Une Plateforme ETL parallèle et distribuée pour l'intégration de données massives

Mahfoud Bala*, Oussama Mokeddem*,
Omar Boussaid**, Zaia Alimazighi***

*LRDSI, Université Saad Dahleb, Blida 1, Algérie
{mahfoud.bala, mokeddem.dev}@gmail.com,
**ERIC, Université Lumière, Lyon 2, France
omar.boussaid@univ-lyon2.fr,
***LSI, USTHB, Alger, Algérie
zalimazighi@usthb.dz

Résumé. Nous nous intéressons, dans ce papier, à l'impact des données massives dans un environnement décisionnel et plus particulièrement sur la phase d'intégration des données. Dans ce contexte, nous avons développé une plateforme, baptisée *P-ETL (Parallel-ETL)*, destinée à l'entreposage de données massives selon le paradigme *MapReduce*. *P-ETL* permet le paramétrage de processus *ETL* (workflow) et un paramétrage avancé relatif à l'environnement parallèle et distribué. Ce papier décrit la plateforme *P-ETL* en vue d'une démonstration. Face à des jeux de données allant de $244 * 10^6$ à $7,317 * 10^9$ *tuples*, les expérimentations menées ont montré l'amélioration significative des performances de *P-ETL* lorsque la taille du *cluster* et le nombre des tâches parallèles augmentent.

1 Introduction

Les données massives, appelées communément *"big data"*, impactent directement le processus *ETL* (*Extracting-Transforming-Loading*) vu que celui-ci est le premier composant du système décisionnel confronté à ces données. Peu de travaux ont traité sur la problématique des données massives dans le processus *ETL*. Liu et al. (2011) ont proposé une approche parallèle/distribuée appelée *ETLMR* consistant à améliorer les performances de la phase de transformation (T) et de chargement (L) de l'*ETL* et ce en adoptant, pour chacune des deux phases, des stratégies de distribution appropriées. Les expérimentations de Misra et al. (2013) ont montré que le paradigme *MapReduce* est prometteur et que les solutions *ETL* basées sur des *frameworks open source* tel que *Apache Hadoop* sont plus performantes et moins couteuses par rapport aux solutions *ETL* commercialisées. Contrairement aux travaux de Liu et al. (2011), ceux de Misra et al. (2013) considèrent la phase d'extraction (E) de l'*ETL* très couteuse ; celle-ci a été traitée dans un environnement parallèle/distribué selon le paradigme *MapReduce*. (Liu et al., 2012) est une démonstration du prototype *ETLMR*. Dans (Liu et al., 2014), les auteurs proposent une plateforme *CloudETL* basée sur *Apache Hadoop* et *Apache Hive* où les performances ont été nettement améliorées par rapport à celles d'*ETLMR* (Liu et al.,

2011). Les plateformes *ETLMR* (Liu et al., 2011) et *CloudETL* (Liu et al., 2014) sont basées sur du code Python, et par conséquent celles-ci sont destinées aux informaticiens développeurs de solutions *ETL* parallèles/distribuées. Dans le but de vulgariser ce type de plateformes et les rendre accessibles aux utilisateurs finaux, nous proposons, dans ce papier, une plateforme baptisée *P-ETL (Parallel-ETL)* développée sous l'environnement *Apache Hadoop* [1]. Le paramétrage d'un processus se fait, de bout-en-bout, sur une interface unique structurée en trois onglets, chacun concerne une phase du processus (E, T, L). Le même paramétrage peut s'effectuer dans un fichier XML pour un traitement en batch. Nous avons adapté le schéma classique de l'*ETL* dans l'environnement *MapReduce*. Ainsi, *P-ETL* procède en cinq phases : *(E)xtracting, (P)artitioning, (T)ransforming, (R)educing et (L)oading* ; au lieu d'*ETL*, on parle plutôt d'*EPTRL*.

2 Les bases de P-ETL

Nous présentons dans cette section les bases et les principes fondamentaux de *P-ETL* en exposant les techniques de partitionnement supportées, l'adaptation des phases *Map* et *Reduce* aux spécificités de l'*ETL* et nous terminons par l'architecture globale de *P-ETL*.

2.1 Partitionnement des données

Dans le but de distribuer/paralléliser le processus *ETL*, les données sources doivent être elles aussi distribuées pour permettre à plusieurs tâches de s'exécuter de façon parallèle où chacune traite sa propre partition de données. *P-ETL* offre trois types de partitionnement. Afin d'assurer une charge plus ou moins équitable entre les différentes tâches parallèles, le choix du type de partitionnement est important. La présence d'un taux élevé, dans une partition de données, de tuples avec des valeurs creuses implique une charge faible en termes de traitement pour la tâche. En effet, les tuples en question seront rejetés par un filtre tel que *NOT NULL*.

Simple : étant donné un volume de données source *v*, le type de partitionnement *simple* génère des partitions égales selon l'équation 1 où *nb_part* étant le nombre de partitions.

$$taille(partition) = taille(v)/nb\_part \tag{1}$$

Round Robin (RR) : Avec la technique *Round Robin*, l'affectation d'un *tuple* depuis le volume source *v* vers une partition de données *p* est basée sur l'équation 2. *rang (tuple)* étant le rang du *tuple* dans le volume *v* et *nb_part* étant le nombre de partitions.

$$p = rang(tuple) \bmod nb\_part \tag{2}$$

Round Robin par Bloc (RRB) : Cette technique est similaire à *Round Robin*. Dans le but d'accélérer le partitionnement, un bloc de *tuples* est affecté à la partition, plutôt qu'une affectation *tuple* par *tuple*.

1. http ://hadoop.apache.org/

2.2 Les mappers et les reducers

Dans la plateforme *P-ETL*, les primitives *map()* et *reduce()* ont été adaptées aux spécificités de l'*ETL*. Le rôle assigné à un *mapper* est la normalisation des données (nettoyage, filtrage, conversion, ...). Le *mapper* traite chaque *row* dans un tunnel de transformations (T_1, T_2 ...) où chaque T_i est chargé d'une opération particulière telles que le nettoyage, filtrage, projection, conversion et concaténation. La figure 1 montre un tunnel de transformation constitué de trois fonctions : (i) une projection ($\prod$), (ii) un filtre *NOT NULL* ($\otimes$) et (iii) une fonction d'extraction de l'année *YEAR()*. Un tuple dans un format clé/valeur r_0 est transformé en r_3 après passage par le tunnel.

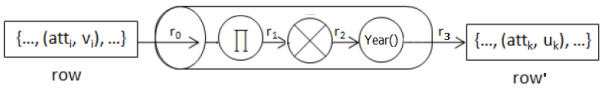

FIG. 1 – *mapper : un tunnel de transformation.*

Le *reducer*, quant à lui, est chargé de la fusion et de l'agrégation des données préparées (*UNION, JOIN, COUNT, SUM, MAX, ...*). La figure 2 montre les résultats partiels obtenus par les *Mappers* triés selon leur destination (*Reducer*). Au niveau d'un $Reducer_i$, la primitive *Shuffle()* consiste à capturer tous les tuples à destination de celui-ci et à les stocker dans sa zone locale. Ils seront, ensuite, agrégés et chargés dans un entrepôt de données.

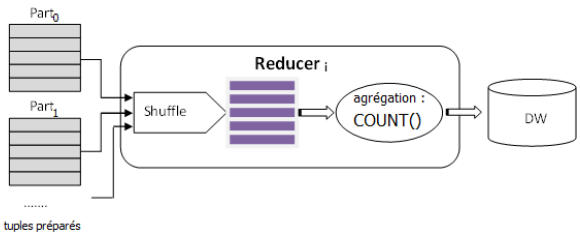

FIG. 2 – *Architecture du reducer*

2.3 Architecture de P-ETL

La plateforme *P-ETL* est organisée en cinq modules : *(E)xtracting*, *(P)artitioning*, *(T)ransforming*, *(R)educing* et *(L)oading* (Figure 3). Après extraction (E), les données sources sont chargées dans le système de fichier distribué de *Hadoop* (*HDFS*). Ensuite, un partitionnement logique des données (P) s'effectue selon le choix de l'utilisateur final (*Simple, RR, RRB*). Les partitions des données ainsi générées seront soumises au processus *MapReduce*. Chaque *mapper* est en charge de transformer (T) les données de sa partition (nettoyage, filtrage, conversion, ...).

Les fonctions de fusion et d'agrégation des données sont différées pour être exécutées dans la phase *Reduce* (R). A ce niveau, les données deviennent pertinentes et peuvent alors être chargées (L) dans l'entrepôt de données.

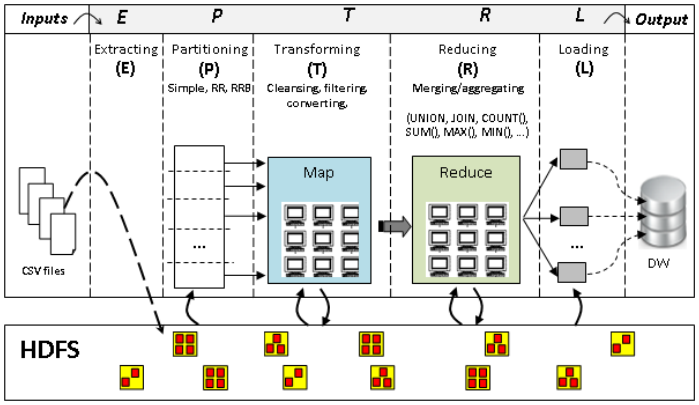

FIG. 3 – *Architecture de P-ETL.*

3 Paramétrage d'un processus dans P-ETL

Le paramétrage d'un processus se fait sur une interface unique organisée en trois onglets (*Extract, Transform, Load*) dédiés au processus lui-même (*workflow*), plus une partie "*Advanced Parameters*" réservée pour la configuration de l'environnement parallèle/distribué de *Apache Hadoop* (Figure 4). Pour la configuration du processus, l'utilisateur doit commencer par l'onglet "*Extract*". Les paramètres disponibles sur les deux autres onglets "*Transform*" et "*Load*" dépendent du premier, principalement du format de la source, sa structure et son emplacement. En revanche, la partie "*Advanced Parameters*" peut être utilisée indépendamment des trois onglets.

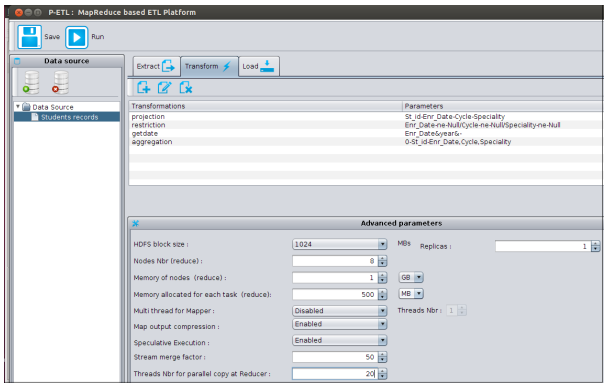

FIG. 4 – *Interface de configuration P-ETL.*

Dans l'onglet "*Extract*", l'utilisateur doit, en premier lieu, localiser les données sources. Le format pivot des données est le fichier *csv* qui est adopté par toutes les plateformes *MapReduce* vu sa légèreté (absence de méta-données). Afin d'accélérer le chargement des données sources dans le système *HDFS*, l'utilisateur pourra activer la compression de celles-ci. Ensuite,

l'utilisateur doit choisir un type de partitionnement des données sources (*simple, Round Robin, Round Robin by block*) ainsi que le nombre de partitions (égal au nombre de *mappers*). Enfin, il doit paramétrer le mode de lecture du *mapreader* à partir des partitions de données (*Ligne par ligne, nombre de lignes, taille en KO*). La figure 4 montre l'onglet "*Transform*" qui fournit à l'utilisateur une liste de fonctions de transformation et d'agrégation. Chaque fonction insérée doit être paramétrée (*entrées, conditions, expressions, ...*). Le tunnel de transformation ainsi constitué s'exécutera dans l'ordre d'insertion des fonctions. Enfin, l'onglet "*Load*" permet la configuration du chargement des données préparées dans l'entrepôt de données et comporte la destination des données (*entrepôt de données, magasin de données, cube de données*), la compression des données avant leur chargement dans le système *HDFS* ainsi que le caractère séparateur du fichier *csv* cible. Concernant le paramétrage de l'environnement parallèle/distribué dans *Apache Hadoop* (taille d'un bloc *HDFS*, nombre de noeuds impliqué dans le processus, taille mémoire réservée à un noeud et à une tâche, compression des résultats des *mappers* avant de les soumettre aux *reducers*, ...), celui-ci se fait dans la partie "*Advanced Parameters*".

4 Expérimentation

Pour évaluer *P-ETL*, nous avons installé un *cluster* de 19 machines, chacune possède un processeur intel-Core TMi3-3220 CPU@3.30 GHZ x 4 processor, 4GO RAM et 500 GO d'espace disque. Le réseau local (*LAN*) est un Ethernet 100 Mbps. Selon la configuration matérielle présentée ci-dessus, l'environnement *Hadoop* permet d'affecter, au maximum, deux tâches parallèles à un même noeud. Ainsi, deux tâches parallèles sont équivalentes à un noeud dans ce qui suit.

Données de test : Nous avons développé un programme qui génère des données synthétiques relatives aux renseignements des étudiants. Les expérimentations ont été réalisées sur des jeux de données allant de $244 * 10^6$ à $7,317 * 10^9$ *tuples*. Nous exposons, dans ce qui suit, l'expérimentation réalisée sur un fichier *etudiant.csv* de $7,317 * 10^9$ lignes où chacune a une taille de 44 octets. Ce fichier contient les attributs suivants : *Matricule , Date d'inscription, Cycle (Licence, Master, Doctorat), Spécialité, Bourse et Sport*. Le processus *ETL* paramétré pour l'expérimentation se présente en quatre fonctions. La première tâche est la *projection* qui consiste à exclure les attributs *Bourse* et *Sport*. La deuxième tâche du processus est une *restriction* qui filtre les *tuples* et rejette tous ceux présentant une valeur *Null* dans l'un des attributs : *Date d'inscription*, *Cycle*, et *Spécialité*. La troisième tâche est *Year()* qui extrait l'année à partir de la *date d'inscription*. Enfin, la quatrième tâche est une fonction d'agrégation *COUNT()* qui compte le nombre d'étudiants inscrits durant la même année, dans le même cycle et la même spécialité. Il est à noter que durant l'exécution du processus, lorsque *P-ETL* rencontre une agrégation, comme *COUNT()* dans ce cas, celle-ci sera différée pour être exécutée dans la phase *Reduce*.

Résultats : Comme le montre le tableau 1, l'augmentation des tâches parallèles améliore de manière significative les performances du processus. Nous remarquons que le gain de temps entre 24 et 30 tâches est très intéressant (50 mn). En revanche, nous constatons une régression

du gain entre 30 et 38 tâches (6 mn). Nous pourrons conclure qu'au delà d'un certain seuil en termes de tâches parallèles, l'amélioration des performances des processus sous *P-ETL* ne devient plus significative.

Nombre de tâches	24	30	38
Temps de trait. (mn)	143	93	87

TAB. 1 – *Temps de traitement (mn) pour des données de 300 GO*

5 Conclusion

Le processus *ETL* est considéré aujourd'hui comme étant le coeur du système décisionnel puisque toutes les données destinées pour l'analyse transitent par celui-ci. Afin de faire face aux données massives, nous l'avons adapté selon le paradigme *MapReduce* pour permettre son exécution dans un environnement parallèle et distribué. *P-ETL* est basé sur une interface de paramétrage conviviale afin de rendre la plateforme accessible aux utilisateurs finaux. Les résultats des expérimentations montrent une meilleure scalabilité de *P-ETL* face à des volumes de données importants lorsque la taille du cluster augmente.

Références

Liu, X., C. Thomsen, et T. B. Pedersen (2011). Etlmr : a highly scalable dimensional etl framework based on mapreduce. In *Data Warehousing and Knowledge Discovery*, pp. 96–111. Springer.

Liu, X., C. Thomsen, et T. B. Pedersen (2012). Mapreduce-based dimensional etl made easy. *Proceedings of the VLDB Endowment 5*(12), 1882–1885.

Liu, X., C. Thomsen, et T. B. Pedersen (2014). Cloudetl : scalable dimensional etl for hive. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pp. 195–206. ACM.

Misra, S., S. K. Saha, et C. Mazumdar (2013). Performance comparison of hadoop based tools with commercial etl tools–a case study. In *Big Data Analytics*, pp. 176–184. Springer.

Summary

We focus in this paper on the impact of Big Data in a decision-making environment, particularly on the data integration phase. In this context, we developed, under the Apache Hadoop framework, a platform called P-ETL (Parallel-ETL) intented to the large data warehousing (DW) according to the *MapReduce* paradigm. P-ETL allows setting ETL processes (workflow template) and provides an advanced setting that consists of configuring the parallel/distributed processing in Apache Hadoop. This paper demonstrates P-ETL platform. Facing data sets with $244 * 10^6$ until $7,317 * 10^9$ tuples, the conducted experiment shows that increasing the cluster size and the parallel tasks speed-up the P-ETL process.