F.S.D : ... N° D'ordre : ... ...

Université Sâad DAHLAB de Blida

Faculté des Sciences

Département d'Informatique

Mémoire présenté par :

MORSLI Mohammed

En vue d'obtenir le diplôme de MASTER

Domaine : Mathématique et Informatique

Filière : Mathématique et Informatique

Spécialité : Informatique

Option : Ingénierie des Logiciels

Sujet :

Mise en œuvre d'un processus ETL dans un environnement Disco de MapReduce

Soutenue le : 30/06/2013, devant le jury composé de :

M^me.	S. BENSETTITI,	Présidente
M^lle.	I. CHERFA,	Examinatrice
M^r.	S. FERFERA,	Examinateur
M^r.	M. BALA,	Promoteur

Promotion : 2012/2013

Dédicaces

Je dédie ce travail

A toute personne qui sa présence dans ma vie m'est très chère, et particulièrement mes chers parents

Mohammed

Résumé

Notre travail s'inscrit dans le cadre des systèmes d'information décisionnels (SID), plus particulièrement dans la phase d'intégration basée sur un processus ETL. Plus précisément, notre objectif est de proposer et d'implémenter des techniques et des stratégies pour permettre à ce processus de faire face au nouveau phénomène des données massives connues sous le nom de Big Data. Des données de cette dimension (PetaBytes, HexaBytes, ...) mettent l'ETL en difficulté et celui-ci risque de s'exécuter pendant des heures voire des jours ou plutôt risque de ne pas aboutir.

En parallèle, de nouveaux environnements et paradigmes se développent tels que l'informatique dans les nuages (cloud computing) et MapReduce. Le modèle MapReduce est un modèle qui présente une grande cohérence avec le processus ETL et est destiné justement pour le traitement intensif à grande échelle des données massives sur un cluster d'ordinateurs.

Nous avons retenu de mettre en œuvre un framework ETL basé sur le paradigme MapReduce sous l'environnement Disco en partant des limites d'une approche existante (prototype ETLMR, 2011). Notre approche consiste en l'implémentation des stratégies et des techniques pour la parallélisation des données (partitionnement et distribution des données) et la parallélisation des tâches ETL sur un cluster grâce au Framework Disco et ce pour l'amélioration des performances.

Mots clés

Systèmes d'information décisionnels, ETL, Données intensives, MapReduce, ETLMR, Performance, Parallélisation des données, Partitionnement et Distribution, Parallélisation des tâches, Disco.

ملخص

عملنا ينتمي إلى النظم المعلوماتية لصنع القرار، لاسيما في مرحلة التكامل على أساس البرنامج استخراج-تحويل-تحميل (إ.ت.ت أو ETL). تحديدا، هدفنا هو اقتراح و برمجة تقنيات و إستراتيجيات لتمكين هذا البرنامج من مواجهة ظاهرة البيانات الضخمة المعروفة بإسم البيانات الكبيرة (Big Data). هذا الحجم من البيانات (HexaPytes, PetaBytes, ...) يضع ETL في صعوبة و قد يعمل لساعات أو أيام، أو بالأحرى قد لا ينجح.

في موازاة ذلك، بيئات حاسوبية و نماذج جديدة طورت، مثل الحوسبة السحابية (Cloud Computing) و (MapReduce). النموذج MapReduce هو نموذج متناسق للغاية مع عمليات ETL و الذي صمم خصيصا للمعالجة المكثفة للتكتل المعلوماتي في نطاق واسع على مجموعة من الحواسيب الآلية.

لقد اخترنا تنفيذ برمجة في إطار ETL على أساس النموذج MapReduce في البيئة الحاسوبية (Disco) بدءا من سلبيات و نقائص نهج مصمم سابقا (نموذج ETLMR ، 2011). نهجنا هو برمجة إستراتيجيات و تقنيات لموازاة البيانات (تقسيم و توزيع البيانات) و موازاة عمليات ETL اعتمادا على البيئة الحاسوبية ديسكو (Disco) لتحسين الأداء.

الكلمات الرئيسية

أنظمة صنع القرار، ETL ، البيانات الضخمة، MapReduce ، ETLMR ،تحسين الأداء، موازاة البيانات، التقسيم و التوزيع، موازاة العمليات، ديسكو (Disco)

Abstract

Our work is part of the decision support systems (DSS), particularly in the integration based on ETL phase. More specifically, our aim is to propose and implement techniques and strategies to enable this process to face the new phenomenon of massive data known as Big Data. Data of this size (PetaBytes, HexaBytes ...) put the ETL in difficulty and it may run for hours or days, or rather may not succeed.

In parallel, new environments and paradigms get developed such as cloud computing and MapReduce. The MapReduce model is one that is highly consistent with the ETL process and is intended precisely for intensive large-scale massive data processing on a cluster of computers.

We chose to implement an ETL framework based on the MapReduce paradigm in the Disco environment starting from limits of an existing approach (prototype ETLMR, 2011). Our approach is the implementation of strategies and techniques for data parallelism (partitioning and data distribution) and the parallelization of ETL tasks on a cluster with the Framework Disco for performance improvement.

Keywords

Decision support systems, ETL, Big Data, MapReduce, ETLMR, Performance, Data parallelization, Partitioning and Distribution, Tasks parallelization, Disco.

Table des matières

Chapitre II : Processus ETL

Partie II : Processus ETL dans un environnement MapReduce

Chapitre III : Paradigme MapReduce

Chapitre IV : Prototype ETLMR

Partie III : Mise en œuvre du système

Chapitre V : Conception du système

Chapitre VI : Implémentation du système

Liste des figures

Liste des tableaux

Liste des acronymes

SI	Système d'Information
SID	Système d'Information Décisionnel
ED	Entrepôt de Données
DW	Data Warehouse
ETL	Extract Transform Load
SGBD	Système de Gestion de Bases de Données
CSV	Comma Separated Values
XML	eXtensible Markup Language
HTML	Hyper Text Markup Language
ISO	International Standardization Organization
EBCDIC	Extended Binary Coded Decimal Interchange Code
ASCII	American Standard Code for Information Interchange
SE	Système d'Exploitation
BD	Base de Données
BDR	Base de Données Relationnelle
OLAP	OnLine Analytical Processing
E/S	Entrée/Sortie
FS	File System
NTFS	New Technology File system
FAT	File Allocation Table
DFS	Distributed File System
MB	Mega Byte
GB	Giga Byte
RPC	Remote Procedure Call
GFS	Google File system
HDFS	Hadoop Distributed File System
NRC	Nokia Research Center
DDFS	Disco Distributed File System
ETLMR	Extract Transform Load on Map/Reduce

ODOT	One Dimension One Task
ODAT	One Dimension All Tasks
DBMS	Database Management System
PDI	Pentaho Data Integration
EIAO	European Internet Accessibility Observatory
TDR	Table de Données Relationnelle
TD	Table de Données
SQL	Structured Query Language
SGBDR	Système de Gestion de Bases de Données Relationnelles
CPU	Central Processing Unit
RAM	Random Access Memory
OTOT	One Table One Task
OTAT	One Table All Tasks
OTnT	One Table 'n' Tasks
nTOT	'n' Tables One Task
SSH	Secure SHell
IDE	Integrated Developpement Environnement
SGBDRO	Système de Gestion de Base de Données Relationnelles et Objets
PL	Procedural Language
API	Application Programming Interface

Introduction générale

Introduction générale

Contexte général

Face à la mondialisation et à la concurrence grandissante, la prise de décision est devenue cruciale pour les dirigeants d'entreprises. L'efficacité de cette prise de décision repose sur la mise à disposition d'informations pertinentes et d'outils adaptés. Le décisionnel vise à transformer les données opérationnelles en informations à des fins d'analyse et d'aide à la décision. Les entrepôts de données constituent de nos jours l'infrastructure de base des systèmes d'aide à la décision. Ils permettent de constituer de grands volumes d'informations historisées et précalculées permettant ainsi de découvrir des tendances et aboutir à des conclusions indispensables aux organisations. Pour amener ces informations à l'entrepôt, un processus communément appelé Extracting-Transforming-Loading (ETL) est nécessaire. Celui-ci extrait des informations des systèmes sources divers, les transforme pour leur donner de la qualité et de la valeur et les charge enfin dans l'entrepôt de données

Lors des premiers projets décisionnels, la phase de la collecte des données et de leur transformation était souvent sous-estimée. C'est peut-être là une des principales explications des échecs de réalisations et de très nombreux dépassements de budget. Pourtant cette phase d'extraction et de transformation préalable représente (selon les spécialistes du milieu) à peu près les trois quart du projet décisionnel. Un ETL réalise une tâche complexe que ce soit pour alimenter un entrepôt de données ou autres cibles. Il prend en charge l'une des fonctions les plus essentielles de la chaîne décisionnelle. Il est donc indispensable de traiter le processus ETL comme projet à part entière vu sa complexité et il doit être réalisé en se basant sur une méthodologie de développement bien définie.

Notre travail se focalise principalement sur le processus ETL d'entreposage de données et ce point de vue complexité et importance du processus dans la réussite d'un projet décisionnel. Nous nous plaçons, bien sûr, dans le cadre de grands projets décisionnels caractérisés par un grand nombre de sources de données ainsi

qu'une très importante volumétrie (TeraBytes voire PetaBytes) mais aussi un nombre important et une complexité au niveau des tâches ETL.

De nouvelles technologies ont vu le jour récemment visant à faire face aux besoins croissants en termes de calcul et de stockage. Ils ont suscités ces dernières années une attention particulière, sous l'impulsion notamment d'acteurs tels que Google, Microsoft, Amazon, Yahoo! ou encore Facebook. Leurs avancées dans le domaine des machines parallèles et des réseaux à très hauts débits nous laissent croire que les environnements répartis et parallèles formeront (ou forment déjà) les nouveaux standards des architectures à venir.

MapReduce est une approche bien établie dans le domaine de recherche du calcul parallèle et distribué. C'est un modèle récemment émergé, qui a gagné en popularité très rapidement grâce à Google au cours des dernières années. Les abstractions spécialisées comme MapReduce ont été développées pour gérer efficacement la charge de travail des processus qui manipulent intensivement les données.

L'avènement de ces environnements constitue une réelle opportunité pour développer et faire migrer le processus ETL fonctionnant dans un modèle classique vers un environnement basé sur le paradigme MapReduce.

Problématique

Le processus ETL réalise une tâche essentielle dans un projet décisionnel. Actuellement, il fait face à un nouveau handicap vu l'accroissement sans cesse des données et la complexité de ces tâches, qui détériorent de manière significative les performances de ce dernier. Deux problèmes principaux se posent lorsqu'on essaye de tenir compte de cet handicap : effectuer le calcul d'une manière efficace en termes de latences et fournir l'espace de stockage capable de satisfaire les exigences de la manipulation des données intensives.

Il apparaît donc nécessaire de concevoir des techniques et outils pour l'optimisation des performances du processus ETL.

Objectifs

Notre travail vise à répondre à la problématique de performance de l'ETL citée ci-dessus et plus particulièrement à traiter conjointement la problématique du volume des données et celle de la performance des tâches ETL. Notre idée consiste à distribuer les données sources pour, d'une part, répartir les données sur plusieurs sites (parallélisation des données) et, d'autre part, permettre un traitement parallèle des tâches ETL (parallélisation des tâches) pour l'accélérer. Généralement, un processus de distribution fragmente tout d'abord un ensemble de données avant de le répartir. Nous procédons donc en deux étapes : préparation des données sources et fragmentation, puis répartition.

Ainsi, nous mettons en œuvre le processus ETL dans un environnement de calcul parallèle MapReduce de Disco que nous avons adopté à l'occasion de ce projet.

Organisation du mémoire

Notre mémoire est organisé autour de trois parties :

Dans la 1ère partie, nous présentons les fondamentaux sur les systèmes d'information décisionnels. Cette partie est constituée de deux chapitres :

Dans le 1er chapitre, nous exposons les généralités sur les systèmes d'information décisionnels et les avantages que propose l'activité d'analyse et de prise de décision. Le second chapitre présente le processus ETL en soulignant ces grandes fonctionnalités et en mettant l'accent sur les différents facteurs relatifs à son bon fonctionnement.

Dans la 2ème partie, nous allons essentiellement nous intéresser au processus ETL dans un environnement MapReduce. Cette partie est composée de deux chapitres aussi :

Le 3ème chapitre présente les concepts de base nécessaire à la compréhension du paradigme MapReduce. Le 4ème chapitre concerne un travail existant d'un

processus ETL dans un environnement parallèle MapReduce, qui est le prototype ETLMR.

Dans la 3ème et dernière partie, nous présentons la conception et la mise en œuvre de notre système, ainsi que sa validation. Cette partie comprend deux chapitres :

Vous trouverez dans le 5ème chapitre l'étude détaillée de notre système. Dans le 6ème et dernier chapitre, nous allons présenter l'implémentation de notre système ainsi que les expérimentations nécessaires à sa validation.

Nous clôturons ce manuscrit par une conclusion et des perspectives pour des travaux futurs.

Partie I :

Fondamentaux sur les systèmes d'information décisionnels

Chapitre I :

Généralités sur les systèmes d'information décisionnels

I. Introduction

Face à une complexité de l'informatique, un environnement instable où les rachats et fusions d'entreprises sont nombreux, une concurrence omniprésente et une internationalisation des échanges où chaque choix stratégique peut être vital, les décideurs ont besoin d'avoir une vision claire de leurs affaires à tout moment, très rapidement et à l'aide d'outils faciles à utiliser, puissants et fiables visant à faciliter la tâche de pilotage de l'entreprise sans perturber le système de production existant.

Une entreprise se doit en permanence de pouvoir se situer par rapport à la concurrence, mais également par rapport à la demande et à ce qu'elle peut offrir, quelle entreprise ne voudrait pas fidéliser ses clients en les identifiant mieux pour leur proposer les produits ou services susceptibles de les intéresser, maîtriser les risques qu'elle prend, optimiser ses activités, exploiter intelligemment les données stockées et avoir plus d'information que ses concurrents.

Pour faire face aux nouveaux enjeux économiques, l'entreprise doit anticiper, l'anticipation ne peut être efficace qu'en s'appuyant sur l'information pertinente.

D'où l'ère de l'information décisionnelle, qui offre aux entreprises les moyens d'exploiter les données stockées, souvent restées au stade d'archives et éparpillées sur de multiples systèmes hétérogènes, donc non organisées dans une perspective décisionnelle. Ces moyens permettent d'en extraire le contenu informationnel qui servira de miroir de l'état de l'entreprise et qui permettra la compréhension du passé et du coup à une meilleure anticipation du futur.

II. Informatique de production et informatique de décision

Les Systèmes d'Information (SI) jouent un rôle vital dans le développement et le succès des entreprises. L'un des modèles de conception les plus utilisés pour ces systèmes est le modèle relationnel, il permet de stocker et de manipuler d'importants volumes de données.

Il existe deux grandes familles de systèmes d'information, on distingue en effet :

- Les SI de production ou opérationnels et,
- Les SI d'aide à la décision ou décisionnels.

Les systèmes opérationnels ou encore transactionnels sont conçus pour supporter les traitements quotidiens de l'entreprise, tels : ajouter une commande, modifier une adresse de livraison, rechercher les informations d'un client, etc. L'utilisateur dans le monde transactionnel, pose des questions relativement simples, par exemple « quel est le chiffre d'affaire du mois dernier ?». Les besoins du décideur sont totalement différents. Il ne pose pas la question précédente mais plutôt une question du genre « quelles sont les ventes du produit X pendant le trimestre A de l'année B dans la région C ? ». Cette requête est complexe, lente et pénalise le système de production. Face à ce problème les entreprises ont recours à des nouvelles organisations spécifiques, qui permettent de mémoriser un grand volume de jeux de données, et qui soient structurées de façon à avoir un moteur de recherche rapide, efficace et approprié, on parle des systèmes d'information décisionnels.

II.1. Informatique de production

Les SI opérationnels sont utilisés pour la gestion du quotidien, qui permet de supporter les traitements au jour le jour de l'entreprise (ajout, consultation, suppression et modification). Ils sont associés à des applications développées pour répondre à des problématiques métiers, et optimisées pour gérer des transactions peu consommatrices de ressources. Leur objectif principal est la saisie puis le traitement de données, ainsi que la production des résultats en sortie. D'une manière générale, ces systèmes manipulent un volume de données important tout en garantissant un

accès rapide à l'information. L'intégrité des données est nécessaire pour ce genre de systèmes (il faut par exemple interdire la modification simultanée d'une même donnée par deux utilisateurs différents). La réponse à des requêtes généralement peu complexes permet des temps de réponse relativement réduits.

Autre caractéristique de ces systèmes est qu'ils conservent leur état instantané. Dans la plupart des cas, l'évolution n'est pas conservée. On garde simplement une trace des versions instantanées pour la reprise en cas de pannes et/ou pour des raisons légales.

II.2. Informatique de décision

II.2.1. Définition

Les systèmes d'information décisionnels ont été définis dans [1] comme suit :

« Un système d'information décisionnel (SID) est un ensemble de données organisées de façon spécifique, facilement accessibles et appropriées à la prise de décision. On peut aussi le voir comme une représentation intelligente de ces données au travers d'outils spécialisés. La finalité d'un système décisionnel est le pilotage de l'entreprise ».

Les moyens de parvenir à une activité de pilotage passent par une information riche, pertinente, détaillée, historisée, fiable et stable. Le pilotage induit également l'utilisation d'outils d'analyse et de restitution puissants et adaptés à chacun des thèmes concernés, ainsi qu'une forte capacité à faire évoluer les données et les outils.

II.2.2. Architecture

Un système décisionnel ne peut se construire en marge des autres systèmes de l'entreprise. Il nécessite une vision globale du système d'information opérationnel. On regroupe donc, dans un référentiel commun, des éléments provenant de plusieurs applications, orientés métiers et représentant la mémoire de l'entreprise. On parle

généralement d'architecture ou modèle n-tiers en raison de différentes couches possibles pour gérer les données et réaliser les analyses. Ce modèle peut être représenté classiquement ainsi :

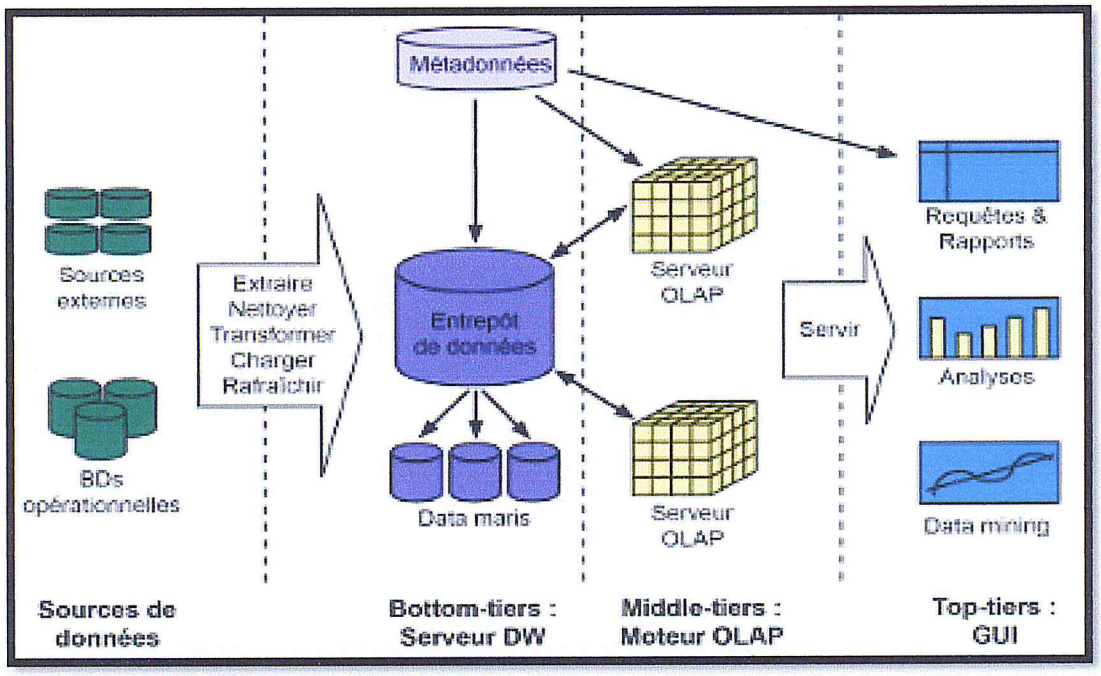


Figure 1 : Architecture du système d'information décisionnel [2]

L'architecture décisionnelle met en jeu quatre niveaux sur deux phases caractéristiques que sont l'intégration des données et l'analyse. Ces phases sont basées sur cinq éléments essentiels qui composent le système décisionnel:

1. **Sources de données**

   Il s'agit majoritairement de données internes à l'entreprise, celles des fichiers et des bases de données opérationnelles du système de gestion de l'entreprise. Ce peut être aussi des sources externes, récupérées via des services distants, par exemple, des web services.

2. **Entrepôts de données**

   C'est une base de données massive et centralisée, structurée spécifiquement pour les analyses et les interrogations décisionnelles.

3. **Magasins de données (Data Marts)**

   Ce sont des extraits de l'entrepôt où chacun focalise sur une partie du métier, contenant un volume moindre de données et permettant des analyses plus rapides.

4. **Cubes de données**

   Ce sont des contextes d'analyse multidimensionnels (multicritères).

5. **Outils d'analyse**

   Les outils d'analyse sont la partie visible offerte aux utilisateurs souvent non informaticiens (directeurs, chefs de services, etc.). Par leur biais, les analystes peuvent manipuler facilement les données contenues dans les entrepôts et les magasins de données. Ils offrent de différentes représentations de l'information : tableaux, graphiques, statistiques, etc.

II.2.3 Comparaison entre l'informatique productive et l'informatique décisionnelle

Dans l'entreprise les données sont stockées sous différents systèmes et différents formats selon des processus transactionnels, ces processus mettent en avant la mise à jour en temps réel, la rapidité d'accès, la garantie d'intégrité et la sécurité des transactions effectuées (en particulier les bases réparties en plusieurs lieux physiques). Ces processus reflètent l'exigence de l'informatique transactionnelle, mais s'avèrent inadaptés pour bien répondre aux volumes et à la complexité des traitements d'analyse.

Le système de production contient des informations concernant l'activité de l'entreprise (les ventes, les produits, les stockes, les clients, les fournisseurs, etc.)

Les décideurs dans l'entreprise sont à la recherche de systèmes exploitant efficacement ces informations pour des fins d'analyse et de prise de décision. Ces systèmes utilisent des processus d'aide à la décision qui doivent mettre en avant la rapidité et l'interactivité d'analyses pour des données généralement volumineuses, variées et historisées (qui gardent l'évolution des données). Les utilisateurs du

système décisionnel sont peu nombreux que les utilisateurs du système productif, on parle de décideurs.

Le Tableaux 1 compare des caractéristiques sur les données et leur utilisation, des deux systèmes : de production et de décision [TES 00] et [MAI 06].

	Opérationnel	**Décisionnel**
Données	Orientées application	Orientées activité (thème, sujet, …)
	Détaillées	Condensées (ou agrégées)
	Courantes et précises au moment de l'accès	Historiques
	Petites quantités de données utilisées par un traitement	Grandes quantités de données utilisées par les traitements
Utilisation	Mise à jour interactive	Pas de mise à jour interactive
	Interrogations avec des requêtes prédéfinies et peu complexes permettent des temps de réponse relativement réduits	Interrogations avec des requêtes imprévisibles et complexes qui résultent à des réponses moins rapides
	Accéder d'une façon unitaire par une personne à la fois (respecte l'intégrité)	Utilisées par l'ensemble des analystes avec des accès non concurrents.
	Forte portabilité d'accès (utilisateurs nombreux et variés : employés, dirigeants, …)	Faible portabilité d'accès (utilisateurs peu nombreux, que les décideurs)
	Utilisé de façon répétitive	Utilisé de façon aléatoire

Tableau 1 : Comparaison des systèmes opérationnels à des systèmes décisionnels

III. Entrepôts de données

Toute entreprise possède actuellement d'importants volumes de données, stockés le plus souvent dans différents médias (bases de données, documents, papier, ...) et a besoin de se permettre une exploitation efficace et performante de ces données pour tirer des informations importantes. S'il existe effectivement des informations importantes, il en est nécessaire de construire une structure pour les héberger, les organiser et les restituer à des fins d'analyse. Cette structure est « L'*Entrepôt de Données* (ED)» ou « Data Warehouse (DW) » en anglais. Ce n'est pas une usine pour produire l'information, mais plutôt un moyen de la mettre à disposition des utilisateurs de manière efficace et organisée.

III.1. Définition classique

Le concept d'entrepôt de données a pris forme au début des années 90 par Bill Inmon, il est devenu depuis *la clé de voûte de ce que l'on appelle l'informatique décisionnelle*.

Depuis, de nombreuses définitions ont été proposées, soit académiques, soit par des éditeurs d'outils et de bases de données ou par des constructeurs cherchant à orienter ces définitions dans un sens mettant en valeur leur produits.

La définition la plus appropriée est celle de W.H. Inmon dans son ouvrage de référence « Building the Data Warehouse » [INM 05] :

« *L'entrepôt de données est une collection de données intégrées, orientées sujet, non-volatiles, historisées, résumées et disponibles pour l'interrogation et l'analyse*». Cette définition englobe différents termes que nous explicitons :

1. **Données intégrées**

   Un entrepôt de données permet d'intégrer des données provenant de sources éventuellement hétérogènes et dispersées. Avant d'être incorporées dans l'entrepôt de données, les données doivent être mise en forme et unifiées afin d'avoir un état cohérent. Pour y parvenir, l'intégration nécessite une bonne

connaissance des sources de données pour une forte normalisation de données. Mais aussi, la maîtrise de la sémantique et la prise en compte des contraintes référentielles et des règles de gestion. Ces notions sont énoncées et administrées au sein des métadonnées[1] de l'entrepôt de données.

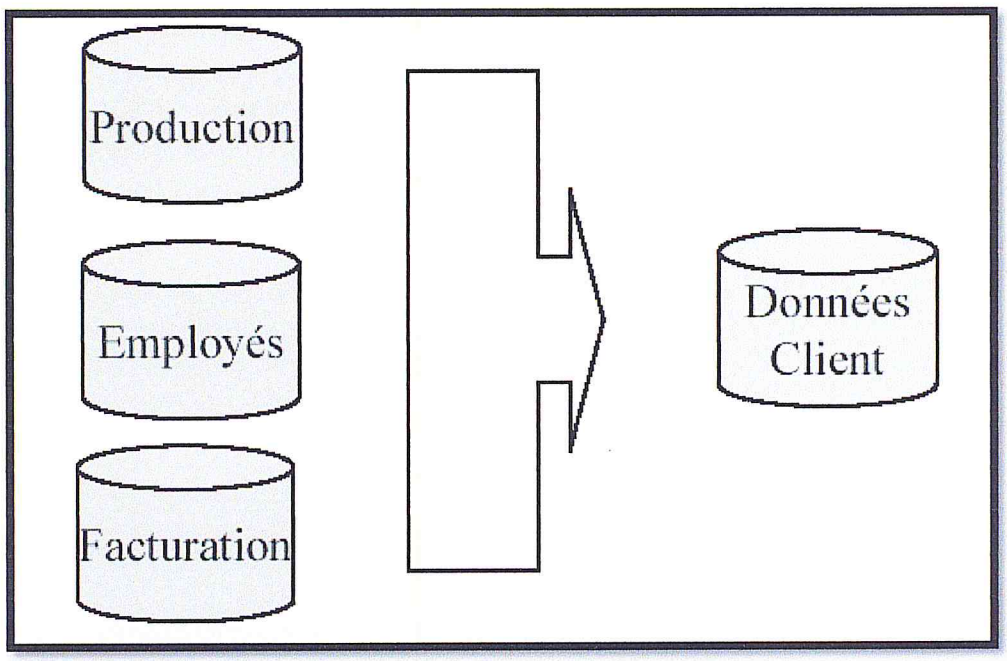


Figure 2 : Données intégrées dans un entrepôt de données [3]

2. Orientées sujet

Les données sont orientées « sujet » dans la mesure où elles sont organisées par thème. L'entrepôt de données regroupe en son sein des informations de différents métiers (fabrication, achat, qualité) de l'entreprise. Généralement, chaque métier possède des informations de familles communes, mais relatives à un sujet différent (clients, produits, contrats) [MAI 06]. Cet état de fait peut être représenté par le schéma de la Figure 3.

Le fait qu'un entrepôt de données soit orienté sujet permet une analyse plus pertinente des données critiques car on synthétise toutes les facettes du sujet que l'on étudie.

[1] Métadonnée : est une donnée servant à définir ou décrire une autre donnée cible.

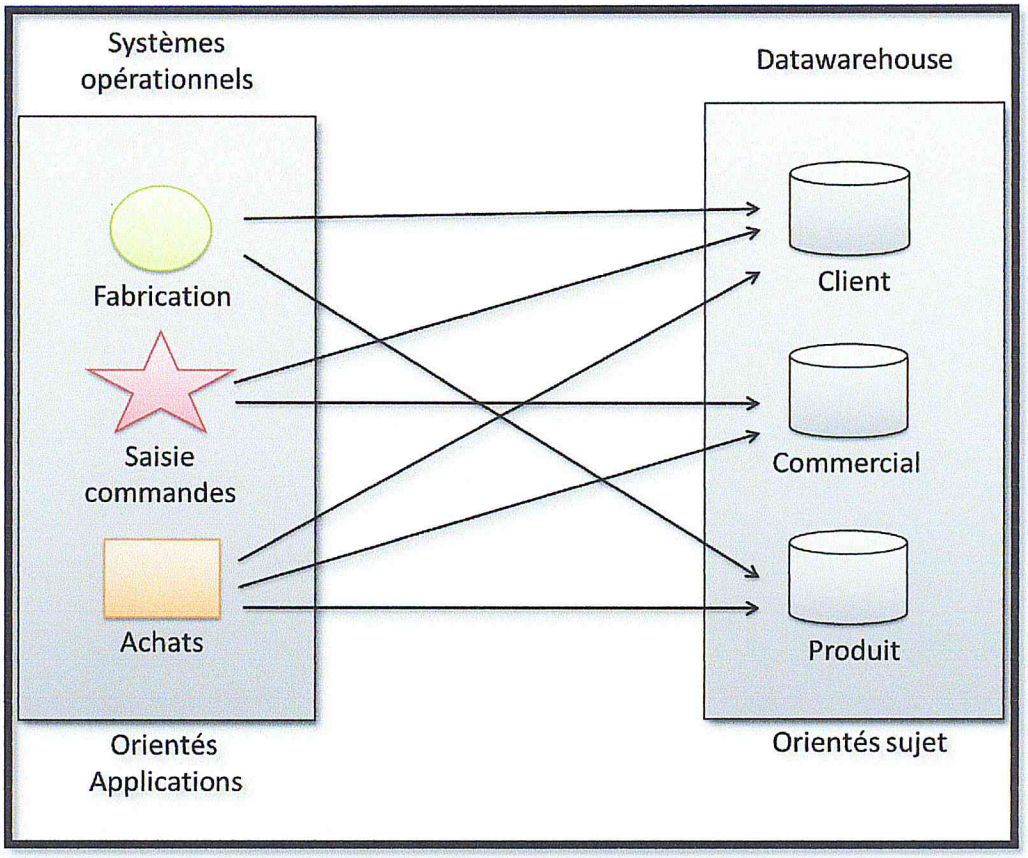


Figure 3 : Données orientées sujet dans un entrepôt de données [MAI 06]

3. Données historisées

L'historisation des données est nécessaire pour suivre dans le temps l'évolution des différentes valeurs des indicateurs à analyser afin de rendre possible la réalisation d'analyses comparatives au cours du temps. Ainsi, permettre de prédire les évolutions futures en s'appuyant sur les évolutions passées.

4. Données non-volatiles

Les données de l'entrepôt de données sont figées, en lectures seules et non modifiables afin de conserver la traçabilité des informations et des décisions prises. L'entrepôt de données est stable, ses données sont figées dans le temps, et la non-volatilité des données serait en quelque sorte la conséquence directe de l'historisation dans ce cas.

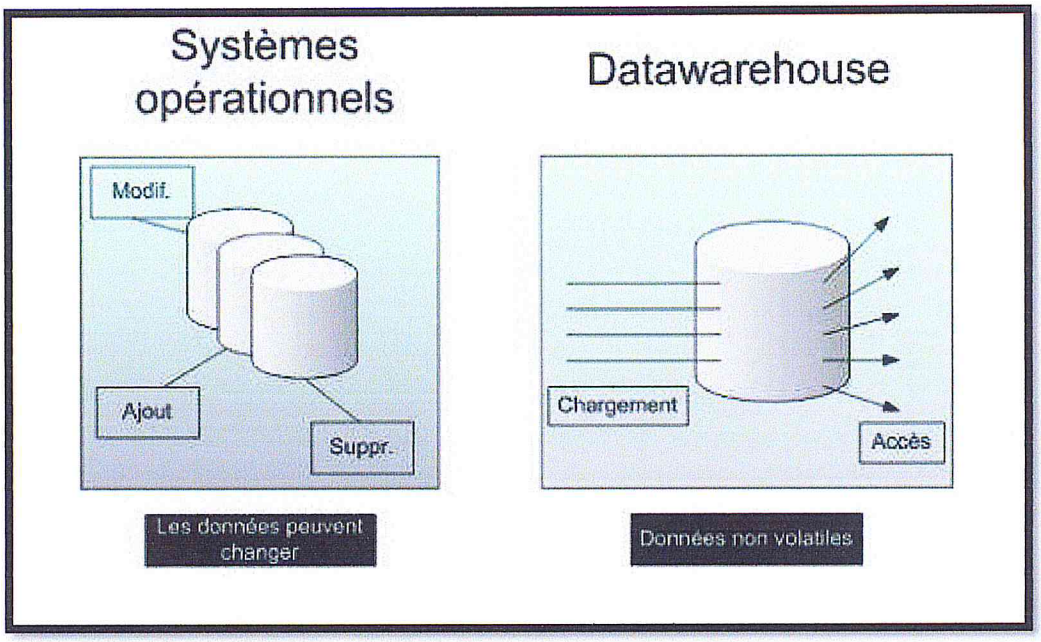


Figure 4 : Données non-volatiles dans un entrepôt de données [MAI 06]

5. **Données résumées**

   Ce sont des données issues en moins détailles, donc regroupées, agrégées et réorganisées afin de faciliter le processus de prise de décision.

6. **Disponibles pour l'interrogation et l'analyse**

   Les utilisateurs doivent consulter les données réorganisées de l'entrepôt en fonction de leurs droits d'accès au travers d'outils interactifs d'aide à la manipulation et l'analyse.

Le modèle multidimensionnel répond fortement à ces cinq critères.

III.2. Modèle multidimensionnel

L'entrepôt de données héberge toute information utile provenant des systèmes de production et systèmes externes. Avant d'être intégrée dans l'entrepôt de données, l'information doit être extraite et nettoyée. Ensuite, elle doit être mise en forme de manière à devenir compréhensible pour l'utilisateur final. Cette mise en forme de

l'information passe par une réorganisation et une structuration dans un modèle spécifique plus proche des utilisateurs finaux et adapté à leurs besoins d'analyse et prise de décision. On parle du modèle multidimensionnel.

Pour arriver à construire un modèle approprié pour un entrepôt de données, nous pouvons choisir de multiples schémas. Avant de parler et décrire les différents schémas, nous commençons par quelques concepts de base.

III.2.1. Faits

Le fait modélise le sujet de l'analyse. Un fait est formé de mesures (données observables, indicateurs, ...) correspondant aux informations de l'activité analysée [TES 00].

Les mesures sont généralement numériques et additives pour permettre de résumer un grand nombre d'enregistrements (on peut les additionner, les dénombrer ou bien calculer le minimum, le maximum ou la moyenne).

Exemple : le fait « vente » peut être constitué de mesures d'activités suivantes : quantité de produits de vente « Quantité » et montant de ventes « Total ».

Le sujet analysé, c.à.d. le fait, est analysé suivants différents perspectives. Ces perspectives correspondent à une catégorie utilisée pour caractériser les mesures d'activités analysées ; on parle de la catégorie des dimensions.

III.2.2. Dimensions

Une dimension modélise une perspective de l'analyse. Une dimension se compose de paramètres correspondant aux informations faisant varier les mesures de l'activité [TES 00].

Ceci dit, une dimension est le critère suivant lequel on souhaite évaluer, quantifier et qualifier le fait. Elle se compose de paramètres, aussi dits attributs. Chaque

paramètre peut prendre différentes valeurs faisant varier les mesures de l'activité à analyser.

Une dimension peut être décomposée en hiérarchies afin de permettre à l'utilisateur d'examiner ses indicateurs (mesures) à différents niveaux de détail, tout en allant du niveau global au niveau le plus fin ou inversement. Par exemple, la date peut être décomposée en *année*, *mois*, *semaine* et *jour*.

III.2.3. Cubes

Un cube de données offre une abstraction très proche de la façon avec laquelle l'analyste voit et interroge les données. Les données seront toujours des *faits* à analyser suivant une ou plusieurs *dimensions* qui déterminent une *mesure* d'intérêt. Par exemple, dans le cas de ventes de produits à des clients dans le temps (trois dimensions, donc un vrai cube), les faits sont les ventes, les dimensions sont les clients, les produits et le temps, la mesure d'intérêt est le montant des ventes. Cela revient à dire que pour chaque combinaison des trois dimensions (clients, produits, temps), on peut accéder à la mesure numérique (cellule sur le schéma du cube) associée au fait vente [4].

La sélection dans la Figure 5 ci-après montre les ventes du produit A dans tout temps et pour tout client. Pour obtenir le total des ventes du produit A, il suffit juste d'agréger (sommer) les éléments de la sélection indiquée.

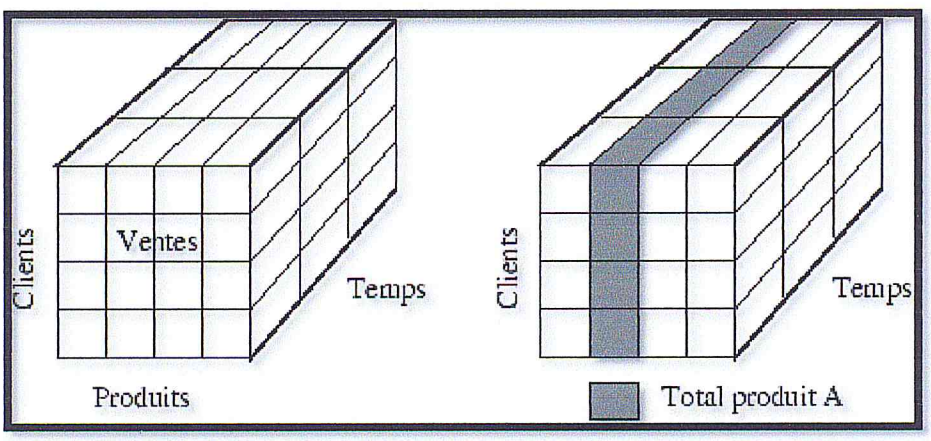


Figure 5 : Exemple de cube de données [4]

III.3. Modélisation

L'un des axes visés pour qualifier un modèle décisionnel est la lisibilité du modèle d'un point de vue de l'utilisateur final. Ce qui qualifie la modélisation multidimensionnelle est qu'elle soit assez proche de la manière de penser des analystes et adaptée à leurs besoins. Cette modélisation consiste à considérer un sujet analysé comme étant un point dans un espace à plusieurs dimensions (axes d'analyse). Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse.

On part du principe que les données sont des faits à analyser selon plusieurs dimensions, et il est possible de réaliser une structure de données simple qui correspond à ce besoin de modélisation multidimensionnelle. Cette structure est constituée d'une table de faits centrale et des tables de dimensions situées autour.

Au niveau logique, cela peut se traduire par trois schémas différents : en étoile, en flocon de neige ou en constellation.

III.3.1. Schéma en étoile

Dans un modèle en étoile, tous les faits à analyser sont définis dans une simple table relationnelle. Cette table va être reliée par sa clé primaire à d'autres tables correspondant aux dimensions. Les relations de la table de faits sont toutes de type « un-à-plusieurs » et sa clé primaire est la concaténation des clés primaires de toutes les tables de dimensions qui gravitent autour d'elle.

La Figure 6 montre un schéma en étoile en décrivant les ventes réalisées dans les différents magasins d'une entreprise au cours d'un jour. Dans ce cas, nous avons une étoile centrale avec une table de faits appelée Ventes, qui contient les mesures (le prix des produits vendus « Montant », la quantité des produits vendus « Quantité ») et qui peut être agrégée de diverses façons. Autour de la table de faits, nous avons les diverses dimensions : Temps, Produit et Magasin. Ces dimensions fournissent la base pour l'agrégation des mesures de la table de faits.

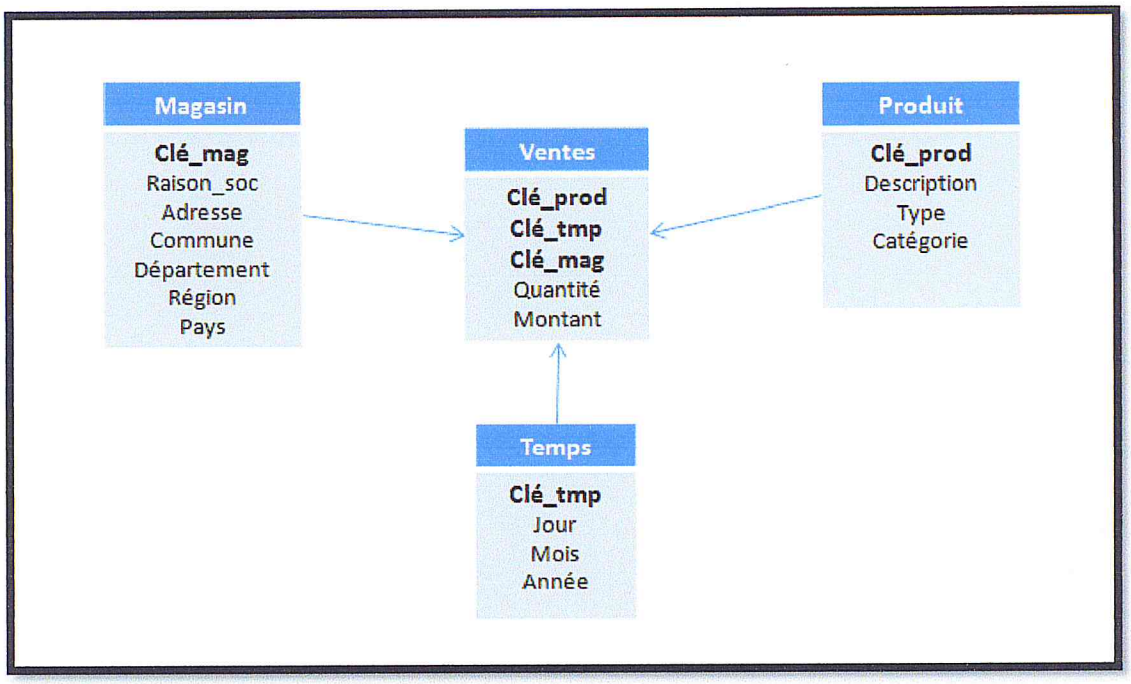


Figure 6 : Exemple d'un schéma en étoile

III.3.2. Schéma en flocon de neige

Le schéma en flocon de neige dérive du schéma précédent avec une table de faits centrale et autour d'elles les différentes tables de dimensions, qui sont éclatées ou décomposées en sous tables selon la hiérarchie de ces dimensions. L'avantage de cette modélisation est de formaliser une hiérarchie au sein d'une dimension, ce qui peut faciliter l'analyse. Un autre avantage est représenté par la normalisation de ces dimensions, car nous réduisons leurs tailles.

Néanmoins dans [KIM 04], l'auteur démontre que c'est une perte de temps de normaliser les relations des dimensions dans le but d'économiser de l'espace disque. Par contre, cette normalisation rend plus complexe la lisibilité et la gestion dans ce type de schéma. En effet, ce type de schéma augmente le nombre de jointures à réaliser dans l'exécution d'une requête.

Les hiérarchies pour le schéma en flocon de neige de l'exemple de la Figure 6 sont :

Dimension Temps = Jour → Mois → Année

Dimension Magasin = Commune → Département→ Pays

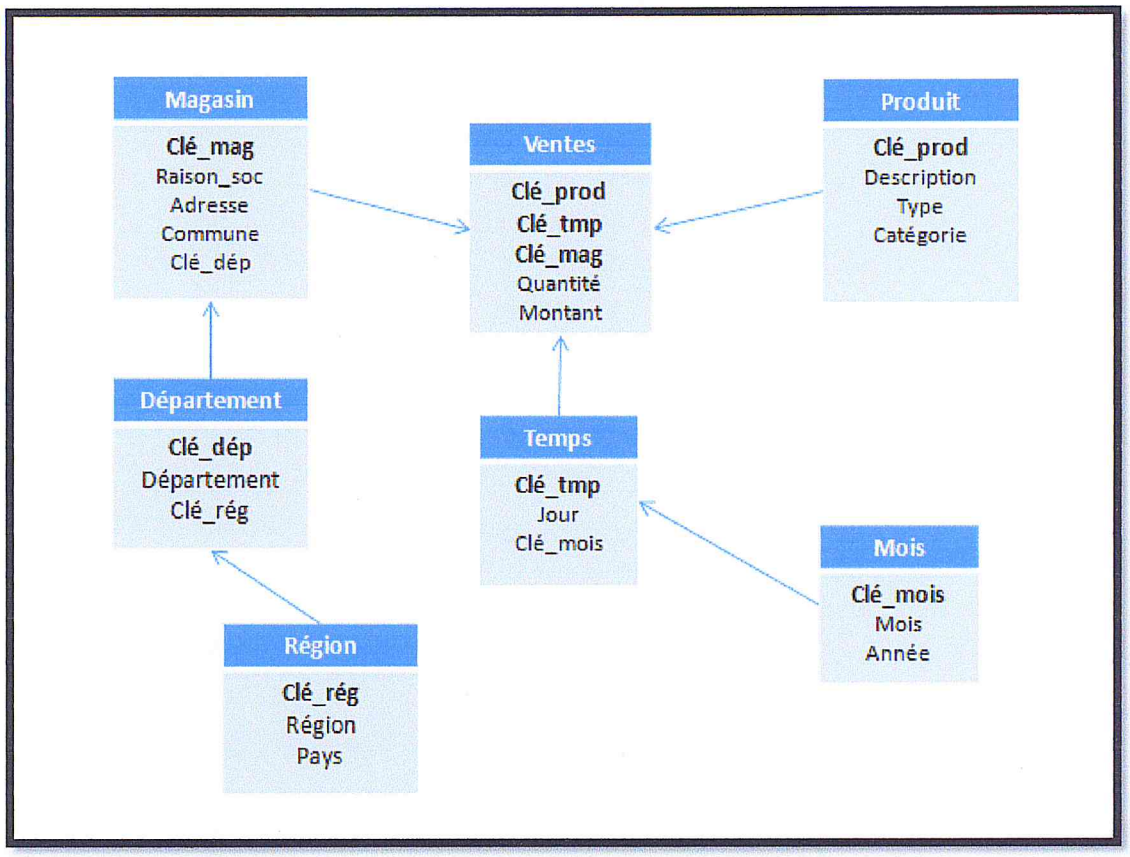


Figure 7 : Exemple d'un schéma en flocon de neige

III.3.3. Schéma en constellation

Le schéma en constellation est encore basé sur le modèle en étoile, mais on rassemble plusieurs tables de faits qui utilisent les mêmes dimensions. Un modèle en constellation fusionne donc plusieurs modèles en étoile qui utilisent des dimensions communes.

IV. Conclusion

Nous avons présenté dans ce chapitre les systèmes d'informations décisionnels et les avantages que propose l'activité d'analyse et de prise de décision. Le système

comprend cinq éléments que nous avons décrits : sources, entrepôts, magasins, cubes et les outils d'analyse. Ensuite, nous avons pu évoquer les principaux concepts liés à la conception et la construction des entrepôts de données.

Pour nourrir un entrepôt de données, il faut établir des flux de données entre l'entrepôt et les bases de production. Pour ce faire, on met en place des interfaces d'extraction, de transformation et d'alimentation entre les bases de production et l'entrepôt de données. C'est ce qu'on appelle un outil *Extract Transform Load* (*ETL*). Le chapitre suivant sera consacré à l'outil ETL.

Partie I :

Fondamentaux sur les systèmes d'information décisionnels

Chapitre II :

Processus ETL

I. Introduction

Les données d'un entrepôt sont, pour la plupart, issues de différentes sources de données opérationnelles de l'entreprise. Au fil du temps, les systèmes opérationnels de l'entreprise ont tendance à se complexifier, les technologies évoluent et se diversifient, les sources de données se multiplient. Des solutions logicielles sont alors nécessaires à leur intégration et à leur homogénéisation. Les outils les plus communément utilisés dans ce domaine sont appelés les ETL (Extract, Transform, Load ou Extraction, Transformation, Chargement en français). Ce type d'outil est destiné tout simplement à collecter des données et les préparer pour alimenter l'entrepôt de données, comme présenté dans la Figure 8 ci-après. Concrètement, on dispose de sources (souvent hétérogènes) que l'on extrait pour alimenter un entrepôt servant à leurs analyses. Les sources peuvent aussi bien être des bases de données (de n'importe quel SGBD), des fichiers (Txt, CSV, Excel, XML, HTML, etc.) et voir d'autres formats. Les ETL s'occupent de transformer ces sources, via de nombreuses composants, en une ou plusieurs cibles qui peuvent être, là aussi, de n'importe quels formats.

Lors des premiers projets de l'informatique décisionnelle, cette phase de collecte et de préparation des données était généralement sous-estimée. C'est peut-être là une des principales explications des échecs de réalisations et de très nombreux dépassements de budget. On retient que cette phase de collecte et de préparation préalable représente à peu près jusqu'à ¾ du projet [5].

II. Processus ETL (Extract, Transform, Load)

Comme expliqué précédemment, ETL signifie Extract, Transform, Load (Extraction, Transformation, Chargement). Ce sont les trois principales étapes que doit impérativement implémenter un outil ETL (Figure 8). Effectuées dans l'ordre, elles forment un traitement, un processus ou un scénario (selon les diverses appellations des logiciels). Nous allons à présent les détailler.

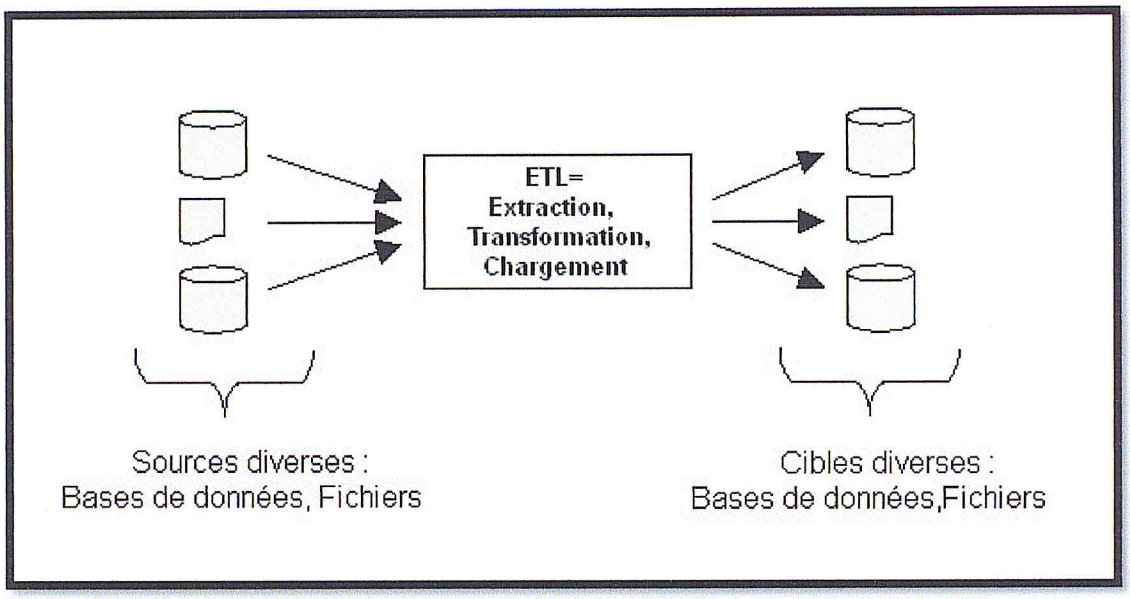


Figure 1 : Schéma d'un processus ETL [6]

II.1. Extraction des données

La première étape du processus ETL concerne l'extraction des données qui sont souvent hétérogènes. Cette extraction consiste à collecter les données utiles à partir du système de production (système source). Cet aspect est le plus difficile de l'ETL, l'extraction correcte des données prépare le terrain pour les procédures à venir qui sont, la transformation et le chargement.

La plupart des projets d'entreposage de données consistent à consolider les données provenant des systèmes sources différents. Chaque système peut également utiliser une organisation de données différente. On y trouve celle des SGBD qui respecte un schéma prédéfini, ce type de systèmes sources est dit structuré. Aussi, il existe celui non-structuré où la représentation de l'information pour ce type de sources n'adhère à aucun format, un fichier plat tel un fichier texte en est l'exemple. Entre ces deux types totalement opposés, il surgit un autre type qui est le semi-structuré. Dans ce type de systèmes sources, la structure n'est pas strictement prédéfinie. Cependant, les éléments de données portent leurs propres

définitions sémantiques sous formes de labels (balises), tels les fichiers XML et HTML.

Il est aussi important de pouvoir analyser lors de l'extraction de données. Il faut donc connaître les propriétés de celles-ci, savoir si par exemple cette donnée est de type entier ou chaîne de caractères et quelle est sa taille maximale. Cela peut paraître simple lorsque la source provient d'un SGBD mais s'avère plus complexe lorsqu'elle provient d'un fichier plat. Il faut aussi pouvoir reconnaître les clés primaires et étrangères permettant respectivement d'identifier une table de façon unique, et de garantir l'intégrité des données.

L'extraction doit aussi permettre l'homogénéisation et la préparation des données à la phase de transformation en leur générant une représentation uniforme c.à.d. en les convertissant dans un format unique qui est approprié pour un traitement de transformation.

Enfin, cette première étape doit être effectuée le plus rapidement possible en exploitant au minimum les ressources du système. Etant donné qu'un ETL peut occuper et ce, pendant de nombreuse heures, une grande partie des processus disponibles, on planifie donc souvent l'extraction à des moments opportuns afin d'éviter de saturer et perturber le système de production, comme la lancer la nuit par exemple. Pour gagner du temps, l'objectif de cette étape est aussi de filtrer au maximum les données. Par exemple, il faut que l'on puisse extraire les données uniquement mises à jour ou ajoutées après la dernière extraction. Certains outils commencent à fournir ce type de fonctionnement pour optimiser la phase d'extraction.

L'étape d'extraction est donc très importante. Elle doit être performante et complète pour pouvoir disposer d'un bon outil ETL.

II.2. Transformation des données

Cette seconde étape consiste à transformer les données, elle permet d'effectuer différentes opérations sur celles-ci afin de les nettoyer, les concilier, d'obtenir un

format qui respecte celui de la base cible c.à.d. l'entrepôt et de les dériver pour la prochaine étape qui est le chargement.

L'étude de la qualité des données et le nettoyage des données dans la phase de transformation sont les premiers pas les plus importants car les fichiers ou les bases de données utilisées en tant que sources peuvent contenir beaucoup d'inexactitudes et d'inconsistances. Nous y trouvons notamment des noms mal orthographiés, des textes inversés, des champs manquants, des codes non à jour, des informations incomplètes et l'utilisation d'abréviations différentes. Ces erreurs résultent d'une multitude de différents facteurs. Dans la majorité des cas, elles peuvent avoir été causées intentionnellement, comme dans le cas des fraudes par exemple. Des dysfonctionnements matériels ou logiciels peuvent aussi en être l'origine. Evidemment, de telles fautes peuvent causer des erreurs considérables lors de la lecture des analyses des données, lesquelles ne peuvent être mesurées.

L'objectif principal du nettoyage des données est donc d'assurer l'exactitude, la pertinence et la consistance des données à travers une organisation ou différentes divisions de l'organisation et de leur assurer que les décisions prises le sont sur des informations consistantes et justes.

Même nettoyées, les données doivent être aussi formatées de manière à être acceptées par l'entrepôt de données. Pour cela, il faut parfois avoir la taille des champs, leur type, l'ordre de ces champs au sein de l'enregistrement, etc.

Les transformations sont le cœur d'un ETL. Elles se présentent comme une série de règles ou de fonctions appliquées aux données extraites de la source. L'utilisateur définit des correspondances entre les schémas des sources et de la base cible. L'ETL s'appuie sur ces correspondances pour appliquer les transformations nécessaires sur les données sources et résoudre ainsi l'hétérogénéité sémantique. Pour certaines sources de données, il faudra une manipulation très légère voir aucune, de données. Dans d'autre cas, un ou plusieurs types des transformations suivantes peuvent être nécessaires pour répondre aux besoins de la base de données cible :

1. Sélectionner uniquement certaines colonnes pour charger (ou sélectionner les colonnes à ne pas charger), c'est de la projection (découpage des colonnes). Par exemple, si la source de données comporte trois colonnes (appelées également attributs) : roll_no, l'âge et le salaire, l'extraction peut ne prendre que roll_no et le salaire.
2. Traduire les valeurs codées (par exemple, quand les systèmes sources stockent « 1 » pour « masculin » et « 2 » pour « féminin », mais l'entrepôt de données inscrit « M » pour « masculin » et « F » pour « féminin ») ou les abréviations cryptiques appropriées à l'entreprise (AC, EN, RU et SU par exemple) par des valeurs compréhensibles pour l'utilisateur final.
3. Joindre ou fusionner les données provenant de sources multiples (par exemple, rassembler sur une seule et unique entité les informations d'un produit qui peuvent provenir de différentes sources de données lors d'une recherche)
4. Transposition ou pivotant (tournant plusieurs colonnes en plusieurs lignes et vice versa)
5. Fractionnement ou division d'une colonne en plusieurs colonnes (par exemple, mettre une liste séparée par des virgules qui est spécifiée comme une chaine dans une colonne, sous forme de valeurs individuelles dans différentes colonnes)
6. Suppression des doublons (l'élimination des données redondantes) ou la déduplication de données (relier les doublons du système source à un unique enregistrement dans l'entrepôt de données).
7. Elimination des valeurs nulles (par exemple, le mécanisme d'extraction peut ignorer tous les enregistrements dont le salaire n'est pas mentionné, salaire=Null) ou leur remplacement avec des valeurs prédéfinies (valeurs par défaut).
8. Filtrage de données. Celui-là est bien différent de celui de l'extraction puisque l'on filtre selon des critères à définir, par exemple on va filtrer les produits dont le prix est supérieur à 1000 euros.
9. Conversion des différents formats de dates possibles (format ISO : aaaa/mm/jj, format Européen : jj/mm/aaaa).

10. Conversion des chaines de caractères d'un système de codage à un autre, par exemple convertir les données d'un format EBCDIC vers ASCII.
11. Calcul et agrégation (par exemple, cumul -résumant plusieurs lignes de données- total des ventes pour chaque magasin et chaque région : total des ventes = quantité * prix unitaire)
12. Analyse lexicale des champs sources. Certains outils peuvent également réaliser ce genre d'analyse, ils seront donc capables de comprendre que les champs suivants signifient la même chose : « Boulvd », « Bd », « Boulevard ».
13. L'application de toute forme de validation de données simple ou complexe (par exemple, s'assurer que les valeurs de données entrent dans les domaines définis pour celles-ci). Si la validation échoue, cela peut entrainer un rejet complet ou partiel des données, et donc aucune, certaines ou toutes les données seront remises à l'étape suivante, en fonction de la conception et de la règle de gestion des exceptions.

Enfin, développer des programmes de conversions, de nettoyage et de standardisation n'est pas un travail difficile en soi, mais les mettre en œuvre à l'échelle de l'entreprise complète représente un véritable défi.

II.3. Chargement des données

Le chargement est la dernière phase de l'alimentation de l'entrepôt de données. Elle s'occupe de charger les données préalablement extraites et transformées vers l'entrepôt de données et/ou autres cibles. Les contraintes définies dans le schéma de l'entrepôt de données telles l'unicité, l'intégrité référentielle et les champs obligatoires contribuent à la performance de la qualité globale des données du processus ETL. Une autre technique pour garantir la cohérence des données consiste à gérer les valeurs des clés de tables de l'entrepôt d'une manière automatique au moment du chargement (insertion) des nouvelles données.

La réalimentation ou rafraîchissement d'un entrepôt de données est un procédé un peu différent de l'alimentation initiale (la toute première alimentation de

l'entrepôt) et cela varie selon les exigences de l'organisation. Certains entrepôts de données peuvent remplacer les informations existantes. D'autres peuvent ajouter de nouvelles données sous une forme historisée, par exemple, toutes les heures. Pour le comprendre, considérons un entrepôt de données qui est nécessaire pour garder des records de vente de l'année dernière, ensuite, on écrase les données qui ont plus d'un an avec des données plus récentes.

Le calendrier et la portée de remplacer ou d'ajouter sont des choix de conception stratégiques dépendants du temps disponible et les besoins de l'entreprise –bien sûr–, des systèmes plus complexes peuvent garder une piste de vérification et l'historique de toutes les modifications apportées aux données chargées dans l'entrepôt de données.

La phase du chargement est une phase plutôt mécanique et la moins complexe, mais elle risque d'être assez longue. Il faut mettre en place des stratégies pour assurer de bonnes conditions à sa réalisation et, comme dans les deux phases précédentes, de définir la gestion des erreurs (une chaine de caractères ne doit pas être insérée dans un champ fait pour les entiers).

Le chargement n'est pas à négliger pour un bon outil ETL, il doit, là aussi, être complet et performant.

III. Complexité et challenges

L'une des plus difficile parties d'un projet d'entreposage de données est le processus d'alimentation ETL. Le développement d'un système ETL compte normalement pour jusqu'à 75% des efforts d'un projet décisionnel d'après [5]. Après avoir vu ce qui était un système ETL, ses fonctionnalités et dans quel contexte il est utilisé, maintenant on passe à la complexité et les nombreux challenges que comprend l'implémentation et la mise en œuvre d'un processus ETL efficace et fiable.

ETL peut entraîner une complexité considérable, et d'importants problèmes fonctionnels peuvent se produire avec un système ETL mal conçu.

III.1. Complexité de l'extraction

III.1.1. Sources diverses et disparates sur différentes plateformes et SE

Alors que les systèmes d'information se complexifient, la variété des sources de données s'accroît également. Un processus ETL doit être capable de s'adapter à des sources de données hétérogènes et variées. Dans ce cas, l'outil ETL doit disposer d'une large palette de connecteurs à des BD, fichiers, services web, etc.

La diversité et l'hétérogénéité des sources d'information surtout celles distribuées sont une des principales difficultés rencontrées par les utilisateurs du Web aujourd'hui, par exemple. Cette hétérogénéité peut provenir du type de la structure des sources (sources structurées, semi-structurées ou non-structurées) et/ou du format de celles-ci.

Dans ce cas, il doit y avoir des informations complètes et détaillées sur chaque structure des sources impliquées dans l'extraction pour assurer de bonnes conditions de déroulement de cette phase et de pouvoir gérer les imprévus face à l'inconsistance de ces sources à cause d'un format différent ou difficilement interprétable.

III.1.2. Fusion et sources de données externes

Face à la diversité des sources (fichiers, BD, etc.), il faut préparer chacune d'entre elles afin de pouvoir fusionner les informations.

Outre, un des grands problèmes de la fusion est de traiter les données venant de l'extérieur du système. Il faut maintenir une surveillance permanente du système d'information pour pouvoir les identifier et s'assurer que ce sont les bonnes données qui sont recensées. De plus, la forme des données externes qui est souvent totalement anarchique accentue la difficulté. Pour que ces données soient utiles, elles nécessitent un reformatage pour pouvoir les incorporer dans une forme exploitable pour l'entreprise.

III.1.3. Volumétrie de données importante

Les volumes de données sont en croissance exponentielle, et les processus ETL doivent traiter des quantités importantes de données granulaires (produits vendus, appels téléphoniques, transactions bancaires, etc.). Certains systèmes décisionnels sont mis à jour de façon incrémentale (différentielle), alors que d'autres sont rechargés dans leur totalité à chaque itération. Dans ce cas, l'ETL doit nécessairement être capable de gérer de grandes quantités de données malgré que cette gestion implique des temps de traitement qui peuvent être très importants.

III.1.4. Applications legacy

Au fil du temps, les applications qui servaient autrefois les besoins de l'entreprise deviennent lourdes, cassantes, anciennes et dépassées mais toujours en cours d'utilisation. ETL doit être capable de s'adapter à ce genre d'applications dites « legacy », ou autres technologies obsolètes pour se permettre de migrer les données dans de nouveaux systèmes ou seulement de les intégrer.

III.1.5. Historique de changement non-préservé

Avoir l'historique de modification des données sources permet une alimentation incrémentale de l'entrepôt de données lors du rafraîchissement, c.à.d. de ne recharger l'entrepôt qu'avec les données modélisées ou ajoutées depuis la dernière extraction, on parle de « Change Data Capture (CDC) ». Ce fonctionnement permet d'optimiser la phase d'extraction en capturant que les données ayant subies des modifications ou ayant été nouvellement créées plutôt que recharger la totalité des données. L'idée consiste à réaliser des extractions distinguées en utilisant un mécanisme de marquage des données par examen de la date de la dernière modification associée à la donnée, par exemple.

III.2. Complexité de la transformation

Les besoins en termes de décision ont tendance à être de plus en plus variés. L'étape de transformation suit cette évolution et doit pouvoir proposer un éventail

de transformation de plus en plus riche. Ces transformations peuvent être très complexes. Les données doivent être agrégées, calculées, parsées, etc.

Voici les complexités qui figurent à la phase de transformation :

III.2.1. Qualité de données

Lorsque l'on déplace les données d'un environnement de production vers un environnement d'analyse, certaines défaillances dans la qualité des données apparaissent. Il est évident qu'une mauvaise qualité de ces données, qui sont de plus en plus utilisées par des processus de décision critiques, affectent très négativement l'efficacité de l'entreprise. D'après une étude du Gartner Group, la plupart des initiatives de réingénierie de l'information n'aboutiront pas à cause d'un manque de qualité de données [7].

Ces déficiences peuvent être liées à la forme et la structure des données comme au contenu. On y trouve au niveau lexical des erreurs dans les données sources telles que des dates incorrectes et aussi des caractères spéciaux hérités de l'époque des terminaux, par exemple. Ce genre de problème persistera toujours si aucune démarche de standardisation n'a été adoptée. On peut aussi s'y trouver même face à des incohérences liées, par exemple, à l'évolution de la signification de certains codes au cours du temps ou lorsqu'on essaie de réconcilier des données venant de sources différentes.

Pour remédier à ces déficiences, l'ETL doit nécessairement procéder à l'amélioration de la qualité des données. Il faut aussi réfléchir aux causes profondes de ces problèmes et adapter les procédures pour les éviter.

III.2.2. Structures cibles diverses

Les structures et les applications décisionnelles incluent des entrepôts de données, des magasins de données, des applications OLAP pour l'analyse, les tableaux de bord, etc. Toutes ces structures cibles présentent des besoins différents en termes de transformation de données, ainsi que des latences différentes.

III.2.3. Incohérence entre les différentes sources

Vu la diversité des sources internes et externes, la réconciliation peut nécessiter des transformations complexes :

- Application des diverses techniques de conversion ;

- Fournir des valeurs appropriées aux données manquantes ;

- Convertir les divers formats en un format commun (ASCII par exemple).

- Effectuer des calculs complexes.

- Reformatage des données externes

- Etc.

III.3. Complexité du chargement

III.3.1. Chargement initial

Le chargement est la dernière phase de l'alimentation de l'entrepôt de données. C'est une phase délicate notamment lorsque les volumes sont importants. Gérer l'insertion de plusieurs milliers de lignes lors du chargement initial prend un temps très important. Cela implique la lourdeur du système, une consommation des ressources très élevée et un maintien difficile des tests de la qualité des données insérées.

III.3.2. Réalimentation et entreposage en temps réel

Traditionnellement, les entrepôts de données ne contiennent pas les données exactement du jour présent. Ils sont habituellement chargés avec des données des systèmes opérationnels au plus par semaine ou, dans un certain cas, toutes les nuits, mais en tout cas, ils représentent toujours une fenêtre du passé. Le rythme rapide des affaires d'aujourd'hui est entrain de rendre ces systèmes d'historisation moins de valeurs pour les problèmes auxquels les gestionnaires et les responsables

gouvernementaux sont confrontés dans le monde réel. Les ventes du matin sur le côté Est aura une incidence sur la façon dont les provisions sont stockées sur le côté Ouest. Les compagnies aériennes et les agences gouvernementales doivent être capables d'analyser l'information la plus à jour lorsque l'on essaye de détecter les groupes suspects de passagers ou de l'activité potentiellement illégale [8].

Alors que le décisionnel se rapproche du temps réel, les entrepôts et les magasins de données doivent être rafraîchis plus souvent, dans des délais de chargement toujours plus courtes.

III.4. Complexité d'intégrité

L'exécution d'un processus ETL peut être source de dysfonctionnements. Cela peut avoir des conséquences sur les données telle la perte du flux de données lorsqu'un processus est interrompu ; donc perte de la cohérence des données, et même si le processus est repris on peut s'y trouver éventuellement face à un résultat différent. L'ETL doit fournir des outils permettant de conserver l'état et la cohérence des données si le processus est interrompu, d'où la *recouvrabilité*, et la relance du processus lorsque celui-ci a été arrêté avant même d'être terminé, d'où la *reprise*.

IV. Optimisation et performance

Le rôle d'un outil ETL est de transférer des données depuis une ou plusieurs sources de données (BD, fichiers plats, etc.), vers une ou plusieurs structures cibles (ED, magasins de données, etc.). Ce type d'outil doit nécessairement être capable de réaliser son rôle avec le meilleur scénario possible. La construction d'un outil ETL efficace et performant, avec des capacités de traitement de gros volumes de données et une adaptation subtile à la complexité des tâches de traitement sur les données, est critique et essentiel afin de pouvoir couvrir des besoins beaucoup plus large que ceux d'un simple ETL.

Afin d'atteindre un niveau de performance optimale dans les traitements de données, on s'y trouve confrontés à des problématiques qui nécessitent des interventions d'optimisation qui vont dépondre de plusieurs facteurs. Ces facteurs sont –bien évidemment– en relation avec les sources de données, les flux de données, les traitements de transformation et les structures de destination. Voici quelques solutions possibles à ce sujet :

IV.1. Ressources et consommation

Les processus ETL sont gourmands en termes de ressources. Du coup, le processus ETL complet est programmé pour être exécuté à des moments opportuns, où l'impact sera réduit pour les utilisateurs de ces mêmes ressources. Ces processus ETL se déroule souvent la nuit ou le week-end pour ne pas gêner les utilisateurs qui partagent ces mêmes ressources (réseau ou sources de données). Les données sont potentiellement moins soumises à modifications durant ces périodes.

IV.2. Planificateur de tâches

C'est un outil rangé dans la catégorie « Administration décisionnelle ». Il sert à planifier et programmer l'exécution des processus ETL.

IV.3. Gestionnaire de reprise

Les Procédures d'entreposage de données subdivisent généralement un grand processus ETL en petits morceaux pour une exécution séquentielle ou parallèle. Pour garder trace des flux de données, il est primordial de marquer chaque ligne de données avec « row_id », et étiqueter chaque procédure du processus avec « run_id ». Dans un cas d'échec, avoir ces ID aidera à faire reculer le processus pour relance l'opération échouée.

C'est un gestionnaire pour les dysfonctionnements. Il permet de prendre la relève et de relancer le processus ETL en cas de problème.

IV.4. Points de contrôle

Ce sont des états de certaines phases du processus ETL à un moment clé (par exemple, lorsque la phase est terminée), une fois arrivé à un point de contrôle, l'idée est de sauvegarder tout sur disque, nettoyer certains fichiers temporaires, connecter à l'état, et ainsi de suite. Cela permet la *recouvrabilité* des données en cas de problèmes.

IV.5. Monitoring (supervision)

Le monitoring est une activité de surveillance de diverses mesures d'activités de l'ETL. Il consiste à suivre le bon fonctionnement de l'exécution du processus ETL en temps réel. Cela permet de :

- Surveiller les mesures de performance, en termes de temps de réponse par exemple.
- Surveiller les mesures d'intégrité, une connaissance détaillée de l'état du système à chaque instant (CPU, RAM, etc.)
- Détection précise d'erreurs, affichage des rapports et possibilité de restauration.
- Affichage des statistiques d'exécution
- Identifier les goulets[2] d'étranglement (les maillons faibles limitant les performances globales) lorsqu'ils apparaissent. Par exemple, manque de mémoire vive, trop d'opérations d'E/S, filtrage trop tardif des données, etc.

IV.6. Exécution parallèle

Compte tenu du développement des technologies de l'information et de la communication, on est de plus en plus confronté à un volume de données très important. Les processus ETL travaillent souvent sur de gros volume de données. Ces outils doivent être capables de gérer cette importante volumétrie. Ceci

[2] Un goulet d'étranglement est un point d'un système limitant les performances globales, et pouvant avoir un effet sur les temps de traitement et de réponse. Les goulets d'étranglement peuvent être matériels et/ou logiciels.

nécessite d'adapter les techniques d'extraction, de transformation et de chargement pour que l'outil ETL ne prenne pas trop de temps.

Actuellement, on utilise de plus en plus le parallélisme. Ce dernier peut s'appliquer lorsqu'il faut :

IV.6.1. Traiter une source volumineuse

Lorsque les données de la source se trouvent dans un fichier par exemple, la taille de ce dernier peut avoir un impact non négligeable sur les applications qui doivent le manipuler, ils risquent surtout de rencontrer des difficultés en voulant le charger tout entier.

Une solution consiste à découper le fichier en des morceaux de tailles plus facilement manipulables. De cette manière, on peut aussi accéder à plusieurs parties simultanément.

IV.6.2. Traiter plusieurs sources de données

Lorsqu'on est en présence de plusieurs sources (fichiers par exemple), on peut envisager de traiter un certain nombre en parallèle. Si un processus est occupé à trier un fichier par exemple, on peut demander à un autre de supprimer les doublons d'un autre fichier.

De cette manière, on gagne du temps parce que le traitement du 2ème fichier commence plutôt que si on devrait attendre de finir le tri du 1er fichier.

Signalons tout de même que le multi-processing n'est pleinement opérationnel que sur des systèmes multiprocesseurs ! Heureusement qu'il se trouve qu'aujourd'hui la plupart des machines sont multiprocesseurs.

IV.6.3. Utiliser plusieurs processus ou threads pour une seule tâche

Parfois, il est possible de diviser une tâche ETL en plusieurs parties et les exécuter en parallèle afin de réduire le temps d'exécution global. Le succès de l'exécution en parallèle dépend de la capacité de pouvoir diviser la tâche de

traitement en petites tâches ou parties indépendantes qui peuvent s'exécuter simultanément. Chaque partie doit être effectuée à l'endroit le plus optimal dans la chaine de traitement.

Les possibilités, auxquelles un outil ETL peut être adapté, sont très nombreuses et c'est bien évidemment un critère à retenir pour disposer d'un outil ETL complet, malgré que la mise en place de telles solutions nécessite des compétences particulières, une maintenance lourde et des investissements conséquents.

V. Conclusion

Tout entrepôt de données est construit pour être chargé et ainsi, nous nous sommes focalisés sur le processus d'alimentation (ETL) en soulignant ses grandes fonctionnalités et en mettant l'accent sur les différents facteurs relatifs à son bon fonctionnement vu sa complexité et son importance dans la réussite d'un projet d'entreposage.

En effet, nous avons également tenté de fournir une vision globale sur les solutions qui ont pu être proposées pour faire face à la complexité des tâches ETL et la volumétrie des sources qui croît continuellement, comme le parallélisme. Nous avons pu réaliser que la mise en place de telle solution s'avère efficace pour gérer un processus de plus en plus complexe, manipulant une masse de données de plus en plus conséquente.

Le chapitre suivant vise à décrire un modèle de la solution parallèle qui est actuellement très répondu dans le contexte du Big Data. On parle du MapReduce.

Partie II :

Processus ETL dans un environnement MapReduce

Chapitre III :

Paradigme MapReduce

I. Introduction

MapReduce est un modèle de programmation et une implémentation associée au traitement et à la génération de larges ensembles de données [DGH 04]. Il a été développé chez Google par Jeffrey Dean et Sanjay Ghemawat. Leur motivation a été dérivée de la multitude des calculs qui ont été effectués tous les jours chez Google impliquant d'énormes quantités de données. En général, ces calculs sont conceptuellement simples. Par exemple, trouver la requête la plus fréquente soumise au moteur de recherche de Google pour un certain jour donné, ou garder la trace des pages web explorées jusqu'à présent. Les données d'entrée pour ces calculs seraient si volumineuses qu'il faudrait du traitement réparti (distribué, partagé, ...) sur des centaines ou des milliers de machines dans le but d'obtenir des résultats dans un laps de temps réduit et raisonnable.

II. Systèmes distribués

II.1. Définition

Andrew S. Tanenbaum a défini un système distribué comme étant un ensemble d'unités de traitement indépendantes qui, à l'utilisateur, se comporte comme un système unique [TAN 07].

Bien que cette définition ne soit pas très spécifique, elle couvre tous les cas d'utilisations importantes des systèmes distribués. Un ensemble d'ordinateurs qui sont connectés par un réseau et se partagent certaines parties de leur charge de travail forme un système distribué qui peut être considéré comme un seul ordinateur qui se compose de plusieurs processeurs. Aujourd'hui, les systèmes distribués peuvent être trouvés dans de nombreuses applications.

II.2. Caractéristiques

L'idée générale pour la construction d'un système distribué est que le partage du calcul ou traitement entre plusieurs processeurs est plus rapide que de le faire sur un

seul processeur. Cela implique notamment la notion du calcul parallèle. Un système réparti est capable de l'exécution parallèle.

La performance et la facilité d'utilisation d'un système distribué est décrite par quatre importants indicateurs [CDK 12] :

1. Ouverture
2. Transparence
3. Extensibilité
4. Tolérance aux pannes

II.2.1. Ouverture

Le critère ouverture évalue la capacité du système distribué d'interagir avec d'autres systèmes ou l'utilisateur lui-même. La meilleure façon pour répondre à cette exigence est d'ouvrir les interfaces pour la communication externe, de sorte que les composants externes ou les utilisateurs peuvent accéder au système facilement. Une des conséquences de l'ouverture d'un système est sa capacité d'être étendu par de nouveaux composants, même sans avoir besoin de l'éteindre avant.

II.2.2. Transparence

Comme indiqué dans la définition par Tanenbaum [TAN 07], la transparence d'un système distribué signifie que les composants du système doivent se comporter comme un processeur unique pour l'utilisateur. Cela implique par exemple la dissimulation de la structure du système et ses composants.

II.2.3. Extensibilité

En général, il ya deux dimensions qui sont utilisés pour mesurer la capacité d'un système distribué à l'extensibilité (evolutivité, scalabilité, ...) : le nombre de processeurs et la quantité de la charge de travail.

En théorie, quand on augmente le nombre des unités de traitement, le système gagne en puissance de calcul, il se renforce. Bien que dans la pratique, l'ajout de nouveaux composants est accompagné par une augmentation de la charge de travail. Dans ce

cas, les composants ajoutés n'augmentent pas la puissance globale du système significativement, parce que l'accroissement de la charge de travail compense le gain dans la puissance du calcul.

Il est également important de tester la capacité du système à faire face à une charge de travail élevée. Cela peut augmenter la communication réseau ou même surcharger certains composants du système. Le système doit être capable de gérer cette situation et poursuivre son travail.

II.2.4. Tolérance aux pannes

La tolérance aux pannes est impliquée dans les trois premiers critères parce que c'est le problème fondamental dans la mise en œuvre des systèmes distribués. Non seulement un des composants du système peut être défectueux, mais même la communication entre eux peut être interrompue. Cela augmente la probabilité de défaillance, et à cause de cela quelques parties du système peuvent ne pas fonctionner correctement.

Quand on parle de la tolérance aux pannes on évoque toujours la communication. Les défaillances au sein d'un composant peuvent être gérées par programmation ou tests techniques, mais des erreurs dans les couches de communication ne sont pas facilement détectées et corrigées. Le système distribué doit être capable de gérer ce genre de situation.

II.3. Architectures

En général, il y a deux types d'architecture pour les systèmes distribués, l'architecture client-serveur et le peer-to-peer. Dans certains cas et pour certains systèmes, une solution hybride est envisagée.

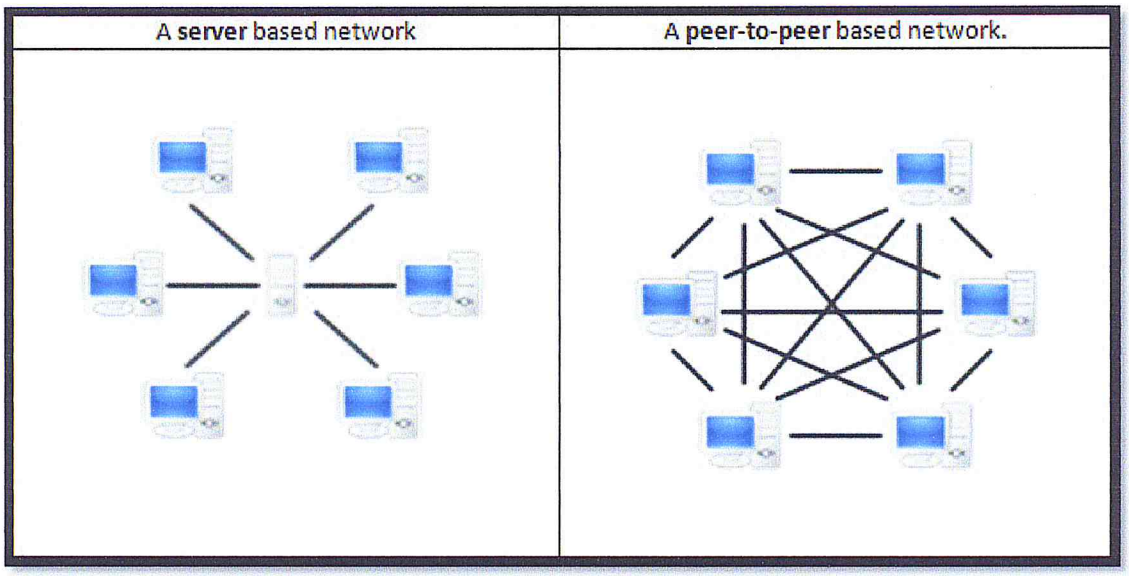


Figure 9 : Architectures des systèmes distribués [9]

II.3.1. Client-serveur

Le modèle client-serveur est constitué de deux types de nœuds dans le réseau, un grand nombre de clients et un petit nombre de serveurs, comme indiqué sur la Figure 9 ci-dessus. Les clients et les serveurs communiquent via un système de requête/réponse. Un client envoie une requête à un serveur pour traiter une tâche spécifique. Le serveur exécute la tâche et renvoie un message de réponse comprenant le résultat au client.

En général, les clients ne sont pas en mesure de se communiquer directement. Ils n'ajoutent, non plus, rien de fonctionnalités supplémentaires au système. Par contre, augmenter le nombre des clients résulte à une plus grande charge de travail pour le serveur. Cela réduit les performances du système.

Que les serveurs possédant les ressources systèmes peuvent être utilisées pour des calculs et traitements. S'il y a une défaillance sur le serveur, le système est incapable de continuer à fonctionner. Le serveur est l'unique point de défaillance dans ce modèle.

En dépit de ces inconvénients, le modèle client-serveur est le modèle de communication le plus fréquent. C'est facile de le mettre en œuvre et de lui appliquer la séparation des tâches dans le réseau.

II.3.2. Peer-to-peer

Dans un environnement peer-to-peer, tous les nœuds du réseau sont égaux. Chacun d'entre eux est capable de fournir les mêmes services et de communiquer avec d'autres nœuds (voir Figure 9). Chaque nœud peut joindre et quitter le réseau à tout moment.

L'idée générale sur un réseau peer-to-peer est qu'il n'y a pas de serveur central. Chaque nœud est à la fois le client et le serveur, d'où le nom peer-to-peer (pair-à-pair en français). A cause de cela, il n'est pas facile de détecter et de gérer des nœuds spéciaux dans le réseau, en particulier dans les réseaux avec un grand nombre de nœuds et une grande fluctuation.

Par conséquent, de nombreuses applications dans la pratique utilisent un serveur d'annuaire pour rendre l'adressage d'un nœud particulier plus facile. Il est également important d'empêcher le serveur d'annuaire de devenir un point de défaillance.

Les nœuds sont égaux et, par conséquent, ils doivent prendre soin des ressources par eux-mêmes. Chaque nœud a ses propres ressources. L'ajout de nouveaux nœuds au réseau augmente automatiquement la puissance du calcul globale du réseau. La défaillance d'un seul nœud peut être compensée par d'autres nœuds et le système reste toujours en mesure de fonctionner correctement.

Les réseaux peer-to-peer sont largement utilisés dans les applications de partage de fichiers, comme Bittorrent.

III. Systèmes de fichiers distribués

III.1. Définition

Un système de fichier (File System ou FS en anglais) est une façon de stocker des informations et de les organiser dans des fichiers, sur des périphériques comme un disque dur, un CD-ROM, une clé USB, etc. Il existe de nombreux systèmes de fichiers, certains ayant plus d'avantage sur d'autres, dont entre autres le NTFS (New Technology File System), FAT (File Allocation Table), etc.

Un système de fichiers distribué (Distributed File System ou DFS, en anglais) est un système de fichiers permettant le partage de données à plusieurs clients au travers du réseau. Contrairement à un système de fichier local, le client n'a pas accès au système de stockage sous-jacent, et interagit avec le système via un protocole adéquat qu'il lui permet de localiser les données avec un chemin d'accès.

Un système de fichiers distribué est donc utilisé par plusieurs machines en même temps qui peuvent ainsi avoir accès à des fichiers distants. Un tel système permet ainsi de partager des données entre plusieurs clients, pour certains répartir la charge entre plusieurs machines, et de garantir la sécurité des données par leur réplication.

III.2. Architecture

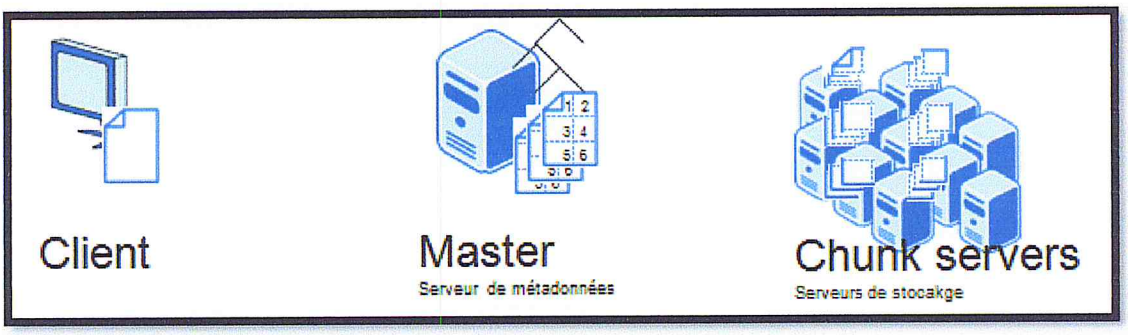


Figure 10 : Architecture d'un système de fichiers distribué [10]

Le but recherché par l'ensemble des systèmes de fichiers distribués est de permettre l'agrégation d'espaces de stockage situés sur des machines différentes de façon la

plus transparente possible pour l'utilisateur. Autrement dit, l'accès aux fichiers distants doit se faire de même manière que l'accès à un fichier local, et plus précisément, les commandes standards (ls, cp, rm, mv, etc.) qui servent respectivement à lister le contenu d'un répertoire, copier, supprimer et déplacer un fichier ou un dossier, doivent avoir le même comportement sur une architecture distribuée qu'au local [11].

Cela suppose que le système puisse connaître l'emplacement physique des données, c.à.d. sur quels serveurs le fichier se trouve. Un fichier peut être stocké sur plusieurs serveurs en même temps s'il y a du stripping (le fichier est découpé en morceaux qui sont placés sur différents serveurs) ou de la redondance (réplication). C'est pourquoi, dans tous les systèmes rencontrés il y a un serveur maître gérant ce que l'on appelle les métadonnées décrivant chaque fichier. Dans ces métadonnées, on trouve le nom, l'emplacement dans l'arborescence, l'emplacement physique, le propriétaire, les droits, les dates d'accès et de dernière modification, etc.

Lors de l'accès à un fichier, une fois que le système connaît le serveur à interroger pour accéder au contenu grâce aux métadonnées, il faut qu'il envoie une requête en direction de ce serveur. La requête arrive alors sur un autre type de serveur (serveur de stockage) qui est responsable de la gestion des fichiers locaux partagés. C'est ce serveur qui écrit ou lit physiquement les données sur le support de stockage. La grande majorité des systèmes de fichiers distribués sont sur ce modèle [11].

IV. MapReduce

IV.1. Qu'est-ce que MapReduce ?

Avec l'avènement des systèmes distribués, un certain nombre de problèmes, comme la distribution soigneuse des données et la parallélisation des tâches, ont rendu la situation difficile pour un programmeur de se concentrer sur le problème réel.

Dean et Ghemawat [DGH 04] ont vu la nécessité d'une abstraction qui aiderait le programmeur à se concentrer sur le problème réel sans avoir à se soucier des

complications du système distribué telles que : la tolérance aux pannes, équilibrage de charge, la distribution des données et la parallélisation des tâches. Et c'est exactement ce que MapReduce a été conçu pour atteindre. Un framework simple et puissant qui permet au programmeur d'écrire de simples unités de traitement en tant que fonctions map et reduce (voire d'autres fonctions). Par la suite, le framework prend automatiquement en charge de paralléliser les tâches sur un grand cluster de machines commodes. Il s'en occupe de tous les problèmes mentionnés précédemment, de la tolérance aux pannes, l'équilibrage de charge, la distribution des données et la parallélisation des tâches.

Le principe de base de MapReduce s'articule donc en deux phases :

- Dans la phase Map, le nœud racine à qui le problème est soumis, le découpe en sous-problèmes, et les délègue à d'autres nœuds (qui peuvent en faire de même récursivement). Les sous-problèmes sont ensuite traités par les différents nœuds à l'aide de la fonction Map qui a un couple (clé, valeur) associé à un ensemble de nouveaux couples (clé, valeur).
- Vient ensuite la phase Reduce, où les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avait sollicités. Celui-ci calcule un résultat partiel à l'aide de la fonction Reduce (réduction). Puis il remonte l'information à son tour.

A la fin, le nœud d'origine peut recomposer une réponse au problème qui lui avait été soumis.

Pour créer un Job[3] MapReduce, le programmeur doit donc spécifier un traitement pour une fonction map et un autre pour une fonction reduce. Cette abstraction est inspirée des primitives map et reduce du langage de programmation Lisp et beaucoup d'autres langages fonctionnels. Le framework MapReduce exécute de multiples instances de ces fonctions en parallèle. La fonction map, qui est définie par l'utilisateur, prend en entrée une paire de clé/valeur par exemple (K_1, V_1), elle la traite, puis elle génère une liste de paires clé/valeur dites intermédiaires, soit la liste de (K_2, V_2). Après, la bibliothèque MapReduce regroupe toutes les valeurs

[3] Un Job MapReduce est une itération Map/Reduce qui regroupe des tâches Maps et/ou Reduces.

intermédiaires V_2 associées à la même clé intermédiaire K_2, puis les soumettre à la fonction reduce, également définie par l'utilisateur. Le reduce prend la paire de clé K_2 et la liste des valeurs regroupées pour K_2 et traite ces valeurs selon le besoin, par exemple les agréger pour former un résultat plus fin, soit la valeur V_3.

Map: $(K_1, V_1) \rightarrow \text{liste } (K_2, V_2)$

Reduce: $(K_2, \text{liste}(V_2)) \rightarrow (K_2, V_3)$

Outre les interfaces *map* et *reduce*, il existe également d'autres interfaces standards, manipulables selon le besoin et offertes par la plupart des frameworks MapReduce : *reader* pour la lecture des données d'entrée, *combine* pour regrouper les données de sortie du *map*, *partition* pour partitionner les données intermédiaires sur les *reducers*[4].

IV.2. Architecture MapReduce

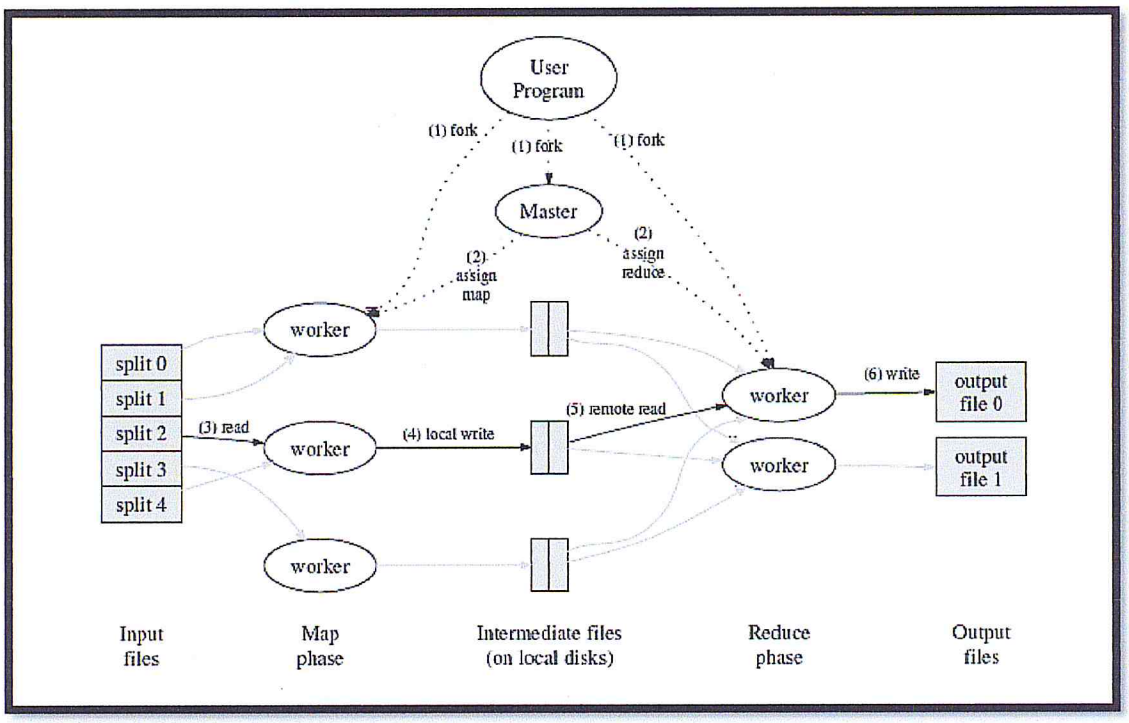


Figure 11 : Vue sur l'exécution distribuée du MapReduce [DGH 04]

[4] reducer désigne le nœud qui prend en charge une tâche reduce (respectivement un mapper pour une tâche map).

La Figure 11 donne un aperçu sur l'exécution du modèle MapReduce. L'exécution d'un job MapReduce en entier comprend les étapes suivantes [DGH 04] :

1. La bibliothèque MapReduce divise d'abord les fichiers de données d'entrée en M blocs (splits) de taille fixe, c'est typiquement entre 16 et 64 Mo par bloc. Ces M blocs sont ensuite transmis aux machines participantes du cluster. Habituellement, il y a 3 exemplaires (répliques) pour chaque bloc pour des raisons de tolérance aux pannes. Après, la bibliothèque démarre de nombreuses copies ou instances du programme utilisateur sur les nœuds du cluster.

2. L'un des nœuds du cluster est spécial, le maître (master en anglais). Les autres sont des workers (travailleurs) à qui le maître assigne des tâches de travail. Il y a M tâches maps et R tâches reduces, à attribuer. R est défini par une configuration spécifiée par l'utilisateur du programme. Le maître sélectionne M workers parmi les workers inactifs (libres) et attribue à chacun une tâche map. Une fois les tâches maps ont généré les données intermédiaires en sortie, le maître attribue ensuite les tâches reduces là aussi aux workers inactifs. Toutes les tâches maps doivent se terminer avant qu'une tâche reduce commence ou peut commencer. C'est parce que les tâches reduces prennent en entrée les données de sortie que génèrent les tâches maps qui peuvent avoir besoin d'être consolidées.

3. Un worker, à qui on a assigné une tâche map, lit le contenu du bloc de données d'entrée correspondant. Il analyse les paires clé/valeur à partir des données d'entrée et soumet chaque paire à une instance de la fonction map définie par l'utilisateur. Les paires clé/valeur intermédiaires générées par les fonctions maps sont stockées dans un buffer de mémoire sur les machines respectives qui les exécutent.

4. Les paires sur le buffer mémoire sont périodiquement écrites sur le disque local et distribuées sur R régions par la fonction de partitionnement. La bibliothèque fournit une fonction de partitionnement par défaut, mais l'utilisateur est autorisé à manipuler (modifier) cette fonction pour se permettre un partitionnement personnalisé. Les emplacements de ces paires de données intermédiaires sur le

disque local sont renvoyés au maître. Le maître transmet ensuite ces emplacements aux workers reduces (reducers).

5. Quand un worker reduce est notifié par le maître à propos de ces emplacements, il utilise des appels aux procédures à distance (RPC) pour lire les données à partir des emplacements désignés. Quand un worker reduce finie de lire toutes les données intermédiaires, il les trie par les clés intermédiaires de telle sorte que toutes les occurrences de la même clé sont regroupées. Si la quantité de données intermédiaires est trop grande pour la tenir en mémoire, un tri externe est utilisé c.à.d. en se servant de la sauvegarde sur disque local. Une fois de plus, l'utilisateur est autorisé à personnaliser les fonctions de trie et de groupement que la bibliothèque MapReduce offre par défaut.

6. Ensuite, le worker reduce parcourt les données intermédiaires triées et pour chaque clé intermédiaire unique rencontrée, il transmet la clé et l'ensemble des valeurs intermédiaires correspondant aux fonctions reduces définies par l'utilisateur. Les données de sortie de la fonction reduce sont ajoutées à un fichier de sortie final.

IV.3. Exemple MapReduce

Pour bien comprendre le MapReduce, nous allons l'illustrer avec un exemple. Voici ci-dessous les algorithmes pour les fonctions map et reduce, qui servent à catégoriser des entiers à pair ou impair et sommer les nombres de chaque catégorie pour le résultat final.

	Algorithme 1 : Fonction map
1	Fonction map (clé : Chaîne, valeurs : Liste d'entiers) : (Chaîne, Entier)
2	Var
3	reste, valeur : Entiers
4	Début
5	Pour chaque valeur dans valeurs Faire

```
6                   reste ← valeur mod 2
7                   Si reste = 0 Alors
8                           Retourner ("pair", valeur)
9                           /* ou Emettre Intermédiairement ("pair", valeur) */
10                  Sinon
11                          Retourner ("impair", valeur)
12                  FinSi
13          FinPour
14  FinFonc
```

	Algorithme 2 : Fonction reduce
1	Fonction reduce (clé : Chaîne, valeurs : Liste d'entiers) : (Chaîne, Entier)
2	Var
3	somme : Entier
4	Début
5	somme ← 0
6	Pour chaque valeur dans valeurs Faire
7	somme ← somme + valeur
8	FinPour
9	Retourner somme
10	/* ou Emettre somme */
11	FinFonc

Dans ce cas, soit la liste des entiers : [1, 2, 3, 4, 5, 6, 7, 8]. Le résultat final sera comme suit :

pair, 20
impair, 16

Voici la Figure 12 qui montre plus de détails sur le déroulement de l'exemple.

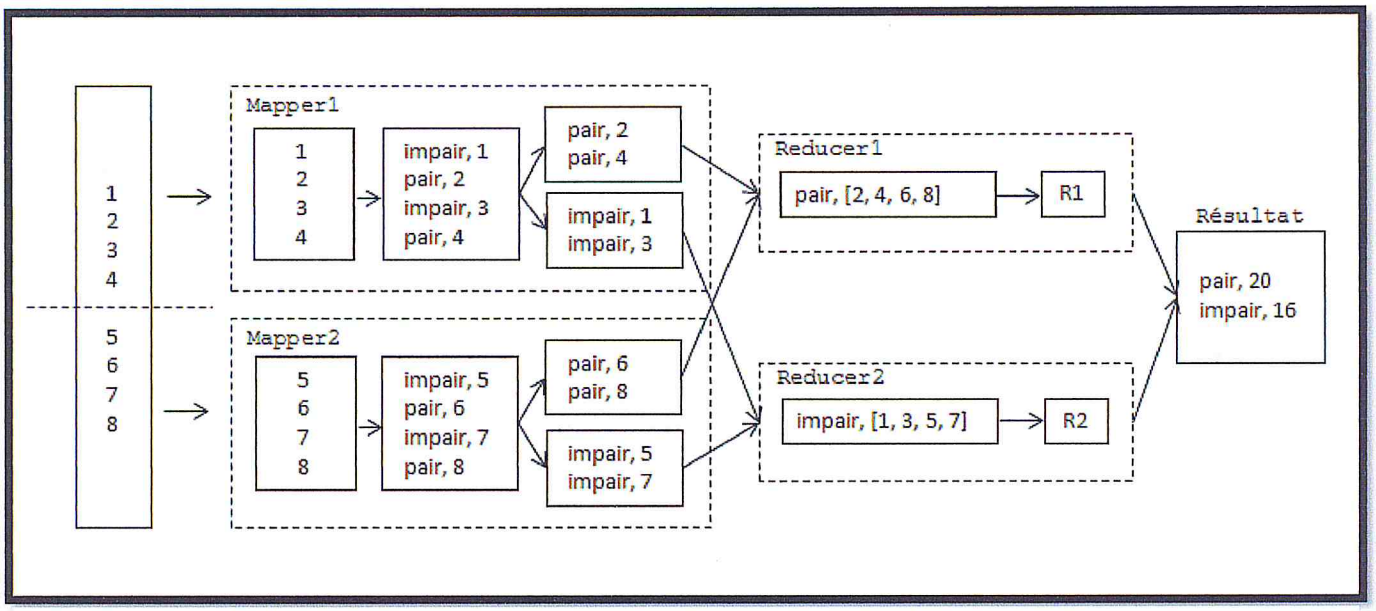


Figure 12 : Schéma d'exécution de l'exemple de catégorisation pair/impair.

IV.4. Implémentations MapReduce

Il y a plusieurs implémentations du paradigme MapReduce dans différents langages de programmation (C++, Java, Python, etc.) et par de nombreux organismes (Google, Apache, Nokia, etc.). Les plus utilisées aujourd'hui sont : Google MapReduce, Apache Hadoop et Disco MapReduce.

IV.4.1. Google MapReduce

Google MapReduce a été développé en 2004 chez Google Inc. [DGH 04]. Il a été conçu pour être en mesure d'exécuter des processus manipulant de la volumétrie importante de données pour le moteur de recherche de l'entreprise. Il est développé pour leurs propres applications, le système est donc propriétaire. Il ne paraît pas être publiquement disponible et il n'y a que peu de détails de leurs connaissances qui sont publiés dans ce domaine. Il est considéré comme le premier prototype d'un système MapReduce distribué.

Le noyau du système est écrit en C++ et il tourne sur des milliers de machines dans les centres de données de l'entreprise à travers le monde.

Google MapReduce opère avec un système de fichiers distribué propriétaire aussi (Google File System ou GFS) pour répondre aux besoins de stockage de gros volumes de données des applications Google, notamment pour tout ce qui concerne ses activités de recherche sur le web.

IV.4.2. Apache Hadoop

Le système Apache Hadoop est un projet de la distribution MapReduce, il appartient à la fondation Apache. Comme il est publié sous la licence d'Apache, son code source est disponible à télécharger. L'un des objectifs du projet est la capacité du système d'être indépendant de toute plateforme (portabilité) puisqu'il est implémenté en Java.

La fondation Apache a développé aussi un système de fichiers distribué propre à elle (Hadoop Distributed File System ou HDFS) pour la gestion, sur clusters, des quantités importantes de données que son Hadoop MapReduce traite.

Hadoop est utilisé dans de nombreuses applications commerciales. En 2008, Yahoo! a commencé à construire des parties d'indexe de son moteur de recherche à l'aide de Hadoop [12]. Il est également utilisé dans le moteur de recherche de produits A9.com d'Amazon et l'Amazon Elastic Compute Cloud (EC2).

IV.4.3. Disco MapReduce

Le système distribué Disco MapReduce [13] a été développé en 2008 par le Centre de Recherche de NOKIA (NRC) à Palo Alto et destiné à résoudre des problèmes réels et complexes dont les traitements impliquent de grands volumes de données qui sont gérés par un système de fichiers distribué (Disco Distributed File System ou DDFS) que le framework offre. Le centre de recherche donne libre-accès à son code source et il peut être téléchargé depuis le site officiel du projet Disco.

Dans le framework Disco, les fonctions map et de reduce doivent être spécifiées en langage Python, mais le noyau du système est écrit dans un langage de programmation fonctionnelle, qui est l'Erlang.

Actuellement, Disco est utilisé par une variété d'industries, comme Allston Trading, Chango, Zemanta, etc.

VI. Conclusion

Dans ce chapitre, nous avons présenté les concepts de base nécessaires à la compréhension de l'environnement MapReduce. La première section donne un aperçu général sur les systèmes de calcul distribué et parallèle. La deuxième section discutait sur les systèmes de fichiers distribués requis pour la gestion de grands volumes de données que les systèmes distribués traitent. Ainsi, un type spécial de système distribué, le système MapReduce, est présenté dans la dernière section.

L'avènement de ce genre d'environnements, qui permet la parallélisation dans l'exécution des processus et au même temps de traiter de grandes quantités de données (téraoctets voire pétaoctets), constitue une réelle opportunité pour développer et faire migrer les processus ETL fonctionnant dans un modèle classique vers des environnements basés sur le paradigme MapReduce.

Dans le chapitre qui suit, nous nous intéressons de près au processus ETL dans un environnement MapReduce en se basant particulièrement sur un travail existant qui a été fait récemment (2011) auquel nous apporterons notre contribution. On parle du Framework ETLMR.

Partie II :

Processus ETL dans un environnement MapReduce

Chapitre IV :

Prototype ETLMR

I. Introduction

Dernièrement, la prise en charge de grandes quantités de données est devenue le souci majeur des entreprises. Il y a une demande excédentaire ces jours-ci pour des outils ETL qui traitent sur de grandes masses de données de manière efficace et performante. La parallélisation s'avère la technologie clé pour atteindre les performances souhaitées.

La technologie MapReduce offre la flexibilité, l'évolutivité et un intérêt considérable pour l'appliquer à la parallélisation de l'ETL. Cependant, MapReduce est un framework générique et manque d'un support direct pour les constructions dimensionnelles de haut niveau, telles que les schémas en étoiles et les schémas en flocons de neige. Il est donc toujours pas facile d'implémenter un programme ETL parallèle avec du MapReduce, cela nécessite beaucoup de rigueur.

Dans ce chapitre, nous allons présenter un prototype d'un ETL basé sur la solution parallèle MapReduce et qui directement dispose des constructions spécifiques au DW/ETL. Nous parlons de l'ETLMR.

II. Framework ETLMR

Récemment, Liu et *al.* (2011) ont mis en œuvre un framework nommé ETLMR [ETLMR 11] et implémenté en Python sous l'environnement Disco pour accélérer l'exécution du processus ETL considéré très lourd en termes de volumétrie de données et de complexité des tâches extraction, transformation et chargement. La contribution principale dans l'ETLMR étant l'intégration des concepts décisionnels comme les dimensions, les faits, le schéma en étoile, le schéma en flocon de neige, etc.

L'ETLMR est un outil dont l'utilisation est à base de script Python. Il a été conçu comme étant une « Librairie Dimensionnelle d'Application » ou « Framework Dimensionnel » migrée en Disco MapReduce pour les programmeurs et les développeurs des ETL afin de pouvoir exploiter tout le noyau en code et ne pas se limiter seulement sur quelques options d'une interface graphique quelconque. Notons

que pour les développeurs c'est plus intéressant d'exprimer leurs solutions en quelques lignes de code que de dessiner des schémas et remplir des propriétés via des boites de dialogues.

La complexité du MapReduce dans l'ETLMR est cachée à l'utilisateur final. Ce dernier ne précise que les transformations à effectuer et les déclarations des sources et des cibles dans un fichier de configuration Python. Ainsi, il est facile de développer un programme ETL parallèle avec seulement quelques lignes de code.

III. Architecture ETLMR

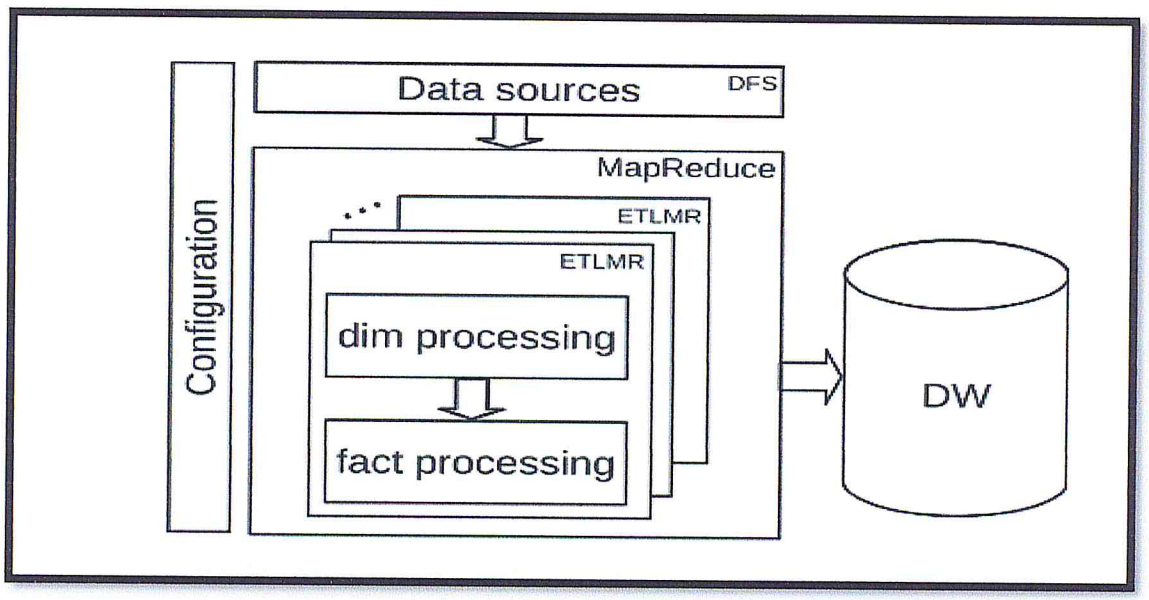


Figure 13: Architecture du framework ETLMR [ETL/MR 11]

La Figure 13 ci-dessus présente l'architecture générale du système ETLMR. Le flux ETL dans l'ETLMR comprend deux phases séquentielles : traitement de dimensions et traitement de faits. Ces phases s'exécutent séparément, chacune dans un job MapReduce différent (démontré par la Figure 14).

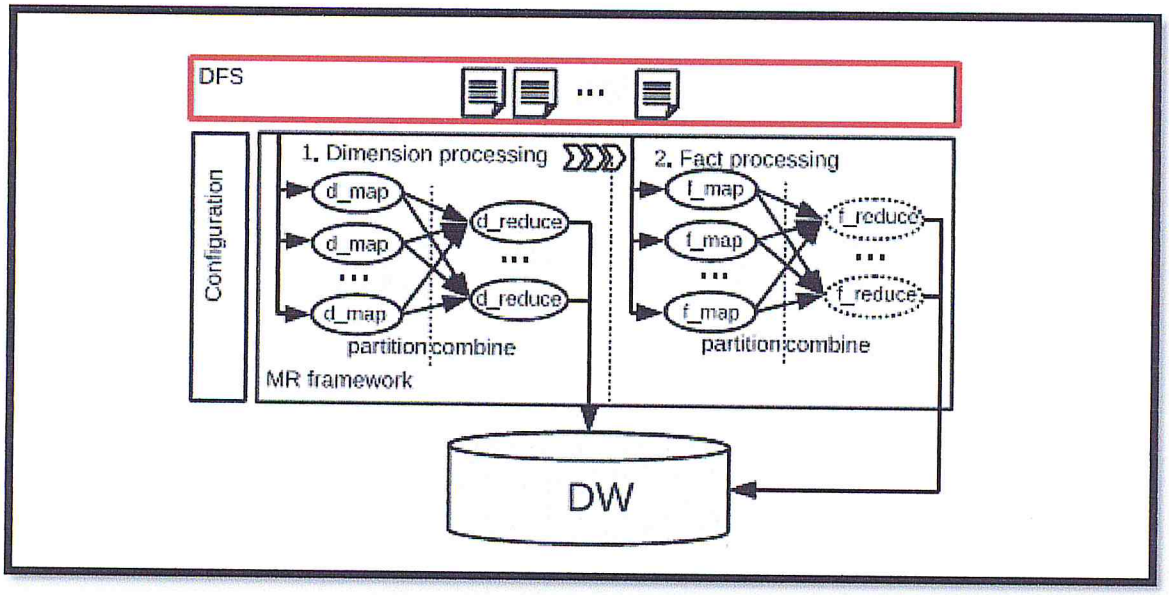


Figure 14 : Flux de données ETLMR en MapReduce [ETLMR 11]

Un job MapReduce se compose d'un certain nombre de tâches (instances) map/reduce fonctionnant en parallèle sur plusieurs nœuds du cluster. Une tâche lit les données à partir des fichiers d'entrée qui sont dans le système de fichiers distribué DFS puis les traite et les insère dans les tables de dimensions et/ou tables de faits de l'entrepôt de données DW.

Voici l'algorithme traduit du [ETLMR 11] qui détaille tout le processus ETLMR :

	Algorithme 3 : Processus ETL dans un framework MapReduce
1)	Partitionner et préparer l'ensemble des données d'entrée ;
2)	Lire les paramètres de configuration et procéder à l'initialisation ;
3)	Lire les données d'entrée et les transmettre à la fonction Map par les MapReaders ;
4)	Traitement des dimensions et chargement dans l'entrepôt de données ;
5)	Synchroniser les dimensions à travers les nœuds du cluster, si applicable ;
6)	Préparer le traitement de faits (connecter aux dimensions)
7)	Lire les données d'entrée pour le traitement de faits et appliquer les transformations sur les données dans les mappers ;
8)	Chargement en vrac des données de faits vers l'entrepôt de données.

Les opérations des lignes 2 à 4 et 6, 7 sont des étapes MapReduce (c.à.d. traités en MapReduce) responsables de l'initialisation, de l'invocation des jobs MapReduce pour le traitement de dimensions/faits et de la sauvegarde des informations traitées. Les étapes représentées par la ligne 1 et 5 ne sont pas traitées en MapReduce, elles sont utilisées respectivement pour la préparation de l'ensemble des données d'entrée (voir partitionnement et distribution) et la synchronisation des dimensions à travers le cluster. Cette dernière est applicable si aucun système de fichiers distribué (DFS) n'est installé.

Le processus ETLMR se déroule en 4 phases principales : préparation et distribution des sources sur le DFS, configuration, traitement de dimensions et traitement de faits.

III.1. Préparation et distribution des sources

Dans l'ETLMR, et avant qu'un job MapReduce puisse démarrer, les sources de données à partir des systèmes de stockage hétérogènes doivent être partitionnées à de multiples pièces (blocs) de tailles approximativement égales, puis distribuées sur l'ensemble des nœuds sur le cluster (voir encadré en rouge sur la Figure 14) pour pouvoir les lire localement et d'éviter d'augmenter de la communication réseau superflue qui peut être due à la centralisation des sources de données. Malheureusement, cette étape ne représente pas une tâche MapReduce dans le processus ETLMR malgré son importance et le temps qu'elle pourrait nécessiter à cause de la volumétrie importante des sources.

Si l'ensemble des données sources est initialement partitionné (pré-partitionné), tels plusieurs fichiers de données à partir de systèmes hétérogènes, les parties (blocs) sont directement traitées et combinées avec du MapReduce. ETLMR dispose de différents MapReaders pour l'intégration des données à partir de différents systèmes sources.

Si les données sources ne sont pas pré-partitionnées, tel un fichier à données intensives (Big Data), Liu et *al.* ont implémenté des MapReaders qui supportent de la lecture de données à partir d'une source singulière (centralisée), c'est de la lecture sélective des données c.à.d. qu'on se partage les données à la volée de l'extraction

(partitionnement à la volée). Cette dernière n'exige donc pas que les données soient partitionnées avant d'être traitées. D'un autre côté, Dean et Ghemawat suggèrent d'utiliser plusieurs traitements MapReduce pour pouvoir préparer et distribuer l'ensemble des données sources, et de traiter les sous-ensembles résultants [ETLMR 11].

III.2. Configuration

ETLMR facilite l'utilisation d'un ETL parallèle par l'emploi d'un fichier de configuration, sur lequel sont stockés tous les paramètres d'exécution, y compris les paramètres concernant les sources de données, les tables cibles, les méthodes et les clés du partitionnement, le nombre de mappers et reducers. Ainsi que la définition des tables de dimensions et les tables de faits. Un objet est créé pour chacune des tables cibles dans une seule instruction où le nom de la table, les noms des attributs,... sont définis. ETLMR supporte différents types de dimensions, normalisées et celles non-normalisées.

Le fichier de configuration permet aussi de spécifier les transformations à appliquer aux données et le niveau de parallélisation (appelé aussi le niveau de performance) en spécifiant le nombre de tâches maps et reduces à utiliser. Le tableau 2 ci-dessous résume les principaux paramètres de configuration.

Paramètres	**Description**
Dim_i	Définition d'une table de dimension, i = 1, …, n
$Fact_i$	Définition d'une table de faits, i = 1, …, m
$Set_{bigdim}(dim_i)$	Ajouter une dimension à la liste des *bigdims*
$Dim_i(a_0, a_1, \ldots, a_n)$	Définition des attributs persistants de la source à la dimension dim_i *(mappage)*
nr_reduce	Nombre de reducers
nr_map	Nombre de mappers

Tableau 2: Les paramètres clés de la configuration ETLMR

Ces paramètres fournissent aux utilisateurs la possibilité d'une configuration flexible des tâches pour être plus efficace. Par exemple, si l'utilisateur connaît auparavant qu'une dimension est une dimension aux données intensives, il peut l'ajouter à la liste des *bigdims*, et du coup, une méthode de traitement appropriée peut être choisie pour réaliser de meilleures performances et un équilibrage de charge.

III.3. Traitement de dimensions

Lors du traitement de dimensions, ETLMR utilise des instances parallèles de la fonction map du job MapReduce pour appliquer les transformations définies par l'utilisateur sur l'ensemble des données sources. Ainsi, la fonction map est utilisée pour relever les attributs significatifs (projection) pour les différentes dimensions. Les sous-ensembles de données qui résultent sont traités ensuite par des instances parallèles de la fonction reduce du job MapReduce pour les insérées dans les tables de dimensions.

ETLMR offre plusieurs méthodes pour ce traitement. La méthode la plus simple est une-dimension-une-tâche (en anglais, One Dimension One Task ou ODOT). Avec ODOT, il n'y a qu'une et qu'une seule tâche reduce pour chaque table de dimension (voir Figure 15). Cela rend plus simple de gérer les données dupliquées (doublons), particulièrement les clés dupliquées puisque toutes les insertions seront faites par une seule tâche reduce.

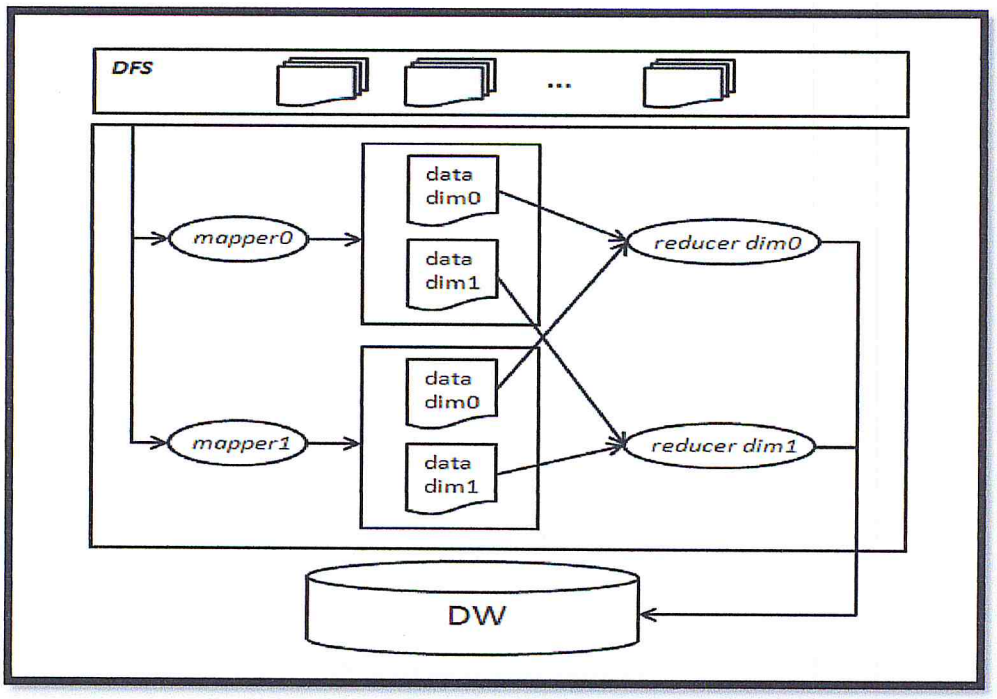


Figure 15 : ODOT

Cependant, cette méthode n'est pas optimale en termes de performance dans le cas où certaines dimensions sont des dimensions à données intensives. Dans ce cas, une autre méthode peut être envisagée : la méthode une-dimension-toutes-les-tâches (en anglais, One Dimension All Tasks ou ODAT). Avec cette méthode, toutes les tâches reduces traitent des données pour toutes les dimensions (voir Figure 16 ci-après). D'un côté cela équilibre les charges mais d'un autre côté cela peut entrainer des problèmes de duplication de données, particulièrement les clés d'unicité. Pour remédier à cela, ETLMR dispose d'une technique (moulinette) « post-fixing » avec laquelle les problèmes de duplications seront corrigés automatiquement après que le job MapReduce soit terminé.

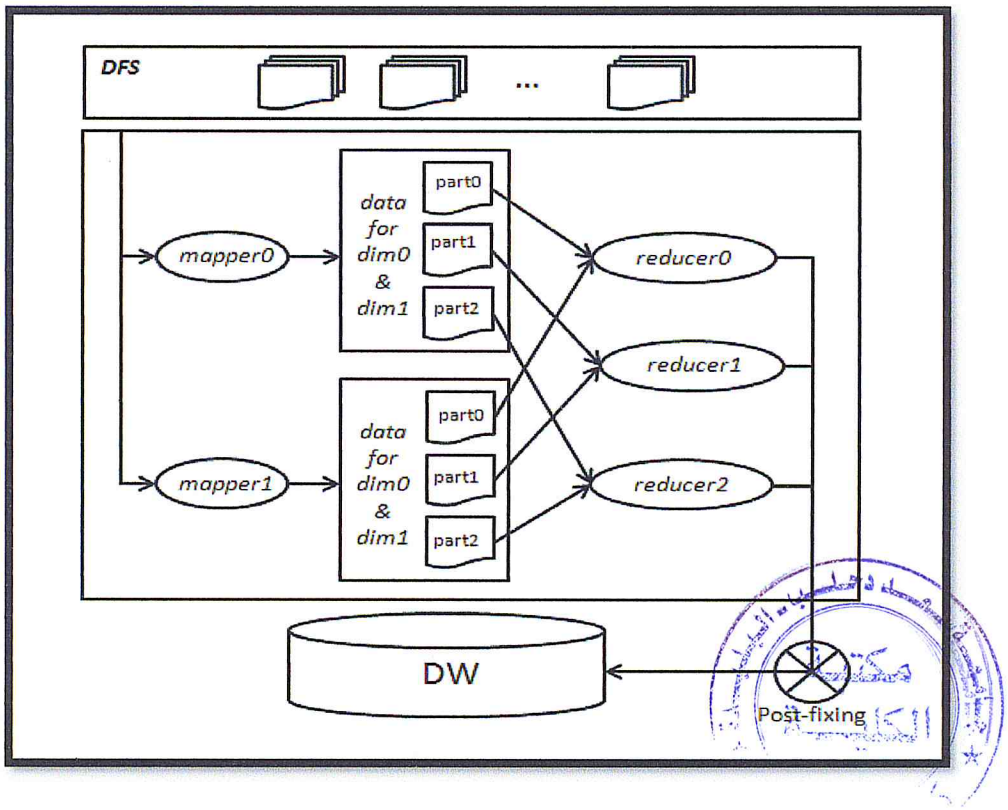


Figure 16 : ODAT

III.4. Traitement de faits

Lors du traitement d'une table de faits, ETLMR recherche dans la phase map les clés des dimensions concernées pour répondre à l'intégrité référentielle. Ensuite, il agrège les données de faits dans la phase reduce si nécessaire et les charge en vrac dans l'entrepôt de données. Plusieurs tâches le font donc en parallèle par le biais d'un job MapReduce (voir « 2. Fact processing » sur la Figure 14).

IV. Caractéristiques et points forts

Il y a plusieurs outils ETL sur le marché, la majorité travaille avec des interfaces graphiques où l'utilisateur définit les flux ETL et les différentes opérations visuellement. Cependant, les utilisateurs développeurs ont du mal à exprimer leurs solutions avec les composants standards trouvés sur l'interface graphique de l'outil

ETL, pour eux c'est plus intéressant de s'exprimer en quelques lignes de code que de remplir des boites de dialogues. ETLMR est un framework qui répond fortement à ce besoin pour gagner du temps en développement et en exécution, et pour ne pas se limiter aux options de l'interface graphique mais exploiter tout le noyau du framework en code.

ETLMR est open-source, léger en termes de taille et facile à utiliser (avec seulement un fichier de configuration qui comprend tous les paramètres d'exécution).

L'ETLMR est basé sur l'utilisation de MapReduce, héritant ainsi de tous les avantages qui découlent de cette technologie (extensibilité, transparence et tolérance aux pannes, etc.). Un autre point fort dû à l'utilisation de MapReduce est de pouvoir être ouvert à plus de nœuds sur le cluster sans avoir à modifier le framework mais seulement ses paramètres de configuration.

ETLMR supporte des constructions dimensionnelles spécifiques au DW/ETL telles qu'une table de dimension, une table de faits, une dimension à données intensives, ..., et d'autres constructions de haut niveau telles que le schéma en étoile et le schéma en flocon de neige, etc. Ainsi, l'ETLMR traite facilement sur les concepts décisionnels, contrairement à d'autres outils ETL basés sur le paradigme MapReduce tels que Pig et Hive qui se manipulent avec des langages à base de requêtes similaires à celles du langage SQL. Ils ne supportent pas les constructions dimensionnelles de façon directe. Il est possible de construire et de traiter un schéma en étoile ou en flocon de neige avec ces outils, mais cela est compliqué et nécessite beaucoup de code (requêtes).

Différents MapReaders sont implémentés en ETLMR pour lire les données à partir de différents systèmes de stockage, tels un reader pour fichiers textes et un DBMS (Database Management System) reader supportant une requête SQL définie par l'utilisateur.

Les résultats des tests et des expériences sur l'ETLMR démontrent une performance considérable qui le favorise fortement face à d'autres outils d'entreposage basés sur le paradigme MapReduce, en particulier Pentaho Data Integration (PDI).

V. Points faibles

ETLMR est un framework destiné aux programmeurs ou développeurs des outils ETL et non pas à n'importe quel utilisateur non informaticien. Il faut avoir une maîtrise du langage Python pour pouvoir l'utiliser. Cet inconvénient est un handicap pour les utilisateurs non-programmeurs (les gestionnaires par exemple).

Beaucoup de tests on été faits particulièrement sur les codes ETLMR où nous avons découvert la rigidité de ce prototype. Le framework ETLMR disponible (mis en téléchargement) ne semble pas convenant pour un autre exemple que celui démontré par l'équipe Liu et *al.* malgré qu'il soit déclaré open-source et très générique, et qu'en intervenant seulement et légèrement sur le fichier de configuration fera l'affaire.

Les codes ETLMR sont très rattachés à l'exemple traité en l'occurrence le projet sur les erreurs détectées sur un échantillon de pages web par rapport aux dimensions serveurs, dates, tests, pages, etc. inspiré du projet « Building a Web Warehouse for Accessibility Data » que Liu et *al.* ont réalisé en 2006 pour l'Observateur Européen d'Accessibilité sur Internet (EIAO, European Internet Accessibility Observatory). En outre, d'autres fonctionnalités affirmées faites sont restreintes du code source du framework, le DBMS-SQL reader en est l'exemple. Liu et *al.* ont implémenté des DBMS MapReaders qui supportent des requêtes SQL définies par l'utilisateur, mais puisque lire à partir d'une BDR (base de données relationnelle) ne faisait pas partie de leur exemple, ces MapReaders ont été omis. Par conséquent, le framework ne contient que les fonctions nécessaires pour faire marcher l'exemple de démonstration. Ensuite, concernant les fonctions de transformation, nous y trouvons peu, précisément que celles de conversion (de type, de temps et de codage).

L'extraction efficiente consiste à lire les données à partir d'une source de type quelconque (structuré, semi-structuré ou non-structuré). ETLMR traite sur des sources de différents formats (BD et fichiers plats), mais non pas sur une source de type semi-structuré telle que le format de données XML qui est très important. Nous ne pouvons pas nier l'évidence que l'XML est devenu le langage universel d'échange

de données informatiques, qu'il s'agisse de les stocker, de les échanger, de les traiter ou de les afficher.

Liu et *al.* se sont penchés surtout sur les étapes du traitement de dimensions et de faits sans donner de l'importance à celle du partitionnement et préparation de l'ensemble des fichiers sources (respectivement des tables sources) potentiellement volumineux, et leur distribution sur le cluster via un DFS. Cette phase semble être faite au moment de la génération des données de tests, selon Liu et *al.*, puisque MapReduce opère souvent sur plusieurs petits fichiers que sur un seul ficher à données intensives, les données de tests sont générées, partitionnées et sauvegardées dans un ensemble de petits fichiers qui servent d'entrée pour les expériences. De plus, cette phase qui sert à préparer et distribuer les données sur le cluster, elle représente une étape non-MapReduce malgré son importance, et son temps d'exécution n'est aucunement compté ou pris par considération lors des tests et les expérimentations, mais plutôt négligé malgré qu'elle prenne un coût important dans le cas des données volumineuses.

VI. Conclusion

L'outil exposé dans ce chapitre représente un axe de recherche concernant le processus ETL parallèle dans un modèle MapReduce. ETLMR supporte directement les constructions spécifiques au DW/ETL, et assure de l'évolutivité et de la transparence grâce à l'utilisation de MapReduce. Cependant, nous avons remarqué que plusieurs facteurs et contraintes compliquent la mise en œuvre de ce prototype, tels que sa rigidité et ses lacunes liées à la phase initiale du processus. L'ETLMR de Liu et *al.* ne prend pas en charge la préparation des données et leur distribution sur le cluster. Nous n'avons trouvé aucune contribution qui s'intéresse à cet aspect. Dans le chapitre qui suit, nous allons présenter les détails de conception de notre contribution à ce propos.

Partie III :

Mise en œuvre du système

Chapitre V :

Conception du système

I. Introduction

Au cours du chapitre précédent, nous avons pu constater que malgré que l'ETLMR, que nous avons retenu pour étude, offre au développeur la possibilité d'implémenter et d'exécuter rapidement un ETL parallèle en MapReduce, il ne couvre pas une phase d'initialisation complète, cette phase représente le maillon faible dans la chaîne ETLMR.

L'étude du prototype ETLMR nous a permis donc d'y voir plus clair sur certaines fonctionnalités qui représentent en fait les points faibles et la rigidité de ce prototype. Cela constitue des idées et de la matière première permettant ainsi de nous orienter en termes de contribution à implémenter notre propre outil pour combler ses lacunes.

L'étude détaillée et la conception de notre système seront le sujet de ce chapitre.

II. Contribution

Nous nous intéressons dans ce chapitre à la conception détaillée de notre système. La problématique soulevée est la suivante : bien que l'ETLMR soit capable de satisfaire les besoins de décideurs au niveau de la réduction du temps de traitement des données volumineuses, il n'offre pas une phase initiale complète qui gère l'aspect de préparation et de partitionnement des données pour leur distribution sur un système de fichiers répartis. Le prototype proposé par Liu et *al.* ne s'intéresse donc pas à la distribution des données sur différents nœuds du cluster DFS, ni classiquement (le cas traditionnel et non parallèle), ni en MapReduce (le cas parallèle).

L'objectif de notre travail serait donc de prendre en charge les manques identifiés sur l'ETLMR dans le but de voir comment les améliorer, et résultant ainsi en un outil de distribution complet basé sur le modèle MapReduce. Il s'agit d'un processus d'extraction, de préparation et de partitionnement de données en parallèle dans un modèle MapReduce pour leur mise en place dans un système de fichiers distribués.

Pour bien concevoir notre système, nous avons mis au point ce processus dans un modèle classique qui est séquentiel et non pas parallèle afin de pouvoir mener à bien

sa décortication dans le modèle MapReduce. Ainsi, nous pourrons comparer les deux approches par la suite en termes de performances.

III. Vue d'ensemble

Pour généraliser notre démarche, nous proposons une architecture générale de notre système que nous pourrons l'adapter par la suite à n'importe quel modèle (classique ou MapReduce).

III.1. Architecture

La Figure ci-dessous illustre l'architecture globale de notre système qui s'agit d'un processus ETL de distribution de données.

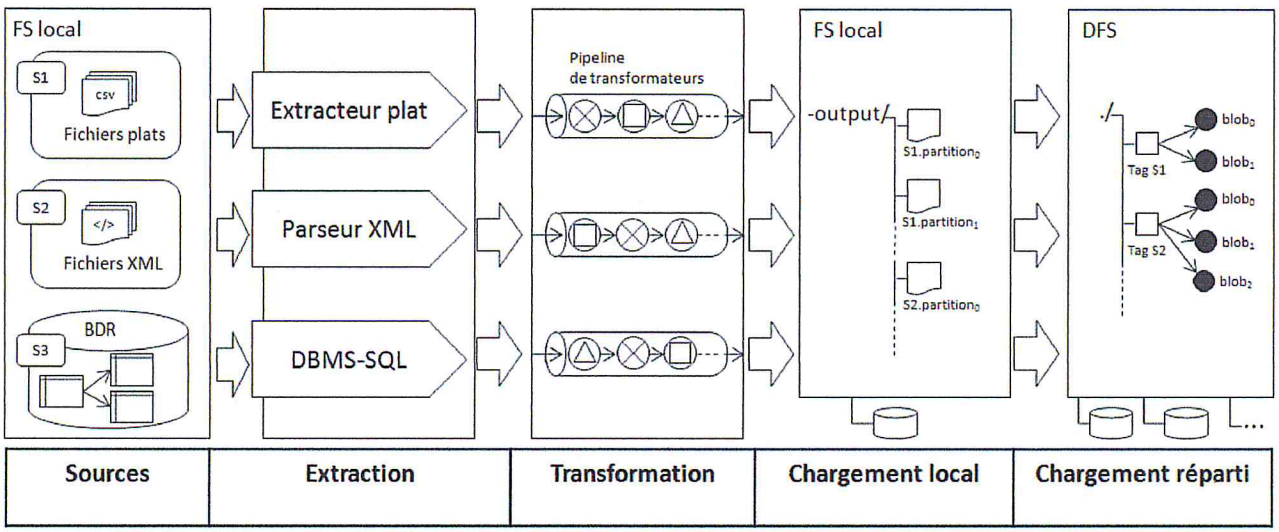


Figure 17: Architecture générale du système

Le schéma de la figure présente les différentes étapes essentielles à la définition de notre système. Il s'agit d'opérer principalement en 4 étapes successives :

- Extraction de données à partir des sources diverses et hétérogènes ;
- Transformation et homogénéisation des données sources préalablement extraites
- Chargement des données issues des étapes précédentes dans différents fragments (partitions) dans le système de fichiers FS local à titre temporaire.

- Chargement et distribution des partitions de données stockées dans l'espace local temporaire vers le système de fichiers répartis DFS.

III.2. Description

III.2.1. Sources de données

L'architecture générale d'un tel système comporte des sources de données à partir desquelles nous extrayons les données. Ces sources sont nombreuses et variées vu qu'elles proviennent de différents systèmes sources qui sont souvent hétérogènes. On ne peut pas parler des systèmes sources sans évoquer l'hétérogénéité. Cette hétérogénéité peut provenir du format ou de la structure des sources (sources structurées: BDR, sources semi-structurées: documents XML, ou non structurées : fichiers plats). Notre système traite cette problématique dans différents niveaux du processus, comme nous le verrons par la suite.

III.2.2. Extraction des données

Il s'agit de la toute première étape du processus ETL de distribution de données. Le système interroge les sources de données de production pour capturer et collecter les données utiles. L'extraction permet ainsi de préparer les données pour les léguer aux autres étapes du processus en vue d'autres manipulations. A notre sens, cette partie joue le rôle d'intermédiaire entre les sources de données et les autres parties du processus. Notre système est compatible avec différents systèmes sources et possède des extracteurs (procédures distinguées et personnalisées pour la collecte des données) adaptés à ces systèmes. Comme indiqué sur le schéma de la Figure 17, le système opère avec trois extracteurs différents :

- Un Extracteur plat, qui vise à extraire les données à partir des fichiers plats au format CSV.
- Un Parseur XML, qui analyse syntaxiquement le schéma XML de la source de données et parcourt ses éléments pour renvoyer des enregistrements de données.

- Un DBMS-SQL, c'est un exécuteur de requêtes SQL qui assure donc un service de DBMS en se connectant à la BDR concernée, exécutant l'ordre SQL et renvoyant les tuples de données qui résultent.

III.2.3. Transformation des données

C'est la deuxième étape du processus de distribution de données. Cette étape est très importante et se situe au cœur du processus. Elle assure en réalité plusieurs tâches qui garantissent la fiabilité des données et leurs qualités. En fait, il s'agit d'une suite d'opérations définie par l'utilisateur dans un premier temps et qui est nécessaire à la modification des données. Cette suite est représentée dans le schéma de la Figure 17 par un pipeline dans lequel les données seront exposées aux différentes transformations une par une (séquentiellement) pour les traiter et les unifier à respecter le schéma de données souhaité. Cette phase assure l'harmonisation des données.

III.2.4. Chargement local des données

C'est une phase mécanique dans le processus, elle permet d'alimenter des cibles dans le FS local par des données issues des phases d'extraction et de transformation précédemment réalisées. Les cibles sont des partitions des sources de données traitées. Selon quelques critères, il permet d'identifier les données et les charger dans des partitions précises. Le mécanisme de chargement est donc sélectif. Au niveau physique et dans le FS local, une partition est nativement représentée par un fichier plat pour le stockage de données.

III.2.5. Chargement distribué des données

C'est la phase finale et la plus mécanique dans notre système. Elle sert à pousser les données vers le DFS. Pour cela, nous faisons appel à l'opération de pousse de données « PUSH » qui est une des fonctionnalités du DFS. Les partitions qui en résultent du chargement local et qui sont sauvegardées dans l'espace temporaire, seront donc copier et répliquer dans les serveurs du DFS et seront appelées « BLOBS ». Chaque groupe de blobs sera identifié par un tag (étiquette) défini par

l'utilisateur. Un tag représente le nom logique approprié à l'élément localisateur de blobs.

IV. Conception

Nous visons dans cette section les détails des différentes phases de notre processus qui n'ont pas été mis en évidence dans la conception globale de notre système.

IV.1. Extraction

Dans une première étape, le processus ETL de distribution de données extrait les données des systèmes sources. Le rôle de l'extraction est de transférer et dériver les données depuis une ou plusieurs sources de données hétérogènes vers les autres étapes du processus. Deux besoins fondamentaux sont à satisfaire : (1) il faut tenir compte tout d'abord de la diversité des sources et adapter le mécanisme d'extraction pour ces dernières. (2) un autre besoin est à satisfaire, c'est celui du besoin de l'homogénéisation de la représentation des données (préparatifs pour la phase de transformation)

IV.1.1 Structures sources

Les sources de données de production de l'entreprise se présentent sous différentes structures et différents formats vu qu'elles sont issues de gestionnaires différents et hétérogènes. La compréhension de la structure de la source est nécessaire pour pouvoir mener à bien l'extraction des données à partir de cette dernière. Dans notre cas, nous étudierons les sources de données suivantes : les fichiers plats au format CSV, les fichiers XML et les BDR.

IV.1.1.1. Fichier au format CSV

CSV est un format de fichier simple (plat) qui est largement utilisé par les scientifiques et les applications d'entreprise [14]. Le mot CSV est l'acronyme de « Comma Separated Values », « Valeurs Séparées par des Virgules » en français. C'est un format informatique ouvert représentant des données tabulaires sous forme

de valeurs séparées par des virgules. Ce format est toutefois assez populaire parce qu'il semble relativement facile à gérer.

Un fichier CSV est un fichier texte, par opposition aux formats dits « binaires », chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau.

Une ligne est une suite ordonnée de caractères terminée par un caractère de fin de ligne « \n ». Les séparateurs (délimiteurs) peuvent variés (tabulation « \t », point-virgule « ; », virgule « , », etc.), où la virgule « , » restera celui par défaut. La première ligne est toujours réservée aux colonnes et le reste des lignes pour les enregistrements de données.

Exemple :

Sexe, Prénom, Année de naissance
M, Alphonse, 1932
F, Béatrice, 1964
F, Charlotte, 1988

IV.1.1.2. Fichier au format XML

XML (eXtended Markup Language) est un format textuel extensible de description de documents. Il permet de s'adapter quasiment à tous les domaines où l'on a besoin de structurer l'information de façon portable. Il définit une syntaxe générique utilisée pour formater des données avec des balises simples et compréhensibles.

XML est un langage à base d'éléments, d'étiquettes, d'attributs et de valeurs. Les balises (tags) ouvrantes (respectivement fermentes) sont constituées d'étiquettes (labels) représentées entre le symbole « < » (respectivement « </ ») et le symbole « > ». Le composant logique constitué de : la balise ouvrante, la valeur puis la balise fermente est appelé « élément » (element). La valeur peut être vide, contenir du texte, d'autres éléments ou contenir un mélange des deux (mixed element content). Les balises définissent la structure du document. L'élément de plus haut niveau englobant tous les autres et n'ayant pas de parents est appelé « élément racine » (root

element). Un élément peut contenir des informations additionnelles appelées « attributs » (attributes). Un attribut est un couple formé d'un nom et d'une valeur représenté à l'intérieur d'une balise ouvrante sous la forme « nom = "valeur" ». Un document XML est un ensemble d'éléments ainsi imbriqués [DAN 03].

En effet, un document XML est représenté physiquement sous la forme d'un fichier texte structuré en éléments à l'aide des balises éventuellement imbriquées.

Voici un exemple d'un document XML simple :

```
<personne>
        <nom>
                <nomDeFamille>Morsli</nomDeFamille>
                <prénom>Mohammed</prénom>
        </nom>
        <profession>Etudiant</profession>
</personne>
```

Les documents XML peuvent parfaitement être utilisés pour le stockage des données tabulaires. Cela peut se faire si la structure du document XML correspond comme suit :

```
<Table>
        <Ligne>
                <Colonne1>…</Colonne1>
                <Colonne1>…</Colonne1>
                …
        </Ligne>
        <Ligne>
                …
        </Ligne>
                …
</Table>
```

Et c'est exactement ce genre de schéma que supporte notre système.

IV.1.1.3. Table de données relationnelle (TDR)

Dans le modèle relationnel, l'unité élémentaire est la table. Les tables constituent la structure logique du modèle relationnel. Un ensemble de tables regroupées forme une base de données.

Le succès du modèle relationnel auprès des chercheurs, concepteurs et utilisateurs est dû à la puissance et la simplicité de ces concepts. En outre, contrairement à certains autres modèles, il repose sur des bases théoriques solides, notamment la théorie des ensembles et la logique des prédicats du premier ordre [GAR 02].

Une table est un ensemble de données organisées sous forme d'un tableau où les colonnes correspondent à des catégories d'information (une colonne peut stocker des numéros de téléphone, une autre des noms, etc.) et les lignes à des enregistrements, également appelées entrées.

Chaque table est l'implémentation physique d'une relation entre les différentes colonnes. Chaque correspondance est définie par une ligne de la table. Voici un exemple simple des données d'une table de données relationnelle :

Sexe	Prénom	Année de naissance
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988

Il y a deux niveaux de travail sur une table :

- Un niveau de définition des données d'une table : qui permet de définir, lier, et contraindre les données via un langage de définition de données.
- Un niveau de manipulation d'une table : qui permet d'ajouter, supprimer et rechercher les données via un langage de manipulation de données.

Actuellement, le langage standardisé pour travailler sur les tables est le langage SQL, il est utilisé sur la plupart des SGBDR.

Notons que nous emploierons à titre de généralisation le terme « table de données (TD) » pour désigner un fichier plat de données CSV, un fichier de données XML ou une table de données relationnelle (TDR). De même, le terme « enregistrement » (record) pour désigner une ligne délimitée CSV, un élément XML (l'élément ligne) ou un tuple de données d'une TDR.

IV.1.2. Modes d'extraction

L'utilisateur souhaite exécuter la première phase de son processus et extraire les données des sources. L'extraction des données est la clé pour la réussite de notre processus. Cette tâche n'est pas aussi simple que cela puisse paraître. Il est donc nécessaire et même primordial d'opter une stratégie pour l'extraction des données. La complexité de l'extraction impose une intention particulière à la manière avec laquelle nous extrayons les données des sources. Pour des raisons d'efficacité, nous faisons appel au concept de buffering (lors de l'extraction des données à partir des sources qui se représentent physiquement par des fichiers) et/ou la technique de lecture d'enregistrements en bloc.

IV.1.2.1. Utilisation de mémoire tampon (Buffering)

Une mémoire tampon, couramment désignée par le terme anglais « buffer », est une zone de mémoire vive utilisé pour entreposer temporairement des données, notamment entre deux processus ou matériels ne travaillant pas au même rythme.

Ainsi, les données reçues d'un périphérique sont le plus souvent rassemblées dans des tampons en attente de leur traitement par l'ordinateur.

Les E/S sont en général bufférisées. En particulier pour la manipulation des fichiers ; les tampons sont systématiquement utilisés. Concrètement, un tampon est une zone mémoire accompagnée d'un indice ou d'un pointeur désignant la position de lecture courante. Dans le cas d'une extraction, et à chaque étape de lecture de données, une partie de l'entrée est copiée dans le tampon pour être utilisée. Cela permet de limiter les appels système « Read » qui sont beaucoup plus lents que les autres types d'opérations, et ainsi améliorer les performances. Le buffer est l'intermédiaire entre le fichier source et tout mécanisme d'extraction de données.

Dans notre cas, l'allocation du tampon se fait de manière automatique et invisible, la taille du tampon par défaut est de 8 Ko. Notre système supporte le contrôle de la gestion de la bufferisation d'un fichier ouvert avant sans utilisation. Nous sommes intervenu à ce niveau pour pouvoir rendre possible la personnalisation de la taille du buffer selon les préférences de l'utilisateur et l'associer au fichier avant le contact d'extraction.

IV.1.2.2. Extraction par enregistrement / bloc d'enregistrements

Notre système fournit la prise en charge de l'extraction d'enregistrements de données en bloc (ou par bloc), ce qui signifie que plusieurs enregistrements peuvent être récupérés à la fois lors d'une seule extraction (appel système Read), plutôt que de récupérer un seul enregistrement à la fois de la table de données. Nous utilisons pour ce mécanisme les tableaux pour stocker les enregistrements de données récupérés lors d'une extraction. Si l'utilisateur opte pour le transfert de données en bloc, il pourra spécifier la quantité de jeu d'enregistrements qui doivent être récupérés à la fois. La spécification se fera par la précision de la taille en octets ou la précision du nombre exacte d'enregistrements, selon le type de la source de données.

Même si l'extraction d'enregistrements en bloc représente un gain de performance comme nous le verrons par la suite, certaines caractéristiques de l'infrastructure sont à prises par considération. Avant de se lancer vers l'extraction d'enregistrements en bloc, il faut tenir compte des capacités du système en termes de mémoire vive (RAM) disponible aux traitements.

IV.1.3. Types d'extraction

En plus des modes de manipulation de l'extraction de données que varie notre système, nous assurons aussi deux types possibles d'extraction de données à partir des systèmes sources : l'extraction exhaustive et l'extraction sélective.

IV.1.3.1. Extraction exhaustive

Dans ce genre d'extraction, la lecture des données ne respecte aucune contrainte, elle se fait de manière la plus simple et la plus directe. La capture de données se fait à fond et rapporte tous les enregistrements, elle est donc complète et intégrale.

IV.1.3.2. Extraction sélective

Dans ce cas, l'extraction des enregistrements est guidée par une certaine contrainte, ce qui signifie qu'elle dépend d'une certaine condition de sélection à vérifier pour qu'un certain enregistrement puisse passer à l'étape suivante. Il s'agit d'une opération de restriction des données. Ce type d'extraction est très utile pour la fragmentation des sources de données dans un environnement de calcul parallèle où le besoin des données appropriées surgit.

IV.1.4. Extracteurs

La méthode de transfert des données à partir des sources s'appelle : « Extracteur ». L'extracteur cache l'hétérogénéité des systèmes sources et joue le rôle d'un intermédiaire entre une table de données et la phase de transformation. Ainsi, les extracteurs représentent l'interface de communication entre les sources et les différentes phases du processus.

Avant de commencer l'extraction, il est important de bien comprendre la nature et le schéma de la source de données. A ce niveau, les informations qui nous intéressent sont celles qui définissent les données. La liste des colonnes ou attributs (fields ou attributes, en anglais) de la table des données en fait partie. Dans le monde des SGBD, ces informations sont communément appelées « métadonnées ». Chaque système source utilise sa propre méthode pour stocker et organiser ces métadonnées. Notre mécanisme d'extraction est adapté à ces méthodes et extrait la liste des colonnes de la table de données dès le premier contact d'extraction si celle-là n'étant pas définie par l'utilisateur.

L'extracteur est associé à une seule table de données et traduit ses enregistrements à la structure pivot que nous avons appelé « Row ». Cette tâche de traduction consiste

à la réécriture des données en connaissance des attributs de la table concernée. Il s'agit en fait d'un accouplement des attributs aux valeurs tout en respectant les correspondances (mappage) entre eux. Ainsi, un enregistrement est converti et le résultat serait un dictionnaire de mappage attributs/valeurs ou un ensemble de couples (clé, valeur) où la clé correspond à l'attribut, comme suit :

Sexe, Prénom, Année de naissance
M, Alphonse, 1932

- Dictionnaire de mappage attributs/valeurs :
{'Sexe' : 'M', 'Prénom' : 'Alphonse', 'Année de naissance' : '1932'}

- Ensemble de couples (clé, valeur) :
{('Sexe', 'M'), ('Prénom', 'Alphonse'), ('Année de naissance', '1932')}

Les données sont représentées sous cette forme visant à faciliter leur compréhension et leur manipulation par la suite. Cela permet aussi un premier niveau d'abstraction qui représente également l'avantage d'homogénéiser la représentation des données.

La Figure suivante démontre le rôle de l'extracteur :

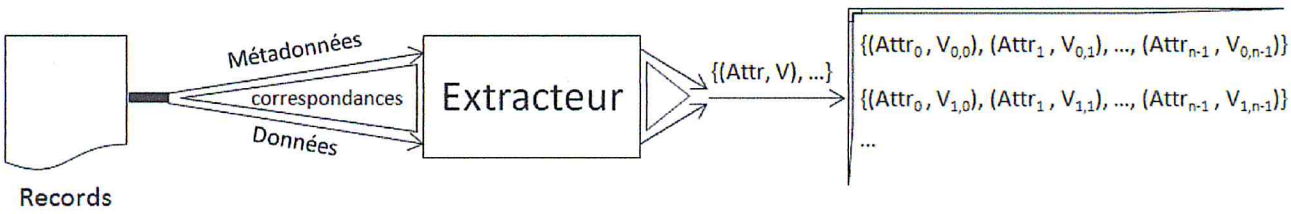


Figure 18 : Schéma de l'extracteur de données

Nous avons conçus des extracteurs adaptés aux sources de données selon trois modèles de données différents que nous avons détaillées précédemment (CSV, XML, DBMS-SQL). Dans ce qui suit, les exemples et l'algorithme de chacun des extracteurs.

IV.1.4.1. Exemple d'un extracteur CSV

La Figure ci-après illustre quelques lignes CSV extraites avec l'extracteur CSV.

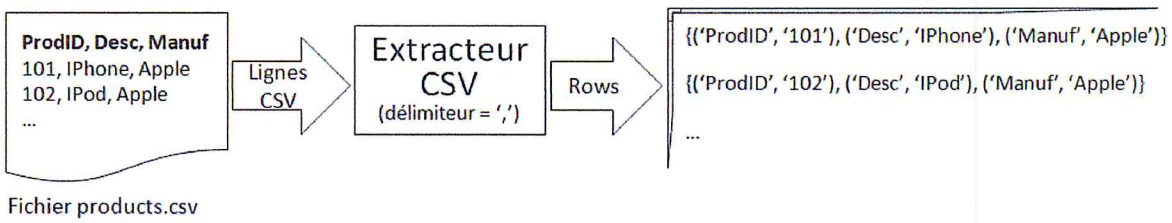


Figure 19 : Schéma d'un exemple d'extracteur CSV

IV.1.4.2. Exemple d'un parseur XML

La Figure ci-dessous illustre quelques éléments XML extraits avec le parseur XML.

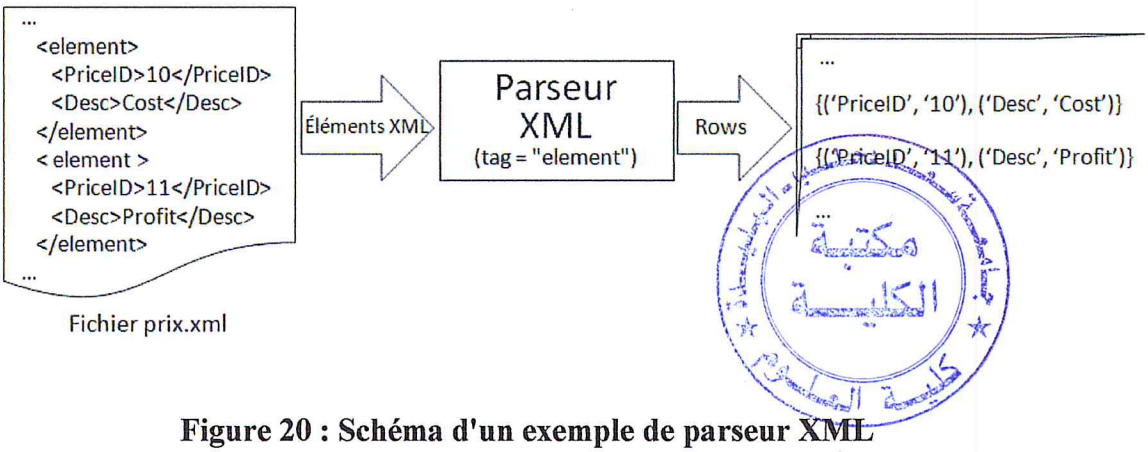


Figure 20 : Schéma d'un exemple de parseur XML

IV.1.4.3. Exemple d'un exécuteur SQL (DBMS-SQL)

La Figure ci-dessous illustre quelques tuples de données extraits avec l'exécuteur SQL.

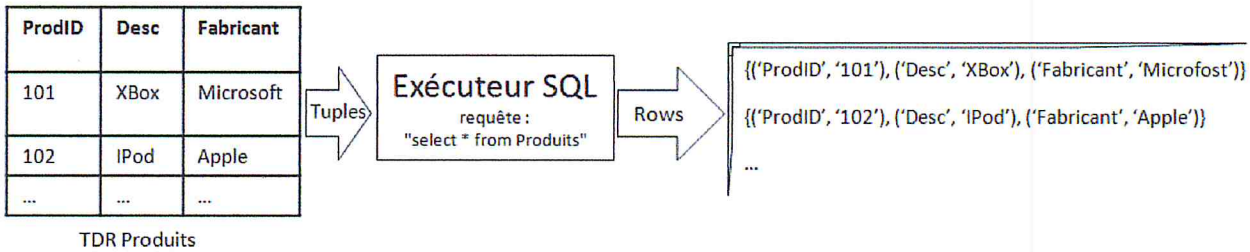


Figure 21 : Schéma d'un exemple d'exécuteur SQL

IV.2. Transformation

La transformation est la seconde étape du processus. Cette étape qui du reste est très importante. Elle consiste en la révision et la réécriture des données à l'aide d'un ensemble de transformations paramétrées par l'utilisateur dans l'étape de définition du processus. Cette étape est bien évidemment indispensable si on veut obtenir des cibles différentes des sources.

C'est dans cette étape là qu'on reçoit les rows dérivés de la phase d'extraction de données. La structure row est très utile pour une manipulation subtile des données à ce niveau. Elle permet d'accéder directement aux données voulues pour modification puisqu'elles sont mappées aux attributs.

IV.2.1. Transformateur

Notre système permet d'identifier chaque row, la source correspondante et les transformations qu'il doit subir pour pouvoir les lui appliquer selon la définition de l'utilisateur. Concrètement, un transformateur ou une transformation est une instruction qui peut être une opération ou une règle dans une fonction. Nous avons pu classer les transformateurs en deux classes : d'agrégation et de non-agrégation. Les transformateurs d'agrégation sont ceux qui opèrent sur plusieurs rows comme dans le cas du groupement de données. Par contre, les transformations de non-agrégation sont ceux qui peuvent opérer sur seulement un row, comme la projection et le filtrage. Notre système supporte seulement les transformateurs de non-agrégation. La Figure 22 ci-dessous présente le schéma et le rôle d'un transformateur de non-agrégation d'une manière générale.

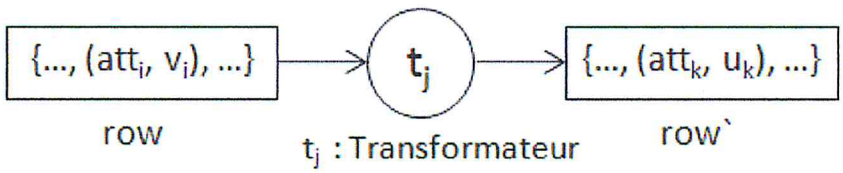


Figure 22: Schéma d'un transformateur de non-agrégation

IV.2.2. Pipeline de transformateurs

A titre de représentation et pour faciliter la compréhension, nous allons schématiser l'application de l'ensemble (liste) des transformateurs sur un row par un pipeline de transformateurs dans lequel entre ce dernier et s'expose aux différents opérateurs de transformation. La Figure 23 ci-dessous illustre l'application des transformations dans un pipeline de (m-1) transformateurs (dont 3 seulement sont présentés).

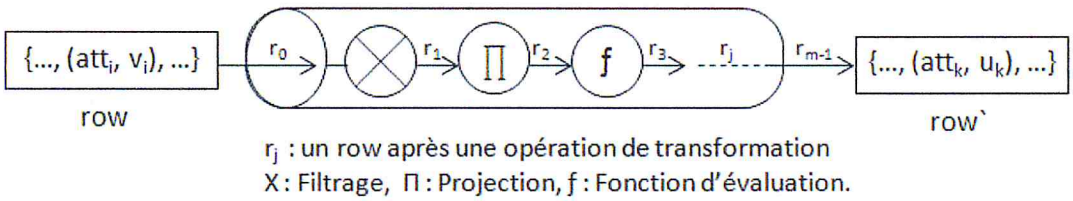


Figure 23 : Schéma d'un pipeline de transformateurs

Dans ce qui suit, l'algorithme de transformation de non-agrégation que nous appliquons.

	Algorithme 4 : Pipeline de transformation
1	Entrée : Row r, Liste de transformateurs T
2	Sortie : Row r
3	Début
4	Pour chaque t dans T Faire
5	r ← t.méthode(r, t.params)
6	FinPour
7	Si r n'est pas vide /* r pourrait être vide dans un cas de filtrage */
8	Retourner r
9	FinSi
10	Fin

IV.2.3. Table des transformateurs

Cette phase révèle donc qu'elle est essentiellement basée sur les fonctions de transformations. Nous mettons à la disposition de l'utilisateur un large éventail de transformateurs que nous avons conçu. Le tableau qui suit décrit toutes les fonctions de transformation possibles.

Nom	Description
ToString	Conversion d'une donnée en chaîne de caractères
ToStrippedString	Suppression des espaces qui sont en plus, à gauche et à droite d'une chaîne de caractères.
ToInteger	Conversion d'une donnée en un entier
ToLong	Conversion d'une donnée en un entier de type Long
ToFloat	Conversion d'une donnée en un réel
ToBoolean	Conversion d'une donnée en un booléen
FloatFormat	Formatage d'une valeur réelle (en spécifiant le nombre de valeurs après la virgule flottante par exemple).
Projection	Fonction permettant de sélectionner un ensemble d'attributs (colonnes) à préserver leurs valeurs correspondantes ou à les rejeter. Avec cette fonction, nous avons initié le partitionnement vertical.
Concatenation	Concaténer la chaîne donnée par l'utilisateur à droite ou à gauche de la valeur qui correspond à la colonne sélectionnée par l'utilisateur aussi.

	mots de la chaîne de caractères commencent toujours par une majuscule.
ToggleCase	Convertir une chaîne de caractères dans le mode « Toggle » où tous les mots de la chaîne sont en majuscules sauf les premières lettres.
FirstLetters	Sélectionner les 'n' premiers caractères à préserver d'une chaîne de caractères.
Replace	Remplacer une valeur (qui correspond à une colonne sélectionnée) par une autre valeur spécifiée par l'utilisateur.
ReplaceMany	Remplacer une liste de valeurs (qui correspond à une colonne sélectionnée) par une valeur définie par l'utilisateur.
Date & Datetime conversions	Convertir une date ou date/time du format ISO au format Européen et vice versa.
GetDate	Extraire seulement la date d'une valeur de type date/time
GetTime	Extraire seulement le temps (time) d'une valeur de type date/time
GetYear	Extraire l'année (year) d'une valeur de type date ou date/time
GetMonth	Extraire le mois (month) d'une valeur de type date ou date/time
NotNull	Fonction de filtrage qui prend un row et le retourne vide si la valeur qui correspond à l'attribut du filtrage est « Null », sinon, elle le retourne tel qu'il est.
ReplaceNullValue	Fonction similaire à NotNull sauf qu'elle ne retourne pas un row vide dans le cas

	de rejet mais elle remplace la valeur « Null » du row par une autre valeur donnée par l'utilisateur. Cette fonction est utile pour la gestion des valeurs creuses.

Tableau 3 : Liste des différentes transformations de notre système

IV.3. Partitionnement

Comme nous l'avons évoqué précédemment, notre travail de manière globale vise à traiter la problématique du volume des données et que nous avons proposé de solutionner par leur répartition dans une architecture distribuée. Mais il est en premier lieu nécessaire de partitionner (fragmenter) l'ensemble des données avant de le répartir.

Le partitionnement consiste à diviser à l'aide d'une fonction de partitionnement un ensemble de données en plusieurs fragments, appelés partitions, de manière à ce que la combinaison des partitions recouvre l'intégralité des données. Une fonction de partitionnement permet d'indiquer comment les données seront réorganisées dans les partitions. Elle est donc celle qui va diriger les données vers l'une ou l'autre de partitions. Techniquement, cette fonction utilise une clé de partitionnement qui va définir la partition à utiliser en fonction d'un plan de partitionnement défini.

Les fonctions de partitionnement peuvent bien être utilisées dans la phase d'extraction des données afin de satisfaire le besoin des données appropriées. Dans un contexte d'extraction, il s'agit de l'extraction sélective ; dans un contexte de partitionnement, cela représente ce que nous avons appelé le « partitionnement à la volée » vu qu'il implique de la fragmentation au moment même de l'extraction des données.

Notre système supporte le partitionnement horizontal, dans lequel nous avons varié les méthodes de partitionnement :

IV.3.1. Partitionnement Simple

La Figure 24 ci-après démontre la logique du partitionnement horizontal simple. Cette logique fut la plus simple et la plus classique des logiques de fragmentation. Elle divise le document horizontalement en 'n' partitions de tailles approximativement égales. Cette taille vaut : taille(document)/n.

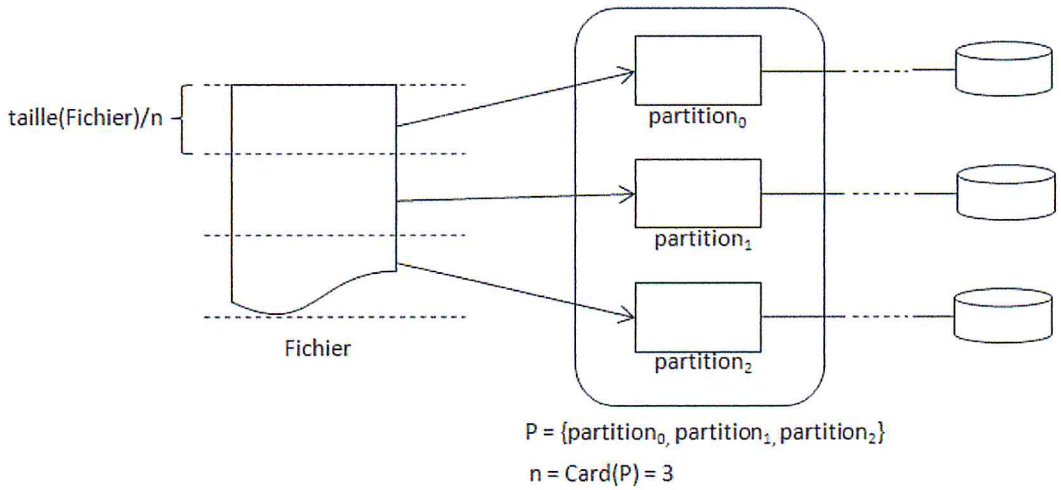


Figure 24 : Plan de partitionnement simple

Voici l'algorithme qui illustre concrètement cette logique :

	Algorithme 5 : Partitionnement simple
1	Fonction QuellePartition(curseur : Entier, partSize : Entier long) : Entier
2	Var
3	partition: Entier
4	Début
5	partition ← PartieEntière(curseur/partSize)
6	Retourner partition
7	Fin.
8	

L'algorithme dépend essentiellement des paramètres « curseur » et « partSize ». Il s'agit respectivement la position de l'enregistrement (ou bloc d'enregistrements) courant et la taille approximative d'une partition. Le « curseur » représente l'emplacement du curseur de lecture qui est positionné sur l'enregistrement (resp. bloc d'enregistrement) en question et évalué en octets. Le « partSize » représente la taille d'une partition qui est spécifiée aussi en octets.

Ce partitionnement n'est utilisé que pour les TD de type fichiers. Dans notre cas, c'est pour les TD CSV et XML.

IV.3.2. Partitionnement Round Robin simple

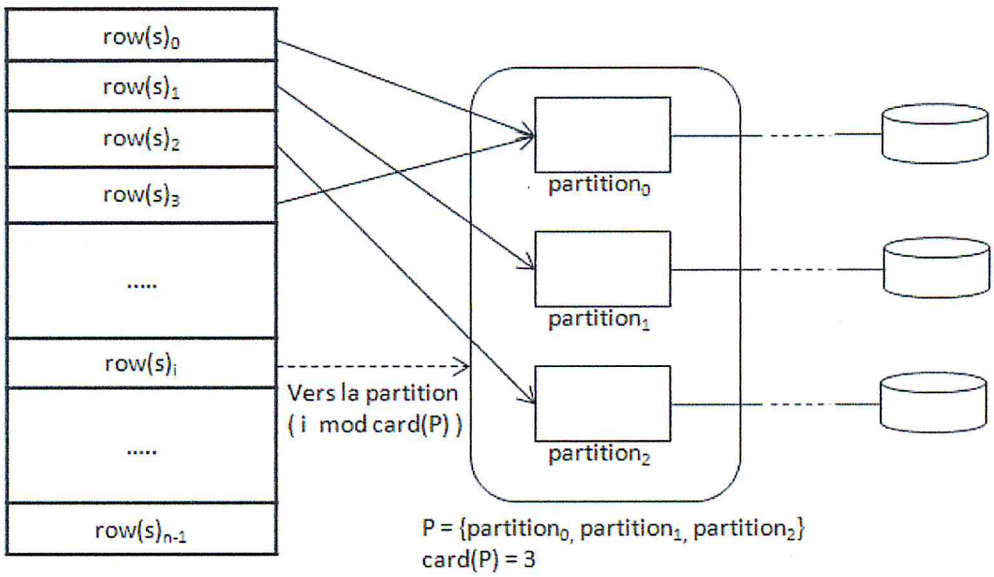


Figure 25 : Plan de partitionnement Round Robin Simple

La Figure 25 ci-dessus présente la fragmentation selon la logique Round Robin. Il s'agit d'une simple stratégie pour dispatcher les rows sur les 'n' partitions. Round Robin possède, par défaut, la propriété d'équilibrer les partitions. Il est utilisé surtout par les applications avec accès séquentiel aux données.

Théoriquement, chaque row (resp. bloc de rows) est rangé dans la partition suivante en séquence ; techniquement, les rows sont assignés selon la formule du Round Robin qui est la suivante : le numéro de la ligne modulo le nombre des partitions 'n'.

Dans notre cas, c'est le rang du row (ou bloc de rows) modulo 'n'. L'algorithme qui suit, représente la fonction Round Robin.

	Algorithme 6 : Partitionnement Round Robin simple
1	Fonction QuellePartition(rang : Entier Long, n : Entier) : Entier
2	Var
3	partition: Entier
4	Début
5	partition ← rang mod n
6	Retourner partition
7	Fin.

IV.3.3. Partitionnement Round Robin par groupes

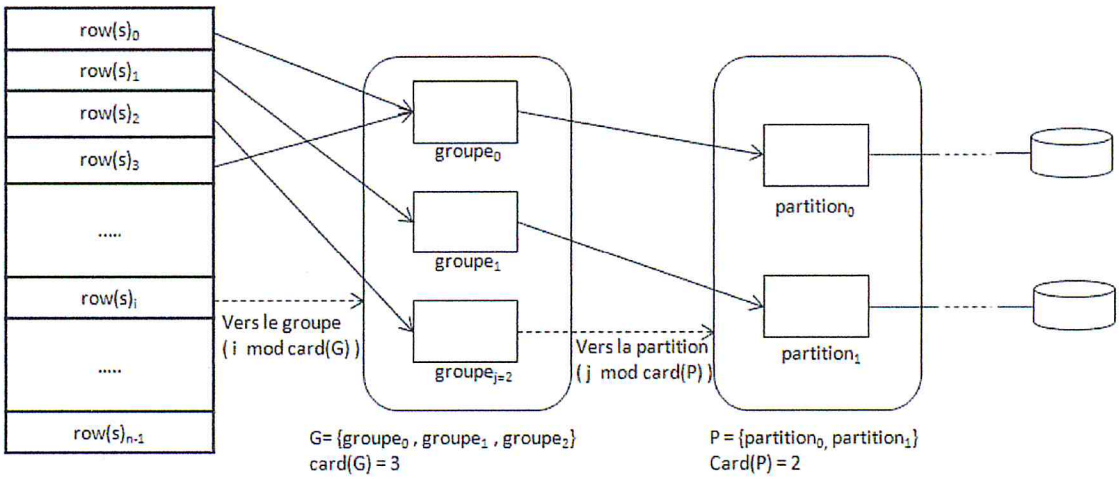


Figure 26 : Plan de partitionnement Round Robin par groupes

Il s'agit d'une application multiple de la logique Round Robin. Cela permet de ranger préalablement les rows dans des groupes, puis les groupes dans les 'n' partitions finales.

Concrètement, la personnalisation se fait au niveau de la formule Round Robin du calcul de la partition, comme suit :

partition ← (rang mod g) mod n, où 'g' représente le nombre de groupes.

IV.3.4. Partitionnement par hachage des attributs

Le partitionnement par hachage des attributs dépend des attributs désignés comme attributs de partitionnement. Les clés du partitionnement seraient donc les valeurs du row qui correspondent à ces attributs. La Figure 11 ci-dessus montre un exemple de partitionnement par hachage des attributs où l'ensemble des attributs du partitionnement se résume à la « Date ».

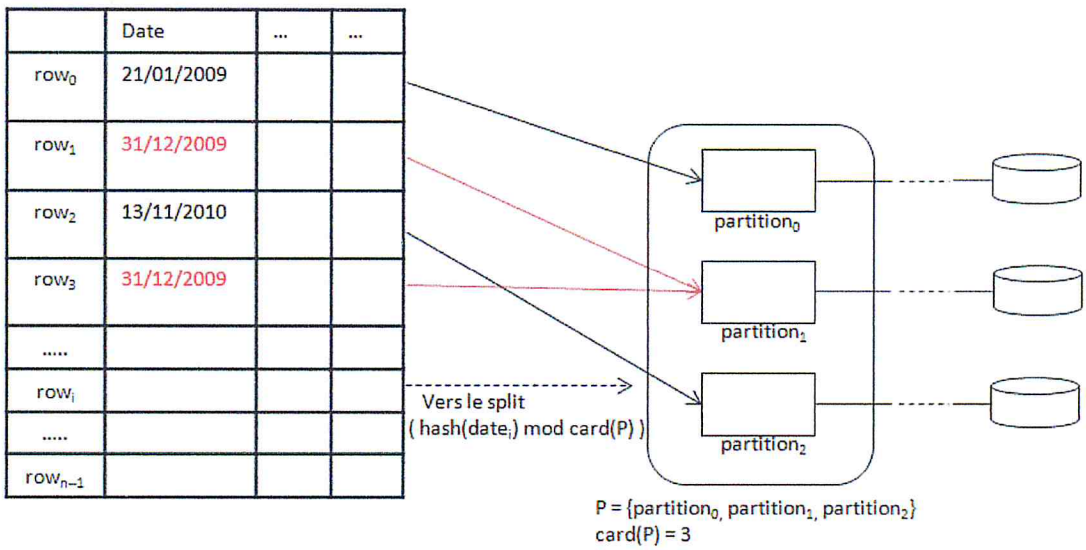


Figure 27 : Plan de partitionnement par hachage simple des attributs

Les rows sont rangés dans une partition particulière à l'aide de l'application d'une fonction du hachage « hash[5] » avec en entrée les clés du partitionnement. Cette fonction assure donc que les rows avec les même clés du partitionnement seront rangés exactement dans la même partition, comme indiqué en rouge sur la Figure ci-dessus.

Pour s'assurer que le résultat retourné par la fonction du hachage 'hash' soit dans le domaine de définition des partitions ([0..n-1]), nous exposons la formule du hachage au modulo. L'algorithme qui suit illustre la formule dont nous parlons :

[5] Une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte servant à identifier rapidement, la donnée initiale. Les fonctions de hachage sont utilisées en informatique et en cryptographie.

Algorithme 7 : Partitionnement par hachage des attributs

```
Fonction QuellePartition(r : Row, hAttrs : Liste de chaînes, n : Entier) : Entier
Var
        partition: Entier
        attr, clé, clés : Chaîne
Début
        clés ← une chaîne vide
        Pour chaque attr dans hAttrs Faire
                clé ← Str( r[attr] )
                clés ← clés + clé
        FinPour
        partition ← hash(clés) mod n
        Retourner partition
Fin.
```

La première partie de l'algorithme sert à déterminer les clés de partitionnement en les concaténant dans une chaîne de caractères globale, puis l'appliquer à la formule de sélection de la partition.

Cette méthode ne peut être appliquée qu'avec le mode d'extraction par enregistrement.

IV.3.5. Partitionnement hybride

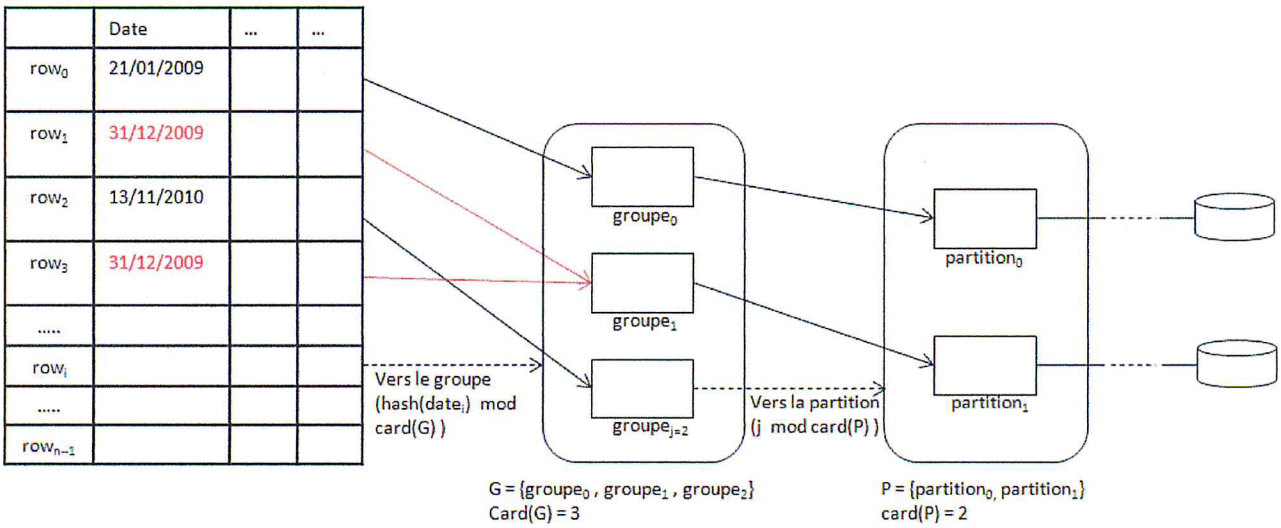


Figure 28 : Plan de partitionnement hybride

Nous adoptons dans cette méthode la composition des deux modes de partitionnement, le mode hachage des attributs et le mode Round Robin, d'où le nom « Hybride ». Dans un premier lieu, les rows seront rangés dans des groupes selon le principe du partitionnement par hachage des attributs. Ensuite, les groupes seront affectés aux partitions finales selon la logique Round Robin. La formule du partitionnement serait donc :

partition ← (hash(clés) mod g) mod n.

IV.4. Chargement

Dans la phase du chargement, notre démarche suit une stratégie constituée de deux étapes : l'allocation temporaire locale (chargement local), puis la répartition (chargement distribué).

IV.4.1. Chargement local

L'allocation locale des données consiste à stocker les partitions en question dans le système de fichiers local temporaire. Il s'agit de ranger les rows (après conversion en lignes CSV) issues des phases d'extraction et de transformation dans exactement les partitions désignées par la phase de fragmentation de données. La création des partitions dans le FS local se fait juste avant les premiers contacts d'insertion. A ce niveau, une partition est représenté physiquement un fichier plat CSV (ou CSV compressé). Ainsi, la bufferisation est impliquée. Une donnée n'est pas inscrite directement dans la partition en question mais elle est stockée dans une mémoire tampon en attente de son envoi effectif pour épargner à la machine le contretemps dû à la différence de débits entre le microprocesseur et les disques de stockage. Quand le tampon est rempli, son contenu est intégralement recopié dans le fichier partition, puis il est mis à jour, et le processus est répété jusqu'à fin des insertions.

De plus, notre système est adapté au chargement d'enregistrements de données en bloc, ce qui signifie que plusieurs enregistrements peuvent être insérés à la fois lors d'un seul appel système « Write », plutôt que d'insérer un seul enregistrement à la

fois dans la partition. Il s'agit de même principe que l'extraction par bloc d'enregistrements mais dans un contexte de chargement.

IV.4.2. Chargement distribué

La distribution de données consiste à répartir et répliquer les partitions qui se trouvent dans le FS local sur l'architecture distribué DFS. Le DFS possède une méthode de pousse de données directe. C'est à cette fonction que nous faisons appel pour accomplir cette tâche. DFS fonctionne sur deux concepts (cibles) : les blobs et les tags. Un blob est un objet arbitraire (fichier) qui à été poussé (répliqué) vers le DSF. Un tag est un objet qui contient les métadonnées sur les blobs. Plus précisément, un tag contient une liste d'URL qui correspond chacun à un blob. Ainsi, nous poussons les partitions d'une certaine table de données vers le DFS tout en gardant le schéma de la répartition dans un tag que porte le nom de la table de données (ou un autre identifiant, cela dépend de l'utilisateur).

V. Variantes

V.1. Variante classique

V.1.1. Architecture détaillée

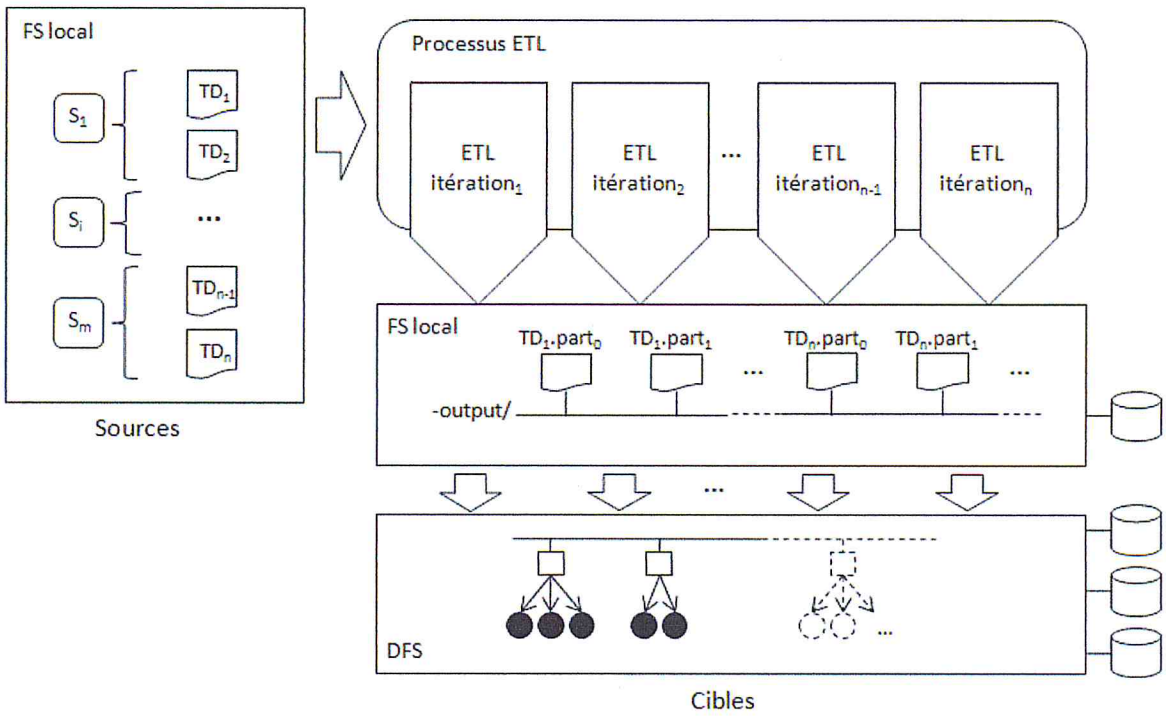


Figure 29 : Schéma détaillé du processus ETL classique

La Figure 29 illustre l'architecture détaillée de notre conception du processus ETL de distribution traditionnelle. Celui-ci se déroule séquentiellement en plusieurs itérations, disons 'n' itérations où 'n' représente le nombre de tables impliquées dans le traitement. Chaque itération concerne une table de données TD. Une itération de traitement comporte toutes les phases du processus ETL de distribution de données à savoir l'extraction, transformation, partitionnement et chargement (local et distribué).

Le processus est séquentiel comme nous l'avons déjà dit, ce qui fait qu'une itération de traitement (i+1) ne peut commencer que si l'itération (i) est complètement achevée, c.à.d. les partitions de la table de données TD(i) en question sont générées et poussées vers le DFS pour que l'itération suivante puisse commencer, et ainsi de suite.

V.1.2. Algorithme

	Algorithme 8 : Itération ETL classique
1	Algorithme ItérationETL
2	Var
3	attrs: Liste de chaînes
4	rows: Itérateur de Rows
5	row: Row
6	Reader, QuellePartition: Fonctions
7	rParams, pParams, wParams: Paramètres
8	nrParts, i, replicas: Entiers
9	parts: Liste de Fichiers
10	p: Fichier
11	délimiteur: caractère
12	premièreLigne, ligne, tag: Chaînes
13	T: Liste de Transformateurs
14	t: Transformateur
15	Configuration
16	- Extraction : Définition du type de la fonction de lecture de données
17	"Reader" (extracteur: CSV, XML, ou DBMS-SQL), ainsi que les

```
        paramètres "rParams" qui vont avec.
    - Transformation : Définition de la liste des transformateurs "T".
    - Partitionnement : Définition du type de la fonction de partitionnement
      "QuellePartition" (Simple, Round Robin, Hachage des attributs, etc.),
      ainsi que l'initialisation de ses paramètres "pParams" qui vont avec, dont
      le paramètre "nrParts" (nombres de partitions) en fait partie.
    - Chargement et distribution : Définition des paramètres d'écriture
      "wParams", dont les paramètres "délimiteur", ''tag'' et ''replicas''
      (nombres de répliques) en font partie.
Début
        (attrs, rows) ← Reader(rParams)
        nrParts ← pParams.GetNrParts()
        parts ← CréerLesPartitions(nrParts, wParams)

        délimiteur ← wParams.GetDélimiteur()
        Pour chaque p dans parts Faire
                premièreLigne ← RelierLesAttrsDansUneLigne(attrs, délimiteur)
                Ecrire(p, premièreLigne)
        FinPour

        Mise à jour des paramètres pParams si nécessaire
        Pour chaque row de rows Faire
                Pour chaque t dans T Faire
                        row ← t.méthode(row, t.params)
                FinPour
                Si row n'est pas vide Alors
                        Mise à jour des paramètres pParams si nécessaire
                        i ← QuellePartition(row, pParams)
                        p ← parts[i]
                        ligne ← ConstruireUneLigneCsv(row, attrs, délimiteur)
                        Ecrire(p, ligne)
                FinSi
```

```
50          FinPour
51          Pour chaque p dans parts Faire
52                  Fermer(p)
53          FinPour
54
55          tag ← wParams.GetTag()
56          replicas ← wParams.GetReplicas()
57          PousserLesPartitionsVersLeDFS(tag, replicas, parts)
58  Fin.
```

V.1.3. Description de l'algorithme

L'algorithme peut être décomposé comme suit :

La ligne (28) correspond à la capture de données et de métadonnées. La capture des données se fait en positionnant un pointeur de type curseur -itérateur- sur les données à extraire. Les métadonnées se résument dans notre cas à la liste des colonnes.

Les lignes (29→30) représentent la partie de création des partitions en récupérant le nombre des partitions « nrParts » et les paramètres du chargement nécessaires « wParams ».

Les lignes (32→36) constituent la partie d'insertion de la ligne des attributs (colonnes) dans les partitions. Il s'agit de la toute première ligne à insérer et qui représente des métadonnées.

Les lignes (38-53) servent au remplissage des partitions. Cette tâche comprend trois parties :

- Transformation du row (resp. bloc de rows dans un cas d'extraction d'enregistrements en bloc). Cette partie est représentée par les lignes (40→42)
- Affectation de la ligne délimitée à la partition adéquate et procéder à l'insertion. Cette partie est représentée par les lignes (43→49)

- Fermeture des partitions après la fin des insertions des lignes de données. Cette partie est représentée par les lignes (51-53)

Les lignes (55-57) correspondent à la partie de pousse des partitions de données vers le DFS avec un tag et nombre de répliques personnalisés.

V.2. Variante MapReduce

V.2.1. Architecture détaillée

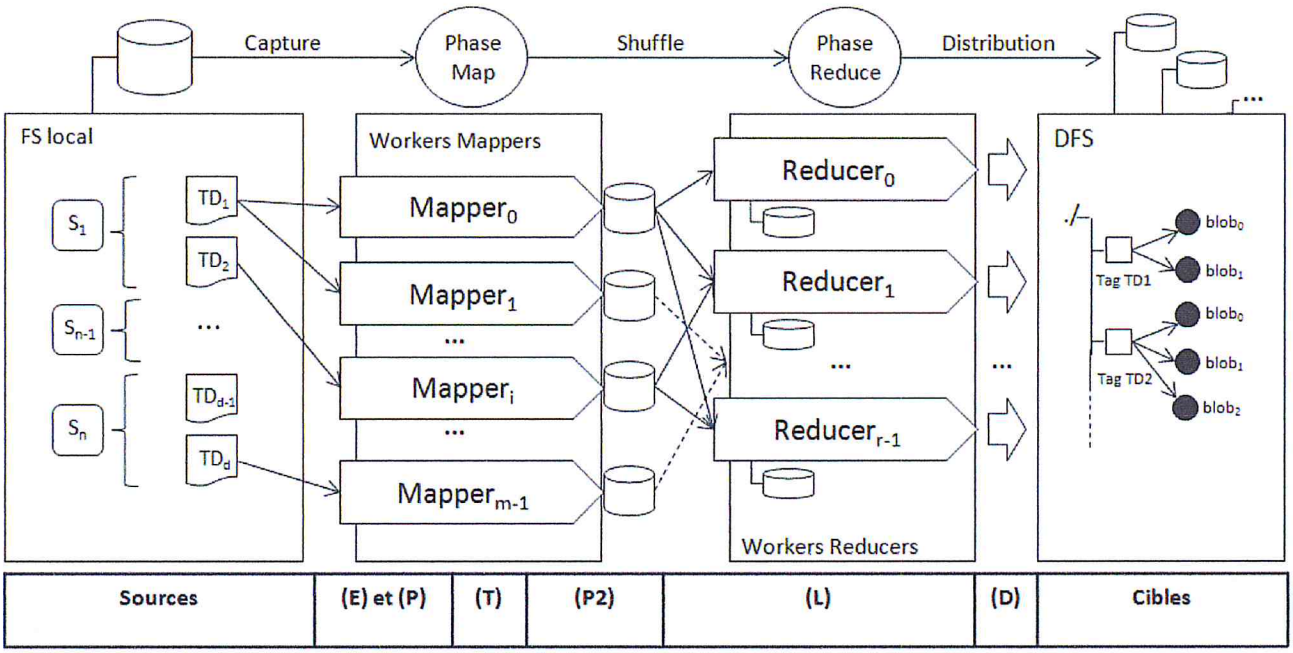


Figure 30 : Schéma détaillé du processus ETL dans le modèle MapReduce

- (E) et (P) : Extraction et Partitionnement (partitionnement à la volée)
- (T) : Transformation de données
- (P2) : Partitionnement de données dans des régions (fragments) dédiés aux Reducers. Il s'agit du 2ème niveau du partitionnement.
- (L) : Chargement local des partitions
- (D) : Distribution des partitions sur le DFS

La Figure 30 ci-dessus représente l'architecture détaillée du processus ETL de distribution de données dans le modèle MapReduce. Le processus comprend cinq

phases dans les deux contextes : MapReduce et ETL. Dans le contexte MapReduce, nous y trouvons : la capture des données, la phase Map, shuffle, la phase Reduce et la distribution des données. Dans le contexte ETL, nous avons : partitionnement à la volée, transformation, encore du partitionnement, chargement local et enfin distribution des données.

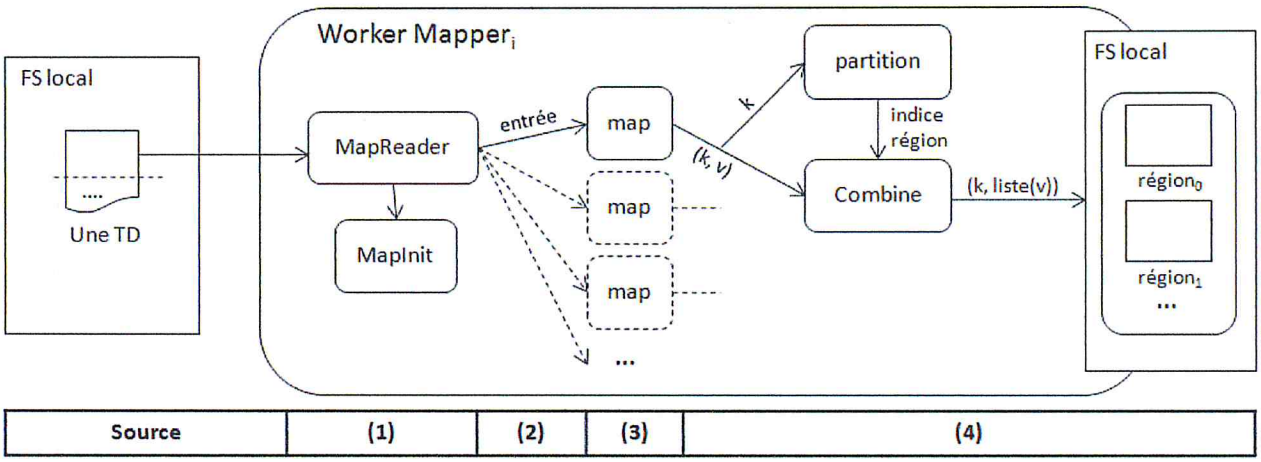
La phase de capture des données s'appuie principalement sur le partitionnement à la volée. L'extraction sélective est très importante pour cette phase. Elle rend possible le traitement des données d'une table volumineuse par plusieurs mappers. Cette phase représente le premier niveau de la parallélisation des données.

Dans la phase Map, on se voit travailler en plusieurs mappers en parallèle. Chaque mapper parcourt la partie de données qui lui est associée de la table des données. De plus, un mapper ne peut travailler qu'avec les données d'une seule table. Le mapper assure la phase de transformation et de partitionnement de données dans des régions (partitions) qui sont dédiées aux reducers. Ce partitionnement représente le deuxième niveau du parallélisation. Les sorties des mappers ou les régions de données se trouvent dans les disques locaux des mappers qui les génèrent. Les reducers ont besoin des sorties des mappers comme entrée de leur traitement. Ils vont donc copier ses régions dans leurs disques locaux. C'est la phase shuffle.

Une fois que les données ont été copiées dans les disques locaux des reducers, ces derniers seront lancés pour un traitement de chargement local des partitions en vue de leur distribution à la fin du traitement.

V.2.2. Mapper

Le Figure 31 ci-après représente le schéma détaillé d'un mapper. Un mapper assure les phases de traitement : capture de données, partitionnement à la volée, transformation et à la fin partitionnement et combinaison. Pour ce faire, un mapper s'organise en fait en 5 primitives : MapReader, MapInit, Map, Partition et Combine.



(1) Capture de données et initialisation
(2) Extraction et partitionnement à la volée
(3) Transformation
(4) Partitionnement et combinaison

Figure 31 : Schéma détaillé du Worker Mapper

Le point d'entrée d'un mapper est une table de données TD qui contient la partie (partition ou split) de données qu'on va lui associer. Un mapper s'approprie une partition de la table de données TD à l'aide du MapReader. Un MapReader parcourt donc le split qui lui est associé et lance séquentiellement une fonction Map pour chaque entrée du split. La fonction Map résulte en une paire clé/valeur (k, v). La clé 'k' est transmise à la fonction Partition pour le calcul de la région dans laquelle la paire (k, v) doit être inscrite. A ce moment, la fonction Combine est en connaissance de l'indice de la région sur laquelle elle doit opérer. Elle sera donc prête à recevoir la paire (k, v) pour la combiner dans la région adéquate. Les valeurs avec la même clé seront groupées dans la même région. Quand la taille du buffer de Combine sera atteinte, le buffer se décharge dans le disque local et se vide en vue d'autres itérations de groupement. Cette fonction permet de réduire les sorties du mapper, il y aura donc moins de données à écrire sur le disque et du coup moins de données à envoyer aux reducers au travers le réseau.

V.2.3. Reducer

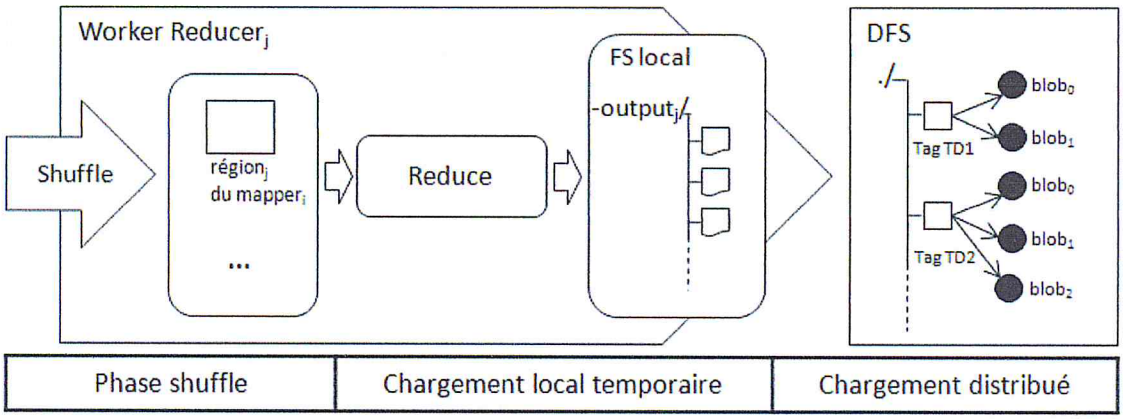


Figure 32 : Schéma détaillé d'un Worker Reducer

La Figure 32 ci-dessus représente un schéma détaillé d'un certain Reducer$_j$. Le Reducer se déroule en trois phases : phase shuffle, phase de chargement local, puis chargement distribué.

La phase shuffle est nécessaire pour que le Reducer puisse avoir ses données d'entrée. On copie donc les région$_j$ de tous les mapper$_i$ Ces régions sont passées à la fonction Reduce pour les traiter et les charger dans des partitions de données sur le FS local du Reducer à titre temporaire vu qu'elles seront distribuées sur le DFS par la suite.

V.2.4. MapReader

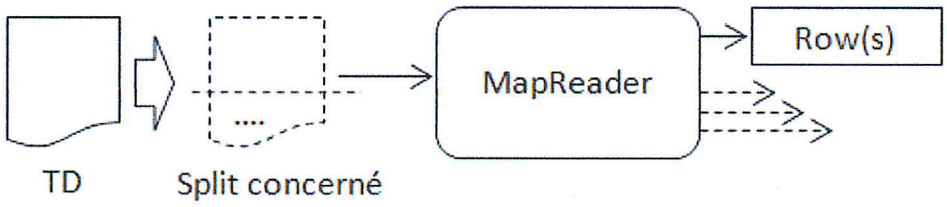


Figure 33 : Primitive MapReader

	Algorithme 9 : MapReader (Partitionneur à la volée)
1	Fonction MapReader() : Itérateur de Rows
2	Var globales
3	t : Table
4	Var
5	nrReaders, cettePartition, i : Entiers
6	row : Row
7	rows : Itérateur de Rows
8	Début
9	nrReaders ← t.GetNrReaders()
10	cettePartition← GetMapperID() mod nrReaders
11	rows ← t.Reader(t.rParams)
12	Mettre à jour les t.pParams si nécessaire
13	Pour chaque row de rows Faire
14	Mettre à jour les t.pParams si nécessaire
15	i ← t.DireMaPartition(row, t.pParams)
16	Si i = cettePartition Alors
17	Renvoyer row
18	FinSi
19	FinPour
20	FinFonc

Le mapper est associé à une seule table de données TD, ainsi donc son interface MapReader. Le MapReader dépend essentiellement de cette table et sa configuration qui est sauvegardée dans une structure de données appelée « Table ». Il s'agit d'un ensemble de (clé, valeur) où la clé représente le nom du paramètre et la valeur donc l'argument. Par exemple :

- ('nrReaders', 3) : c'est pour dire « nrReaders = 3 ». Ce couple représente le nombre de readers pour cette table là. Avec ce paramètre, nous pourrons déduire l'indice de la partition associé à ce MapReader qui est identifié par le « MapperID » du Mapper à qui il appartient.

- ('Reader', CsvReader) : Cela représente la définition du type de l'extracteur associé à la table de données en question. De même, la définition des paramètres qui vont avec l'extracteur : ('rParams', {…}). Cette fonction sert à capturer les données de la TD dans un itérateur de Rows.
- ('MaPartition', RoundRobin) : représente la définition du type de partitionneur qui sera appliqué à la volée de l'extraction. La fonction aura besoin de ses paramètres de partitionnement qui sont définis aussi dans un couple (clé, valeur) : ('pParams', {…}). Cette fonction sert à indiquer à quelle partition un row doit appartenir.

Ainsi, en connaissance de la partition associée au MapReader et la partition à laquelle un row doit appartenir, nous faisons une simple comparaison pour voir si le MapReader pourra s'approprier le row en question ou non.

V.2.5. Map

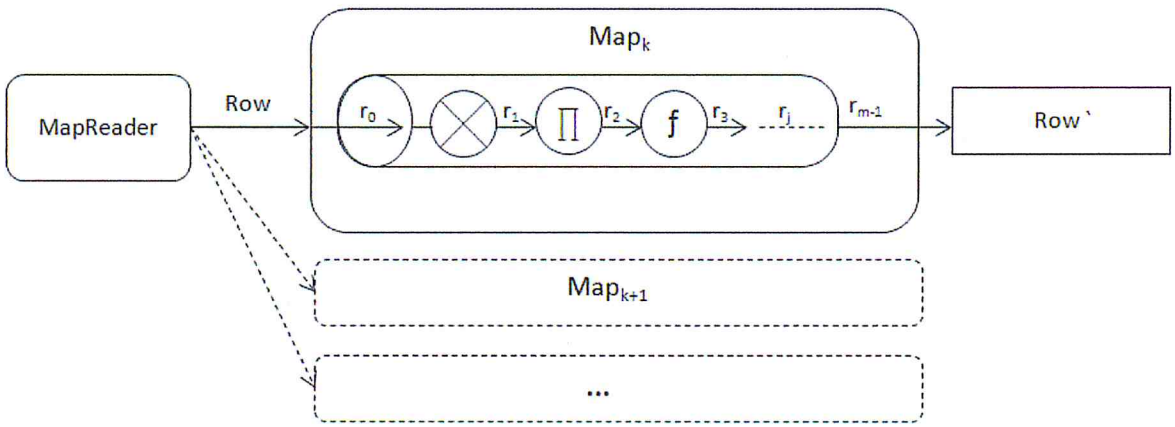


Figure 34 : Primitive Map

Le Figure 34 ci-dessus présente le schéma de la primitive Map. Un Map est déclenché pour chaque Row (resp. bloc de Rows) délégué par le MapReader. Les Map_k, Map_{k+1}, … sont appelés en séquence.

	Algorithme 10 : Map
1	Procédure MapInit (entrées : Itérateur d'objets)
2	Var globales
3	table: Table
4	T: Liste de transformateurs
5	Var
6	Début
7	T ← table.GetTransformers( )
8	FinProc
9	
10	Fonction Map (r : Row) : (clé: Entier, valeur: Row)
11	Var globales
12	T: Liste de Transformateurs
13	Var
14	t: Transformateur
15	key: Entier
16	value: Row
17	Début
18	Pour chaque t de T Faire
19	r ← t.Méthode(r, t.params)
20	FinPour
21	Si r != null Alors
22	key ← null
23	value ← r
24	Retourner (key, value)
25	FinSi
26	FinFonc

L'Algorithme Map se déroule en deux parties : traitement d'initialisation et traitement Map.

Dans le traitement d'initialisation, nous capturons la liste (pipeline) des transformateurs qu'on doit appliquer sur les données de la TD que le mapper s'en charge, soit T.

Un traitement Map est lancé pour chaque Row généré par le MapReader (resp. bloc de Rows) pour lui appliquer les modifications souhaitées dans un pipeline de transformateurs. Le traitement Map renvoi un couple (clé, valeur) dont la valeur est le Row transformé et la clé représente un Null car elle ne nous est pas indispensable à ce niveau.

V.2.6. Partition

Il s'agit du 2ème niveau de partitionnement dans notre système. Cette étape est très importante, elle est nécessaire pour que le mapper puisse disperser ces outputs sur des régions qui sont dédiées aux différents reducers. Concrètement, c'est une fonction qui nous dicte l'indice de la région dans laquelle une paire de clé/valeur doit être combinée.

A ce niveau, nous avons pu manipuler plusieurs stratégies de partitionnement, à savoir OTOT, OTAT, OTnT et nTOT.

V.2.6.1. OTOT

OTOT, qui veut dire « One Table One Task » ou « Une Table, Une Tâche » en français. C'est la stratégie la plus simple à adopter. Il s'agit de traiter les données d'une table par une seule tâche reduce. Ce qui fait qu'on associe chaque table de données à un seul reducer. Cela peut se faire en orientant la sauvegarde des outputs du mapper associé à une table de données vers la région dédiée au reducer associé à cette dernière.

Soit « ototTables » qui représente la liste des tables impliquées dans le traitement. Par exemple, les tables [$table_0$, $table_1$, $table_2$]. Ces tables seront traitées dans ce cas par les reducers [$reducer_0$, $reducer_1$, $reducer_2$].

Voici, dans ce qui suit, l'algorithme pour la stratégie OTOT :

	Algorithme 11 : OTOT
1	Fonction Partition (key: Entier, params: Objet): Entier
2	Var globales
3	tablename: Chaîne
4	Var
5	i, région: Entier
6	ototTables: Liste de chaînes
7	name: Chaîne
8	Début
9	ototTables ← params.ototTables
10	i ← 0
11	Pour chaque name de ototTables Faire
12	Si name = tablename Alors
13	région ← i
14	Aller à FinPour
15	Sinon
16	i ← i + 1
17	FinSi
18	FinPour
19	Retourner région
20	FinFonc

A ce niveau, on est en connaissance du nom de la table de données que le mapper s'en occupe pour traiter ses données. Ce nom est défini préalablement dans un autre niveau. L'indice de la région en question serait donc l'indice (position) de la table qui est identifiée par son nom dans la liste des tables « ototTables ».

Cependant, cette stratégie n'est pas optimale en termes de performance dans le cas où on est en présence des tables de données très volumineuses (à données intensives) par rapport aux autres tables. La différence entre les latences des reducers serait dans ce cas très importante. Pour remédier à cela, les données d'une seule table peuvent

être partagées sur tous les reducers disponibles pour le traitement, d'où la stratégie OTAT.

V.2.6.2. OTAT

C'est la stratégie « One Table All Tasks », « Une Table, Toutes les Tâches » en français. Comme son nom l'indique, il s'agit de traiter les données d'une seule table par toutes les tâches reduces. Plus précisément, ces données seront partagées sur les reducers. L'attribution des régions pour la combinaison des paires (clé, valeur) se fait donc en séquence pour les équilibrer. Pour ce faire, nous adoptons la formule Round Robin qui nous facilite l'équilibrage des régions. Voici l'algorithme qui montre son utilisation :

	Algorithme 12 : OTAT
1	Initialisation:
2	région ← -1
3	Fonction Partition (key: Entier, nrReducers: Entier, params: Objet) : Entier
4	Var globales
5	région : Entier
6	Début
7	région ← (région + 1) mod nrReducers
8	Retourner région
9	FinFonc

Maintenant, et à titre de personnalisation, nous avons pu mettre en œuvre deux autres stratégies : OTnT et nTOT.

V.2.6.3. OTnT

C'est « One Table 'n' Tasks », « Une Table, 'n' Tâches » en français. Cette stratégie dérive de la stratégie OTAT. Avec cette stratégie, nous pourrons limiter le nombre de reducers pour une certaine table de données au lieu qu'elle soit traitée par tous. Cette fonctionnalité est utile dans le cas où nous aurons un nombre important de reducers disponibles pour le traitement.

V.2.6.4. nTOT

C'est la stratégie « 'n' Tables One Task » (« 'n' Tables, Une seul Tâche » en français). Il s'agit d'opérer sur plusieurs tables avec une seule tâche reduce. Cette stratégie est très utile pour les tables moins volumineuses pour regrouper leurs traitements dans un seul reducer. Cette stratégie dérive de la stratégie OTOT, nous l'avons personnalisé afin de rendre possible le traitement des données de plusieurs tables par un seul reducer.

V.2.7. Combine

Il s'agit de la dernière primitive dans un mapper. Après qu'un traitement Partition est achevé, la primitive Combine sera prête à recevoir la paire (clé, valeur) soumise par la primitive Map pour la combiner dans la région adéquate (désignée par la primitive Partition). Voici l'algorithme de la fonction de combinaison :

	Algorithme 13 : Combine
1	Fonction Combine (key: Entier, value: Row, région: Buffer) : (clé: Entier,
2	valeur: Liste de Rows)
3	Var globales
4	size : Entier Long
5	Var
6	n : Entier Long
7	Début
8	Si n n'est pas défini Alors
9	n ← 50000
10	FinSi
11	key ← GetMapperID( )
12	région[key].add(value)
13	size ← size + 1
14	Si size >= n Alors
15	size ← size – taille(buffer)
16	Vider le buffer région en retournant ses éléments
17	FinSi

18	FinFonc

A ce niveau, une région est représentée par un buffer en mémoire avec une taille définie par l'utilisateur et évaluée en nombre de Rows (ou blocs de Rows). Si cette taille n'est pas définie, nous la mettons par défaut à 50.000 Rows (ou blocs de Rows). Il s'agit de la taille maximale du buffer ; une fois rempli, nous déchargeons ses éléments pour une sauvegarde locale.

La clé « key » qui ne nous était pas indispensable, elle l'est à ce niveau. Pour pouvoir garder trace de l'identité du mapper qui a traité les données, il nous est nécessaire de préserver cette information comme clé « key ». Cette clé sert à identifier les données lors du traitement Reduce.

V.2.8. Reduce

Après qu'un reducer shuffle (copie) ses régions des disques locaux des mappers, il fait appel à la primitive Reduce pour l'étape suivante (chargement). Une région est constituée de paires (clé, valeur) où la clé représente l'ID du mapper que nous avons préservé avec les données lors du traitement Combine, et la valeur représente des Rows (resp. blocs de Rows) regroupés dans une liste. La capture de ces paires se fait par l'itérateur de (clé, valeur) qui porte le nom « kviter » dans notre algorithme.

	Algorithme 14 : Reduce
1	Procédure reduce (kviter: Itérateur de (clé: Entier, valeur: Liste de Rows), params: Objet)
2	Var globales
3	config : Ensemble de (clé: Chaîne, valeur: Table)
4	Var
5	lesPartitions: Ensemble de (clé: Chaîne, valeur: Ensemble de (clé: Entier, valeur: Fichier))
6	mapid, reduceid, indice, replicas: Entiers
7	rows: Liste de Rows
8	row: Row
9	tablename, premièreLigne, ligne, tag: Chaînes
10	table: Table

```
11          wParams: Ensemble de (clé: Chaîne, valeur: Objet)
12          attrs: Liste de chaînes
13          délimiteur: caractère
14          partitions: Ensemble de (clé: Entier, valeur: Fichier)
15          partition: Fichier
16  Début
17          lesPartitions ← {}
18          Pour chaque (mapid, rows) de kviter Faire
19                  (table, tablename) ← GetInfoFromConfigByMapperID(config, mapid)
20                  reduceid ← GetReducerID()
21                  indice ←GetIndiceDeLaPartition(table, mapid, reduceid)
22
23                  wParams ← table.GetParamètresduChargement()
24                  attrs ← wParams.GetAttributs()
25                  délimiteur ← wParams.GetDélimiteur()
26                  Si  indice n'est pas défini dans lesPartitions[tablename] Alors
27                          partition ← CréerNouvellePartition(tablename, wParams, indice)
28                          lesPartitions[tablename][indice] ← partition
29                          premièreLigne ← RelierLesAttrsDansUneLigne(attrs, délimiteur)
30                          Ecrire (partition, premièreLigne)
31                  FinSi
32
33                  partition ← lesPartitions[tablename][indice]
34                  Pour chaque row de rows Faire
35                          ligne ← ConstruireUneLigneCsv(row, attrs, délimiteur)
36                          Ecrire (partition, ligne)
37                  FinPour
38          FinPour
39
40          Pour chaque (tablename, partitions) de lesPartitions Faire
41                  Pour chaque (indice, partition) Faire
42                          Fermer(partition)
```

```
43                  FinPour
44          FinPour
45
46          tag ← table.GetTag()
47          replicas ← table.GetReplicas()
48          PousserLesPartitionsVersLeDFS(tag, replicas, lesPartitions)
49  FinProc
```

L'algorithme peut être décomposé comme suit :

La ligne (19) sert à identifier les données par la récupération des informations sur la configuration de la table de données traitée par le mapper dont nous possédons l'identifiant.

Les lignes (20→21) sont utilisées pour déduire l'indice de la partition dans laquelle la liste des Rows (resp. liste des blocs de Rows) doit être insérée. Le calcul de l'indice se fait en fonction de l'identifiant du mapper qui est associé à la liste des Rows et l'identifiant du reducer où la liste a atterri.

Les lignes (23→31) représentent la partie de préparation d'une nouvelle partition pour le chargement. La préparation d'une partition se fait par sa création, l'insertion de la ligne des colonnes et l'ajout de cette partition à la table des partitions « lesPartitions ».

Les lignes (33→37) représente l'activité d'insertion de la liste des Rows (resp. liste des blocs de Rows) dans la partition adéquate après conversion en lignes délimitées.

Les lignes (40→44) correspondent à la partie qui est utilisée pour la fermeture des partitions quand toute activité d'insertion est cessée.

Les lignes (46→48) servent à répartir les partitions sur le DFS avec un tag et nombre de répliques qui sont définis par l'utilisateur.

A la fin, le nombre des partitions qui en résulte pour une seule table de données dépendra du nombre des mappers qui l'ont partitionné à la volée et le nombre des

reducers qui ont chargé ses données. Par exemple, si une table de données a été partitionnée à la volée par 4 mappers où chaque mapper délègue 4 régions (chacune pour un reducer) pour le chargement, nous aurons dans ce cas 16 partitions. C'est l'avantage que le processus parallèle tient (par rapport au processus classique) vu qu'il partitionne les données à deux niveaux. Ce qui fait que, pour avoir 'n' partitions d'une table de données, nous devons la traiter avec 'x' mappers et 'y' reducers tel que 'n' égal au produit 'x' fois 'y'.

VI. Conclusion

Dans ce chapitre, nous avons présenté l'étude détaillée de notre système (schémas, algorithmes, ...). Notre système étant la migration du processus ETL de distribution de données dans le modèle MapReduce. Dans un premier temps, nous avons présenté l'étude de ce processus dans le modèle classique, puis dans le modèle MapReduce. Cette stratégie que nous avons suivie nous a permis de mettre en évidence les étapes nécessaires pour la création d'un tel système.

Le chapitre suivant sera donc consacré à la concrétisation de ce que nous avons conçu. En d'autres termes, la réalisation de notre système, ainsi que sa validation par des tests et des expérimentations.

Partie III :

Mise en œuvre du système

Chapitre VI :

Implémentation du système

I. Introduction

Après avoir présenté les différentes étapes de la conception de notre système, il est temps de passer à la partie pratique. C'est dans cette partie du mémoire que nous allons présenter notre système de manière technique en définissant d'abord les choix en termes de d'environnement et outils de développement ainsi que l'architecture technique de notre système. Ensuite, nous procéderons à la validation de notre outil avec des tests et expérimentations.

L'application développée dans le cadre de ce projet est nommé : « pyETL-Distrib. »

II. Environnement de développement

II.1. Projet de Disco

Pour une définition, Disco est un framework de calcul parallèle distribué basé sur le paradigme MapReduce.

Disco est une plateforme open-source d'analyse de données à grande échelle. La plateforme inclut, entre autres, une implémentation de paradigme MapReduce. Etant un framework original, Disco supporte le calcul parallèle sur des quantités de données massives, fonctionnant sur des clusters d'ordinateurs.

Le noyau de Disco est codé en Erlang, un langage fonctionnel qui est conçu pour les constructions robustes, la tolérance aux pannes et les applications distribuées. Les utilisateurs de Disco, généralement, écrivent (programment) les jobs MapReduce avec du langage Python, ce qui permet d'implémenter même des algorithmes complexes seulement en quelques dizaines de lignes de code.

Disco à été lancé au Centre de Recherche Nokia (NRC), tel un framework léger pour le traitement distribué des données. Disco à été utilisé avec succès chez Nokia et d'ailleurs pour une variété de traitement : l'analyse (parsing), le reformatage, l'analyse journal (log), clustering, la modélisation probabiliste, le datamining, l'indexation et l'apprentissage machine. Avec Disco, ces tâches

peuvent être effectuées aussi bien avec des téraoctets de données, comme il le serait avec seulement quelques mégaoctets. [13]

II.2. Erlang

L'Erlang est un langage de programmation fonctionnelle et concurrente, créé par Ericsson, et qui l'utilise dans plusieurs de ses produits. Il permet le développement d'applications en temps réel à très haute disponibilité. Certains de ses usages sont dans les télécoms, les banques, le commerce électronique, la téléphonie sur ordinateur et la messagerie instantanée. Erlang possède des fonctionnalités de tolérance aux pannes, la programmation concurrente et distribuée. [15]

II.3. OpenSSH client/serveur

OpenSSH est une version LIBRE du protocole SSH (Secure SHell) de connexion réseau utilisé par les utilisateurs techniques de l'Internet. Les utilisateurs de Telnet, rlogin, et ftp ne réalisent pas que leur mot de passe est transmis non chiffré à travers Internet. OpenSSH chiffre tout le trafic (mots de passe inclus) de façon à déjouer les écoutes réseau, les prises de contrôle de connexion, et autres attaques réseaux. De plus, OpenSSH fournit des possibilités de création de tunnels sécurisés, plusieurs méthodes d'authentification et supporte toutes les versions du protocole SSH. [16]

OpenSSH est l'un des prérequis pour la mise en œuvre de l'environnement Disco. Il utilisé pour la gestion de sécurité du cluster Disco, ainsi que les échanges de données et informations entre les nœuds de ce dernier. Le protocole impose un échange de clés de chiffrement en début de connexion. Par la suite, toutes les trames sont chiffrées.

II.4. DDFS (Disco Distributed File System)

DDFS est un système de fichiers distribué propriétaire. Il a été développé par le NRC pour leurs propres applications. Il fournit une couche de stockage distribué pour l'environnement Disco. Il a été conçu spécifiquement pour soutenir les cas

d'utilisation qui sont typiques à Disco et MapReduce en général (stockage et le traitement de quantités massives de données immuables). Il est très approprié pour le stockage, par exemple : les données journal (log data), les grands objets binaires (photos, vidéos, ...).

DDFS est un composant de bas niveau dans la pile de Disco. Il prend soin de la distribution des données, la réplication, la persévérance, l'adressage et l'accès. Il ne fournit pas une fonction de recherche sophistiquée en soi mais elle est étroitement intégrée avec les jobs Disco. [13]

II.5. Système Linux (Ubuntu)

Ubuntu (prononciation « ou-boun-tou » en français) est un système d'exploitation libre, commandité par la société Canonical et une marque déposée par cette même société.

Fondé sur la distribution Linux Debian, ce système d'exploitation est constitué de logiciels libres, et est disponible gratuitement, y compris pour les entreprises, selon un principe lié à la philosophie affichée du projet. On estime qu'il a plus de 25 millions d'utilisateurs des différentes versions pour ordinateurs. [17]

Nous avons développé notre application sous un système d'exploitation Linux suite à la nécessité que Disco ne tourne que sur une machine dotée d'un système Linux/Unix.

II.6. Python

Python est remarquablement un puissant langage de programmation dynamique qui est utilisé dans une grande variété de domaines d'application. Python est souvent comparé au Java et autres langages. Certaines de ses caractéristiques clés incluent: [18]

- Une syntaxe lisible et très claire
- capacités d'introspection forte
- l'orientation objet intuitive
- une modularité complète, supportant les paquets hiérarchiques

- les types de données dynamiques de très haut niveau

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées à chaque traitement. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses comme par exemple un script qui récupérerait la météo sur Internet ou qui s'intégrerait dans un logiciel de conception assistée par ordinateur afin d'automatiser certains enchaînements d'actions répétitives.

II.7. Eclipse/PyDev

Eclipse est un Environnement de Développement Intégré (IDE), c'est à dire un logiciel spécialisé dans l'aide au développement d'applications. Les IDE intègrent au minimum les trois outils indispensables au développement : un éditeur de texte, un compilateur et un débogueur. Eclipse, développé à l'origine par IBM, a la particularité d'être un logiciel libre écrit en JAVA, on le trouve donc gratuitement sur toutes les plateformes (Windows, Linux,...). C'est aujourd'hui un outil largement utilisé dans le monde industriel [19]. Au cours du développement de notre application, nous l'avons utilisé avec PyDev. PyDev est un plug-in Eclipse pour le développement des projets Python. Il propose entre autres les fonctionnalités suivantes : complétion de code, analyse et mise en évidence de la syntaxe Python, debug, etc.

II.8. wxPython

WxPython est une boîte à outils graphique pour le langage de programmation Python. Il permet aux programmeurs Python de créer des programmes avec une interface utilisateur graphique très fonctionnelle, simple et facile. Il est implémenté comme un module d'extension Python (code natif) qui encapsule la fameuse bibliothèque wxWidgets qui est écrit en C++. [20]

Nous avons utilisé ce module de Python pour l'implémentation des différentes interfaces graphiques de notre application.

II.9. PostgreSQL

PostgreSQL est un système de gestion de bases de données relationnelles et objets (SGBDRO), c'est un descendant Open Source du code original de Berkeley. Il supporte une grande partie du standard SQL tout en offrant de nombreuses fonctionnalités modernes : requêtes complexes, clés étrangères, triggers, vues, intégrité transactionnelle, etc.

PostgreSQL est largement reconnu pour son comportement stable, proche d'Oracle. Mais aussi pour ses possibilités de programmation étendues, directement dans le moteur de la base de données, via un langage procédural PL/pgSQL. Le traitement interne des données peut aussi être couplé à d'autres modules externes compilés dans d'autres langages. [21]

Nous utilisons PostgreSQL pour la gestion d'une source de données de tests de type base de données relationnelle.

II.10. Psycopg2

Psycopg est le plus populaire adaptateur de bases de données PostgreSQL pour le langage de programmation Python. Ses caractéristiques principales sont la mise en œuvre complète de la spécification Python DB API 2.0 et la sécurité des threads (plusieurs threads peuvent partager la même connexion). Il a été fortement conçu pour applications multithread qui créent et détruisent beaucoup de curseurs et de faire un grand nombre d'opérations concurrents, telles que INSERT et UPDATE. [22]

Il s'agit d'une API wrapper que nous utilisons pour communiquer avec PostgreSQL via notre extracteur de données DBMS-SQL. Aussi, nous l'utilisons pour l'exportation des données d'une BDR vers un fichier plat CSV.

II.11. ElementTree XML

Il s'agit d'une bibliothèque (module) Python qui comprend des outils pour l'analyse XML en utilisant les API basées sur les événements (tels que les événements début et fin d'un élément XML) et les expressions de recherche [23].

Il comprend des objets conçu pour stocker des structures de données hiérarchiques dans la mémoire. Il permet donc l'analyse, la création et la modification des documents XML.

Nous utilisons ce module dans l'analyse syntaxique d'une source de données de type document XML pour la récupération des enregistrements de données via notre extracteur XML. Il est utilisé aussi lors de la conversion d'un document XML en un fichier CSV.

III. Architecture technique du système

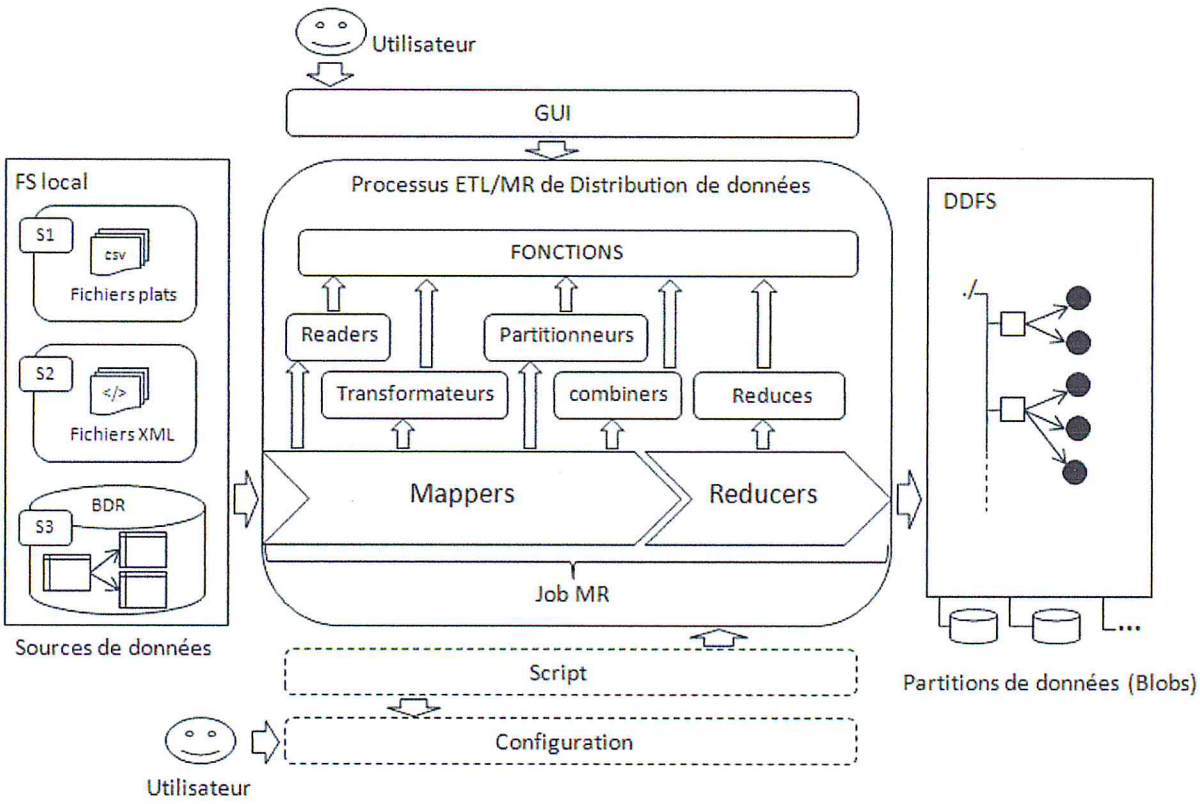


Figure 35 : Architecture technique du système

La Figure 35 ci-dessus illustre l'aspect technique de la réalisation de notre système. Le système comprend différentes parties que nous pouvons ranger dans trois couches différentes :

III.1. Couche interface utilisateur

Il s'agit d'une interface homme-machine avec laquelle l'utilisateur interagit avec notre système pour la définition des différentes phases du processus. Nous avons fait en sorte que notre application soit la plus dynamique possible et qu'elle soit adaptée pour les deux cas d'utilisation possibles : avec une interface graphique et un script d'exécution.

III.1.1. Interface graphique

L'interface graphique servira à utiliser et tester les différentes fonctionnalités de notre application visuellement. L'usager peut-être non-informaticien. Cette version fonctionne sur des systèmes d'exploitation dotés d'interfaces graphiques. L'interface graphique principale peut être démarrée par le module python « ihm.py ».

III.1.2. Script

Avec un script à exécuter « script.py », la définition du processus se fait par un fichier qui comprend tous les paramètres de configuration dont la syntaxe est du Python et qui est nommé « config.py ». Dans ce cas, l'utilisateur doit maitriser au moins les bases de la programmation Python. Cette interface est pour les programmeurs et elle est destinée surtout pour une utilisation sur un système d'exploitation qui n'est pas une édition de bureau mais plutôt une édition serveur où les interfaces graphiques sont exclues et la manipulation se fait en ligne de commande pour des raisons de performance et de sécurité.

III.2. Couche de traitement

C'est la couche principale de notre système. Elle comprend les différents modules nécessaires à l'exécution des Mappers et Reducers (notre processus étant un job MapReduce). Il s'agit des bibliothèques Python où nous avons rangé toutes les fonctionnalités de notre système. Il y en a ceux qui sont appelés par les mappers et d'autres par les reducers.

- **Readers** : ce module contient les définitions de tous les partitionneurs à la volée (extracteurs sélectifs) dans les deux modes d'extraction possibles (par enregistrement ou bloc d'enregistrements) et pour tous les types de sources de données que notre système supporte. Par exemple, le « RoundRobinCsvPartReader » s'en charge de l'extraction de données à partir d'un ficher plat au format CSV et dont la logique de la sélection est du Round Robin. Le module Python Readers est sous le nom : « mapreaders.py »
- **Transformateurs** : c'est dans ce module que réside tous les transformateurs possibles. Nous avons mis à disposition de l'utilisateur une variété de fonctions (filtrage, formatage, traitement de chaînes de caractères, conversion de types, conversion de formats de dates, ...). Ce module Python porte le nom : « transformers.py »
- **Partitionneurs** : un partitionneur s'occupe de la distribution des données (intermédiaires) des mappers sur les reducers. Ce module contient la définition des quatre stratégies de partitionnement possibles : OTOT, OTAT, OTnT, nTOT. Le nom du module est « partitionners.py »
- **Combiners** : ce module contient la fonction de combinaison que nous utilisons pour réduire la taille des résultats intermédiaires en groupant tous les rows (resp. blocs de rows) de la même clé dans une liste. Concrètement, au lieu d'envoyer aux reducers au travers le réseau plusieurs (k, row), nous enverrons certaines paires (k, liste(row)) dont la taille de la liste est spécifiée par l'utilisateur. Le module a le nom « combiners.py »
- **Reduces** : dans ce module, il y a la définition des deux fonctions pour la primitive reduce du reducer : une fonction pour le mode du chargement par enregistrement et l'autre pour celui du chargement en vrac (par bloc d'enregistrements). Le module est sous le nom « reduces.py ».
- **FONCTIONS** : c'est la bibliothèque des fonctions secondaires. Il s'agit de toutes les autres petites fonctions dont tous les autres modules s'en servent. Ce module est sous le nom « foncs.py ».

III.3. Couche de stockage

Notre outil permet de traiter plusieurs sources de données et d'en générer les fragments. Le stockage est une fonctionnalité primordiale dans notre application, du moment qu'elle permet le stockage temporaire des partitions de données avant leur distribution sur le DDFS.

III.3.1. Stockage source

Il s'agit du FS local du master où l'application tourne. Il permet la sauvegarde des différentes sources de données dans trois modèles de données différents que notre outil peut traiter (CSV, XML et le Relationnel).

III.3.2. Stockage cible

Cela consiste en la sauvegarde des partitions de données en deux étapes. En premier lieu, ces dernières sont allouées temporairement sur les FS locaux des machines reducers. En second lieu, les partitions seront réparties sur le DDFS et elles seront tags et blobs. A la fin, le schéma tag/blobs est retourné à l'utilisateur pour le préserver s'il le souhaite.

IV. Présentation de l'application

Nous allons à présent présenter les interfaces graphiques des plus importantes fonctionnalités de notre système. La fenêtre principale de notre outil comporte deux composants principaux : la barre de menu et la barre d'outils.

Les menus principaux dans la barre des menus sont :

- Disco DFS : Englobe toutes les fonctionnalités reliées à la manipulation et la gestion du DDFS (tags, blobs, ...), ainsi que le serveur Disco (statut, démarrer, arrêter, redémarrer et voir l'interface web du Disco).
- Data Export : Présente les deux types d'exportation (conversion) de données possibles (XML/CSV, TDR/CSV)

- Data Processes : Présente les deux processus de traitement de données à savoir le processus classique (non parallèle) et le processus parallèle en MapReduce.

La barre d'outils comprend différents boutons qui sont reliés aux différentes fonctionnalités et options de la barre des menus.

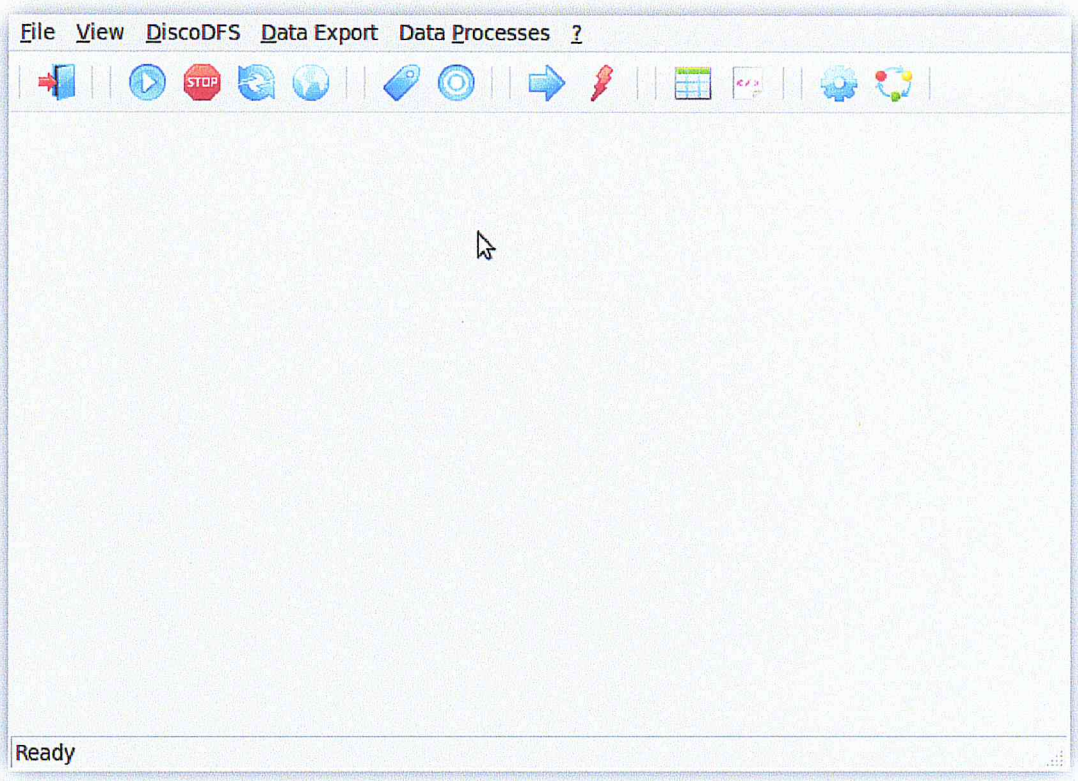


Figure 36 : Fenêtre principale

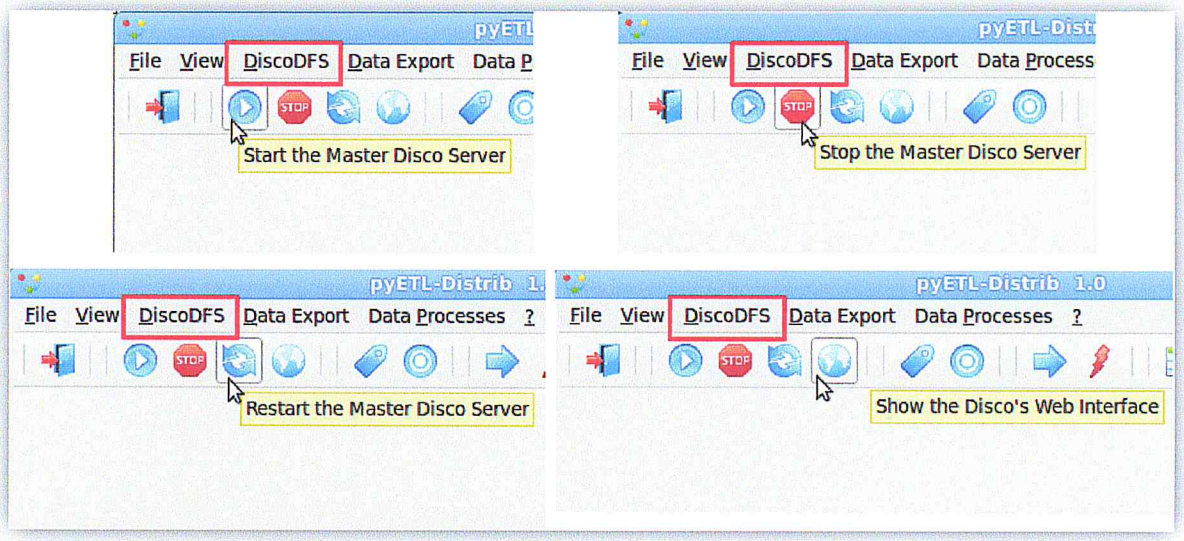


Figure 37 : Contrôle du serveur Disco

La Figure ci-dessus nous montre les différents boutons de la barre d'outils qui servent à contrôler le serveur Disco. Disco ne fournit pas ses options en interface graphique mais plutôt en script dans un API Python. Nous avons habillé visuellement ses fonctionnalités pour faciliter le contrôle du Disco et du DDFS.

Cependant, Disco est doté d'une interface web pour sa configuration et la vérification des statuts des différents nœuds de son cluster.

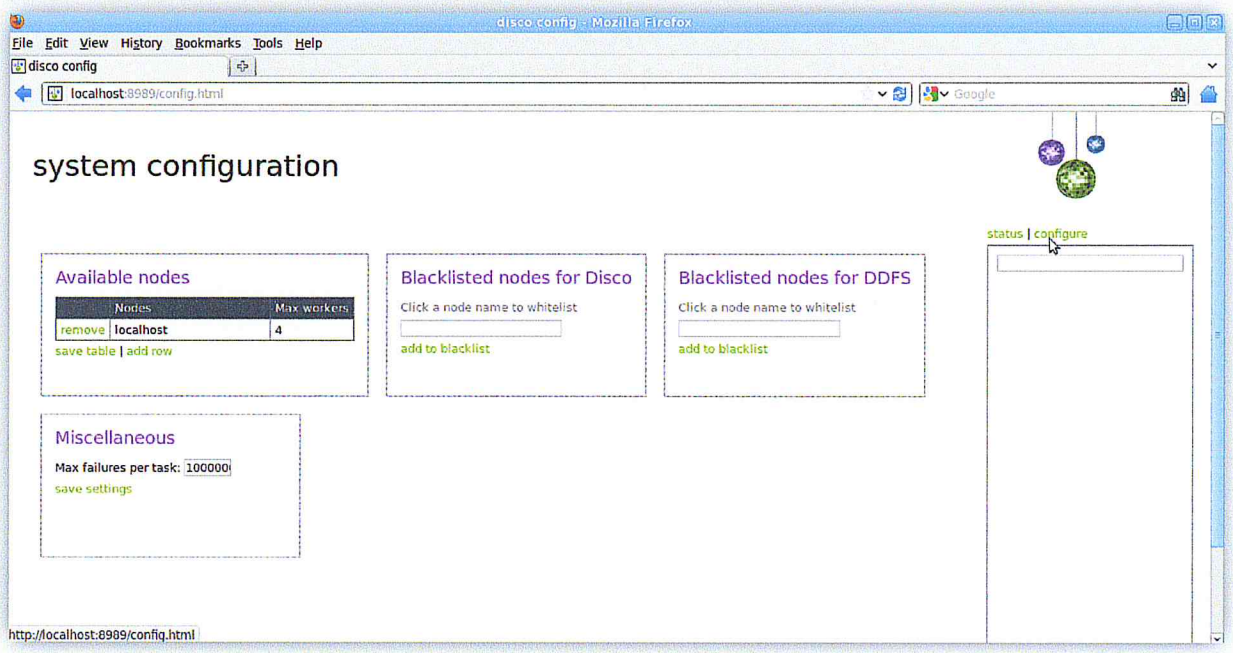


Figure 38 : L'interface web du Disco

A ce niveau, l'utilisateur peut visualiser et consulter concrètement les schémas tags/blobs.

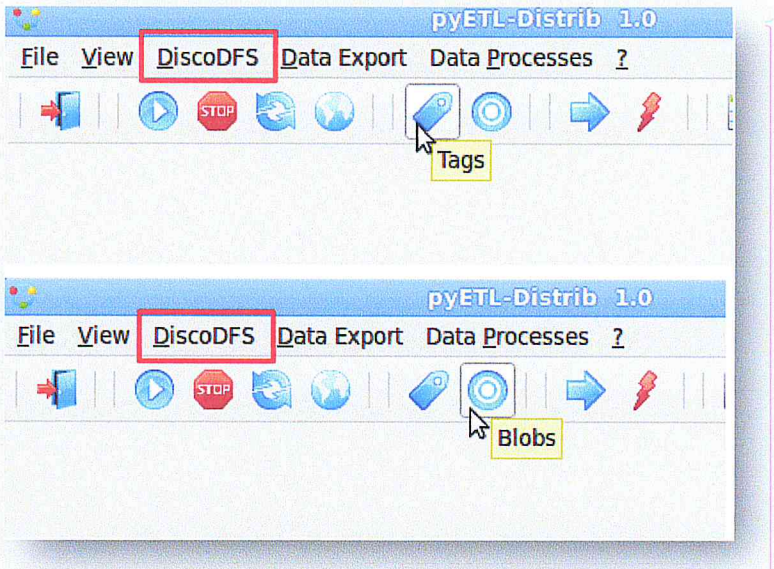


Figure 39 : Affichage des Tags et Blobs du DDFS

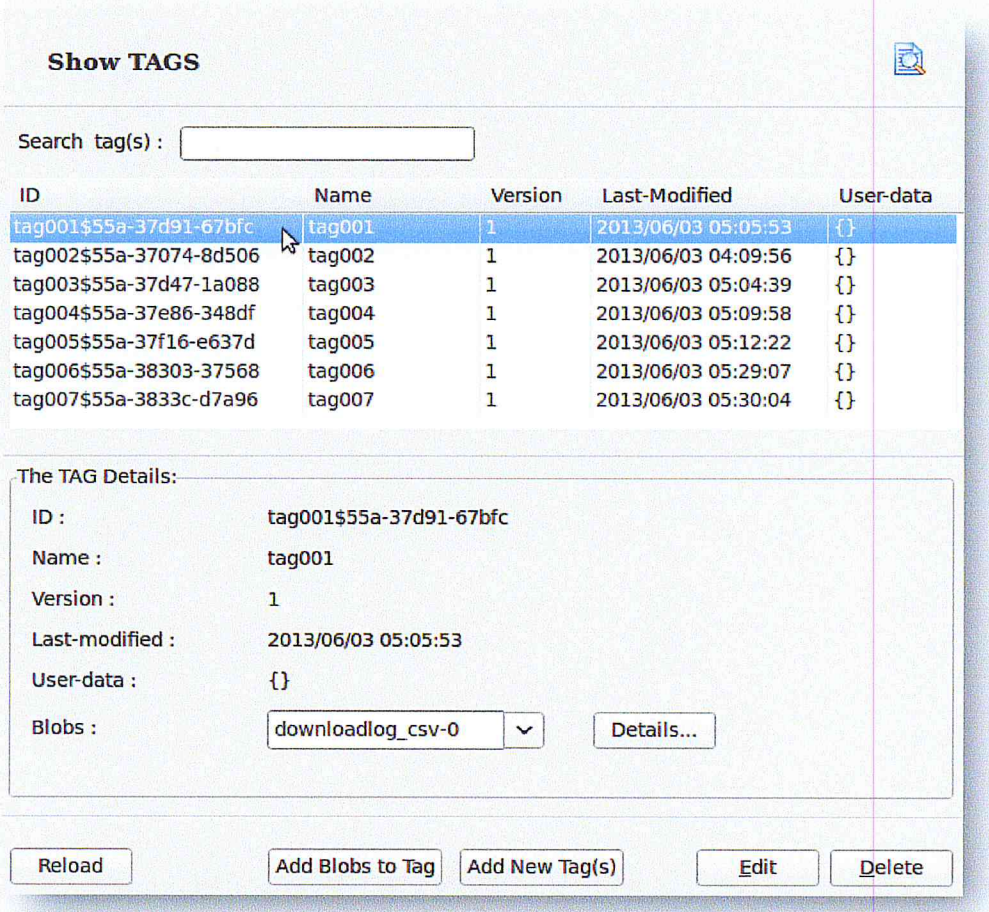


Figure 40 : Consultation des Tags du DDFS

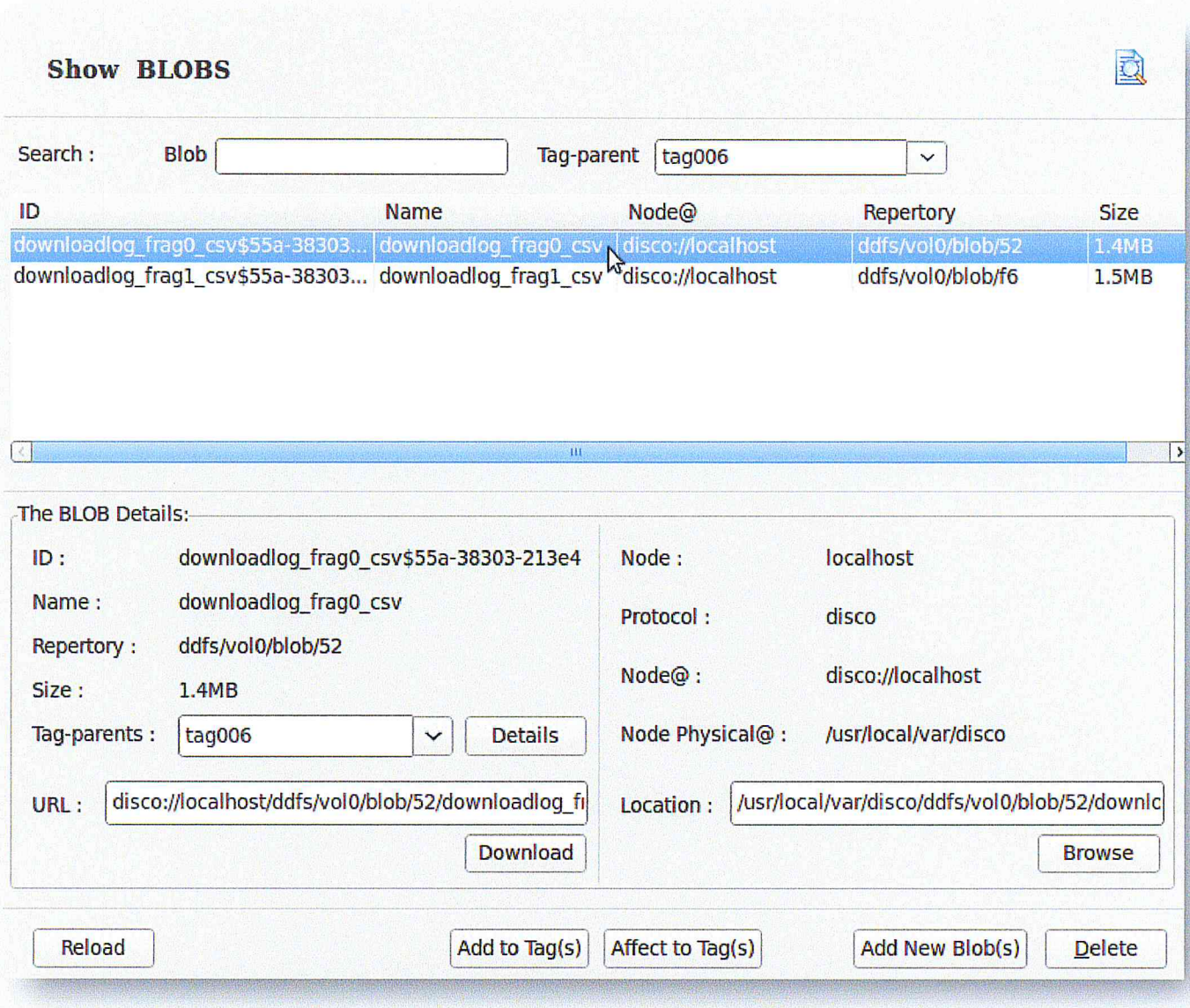


Figure 41 : Consultation des Blobs du DDFS

Après avoir montré notre vulgarisation de l'environnement Disco, nous présentons maintenant l'exportation de données avec un exemple (XML → CSV). L'utilisateur spécifie le fichier XML à exporter et l'emplacement du fichier CSV de destination. Ainsi qu'un paramètre principal qui est le « tag » qui représente la balise de l'élément XML qui couvre un enregistrement de données.

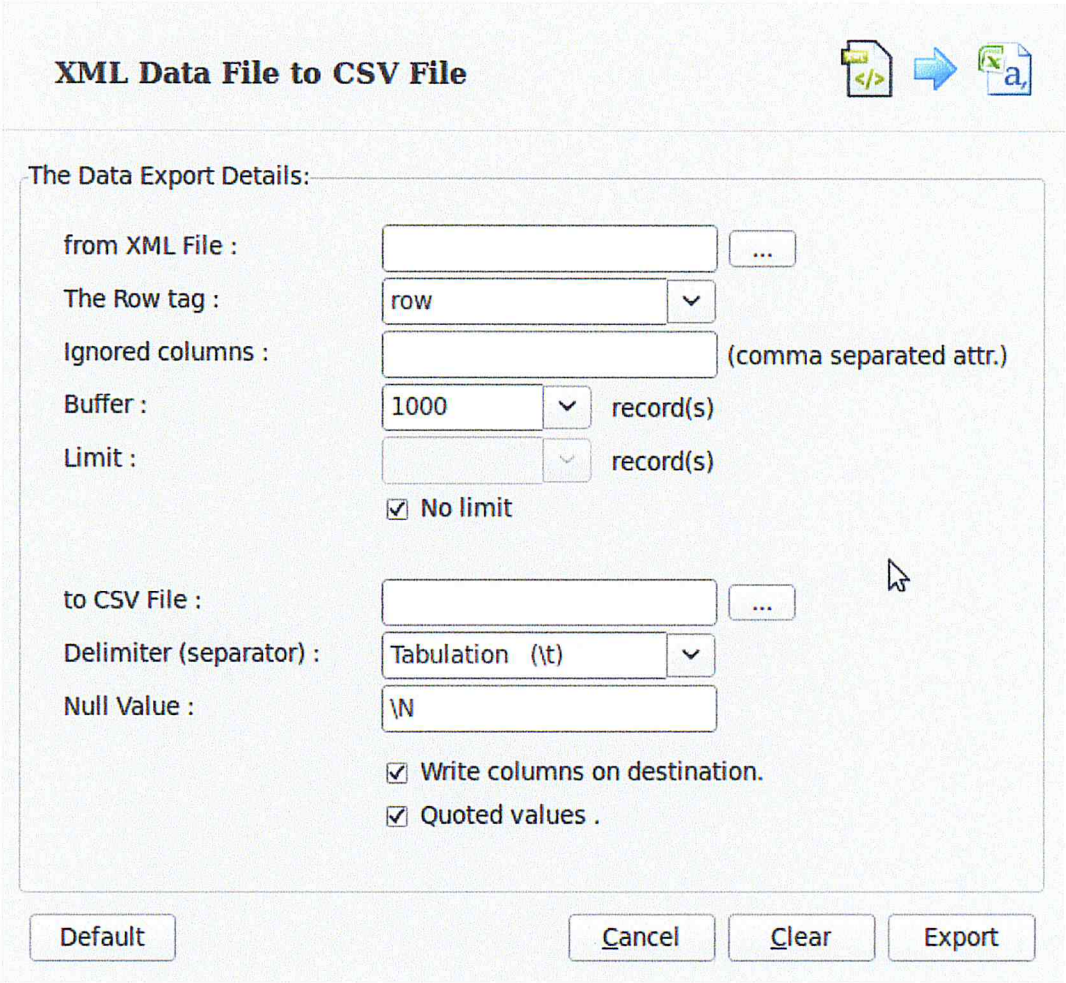


Figure 42 : Exportation de données XML/CSV

A ce niveau, nous passons aux interfaces graphiques qui servent à la définition de notre processus. L'interface de configuration du processus comporte deux parties, la partie gauche y est pour la définition des différentes tables de données et la partie à droite est pour la définition des paramètres globaux du processus qui sont partagé pour toutes les tables.

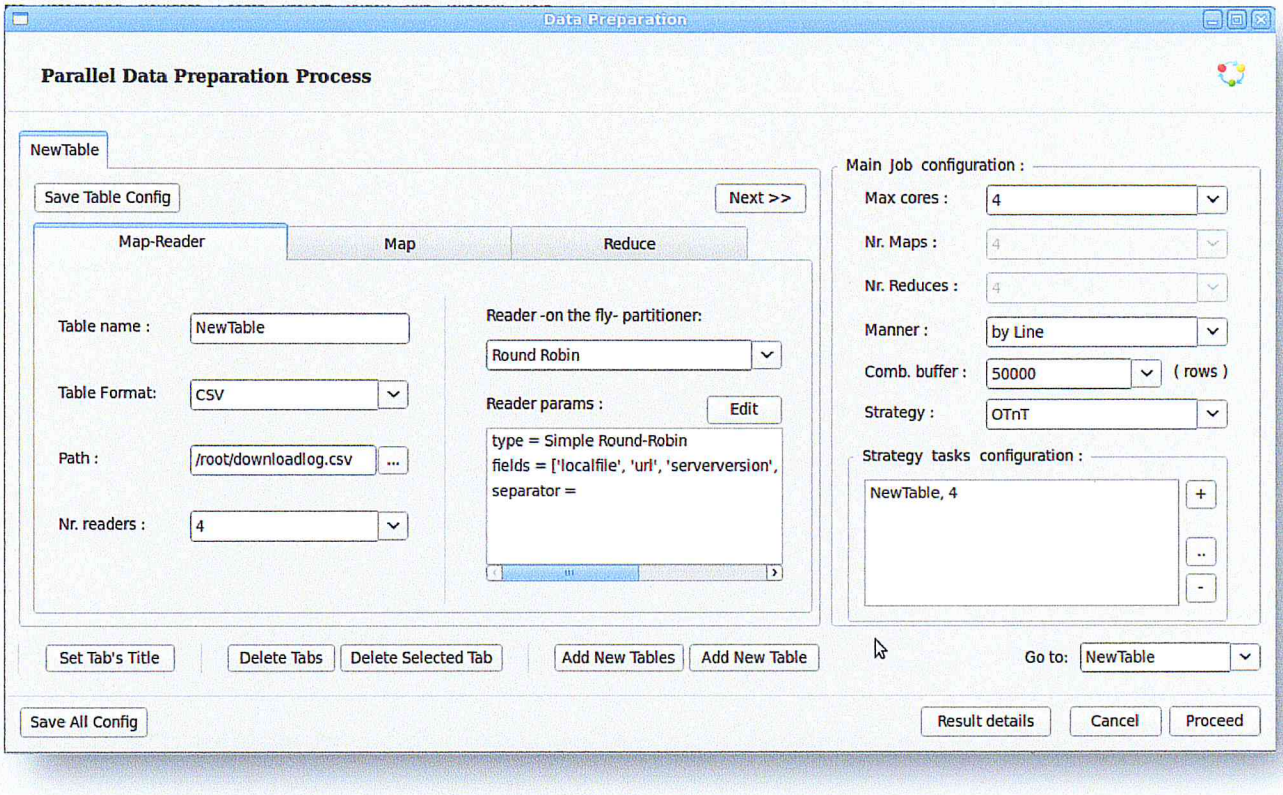


Figure 43 : Définition de notre processus parallèle

Les tables de données (TD) sont définies chacune dans un onglet. Chaque table comporte ces propres onglets où chaque onglet est lié à une phase du processus.

Ainsi, l'onglet MapReader est pour la phase d'extraction. La Figure suivante montre aussi la définition d'un partitionneur à la volée pour cette phase là.

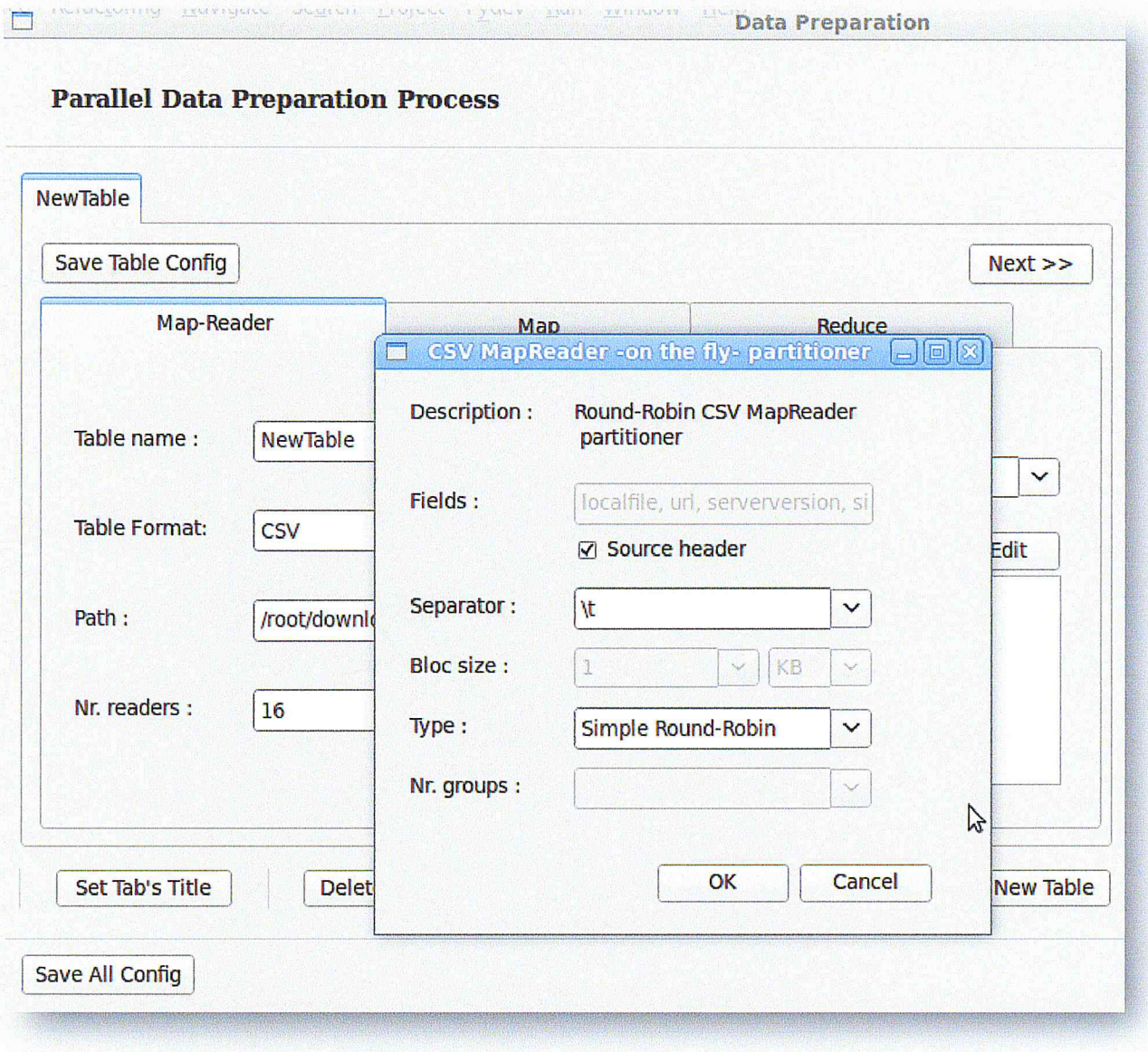


Figure 44 : Définition d'un partitionneur à la volée pour le MapReader

L'onglet Map sert à définir les différents transformateurs pour cette phase de transformation. La Figure montre aussi la définition d'un transformateur de type « Projection ».

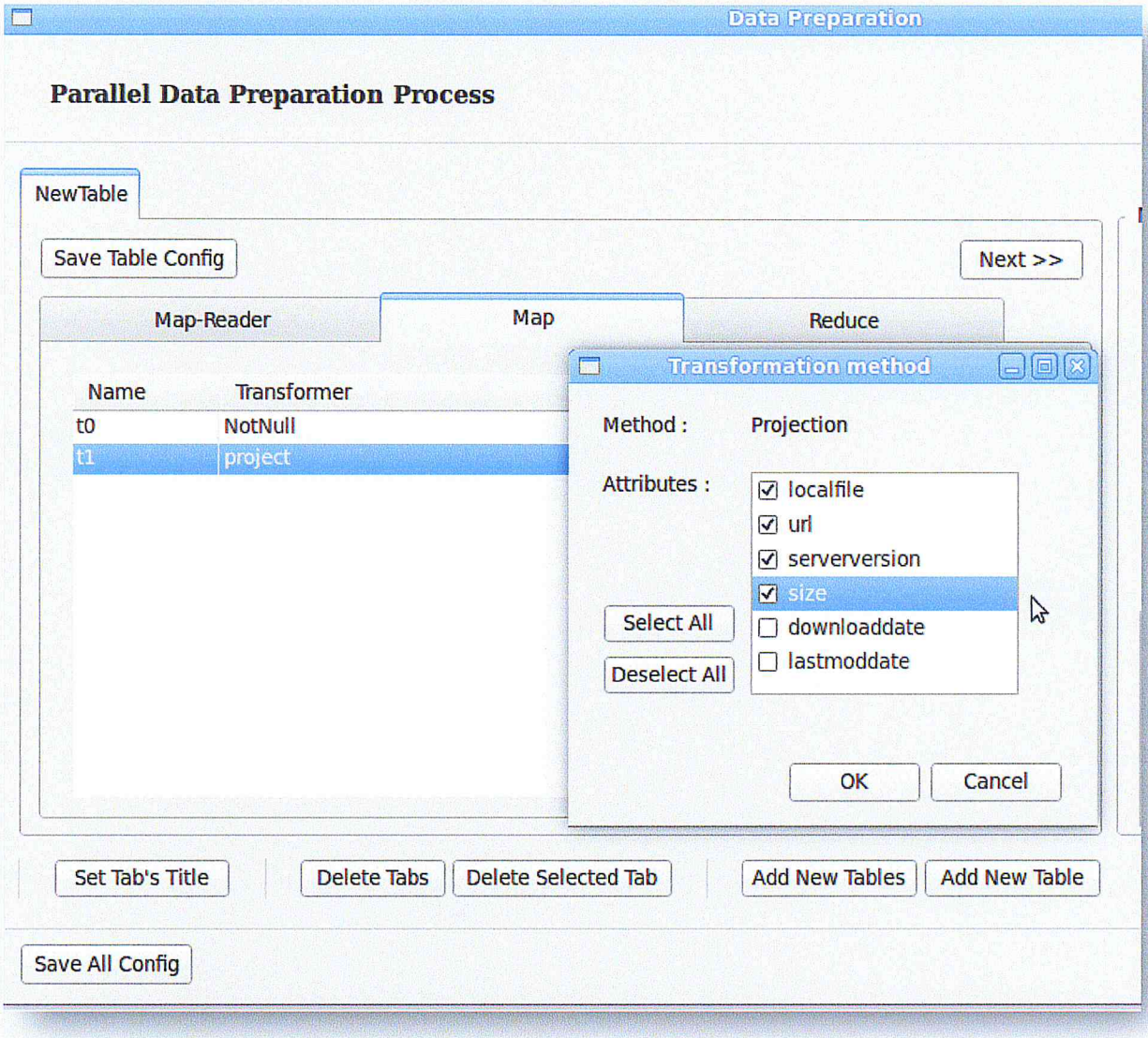


Figure 45 : Définition des transformateurs pour le Map

L'onglet Reduce sert à définir la dernière phase qui est le chargement de données. L'utilisateur définit les différents paramètres concernant les partitions tels que la ligne des attributs, le suffixe d'une partition, le délimiteur, etc. Ainsi que le tag pour la pousse de données vers le DDFS.

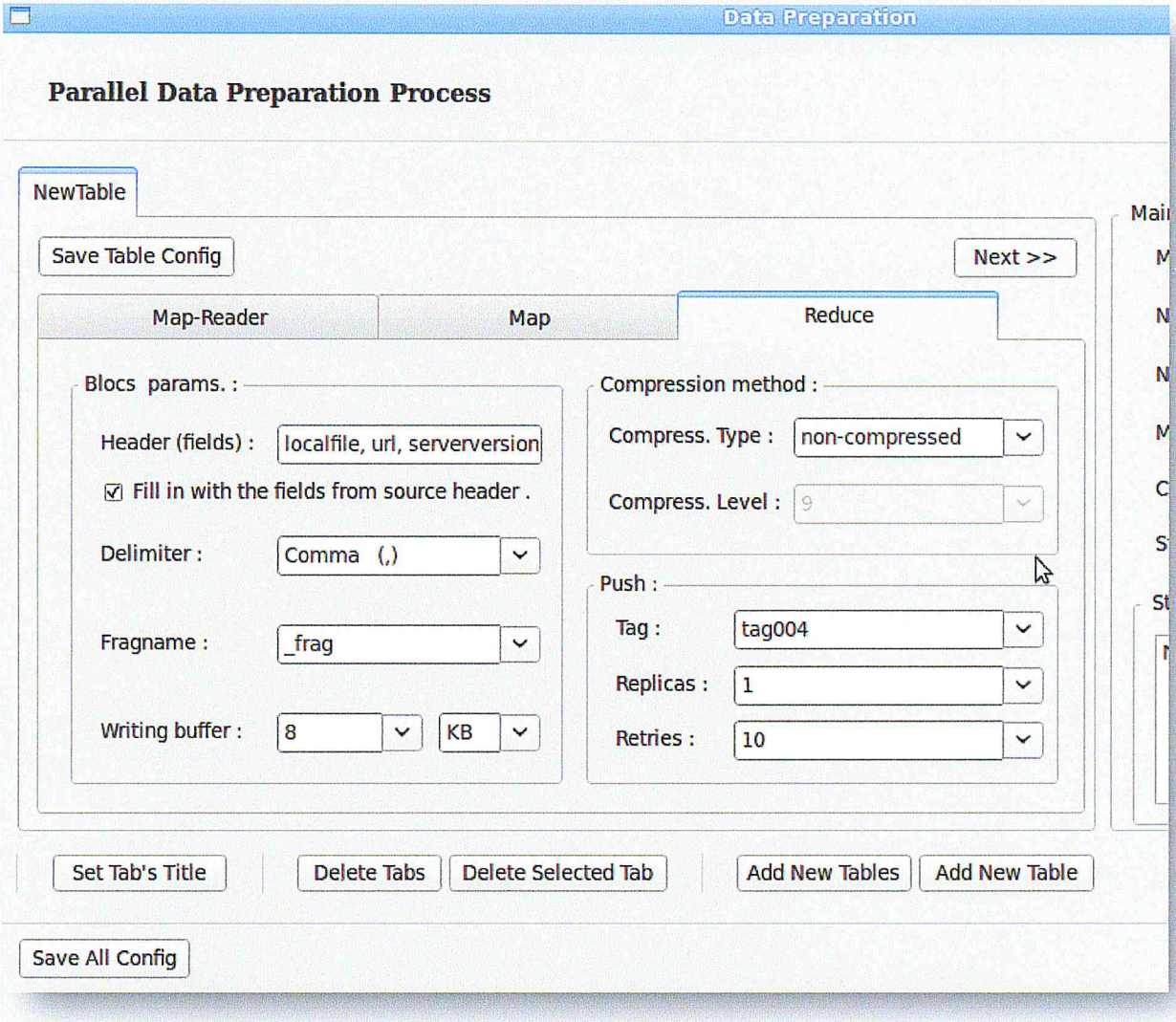


Figure 46 : Définition des paramètres du chargement dans le Reduce

Après la définition de toutes les tables et exécution du processus, voici à quoi ressemblent les résultats retournés.

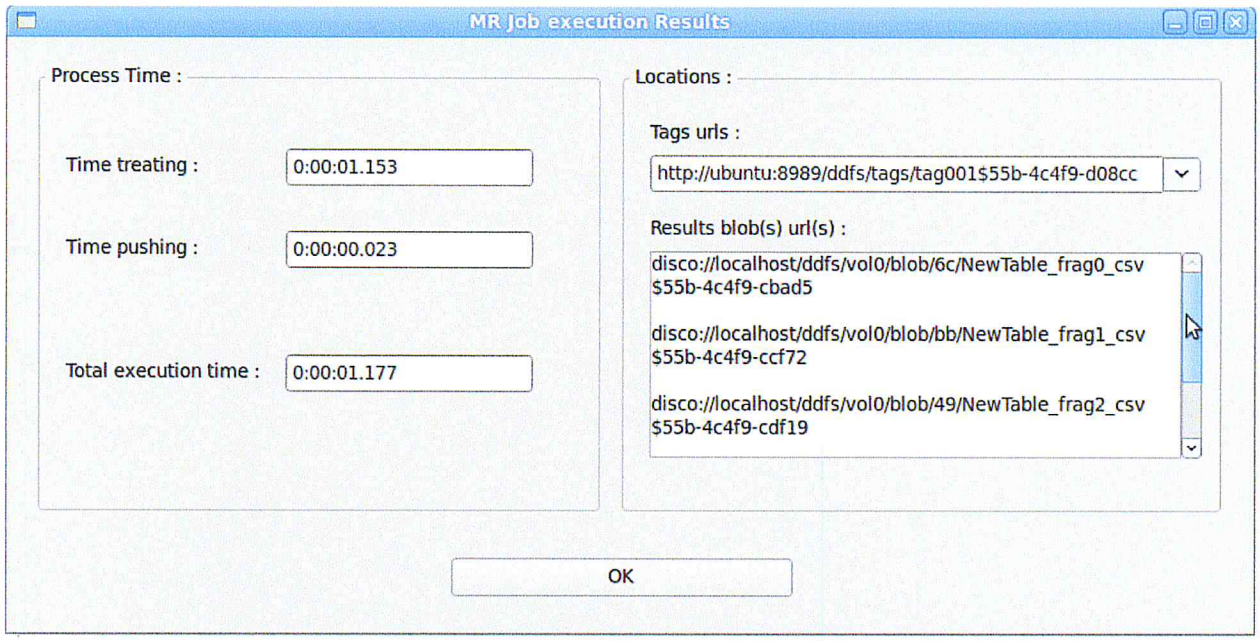


Figure 47 : Résultats de l'exécution du processus

Les résultats comprennent le temps de traitement (compté de la phase extraction jusqu'au chargement local temporaire), le temps de pousse de données vers le DDFS et le temps total. Ainsi que, les URL des tags et des blobs qui font les partitions.

V. Tests et validation

Les performances des temps de traitement sont un critère important pour les outils ETL. Avant d'évaluer les performances de notre système, voici les caractéristiques techniques de l'environnement des tests :

- Processeur Intel Core i3-370M avec 4 cores de 2.40 GHz
- 4 Go de RAM
- Système d'exploitation Ubuntu version 10.04 LTS
- Environnement Disco version 0.4.5

- Interpréteur Python 2.6.5
- Tests lancés via l'interface graphique
- Exécution avec le minimum possible d'applications tournant en arrière plan.

Pour pouvoir évaluer les performances de notre processus parallèle, nous avons effectué plusieurs tests dans différents scénarios. Voici, dans ce qui suit, les deux tests essentiels.

V.1. Test 1

Ceci consiste à évaluer les latences d'exécution du processus ETL/MR de distribution de données pour les différentes sources de données (CSV, XML, TDR) par les différentes méthodes de partitionnement (Simple, Round Robin, Hachage des attributs et Hybride). Les mêmes données de tests ont été générées dans les trois modèles de données différents. Voici les détails de ce cas de test :

- 1 Go de données traitées pour chaque table de données, ce qui fait un peu plus de 25 millions d'enregistrements.
- Fragmentation en 16 partitions (chacune des tables de données sera traitée par 4 mappers et 4 reducers).
- Mode d'extraction par enregistrement
- Buffers de lecture/écriture (pour les fichiers) par défaut (8ko).

Dans ce qui suit les résultats obtenues, les temps d'exécution sont évalués en minutes:

	Simple	Round Robin	Round Robin/groupes	Hachage	Hybride
CSV	5,13	5,39	6,65	8,22	8,92
XML	23,84	29,7	30,05	30,92	31,5
SQL	.	14,06	14,47	15,03	15,93

Tableau 4 : Temps d'exécution du processus parallèle

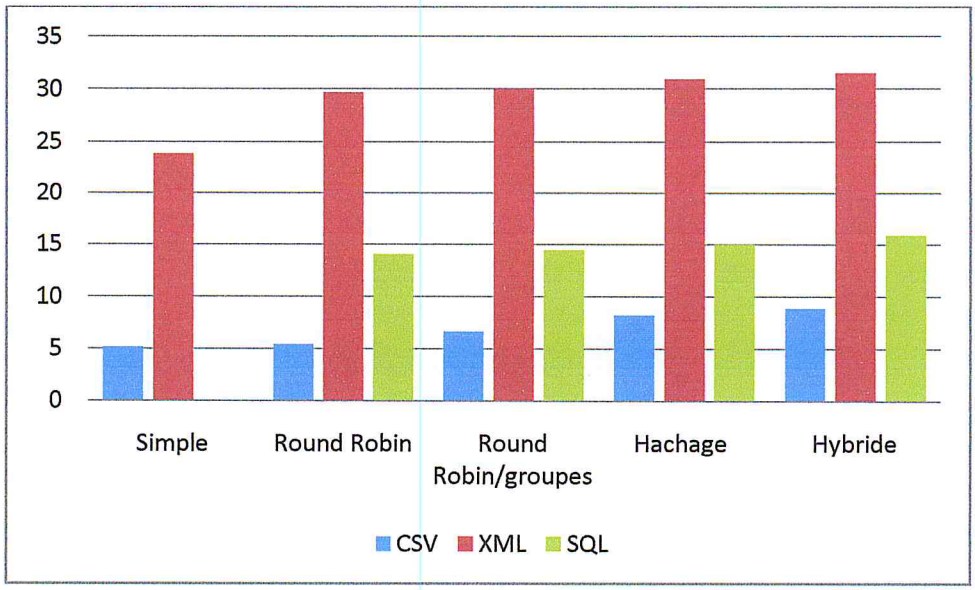


Figure 48 : Diagramme des latences d'exécution du processus parallèle

Le cas d'une table de données relationnelle pour un partitionnement à la volée simple n'apparaît pas dans le test car celui là n'est implémenté que pour les tables de données de type fichiers.

Cependant, les temps de traitements sont avérés beaucoup trop longs pour le fichier XML par rapport aux autres. Cela est principalement dû à la syntaxe du fichier XML qui nécessite une analyse syntaxique pour l'extraction des données, ce qui alourdit le processus. Cela confirme que XML est beaucoup moins performant pour le stockage intensif.

Les temps des traitements d'une table TDR font la moyenne des temps des traitements du fichier XML, mais beaucoup plus longs par rapport au cas du fichier CSV. Le processus dans le cas du format CSV est le plus rapide. Cela est dû à la simplicité de la structure délimitée du fichier CSV qui ne nécessite pas des traitements complexes lors de l'extraction.

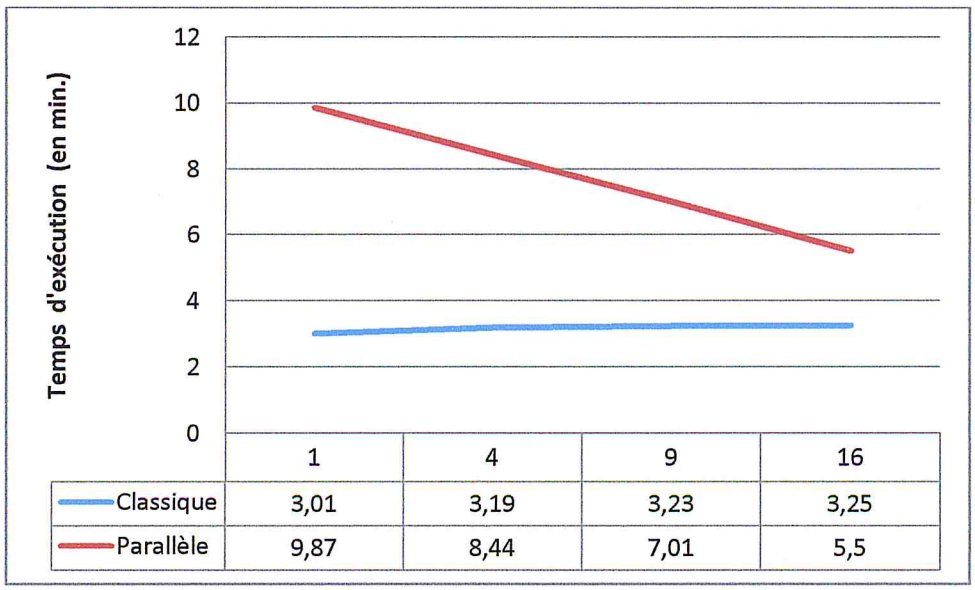
Nous remarquons aussi que le partitionnement simple fut le plus rapide pour tous les types de données.

V.2. Test 2

Il s'agit du même cas de test précédent excepté que là nous le faisons pour les deux processus : classique et parallèle (à titre de comparaison). La table de données de type fichier CSV est traitée par un partitionneur simple à différentes reprises ; pour une fragmentation de 1, 4, 9 et 16 partitions. Pour ce faire, nous impliquons pour le cas parallèle 'x' mappers et 'y' reducers pour avoir (x * y) partitions. Ce qui fait que nous utilisons :

- 1 mapper et 1 reducer pour avoir une seule partition
- 2 mappers et 2 reducers pour avoir 4 partitions
- 3 mappers et 3 reducers pour avoir 9 partitions
- 4 mappers et 4 reducers pour avoir 16 partitions

Voici les résultats obtenus du test:



	1	4	9	16
Classique	3,01	3,19	3,23	3,25
Parallèle	9,87	8,44	7,01	5,5

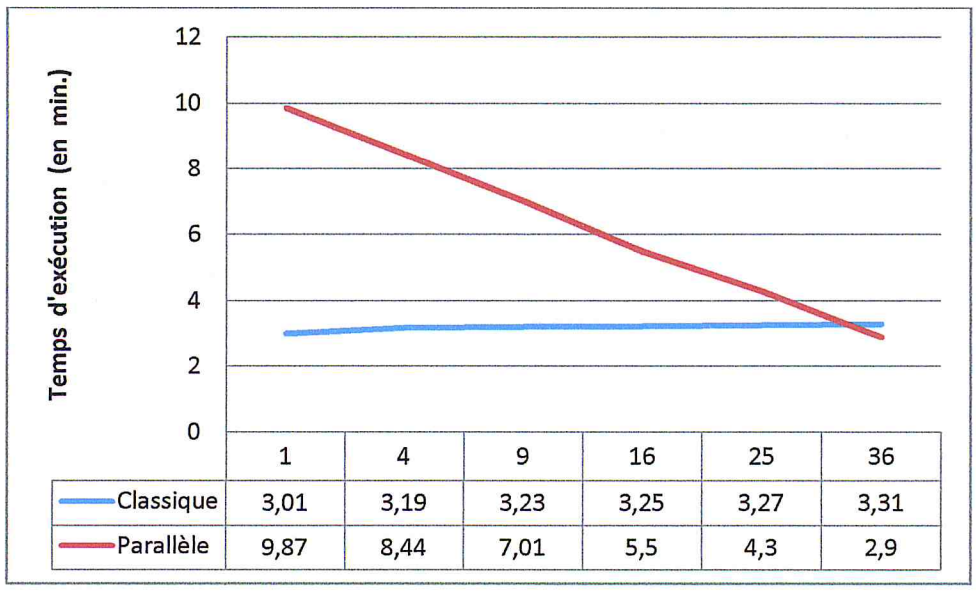
Figure 49 : Graphe comparatif entre le processus classique et parallèle

Le graphe illustré par la Figure 49 représente les temps d'exécution de l'ETL classique et l'ETL/MR de distribution dans différents niveaux du partitionnement (1, 4, 9, 16).

D'après le premier arc qui représente le résultat d'exécution de l'ETL traditionnel, nous remarquons que son temps d'exécution augmente légèrement avec l'augmentation du niveau de partitionnement.

Par contre, le deuxième arc qui trace le cas de l'ETL de distribution basé sur le MapReduce, le temps d'exécution diminue considérablement avec la montée du niveau de partitionnement. Cela est dû à l'implication de plus de mappers et reducers pour satisfaire un niveau du partitionnement plus haut. Dans ce cas, le niveau du partitionnement représente le niveau de performance pour le processus parallèle vu la baisse très importante dans son temps d'exécution.

Bien que l'ETL classique fut le plus rapide pour ces 4 niveaux de partitionnement, le tracé du processus parallèle pourrait le rattraper dans un autre niveau pas loin (vu que la différence de temps est moindre : moins de 3 min. dans le dernier niveau) s'il continue dans cette évolution. Le graphe qui suit représente nos prévisions pour deux autres niveaux de partitionnement (25 et 36 partitions).



	1	4	9	16	25	36
Classique	3,01	3,19	3,23	3,25	3,27	3,31
Parallèle	9,87	8,44	7,01	5,5	4,3	2,9

Figure 50 : Graphe comparatif entre le processus classique et parallèle (prédiction)

Le graphe démontre que le processus parallèle est le plus rapide dans le niveau de 36 partitions avec une continuité de diminution importante.

Nous pouvons constater donc que le processus parallèle peut dépasser le processus classique en performance dans un niveau plus haut que celui de 25 partitions.

Nous concluons donc que notre processus ETL de distribution basée sur le MapReduce est beaucoup plus performant en termes de latences d'exécution par rapport à l'ETL de distribution classique. Autrement dit, la volumétrie des sources de données ne pose plus un problème d'alourdissement en appliquant notre système. Grâce à ces résultats, nous pouvons dire que nous avons atteint nos objectifs.

VI. Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de notre outil qui est le « pyETL-Distrib. » ainsi que ces différentes fonctionnalités en exposant ces interface d'utilisation que nous avons conçu de la manière la plus dynamique possible. Ensuite, nous avons procédé aux tests pour la validation de notre système. Ces tests nous ont révélé la qualité de notre outil.

Conclusion générale

Conclusion générale

Depuis l'adoption des bases de données par les entreprises, l'intégration de données a toujours été au centre de leurs préoccupations. L'intégration de données doit faire face à la problématique d'hétérogénéité, de distribution et de la volumétrie des sources de données. Pour résoudre cette problématique, il existe des approches qui s'appuient sur des outils appelés ETL. Ceux-ci permettent d'extraire des données depuis plusieurs sources de données, de les transformer et les charger dans une ou plusieurs cibles. Les ETL sont nombreux sur le marché mais la plupart des solutions sont lourdes, propriétaires et onéreuses.

Pour pouvoir traiter des volumes de données qui se métamorphosent de façon continue, il devient nécessaire d'utiliser la puissance combinée de nombreux ordinateurs. C'est dans ce contexte qu'a été développé le modèle de calcul à hautes performances dit MapReduce, qui permet d'automatiser la parallélisation de certains processus. Les traitements lourds de données peuvent ainsi être conçus comme une suite de pairs d'opérations « map » et « reduce ». MapReduce automatise le partitionnement des tâches et permet donc de traiter de grands volumes de données dans un temps raisonnable.

Récemment, Liu et *al.* (2011) ont présenté un prototype appelé ETLMR dans le quel ils ont parallélisé les tâches ETL dans un environnement MapReduce. Leur objectif concernait l'accélération des performances du processus ETL. Néanmoins, la problématique des performances d'un processus ETL reste d'actualité surtout avec l'accroissement de la volumétrie des sources de données qui intègre des contenus sur Internet comme les réseaux sociaux et autres.

Dans ce domaine, nous nous sommes plus particulièrement intéressés au problème de la performance du processus ETL en partant des limites qui subsistent dans le prototype ETLMR. Nous avons conjointement abordé dans ce travail la problématique du volume des données et la performance des tâches ETL, et proposer une solution par fragmentation, puis par répartition des données (appelée communément parallélisation des données).

Pour cela, nous nous sommes intéressés dans un premier temps à la fragmentation des données et nous avons proposé des stratégies et des méthodes qui sont à notre connaissance les premiers dans ce domaine. Dans un second temps, nous avons adapté ces techniques pour l'exécution parallèle des tâches ETL en MapReduce. Nous avons finalement mis en œuvre un processus ETL de répartition des données sur une infrastructure de fichiers distribuée dans un environnement Disco de MapReduce.

Une interface utilisateur (graphique ou script) a été mise au point afin d'intégrer les différents modules de l'outil ETL dans une même application. Cette interface a été pensée de manière à ce qu'elle soit la plus dynamique possible, elle permet en effet la définition des différentes phases du processus ETL dans les primitives MapReduce. Cela est visible au niveau de l'interface graphique. Notons que, cette phase qui consistait à mette à disposition des utilisateurs deux interfaces possibles, nous a pris énormément de temps.

La mise en œuvre et l'évaluation des contributions mentionnées ont nécessité un effort considérable. Les principaux défis que nous avons dû surmonter proviennent de plusieurs facteurs : la complexité du projet ETLMR que nous avons étudié longtemps pour identifier ses limites et contribuer pour une amélioration, la nécessité de commencer par la mise en œuvre du processus ETL classique pour procéder après à sa migration dans le modèle MapReduce de manière fine afin d'obtenir de meilleures performances, ainsi que l'habillage graphique de ces deux processus qui était dur et complexe vu que le MapReduce jusqu'à maintenant est adapté beaucoup plus aux scripts qu'aux interfaces visuelles.

Les résultats expérimentaux que nous avons obtenus montrent que notre approche de fragmentation et de répartition sur un système de fichiers distribué est efficace. Nous avons atteint notre objectif de maitriser les volumes de données et d'améliorer le temps de traitement des tâches ETL.

Perspectives

Dans le but d'enrichir notre système, certaines améliorations peuvent être ajoutées par la suite, elles sont définies comme suit :

- Compléter la phase de transformation de données par des transformateurs d'agrégation et de fusion
- Etendre notre système à supporter d'une manière complète le partitionnement vertical
- Intégrer des parseurs XML autre que « ElementTree XML » puisque le traitement des sources XML s'avère le plus long, et évaluer les résultats à titre de comparaison
- Que ce soit pour le partitionnement à la volée ou normal, personnaliser de nouvelles techniques et méthodes de fragmentation, par exemple :
  - Le partitionnement en gammes : il sélectionne une partition en déterminant si la clef de partition est incluse dans un certain éventail de données. Un exemple serait une partition pour toutes les lignes où la colonne code postal a une valeur entre 75000 et 75999.
  - Le partitionnement par liste : une partition contient une liste de valeurs. Si la clef de partitionnement dispose de l'une de ces valeurs, la partition est choisie. Par exemple, toutes les lignes où la colonne Pays contient Islande, Norvège, Suède, Finlande or Danemark peuvent former une partition des pays nordiques.
- Les résultats du processus, tel qu'implémenté dans notre système, sont versés dans des blocs de fichiers plats (csv). Ces derniers peuvent être considérés comme des cubes de données distribués alimentés à la demande (ou à la volée) puisque leur taille est nettement inférieure à celle d'un entrepôt de données. Par contre, si nous souhaitons que cette plateforme soit utilisée pour l'entreposage (alimentation d'un entrepôt de données), deux perspectives sont à étudier :
  - Intégrer la plateforme ETL mise en œuvre avec un environnement SGBR comme Oracle, SQL Server ou d'autres, pour alimenter un

entrepôt hébergé sur ces systèmes adaptés au stockage des entrepôts de données

- Intégrer dans la plateforme un moteur de données NoSQL (modèle en colonnes, basé document, …) capable de stocker des entrepôts de données qui prennent du volume de manière continue.

Bibliographie

[TES 00] Teste, O. (2000). *Modélisation et manipulation d'entrepôts de données complexes et historisées.* Toulouse: Thèse présentée pour l'obtention du Doctorat à l'Université Paul Sabatier.

[MAI 06] Maisons, D. (2006). *Datawarehouse et datamining.* Rapport présenté pour l'obtention de l'Examen Probatoire en Systèmes d'Information.

[INM 05] Inmon, W. H. (2005). *Building the Data Warehouse.* Indiana: Wiley Publishing Inc - Indianapolis, 4ème édition.

[KIM 04] Kimball, R., & Caserta, J. (2004). *The data warehouse ETL toolkit : practical techniques for extracting, cleaning, conforming, and delivering data.* Wiley Publishing, Inc.

[DGH 04] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *Dans le 6ème symposium sur l'OSDI (Operating Systems Design & Implementation)* (pp. 137-150). San Francisco, California: USENIX Association.

[TAN 07] Tanenbaum, A. (2007). *Distributed Systems : Principles and Paradigms* (éd. 2ème). Pearson Prentice Hall.

[CDK 12] Coulouris, G., Dollimore, J., Kindberg, K., & Blair, G. (2012). *Distributed System: Concept & Design* (éd. 5ème). Adison-Wesley (Pearson Education, Inc).

[ETLMR 11] (ETLMR) Xiufeng, L., Christian, T., & Torben Bach, P. (2011). *ETLMR: A Highly Scalable Dimentional ETL Framework based on MapReduce*. DB Tech Reports.

[ETL/MR 11] (ETLMR) Xiufeng, L., Christian, T., & Torben Bach, P. (2011). *The ETLMR MapReduce-Based ETL Framework. Proceedings of the 23rd International Conference*, SSDBM (Scientific and

Statistical Database Management), (pp. 586-588). Portland, USA.

[DAN 03] DANG NGOC T, *Fédération de données semi- structurées avec XML*, (2003).

[GAR 02] GARDARIN G, Base de données, (2002)

Webographie

[1] EISTI. (2008). *Le Décisionnel*. Consulté le 30/12/2012, sur Site du mastère spécialisé de l'EISTI - Informatique Décisionnelle: http://informatique-decisionnelle.masteres.eisti.fr/index.php/lenseignement/le-decisionnel

[2] Tranchant, M. (23/02/2012). *Qu'est-ce que l'informatique décisionnelle ?* Consulté le 31/12/ 2012, sur developpez-com: http://business-intelligence.developpez.com/tutoriels/quest-ce-que-la-bi/

[3] SANTEL LEBORGNE M. *Entrepôts de données*. Consulté le 31/12/2012, sur Institut Gaspard Monge - Université Marne-la-Vallée: http://www-igm.univ-mlv.fr/~dr/XPOSE2005/entrepot

[4] Gilleron, R., & Tommasi, M. (06/2000). *Découverte de connaissances à partir de données*. Consulté le 31/12/2012, sur Grappa - Université Lille3: http://www.grappa.univ-lille3.fr/polys/fouille/

[5] Fernandez, A. (2004). *Performance Management et Business intelligence - Le portail francophone piloter la performance*. Consulté le 31/12/2012, sur piloter-org:

http://www.piloter.org/

[6] OSBI. (2007). *Les ETL.* Consulté le 31/12/2012, sur OSBI, Open Source Business Intelligence: http://www.osbi.fr/business-intelligence/les-etl/

[7] Dubois, L. (01/09/1998). *Achieving Enterprise Data Quality.* Consulté le 31/12/2012, sur Tdan-com: http://www.tdan.com/view-articles/4269

[8] Langseth, J. (2004). *Real-Time Data Warehousing: Challenges and Solutions.* Consulté le 31/12/2012, sur DSSResources: http://dssresources.com/papers/features/langseth/langseth02082004.html

[9] Giga Tribe. *Qu'est ce que le peer-to-peer (p2p).* Consulté le 31/12/2012, sur GigaTribe: http://www.gigatribe.com/fr/aide-introduction-p2p

[10] Bojoly, M. (09/05/2012). *Les Patterns des Grands du Web – Sharding.* Consulté le 31/12/2012, sur Blog Octo: http://blog.octo.com/sharding/

[11] CETMEF. (12/07/2011). *Conception d'un système à haute performance - Les systèmes de fichiers.* Consulté le 31/12/2012, sur CETMEF developpement durable Gouv Fr: http://www.cetmef.developpement-durable.gouv.fr/conception-d-un-systeme-a-haute-a17.html

[12] Yahoo. *Hadoop at Yahoo!* Consulté le 31/12/2012, sur Developper Yahoo: http://developer.yahoo.com/hadoop/

[13] NRC. (2008). *Disco: massive data - minimal code.* Consulté le 31/12/2012, sur Disco project: http://discoproject.org/

[14] Curzon Nassau, *MastPoint : CSV File Format Specification*, Consulté le 28/03/2013 sur http://mastpoint.curzonnassau.com/csv-1203/

[15] http://www.erlang.org/

[16] http://www.openssh.org/fr/

[17] http://fr.wikipedia.org/wiki/Ubuntu

[18] http://www.python.org/

[19] http://www.eclipsetotale.com/

[20] http://www.wxpython.org/what.php

[21] https://fr.wikipedia.org/wiki/PostgreSQL

[22] http://initd.org/psycopg/docs/

[23] http://pymotw.com/2/xml/etree/ElementTree/