

Compte Rendu de Travaux Pratiques

Analyse de Données avec Hadoop MapReduce

Djebbar Anis & Al-Mahmoud Al-Ali Mahmoud

11 décembre 2025

1 Introduction

Ce TP avait pour objectif de nous familiariser avec le framework **Hadoop** et le modèle de programmation **MapReduce** pour traiter des données volumineuses. Dans ce modèle, tout traitement est exprimé en paires $\langle \text{clé}, \text{valeur} \rangle$ à travers deux fonctions :

- **Map** : transforme les entrées brutes en paires intermédiaires ;
- **Reduce** : agrège les valeurs associées à une même clé.

L'environnement Hadoop gère la parallélisation, la distribution des données et la tolérance aux pannes. Nous avons implémenté plusieurs opérateurs classiques : filtrage, group-by, agrégation, jointure et quelques requêtes analytiques métier.

2 Partie 1 – Opérateurs de base

2.1 WordCount filtré

La classe `WordCount` lit le fichier `constitutionFrancaise.txt` depuis `input-wordCount/`. Dans le **Mapper**, nous :

- passons le texte en minuscules ;
- supprimons la ponctuation et les caractères non alphanumériques ;
- découpons en mots et **ne gardons que ceux de longueur** > 4 ;
- émettons pour chaque mot valide la paire $\langle \text{mot}, 1 \rangle$.

Dans le **Reducer**, nous additionnons les occurrences par mot, puis nous **filtrons en sortie** en n'écrivant que les mots dont la fréquence est ≥ 10 . Ce job montre comment combiner pré-traitement (nettoyage) et filtrage sur les phases Map et Reduce.

2.2 GroupBy sur les ventes (Superstore)

Les fichiers de ventes `superstore.csv` sont traités dans le répertoire `input-groupBy/`. Nous avons écrit trois programmes :

Ventes par Date et State (GroupBy) Le Mapper :

- ignore l'en-tête "Row ID" ;
- extrait la date de commande (colonne `Order Date`), l'état (`State`) et le montant des ventes (`Sales`) ;
- construit une **clé composite** `date;state` et émet $\langle \text{date;state}, \text{sales} \rangle$.

Le Reducer somme les ventes par clé pour obtenir, pour chaque couple (Date, State), le total des ventes.

Ventes par Date et Category (GroupByCategory) Sur le même principe, nous changeons la clé composite en `date;category` (en utilisant la colonne `Category`). Le Reducer effectue à nouveau une simple somme des montants `Sales`. Ces deux jobs illustrent l'implémentation de **GROUP BY** sur une ou plusieurs colonnes via une clé composite.

Statistiques par commande (GroupByOrder) Ici, nous voulons, par **Order ID** :

- le nombre de produits distincts ;
- la quantité totale d'articles achetés.

Le Mapper émet pour chaque ligne :

$\langle \text{Order ID}, \text{ProductID}; \text{Quantity} \rangle$.

Dans le Reducer, nous parcourons toutes les valeurs d'une commande :

- un **HashSet** permet de compter les produits distincts (**ProductID**) ;
- une somme sur les quantités donne le total d'exemplaires.

La sortie contient, pour chaque commande, **nbProduitsDistincts** et **nbTotalExemplaires**.

2.3 Jointure Clients / Commandes (Join)

L'exercice de jointure utilise deux fichiers : **customers.tbl** et **orders.tbl** (répertoire **input-join/**). L'objectif est de restituer les couples :

$(\text{CUSTOMERS.name}, \text{ORDERS.comment})$.

Nous avons utilisé **MultipleInputs** pour brancher deux Mappers différents :

- **MapCustomers** lit **customers.tbl**, extrait l'**ID client** et le **Nom**, et émet $\langle \text{customerId}, "C|" + \text{name} \rangle$;
- **MapOrders** lit **orders.tbl**, extrait **customerId** et **comment**, et émet $\langle \text{customerId}, "O|" + \text{orderId} + "|" + \text{comment} \rangle$.

Le **Reducer** est chargé de la jointure *reduce-side* :

- il sépare les valeurs reçues en deux listes : **customers** et **orders** ;
- puis réalise un **produit cartésien** via deux boucles imbriquées pour associer chaque nom de client avec tous ses commentaires de commande ;
- pour chaque paire, il écrit $(\text{customerName}, \text{orderComment})$.

Cette approche respecte la consigne du sujet : copier les valeurs de l'itérateur dans des tableaux temporaires puis joindre côté Reduce.

3 Partie 2 – Requêtes analytiques (Part2Analytics)

La classe **Part2Analytics** contient plusieurs jobs MapReduce enchaînés sur le fichier de faits **Faits_Commandes.csv**. Les colonnes importantes sont notamment : **ID_CLIENT**, **ID_PLAT**, **ID_RESTAURANT**, **ID_LIVREUR**, **PRIX_UNITAIRE**, **FRAIS_LIVRAISON**, **QUANTITE_PLAT**, **DISTANCE_LIVRAISON**, **DELAI_LIVRAISON**, **NOTE_COMMANDE**.

Fréquence et valeur client

Le job **MapClientFrequency / ReduceClientFrequency** regroupe par **ID_CLIENT**. Le Mapper calcule le *montant* de chaque commande :

$\text{montant} = \text{prix unitaire} \times \text{quantité} + \text{frais de livraison},$

et émet $\langle \text{idClient}, \text{"montant|note"} \rangle$. Le Reducer en déduit pour chaque client :

- le nombre de commandes ;
- le panier moyen ;
- la valeur totale du client ;
- la satisfaction moyenne ;
- une estimation de fréquence annuelle.

Plats les plus commandés

Le job **MapMostOrderedDishes / ReduceMostOrderedDishes** regroupe par **ID_PLAT**. Il calcule pour chaque plat :

- le nombre de commandes et la quantité totale ;
- le revenu généré ($\text{prix} \times \text{quantité}$) ;
- la note moyenne et une part approximative de commandes (%).

Performance des restaurants

Deux jobs ciblent les restaurants :

- **MapRestaurantDelay** / **ReduceRestaurantDelay** : calcule le délai moyen de livraison et la satisfaction réelle, puis qualifie le respect des délais (*conforme* / *dégradé*) ;
- **MapRestaurantLate** / **ReduceRestaurantLate** : se concentre sur le **taux de retards** (délai > 35 minutes) et attribue un niveau d'alerte (*pas de retard*, *retards mineurs*, *problème sérieux*).

Ponctualité des livreurs

Enfin, le job **MapLivreurPerformance** / **ReduceLivreurPerformance** regroupe par **ID_LIVREUR** et mesure :

- le délai moyen de livraison et la distance moyenne parcourue ;
- le taux de livraisons ponctuelles (délai ≤ 25 min) ;
- le taux de retards critiques (délai > 35 min) ;
- la satisfaction moyenne des clients ;
- une évaluation globale (*Excellent*, *Bon*, *À améliorer*).

4 Conclusion

Ce TP nous a permis de :

- manipuler le modèle MapReduce sur des tâches simples (WordCount) puis plus proches du SQL (GROUP BY, agrégations) ;
- réaliser une jointure *reduce-side* entre deux fichiers avec **MultipleInputs** ;
- traduire des requêtes analytiques métier en jobs MapReduce (valeur client, popularité des plats, délai et ponctualité des restaurants et livreurs).

La difficulté principale a été de bien gérer :

- le format des fichiers (CSV, séparateur, ordre des colonnes) ;
- le filtrage des en-têtes et lignes invalides ;
- l'encodage des informations dans les valeurs (avec "|" ou ";").

Au final, l'exercice montre que l'on peut reproduire une grande partie des requêtes analytiques d'un entrepôt de données uniquement avec des fonctions **map()** et **reduce()**.