École Centrale de Lyon

MSO 3.1 - Technologies informatiques du Big Data
Option Informatique
Rapport

TP1 – Hadoop map-reduce

Élèves :
Thibaud MIQUEL

Enseignant :
Stéphane DERRODE

Table des matières

Table des figures

1 Présentation du TP

1.1 Objectif du TP

L'objectif de ce TP est d'utiliser la bibliothèque MRJob pour coder des requêtes et les exécuter directement sur Hadoop en passant par Docker.

1.2 Présentation d'Hadoop

Hadoop est un framework Java open source utilisé pour le stockage et traitement des big data. Les données sont stockées sur des serveurs standard peu coûteux configurés en clusters.

Dans son système de fichiers distribué HDFS (Hadoop Distributed File System), les données sont divisées en datasets de taille réduite et enregistrées sur des clusters de serveurs standard. Ces serveurs de base étant construits avec des configurations matérielles très simples, ils sont peu coûteux et peuvent évoluer très facilement pour suivre l'augmentation des volumes de données.

Hadoop utilise le modèle de programmation fonctionnelle MapReduce pour exécuter un traitement parallèle entre les datasets. Lorsqu'une requête est transmise à la base de données, les jobs (et les données) spécifiés dans cette requête ne sont pas traités séquentiellement, mais répartis et exécutés simultanément sur différents serveurs. À l'issue du traitement, les résultats des différents jobs sont regroupés et renvoyés à l'application, ce qui améliore considérablement la vitesse de traitement.

YARN est l'abréviation de « Yet Another Resource Negotiator » (plus simplement, un négociateur de ressources). Cet élément assure la gestion et planification des ressources (clusters) Hadoop et décide de ce qui doit se passer dans chaque nœud de données. Le nœud maître central qui gère toutes les demandes de traitement est le « Resource Manager ». Le Resource Manager interagit avec les différents Node Managers : chaque DataNode esclave possède son propre Node Manager pour l'exécution des tâches.

1.3 Présentation de Map Reduce

MapReduce est un Framework de traitement de données en clusters, inventé par Google, dans lequel sont effectués des calculs parallèles, et souvent distribués, de données potentiellement très volumineuses.

Il s'agit d'un composant central du Framework logiciel Apache Hadoop, qui permet le traitement robuste et distribué d'ensembles de données non structurées massifs sur des clusters d'ordinateurs, au sein desquels chaque nœud possède son propre espace de stockage. Concrètement, le framework propose deux fonctionnalités principales. Il répartit le travail de calcul sur les différents nœuds du cluster (map), puis les organise et réduit les résultats fournis par chaque nœud en une seule réponse cohérente à une requête (reduce).

https ://github.com/mattwg/mrjob-examples

2 Réponse aux questions

2.1 Exercice 1 - Questionner un fichier de ventes

2.1.1 1 - Quel est le nombre d'achats effectués pour chaque catégorie d'achat ?

Pour la question 1 :

Q1.py > MRachatcateg > reducer

```python
#!/usr/bin/env python3

#-*- coding: utf-8 -*-
from mrjob.job import MRJob
import sys

class MRachatcateg(MRJob):
    def mapper(self, _, line):
        for word in [(line.split("\t"))[3]]:
            yield(word, 1)
    def reducer(self, word, counts):
        yield(word, sum(counts))

if __name__ == '__main__':
    MRachatcateg.run()
```

FIGURE 1 – Code pour la question 1

```
Streaming final output from hdfs:///user/root/tmp/mrjob/Q1.root.20220305.161437.723220/out
"Baby"  230293
"Books" 229787
"CDs"   230039
"Cameras"       229320
"Children's Clothing"   230469
"Computers"     229059
"Consumer Electronics"  229761
"Crafts"        229749
"DVDs"  230274
"Garden"        230073
"Health and Beauty"     229667
"Men's Clothing"        230430
"Music" 230150
"Pet Supplies"  229222
"Sporting Goods"        229932
"Toys"  229964
"Video Games"   230237
"Women's Clothing"      230050
Removing HDFS temp directory hdfs:///user/root/tmp/mrjob/Q1.root.20220305.161437.723220...
```

FIGURE 2 – Résultat pour la question 1

2.1.2 2 - Quelle est la somme totale dépensée pour chaque catégorie d'achat ?

Pour la question 2 :

Q2.py > ...

```python
#-*- coding: utf-8 -*-
from mrjob.job import MRJob
import sys

class MRsommetotcateg(MRJob):
    def mapper(self, _, line):
        for word in [(line.split("\t"))[3]]:
            yield(word, float((line.split("\t"))[4]))
    def reducer(self, word, counts):
        yield(word, sum(counts))

if __name__ == '__main__':
    MRsommetotcateg.run()
```

FIGURE 3 – Code pour la question 2

```
Streaming final output from hdfs:///user/root/tmp/mrjob/Q2.root.20220305.162328.951440/ou
"Baby"  57491808.439999565
"Books" 57450757.910000086
"CDs"   57410753.04000101
"Cameras"       57299046.64000095
"Children's Clothing"   57624820.93999975
"Computers"     57315406.31999985
"Consumer Electronics"  57452374.130001
"Crafts"        57418154.50000002
"DVDs"  57649212.13999978
"Garden"        57539833.109999985
"Health and Beauty"     57481589.56000032
"Men's Clothing"        57621279.04
"Music" 57495489.70000011
"Pet Supplies"  57197250.240000114
"Sporting Goods"        57599085.88999987
"Toys"  57463477.10999886
"Video Games"   57513165.579998754
"Women's Clothing"      57434448.969999254
Removing HDFS temp directory hdfs:///user/root/tmp/mrjob/Q2.root.20220305.162328.951440...
```

FIGURE 4 – Résultat pour la question 2

2.1.3 3 - Quelle somme est dépensée dans la ville de San Francisco dans chaque moyen de paiement ?

Pour la question 3 :

```python
#!/usr/bin/env python3

#-*- coding: utf-8 -*-
from mrjob.job import MRJob
import sys

class MRsommeSFmoyen(MRJob):
    def mapper(self, _, line):
        splitted_line=line.split("\t")
        if splitted_line[2]=="San Francisco":
            yield(splitted_line[-1], splitted_line[4])
    def reducer(self, word, counts):
        yield(word,sum(counts))

if __name__ == '__main__':
    MRsommeSFmoyen.run()
```

FIGURE 5 – Code pour la question 3

```
Streaming final output from hdfs:///user/root/tmp/mrjob/Q3.root.20220305.163608.016365/outp
"Amex"  1984573.8900000043
"Cash"  1982297.8699999987
"Discover"      2009176.530000004
"MasterCard"    2007667.0500000059
"Visa"  2011855.1999999892
Removing HDFS temp directory hdfs:///user/root/tmp/mrjob/Q3.root.20220305.163608.016365...
```

FIGURE 6 – Résultat pour la question 3

2.1.4 4 - Dans quelle ville la catégorie Women's Clothing a permis de générer le plus d'argent Cash ?

Pour la question 4 :

```python
#-*- coding: utf-8 -*-
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys

class MRmaxWomClothinCash(MRJob):
    def mapper(self, _, line):
        splitted_line = line.split("\t")
        if splitted_line[3]=="Women's Clothing" and splitted_line[5]=="Cash":
            yield(splitted_line[2], float(splitted_line[4]))

    def reducer_int(self, word, counts):
        yield None, (word,sum(counts))

    def reducer_find_max(self, _, pairs):
    # each item of word_count_pairs is (count, word),
    # so yielding one results in key=counts, value=word
        yield max(pairs)

    def steps(self):
        return [
            MRStep(mapper=self.mapper,
                   reducer=self.reducer_int),
            MRStep(reducer=self.reducer_find_max)
        ]

if __name__ == '__main__':
    MRmaxWomClothinCash.run()
```

FIGURE 7 – Code pour la question 4

```
job output is in hdfs:///user/root/tmp/mrjob/Q4.root.20220305.
Streaming final output from hdfs:///user/root/tmp/mrjob/Q4.roo
"Winston\u2013Salem"    107966.70999999992
Removing HDFS temp directory hdfs:///user/root/tmp/mrjob/Q4.ro
Removing temp directory /tmp/Q4.root.20220305.171305.735244...
```

FIGURE 8 – Résultat pour la question 4

2.1.5 5 - Ajoutez une requête originale et complexe (i.e. nombre de STEPS > 1) de votre choix sur ce fichier. Dans le CR, prenez le temps de m'expliquez l'objectif de la requête, de présenter le code et le résultat obtenu : tableau, graphique...

Ma requête consiste à renvoyer pour chaque ville et pour chaque catégorie présente dans ces villes le moyen de paiement ayant la somme de dépense la plus faible selon cette granularité d'information.

Pour cela on effectuera donc 2 étapes de map-reduce.

La première consiste à mapper tous les triplets de : city, catégorie d'achat et moyen de paiement avec leur montant puis la phase de reduce consiste à sommer ces montants.

La 2eme etapes consistes à mapper tous les duets de city et catégorie d'achat avec leur duet de moyen de paiement et de montant d'achat puis la phase de reduce consiste à prendre pour le premier duet le 2eme duet avec le montant minimal.

Q5.py > MRminCategCityPayment > steps

```python
#-*- coding: utf-8 -*-
from mrjob.job import MRJob
from mrjob.step import MRStep
import sys

class MRminCategCityPayment(MRJob):
    def mapper_1(self, _, line):
        splitted_line = line.split("\t")
        yield((splitted_line[2],splitted_line[3],splitted_line[5]), float(splitted_line[4]))

    def reducer_1(self, word, counts):
        yield(word,sum(counts))

    def mapper_2(self, word, counts):
        yield((word[0],word[1]), (word[2],counts))

    def reducer_2(self, word, counts):
        yield(word,min(counts))

    def steps(self):
        return [
            MRStep(mapper=self.mapper_1,
                   reducer=self.reducer_1),
            MRStep(mapper=self.mapper_2,
                   reducer=self.reducer_2)
        ]

if __name__ == '__main__':
    MRminCategCityPayment.run()
```

FIGURE 9 – Code pour la question 5

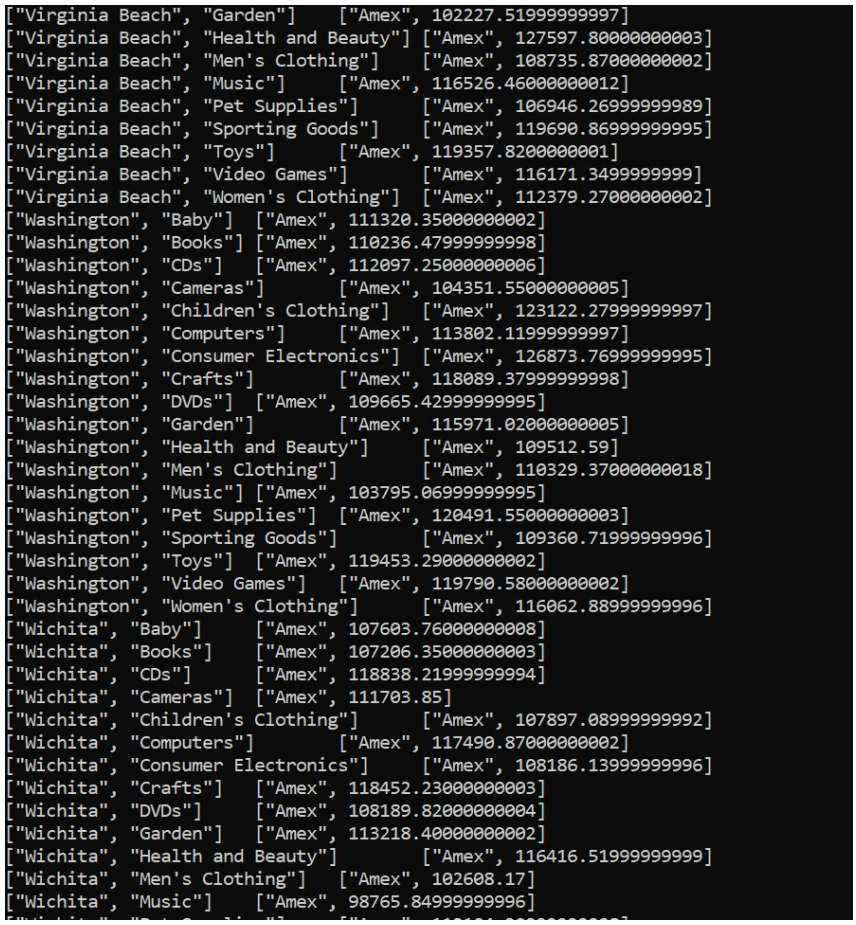

FIGURE 10 – Résultat pour la question 5

On peut voir que ce moyen de paiement est quasi systèmatiquement "Amex"

2.2 Exercice 2 - Anagramme

Étant donné un fichier de mots, écrire un script map-reduce qui détecte les mots ayant exactement les mêmes lettres (mais dans un ordre différent).

Cet algorithme devra fonctionner sous environnement Hadoop, avec plusieurs mapper et reducer en parallèle. L'algorithme ne devra pas tenir compte de la présence éventuelle de majuscules dans les mots.

anagram.py > ...

```python
from mrjob.job import MRJob

class MRAnagram(MRJob):

    def mapper(self, _, line):
        # On convertit les mots en liste de lettre minuscule puis on ordonne cette liste
        letters = list(str(line).lower())
        letters.sort()

        # on utilise comme clé la liste de lettre ordonnée
        yield letters, line

    def reducer(self, _, words):
        # on renvoie les mots ayant au moins 2 anagrames en matchant par liste ordonnée
        anagrams = [w for w in words]
        if len(anagrams) > 1:
            yield len(anagrams), anagrams


if __name__ == "__main__":
    MRAnagram.run()
```

FIGURE 11 – Code pour la partie 2

```
2       ["sprouts", "stupors"]
2       ["sports", "strops"]
2       ["prossy", "spyros"]
3       ["stroup", "sprout", "stupor"]
2       ["sporty", "sproty"]
5       ["sprot", "prost", "strop", "ports", "sport"]
3       ["pours", "roups", "porus"]
2       ["pros", "spor"]
2       ["pottur", "prutot"]
2       ["port", "trop"]
2       ["pour", "roup"]
3       ["pyro", "pory", "ropy"]
2       ["pro", "por"]
4       ["tossup", "spouts", "stoups", "uptoss"]
3       ["posts", "stops", "spots"]
2       ["poss", "sops"]
2       ["outputs", "putouts"]
2       ["spouty", "outspy"]
4       ["stoup", "potus", "spout", "pouts"]
3       ["typos", "topsy", "potsy"]
6       ["post", "tops", "opts", "pots", "spot", "stop"]
2       ["soupy", "pousy"]
2       ["opus", "soup"]
3       ["pows", "swop", "wops"]
2       ["posy", "opsy"]
3       ["ops", "pos", "sop"]
2       ["output", "putout"]
2       ["toup", "pout"]
3       ["opt", "pot", "top"]
2       ["pow", "wop"]
2       ["po", "op"]
2       ["torr", "rort"]
4       ["rousts", "tussor", "trouss", "stours"]
2       ["strows", "worsts"]
3       ["sorts", "sstor", "ssort"]
2       ["sorus", "sours"]
3       ["trouts", "tutors", "strout"]
2       ["torts", "trots"]
7       ["routs", "sutor", "roust", "torus", "tours", "stour", "ru
4       ["worst", "strow", "worts", "trows"]
```

FIGURE 12 – Résultat pour la partie 2