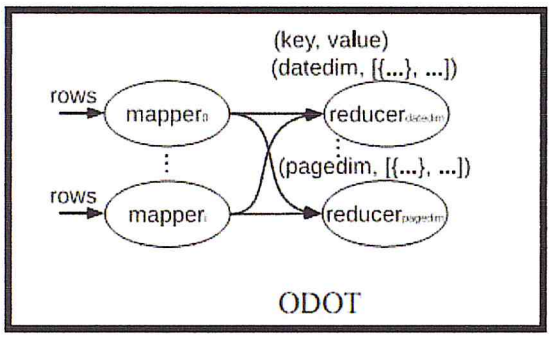


Figure 3.4. ODOT [Liu et *al.* 2011].

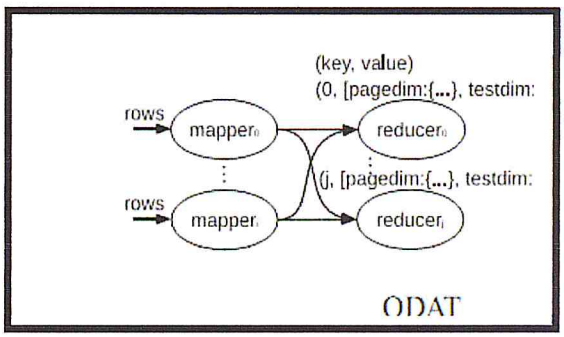


Figure 3.5. ODAT [Liu et *al.* 2011].

4.3.4.1.3. Hybride

Hybride est une combinaison des caractéristiques des approches ODOT et ODAT. Dans cette approche, les dimensions sont divisées en deux groupes, les dimensions des données intensives et les autres dimensions. Les données entrantes de ces dimensions intensives sont partitionnées selon la clé naturelle et sont traitées par toutes les tâches map (comme ODAT), par contre les autres dimensions sont traitées par les Reducers [Liu et al. 2011].

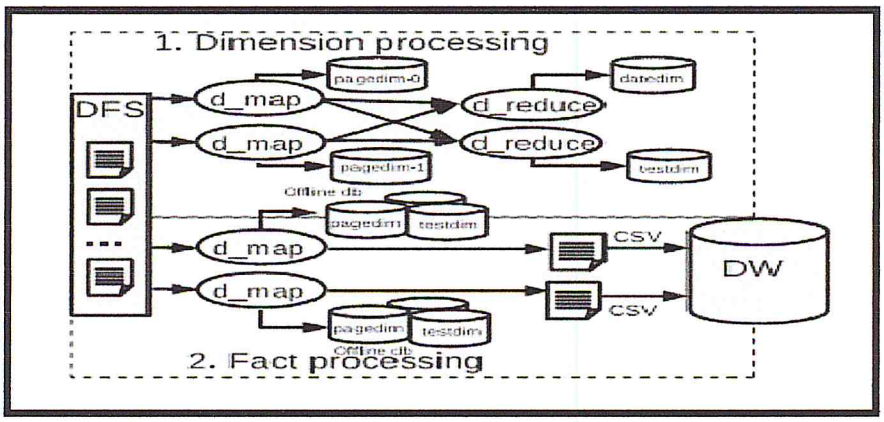


Figure 3.6. Hybride [Liu et *al.* 2011].

4.3.4.2. Chargement direct /indirect

Dans ODAT et ODOT, les tâches maps/reduces interagissent directement avec l'entrepôt de données, mais la performance est affaiblie en termes de coût de communication. Afin d'optimiser, un schéma offline a été proposé dans lequel les tâches maps/reduces n'interagissent pas avec l'entrepôt de données, mais avec des dimensions offline [Liu et *al.* 2011].

4.3.5. Synchronisation de dimensions

Dans le traitement de dimension et à la fin de job MapReduce les sorties de reduce de différents nœuds sont écrites dans le DFS. Ce dernier est le responsable de la synchronisation des dimensions via le network à travers le cluster, ces dimensions seront utilisées dans le traitement de fait [Liu et *al.* 2011].

4.3.6. Préparation et partitionnement

Dans ETLMR, les dimensions et les faits sont traités par des jobs différents. Pour le traitement de fait, le DFS distribue les données sources aux nœuds qui exécutent la tâche map/reduce.

4.3.7. Traitement de fait

Le traitement de fait est la deuxième phase dans ETLMR après le traitement des dimensions. Chaque tâche map/reduce traite un ensemble de données de taille égale, y compris la lecture, la recherche des clés des tables de dimensions. Si un fait est un fait agrégé, les Reducers sont configurés pour des mesures de calcul sur toutes les lignes en utilisant des fonctions d'agrégation telles que somme, moyenne et comptage. Pour optimiser les performances, les Reducers peuvent être omis si aucune agrégation n'est nécessaire [Liu et *al.* 2011].

4.3.8. Chargement de fait par blocs

ETLMR fournit une classe appelée BulkFactTable qui supporte le chargement par blocs de table de fait pour transférer les données traitées de la mémoire principale vers l'entrepôt de données. Ils ont utilisé un UDF appelé bulk loader qui peut être configuré pour satisfaire les différents type de SGBD [Liu et *al.* 2011].

4.4. Limites ETLMR

Durant notre étude de ce prototype, nous avons remarqué que l'équipe d'ETLMR a fait beaucoup d'efforts dans le partitionnement des données, transformation et aussi dans le chargement. Cependant, nous voyons que :

- ETLMR ne permet pas d'assurer la préparation des données, alors que l'extraction des données est faite à partir des fichiers préparés au préalable et non pas directement à partir des sources de données.

- Pour le chargement, ils ont fait un chargement dans un DWH centralisé. Donc cette centralisation de l'entrepôt peut se refléter négativement sur sa performance.

5. Conclusion

Ce paradigme du calcul Massivement parallèle offre une grande souplesse dans le traitement des données surtout dans le domaine de « Big Data ». Cette souplesse est étalée dans la capacité de traitement d'un grand volume de données en un temps record. Aussi, ce modèle peut s'exécuter dans un cluster ou même sur une machine multicore (multiprocesseur).

Dans ce chapitre, nous avons présenté le principe de fonctionnement de ce modèle ainsi leur implémentation dans un framework. Aujourd'hui des nombreuses implémentations open source du principe général de ce modèle sont apparues : Hadoop (Yahoo !puis Fondation Apache), Disco (Nokia), MrJob (Yelp !), etc. À la fin, nous avons illustré le principe de fonctionnement de ce modèle par une démonstration complète de travail de Lui et al 2011 sur l'intégration d'un processus ETL classique dans le modèle MapReduce. Le chapitre suivant sera consacré à notre conception où nous citons le fonctionnement de différentes méthodes implémentées pour la réalisation de notre système.

Partie II

Mise en œuvre

Chapitre 04

Conception de notre Système

La conception est une étape indispensable dans la mise en œuvre d'une application. Dans ce chapitre nous allons procéder à une description détaillée et complète de notre conception, où nous décrirons le principe de fonctionnement de chacune des méthodes proposées.

L'objectif de notre travail est de mettre en œuvre le processus ETL sous l'environnement Hadoop Pig afin de l'exécuter dans la séquence MapReduce, et ceci pour performer ce processus. Un travail dans ce contexte est déjà fait par Liu, Tomson et Pederson (voir chap 03, section 4 - Travail Liu, Thomson et Pederson). Cependant, ce travail présente certaines failles ; par exemple, leur prototype disponible ne permet l'extraction de données qu'à partir des fichiers plats. Pour permettre une meilleure performance au processus ETL, nous avons développé un prototype qui permet l'extraction de données à partir d'une Base de données relationnelle (testé sur le SGBD MySQL). Afin de traiter ces données en parallèle, celles-ci seront partitionnées pour produire plusieurs fragments (blocs) où chacun sera pris en charge par un Mapper. Les résultats seront chargés dans des cubes de données à la volée pour l'analyse en ligne dans un format natif de Hadoop.

1. Architecture du système

L'architecture se présente en 04 niveaux :

- Les sources de données (MySQL et fichiers plats),
- L'extraction des données et le partitionnement des données,
- Transformations des données,
- Chargement dans des cubes distribués.

La figure ci-dessous représente l'architecture du système d'alimentation de cubes de données à la demande. Notre contribution est identifiée dans les parties colorées en jaune.

Dans ce qui suit, nous allons décrire d'une manière approfondie les différentes méthodes proposées en illustrant leurs fonctionnements par des algorithmes.

2. Extraction de données

L'extraction de données est la première étape dans le processus ETL. La caractéristique principale de cette tâche est d'extraire des données à partir de différents systèmes. Comme nous l'avons cité précédemment, l'environnement Hadoop ne permet l'extraction de données qu'à partir de fichiers plats (de format natif). Pour y remédier, nous avons proposé une méthode d'extraction à partir d'une base de données MySQL. De même, les sources de données intensives peuvent alourdir le processus lors du traitement. Pour cela nous avons mis au point deux partitionneurs de données. Nous avons également implémenté un convertisseur de format de données sources pour éviter de rester en ligne avec la base de données pendant toute la procédure d'extraction, transformation et chargement.

2.1. Extraction à partir d'une base MySQL

Pour permettre l'extraction à partir d'une base de données MySQL, nous avons mis en œuvre une UDF d'extraction appelée « MySqlLoader », voici sa signature : *MySqlLoader (host, base, user, password)*. Pour ce type d'extraction, l'utilisateur devra introduire les paramètres suivants constituant la chaine de connexion au serveur (hôte) :

- **host** : '/127.0.0.1/' ou localhost.
- **base** : nom de la base MySQL.
- **table** : nom de la table à extraire
- **user** et **password** : qui sont respectivement le nom d'utilisateur et mot de passe identifiant le compte d'accès dans la base MySQL.

ALGORITHME 1 MySqlLoader

Classe MySqlLoader hérite de LoadFunc
Attributs
Reader : Tuple[0..n-1]
next: entier

Chapitre 04 : Conception de notre système

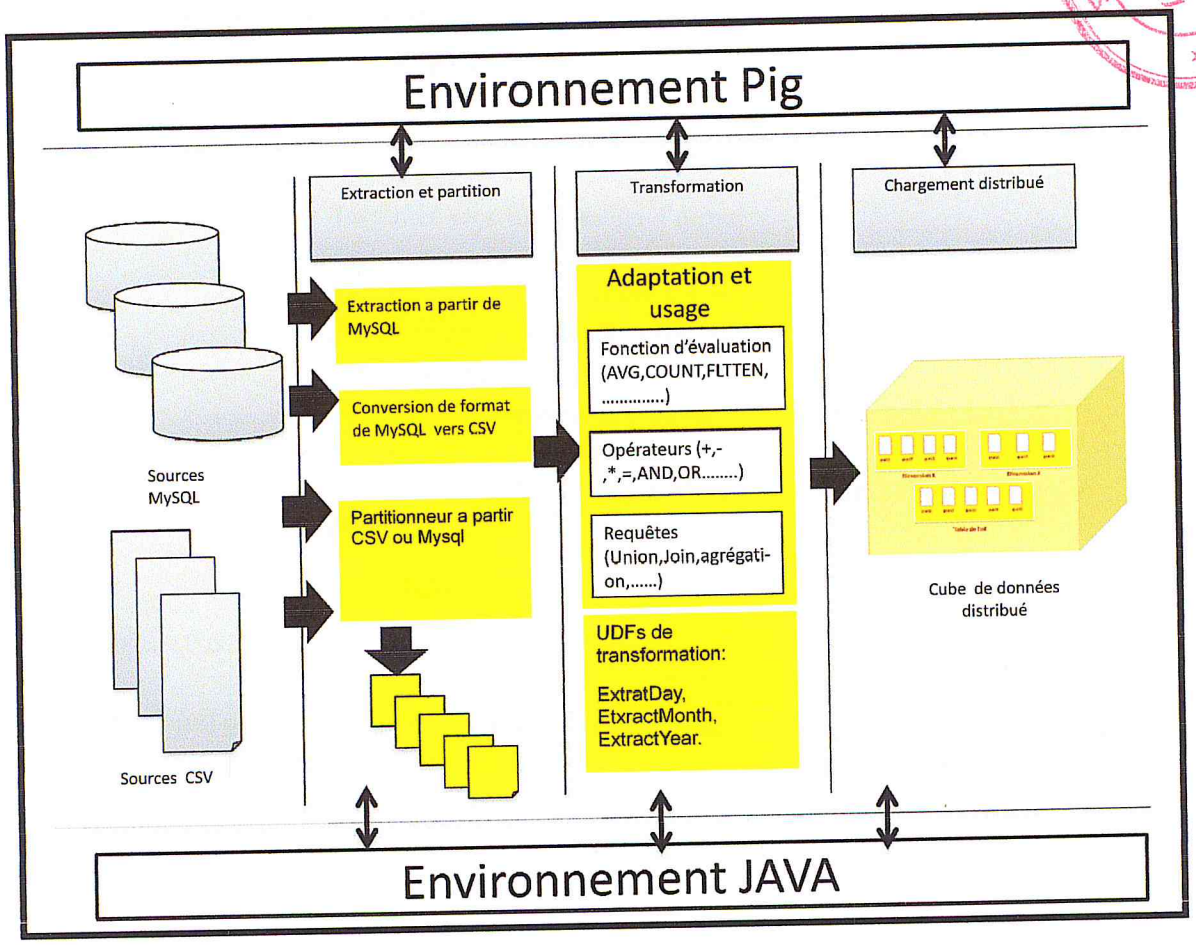


Figure 4.1. Architecture du système d'alimentation de cubes à la demande

Pour l'extraction, nous avons deux cas possibles : une extraction directe des données à partir des systèmes sources et leur passage à la phase suivante (transformation) ou une extraction indirecte dans le cas où les sources seront volumineuses (même sur un seul nœud du cluster). Dans le deuxième cas, les données sont extraites (converties également), puis partitionnées en blocs de format natif (fichiers CSV).

Et pour la transformation, nous avons adopté les éléments de Pig, telles que les fonctions d'évaluation (AVG, SUM, etc.), les opérateurs (arithmétiques : +,-, etc. et

```
            Retourner tg
        FinSi
    FinFonction
FinClasse
```

2.2. Conversion de format de données

Nous avons implémenté un convertisseur de format de données nommé « MySqlLoaderToCsv », voici sa signature : *MySqlLoaderToCsv (hote, base, table, user, passe, destination, délimiteur)*. Cette UDF offre la possibilité de convertir des tables d'une base de données MySQL vers des fichiers CSV pour éviter le problème de multi-accès à la base de données MySQL après l'étape d'extraction (lors des transformations) de ces données. Le principe est le même que l'extraction on rajoute juste la destination des données, le délimiteur et une fonction appelée « makeCsvLine ()» permettant de convertir chaque ligne extraite au format CSV. Cette fonction prend comme paramètre la ligne et le délimiteur.

ALGORITHME2 MySqlLoaderToCsv

```
Classe MySqlLoaderToCSV hérite de LoadFunc
    Attributs
        reader : Tuple[0..n-1]
        cible: chaîne
        Fini: booléen
    Constructeur MySqlLoaderToCSV (hote, base, table, user, passe, destination: chaînes, délimiteur:
        caractère)
    Variables
        cnx: Connexion
        req, emplacement, ligne: chaînes
        fic: Fichier
    Début
        Chargement du pilote MySql pour la connexion avec la base de données
        Si pilote Non Chanrgé Alors
                Terminer avec un message d'erreur
        Sinon
                cnx ← ChaineConnexion (hote, base, user, passe)
                Si cnx = null Alors
                        Terminer avec un message d'erreur
                Sinon
                        req ← "SELECT * FROM " + table
                        reader ← cnx.ExécuterRequête(req)
                        Si reader = null Alors
```

2.3. Partitionnement de données

Dans un contexte Big data, dont le traitement exige beaucoup de temps par rapport aux autres données et afin d'accélérer le traitement, celui-ci devra être parallélisé et pris en charge par plusieurs tâches parallèles. Pour ce faire, une phase de partitionnement s'impose. Elle consiste à fragmenter la quantité de données en plusieurs partitions où chacune sera soumise à une tâche (Map) Nous avons proposé deux méthodes de partitionnement selon le choix de l'utilisateur : un partitionnement simple et un partitionnement Round-Robin.

2.3.1. Partitionnement simple

Nous adoptons, dans cette méthode, le même principe utilisé dans HDFS de Hadoop. Un fichier volumineux est partitionné en d'autres fichiers de tailles plus petites appelés « blocs ». La taille de ces blocs est spécifiée par l'utilisateur, donc ce dernier peut augmenter ou diminuer cette taille selon son besoin. Grâce à cette méthode, nous pouvons contrôler le nombre de Mappers qui seront exécutés dans un job MapReduce de façon que le nombre de blocs est égal au nombre de tâches Map à exécuter. Puisque nous avons deux types de sources de données, nous avons mis en œuvre deux UDFs, la première traite sur des sources de type CSV et l'autre sur une base de type MySQL. Elles sont appelées respectivement « CSVLoaderPartitionner » et « MySqlLoaderToCsvPartitionner ».

Le principe de base de ces UDFs est le même, il consiste à extraire les données d'une source d'un certain format, les partitionner puis les charger dans les blocs correspondants. La différence réside juste dans le format des données en entrée. La première UDF lit les données à partir des fichiers simples (plats), et voici sa signature : *CSVLoaderPartitionner* (*Source*, *Delimiter*, *Size*, *Destination*, *Frag*).

L'utilisateur devra introduire les paramètres suivants :

- **Source :** Le chemin complet du fichier source CSV.
- **Delimiter** : Le délimiteur ou séparateur des champs pour le format CSV. Les plus utilisés sont : « \t », « ; », «,».
- **Size :** La taille d'une partition (ou bloc) en octets.
- **Destination** : L'emplacement cible des partitions (blocs).
- **Frag ou Fragname :** Le suffixe à ajouter au nom de la source pour désigner la partition. Les plus utilisés sont : « bloc », « frag », « part », « chunk ».

ALGORITHME 4 CSVLoaderPartitionner

```
Classe CSVLoaderPartitionner hérite de LoadFunc
    Attributs
        reader: Fichier
        partition: Liste<Fichier>[0..n-1]
        partnames: Liste<chaîne>[0..n-1]
        Fini: booléen
    Constructeur CSVLoaderPartitionner (source, destination, suffixe: chaînes, délimiteur:
        caractère, size: entier long)
    Variables
        i,p : entier
        longueur: entier long
        emplacement, ligne, ligneDesColonnes: chaînes
        Début
                Ouvrir source dans reader en lecture
                Si reader != null Alors
                        Lire (reader, ligneDesColonnes)
                        i ← 0
                        Répéter
                                // Création d'une partition
                                emplacement ← destination + Nom(source) + suffixe
                                + Str(p) + ".csv"
                                Ajouter emplacement à la liste des partnames
                                Ouvrir emplacement dans partition[p] en écriture
                                Ecrire (partition[p], ligneDescolonnes)

                                // Remplissage de la partition
                                longueur ← 0
                                Répéter
                                        Lire (reader, ligne)
                                        Ecrire (partition[p], ligne)
                                        longueur ← longueur + taille(ligne)
                                        i ← i + 1
                                Jusqu'à longueur >= size
                                p ← p + 1
                                Fermer partition
                        Jusqu'à fin de fichier
                        Fermer reader
                Finsi
     FinConstructeur

    Fonction GetNext(): PigTuple
    Variables
        tg: PigTuple
    Début
```

```
        Si ce n'est pas Fini Alors
                Fini ← Vrai
                tg ← ConstruireUnPigTuple(partnames)
                // tg est un PigTuple qui contiendra les noms des partitions et les fournir en sortie
                Retourner (tg)
        Sinon
                Retourner null   // marquer la fin de la retourne des tuples
        FinSi
    FinFonction
FinClasse
```

La deuxième UDF lit à partir d'une base de données relationnelle de type MySQL, chaque ligne extraite doit être convertie au format CSV ; les lignes converties sont distribuées par la méthode de partitionnement décrite ci-dessus. Voici la signature de cette UDF : *MySqlLoaderToCsvPartitionner (host, base, table, user, passwd, destination, size, frag, delimiter).*

En plus des paramètres déjà expliqués dans CSVLoaderPartitionner, l'utilisateur de cette UDF devra introduire aussi des paramètres concernant la BD, i.e. l'adresse du serveur de la BD (host), le nom de la BD (base), le nom de la table source (table), le nom d'utilisateur de la BD (user) et son mot de passe (passwd).

ALGORITHME 5 MySqlLoaderToCsvPartitionner

```
Classe MySqlLoaderToCSVPartitionner hérite de LoadFunc
    Attributs
        reader : Tuple[0..n-1]
        partnames: Liste<chaîne>[ ]
        Fini: booléen
    Constructeur MySqlLoaderToCSVPartitionner (hote, base, table, user, passe, destination,
        suffixe: chaînes, délimiteur: caractère, size: entier long)
    Variables
        cnx: Connexion
        req, ligne, ligneDesColonnes: chaînes
        colonnes: Tuple
        i, p : entier
        longueur: entier long
        partition: Fichier
    Début
        Chargement du pilote MySql pour la connexion avec la base de données
        Si pilote Non Chargé Alors
                Terminer avec un message d'erreur
```

```
    Sinon
        cnx ← ChaineConnexion (hote, base, user, passe)
        Si cnx = null Alors
            Terminer avec un message d'erreur
        Sinon
            req ← "SELECT * FROM  " + table
            reader ← cnx.ExécuterRequête(req)
            Si reader = null Alors
                Terminer avec le message "pas de résultat à fournir!"
            Sinon
                colonnes ← cnx.GetColumns(table)
                ligneDesColonne ← MakeCSVLine(colonnes, délimiteur)
                i ← 0
                Répéter
                    // Création d'une partition
                    emplacement ← destination + table + suffixe + Str(p) +
                    ".csv"
                    Ajouter emplacement à la liste des partnames
                    Ouvrir partition dans emplacement en écriture
                    Ecrire (partition, ligneDescolonnes)

                    // Remplissage de la partition
                    longueur ← 0
                    Répéter
                        ligne ← MakeCSVLine(reader[i], délimiteur)
                        Ecrire (partition, ligne)
                        longueur ← longueur + taille(ligne)
                        i ← i + 1
                    Jusqu'à  longueur >= size OU  i >= n
                    p ← p + 1
                    Fermer partition
                Jusqu'à i >= n
                Fermer cnx
            FinSi
        FinSi
    FinSi
FinConstructeur

Fonction GetNext(): PigTuple
Variables
    tg: PigTuple
Début
    Si ce n'est pas Fini Alors
        Fini ← Vrai
        tg ← ConstruireUnPigTuple(partnames)
        Retourner  tg
```

```
        Sinon
                Retourner null
        FinSi
    FinFonction
FinClasse
```

2.3.2. Partitionnement par la méthode *Round-Robin*

Le principe est simple, Cette méthode permet de distribuer les lignes sur les tâches, tel que la ligne numéro **n** est assignée à la tache numéro (**n** mod **nr_map**) ou **nr_map** est le nombre de tâche. Dans cette méthode, l'utilisateur spécifié le nombre de Mappers à utiliser dans le programme, ensuite par la formule de partitionnement ci-dessous les lignes sont distribuées sur ces Mappers de façon que pour chaque Mapper une partition a été créée.

Pour ce faire nous avons mise en œuvre un UDF appelée « RoundRobinLoader », voici sa signature : *RoundRobinLoader (delimiter, nr_map, partition)*. Cette UDF donne la possibilité aux utilisateurs de partitionner un fichier volumineux en d'autres fichiers de tailles plus petits en basant sur le nombre de Mappers spécifié. Donc notre système peut adapter d'autres méthodes de partitionnement plus de la méthode citée au-dessus.

ALGORITHME 6 RoundRobinLoader

```
Classe RoundRobinLoader  hérite de LoadFunc
    Attributs
        Reader : Tuple[0..n-1]
        Nbr_mapper : entier
        nb_line : entier
    Constructeur  RoundRobinLoader (delimiter : caractaire, nr_map, partition : entier)
    Début
        Nbr_mapper ← 1  // au moin un seul mapper
        Si  nr_map <= 0 Alors
                nr_map ← nbr_mapper
        FinSi
        // il faut que partition soit inferieur au nr_map
        Si partition >= nr_map Alors
                Afficher message d'erreur
        FinSi
    FinConstructeur

    // Fonction qui sert à fournir un tuple de données, c'est une fonction surchargeable de l'UDF,
        cette fonction est appellée continuellement par l'UDF jusqu'à ce qu'elle un null:
    Fonction GetNext(): PigTuple
```

```
    Variables
        t: Tuple
        tg: PigTuple
    Début
        Si reader = null ou  taille(reader) <= nb_line Alors
                Retourner null
        Sinon
                nb_line ← nb_line +1
                Si (nb_line mod nr_map) = partition  Alors
                        t ← reader[nb_line]
                        tg ← ConstruireUnPigTuple(t)
                        Retourner  tg
                FinSi
        FinSi
    FinFonction
FinClasse
```

3. Transformation de données

La transformation de données est la tâche la plus complexe dans le processus et qui demande beaucoup de réflexion. L'environnement Hadoop Pig est dédié pour le traitement de données intensives mais pas orienté DataWarehousing, pour cela nous avons rajouté quelques fonctions, non disponible dans Pig, nécessaires pour la transformation de données telles que :ExtractYear, ExtractMonth et ExtractDay qui permettent d'extraire l'année, le mois et le jour à partir d'une date donnée.

4. Chargement de données

Le chargement est la dernière phase d'un processus ETL. Le contexte dans lequel nous travaillons est caractérisé d'abord par des données intensives et un environnement virtuel et distribué. La destination finale des données préparées est généralement l'entrepôt de données (datawarehouse) ou le magasin de données (datamart). En termes de stockage, le framework Hadoop avec son moteur de base et le module HDFS ne sont pas très appropriés pour héberger de grandes quantités de données. L'organisation des volumes de données sous forme de blocs distribués est destinée plutôt pour les traitements. Pour le stockage, il faut faire appel à d'autres solutions dédiées pour les données intensives telles que Hadoop Hbase, Cassandra, etc. L'environnement dans lequel nous avons mis en

œuvre notre prototype permettra, par contre, l'alimentation des cubes sans entreposage, c'est ce qui est appelée communément Cubes de données pour l'analyse en ligne à la volée ou à la demande. Comme les données sont massives, le stockage centralisé d'un cube dans le système Hadoop peut s'avérer négatif sur les performances du chargement. L'impact de cette centralisation peut se résumer comme suit :

Malgré que la taille d'un cube est très petite par rapport à celle d'un entrepôt de données, le stockage du cube sur une seule machine du cluster peut provoquer une charge qui dégrade les performances du chargement mais aussi celles de l'analyse en ligne lorsqu'il s'agit d'exploiter ce cube par des outils de restitution.

Pour cela, nous avons proposé une méthode de chargement distribuée dans des cubes de données à la volée.

4.1. Description

La méthode adoptée permet donc l'alimentation de cubes de données distribués. Pour ce faire, l'utilisateur doit paramétrer cette méthode en donnant des valeurs pour le blocname, startIndex, path, size et le delimiter dont voici la désignation de chaque paramètre :

- **blocname** : permet de nommer les partitions selon le choix ou le besoin d'utilisateur par exemple : part, bloc, fragment, etc.
- **startIndex** : permet de spécifier la valeur initiale (de départ), par exemple : 0, 1, 2, 3, etc.
- **path** : permet de spécifier le chemin de cube de données où les partitions seront stockées. Par exemple, si on veut stocker les données d'une dimension « titlesdim » dans le cube de données « CUBE1 », on ajoute le nom de la dimension à la fin de notre path « /CUBE1/titlesdim » pour avoir un chemin complet comme suit : « /root/CUBE1/titlesdim », dont ce dernier contient toutes les partitions de la dimension « titlesdim ».
- **size** : la taille maximale que peut prendre une partition.
- **delimiter :** séparateur de champs.

Au début, la méthode permet de créer et de nommer la première partition. Le système de nommage consiste à la concaténant du paramètre path avec un « / », blocname et

startIndex. Ensuite, vient l'étape de l'insertion des nouveaux tuples dans cette partition jusqu'à que la taille des tuples insérés sera égale au « size ». Dans ce dernier cas, il incrémente le startIndex et crée la deuxième partition et ainsi de suite jusqu'a ce qu'il n'y aura aucun tuple à insérer.

Algorithme 7 MyStorerDistributer

```
Classe MyStorerDistributer hérite de StoreFunc
    Attributs
        destination, suffixe: chaînes
        délimiteur: caractère
        blocsize, longueur: entier long
        next: entier
        bloc: Fichier

    Constructeur MyStorerDistributer (path, fragname: chaînes, séparateur: caractère, size: entier
    long)
    Variables
        Méga: entier long
    Début
        Méga ← 1024*1024
        Si path existe Alors
                destination ← path
        Sinon
                destination ← "/tmp/"
        FinSi
        suffixe ← fragname OU "bloc"
        délimiteur ← séparateur OU '\t'
        blocsize ← size OU 64*Méga
        next ← 0
        longueur ← 0
     FinConstructeur

    // Cette fonction est appelée continuellement par l'UDF en lui fournissant un PigTuple à
    chaque fois, jusqu'à ce qu'il n'y aura plus de tuples à lui fournir.
    Fonction  PutNext(PigTuple tg)
    Variables
        Colonnes, t: Tuples
        emplacement, ligne, ligneDesColonnes: chaînes
    Début
        // Création d'un nouveau bloc
        Si longueur = 0 Alors
                emplacement ← destination + suffixe + Str(next) + ".csv"
                Ouvrir bloc dans emplacement en écriture
```

```
                colonnes ← tg.GetColumns( )
                ligneDesColonnes ← MakeCSVLine(colonnes, délimiteur)
                Ecrire (bloc, ligneDesColonnes)
        FinSi
        //  Conversion du PigTuple tg à un simple Tuple
        t ← ConstruireUnSimpleTuple(tg)
        ligne ← MakeCSVLine(t, délimiteur)
        Ecrire (bloc, ligne)
        longueur ← longueur + taille(ligne)

        Si longueur >= blocsize Alors
                Fermer bloc
                longueur ← 0
                next ← next + 1
        FinSi
    FinFonction
FinClasse
```

Figure 4.2.Chargement de cubes de données.

5. Conclusion

Au cours de ce chapitre, nous avons tenté d'expliquer toutes les méthodes proposées pour permettre la mise en œuvre de l'ETL sous l'environnement parallèle Hadoop Pig dont le but est de performer ce processus. Le chapitre suivant présentera les détails de notre implémentation de ces méthodes.

Chapitre 05

Implémentation et validation

Dans ce chapitre, nous présenterons en premier lieu la configuration matérielle et logicielle nécessaire la pour manipulation du Framework MapReduce. Ensuite, nous allons présenter le choix des outils logiciels utilisés pour réaliser ce projet où nous citons le projet Hadoop et leurs composants ainsi que le framework Pig et le langage PigLatin. Enfin, nous allons décrire l'implémentation du notre système en détaillant les différentes fonctionnalités de ce dernier.

1. Configuration nécessaire

Comme nous l'avons mentionné dans le chapitre 03, le modèle MapReduce est basé sur le traitement parallèle sur un cluster des nœuds ou même dans une seule machine multiprocesseurs (multicores). Donc, la manipulation du MapReduce nécessite au mois une machine multicore.

1.1. Configuration matérielle

1.1.1. Machine multicore (n cores ou n CPUs)

Avec un système multicore à 'n' processeurs, on peut se charger de 'n' tâches (map ou reduce) parallèles. Une tâche va jouer le rôle de master et les autres sont les esclaves.

1.2. Configuration logicielle

1.2.1. Système d'exploitation supporte les normes POSIX

Tous les Frameworks implémentant le MapReduce jusqu'à présent ont été démontrés sur des systèmes d'exploitation qui supportent les normes POSIX et non pas sur des systèmes Win32/64 (les Windows 32 ou 64 bits). Parmi les systèmes d'exploitation qui supportent le POSIX, nous pouvons citer les systèmes Linux/Unix, MacOS X, et Windows NT.

1.2.2. Serveur HTTP Linux/Unix

Celui qui assure la communication entre l'utilisateur et le maître suivant un certain protocole, le plus simple est le fameux protocole HTTP.

1.2.3. SSH daemons (SSH server/client)

C'est un protocole de communication sécurisé qui s'applique dans ce cas sur le Master et les nœuds esclaves, ces derniers seront lancés par le Master à travers ce protocole.

1.2.4. DFS (Distributed File System)

C'est le système de fichiers distribués, il est utilisé pour le partitionnement de données et la synchronisation des tâches. Nous trouvons que Hadoop comporte un DFS qui est noté HDFS (Hadoop DFS).

1.2.5. Framework MapReduce & l'environnement d'exécution

Ce sont les librairies d'implémentation de paradigme MapReduce, il existe plusieurs implémentations dans plusieurs langages comme le Java pour le projet Hadoop. L'utilisation de ce Framework nécessite l'environnement d'exécution adéquat, selon le langage d'implémentation.

1.2.6. Langage utilisateur

C'est le langage de programmation utilisé par l'utilisateur pour interroger le 'Master' et lui communiquer les UDFs (User Defined Functions). Ces derniers sont des fonctions définies par l'utilisateur.

A cause de ces besoins logiciels et matériels, nous choisissons de travailler avec les solutions suivantes présentées ci-dessous :

2. Outils de développement logiciels

2.1. Projet Hadoop

Hadoop est un environnement d'exécution distribué, scalable et performant. Il propose une nouvelle façon de stocker des données massives et les traitées de manière parallèle en un temps record. Hadoop à été créé par Doug Cutting et fait, en 2009, partie des projets de la fondation logicielle Apache. C'est un Framework java libre destiné aux applications distribuées, Il permet aux ces application de travailler avec des milliers des **nœuds** et des pétaoctets des données sur un grand cluster [White 2009] .

2.2. Hadoop Distributed File System

C'est un système de fichier distribué destiné de mettre en place un grand volume de données et fournir un haut niveau pour accéder aux informations, basé sur Google File System (GFS). HDFS stocke les données dans les nœuds de manière transparente de point de vue de l'utilisateur et fournit une bonne liaison entre eux à travers le cluster [W05].Plus de détails sur l'HDFS, sont disponibles en Annexe A.

2.3. Hadoop MapReduce

Hadoop MapReduce est un framework de développement informatique, utilisé pour simplifier l'écriture des applications qui traitent une grande quantité de données sur un large cluster. Hadoop peut exécuter ces applications qui peuvent être écrites en plusieurs langages comme Java, Ruby, Python, et C++ [W01].

2.4. Framework Pig

Pig est une plateforme de haut niveau destinée pour l'analyse de données intensives. Pig est inspiré par Yahoo ! [W06] ; en 2007, Hadoop commence d'adopter le Pig, après il est gradué par Incubateur et devient un subprojet de apache Hadoop. En 2010, le Pig a devenu le niveau le plus haut de projet apache [W04].

La structure de Pig est divisée en deux pièces [White 2009] :

- Le langage utilisé pour exprimer l'échange de données traitées, nommé PigLatin.
- L'environnement d'exécution « Compilateur » pour exécuter les programmes Pig. Actuellement, il ya deux environnement : exécution locale dans un seul *JVM* et exécution distribuée dans un Hadoop cluster.

2.5. Ubuntu Linux

Ubuntu est une distribution Linux qui réunit stabilité et convivialité. Linux ou GNU/Linux est un système d'exploitation libre multitâche, multiplateforme et multiutilisateurs de type Unix. Pour l'utilisateur final, Linux se présente sous la forme d'une distribution Linux, commerciale ou non, c'est-à-dire d'une solution prête à être installée comprenant une sélection complète et cohérente de logiciels, des programmes d'installation et d'administration de l'ordinateur, ainsi qu'un mécanisme facilitant l'installation et la mise à jour des logiciels. Linux est aujourd'hui utilisé sur de nombreuses plate-formes, du plus puissant super-ordinateurs aux systèmes embarqués tels que téléphone portable, assistant personnel, modem Freebox, lecteur vidéo DivX, etc., en passant par les ordinateurs personnels, PC et Mac, sur lesquels il peut être installé seul ou en parallèle avec Microsoft Windows ou Mac OS[W02].

2.6. Langage PigLatin

C'est un langage du flux de données qui a été proposé pour offrir un compromis entre le langage déclaratif de type SQL et le style procédural bas niveau de MapReduce. C'est un langage de script tel qu'un script Pig latin se compose par une série d'opérations et de transformations qui s'appliquent aux entrées « inputs » afin de produire un ou plusieurs sorties « outputs », ces transformations sont exécutées dans une série de jobs MapReduce. Plus de détails sur ce langage, sont disponibles en Annexe B.

2.7. Langage JAVA

Java est le nom d'une technologie mise au point par Sun Microsystems qui permet de produire des logiciels indépendants de toute architecture matérielle. Java est à la fois un langage de programmation et une plateforme d'exécution. Le langage java a la particularité principale d'être portable sur plusieurs systèmes d'exploitation tels que Windows, Mac OS ou Linux. C'est la plateforme qui garantit la portabilité des applications développées en Java [W03].

3. Outil matériel

3.1. Machine multicore

Une machine multicore ne peut avoir un grand nombre de processeurs sans engendrer le problème de placement. La figure ci-dessous présente l'architecture de Hadoop Pig dans une machine Multicore (quatre-cores).

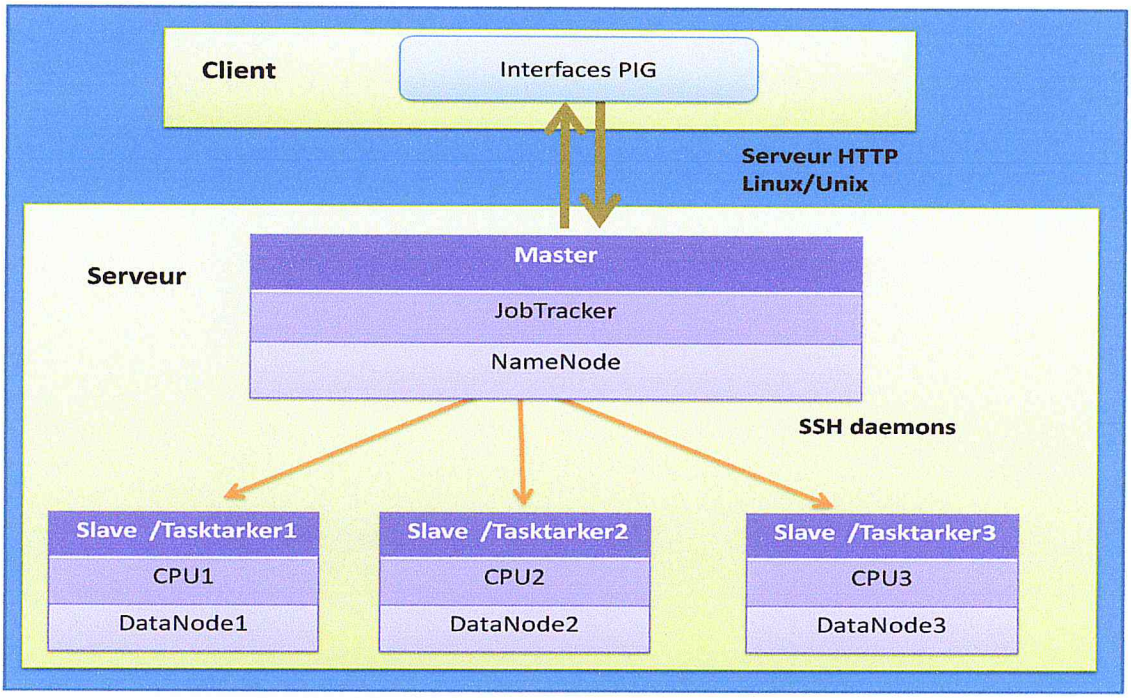


Figure 5.1 .Architecture de Hadoop Pig dans une machine Multicore

Comme présenté dans la figure 5.1, dans notre cas, l'utilisateur (client) et le master s'exécuteront sur une même machine. L'utilisateur communique ses requêtes au master à l'adresse (http://localhost) ou (http://127.0.0.1).

La communication entre client / master, et master slaves passe comme suit :

1. Le Master reçoit les requêtes du travail « job », à distribuer sur les nœuds slaves, par l'utilisateur qui à son tour envoie ces requêtes via un protocole de communication HTTP. L'utilisateur peut être un client ou un utilisateur développeur (programmeur).

2. Le Master démarre les Slaves tout en leurs communicant leurs tâches via le protocole de communication sécurisé SSH, il fait référence pour l'accès distant sur les stations Linux.

4. Implémentation

Pig fournit un support étendu pour les fonctions définies par l'utilisateur « UDF » comme un moyen de spécifier un traitement personnalisé.

4.1. Implémentation des fonctions d'extraction

Pour les fonctions d'extraction, Pig fournit une classe abstraite LoadFunc, qui est la classe principale pour étendre la mise en œuvre d'un extracteur (Loader). Telles que, les méthodes qui ont besoin d'être remplacées sont expliquées ci-dessous :

- **getInputFormat () :** Cette méthode est appelée par Pig pour obtenir le inputFormat utilisé par le Loader.
- **setLocation () :** Cette méthode est appelée par Pig pour communiquer l'emplacement de la charge au Loader. Le Loader devrait utiliser cette méthode pour communiquer les mêmes informations que l'inputFormat. Cette méthode est appelée plusieurs fois par Pig pour s'assurer qu'il n'ya pas d'effets secondaires incompatibles en raison de multiples appelles.
- **prepareToRead () :** Grâce à ce procédé, le RecordReader associé à l'inputFormat fourni par le LoadFunc est passé à ce dernier. Le RecordReader peut ensuite être utilisé par la mise en œuvre de getNext () pour retourner un tuple représentant un enregistrement de données au Pig.
- **getNext () :** Le sens de getNext () n'a pas changé, elle est appelée par l'exécution de Pig pour obtenir le prochain tuple de données. Dans cette méthode, la mise en œuvre devrait utiliser le RecordReader pour la construction du tuple de retour.

Pour ce type de fonctions nous avons développé cinq UDFs sont les suivantes :

- **MySqlLoader() :** permet d'extraire les données à partir d'une base de données relationnelle « MySQL » .
- **MySqlLoaderToCsv() :** permet d'extraire et de convertir des données d'une table MySQL vers des données de format CSV.
- **CsvLoaderPartitionner() :** permet d'extraire et de partitionner un fichier « CSV » volumineux en d'autres fichiers CSV de tailles plus petites par une méthode simple.

- **RoundRobinPartitionner() :** permet d'extraire et de partitionner un fichier « CSV » volumineux en plusieurs partitions de tailles plus petites par la méthode RoundRobin .
- **MysqlLoaderToCsvPartitionner() :** permet d'extraire ,de convertir et de partitionner une table relationnelle « MySQL » volumineuse en d'autres fichiers de type CSV de tailles plus petites.

4.2. Implémentation des fonctions de transformation

Pour les fonctions de transformations, Pig fournit une classe abstraite EvalFun, qui est la classe principale pour étendre la mise en œuvre des agrégations comme : Somme, comptage Max, etc. et autres transformations possibles. Telle que, la méthode qui a besoin d'être remplacée est expliquée ci-dessous :

- **exec () :** Cette fonction sera appelée par l'exécution de Pig à chaque entrée. L'entrée de cette fonction est un tuple contenant les paramètres d'entrée.

Pour ce type de fonctions nous avons développé trois UDFs sont les suivantes :

- **ExtractYear() :** permet d'extraire l'année à partir d'une date complet.
- **ExtractMonth () :** permet d'extraire le mois à partir d'une date complet.
- **ExtractDay() :** permet d'extraire le jour à partir d'une date complet.

4.3. Implémentation pour les fonctions de chargement

Pour les fonctions de chargement, Pig fournit une classe abstraite StoreFunc, qui est la classe principale pour étendre la mise en œuvre d'un stockeur (Storer). Telles que, les méthodes qui ont besoin d'être remplacées sont expliquées ci-dessous :

- **getOutputFormat() :** Cette méthode sera appelée par Pig pour obtenir le OutputFormat utilisé par le Storer.

- **SetStoreLocation () :** Cette méthode est appelée par Pig pour communiquer l'emplacement de stockage. Cette méthode est appelée plusieurs fois par Pig et elle doit s'assurer qu'il n'ya pas d'effets secondaire incompatibles en raison des multiples appels.

- **prepareToWrite () :** Dans la nouvelle API, l'écriture de données se fait par l'outputFormat (). Le recordWriter peut ensuit être utilisé par la mise en œuvre de putNext() pour écrire un tuple représentant un enregistrement de données d'une manière attendue par le recordWriter .

- **putNext () :** Est appelée par l'exécution de Pig pour écrire le prochain tuple de données dans la nouvelle API, ici il s'agit de la méthode dans laquelle la mise en œuvre utilisera le RecordWriter pour écrire dans l'objet tuple.

Pour ce type de fonctions nous avons développé une UDF :

- **MyStorerDistributer() :** permet le chargement distribué dans un cube de données.

4.4. Implémentation de notre API

Pour pouvoir intégrer les commandes nécessaires de Hadoop et Pig dans le langage java afin de démarrer/arrêter les daemons (JobTracker, TaskTracker, NameNode et DataNode), manipuler les fichiers d'HDFS, lancer un script PigLatin et aussi afficher les résultats d'exécution des jobs MapReduce dans la console d'eclipse. Nous avons proposé de créer une API java Pig (Framework basé code) qui rassemble tous les composants nécessaires pour garantir la fonctionnalité des opérations au-dessus.

- **Opérations de Hadoop et HDFS**

Pour les commandes liées au Hadoop, par exemple : start-all.sh, stop-all.sh (démarrer /arrêter tous le daemons) et HDFS comme : ls,mv, copyFrmLocal, etc. (pour la manipulation des fichiers), nous avons crée deux classes JCommand et HDFS Manip (voir le tableau).

- **Operations de Pig**

Pour définir la syntaxe complète de langage PigLatin (type de données, expressions, fonctions, opérateurs, statements, etc.) dans le langage Java, et pour exécuter un script PigLatin directement dans la console d'eclipse, nous avons développé des classes citées dans le tableau suivant :

Classe	Fonctions	Rôle
JCommand	+**execCommand**()	-Exécution des commandes système et renvoyer leur résultat d'exécution
HDFSManip	+**ls** (path, recurssif) + **mv** (src, dst) + **cp** (src, dst) +**copyFromLocal** (localsrc,dst) +**rm** (src, skipTrash, recurssif)	-Manipulation de Fichiers HDFS : afficher des fichiers, renommés, copiés, supprimés, etc.
DataType		-Déclaration globale des types de données

Column	+**getName()** +**setName**(name) +**getType**() +**setType**(type)	-Définition de la structure d'une colonne d'une table
Foncs	+**quote**(expression) +**getProjectPath**() +**getAllMyPath**(filename) +**drawTuple**(columns, prenthese) +**fotmatSize**(bytes) +**tokensToListofString**(tokens) +**getFirst**(url,delim)	-Définition de toutes les fonctions nécessaire pour la manipulation des données dans Pig.
Table	+**getName**() + **setName**(name) + **getAlias**() + **setAlias**(alias) + **getPath**() + **setPath**(path) +**getMethod**() + **setMethod**(method) +**getParams**() + **setParams**(params) + **getcolumns**() + **setcolumns**(columns)	- Définition de la structure d'une table de base de données et les métadonnées nécessaire pour manipuler cette table dans Pig
TableCube	Les mêmes fonctions de la classeTable plus : +**getReference**() +**setReference**(reference)	-Définition de la structure d'une table (dimension ou fait) dans un cube de données
MyPigEvalFunction	+**evaluate**(function, expressions) +**AVG**(expression) +**CONCAT**(expression1,expression2) + **COUNT**(expression) + **COUNT_STAR**(expression) + **DIFF**(expression1, expression2) + **IsEmpty** (expression) +**MAX**(expression) + **MIN**(expression) + **SIZE**(expression) + **SUM**(expression) + **TOKENIZE**(expression + **CAST**(String, field) + **FLATTEN**(tuple) +**AS**(expression, attr) + **getAttrib**(index, attr)	-Définition des fonctions algébriques pour la manipulation de données dans Pig
MyPigOperator	+**addition**(expression1,expression2, parenthese) +**substraction**(expression1,expression2,	- Définition de tous les opérateurs possibles et

	parentheses); +**multiplication**(expression1,expression2, parentheses) +**division**(expression1,expression2, parentheses) +**modulo**(expression1,expression2:String, parentheses) +**bincond**(condition,valueIfTrue, valueIfFalse) + **equal**(expression1, expression2) + **notEqual**(expression1, expression2) + **lessThan**(expression1, expression2) +**greaterThan** (expression1, expression2) +**lessThanOrEqualTo**(expression1, expression2) +**greaterThanOrEqualTo** (expression1, expression2) +**String matches**(expression1, expression2) + **String isNull**(expression) + **isNotNull**(expression) +**AND**(expression1,expression2) + **OR**(expression1, expression2) + **NOT**(expression) + **positive**(expression) + **negative**(expression)	nécessaires pour manipuler les données dans Pig
MyPigQuery	+**group**(target, aliases,expressions , method, params, parallel) +**join**(target, expressions,parallel) +**Split**(alias,aliases ,expressions) +**cross**(target,alias, parallel) +**distinct**(target, alias, parallel) +**filter**(target,alias, expression) + **foreachGenerate**(target, alias, expressions) + **limit**(target, alias, count) +**load**(alias,path,method,params,columns) +**sample**(target,alias, size) + **store**(alias, output, method, params) +**union**(target, aliases)	-Définition de toutes les opérations relationnelles pour la manipulation des données dans Pig

MyPigServer	+**existsFile**(filename) + **executeQuery**(query) + **dumpSchema**(alias) + **dump**(alias) +**openIterator** (id) + **registerJar**(jarPath) +**setdefaultParallel**(p) + **fileSize**(filename +**renameFile**(source,target) + **deleteFile**(filename) + **listPath**(dir) + **mkdirs**(dirs)	-Définition de toutes les opérations nécessaires pour exécuter le script PigLatin dans Java
Script		Décrire la procédure d'extraction, de transformation et du chargement dans un cube distribué en utilisant tous les classes citées précédemment dans ce tableau

Tableau5.1. Différents fonctions de Notre API

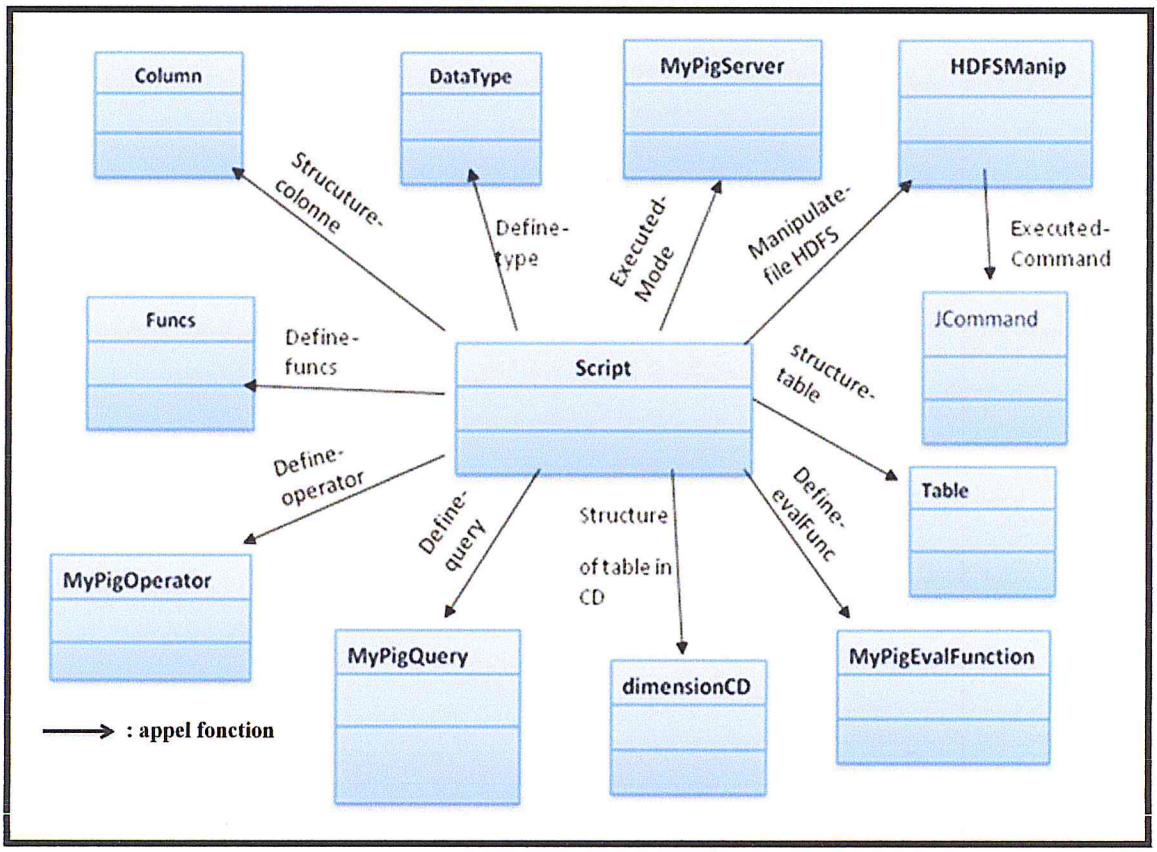


Figure 5.2. Architecture de notre API.

5. Vulgarisation de l'environnement Hadoop

Nous avons développées en plus une interface pour faciliter à un débutant l'accès aux paramètres Hadoop.

L'interface principale (voire figure 5.3) comporte un menu avec trois éléments : Système, Stockage d'HDFS et Processus.

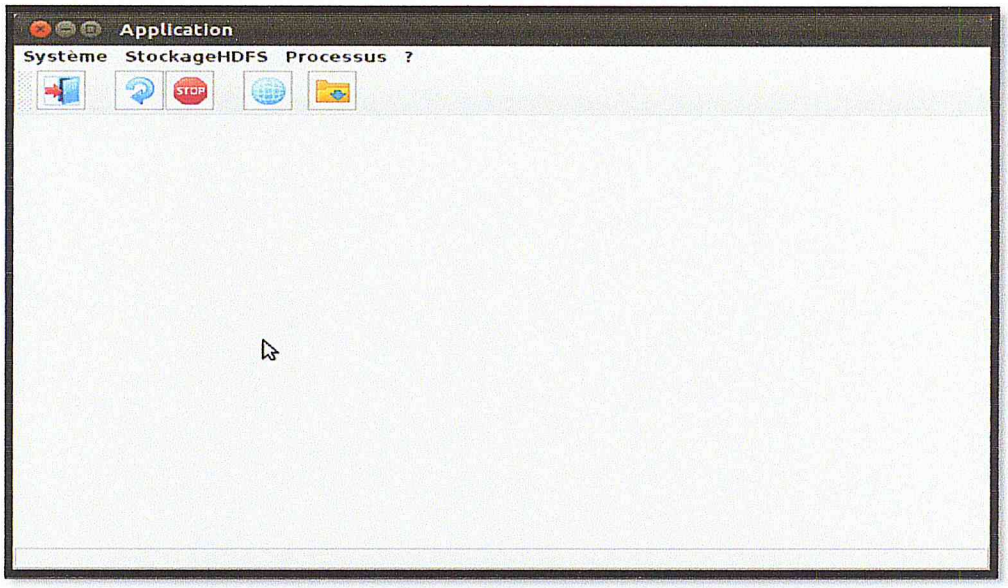


Figure 5.3. L'interface principale

Le menu « Système » (voire figure 5.4) permet la gestion i.e. L'allumage et l'éteignement des composants de Hadoop (HDFS et MapReduce). Ces composants peuvent être allumés ensemble ou séparément, voire les figures ci-dessous.

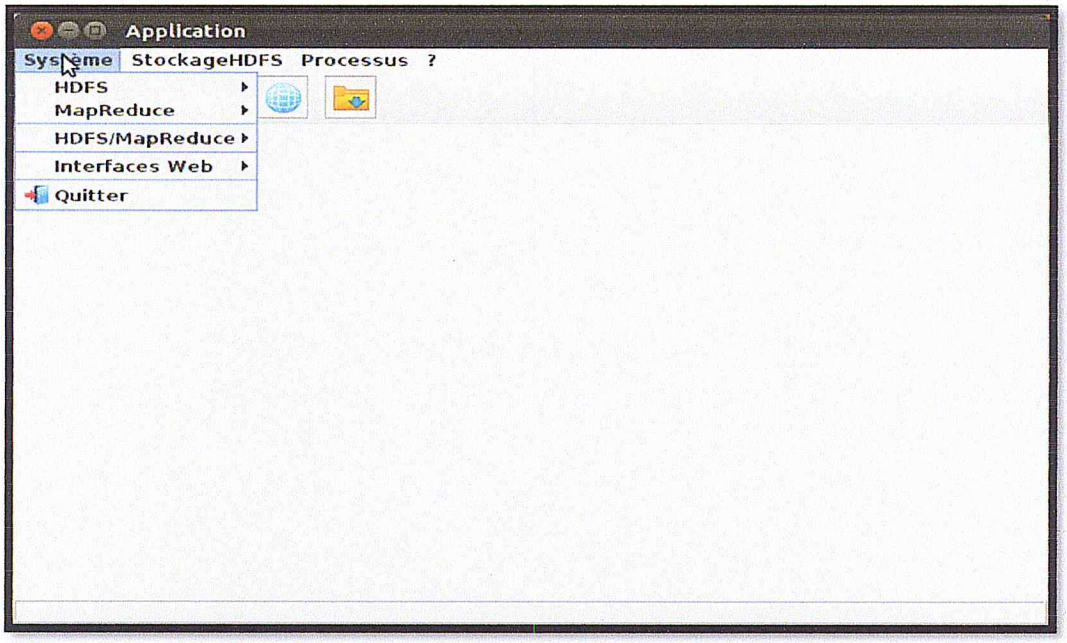


Figure 5.4. Présentation du menu système

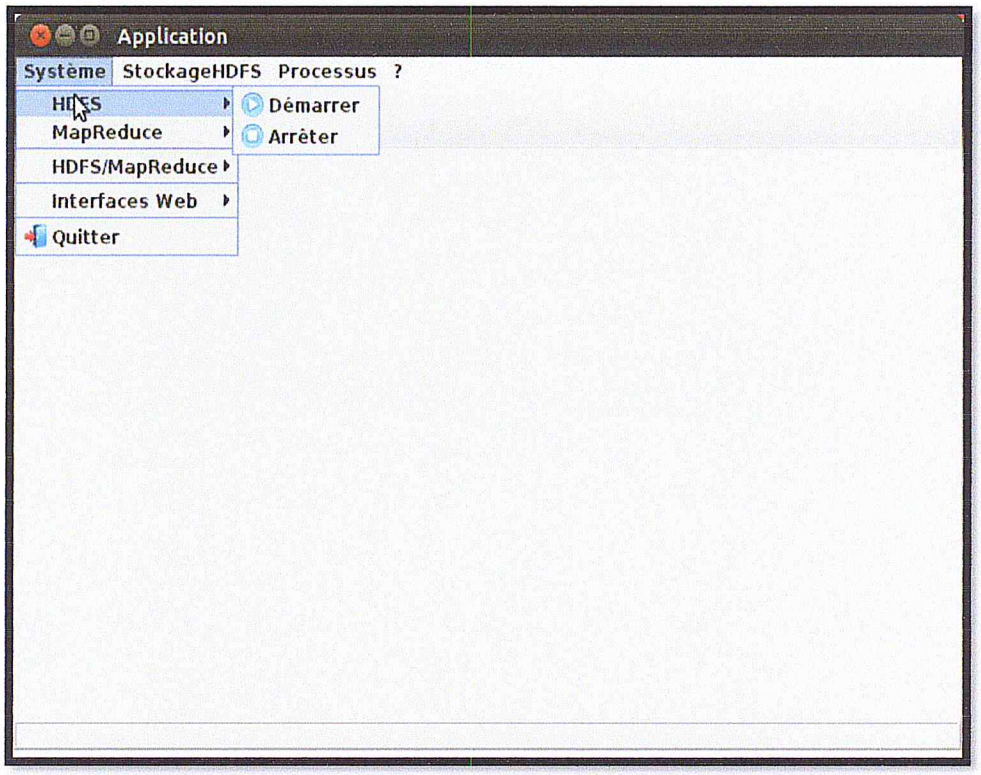


Figure 5.5. Démarrage et l'arrêt de l'HDFS

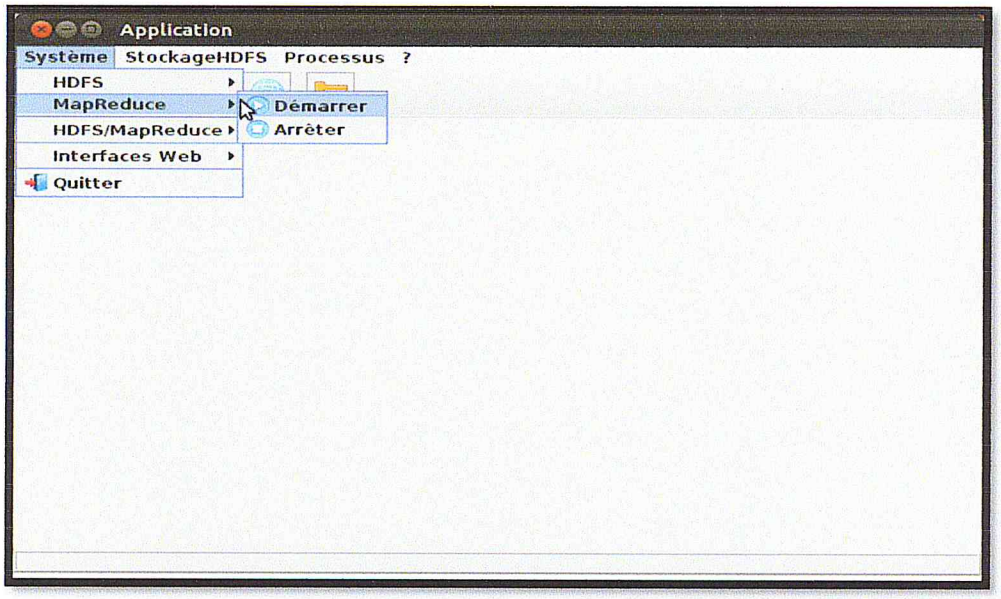


Figure 5.6. Démarrage et l'arrêt du MapReduce

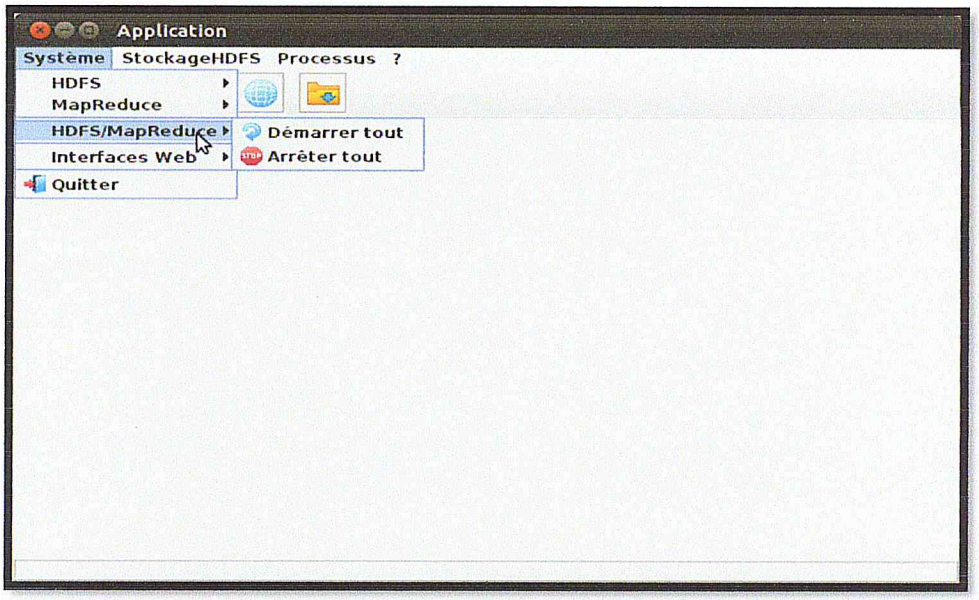


Figure 5.7. Démarrage et l'arrêt de l'HDFS et MapReduce

Ainsi, Hadoop est livré avec plusieurs interfaces web qui sont par défaut disponibles aux endroits suivants :

- http://localhost:50070/ - interface web du démon NameNode
- http://localhost:50030/ - interface web du démon JobTracker
- http://localhost:50060/ - interface web du démon TaskTracker

Les interfaces Web fournissent des renseignements concis sur ce qui se passe dans le cluster Hadoop. Voici la liste de toutes ces interfaces ainsi que leur explication :

1. NameNode Interface Web (HDFS couche)

Le nom du nœud de l'interface utilisateur Web affiche un récapitulatif de cluster, y compris des informations sur la capacité totale / restante, nœuds vivants et morts. En outre, il permet de parcourir l'espace de noms HDFS et d'afficher le contenu de ses fichiers dans le navigateur. Il donne également l'accès à la «machine locale » Fichiers journaux d'Hadoop.

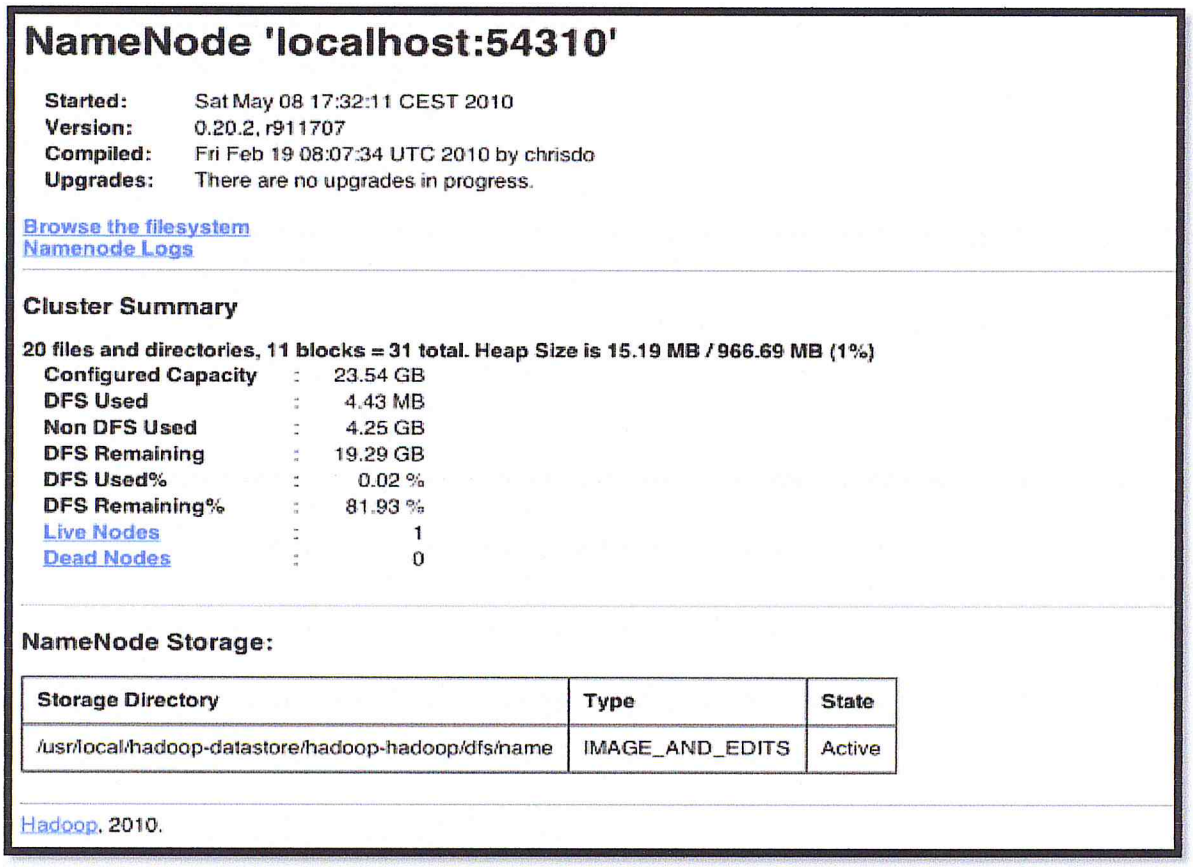


Figure 5. 8. Interface de Hadoop NameNode web [W09].

2. JobTracker interface Web (couche MapReduce)

L'interface web de jobTracker fournit des informations sur les statistiques d'emploi général du cluster Hadoop : en cours / travaux terminés / échec et le fichier journal des

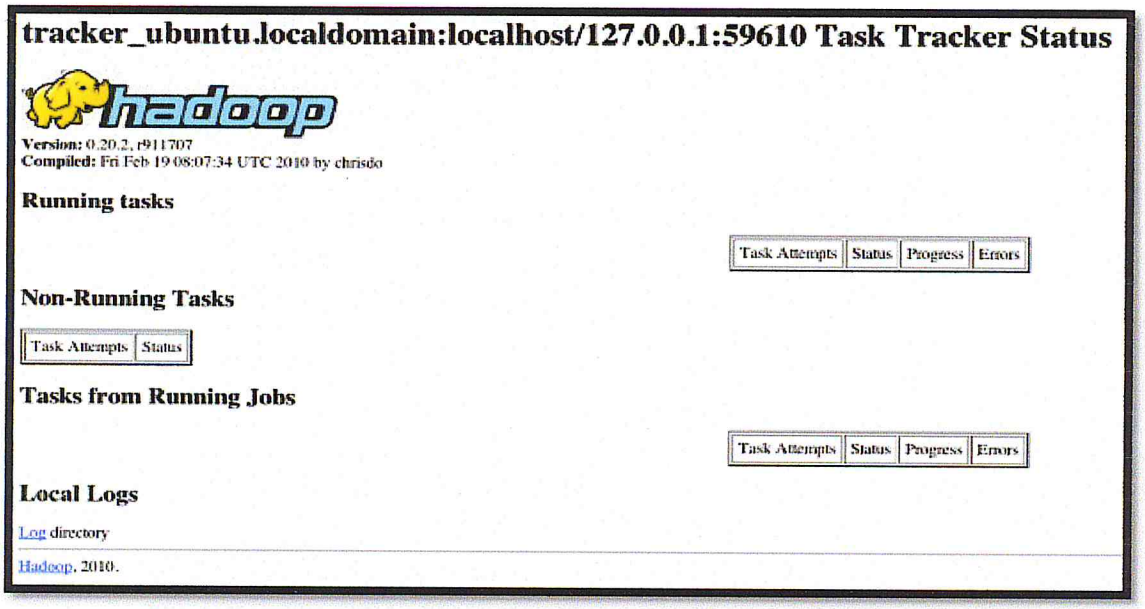


Figure 5.10. Interface Hadoop de Tasks Trackers Web [W09].

L'élément interface web de « Système » permet l'ouverture de toutes ces interfaces (voire figure 5.11).

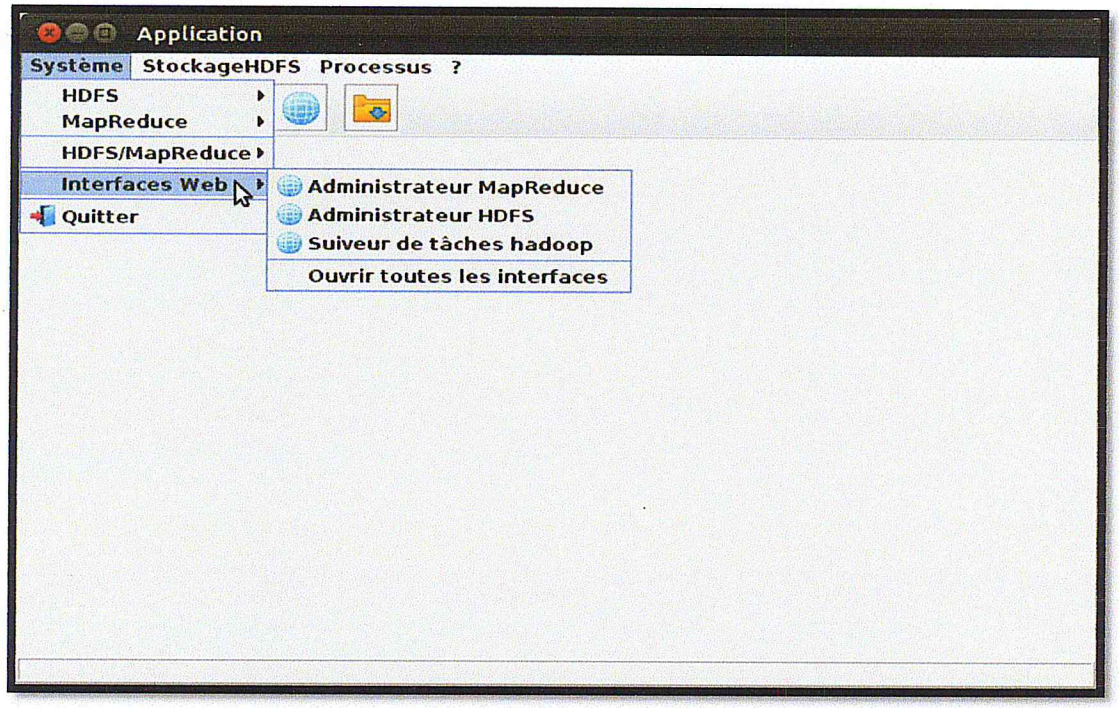


Figure 5.11. Ouverture des interfaces web de Hadoop

Comme illustre la figure 5.12, l'élément de menu « StockageHDFS » permet de visualiser les fichiers stockés dans l'HDFS, soit via l'interface web fournie par Hadoop (voir figure 5.13) ou encore par le biais de notre propre interface (voir figure 5.14).

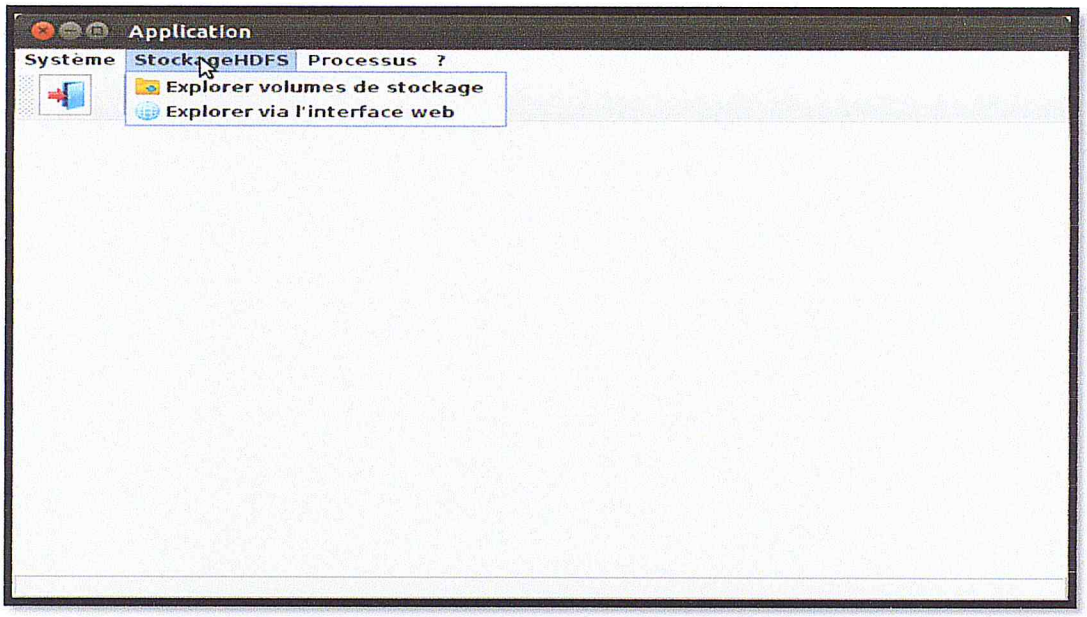


Figure 5.12. Stockage de l'HDFS

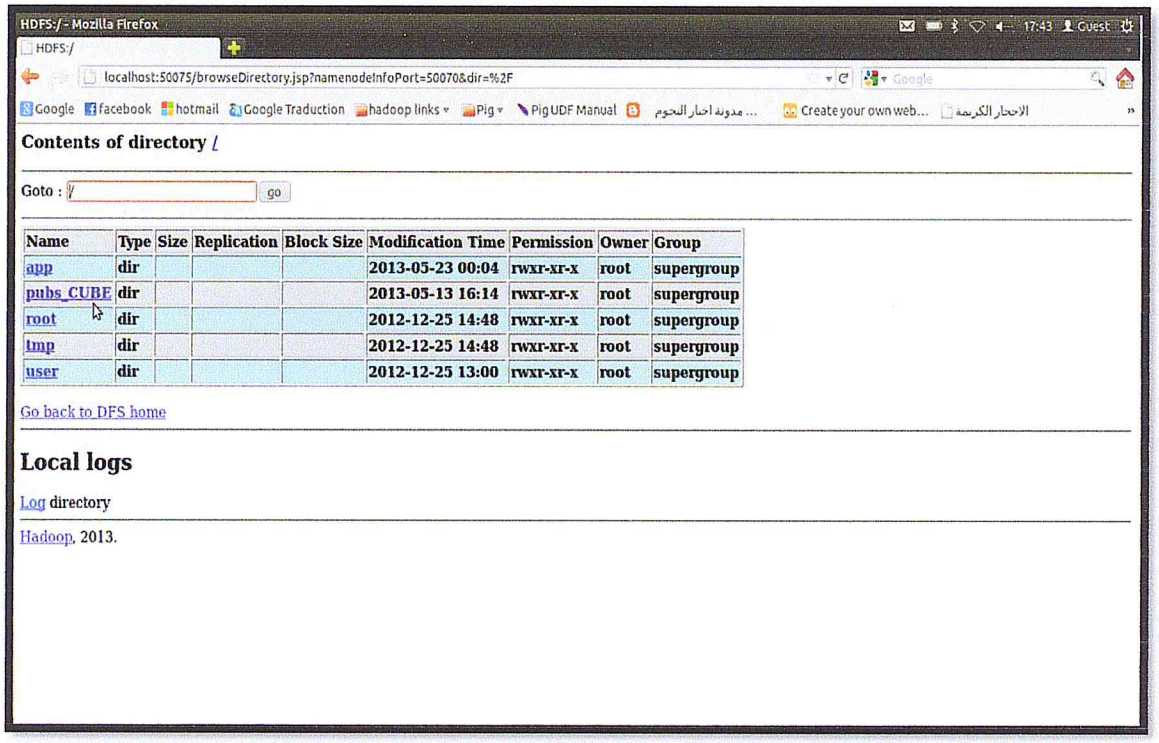


Figure 5.13. Stockage de l'HDFS via l'interface web fournie par Hadoop

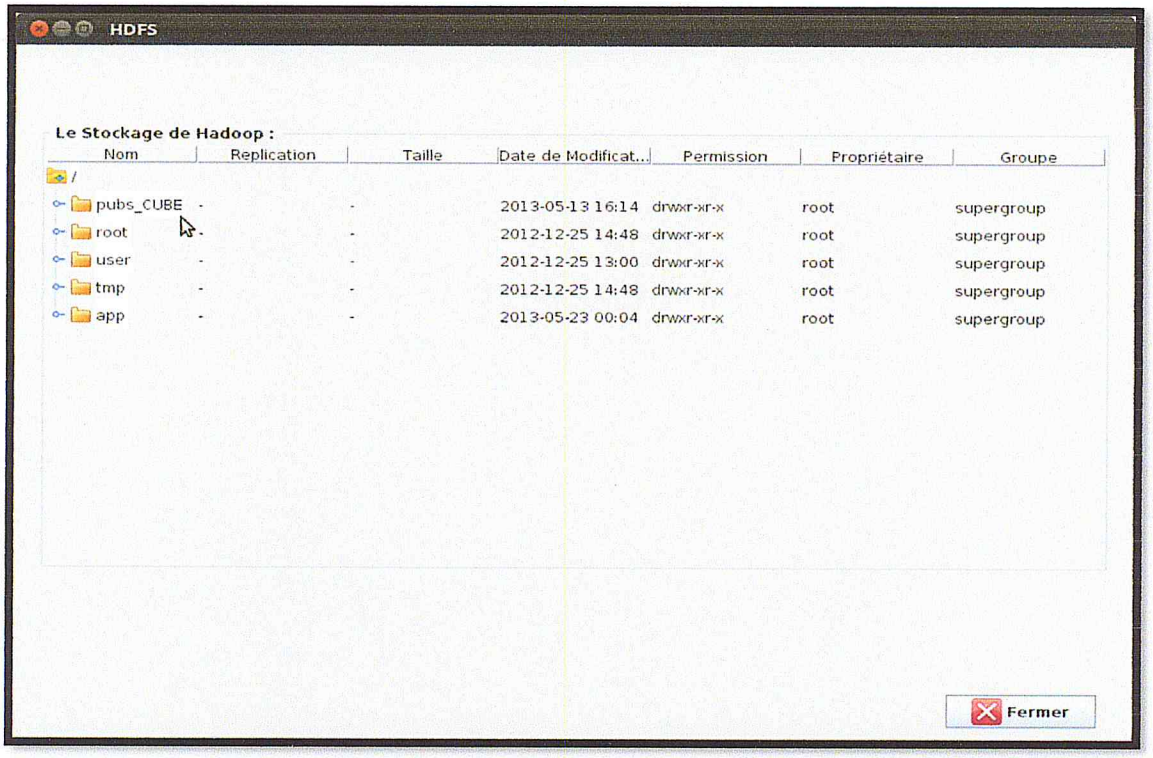


Figure 5.14. Stockage de l'HDFS par le biais de notre propre interface

6. Illustration de notre système

Afin d'illustrer le fonctionnement général de notre système, nous avons appliqué les fonctions développées sur le schéma montré dans la figure 5.15 dont le but est de calculer le chiffre d'affaire (CA) des ventes d'ouvrages selon les axes : auteur (au_id), titre (title_id), éditeur (pub_id), année (year), point de vente (store_id).

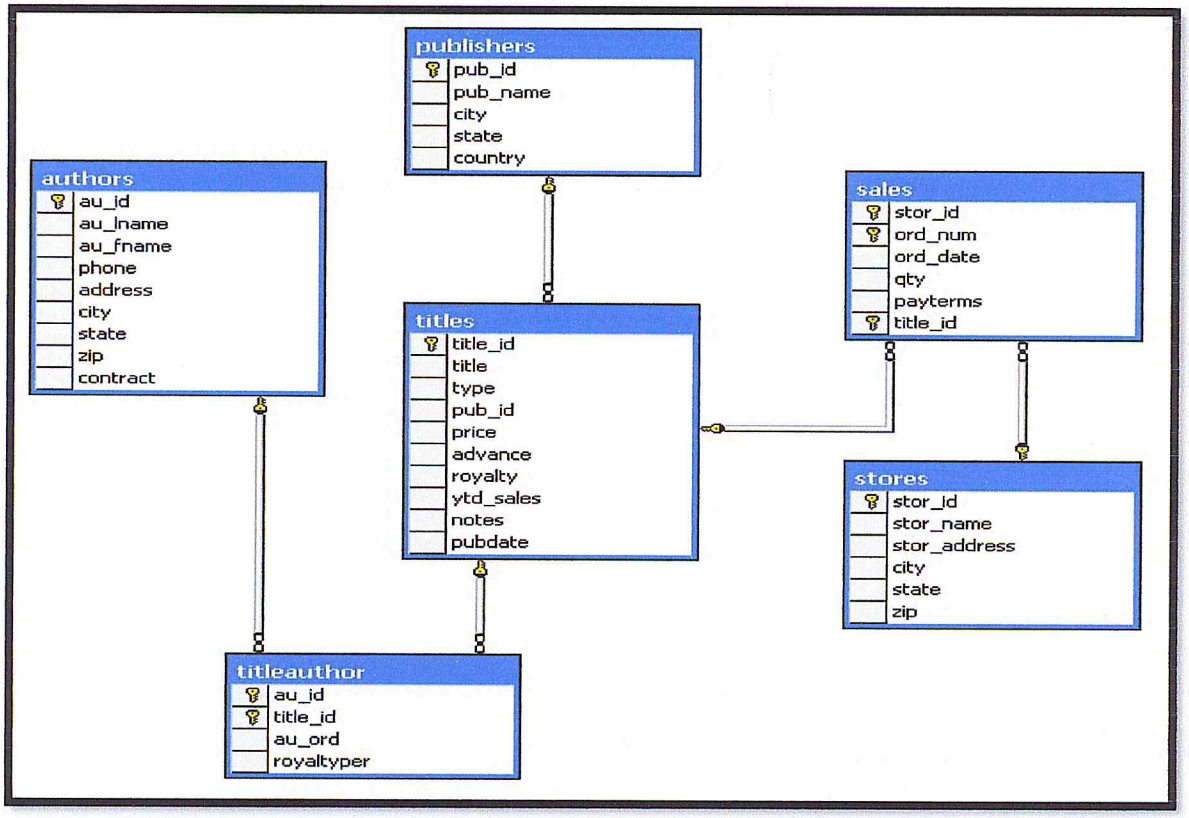


Figure 5.15 Schéma de la base de production Pubs

Le résultat après l'exécution de notre système est décrit par les figures 5.16 et 5.17. La première figure représente le schéma du cube de données et la deuxième montre le stockage distribué du cube de données.

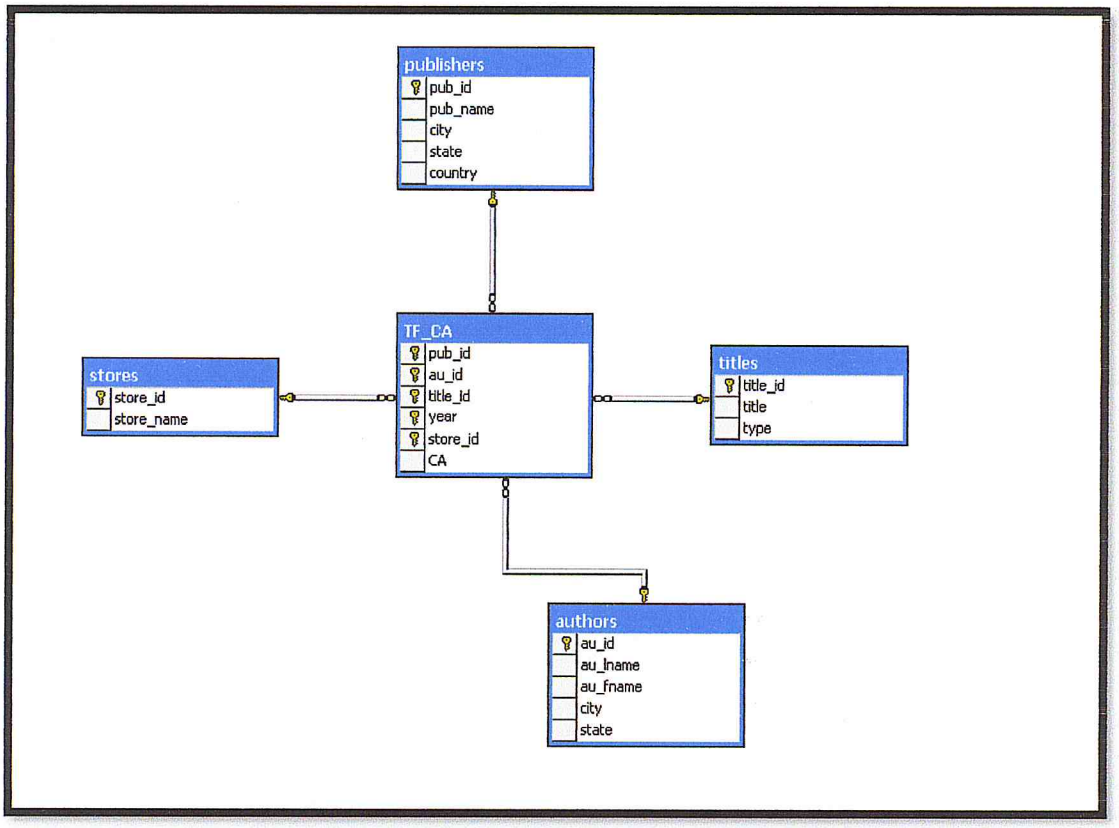


Figure 5.16. Schéma du cube de données

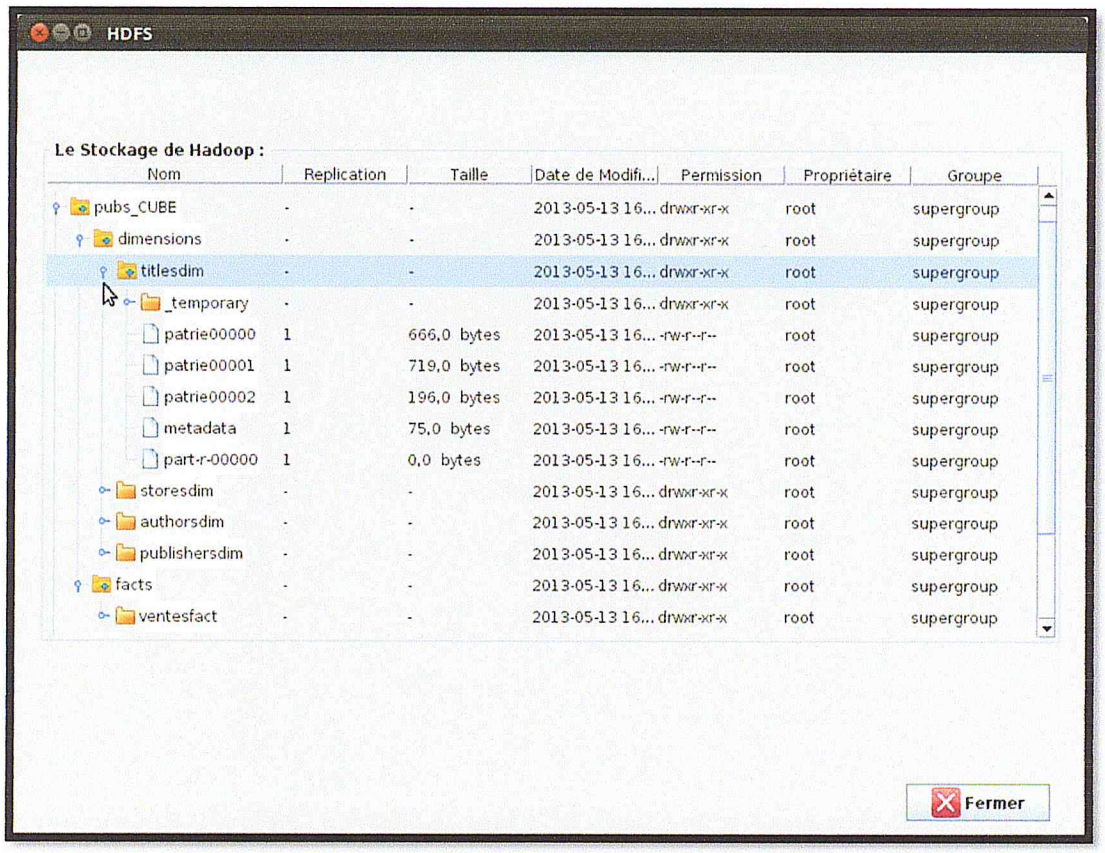


Figure 5.17. Stockage distribué du cube de données

Pour ce faire, nous allons présenter les étapes de manipulation des données depuis l'extraction jusqu'au chargement dans le cube de données en utilisant les fonctions développées (UDF). D'abord, pour pouvoir utiliser les UDFs sous Hadoop Pig, il suffit d'ajouter la commande suivante dans le script Pig :

```
register /root/PFE/MyUdfs.jar;
```

Script 1. Enregistrer le jar des UDFs développés

Ensuite, nous avons fait une extraction de données à partir de deux sources de données : MySQL et les fichiers plats de format CSV. Nous avons remarqué que l'extraction directe à partir des bases MySQL provoque un problème si l'application reste connectée à la base MySQL lors des transformations. Pour y remédier, nous avons utilisé l'UDF de conversion de format MySQL vers CSV, puis nous avons réalisé l'extraction à partir des fichiers transformés et des fichiers sources d'origine de format CSV, comme le montre le script 2 suivant :

```
-----------------------------------------------------------------------------
--transformation de format de mysql au csv
-----------------------------------------------------------------------------
authorsqltocsv= load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv ('localhost',
'pubs', 'authors', 'root', '13', '/root/PFE/tabPubsSql') as (au_id:chararray,
au_lname:chararray, au_fname:chararray, phone:chararray, address:chararray,
city:chararray, state:chararray, zip:chararray, contract:chararray);

publishersqltocsv = load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv ('localhost',
'pubs', 'publishers', 'root', '13', '/root/PFE/tabPubsSql') as (pub_id:chararray,
pub_name:chararray, city:chararray, state:chararray, country:chararray);

titlesqltocsv = load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv ('localhost',
'pubs', 'titles', 'root', '13', '/root/PFE/tabPubsSql') as (title_id:chararray, title:chararray,
type:chararray, pub_id:chararray, price:float, advance:float, royalty:int, ytd_sales:int,
notes:chararray, pubdate:chararray);

titleauthorsqltocsv = load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv
('localhost', 'pubs', 'titleauthor', 'root', '13', '/root/PFE/tabPubsSql') as (au_id:chararray,
title_id:chararray, au_ord:int, auyaltype:int);

storesqltocsv = load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv ('localhost',
'pubs', 'stores', 'root', '13', '/root/PFE/tabPubsSql') as (store_id:chararray,
store_name:chararray, store_address:chararray, city:chararray, state:chararray,
zip:chararray);

salesqltocsv = load '/root/PFE/vide' using org.myloaders.MySqlLoaderToCsv ('localhost',
'pubs', 'sales', 'root', '13', '/root/PFE/tabPubsSql') as (store_id:chararray, ord_num:chararray,
ord_date:chararray, qty:int, payterms:chararray, title_id:chararray);
```

```
-------------------------------------------------------------------------------
-- Chargement à partir des fichiers csv
-------------------------------------------------------------------------------

authorscsv = load '/root/PFE/tabPubsCsv/authors.csv' using PigStorage('\t') as
(au_id:chararray,au_lname:chararray,au_fname:chararray,phone:chararray,address:chararr
ay,city:chararray,state:chararray,zip:chararray,contract:chararray);

publisherscsv = load '/root/PFE/tabPubsCsv/publishers.csv' using PigStorage('\t')
as(pub_id:chararray, pub_name:chararray, city:chararray, state:chararray,
country:chararray);

titlescsv = load '/root/PFE/tabPubsCsv/titles.csv' using PigStorage('\t') as
(title_id:chararray,title:chararray, type:chararray, Tpub_id:chararray, price:float,
advance:float, royalty:int, ytd_sales:int, notes:chararray, pubdate:chararray);

titleauthorcsv = load '/root/PFE/tabPubsCsv/titleauthor.csv' using PigStorage('\t') as
(TAau_id:chararray, TAtitle_id:chararray, au_ord:int, auyaltype:int);

storescsv = load '/root/PFE/tabPubsCsv/stores.csv' using  PigStorage('\t') as
(store_id:chararray, store_name:chararray, store_address:chararray, city:chararray,
state:chararray, zip:chararray);

salescsv = load '/root/PFE/tabPubsCsv/sales.csv' using PigStorage('\t') as
(SALstore_id:chararray,ord_num:chararray,ord_date:chararray,qty:int,payterms:chararray,
SALtitle_id:chararray);

-------------------------------------------------------------------------------
--chargement a partir des fichiers transformés de mysql au csv
-------------------------------------------------------------------------------

authorsql = load '/root/PFE/tabPubsSql/authors.csv' using PigStorage('\t') as
(au_id:chararray,au_lname:chararray,au_fname:chararray,phone:chararray,address:chararr
ay,city:chararray,state:chararray,zip:chararray,contract:chararray);

publishersql = load '/root/PFE/tabPubsSql/publishers.csv' using PigStorage('\t') as
(pub_id:chararray, pub_name:chararray, city:chararray, state:chararray, country:chararray);

titlesql = load '/root/PFE/tabPubsSql/titles.csv' using PigStorage('\t') as (title_id:chararray,
title:chararray, type:chararray, Tpub_id:chararray, price:float,advance:float,
royalty:int,ytd_sales:int, notes:chararray, pubdate:chararray);

titleauthorsql = load '/root/PFE/tabPubsSql/titleauthor.csv' using PigStorage('\t') as
(TAau_id:chararray, TAtitle_id:chararray, au_ord:int, auyaltype:int);

storesql = load '/root/PFE/tabPubsSql/stores.csv' using  PigStorage('\t') as (store_id:chararray,
store_name:chararray, store_address:chararray, city:chararray, state:chararray,
zip:chararray);

salesql = load '/root/PFE/tabPubsSql/sales.csv' using PigStorage('\t') as
(SALstore_id:chararray, ord_num:chararray, ord_date:chararray, qty:int,
payterms:chararray, SALtitle_id:chararray);
```

Script 2 : Conversion et Extraction de données

Après le chargement des données a partir des deux sourcesPubs de type MySQL et Pubs de type CSV, l'UDF de conversion MySQLLoaderToCSV a permis d'avoir toutes les données dans un format unique de type CSV.

Il s'agit maintenant de fusionner les données de même nature distribuées sur les deux sources avec la clause UNION. La clause DISTINCT éliminera d'éventuels tuples doublons (voir script 3).

```
-----------------------------------------------------------------
--Union
-----------------------------------------------------------------
authors_u = union authorscsv, authorsql;
sales_u = union salescsv,salesql;
publishers_u = union publisherscsv, publishersql;
titles_u = union titlescsv,titlesql;
stores_u = union storescsv,storesql;
titleauthor_u = union titleauthorcsv, titleauthorsql;
-----------------------------------------------------------------
--Distinct
-----------------------------------------------------------------
titles      = distinct titles_u;
publishers  = distinct publishers_u;
authors     = distinct authors_u;

titleauthor = distinct titleauthor_u;
stores      = distinct stores_u;
sales       = distinct sales_u;
```

Script 3 : Les clauses UNION et DISTINCT

Maintenant que les données sont préparées, il ne reste qu'à sélectionner celles-ci pour l'alimentation des dimensions du cube de données et ce grâce à l'UDF « MyStorerDistributer ». Lors de la génération de chaque dimension, on sauvegarde les clés des dimensions dont on aura besoin au moment de l'application de l'intégrité référentielle (voir script 4).

```
-----------------------------------------------------------------------------------
--Alimentation des tables de dimensions avec UDF MyStorerDistributer
-----------------------------------------------------------------------------------
titlesdim = foreach titles generate title_id, title, type,price ;
store titlesdim into '/root/pubsCUBE/titlesdim' using
org.mystorers.MyStorerDistributer ('/root/pubsCUBE/titlesdim', '64', '1', 'part', '\t');

titlesdimKeys = foreach titlesdim generate title_id as title_pk;
```

```
authorsdim = FOREACH authors GENERATE au_id, CONCAT(CONCAT(au_lname,
' '), au_fname), city,state;


store authorsdim into '/root/pubsCUBE/authorsdim' using
org.mystorers.MyStorerDistributer ('/root/pubsCUBE/authorsdim', '64', '1', 'part',
'\t');
authorsdimKeys = foreach authorsdim generate au_id as author_pk;

publishersdim = foreach publishers generate pub_id,pub_name,city,state,country;
store publishersdim into '/root/pubsCUBE/publishersdim' using
org.mystorers.MyStorerDistributer('/root/pubsCUBE/publishersdim','64','1','part','\t');
publishersdimKeys = foreach publishersdim generate pub_id as publisher_pk;


storesdim = foreach stores generate store_id,store_name;
store storesdim into '/root/pubsCUBE/storesdim' using
org.mystorers.MyStorerDistributer('/root/pubsCUBE/storesdim','64','1','part','\t');
storesdimKeys = foreach storesdim generate store_id as store_pk;
```

Script 4. Génération et stockage des dimensions

Après que les dimensions soient stockées, nous faisons le traitement pour la table de fait «vente » par les jointures nécessaires pour la construction de cette dernière.

Selon les clés de la table de faits, on fait la jointure avec les clés des dimensions qu'on a sauvegardées. Comme ça, on filtre les tuples de faits et on ne garde que ceux qui ont des clés qui existent dans les dimensions. Puis après, on retire les clés des dimensions à partir du résultat de la jointure avec une projection et là on obtient nos tuples de faits référencés (voir script5).

```
-------------------------------------------------
--generation de la table de fait vente--
-------------------------------------------------

titlesjoinsales  = JOIN titles by title_id, sales by SALtitle_id;
joinstores = JOIN titlesjoinsales by SALstore_id,stores by store_id;
joinpublishers = JOIN joinstores by Tpub_id,publishers by pub_id;
jointitleauthor = JOIN joinpublishers by title_id,titleauthor by TAtitle_id;
joinauthors = JOIN jointitleauthor by TAau_id, authors by au_id;


ventes = FOREACH joinauthors GENERATE title_id, type, pub_id, au_id,
store_id,org.myevals.ExtractYear(ord_date) AS year, qty, qty*price AS partprice;

ventesgroups = GROUP ventes BY (title_id, type, pub_id, au_id, store_id, year);
```

```
ventesfact = FOREACH ventesgroups GENERATE FLATTEN(group), (int)
SUM($1.qty) AS totalqty, SUM($1.partprice) AS CA;

res1 = JOIN titlesdimKeys by title_pk,ventesfact by title_id ;
res2 = JOIN publishersdimKeys by publisher_pk, res1 by pub_id ;
res3 = JOIN storesdimKeys by store_pk,res2 by store_id;
res4 = JOIN authorsdimKeys by author_pk,res3 by au_id;

ventesfact_ref = foreach res4 generate title_id, type, pub_id, au_id, store_id, year,
totalqty, CA;

store ventesfact_ref into '/root/pubsCUBE/ventefact' using
org.mystorers.MyStorerDistributer('/root/pubsCUBE/ventefact','64','1','part','\t');
```

Script 5. Génération et stockage de table de fait

7. Traitement de modèle MapReduce

Hadoop Pig exécute les séquences Map et Reduce d'une manière transparente à l'utilisateur, et pour donner une idée de la compilation de Pig en MapReduce, nous avons sélectionné un bon exemple qui consiste à trouver les 10 premiers pages les plus visitées dans chaque catégorie. Ci-dessous nous montrons le script et sa compilation en MapReduce :

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);
store topUrls into '/data/topUrls';
```

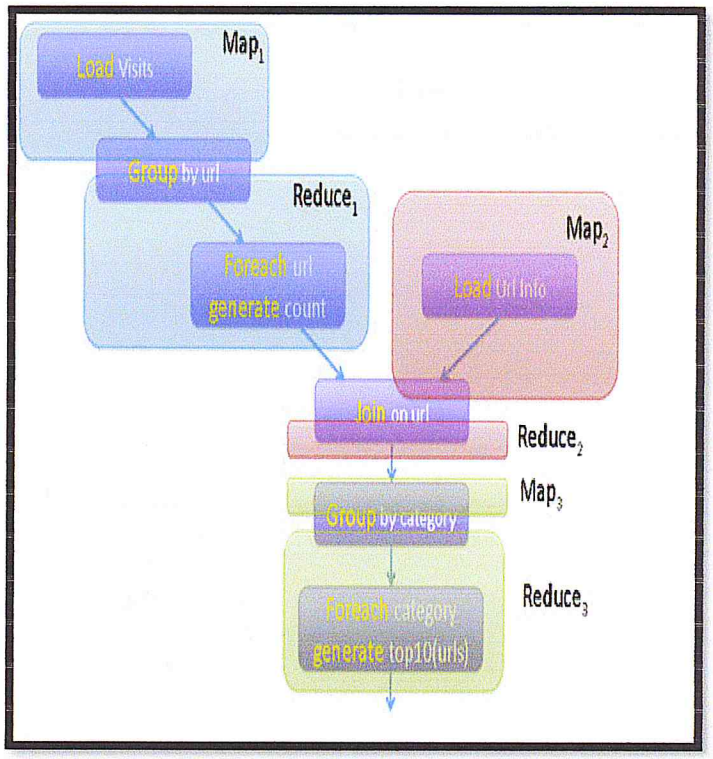


Figure 5.18. Traitement de modèle MapReduce

[Olston et al. 2008].

8. Conclusion

Au fil de ce dernier maillon de notre travail, nous avons présenté l'implémentation de notre système. Nous avons d'abord spécifié nos besoins logiciels et matériels ; puis, nous avons décrit la configuration utilisée pour l'implémentation de notre système. Dans la partie implémentation, nous avons présenté toutes les fonctions implémentées dans notre système ainsi que les différentes fonctionnalités de notre application.

Conclusion Générale et perspectives

Conclusion générale et perspectives

Les systèmes décisionnels sont un ensemble de technologies destinées pour permettre aux décideurs de comprendre les données de pilotage plus rapidement, de telle sorte qu'ils prennent les meilleures décisions au moment opportun. La chaine décisionnelle est constituée de trois phases à savoir (1) l'alimentation du Datawarehouse (2) la modélisation du datawarehouse et (3) la restitution des données (analyse en ligne).

L'alimentation de l'entrepôt de données est considérée comme étant la phase la plus critique dans la chaine décisionnelle à cause de la diversité des sources de production, de leur hétérogénéité et de leur volumétrie. Cette phase est assurée par le processus ETL. Les systèmes ETL (Extraction-Transformation-Loading) sont une catégorie d'outils ayant pour tâches l'extraction des données sources, leur homogénéisation, nettoyage, filtrage, etc. et enfin leur chargement dans l'ED.

Le travail réalisé dans ce mémoire consiste à la mise en œuvre d'un processus ETL sous un environnement parallèle et distribué à savoir Hadoop Pig. Pour ce faire, nous avons commencé par la littérature sur les systèmes décisionnels et particulièrement sur les processus ETL. Nous avons enchainé par une présentation du modèle de parallélisation et de distribution des traitements intensifs appelé MapReduce où nous avons étudié aussi l'environnement Apache Hadoop et le module Hadoop Pig avec son langage associé Pig Latin.

Dans la partie mise en œuvre, nous avons implémenté des méthodes qui permettent l'extraction et le partitionnement des données à partir de deux sources distinctes (fichiers CSV et des bases de données MySQL). Ces méthodes permettent aussi d'effectuer des transformations sur les données avant de les charger dans un cube de données distribué.

Les expérimentations pour ce travail nécessitent une infrastructure de type cluster. Malheureusement, à défaut d'une telle plateforme, nous avons testé sur notre machine (LapTop) sur des quantités de données ne dépassant pas 1 GO. Les résultats montrent des performances linéaires pour l'ETL dans un modèle MapReduce

Comme tout projet, celui-ci mérite des améliorations et une continuité que nous pourrons résumer comme suit :

Notre application représente une API java Pig destinée aux développeurs de solutions ETL qui vont pouvoir faire appel à cette API dans leurs codes sources. Par contre, la création d'une interface utilisateur peut être une perspective intéressante pour permettre à un simple usager (non-informaticien) de profiter du système en effectuant le paramétrage de l'ETL grâce à des assistants visuels.

Les expérimentations ont été réalisées sur une seule JVM. Ces tests doivent se faire dans un cluster avec des tailles d'une dimension Giga-Bytes, Téra-Bytes voire Péta-Bytes ; ce qui permettra d'évaluer sérieusement les performances ETL dans un environnement MapReduce par rapport à un environnement classique.

Enrichir davantage l'environnement Hadoop Pig :

1. Intégration d'autres sources de données comme documents XML, fichier Excel, base de données oracle, etc.
2. Enrichissement de l'environnement par des transformateurs de données (fonctions de type chaine, fonction de conversion de type de données, projection, etc.).
3. Utilisation d'un autre modèle de données au niveau Hadoop pour le stockage éventuel d'un entrepôt de données d'une dimension importante (Téra-Bytes, Péta-Bytes, Hexa-Bytes, etc.). Pour ce faire, il faut envisager l'utilisation des SGBD de type NoSQL (tel que Hbase).

Annexe A

Hadoop Distributed File System (HDFS)

1. Hadoop Distributed File system

1.1.Définition

Un système de fichiers distribué stocke des données dans des machines différentes. C'est un composant Hadoop qui permet de gérer des donnés sous forme des fichiers de taille de 64Mo. HDFS permet de gérer des fichiers de taille importante (plusieurs giga-octets), il permet ainsi de fonctionner sur un cluster Hadoop. Ces fichiers sont rarement supprimés et réécrits. Ces derniers étant de très grande taille, ils sont découpés en blocs de 64 Mo stockés sur différentes machines [W05].

Blocs : La taille d'un bloc est habituellement 64Mo, mais cette taille peut être modifiée dans certain cas. Un bloc est stocké sous la forme d'un simple fichier Linux. Le nombre moyen de copies d'un fichier est de 3, les différentes copies sont stockées dans des nœuds différents [W05].

Nœud : les nœuds du système sont de deux types différents : le NameNode, DataNode.

1.2.Architecture de l'HDFS

HDFS possède une architecture de type maître/esclave (master/slave). Pour un cluster, il existe un unique Namenode (maître) qui supervise l'espace de nommage et reçoit les requêtes de consultation des données. Chaque espace de stockage est géré par un Datanode (esclave). Les fichiers de base de données étant volumineux, donc ils sont découpés en blocs de taille fixe de 64MB (sauf le dernier) et les stockés dans plusieurs Datanodes. Le Namenode exécute les opérations d'ouverture, de fermeture et de changement de nom de fichier, répertoire, etc. Il s'occupe de la cartographie des blocs des Datanodes en gérant toutes les métas-données. Les Datanodes se chargent des requêtes de lecture et d'écriture. Ils ont également, comme des tâches, les opérations de création, réplication ou suppression de fichiers. Les clients communiquent ainsi à la fois au Namenode et au Datanode. Les différents nœuds qui jouent le rôle de maître ou des esclaves doivent exécuter java pour démarrer leur Namenode ou leur Datanodes [W05].

La figure ci-dessous présente l'architecture générale de L'HDFS

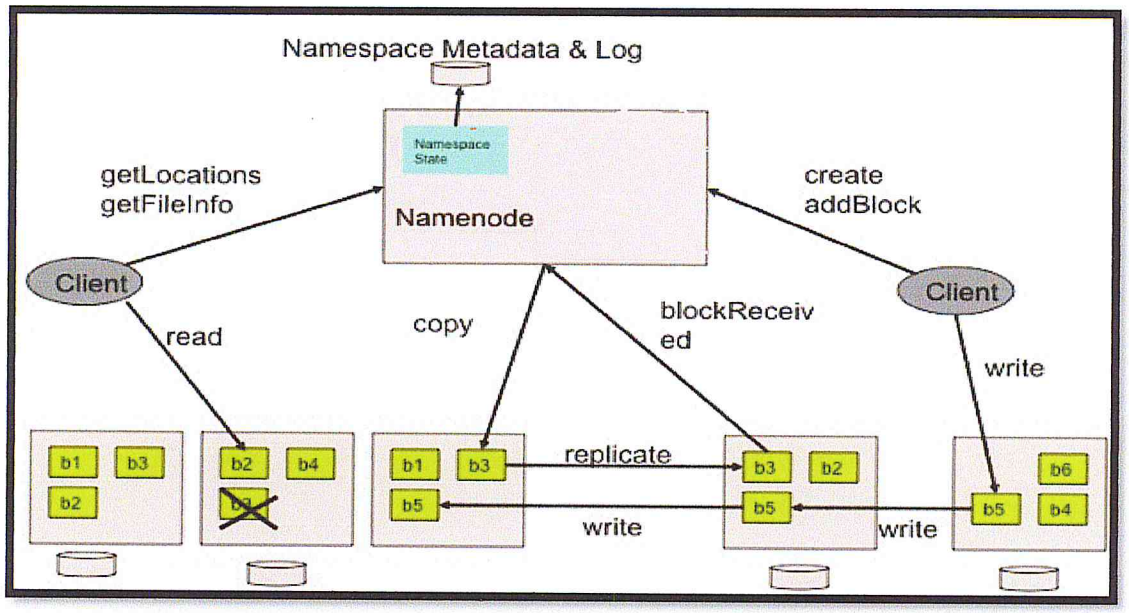


Figure A.2. Architecture de l'HDFS [W05].

1.1.Requête de lecture dans l'HDFS

Un client est connecté directement au Datanodes pour lire leurs données et il est guidé par le Namenode qui lui donne le bon Datanode de chaque bloc. Nous allons expliquer le déroulement de la requête de lecture de données à partir l'HDFS dans les étapes suivantes :

1. Le client œuvre le fichier souhaité à lire par l'appel de la fonction **open ()** de l'objet FileSystem, qui est une instance de DistributedFileSystem.
2. DistributedFileSystem appel le Namenode en utilisant le RPC afin de déterminer l'emplacement des blocs associés au fichier à lire. Pour chaque bloc, le Namenode retourne les adresses des datanodes qui ont une copie de ce bloc. DistributedFileSystem retourne aussi un FSDataInputstream (unité de sortie supporte les fichiers) au client pour lire les données à partir d'elle.
3. Le client appel la fonction **Read ()** sur le stream (flux). DFSInputStream, qui a enregistré les adresses de datanodes pour les premiers blocs dans un fichier, puis se connecte à la première DataNode pour le premier bloc dans le fichier
4. Les données sont transmises en continu à partir de la DataNode vers le client, qui appelle read () à plusieurs reprises sur le flux.
5. Quand il aboutit à la fin de bloc, DFSInputStream ferme la connexion de Namenode, ensuite trouve le bon Namenode pour le prochain bloc.
6. Quand le client termine la lecture, il appelle la fonction **close ()** dans le FSDataInputStream.

La figure ci-dessous rassemble toutes les étapes expliquées dessus :

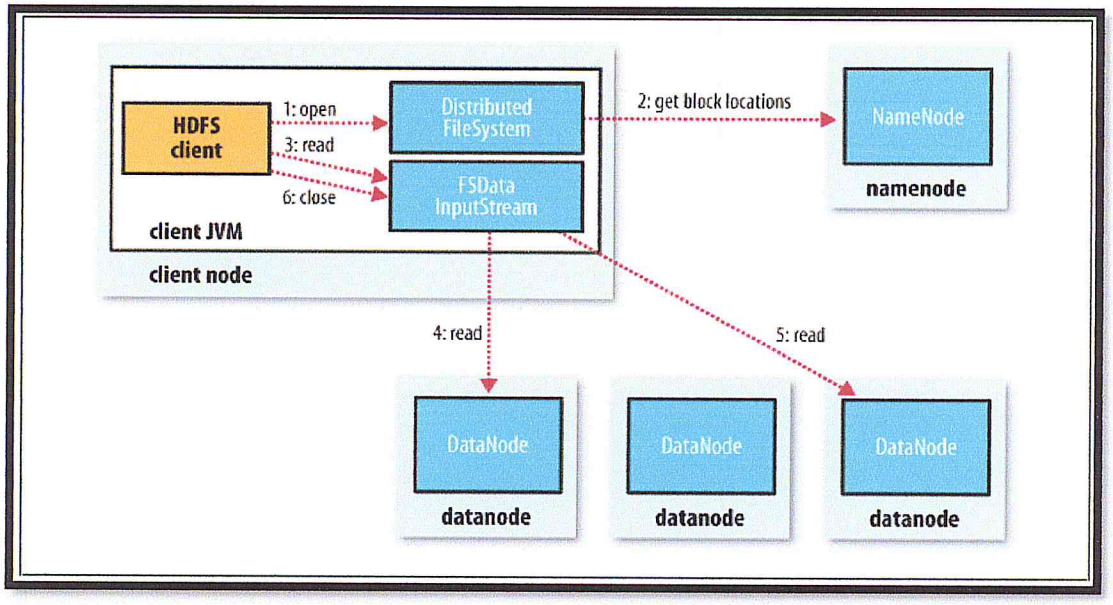


Figure A.3. Un client lance une requête de lecture de données d'HDFS [White 2009].

1.2. Requête d'écriture dans l'HDFS

Un client est aussi contacté au DataNodes pour écrire les données. Nous allons expliquer le déroulement de la requête d'écriture de données dans l'HDFS par les étapes suivantes :

1. Le client crée le fichier par l'appel de create () sur DistributedFileSystem.
2. DistributedFileSystem éffectue un appel RPC au NameNode pour créer un nouveau fichier déposé dans l'espace de noms du système de fichiers, sans les blocs qui sont lui associés.
3. Dis que le client écrit des données, le DFSOutputStream les divise en paquets, qui seront placés dans une file d'attente interne, dite file d'attente des données. Cette dernière est utilisée par le Data Streamer qui a comme responsabilité de demander au NameNode d'allouer des nouveaux blocs en choisissant une liste de DataNodes appropriée pour stocker les doublons (replicats).
4. DataStreamer stream les paquets vers la DataNode d'abord dans le pipeline, qui stocke le paquet et l'envoie au deuxième DataNode dans le pipeline. De même, le deuxième DataNode stocke le paquet et le transmet à la troisième (et dernière DataNode dans le pipeline).

5. DFSOutputStream maintient également une file d'attente interne de paquets qui sont en attente d'être reconnus par DataNodes, appelée file d'attente ACK. Un paquet est supprimé de la file d'attente ACK seulement quand il sera reconnu par toutes les datanodes de le pipeline.

6. Lorsque le client termine l'écriture de données, il appelle close () sur le flux.

7. Cette action tire tous les paquets restants à la conduite DataNode et attend l'accusé avant de contacter le NameNode pour signaler que le dossier est complet.

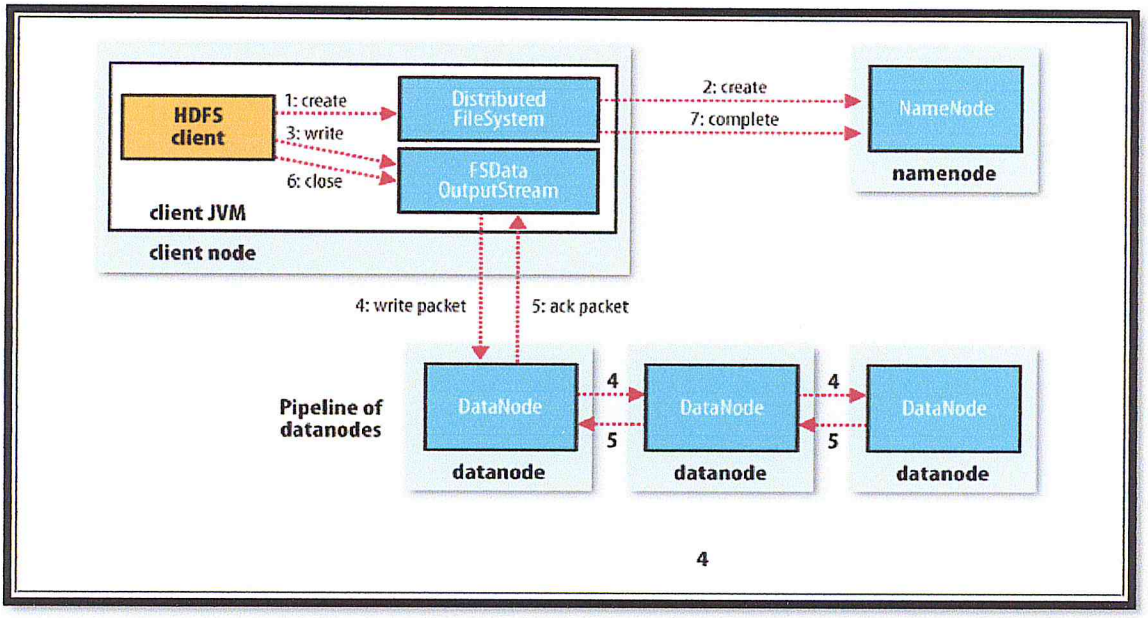


Figure A.4. Un client lance une requête d'écriture de données dans l'HDFS [White 2009].

1.5. Les caractéristiques de l'HDFS

Ce système est conçu pour fonctionner sur un matériel standard, il est destiné aux architectures comportant de nombreux nœuds ; donc Une installation HDFS étant parfois composée de milliers de serveurs, la probabilité d'apparition de panne hardware est très forte pour cela l'HDFS doit détecter ces erreurs rapidement et automatiquement. Les fichiers dans l'HDFS peuvent être répartis sur plusieurs (nœuds) machines, et peuvent être aussi divisés en plusieurs morceaux (blocs). Ces blocs sont répliqués pour éviter la perte des données en cas de panne (les réplicas sont conservés dans d'autres nœuds), et aussi pour équilibrer la charge des disques des nœuds, et puisque l'HDFS est adapté aux applications qui traitent de très grosses quantités de données donc il fournit un accès haut débit à ces données [W05].

Annexe B

Hadoop Pig et Langage Pig Latin

1. Framework Pig

1.1. Histoire de Pig

Pig commence comme un projet de recherche de Yahoo!, les chercheurs ont trouvé que le MapReduce présenter par Hadoop à un bas niveau, et très rigide, le code d'utilisateurs est difficile à maintenir et de réutiliser. En même temps, ils ont observé que les utilisateurs de MapReduce ne sont pas à l'aise avec les langages déclaratifs comme SQL. Ainsi, ils ont décidé de produire un nouveau langage qui permet de créer un compromis entre le langage déclaratif de type SQL et le style procédural bas niveau de MapReduce. L'utilisateur Hadoop commence d'adopter le Pig, un group d'ingénieurs de développement était assemblés pour décide de prendre le prototype de recherche et de l'intégrer dans une production d'un produit de qualité. Au même temps dans 2007, le Pig est était open source via apache, la première sortie de Pig est en septembre 2008, ensuite le Pig est gradué par Incubateur et devient un subprojet de apache Hadoop. En 2009 une autre compagnie utilise le Pig pour le traitement de leurs données Comme Amazon ajoute le Pig pour leur propre élastique service de MapReduce. En 2010, le Pig continue l'adoption, augmente et gradue jusqu'à devenir le niveau le plus haut de projet Apache [W04].

1.2 Définition

Pig est une plate-forme pour l'analyse de grands ensembles de données qui est spécifiquement ciblé pour les programmeurs procéduraux utilisant le MapReduce. Pig et inspiré par Yahoo !

La structure de Pig est divisée en deux pièces [White 2009] :

1. Le langage utilisé pour expresse les flux de données, nommé PigLatin.
2. L'environnement d'exécution (Compilateur) pour exécuter les programmes Pig. Actuellement, il ya deux environnements : exécution locale dans un seul JVM (java Virtual Machine) et exécution distribuée dans un Hadoop cluster.

3. Langage PigLatin

2.1. Présentation

Le Pig latin est la version procédurale de SQL. Ce langage offre une manipulation de données à haut niveau dans un style procédural. Les cas d'utilisation de PigLatin soignés de tomber sur trois grandes catégories : Extract Transform Load (ETL), recherche la ligne de donnée, le traitement itérative.

1.1.1. Modèle de données

Pig utilise trois catégories de données scalaire, texte et complexe :

La table ci-dessous résume les trois catégories de données dans le langage PigLatin [White 2009].

Catégorie	Type	Description	Exemple littérale
Numérique	Int	32-bit signed integer	1
	Long	64-bit signed integer	1L
	Float	32-bit floating-point number	1.0F
	Double	64-bit floating-point number	1.0
Texte	Chararray	Character array in UTF-16 format	'a'
Complexe	Tuple	lise d'élément de données non ordonné où l'élément est de type quelconque	(1, 'pomegranate')
	Bag	une collection de tupie	{(1,'pomegranate'),(2)}
	Map	ensemble de paires clés valeur, telque la clé est de type atomique, et la valeur peut prendre différentes types	[1#'pomegranate']

Tableau B.1. Type de données dans le PigLatin

2.1.2. Opérateurs relationnel de Pig

Le système Pig fournit aux utilisateurs de PigLatin des opérateurs relationnels pour lire les données et les manipuler, ensuite charger le résultat dans un fichier ou parfois l'afficher sur l'écran.

Le tableau ci-dessous résume les commandes usuelles dans un Programme PigLatin ainsi que leurs fonctions dans ce programme [White 2009] :

Opérateurs	Fonction
Load	Lire les données à partir de fichiers systèmes.
Store	Ecrire les données dans le fichier système.

Foreach... generate	Applique une expression pour chaque enregistrement et génère une ou plusieurs enregistrements.
Filter	Applique un prédicat (restriction) et enlève les enregistrements qui ne retournent pas vrai.
Group/Cogroup	Collecte des enregistrements de même clé à partir d'un ou de plusieurs entrées.
Join	Joint deux ou plusieurs entrées en basant sur la clé.
Order	Trie une ou plusieurs enregistrements en basant sur la clé.
Distinct	Enlève les enregistrements doubles.
Union	Fusionne deux ensembles de données.
Dump	Ecrire la sortie dans l'écran.
Limit	Limite le nombre d'enregistrement.

Tableau B.2 .Opérateurs usuelles dans le PigLatin

2.1.3. Expressions dans le PigLatin

Une expression est une chose qui évolue pour produire une valeur. L'expression peut être utilisée dans le pig comme une partie de statements contient un opérateur relationnel.

Pig est riche en expression, la plupart sont aplanies pour les autres langages de programmation.

Le tableau ci-dessous liste les expressions avec une brève description et exemple [White 2009 :

Catégorie	Expressions	Description	Exemple
Constante	Littéral	Valeur constante	1.0, 'a'
Champ (par position)	$n	Champ dans la position n	$0
Champ (par nom)	F	champ nommé f	Année
Projection	c.$n, c.f	Champ dans le conteneur c (relation, bag, or tuple) par position, par nom	records.$0, records.year
Map lookup	m#k	Valeur associé par une clé k dans le map m	Items #'Coat'
Cast	(t) f	Cast de champ f de type t	(int) year
Arithmétique	x + y, x - y x * y, x / y x % y	Addition, soustraction Multiplication, division Modulo, le reste de	$1+ $2, $1 - $2 $1 * $2, $1 / $2 $1 % $2

	+x, -x	division de x sur y Unary positive, negation	+1, –1
Conditionnel	x ? y : z	Bincond/ternaire, return y si x est true, z sinon	quality == 0 ? 0 : 1
Comparaison	x == y, x != y x > y, x < y x >= y, x <= y x matches y	Egal, non égal Grand que, moins que Grand que ou égal, moins égal que Pattern matching avec expression régulière	quality == 0, temperature != 99 quality > 0, quality < 10 quality >= 1, quality <= 9 quality matches '[01459]'
Boolean	x is null x is not null x or y x and y non x	Est null Est non null OR logique AND logique Negation	température est null température est non null q == 0 or q == 1 q == 0 and r == 0 non q
Fonctionnel	fn(f1,f2,...)	Invocation de fonction fn pour fields f1, f2,...	isGood(quality)

Tableau B.3 .Expressions dans PigLatin

2.1.4. Fonction définie par l'utilisateur (UDF)

Le Pig est un Open Source java. Les fondateurs (concepteurs) de Pig trouvent que la modification ou la maintenance d'un code est très rédige, donc ils ont pensé de donner la main aux utilisateurs de définir leurs propres fonctions « UDF » ou « Fonction Définie par Utilisateurs ». Ces fonctions sont écrites en java en utilisant des types de données simples. Pour intégrer ces UDFs dans le programme Pig, il faut les convertir au format de JAR. Pig fournit deux statements **Register** et **Define**, pour être possible d'incorporâtes ces UDFs [White 2009].

- **Register :** permet d'enregistrer le JAR avec le runtime de Pig.
- **Define :** permet decréer un lien pour l'UDF.

Pig supporte deux catégories principales d'UDFs : **Eval et Load/Store function**,

- **EvalFunction :** la plupart des UDFs sont de type Eval, ce sont des fonctions d'agrégation comme Max,Count,Sum, etc.

L'utilisateur peut spécifier sa fonction appelée Filterfunction. Cette fonction est de type spécial d'Evalfunction, utilisé dans l'opérateur FILTER pour éliminer des tuples non demandés, ou bien dans le cas d'utilisation des conditions booléennes comme « IsEmpty ».

Le tableau ci-dessous liste les méthodes essentielles de Pig pour construire une Eval Function ou Store/Load Function [W07] :

Type de fonction	Méthodes	Type de retour	Rôle
LoadFun	getInputFormat()	InputFormat	Cette méthode sera appelée par Pig pour obtenir l' inputFormat utilisé par le chargeur (Loader).
	getNext()	Tuple	Cette méthode appelle le Pig Runtime pour obtenir le prochain tuple dans les données.
	prepareToRead()	/	Grâce à cette méthode, le RecordReader associé à l'inputFormat fourni par le LoadFunc est passé à la LoadFunc. Le RecordReader peut ensuite être utilisé par la mise en œuvre de getNext () pour retourner un tuple représentant un enregistrement de données au Pig.
	setLocation()	/	Cette méthode est appelée par Pig pour communiquer l'emplacement de la charge (load) au chargeur.
StorFun	prepareToWrite ()		Dans prepareToWrite (), le RecordWriter associé à la OutputFormat fourni par l'StoreFunc est passé à la StoreFunc
	getOutputFo-rmat()	OutputFormat	Cette méthode sera appelée par Pig pour obtenir l'OutputFormat utilisé par le stockeur (storer).

	setStoreLocation()		Cette méthode est appelée par Pig pour communiquer l'emplacement du magasin à l'entreposeur
	putNext()	Tuple	Cette méthode est appelée par Runtime de Pig pour écrire le prochain tuple de données.
EvalFun	exec()	Tuple	Cette fonction est appelée à chaque tuple d'entrée. L'entrée de cette fonction est un tuple contenant les paramètres d'entrée dans l'ordre où ils sont passés à la fonction dans le script de Pig.

Tableau B.4 .Méthodes de Pig pour la manipulation des UDFs.

2.2. Environnement d'exécution de Pig

Pig est exécuté dans la part client d'une application, pour compiler un programme PigLatin, l'utilisateur peut utiliser deux modes d'exécutions [White 2009] :

2.2.1. Mode local

Dans ce mode, Pig est exécuté dans une seule JVM et accédé aux systèmes de fichiers locaux .Ce mode est adaptable lorsqu'on traite des données de petite taille.

- **JVM (Java Virtual Machine)**

La machine virtuelle Java ou JVM est un environnement d'exécution pour des applications Java. C'est l'un des éléments les plus importants de la plate-forme Java. Elle assure l'indépendance du matériel et du système d'exploitation lors de l'exécution des applications Java. Une application Java ne s'exécute pas directement dans le système d'exploitation mais dans une machine virtuelle qui s'exécute dans le système d'exploitation et propose une couche d'abstraction entre l'application Java et ce système [W08].

2.2.2. Mode Hadoop

Pig transforme les requêtes d'un script en jobs MapReduce et les exécute on Hadoop cluster. Ce mode est utilisable dans le cas de BigData.

2.3. Méthodes d'exécution

Pig fournit trois méthodes différentes pour exécuter un programme PigLatin. Ces modes sont considérés comme des modes d'interaction pour l'utilisateur, dans lequel ces méthodes sont adaptables dans les deux modes d'exécution local et Hadoop [Olston et *al.* 2008] :

2.3.1. Script (Batch)

Dans ce mode, l'utilisateur crée un fichier exécutable d'extension (.pig). Ce fichier contient une série de commandes qui commence toujours par la commande **Load** et se termine par la commande de stockage **Store**.

2.3.2. Grunt

C'est un interactif Shell pour l'exécution des commandes Pig où l'utilisateur écrit les commandes une par une dans la console, mais le plan de compilation et d'exécution est déclenché quand il appelle la commande Store.

2.3.3. Embedded

L'utilisateur peut exécuter des programmes Pig en utilisant java, comme l'utilisation d'un SGBD pour exécuter SQL dans les programmes java.

2.4. Compilation de Pig en MapReduce

Le système Pig prend comme entrée un programme Pig Latin ensuite il le compile dans un ou plusieurs jobs MapReduce. Puis, il exécute ces jobs dans un Hadoop cluster, mais avant de l'exécuter en MapReduce, ce programme passe par une série d'étapes de transformation.

La figure ci-dessous démontre les étapes à l'aide d'un exemple de programme PigLatin :

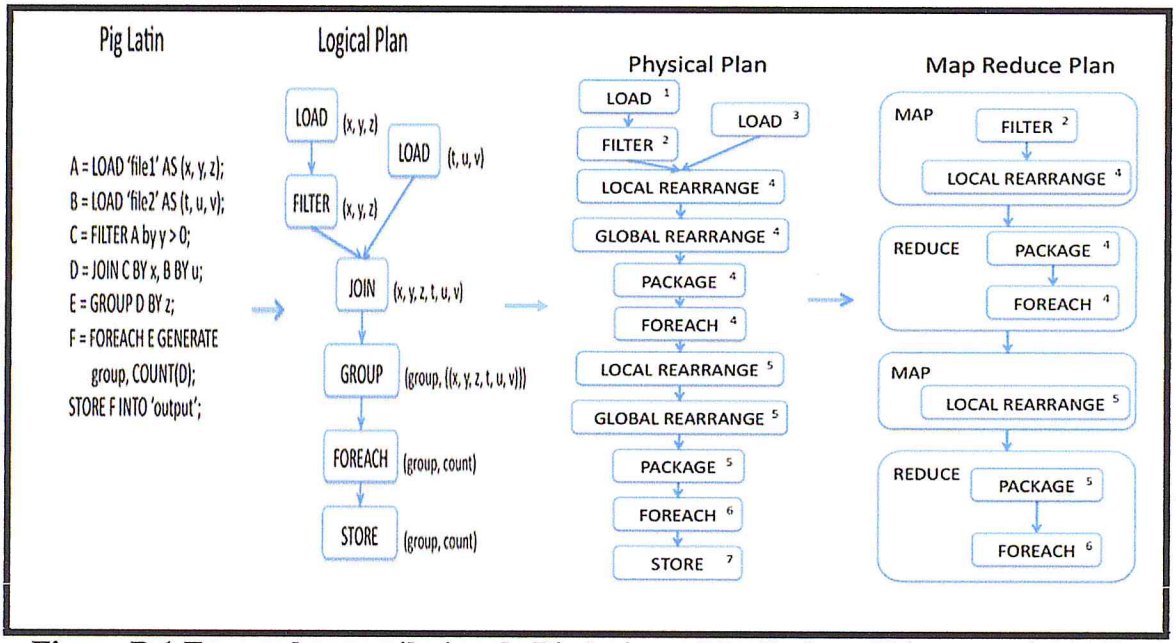


Figure B.1.Etapes de compilation de Pig latin en MapReduce [Olston et *al.* 2008]

Comme la figure B.1 présente, la compilation d'un programme Pig en MapReduce passe par trois étapes qui sont les suivantes :

1. Pig Latin vers Plan logique

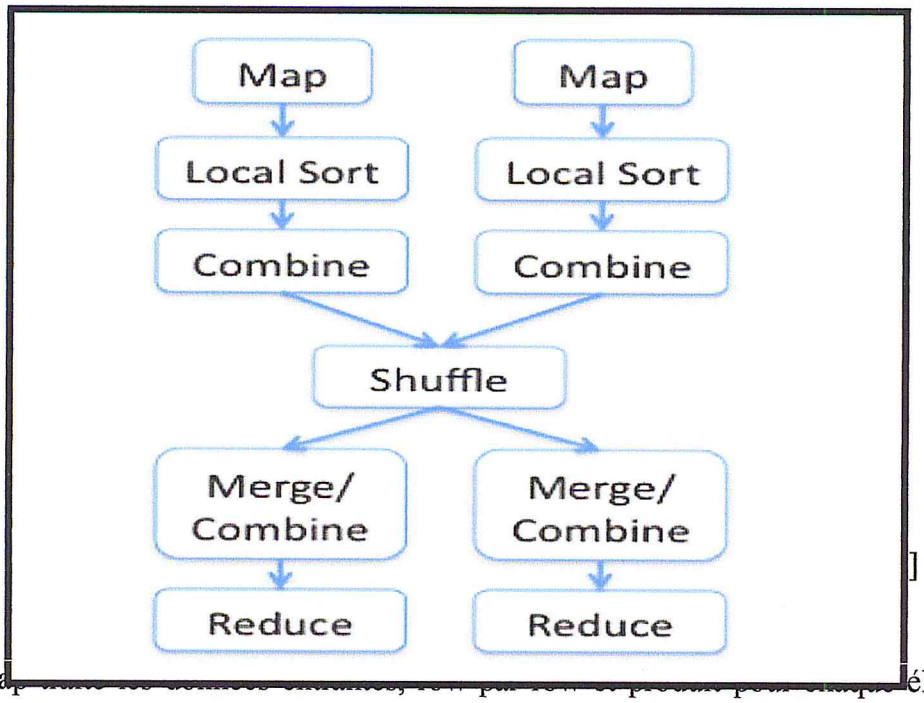
L'entrée de cette étape est un programme pig Latin, le système Pig vérifie la syntaxe de ce programme comme la vérification de la référence de toutes les variables, la capacité d'instanciation des classes correspondantes aux UDFs, etc. La sortie de cette étape est un plan logique avec une correspondance un par un de script PigLatin, ce plan est structuré sous format de graphe acyclique direct (DAG) où les nœuds représentent les opérateurs qui traitent les données et les arcs sont les données à traiter.

2. Plan logique vers Plan physique

Comme démontre la figure B.1 les opérateurs dans le plan logique sont présentés par identificateur (id). Ces opérateurs sont transformés en opérateurs de plan physique, certains opérateurs logiques comme LOAD, STORE, FOREACH restent les mêmes. Par contre, le GROUP, JOIN sont transformés en série de trois opérateurs physiques : local rearrange, global rearrange, package. La combinaison entre le local et global rearrange résulte des tuples de même clé et qui sont exécutés dans la même machine. L'opérateur package place les tuples adjacents qui ont la même clé dans un seul tuple.

3. Plan physique vers Plan MapReduce

Une fois le plan physique est construit, Pig assigne les opérateurs physiques vers les épates Hadoop qui sont les étapes d'exécution de modèle MapReduce de Hadoop, la figure ci-dessous démontre ces étapes :



1. Ma élément de row une clé.
2. Local Sort ordonne les données produites par chaque machine Map par clé.
3. Ensuite, ces données ordonnées passent par une étape optionnelle (Combiner) pour une agrégation partiale par clé.
4. Le Shuffle redistribue les données aux machines pour réaliser une organisation globale de données ordonnées par clé.
5. Toutes les données envoyées vers une machine particulière sont combinées dans une seule pièce dans l'étape Merge.
6. Finalement, l'étape reduce traite les données associées avec clé chacune appart, souvent elle applique certaines agrégations.

Dans le plan MapReduce, local rearrange annote les tuples avec les clés et laisse le reste de travail aux Local Sort de Hadoop. Global Rearrange est enlevé parce que sa logique est implémentée par le Shuffle et Merge de Hadoop. Load et Store sont souvent enlevés parce que le framework Hadoop est soucié pour écrire et lire les données.

Annexe C

Procédure d'installation de Hadoop et Hadoop Pig

1. Procédure d'installation de Hadoop sous Linux

Pour installer l'environnement Hadoop sous Linux, il faut suivre les étapes suivantes (voir le site [W09]) :

1.1. Precondition :

- **Passé à la session root :** pour qu'on possède tous les droits et les privilèges il faut à chaque fois travailler avec la session « root », pour passer à la session root, il faut passer par les étapes les suivantes :

1. Ouvrir le Shell et écrire la commande suivante :

```
sudo passwd root
```

Pour entrer un mot de passe à la session root

2. Redémarrer le PC
3. Au lieu d'entrer avec l'ancienne session, entrer avec "Other Session"

Pour l'username c'est : root

Pour le passwwd : le nouveau mot de passe

- **Connecté à internet**
- **Installation manuelle de java :**
  1. Installer « Icedtea java6 web start»
  2. Installer « open jdk java6 runtime »
- **Installation manuelle du SSH :**
  1. Installer « Secure shell client and server »
  2. Installer « ssh askpass »

- **Configuration de SSH :** Tout d'abord, nous devons générer une clé SSH pour l'utilisateur « root » via la commande suivante :

```
ssh-keygen -t rsa -P ""
```

Deuxièmement, vous devez activer l'accès de SSH pour votre machine locale avec cette clé nouvellement créée via la commande suivante :

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

La dernière étape consiste à tester la configuration SSH en vous connectant à votre ordinateur local avec l'utilisateur « root » par la commande :

```
ssh localhost
```

- **Désactivation d'IPV6 :**

1. Pour désactiver IPv6 sur Ubuntu, ouvert / etc / sysctl.conf dans l'éditeur de votre choix. Si on utilise le Shell on utilise la commande suivante :

```
gedit /etc/sysctl.conf
```

2. Ajouter les lignes suivantes à la fin du ce fichier :

```
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

3. redémarrer le PC
4. Vous pouvez vérifier si IPv6 est activé sur votre machine avec la commande suivante :

```
cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

Une valeur de retour d'IPv6 est activée, une valeur de 1 signifie qu'est désactivé

1.2. Installation d'Hadoop :

1. **Télécharger Hadoop** : vous devez télécharger Hadoop à partir du site officiel [W10], et extraire le contenu du package Hadoop à l'emplacement « home » ensuite le renommer du Hadoop.x.x.x au Hadoop.
2. **Mise à jour $ HOME / .bashrc**

2.1.Ajouter des lignes à la fin du fichier $ HOME / .bashrc :

- Ouvrir le fichier par : **gedit .bashrc**

- Ajouter les lignes suivantes :

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/root/hadoop
```

```
# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
# If you have LZO compression enabled in your Hadoop cluster and
# compress job outputs with LZOP (not covered in this tutorial):
# Conveniently inspect an LZOP compressed file from the command
# line; run via:
#
# $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
#
# Requires installed 'lzop' command.
#
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

1.3. Configuration de l'HDFS :

1. Mise à jour de fichier hadoop-env.sh :

- Ouvrir/conf/hadoop-env.sh dans l'éditeur pour définir l'environnement JAVA_HOME variable pour le Sun JDK/JRE 6répertoire

- Changer les lignes suivantes :

```
# The java implementation to use.  Required.
 # export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

Par:

```
# The java implementation to use.  Required.

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
```

2. **Créer le répertoire et définissez les autorisations requises** par les commandes suivantes :

```
mkdir -p /app/hadoop/tmp

chmod 750 /app/hadoop/tmp
```

Ouvrir ce fichier qui est dans le chemin « hadoop/conf/hdfs-site.xml », ensuite ajouter les lignes suivantes :

```
<!-- In: conf/hdfs-site.xml -->
    <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.
      The actual number of replications can be specified when the file is created.
      The default is used if replication is not specified in create time.
    </description>
    </property>
```

4. **Formatage de l'HDFS via le Namenode :** Pour formater le système de fichiers (qui initialise simplement le répertoire spécifié par la variable dfs.name.dir), exécutez la commande suivante :

```
hadoop/bin/hadoop namenode -format
```

5. **Vérification de fonctionnement** : pour tester le fonctionnement de l'environnement il faut tester les commandes essentielles suivantes :

1. **Démarrer le JobTracker, le Namenode, le Datanode, et les TaskTrackers** sur votre machine exécutée la commande **:**

```
hadoop/bin/start-all.sh
```

2. **Vérifier si Hadoop est à l'écoute sur le port de configuration via la commande suivante :**

```
netstat -plten | grep java
```

3. **Pour arrêter tous les démons** tournent sur votre machine exécutée la commande suivante :

```
hadoop/bin/stop-all.sh
```

4. Exécution d'un job MapReduce :

Nous allons maintenant exécuter votre premier Hadoop MapReduce travail. Nous allons utiliser l'exemple : WordCount qui lit les fichiers texte et les chiffres de la fréquence à laquelle se produisent mots. L'entrée est des fichiers textes et la sortie est des fichiers texte aussi. Ce programme permet de compter combien de fois les mots sont apparus dans chaque ligne de ces fichiers entrantes.

4.1. Redémarrer leJobTracker, le Namenode, le Datanode, et les TaskTrackers sur votre machine

4.2. Avant d'exécuter le travail réel MapReduce, nous avons d'abord besoin de copier les fichiers à partir de votre système de fichiers local vers l' HDFS,pour copier ces fichiers textes (text1, texte2) qui sont par exemple localisés dans un dossier nommé « inputdata » nous utilisons la commande suivante :

```
hadoop/bin/hadoop dfs -copyFromLocal /tmp/inputdata /root/inputdata
```

4.3. Exécuter le travail MapReduce via la commande suivante :

```
jar hadoop/hadoop*examples*.jar wordcount /root/input  /root/output
```

5. Ajouter les interfaces suivantes dans le navigateur web

Hadoop est livré avec plusieurs interfaces web qui sont par défaut (voir conf / hadoop-default.xml) disponibles aux endroits suivants :

http://localhost:50030/ - interface utilisateur Web pour le job MapReduce tracker (s)

http://localhost:50060/ – web UI for task tracker(s)

http://localhost:50070/ – web UI for HDFS Namenode(s)

Ces interfaces Web fournissent des renseignements concis sur ce qui se passe dans votre cluster Hadoop.

2. Procédure de configuration de Pig sous Linux

3. Mise à jour des fichiers

Ajouter les Extraits suivants entre la partie <configuration> ... </ configuration> dans les fichiers de configuration XML respectifs.

3.1 Pour le fichier **conf** / core-site.xml

Ouvrir ce fichier core-site.xml qui est dans le chemin « hadoop/conf/core-site.xml », ensuite ajouter les lignes suivantes :

```
<!-- In: conf/core-site.xml -->
    <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
    </property>
    <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:54310</value>
    <description>The name of the default file system.  A URI whose
      scheme and authority determine the FileSystem implementation.  The
      uri's scheme determines the config property (fs.SCHEME.impl) naming
      the FileSystem implementation class.  The uri's authority is used to
      determine the host, port, etc. for a filesystem.</description>
    </property>
```

3.2. Pour le fichier **conf / mapred-site.xml :**

Ouvrir ce fichier quiest dans le chemin « hadoop/conf/mapred-site.xml », ensuite ajouter les lignes suivantes :

```
<!-- In: conf/mapred-site.xml -->
    <property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
    <description>The host and port that the MapReduce job tracker runs
      at.  If "local", then jobs are run in-process as a single map
      and reduce task.
    </description>
    </property>
```

3.3. Pour le fichier **conf / hdfs-site.xml**

Pour configurer pig dans linux, il faut que le Hadoop soit déjà installé dans la machine ensuite on suivre les étapes suivantes :

1. Connecter à internet
2. Télécharger le pig à partir de ce cite [W11].
3. Extraire le contenu d'Archive et le déplacer ver le root « home »
4. Renommer le fichier à extraire à partir pig-0.10.0 vers pig seulement
5. Mise à jour de Fichier .bashrc

   5.1.Ouvrir le fichier nommé .bashrc et ajouter les lignes suivantes :

```
# Set Pig-related environment variables

export PIG_HOME=/root/pig

    # Add Pig bin/ directory to PATH

    export PATH=$PATH:$PIG_HOME/bin
```

6. Tester le fonctionnement de Pig

   6.1. Démarrer le Hadoop par la commande suivante :

```
hadoop/bin/start-all.sh
```

   6.2. Démarrer le help de pig via la commande suivante :

```
pig -help
```

Si ça donne le help, donc le pig marche correctement.

Références Bibliographiques et Webographiques

Références Bibliographiques

[Benitez et *al.* 2001] : Benitez ,E., Collet, C. & Adiba, M. (2001). Entrepôts de données : caractéristiques et problématique. Revue TSI, 20(2).2001

[Codd 1993] : Codd ,E.F. (1993) .Providing OLAP (On-Line Analytical Pro anITmandate .Technical report,1993

[Dean & Ghemawat, 2004] : Dean, J. & Ghemawat, S. (2004).MapReduce: Simplied data processing on large clusters .In Proc.OSDI,13,138,142

[Desrosiers 2011] : Desrosiers, C. (2011) .Entrepôt de données et intélligence d'affaires, cours de department de genie logiciel et des TI,Ecole de Technologie Supérieur,Québec,2011 .

[De Malleray 2008] : De Malleray,E. (2008). Meta données et analyses multidimensionnelles à travers les hypercubes. Mémoire de recherche scientifique, école nationale supérieure des mines de nancy – loria, France, 2008.

[Inmon 2005] : Inmon, W . H. (2005).Building the data warehouse. Wiley, 2005. William Inmon. Building the DataWarehouse .QED Technical PublishingGroup, Wellesley, Massachusetts, U.S.A,2005.

[Kimball&Caserta, 2004] : Kimball, R. & Caserta , J. (2004). The datawarehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data Wiley Publishing, Inc., Canada.

[Kimball 1996] : Kimball, R. (1996). Entrepôts de données, Guide pratique du

concepteur Datawarehouse. John Wiley and Sons, Inc., USA, 1999.

[Liu et *al.* 2011] : Liu, X., Thomsen, C. & Pedersen, T.(2011) ETLMR: A Highly Scalabl Dimensional ETL Framework based on MapReduce. TR-29 A DB Technical Report, in Proceedings of 13th International Conference on Data Warehousing and Knowledge, Toulouse, France, August 2011, pp. 96-11.

[Moise 2011] : Moise, D.M. (2011) Optimizing data management for MapReduce applications on large scale distributed infrastructures, Thèse de doctorat, Université européenne de Bretagne, 2011.

[Serna Encinas 2005]: Serna Encinas, M.T. (2005). Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation. Thèse de doctorat, Université de Joseph Fourier, 27 juin 2005.

[Olston et *al.* 2008] : Olston, C., Reed, B., Srivastava , U., Kumar, R. & Tomkins, A. (2008).PigLatin: A not- so-foreign language for data processing .In Proc.ACM SIGMOD international conference on Management of data, pages 1099-1110, 9–12 June, 2008,Vancouver, BC,Canada.

[Ponniah 2010] : Ponniah, P. (2010) .Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. John Willy & Sons, INC.Canada ,2010.

[Teste 2000] : Teste,O. (2000).Modélisation et manipulation d'entrepôts de données complexes et historisées .Thèse de Doctorat de l'Institut de Recherche en Informatique Université Paul Sabatier de

Toulouse (France), Décembre 2000.

[White 2009] : White, T. (2009). Hadoop: The Definitive Guide .O'Reilly, Media, Inc.USA, 2009

[Khouri et *al.* 2009] Khouri,S., Bellatreche, L. & Fankam , C. (2009) SISROM2C : Un outil de modélisation conceptuelle à base ontologique d'un entrepôt de données. Juin, 2009, pp. 123-138

RéférencesWebographie

[W01] : http://hadoop.apache.org/docs/r0.20.2/mapred_tutorial.html

[W02] : http://www.ubuntu-fr.org

[W03] : http://www.jmdoudoux.fr/java/dej/chap-frameworks.htm

[W04] : http://ofps.orielly.com/titles/9781449302641/intro.html

[W05] : http://hadoop.apache.org/docs/r0.20./hdfs_design.html

[W06] : http://wiki.apache.org/pig/OldFrontPage

[W07] : http://wiki.apache.org/pig/UDFanual

[W08] : http://jmdoudoux.fr/java/dej/chap-jvm.htm

[W09] : http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[W10] : http://www.globalish.com/am/hadoop/core/hadoop-0.20.2/

[W11] : http://archive.apache.org/dist/pig/pig-0.10.0/pig-0.10.0.tar.gz

[W12] : http://www.redbooks.ibm.com/redbooks/pdfs/sg242238.pdf

[W13] : http://www-irisa.univ-ubs.fr/Michele.Raphalen/cours/SID.pdf