École Centrale Lyon

Technologies informatiques du Big Data
TP1
Rapport

GROUPE 2 - TP Hadoop/MRJobs

Librarie MRJobs

Élèves :
Matheus Machado e Silva

Enseignant :
Stéphane DERRODE

6 février 2023

Table des matières

1 Introduction

MRJob et Hadoop sont deux outils puissants pour le traitement de données en parallèle. MRJob est une bibliothèque Python qui permet d'écrire des algorithmes MapReduce en utilisant une syntaxe simple et intuitive. Hadoop, quant à lui, est un système de gestion de données distribuées, qui permet de traiter de gros volumes de données de manière efficace. En utilisant MRJob et Hadoop ensemble, les développeurs peuvent construire des solutions de traitement de données en parallèle pour des cas d'utilisation complexes et à grande échelle.

Dans ce rapport, on observe les avantages de l'utilisation de MRJob et Hadoop pour résoudre des problèmes de traitement de données dans un environnement docker. On étudie également les cas d'utilisation dans un contexte de ventes, ainsi que les meilleures pratiques pour développer des algorithmes MapReduce efficaces.

2 Exercice 1 - Questionner un fichier de ventes

Tout d'abord, on observe que la première ligne du fichier csv contient les noms des champs et qu'elle doit être ignorée dans la ligne de code, pour cela une vérification du premier champ est effectuée.

```
# Sauter la première ligne
if line.startswith('ORDERNUMBER'):
    return
```

Ensuite, étant donné le champ vide existant et séparé par une simple virgule, la commande line.split(',') est utilisée et aucun problème n'apparaît.

```
product_line = line.split(',')[10]
```

Le 10 représente la colonne de productline.

2.1 Quelles sont toutes les catégories de produits (PRODUCTLINE) présentes dans le fichier ?

On ne présentera dans le rapport que le premier code à des fins d'explication, les autres seront disponibles dans le **dossier «code sources »**.

```
1 from mrjob.job import MRJob
2
3 class ProductLines(MRJob):
4     def mapper(self, _, line):
```

```
        # Sauter la première ligne (les headers)
        if line.startswith('ORDERNUMBER'): #premier champs (à sauter)
            return
        product_line = line.split(',')[10] #colonne 10 product-line
        yield None, product_line

    def reducer(self, key, values):
        #regrouper les entrées par catégorie de produit
        unique_product_lines = set(values)
        for product_line in unique_product_lines:
            yield product_line, None

if __name__ == '__main__':
    ProductLines.run()

```

Ce code effectue une analyse map-reduce sur le fichier sales_data_sample.csv en utilisant la bibliothèque MRJob. Le mapper lit chaque ligne du fichier, saute la première ligne qui contient les noms de champs et extrait la colonne de la catégorie de produit, qui est ensuite renvoyée avec une valeur de clé 1. Le reducteur regroupe ensuite les entrées par clé (catégorie de produit) et ne renvoie que les clés uniques, ce qui donne la liste complète des catégories de produits dans le fichier. On observe que pour résoudre cette problème, on a utilise un pair map-reduce. Le résultat obtenu ci-dessous :

```
"Trucks and Buses"      null
"Planes"        null
"Vintage Cars"  null
"Trains"        null
"Motorcycles"   null
"Ships" null
"Classic Cars"  null
```

Le fichier code source est disponible au nom de **question1.py** et le résultat **result1.txt**.

2.2 Quel est le nombre de ventes (QUANTITYORDERED) réalisé pour chaque catégorie de produits ?

Dans ce code, pour la fonction mapper, on défine la catégorie de produits (colonne PRODUCTLINE) et la quantité de vente (colonne QUANTITYORDERED) en tant que

clé-valeur. Dans la fonction reducer, on utilise la fonction "sum" pour sommer les quantités pour chaque catégorie de produits et on obtient le résultat en tant que clé-valeur. Le fichier code source est disponible au nom de **question2.py** et le résultat est affiché ci-dessous :

```
"Vintage Cars"  21069
"Classic Cars"  33992
"Planes"        10727
"Ships" 8127
"Trains"        2712
"Trucks and Buses"      10777
"Motorcycles"   11663
```

2.3 Quel est le chiffre d'affaire (cf QUANTITYORDERED et PRICEEACH) par an et par catégorie de produits (tous pays confondus)

L'année est extraite à partir de la colonne YEAR_ID et le chiffre d'affaires est calculé en multipliant QUANTITYORDERED par PRICEEACH. Les données sont produites en tant que clé-valeur avec la catégorie de produits et l'année en tant que clé et le chiffre d'affaires en tant que valeur. La fonction reducer somme ensuite les chiffres d'affaires pour chaque catégorie de produits et année. Le fichier code source est disponible au nom de **question3.py** et le résultat est affiché ci-dessous :

```
["Vintage Cars", "2003"]        "575626.41"
["Trucks and Buses", "2004"]    "440763.39"
["Trucks and Buses", "2005"]    "146772.36"
["Planes", "2005"]      "175848.66"
["Ships", "2003"]       "233746.81"
["Ships", "2004"]       "324998.78"
["Ships", "2005"]       "119194.81"
["Trains", "2003"]      "70955.24"
["Trains", "2004"]      "102157.70"
.
.
. (résultat complet dans le fichier result3.txt)
```

2.4 Pour chaque pays, quel est le magasin (CUSTOMERNAME) qui a réalisé le plus gros chiffre d'affaire dans la catégorie de produits Trucks and Buses ?

Dans la fonction mapper, seuls les enregistrements avec une catégorie de produits "Trucks and Buses" sont produits en tant que clé-valeur. La fonction reducer somme ensuite les chiffres d'affaires pour chaque magasin pour chaque pays. Enfin, la fonction reducer_final trouve le magasin avec le chiffre d'affaires le plus élevé pour chaque pays. Le fichier code source est disponible au nom de **question4.py** et le résultat est affiché ci-dessous :

```
"WX3 6FW"       ["\"Stylish Desk Decors", "25280.40"]
"Singapore"     ["Handji Gifts& Co", "40078.59"]
"Sweden"        ["Scandinavian Gift Ideas", "23775.52"]
"USA"   ["Corporate Gift Ideas Co.", "8405.70"]
"Denmark"       ["Heintze Collectables", "7300.00"]
.
.
. (résultat complet dans le fichier result4.txt)
```

2.5 Requête originale et complexe : Pour chaque pays, quel est le ville qui a le plus grande nombre de ventes dans la catégorie de produits Motorcycle ?

Ce code lit les données depuis un fichier csv et utilise deux étapes de réduction pour trouver la ville avec le plus grand nombre de ventes pour chaque pays. La première étape de réduction crée un dictionnaire avec la ville comme clé et la quantité totale de ventes comme valeur. La seconde étape de réduction trouve la ville avec la valeur maximale. Le code est disponible sur le fichier **question5.py**

```
"Norway"        ["Stavern", 212]
"Osaka" [" Kita-ku\"", 173]
"Philippines"   ["Makati City", 241]
"Sweden"        ["Boras", 133]
"T2F 8M4"       ["", 41]
"USA"   ["NYC", 931]
.
.
. (résultat complet dans le fichier result5.txt)
```

On note des résultats qui montre les apparitions des grandes villes comme centre de ventes, comme New York City et Paris.

3 Exercice 2 - Particularité lexicale

Ce code utilise MRJob pour décomposer le traitement en étapes, chacune effectuant une tâche spécifique. Le mapper lit les mots et compte le nombre de voyelles. S'il trouve une voyelle il garde la voyelle (variable "vowel" et found_vowel = "True") et vérifie les autres lettres de ce mot. Pour le cas qu'il n'y a qu'une voyelle, il rend le mot et sa longueur associée pour la voyelle correspondante. Sinon, il ne la prendre pas en compte. Le réducteur calcule le plus long mot pour chaque voyelle. Le code source est disponible sur le nom de **exercice2.py**. Le résultat est disponible ci-dessous et affiche la voyelle, la quantité de lettre du mot et le mot le plus grand pour le fichier **word_alpha.txt en anglais** :

```
"y"     [8, "symphysy"]
"a"     [13, "landsmanshaft"]
"i"     [13, "whipstitching"]
"o"     [13, "loxolophodont"]
"u"     [10, "untruthful"]
"e"     [16, "strengthlessness"]
```