



## TP n°03 : Map Reduce

---

### Table des matières

- I. Objectif du TP
- II. Map Reduce
  - A. Présentation
  - B. Wordcount
    - 1. Création du projet en Python
    - 2. Utilisation de notre Map-Reduce dans Hadoop
    - 3. Vérification par les outils Web d'Hadoop
- III. Exercice suite au TP

### I. Objectif du TP

- Après le TP02\_Hadoop\_HDFS
- Initiation au Framework Hadoop et au patron MapReduce en Python

## II. Map Reduce

### A. Présentation

Un Job Map-Reduce se compose principalement de deux types de programmes:

- Mappers : permettent d'extraire les données nécessaires sous forme de clef/valeur, pour pouvoir ensuite les trier selon la clef
- Reducers : prennent un ensemble de données triées selon leur clef, et effectuent le traitement nécessaire sur ces données (somme, moyenne, total...)

### B. Wordcount

Nous allons tester un programme MapReduce grâce à un exemple très simple, le WordCount, l'équivalent du HelloWorld pour les applications de traitement de données. Le Wordcount permet de calculer le nombre de mots dans un fichier donné, en décomposant le calcul en deux étapes:

- L'étape de Mapping, qui permet de découper le texte en mots et de délivrer en sortie un flux textuel, où chaque ligne contient le mot trouvé, suivi de la valeur 1 (pour dire que le mot a été trouvé une fois)
- L'étape de Reducing, qui permet de faire la somme des 1 pour chaque mot, pour trouver le nombre total d'occurrences de ce mot dans le texte.

#### 1. Création du projet en Python

Sur le master créer deux fichier python mapper et reducer comme suite : `docker exec -it hadoop-master bash`

Note: si votre conteneur n'est pas démarré lancer la commande suivante : `docker restart hadoop-master`

##### *Création du fichier mapper.py*

Vi mapper.py

```
import sys import logging

logging.basicConfig(filename='debug.log',level=logging.DEBUG)
logging.debug("Entering mapper.py")
for line in sys.stdin:
    logging.debug("Inside for loop " + line)
    line = line.strip()
    for word in line.split(" "):
        if len(word) > 0 :
            print("%s\t%i" %(word.lower(), 1))
```

### Création du fichier reducer.py

Vi reducer.py

```
import sys
import logging

logging.basicConfig(filename='debug.log', level=logging.DEBUG)

current_word = None
current_count = 0
word = None

print "Entering reducer.py"
for line in sys.stdin:
    line = line.strip()
    parts = line.split('\t', 1)
    if len(parts) < 2:
        continue
    word = parts[0]
    count = parts[1]
    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if(current_word):
            print("%s\t%s" %(current_word,current_count))
        current_count = count
        current_word = word

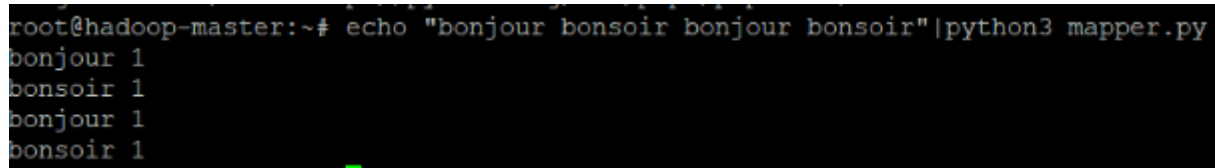
if current_word == word:
    print("%s\t%s" %(current_word, current_count))
```

### Tester notre code en local

Mapper uniquement en lançant la commande suivante

```
echo "bonjour bonsoir bonjour bonsoir" | python mapper.py
```

Résultat :



```
root@hadoop-master:~# echo "bonjour bonsoir bonjour bonsoir"|python3 mapper.py
bonjour 1
bonsoir 1
bonjour 1
bonsoir 1
```

## Mapper et Reducer

```
echo "bonjour bonsoir bonjour bonsoir" | python mapper.py | python reducer.py
```

Résultat :

```
root@hadoop-master:~# echo "bonjour bonsoir bonjour bonsoir"|python3 mapper.py |
python3 reducer.py
bonjour 1
bonsoir 1
bonjour 1
bonsoir 1
```

**Note:** N'oubliez pas que entre les deux fonctionnalités map et reduce, on le sort, lancer la commande suivante

```
:echo "bonjour bonsoir bonjour bonsoir" | python mapper.py | sort -k1,1 | python
reducer.py
```

Résultat :

```
root@hadoop-master:~# echo "bonjour bonsoir bonjour bonsoir"|python3 mapper.py | s
ort -k1,1 |python3 reducer.py
bonjour 2
bonsoir 2
```

## 2. Utilisation de notre Map-Reduce dans Hadoop

- Lancer les services Hadoop : `./start-hadoop.sh`
- Suppression du dossier /output (à faire au cas où) : `hdfs dfs -rmkdir /output`
- Vérifier si le fichier **hadoop-streaming-2.7.2.jar** existe bien dans le dossier :  
`ls $HADOOP_HOME/share/hadoop/tools/lib`
- S'il n'existe pas, télécharger le jar depuis internet et copier le dans le chemin :  
`$HADOOP_HOME/share/hadoop/tools/lib`
- **Note :** si vous avez téléchargé le fichier sur votre machine
  - Copier les fichiers dans un dossier de partage sous Docker (dans un dossier hadoop par exemple)
  - `docker cp hadoop/ hadoop-streaming-2.7.2.jar hadoop-master:/root/ hadoop-streaming-2.7.2.jar`
- Lancer le mapper et le reducer sur hadoop avec le fichier purchases.txt :
  - `hadoop fs -mkdir /user/root/output`
  - `hadoop fs -mkdir /user/root/intput`
  - `hadoop fs -put purchases.txt /user/root/intput`
  - `hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming- 2.7.2.jar -file /root/mapper.py -mapper "python3 mapper.py" -file /root/reducer.py -reducer "python3 reducer.py" -input /user/root/input/purchases.txt -output /user/root/output/wordcount`
- Afficher le résultat stocké sur HDFS:  
`hadoop fs -tail output/ wordcount /part-00000`

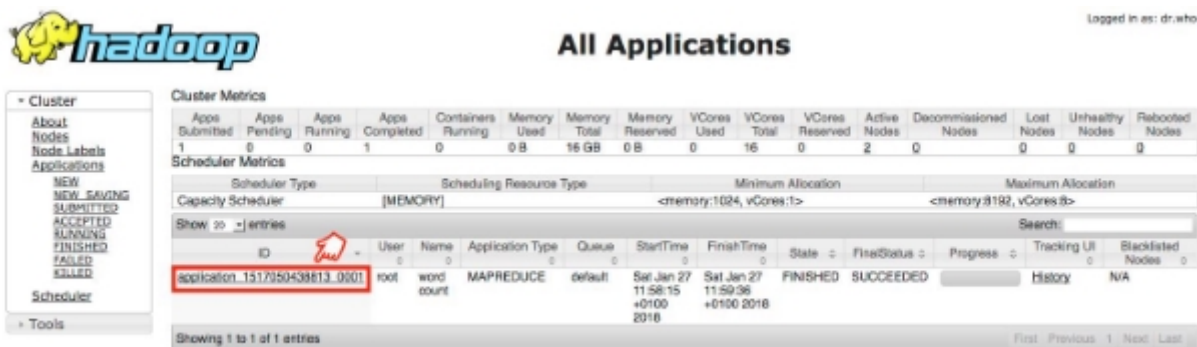
```

Petersburg      8430
Philadelphia    8471
Phoenix 8431
Pittsburgh      8470
Plano 8323
Portland        8367
Raleigh 8345
Reno 8334
Richmond        8388
Riverside       8338
Rochester       8440
Rouge 8396
Sacramento      8597
Saint 8494
San 42110
Santa 8416
Scottsdale      8443
Seattle 8339
Spokane 8356
Sporting        48207
Springs 8534
St. 16881
Stockton        8289
Supplies        48265
Tampa 8400
Toledo 8314
Toys 48463
Tucson 8546
Tulsa 8444
Vegas 16957
Video 48439
Virginia        8465
Visa 174018
Vista 8510
Washington      8477
Wayne 8527
Wichita 8547
Winston-Salem   8459
Women's 48252
Worth 8462
York 8529
and 48408

```

### 3. Vérification par les outils Web d'Hadoop

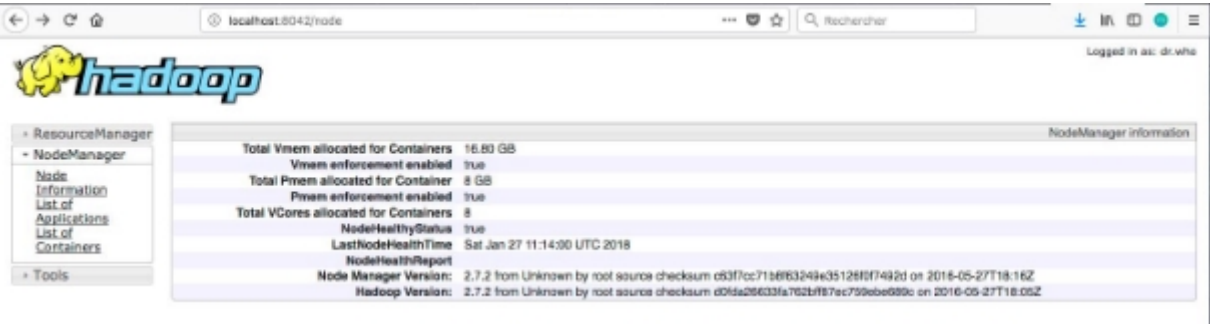
Il vous est possible de monitorer vos Jobs Map Reduce, en allant à la page: <http://localhost:8088>. Vous trouverez votre Job dans la liste des applications comme suit:



The screenshot shows the Hadoop YARN web interface. On the left is a navigation menu with options like Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, and Scheduler. The main area is titled 'All Applications' and displays a table of application metrics. A red box highlights the application ID 'application\_1317950438613\_0601' in the 'ID' column. The application is in the 'FINISHED' state with a 'SUCCEEDED' final status. The table includes columns for App ID, User, Name, Application Type, Queue, Start Time, Finish Time, State, Final Status, Progress, Tracking URL, and Stacklisted Nodes.

App ID	User	Name	Application Type	Queue	Start Time	Finish Time	State	Final Status	Progress	Tracking URL	Stacklisted Nodes
application_1317950438613_0601	root	word count	MAPREDUCE	default	Sat Jan 27 11:58:15 +0100 2018	Sat Jan 27 11:59:36 +0100 2018	FINISHED	SUCCEEDED		History	N/A

Il est également possible de voir le comportement des noeuds esclaves, en allant à l'adresse: <http://localhost:8041> pour **slave1**, et <http://localhost:8042> pour **slave2**. Vous obtiendrez ce qui suit:



III. Exercice suite au TP

- Écrire un Job **Map Reduce** permettant, à partir du fichier **purchases initial**, de déterminer le total des ventes par magasin.
- Écrire un Job **Map Reduce** permettant, à partir du fichier **purchases initial**, de déterminer le total des ventes, la moyenne des ventes et nombre des ventes par magasin.

La structure du fichier **purchases** est de la forme suivante:

date	temps	magasin	produit	cout	paiement
------	-------	---------	---------	------	----------

- Veiller à toujours tester votre code en local avant de lancer un job sur le cluster. TP02 Hadoop Map reduce Page : 8