# PF-ETL : vers l'intégration de données massives dans les fonctionnalités d'ETL

**Mahfoud Bala[1], Omar Boussaid[2], Zaia Alimazighi[3], Fadila Bentayeb[2]**

*1. LRDSI, Université Saad Dahleb de Blida*
*BP 270, route de Soumaa, Blida, Algérie*
*mahfoud.bala@gmail.com*

*2. ERIC, Université Lumière, Lyon 2*
*5, Avenue Pierre Mendès, 69676 Bron Cedex – France*
*{omar.boussaid, fadila.bentayeb}@univ-lyon2.fr*

*3. LSI, Université des Sciences et des Technologies Houari Boumédiène, Alger*
*BP 32, EL ALIA 16111 Bab Ezzouar, Alger, Algérie*
*zalimazighi@usthb.dz*

RESUME. *Un processus ETL (Extracting-Transforming-Loading) est responsable d'extraire des données à partir de sources hétérogènes, les transformer et enfin les charger dans un entrepôt de données. Les nouvelles technologies, particulièrement Internet et le Web 2.0, générant des données à une vitesse croissante, ont mis les systèmes d'information (SI) face au défi du Big Data. Ces données sont caractérisées par, en plus de leur volumétrie et la vitesse avec laquelle elles sont générées, une hétérogénéité plus importante suite à l'émergence de nouvelles structures de données. Les systèmes d'intégration et l'ETL en particulier doivent être repensés et adaptés afin de faire face à l'impact des Big Data. Dans ce contexte et pour mieux gérer l'intégration de données massives, nous proposons une nouvelle approche du processus ETL pour lequel nous définissons des fonctionnalités pouvant s'exécuter sur un cluster selon le modèle MapReduce (MR).*

ABSTRACT. *ETL process (Extracting, Transforming, Loading) is responsible for extracting data from heterogeneous sources, transforming and finally loading them into a data warehouse. New technologies, particularly Internet and Web 2.0, generating data at an increasing rate, put the information systems (IS) face to the challenge of Big Data. These data are characterized by, in addition to their excessive sizes and speed with which they are generated, greater heterogeneity due to the emergence of new data structures. Integration systems and ETL in particular should be revisited and adapted to cope with the impact of Big Data. In this context and to better manage the integration of Big data, we propose a new approach to ETL process for which we define features that can be run easily on a cluster with MapReduce (MR) model.*

MOTS-CLES : *ETL, Données massives, Entrepôts de données, MapReduce, Cluster*

KEYWORDS: *ETL, Big Data, Data warehouse, MapReduce, Cluster*

## 1. Introduction

L'utilisation à grande échelle d'Internet, du Web 2.0 et des réseaux sociaux produit, instantanément, des volumes non habituels de données. Des *jobs MapReduce* exécutés en continu sur des *clusters* de *Google* traitent plus de vingt *PetaBytes* de données par jour (cf. Ghemawat et Dean, 2008). Cette explosion de données est une opportunité dans l'émergence de nouvelles applications métiers mais en même temps problématique face aux capacités limitées des machines et des applications. Ces données massives, connues aujourd'hui sous le nom de *Big Data,* sont caractérisées par les trois V (cf. Mohanty et *al.*, 2013) : Volume qui implique la quantité des données qui va au-delà des unités habituelles, la Vélocité qui implique la rapidité avec laquelle ces données se génèrent et doivent être traitées et enfin la Variété qui implique la diversité des formats et structures. En parallèle à l'émergence du *Big Data*, de nouveaux paradigmes ont vu le jour tels que le *Cloud Computing* (cf. Barrie Sosinsky, 2011) et *MapReduce (MR)* (cf. Dean et Ghemawat, 2004). Par ailleurs, de nouveaux modèles de données non-relationnelles émergent, appelés modèles *NoSQL (Not Only SQL)* (cf. Han et *al.,* 2011).

L'objectif de cet article est d'apporter des solutions aux problèmes posés par les *Big Data* dans un environnement décisionnel. Nous nous intéressons en particulier à l'intégration de données massives dans un entrepôt de données. Nous proposons alors un processus d'ETL parallèle, appelé *PF-ETL (Parallel Functionality-ETL),* composé d'un ensemble de fonctionnalités selon le paradigme *MR*. Les solutions proposées par la communauté, dans ce contexte, consistent à instancier le processus ETL sur plusieurs nœuds d'un cluster. Chacune des instances traite une partition des données sources de façon parallèle afin d'améliorer les performances du processus ETL. A notre connaissance, *PF-ETL*, constitue une nouvelle approche dans le domaine de l'intégration de données. Nous définissons tout d'abord un processus ETL à un niveau très fin en le décrivant par un ensemble de fonctionnalités de base. Celles-ci sont conçues avec le modèle *MR* de manière à pouvoir les instancier ainsi que le processus ETL sur les différents nœuds d'un cluster. *PF-ETL* permet alors un parallélisme selon le modèle *MR* à deux niveaux (1) niveau fonctionnalité d'ETL et (2) niveau processus ETL ; ce qui permettra d'améliorer davantage les performances de l'ETL face aux *Big Data*. Plusieurs travaux sur l'ETL existent dans la littérature. Nous proposons une classification de ces travaux, sous forme de familles d'approches, selon des critères relevant de la parallélisation et comparons ensuite notre proposition par rapport à ces approches. Pour valider notre processus *PF-ETL*, nous avons développé un prototype logiciel et mené des expérimentations.

Le présent article est organisé de la manière suivante. Dans la section 2, nous définissons le processus ETL. Dans la section 3, nous exposons un état de l'art sur les travaux dans le domaine de l'ETL. Nous proposons ensuite dans la section 4 notre classification des processus ETL proposés dans la littérature sous forme d'approches selon le critère de parallélisation. La section 5 est consacrée à notre processus ETL parallèle, *PF-ETL*. Nous développons dans la section 6 les différentes expérimentations que nous avons menées et présentons nos résultats. Nous concluons et présentons nos travaux futurs de recherche dans la section 7.

## 2. Processus ETL (*Extracting-Transforming-Loading*)

Le processus ETL est un ensemble de tâches classées en trois catégories. Les tâches d'extraction (E) permettent de se connecter sur diverses sources pour extraire les données les plus pertinentes. Les tâches de transformation (T) préparent les données en vue de leur affecter des propriétés en termes de qualité, de format et de pertinence. Enfin, les tâches de chargement (L), basées sur un schéma de mappage, permettent de charger les données dans l'entrepôt de données (*DW : Data Warehouse*). La couche inférieure de la figure 1 présente la partie statique qui montre les *sources* de données, le *Data Staging Area (DSA)* vers lequel sont rapatriées les données pour être préparées et le *DW* représentant la destination finale des données. Dans la couche supérieure, sont représentées les trois phases d'ETL à savoir *extraction*, *transformation* et *chargement*.

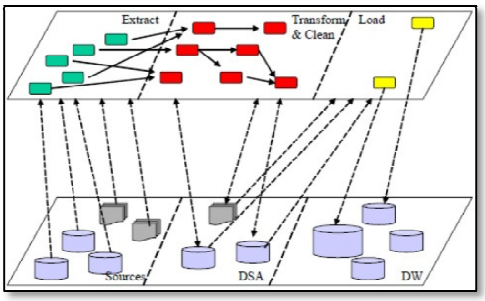


*Figure 1. Environnement d'un processus ETL (cf. Vassiliadis et al., 2002)*

Nous assistons, depuis plus d'une décennie, à une évolution ayant affecté l'aspect statique du processus ETL suite à l'émergence du *Big Data* et aux nouvelles technologies de stockage et son aspect dynamique suite à l'avènement du paradigme *MR* et des environnements *cloud computing*. L'ETL devra s'adapter à cette évolution en intégrant ces nouvelles technologies qui impactent, bien évidemment, son architecture et son processus mais tout en préservant sa vocation qui est l'intégration des données pour des fins d'analyse.

## 3. Etat de l'art

L'une des premières contributions dans le domaine de l'ETL est celle de Vassiliadis et *al.* (2002). Il s'agit d'une approche de modélisation basée sur un formalisme graphique non standard ; *ARKTOS II* étant le prototype mis en œuvre. Trujillo et Luján-Mora (2003) se sont intéressés, eux aussi, à la modélisation de l'ETL dans une approche plus globale basée sur des choix standards, en particulier un formalisme qui s'appuie sur une extension de la notation *UML* (*Unified Modeling Language*). El Akkaoui et Zemányi (2009) ont adopté, quant à eux, la notation standard *BPMN* (*Business Process Model and Notation*) dédiée à la modélisation

des processus métier. Ce travail a été poursuivi par El Akkaoui et *al.* (2011) en proposant un framework de modélisation basé sur un métamodèle dans une architecture *MDD (Model Driven Development)*.

Le travail de Liu et *al*. (2011) s'intéresse aux performances de l'ETL face au *Big Data* et adopte le modèle *MR*. Cette approche a été implémentée dans un prototype appelé *ETLMR*, version *MR* du prototype *PygramETL* (cf. Thomson et Pederson, 2009a). Bala et Alimazighi (2013) ont proposé une modélisation de l'ETL pour le *Big Data* selon le paradigme *MR* en adoptant le formalisme de Vassiliadis et *al.* (2002) enrichi par des notations graphiques pour modéliser les spécificités du modèle *MR*. Récemment, Oliveira et Belo (2013) ont proposé une approche de modélisation qui consiste en une vue résumée du processus ETL et adopte le modèle *Reo* (cf. Arbab, 2003). Nous considérons que cette contribution pourrait être intéressante mais n'est pas suffisamment mature et mérite une personnalisation du modèle *Reo* pour prendre en charge les spécificités de l'ETL. Très récemment, les expérimentations de Misra et *al.* (2013) montrent que les solutions ETL basées sur des frameworks *MR* open source tel que *Hadoop* sont très performantes et moins coûteuses par rapport aux outils ETL commercialisés.

Il apparait à travers cet état de l'art que les contributions portent essentiellement sur la modélisation du processus ETL et l'amélioration de ses performances. Concernant la modélisation du processus ETL, l'approche de Vassiliadis et *al.* (2002) propose un schéma qui intègre la plupart des aspects d'ETL à un niveau très fin, autrement dit l'attribut. Cependant, elle ne prévoit pas de découpage de l'ETL en sous-processus pour une meilleure lisibilité du modèle, chose qui est proposée, dans une démarche *top-down,* par Trujillo et Luján-Mora (2003) et El Akkaoui et Zemányi (2009). Le travail de Bala et Alimazighi (2013) est une contribution de modélisation qui tente d'intégrer les *Big Data* dans un modèle *MR*. Quant aux travaux ayant pour objectif l'amélioration des performances, ceux-ci ont adopté le modèle *MR* qui accélère considérablement l'ETL face aux *Big Data* (cf. Misra et *al.,* 2013). Contrairement à Liu et *al.* (2011), Misra et *al.* (2013) considèrent la phase d'extraction (E) de l'ETL coûteuse et l'ont traitée avec le modèle *MR*. Nous considérons, à l'issue de cet état de l'art, que l'ETL face à l'évolution des données ainsi qu'à l'émergence de nouveaux paradigmes est une problématique qui reste d'actualité.

## 4. Classification des travaux sur le processus ETL

Nous proposons, dans cette section, une classification des travaux sur l'ETL sous forme de deux approches en se basant sur la parallélisation de celui-ci : « approche classique » et « approche basée sur le modèle *MR* ».

### *4.1. Processus ETL classique*

Dans ce papier, nous considérons que le processus ETL est classique lorsque celui-ci s'exécute sur un serveur en une seule instance (une seule exécution en même temps) et où les données sont de taille modérée.

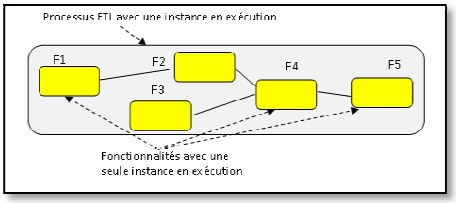


*Figure 2. Processus ETL classique*

Dans ce contexte, seules les fonctionnalités d'ETL indépendantes peuvent s'exécuter en parallèle. Une fonctionnalité d'ETL, telles que *Changing Data Capture* (*CDC*), *Surrogate Key* (*SK*), *Slowly Changing Dimension (SCD)*, *Surrogate Key Pipeline (SKP)*, est une fonction de base qui prend en charge un aspect particulier dans un processus ETL. La fonctionnalité *CDC*, par exemple, permet dans la phase d'extraction d'un processus ETL d'identifier, dans les systèmes sources, les tuples affectés par des changements afin de les capturer et les considérer dans le rafraichissement du *DW*. Dans la figure 2, nous remarquons que les fonctionnalités *F1* et *F3* ou *F2* et *F3* peuvent s'exécuter en parallèle.

### *4.2. Processus ETL basé sur le modèle MapReduce (MR)*

Le processus ETL, avec un schéma classique, ne pourra pas faire face à l'intégration de données massives. Le paradigme *MR* permet de partitionner de gros volumes de données dont chaque partition sera soumise à une instance du processus ETL.

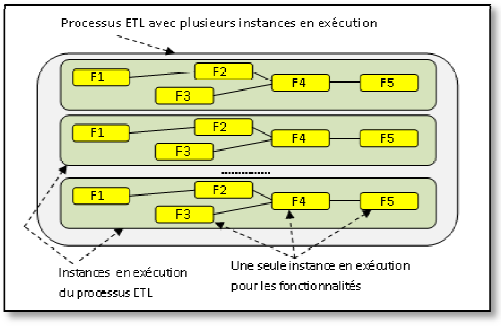


*Figure 3. Processus ETL basé sur le modèle MR*

Comme le montre la figure 3, plusieurs instances du processus ETL s'exécutent en parallèle où chacune d'elles traitera les données de sa partition dans une phase appelée *Map*. Les résultats partiels seront fusionnés et agrégés, dans une phase *Reduce,* pour obtenir des données prêtes à être chargées dans le *DW*.

### *4.3. Classification des travaux sur l'ETL*

Les travaux présentés dans le tableau 1 portent sur la modélisation ou sur l'optimisation du processus ETL mais interviennent tous à un niveau processus. Les fonctionnalités d'ETL étant les fonctions de base autour desquelles s'articule ce processus. Elles méritent une étude approfondie, notamment les plus courantes d'entre-elles, afin de garantir, à un niveau très fin, une robustesse, fiabilité et optimisation du processus ETL.

*Tableau 1. Classification des travaux sur le processus ETL*

| Contributions | Objectif | Classification |
|---|---|---|
| Vassiliadis et *al.* (2002) | Modélisation | Approche Classique |
| Trujillo et Luján-Mora (2003) | Modélisation | Approche Classique |
| El Akkaoui et Zemányi (2009) | Modélisation | Approche Classique |
| Liu et *al.* (2011) | Performances | Approche basée sur *MR* |
| Bala et Alimazighi (2013) | Modélisation | Approche basée sur *MR* |
| Oliveira et Belo (2013) | Modélisation | Approche Classique |
| Misra et *al.* (2013) | Performances | Approche basée sur *MR* |
| Bala et *al.* (2014) | Performances | Approche basée sur *MR* |

## 5. *PF-ETL* : ETL parallèle basé sur des fonctionnalités

Pour prendre en charge les problèmes posés par les *Big Data*, nous proposons dans cet article un processus ETL, baptisé *PF-ETL (Parallel Functionality-ETL)*, décrit par un ensemble de fonctionnalités où chacune peut s'exécuter en plusieurs instances parallèles (plusieurs exécutions simultanées de la même fonctionnalité) afin de tirer profit des nouveaux environnements informatiques. Notre processus *PF-ETL* s'inscrit dans un objectif d'amélioration de performances (Tableau 1). Parmi les caractéristiques des *Big Data*, nous nous sommes intéressés en particulier au volume et à la vélocité des données. La variété en termes de format n'est pas traitée dans ce travail. Pour intégrer les *Big Data* dans un *DW*, nous proposons une nouvelle architecture décisionnelle dédiée dans laquelle le processus ETL parallèle est décrit. Pour déployer notre approche, nous allons nous fixer sur la fonctionnalité *CDC*. Pour plus de clarté, nous présentons la fonctionnalité *CDC* à la fois dans un environnement classique et dans un environnement *Big Data*.

### *5.1. Fonctionnalité ETL vs Tâche ETL*

Il faut noter qu'il y a une différence entre une fonctionnalité et une tâche d'ETL. La fonctionnalité d'ETL est une fonction de base qui prend en charge un bout de traitement dans l'ETL telles que *Changing Data Capture (CDC)*, *Data Quality Validation (DQV)*, *Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP)*. La tâche d'ETL, quant à elle, est une instance d'une fonctionnalité d'ETL. Par exemple, soit un processus ETL contenant deux tâches *SK1* et *SK2* qui permettent de générer respectivement une clé de substitution pour les tuples à insérer dans les tables de dimension *PRODUIT* et *CLIENT* d'un *DW*. *SK1* et *SK2* sont deux tâches différentes mais toutes les deux se basent sur *SK*. Dans ce qui suit, nous décrivons un processus ETL en fonction de ses fonctionnalités.

### *5.2. Principe de PF-ETL*

Le processus *PF-ETL* que nous proposons est orienté fonctionnalités et est basé sur le paradigme *MR*. Pour chacune de ses fonctionnalités, nous appliquons le même principe que celui adopté, à un niveau processus, par l'approche « ETL basé sur le modèle *MR* » de la figure 3. Comme le montre la figure 4, le processus ETL s'exécute en une seule instance alors que chacune de ses fonctionnalités s'exécute en plusieurs instances.

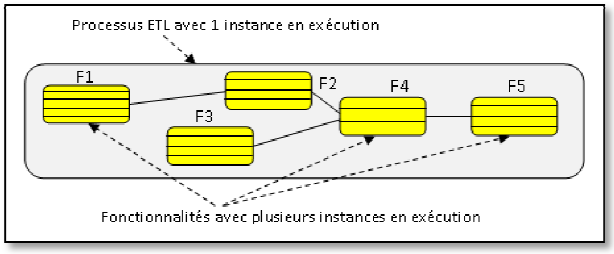


*Figure 4. PF-ETL (Parallel Functionality-ETL)*

Par exemple, la fonctionnalité F4 (forme ovale) qui s'exécute en trois instances (fragments séparés par des traits), a reçu des données en entrée à partir de F2 et F3. Ces inputs sont partitionnés et chacune des trois partitions est soumise à une instance de F4 (mapper). Les résultats partiels fournis par les trois mappers sont fusionnés par des reducers pour constituer le résultat de F4. *PF-ETL* n'est pas suffisant pour obtenir de bonnes performances à un niveau processus si celui-ci présente beaucoup de fonctionnalités séquentielles. Le concepteur pourra, après avoir constitué un processus selon *PF-ETL*, le paramétrer pour l'exécuter, lui aussi, en plusieurs instances. Il s'agit d'une approche hybride qui adopte, en même temps, les principes de l'approche « ETL basé sur le modèle *MR* » et ceux de *PF-ETL* comme le montre la figure 5. Il faut noter que l'approche hybride exige plus de

ressources. Lorsque le processus ETL s'exécute selon l'approche *PF-ETL* et atteint *F4*, il nécessitera dix tâches si *F4* s'exécute en dix instances. Le même processus exécuté selon l'approche hybride, nécessitera cent tâches parallèles si, en plus des dix instances de *F4*, lui-même s'exécute en dix instances.

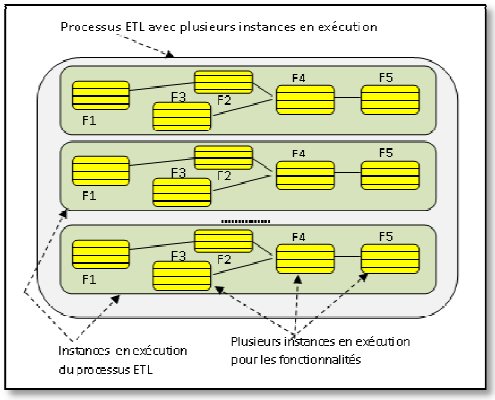


*Figure 5 : PF-ETL Hybride*

### *5.3. Changing Data Capture (CDC) dans un environnement classique*

La plupart des travaux qui existent dans la littérature (cf. Liu et *al.*, 2011), performent uniquement la phase (T) de l'ETL. Or, la phase (E) est problématique lorsqu'il s'agit de données massives. La fonctionnalité *CDC* consiste à identifier, dans les systèmes sources, les données affectées par des changements *(INSERT, UPDATE, DELETE*) pour être capturées et traitées dans le rafraichissement du *DW* (cf. Kimball et Caserta, 2004). La technique la plus utilisée est celle basée sur les *snapshots*. Comme le montre la figure 6, la table source notée *TS* contient des données dans leur version récente (*snapshot j*), la table du *DSA* notée *TSvp* contient les mêmes données dans leur version précédente (*snapshot j-1*). Le processus montre deux étapes (1) le chargement complet de *TS* dans le *DSA* et (2) la comparaison entre *TS* et *TSvp.* Les tuples *01, 03 et 23* seront rejetés par *CDC* puisqu'ils ne présentent aucun changement depuis le chargement précédent (*TSvp*). Par contre, *02* et *22* ont connu des mises à jour et seront capturés comme modification (*UPDATE*). Les tuples *25, 26* et *27* seront capturés comme nouveaux (*INSERT*) puisqu'ils n'apparaissent pas dans *TSvp*. Le tuple *24*, quant à lui, est considéré comme suppression (*DELETE*) puisqu'il apparait dans *TSvp* mais n'apparait plus dans *TS*.

L'algorithme 1, décrit la fonctionnalité *CDC* dans un schéma classique. La ligne 5 charge *TS* dans *DSA*. Les lignes *6* et *7* trient *TS* et *TSvp* selon la clef *#A1* afin de détecter les insertions *(TSvp.A1>TS.A1),* suppressions *(TSvp.A1<TS.A1)* et modifications *(TSvp.A1=TS.A1* avec une différence dans au moins un attribut*).* Les autres cas seront rejetés *(*copie similaire du tuple dans *TS* et *TSvp*) puisqu'aucun changement n'a eu lieu depuis le dernier rafraichissement. Pour détecter les cas de

modifications, nous appliquons une fonction de hachage connue sous le nom de *CRC* (*Cyclic Redundancy Check*) (Freivald et *al*., 1999) sur les tuples de TS et de TSvp ayant la même valeur de la clef.

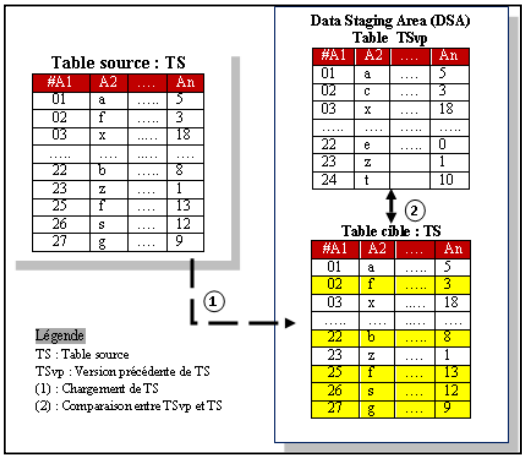


*Figure 6 : Principe de la fonctionnalité CDC basée sur des snapshots*

*Algorithme 1. Fonctionnalité CDC*

---

1: ***CDC_SNAPSHOTS***
2: **entrées**: TS, TSvp
3: **sortie**: DSA
4: **variables** VT1 : enregistrement de type TSvp . VT2 : enregistrement de type TS
5: TS (DSA) ← TS (système source)
6: trier TSvp sur A1 par ordre croissant
7: trier TS sur A1 par ordre croissant
8: **tantque non fin** (TS) **et non fin** (TSvp) **faire**
9: **lire** (TSvp, TV1)
10: **lire** (TS, TV2)
11 : **si** VT1.A1= VT2.A1
12: **si** CRC(VT1) # CRC(VT2) **alors**
13: capturer le tuple VT2 comme modification (*update*)
14: **fin condition ;**
15: **sinon**
16: **si** VT1.A1 < VT2.A1 **alors**
17: capturer le tuple VT1 comme suppression (*delete*)
18: **sinon**
19: capturer le tuple VT2 comme insertion (*insert*)
20: **fin condition**
21: **fin condition**
22: **fin boucle**
23: **tant que non fin** (TS) **faire**
24: capturer le tuple VT2 comme insertion (*insert*)

25: **lire** (TS, VT2)
26: **fin boucle**
27: **tant que non fin** (TSvp) **faire**
28: capturer le tuple VT1 comme suppression (*delete*)
29: **lire**(TSvp, VT1)
30: **fin boucle**
31: **sortie**

### *5.4. Changing Data Capture (CDC) dans l'approche PF-ETL*

Pour proposer un schéma de *CDC* dans un environnement *Big Data*, nous considérons que *TS* et *TSvp* sont massives, l'exécution de *CDC* se fera sur un *cluster* et nous adoptons le modèle *MR*. Le schéma classique de *CDC* sera complété par de nouveaux aspects à savoir (1) partitionnement des données, (2) utilisation de tables d'index (*Lookup)*, (3) processus de capture d'insertions et de modifications et (4) processus de capture de suppressions.

#### *5.4.1. Partitionnement des données.*

Nous adoptons la règle « *diviser pour régner* » qui consiste, dans ce contexte, à partitionner *TS* et *TSvp* avec une volumétrie excessive de manière à obtenir des volumes habituels de données. *TS* sera chargée dans le *DSA* puis partitionnée pour permettre un traitement parallèle des partitions générées. *TSvp* sera partitionnée pour éviter de rechercher les tuples de *TS* dans un grand volume de données.

#### *5.4.2. Tables Lookup*

Pour éviter de rechercher un tuple dans toutes les partitions de *TSvp* et de *TS*, nous utilisons des tables *Lookup* notées respectivement *LookupTSvp* et *LookupTS*. Elles permettent d'identifier, pour un tuple donné, la partition qui pourra le contenir. Voici quelques détails sur l'utilisation des tables *Lookup* :

- *LookupTSvp* et *LookupTS* contiennent, respectivement, les valeurs min et max des clefs naturelles (*#NK*) de chaque partition de *TSvp* et de *TS*;
- Pour un tuple $T_i$ de *TS*, il s'agit de rechercher, dans *LookupTSvp,* la partition $Ptsvp_k$ de *TSvp* vérifiant l'expression :

$$LookupTSvp.NKmin \leq Ti.NK \leq LookupTSvp.NKmax \qquad (1)$$

- Il est possible qu'un tuple $T_i$ de *TS,* vérifiant bien l'expression (1), n'existe pas dans $Ptsvp_k$ ; il s'agit, dans ce cas, d'un nouveau inséré dans *TS* ;
- Pour un tuple $T_j$ de *TSvp*, il s'agit de rechercher, dans *LookupTS,* la partition $Pts_k$ de *TS* vérifiant l'expression :

$$LookupTS.NKmin \leq Tj.NK \leq LookupTS.NKmax \qquad (2)$$

– Il est possible qu'un tuple $T_j$ de *TSvp,* vérifiant bien l'expression (2), n'existe pas dans $Pts_k$ ; il s'agit, dans ce cas, d'un tuple supprimé dans *TS*.

### *5.4.3. Processus IUDCP et DDCP*

Nous proposons deux processus parallèles dans le nouveau schéma de *CDC* qui prennent en charge (1) la capture des insertions et modifications (*IUDCP*) et (2) la capture des suppressions (*DDCP*). Chacun s'exécute en plusieurs instances parallèles. Chaque instance d'*IUDCP* et de *DDCP* prennent en charge respectivement une partition de *TS* et une partition de *TSvp*. Si *TS* et *TSvp* sont partitionnées chacune, en 10 partitions, *CDC* s'exécutera en 20 tâches parallèles.

#### 5.4.3.1. *IUDC Process* : Traitement parallèle des partitions *TS*

La figure 7 décrit *IUDCP*. Chaque partition $Pts_i$ est confiée à $Map_i$ chargé de vérifier, pour chacun de ses tuples, l'existence de celui-ci dans *TSvp*. Pour ce faire, le *mapper* passe par *LookupTSvp* pour identifier la partition $Ptsvp_k$ qui pourra le contenir (*#NK*). Dès que la partition $Ptsvp_k$ est identifiée, trois cas peuvent se présenter : (1) *#NK* inexistante dans $Ptsvp_k$ ; il s'agit d'une insertion (*INSERT*), (2) *#NK* existe avec une copie similaire du tuple dans $Ptsvp_k$ ; le tuple est rejeté (3) *#NK* existe dans $Ptsvp_k$ avec un changement ; il s'agit d'une modification *(UPDATE).*

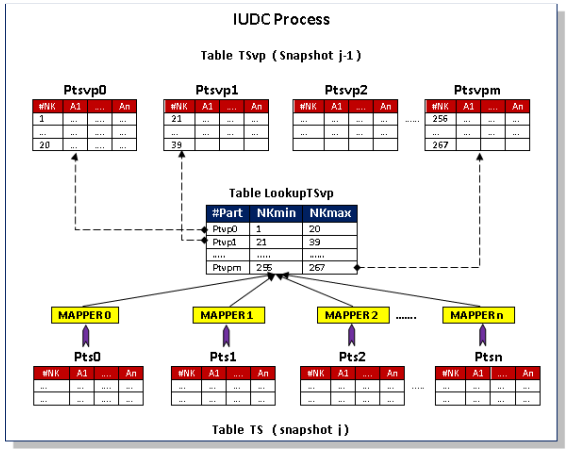


*Figure 7 : Principe de fonctionnement du processus IUDCP*

#### 5.4.3.2. *DDC Process* : Traitement parallèle des partitions de *TSvp*

Dans *IUDCP,* deux *mappers* traitant deux partitions différentes $Pts_i$ et $Pts_j$ peuvent être orientés par *LookupTSvp* vers une même partition $Ptsvp_k$. Ainsi, un

même tuple de celle-ci pourra alors être capturé plusieurs fois comme suppression. C'est la raison pour laquelle, nous avons proposé un autre processus appelé *DDCP*.

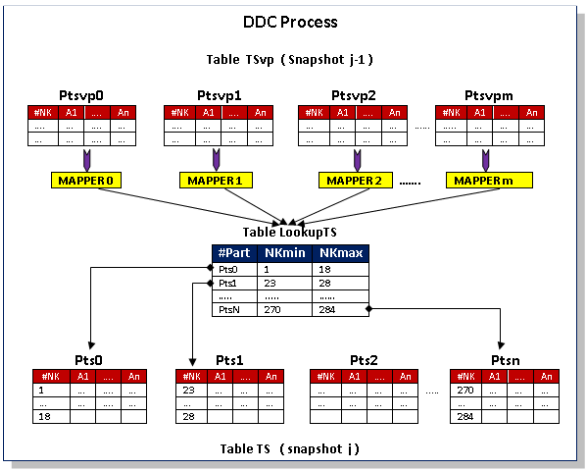


*Figure 8 : Principe de fonctionnement du processus DDCP*

Comme le montre la figure 8, chaque partition $Ptsvp_i$ est confiée à $Map_i$ chargé de vérifier, pour chacun de ses tuples, l'existence de celui-ci dans *TS*. Pour ce faire, le mapper passe par *LookupTS* pour identifier $Pts_k$ de *TS* qui pourra contenir le tuple et dès que celle-ci est identifiée, nous retenons le cas où le tuple n'existe pas *(DELETE)*.

### *5.4.4. Algorithmes*

Le programme principal *CDC_BigData* consiste à partitionner *TS* et *TSvp*, générer *LookupTS* et *LookupTSvp* et enfin lancer, en parallèle, *IUDCP et DDCP*. L'algorithme 2 montre comment *IUDCP* pousse les partitions de *TS* vers le processus *MR* qui s'exécutera en un nombre d'instances égal au nombre de partitions de *TS*. L'algorithme décrivant *DDCP* fonctionne avec le même principe.

*Algorithme 2. Processus IUDCP*

```
1:  IUDCP
2:  entrées: LookupTS
3:  variables VT: enregistrement de type LookupTS ;  i: integer ;  i←1
4:  ouvrir (LookupTS)
5:  tant que non fin (LookupTS) faire
6:        lire (LookupTS, VT)
7:        soumettre VT.Part à iu_mapreader(i); i←i+1
8:  fin boucle
9:  iu_map()     // appel de la fonction iu_map()
```

L'algorithme 3 est chargé de capturer les insertions et les modifications effectuées dans *TS*. Une partition $Pts_i$ sera traitée par une instance de *iu_map()*. Les lignes 6-12 recherchent $Ptsvp_K$ qui pourra contenir le tuple lu dans la ligne 7. Les lignes 13-18 traitent le cas où $Ptsvp_K$ est localisée. Celle-ci est parcourue jusqu'à détection du tuple. Les lignes 19-22 capturent les modifications en identifiant les tuples semblables (même valeur de la clef*)* de *TS* et *TSvp* et vérifient les éventuels changements avec la fonction CRC (ligne 20) ayant affecté le tuple, auquel cas le tuple est capturé comme modification (*UPDATE*). Les lignes 23-24 traitent le cas où le tuple n'existe pas dans la partition, celui-ci est capturé comme insertion (*INSERT*). Les lignes 26-27 capturent comme insertion (*INSERT*) les tuples où *LookupTSvp* montre qu'ils n'apparaissent dans aucune des partitions de *TSvp.*

*Algorithme 3. Tâche de capture d'insertions et modification de données*

```
1:  IU_MAP(Pts)
2:  entrées: Ptsvp, LookupTS
3:  sortie: DSA
4:  VT1 : enreg. de type TS, VT2 : enreg. de type LookupTSvp ; VT3 : enreg. de type  Ptsvp
5:  ouvrir (Pts)
6:  tantque non fin (Pts) faire
7:      lire (Pts, VT1)
8:      ouvrir (LookupTSvp)
9:      lire (LookupTSvp, VT2)
10:     tant que non fin (LookupTSvp)  et VT1.NK > VT2.NKmax faire
11:             lire (LookupTSvp, VT2)
12:     fin boucle
13:     si non fin (LookupTSvp) alors
14:             ouvrir (VT2.Part)
15:             lire (VT2.Part, VT3)
16:             tant que non fin (VT2.Part) et VT1.NK > VT3.NK
17:                     lire (VT2.Part, VT3)
18:             fin boucle
19:             si VT1.NK=VT3.NK alors
20:                     si CRC(VT1) # CRC(VT3) alors
21:                             capturer le tuple VT1 comme modification (update)
22:                     fin condition
23:             sinon
24:                     capturer le tuple VT1 comme insertion (insert)
25:             fin condition
26:     sinon
27:             capturer le tuple VT1 comme insertion (insert)
28:     fin condition
29: fin boucle
30: sortie
```

L'algorithme 4 est chargé de capturer les suppressions. Une partition $Ptsvp_i$ sera traitée par une instance de *d_map()*. Les lignes 6-12 localisent la partition $Pts_i$ qui pourra contenir le tuple lu dans la ligne 7. Les lignes 13-18 traitent le cas où la

partition $Ptsvp_k$ est localisée. Si celle-ci ne contient pas le tuple (ligne 19), il sera capturé comme suppression (ligne 20). Les lignes 22-24 capturent comme suppression les tuples où *LookupTS* montre qu'ils n'apparaissent dans aucune partition de *TS*.

*Algorithme 4. Tâches de capture de suppression de données*

```
1:  D_MAP(Pts)
2:  entrées: Pts, LookupTSvp
3:  sortie: DSA
4:  VT1 : enreg de type TSvp ; VT2 : enreg de type LookupTS ; VT3 : enregide type Pts
5:  ouvrir (Ptsvp)
6:  tantque non fin (Ptsvp) faire
7:      lire (Ptsvp, VT1)
8:      ouvrir (LookupTS)
9:      lire (LookupTS, VT2)
10:     tant que non fin (LookupTS) et VT1.NK > VT2.NKmax faire
11:         lire (LookupTS, VT2)
12:     fin boucle
13:     si non fin (LookupTS) alors
14:         ouvrir (VT2.Part)
15:         lire (VT2.Part, VT3)
16:         tant que non fin (VT2.Part) et VT1.NK > VT3.NK
17:             lire (VT2.Part, VT3)
18:         fin boucle
19:         si VT1.NK < VT3.NK alors
20:             capturer le tuple VT1 comme suppression (delete)
21:         fin condition
22:     sinon
23:         capturer le tuple VT1 comme suppression (delete)
24:     fin condition
25: fin boucle
26: sortie
```

## 6. Implémentation et expérimentations

Notre prototype *PF-ETL* a été développé avec *java* 1.7 (*java* ™ *SE Runtime Environment*) sous *Netbeans IDE* 7.4. Nous avons déployé un *cluster* avec dix nœuds dont chacun possède un processeur *Intel Core* i5-2500 *CPU* @ 3.30 *GHz* x 4, *OS type* : 64-bit, *Ubuntu* 12.10 et le framework *Hadoop* 1.2.0.

Pour valider notre approche *PF-ETL*, nous avons défini deux scénarios d'expérimentations. Notre objectif est de comparer *PF-ETL* basé sur les fonctionnalités (niveau de granularité fin) avec un ETL parallèle à un niveau processus (niveau de granularité élevé). Nous présentons dans ce qui suit les résultats obtenus selon le deuxième scénario. Les tests selon le premier scénario étant en cours d'élaboration. Nous espérons avoir les premiers résultats très prochainement afin de procéder à une étude comparative entre les deux scénarios et

montré que la décomposition d'un processus ETL en un ensemble de fonctionnalités permet le passage à l'échelle plus facilement notamment dans le cadre des *Big Data*. Nous avons mené des expérimentations qui nous permettent de mesurer les performances d'un processus ETL constitué de huit fonctionnalités (1) *Extraction* des données (2) *partitionnement* et *distribution* des données sur le *cluster* (3) *projection* (4) *restriction* avec *NOT NULL* (5) *extraction* de l'année avec *YEAR()* (6) *agrégation* avec *COUNT()* (7) *fusion* des données (8) *chargement* dans un *DW*. Les temps d'exécution sont évalués en faisant varier la taille des données sources et le nombre de tâches *MapReduce* s'exécutant en parallèle. Le tableau 2 montre que la distribution des données et des traitements sur un nombre de tâches MapReduce plus important donne plus de capacité à l'ETL pour faire face aux données massives et améliore les temps d'exécution.

*Tableau 2. Temps d'exécution (mn) de l'ETL à un niveau processus*

| | Nombre de tâches | | | |
|---|---|---|---|---|
| Taille (GO) | 5 | 7 | 8 | 11 |
| 30 | 39,37 | 34,2 | 30,6 | 26,15 |
| 50 | 53 | 51 | 48,5 | 45,2 |
| 80 | 79 | 73,4 | 68 | 62,8 |
| 100 | 102 | 93,7 | 85,4 | 80,3 |

## 7. Conclusion

Les SID se trouvent aujourd'hui face à un défi majeur caractérisé par les *Big Data* et doivent donc s'adapter aux nouveaux paradigmes tels que le *cloud computing* et le modèle *MR*. Le processus ETL étant le cœur d'un SID puisque toutes les données destinées à l'analyse y transitent. Il faudra étudier de manière profonde l'impact de ces nouvelles technologies sur l'architecture, la modélisation et les performances de l'ETL. Dans ce contexte, nous avons proposé une approche parallèle pour les processus ETL dont les fonctionnalités s'exécutent selon le modèle *MR*. Dans un futur proche, nous envisageons de mener des tests à plus grande échelle. Par ailleurs, le *cloud computing* est un environnement qui dispose de toutes les ressources en termes d'infrastructures, de plateformes, en particulier des plateformes *MR* et d'applications pour offrir des services de qualité avec des coûts raisonnables. Notre approche *PF-ETL* dispose des qualités techniques en vue d'une migration vers un environnement *cloud*. Dans cette perspective, nos travaux futurs devront prendre en considération des aspects liés à la virtualisation et à l'architecture *SOA*.

## Références

Arbab F. (2004). Reo: A Channel-based Coordination Model for Component Composition, *Mathematical Structures in Computer Science archive, Volume 14, Issue 3, p. 329–366.*

Bala M. et Alimazighi Z. (2013). Modélisation de processus ETL dans un modèle MapReduce, *Conférence Maghrébine sur les Avancées des Systèmes Décisionnels (ASD'13), p. 1–12,* Marrakech, Maroc.

Dean J. et Ghemawat S. (2004). MapReduce: Simplified Data Processing on Large Clusters, *6th Symposium on Operating Systems Design & Implementation (OSDI '04), p. 137–150,* San Francisco, CA, USA.

El Akkaoui Z. et Zemányi E. (2009). Defining ETLWorfklows using BPMN and BPEL, *DOLAP'09, p. 41–48*, Hong Kong, China.

El Akkaoui Z., Zimányi E., et Mazón J. N., Trujillo J. (2011). A Model-Driven Framework for ETL Process Development, *DOLAP'11, p. 45–52,* Glasgow, Scotland, UK.

Freivald, M. P., Noble A. C. et Richards M. S. (1999). Change-detection tool indicating degree and location of change of internet documents by comparison of cyclic-redundancy-check (crc) signatures. *US Patent 5,898,836.*

Ghemawat S. et Dean J. (2008). MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, *Volume 51, Issue 1, p. 107-113,* New York, USA.

Han, J., E. Haihong, G. Le, et J. Du (2011). Survey on NoSQL database. *6th International Conference on Pervasive Computing and Applications (ICPCA'11), pp. 363–366*. Port Elizabeth, South Africa.

Kimball R. et Caserta J. (2004). *The Data Warehouse ETL Toolkit, p. 105–254,* Wiley Publishing, Inc., Indianapolis, USA.

Liu X., Thomsen C., et Pedersen T. B. (2011). ETLMR : A Highly Scalable Dimensional ETL Framework based on Mapreduce, *in proceedings of 13th International Conference on Data Warehousing and Knowledge, p. 96–111*, Toulouse, France.

Misra S., Saha S.K., et Mazumdar C. (2013). Performance Comparison of Hadoop Based Tools with Commercial ETL Tools – A Case Study, *Big Data Analytics (BDA'13), LNCS 8302, p. 176–184, 2013,* Mysore, India.

Mohanty S., Jagadeesh M., et Srivatsa H. (2013). *Big Data Imperatives*, *p. 1–22,* Apress, NY, USA.

Oliveira B. et Belo O. (2013). Using Reo on ETL Conceptual Modelling A First Approach, *DOLAP'13, p. 55–60*, San Francisco, CA, USA.

Sosinsky B. (2011). *Cloud Computing Bible*, *p. 45–88,* Wiley Publishing, Inc, Indiana, USA.

Thomsen C. et Pedersen T. (2009a). Pygrametl: A Powerful Programming Framework for Extract-Transform-Load Programmers. *In Proc. of DOLAP, p. 49–56,* Hong Kong, China.

Thomsen C. et Pedersen T. (2009b). Building aWebWarehouse for Accessibility Data, *In Proc. of DOLAP'09, p. 43–50*, Hong Kong, China.

Trujillo J. et Luján-Mora S. (2003). A UML Based Approach for Modeling ETL Processes in Data Warehouses*, ER 2003, LNCS 2813, p. 307–320, Springer-Verlag* Berlin Heidelberg.

Vassiliadis P., Simitsis A., et Skiadopoulos S. (2002). Conceptual Modeling for ETL Processes, *DOLAP'02, p. 14–21,* McLean, Virginia, USA.