

FILIÈRE INGÉNIERIE DES SYSTÈMES UBIQUITAIRES ET DISTRIBUÉS – CLOUD ET IoT (SUD)

TP 2 : Installation et utilisation d'Apache Spark

MATIÈRE

SÉCURITÉ DANS LE CLOUD ET L'IOT

Élèves :
Mahjouba LABAALLI
Fatima OUBELKAS

Enseignant



Table des matières

Introduction	4
1 Installation et configuration de Hadoop dans un nœud unique en local	5
2 Installation et configuration de spark en local	5
2.1 Récupération des fichiers sources	5
2.2 Décompression du fichier spark récupéré	6
2.3 Configuration du PATH dans le fichier .bashrc	7
2.4 Installation de Python	8
3 Manipulation des RDDs en utilisant le terminal pyspark	9
3.1 Enregistrement et chargement des Textfile	9
3.2 Enregistrement et chargement des SequenceFiles	11
3.3 Utilisation d'une fonction nommée	11
3.4 Utilisation d'une fonction anonyme	12
3.5 Utilisation de « parallelize »	12
3.6 Utilisation de « wholeTextFiles »	13
3.7 Utilisation de « flatMap » et « distinct »	14
3.8 Utilisation de « subtract » et « zip »	14
3.9 Utilisation de intersection et union	15
4 Connexion de Spark à une distribution de Hadoop	16
5 Exécution du « Word Count » en utilisant le terminal scala et python	16
5.1 Dépôt du poeme.txt dans le HDFS	16
5.2 Exécution du code du « Word Count »	17
5.2.1 Exécution du code du « Word Count » en scala	17
5.2.2 Exécution du code du « Word Count » en pyspark	18
6 Exécution du « Word Count » en utilisant un script python	19
7 Exécution d'une application Spark Batch en Java	20
7.1 Installation d'Apache Maven	20
7.2 Reconfiguration du projet Maven	23
7.3 Nettoyage et Formatage du nœud hadoop	25
7.4 Dépôt du poeme.txt dans HDFS	26
Conclusion	28

Table des figures

1	L'écosystème Spark	4
2	Test de la configuration du hadoop	5
3	Les fichiers nécessaires pour le TP	6
4	Décompression du fichier spark récupéré	7
5	Configuration du PATH dans le fichier .bashrc	7
6	Installation de Python2	8
7	Le terminal Scala	8
8	Le terminal pyspark	9
9	Le fichier ValeursINPT.txt	9
10	Affichage du contenu de ValeursINPT.txt en utilisant le termina pyspark	10
11	Filtrage du contenu de ValeursINPT.txt en utilisant le termina pyspark	10
12	Le contenu du répertoire values_startsWithN	10
13	Enregistrement et chargement des SequenceFiles	11
14	Le contenu du répertoire /fileseq1	11
15	Utilisation d'une fonction nommée	12
16	Utilisation d'une fonction anonyme	12
17	Utilisation de « parallelize »	13
18	Les fichiers file1.json et file2.json	13
19	Utilisation de « wholeTextFiles »	13
20	Utilisation de « flatMap » et « distinct »	14
21	Utilisation de « subtract »	15
22	Utilisation de « zip »	15
23	Utilisation de intersection	15
24	Utilisation de union	16
25	Connexion de Spark à une distribution de Hadoop	16
26	Le contenu du fichier conf/spark-env.sh	16
27	Dépôt du poeme.txt dans le HDFS	17
28	Exécution du code du « Word Count » en scala	17
29	Le résultat du programme Word Count	18
30	Exécution du code du « Word Count » en pyspark	18
31	Le résultat du programme Word Count en pyspark	19
32	Le fichier « word_count.py »	19
33	L'exécution du fichier « word_count.py »	20
34	Visualisation des résultat du fichier « word_count.py »	20
35	Le fichier « apache-maven-3.5.0-bin.tar.gz »	21
36	Le fichier /etc/profile	21
37	Teste de la configuration de maven	21
38	Les paramètres pour la création du projet maven	22
39	Message de « Build success »	22
40	mvn package	22
41	L'arborescence du projet	23
42	Le contenu entre <properties>et </properties>	23
43	Le contenu entre <dependencies> et </dependencies>	24
44	Le contenu du fichier WordCountTask.java	24
45	Compilation du code	25

46	Nettoyage du noeud hadoop	25
47	Formatage du noeud hadoop	25
48	Test si le noeud est disponible	26
49	Dépôt du poeme.txt dans HDFS	26
50	spark-submit	27
51	Affichage des résultats	27

Introduction

Spark représente un système de traitement rapide et parallèle, offrant des interfaces de haut niveau dans des langages tels que Java, Scala, Python, et R. Il intègre un moteur optimisé qui prend en charge l'exécution de graphes, ainsi qu'un ensemble d'outils avancés incluant Spark SQL pour le traitement de données structurées, MLlib pour l'apprentissage de données, GraphX pour le traitement de graphes, et Spark Streaming pour le traitement des données en continu.

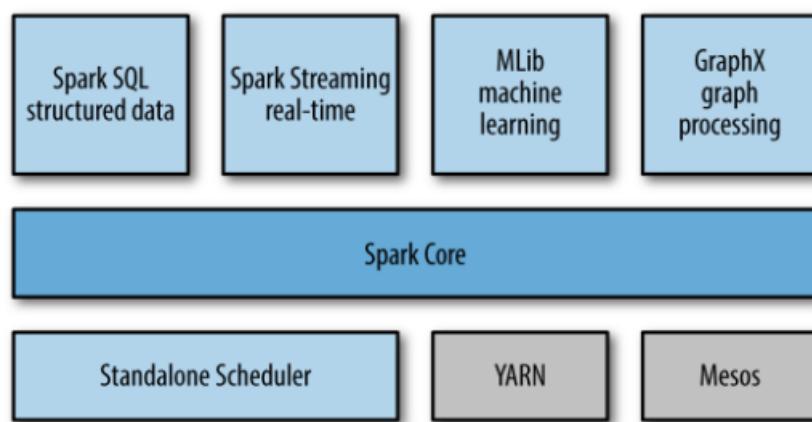


FIGURE 1 – L'écosystème Spark

Les objectifs de ce TP sont les suivants :

- Installation et configuration de Spark dans un nœud unique en local.
- Manipulation des RDDs en utilisant le terminal « pyspark ».
- Connexion de Spark à une distribution de Hadoop.
- Exécution du traitement « Word Count » en utilisant le terminal scala et python.
- Exécution du traitement « Word Count » en utilisant un script python.
- Exécution d'une application Spark Batch en Java.

1 Installation et configuration de Hadoop dans un nœud unique en local

Tout d'abord, créons une machine virtuelle 'ubuntu' en utilisant 'Virtualbox' avec les caractéristiques suivantes : 4096 Mio de mémoire et 20 Go de disque. Suivons ensuite les étapes d'installation et de configuration de Hadoop3 dans un nœud unique, comme détaillé dans la section 1 du TP1. Pour ce faire, nous effectuerons les actions suivantes :

1. Création d'un utilisateur hduser
2. Mise en place de la clé ssh
3. Installation de JAVA 8
4. Installation et configuration d'Apache Hadoop

Finalement, testons notre configuration en tapant : **\$ hdfs dfsadmin -report**

```

hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop/etc/hadoop$ hdfs dfsadmin -report
2023-10-24 22:41:42,215 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Configured Capacity: 21312888832 (19.85 GB)
Present Capacity: 1900490752 (1.77 GB)
DFS Remaining: 1900466176 (1.77 GB)
DFS Used: 24576 (24 KB)
DFS Used%: 0.00%
Replicated Blocks:
    Under replicated blocks: 0
    Blocks with corrupt replicas: 0
    Missing blocks: 0
    Missing blocks (with replication factor 1): 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
Erasure Coded Block Groups:
    Low redundancy block groups: 0
    Block groups with corrupt internal blocks: 0
    Missing block groups: 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
-----
Live datanodes (1):
Name: 127.0.0.1:9866 (localhost)
Hostname: labaallioubelkas-VirtualBox
Decommission Status : Normal
Configured Capacity: 21312888832 (19.85 GB)
DFS Used: 24576 (24 KB)

```

FIGURE 2 – Test de la configuration du hadoop

2 Installation et configuration de spark en local

Dans ce TP, nous allons utiliser Spark en local sur la VM ubuntu créée précédemment. Nous allons exécuter Spark sur Hadoop YARN. YARN s'occupera ainsi de la gestion des ressources pour le déclenchement et l'exécution des Jobs Spark.

2.1 Récupération des fichiers sources

Nous pouvons télécharger la version 3.3.0 du spark à partir du lien suivant : <https://archive.apache.org/dist/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz>

Ensuite nous récupérons le fichier : « poeme.txt » à partir de la plateforme moodle de l'INPT.

Et voici la totalité des documents nécessaires pour ce TP.

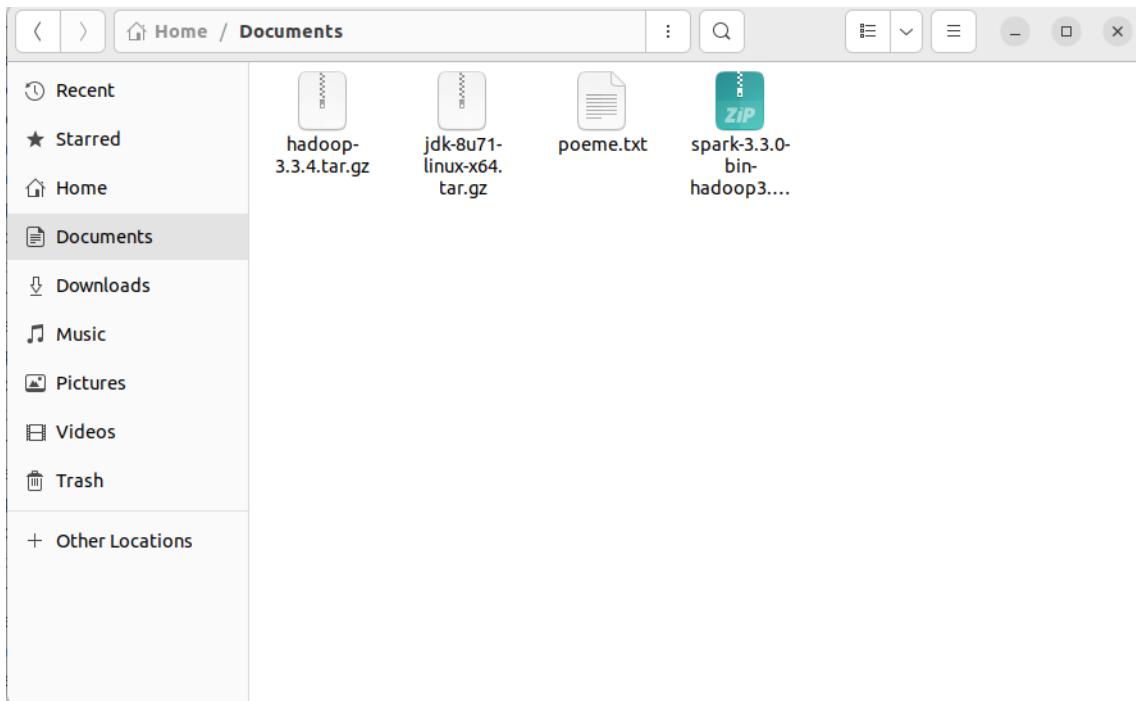


FIGURE 3 – Les fichiers nécessaires pour le TP

Tous ces fichiers sont existes au niveau du lien drive suivant : [lien drive](#)

2.2 Décompression du fichier spark récupéré

Pour décompresser le fichier récupéré, accédons au répertoire où le fichier existe et tapons la commande suivante : `$ tar xvzf spark-3.3.0-bin-hadoop3.tgz`

Ensuite, nous pouvons voir qu'un répertoire spark-3.3.0-bin-hadoop3 a été créé ; Nous le renommons en utilisant la commande : `$ mv spark-3.3.0-bin-hadoop3 spark`, et plaçons le dans le répertoire `/usr/local` avec la commande : `$ sudo mv spark /usr/local/`

```
hduser@labaallioubelkas-VirtualBox: ~/Documents
spark-3.3.0-bin-hadoop3/licenses/LICENSE-javassist.html
spark-3.3.0-bin-hadoop3/licenses/LICENSE-zstd.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-json-formatter.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-matchMedia-polyfill.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-scala.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-jakarta.activation-api.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-automaton.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-javax-transaction-transaction-api.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-jaxb-runtime.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-minlog.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-mustache.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-xmlenc.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-jline.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-istack-commons-runtime.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-py4j.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-vis-timeline.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-blas.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-re2j.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-kryo.txt
spark-3.3.0-bin-hadoop3/licenses/LICENSE-cloudpickle.txt
hduser@labaallioubelkas-VirtualBox:~/Documents$ mv spark-3.3.0-bin-hadoop3 spark
hduser@labaallioubelkas-VirtualBox:~/Documents$ sudo mv spark /usr/local/
hduser@labaallioubelkas-VirtualBox:~/Documents$
```

FIGURE 4 – Décompression du fichier spark récupéré

2.3 Configuration du PATH dans le fichier .bashrc

Nous ouvrons le fichier **.bashrc** en utilisant **\$vim .bashrc** et modifions le **PATH** en ajoutant les deux lignes suivantes enfin d'ajouter le chemin où se trouve le bin de spark :

```
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
```

```
hduser@labaallioubelkas-VirtualBox: ~
if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi

#HADOOP VARIABLES START
export JAVA_HOME=/opt/java/jdk1.8.0_71/
export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
#export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
.bashrc" 133L, 4327B
133,21
Bot
```

FIGURE 5 – Configuration du PATH dans le fichier .bashrc

En fin, pour sauvegarder la configuration effectuée nous tapons la commande suivante :
\$ source .bashrc

2.4 Installation de Python

Pour installer Python, nous tapons dans notre terminal : **\$ sudo apt-get install python2**

```
hduser@labaalliuobelkas-VirtualBox:~$ sudo apt-get install python2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  systemd-hwe-hwdb
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libpython2.7-stdlib libpython2.7-minimal libpython2.7-stdlib python2-minimal
  python2.7 python2.7-minimal
Suggested packages:
  python2-doc python-tk python2.7-doc binfmt-support
The following NEW packages will be installed:
  libpython2.7-stdlib libpython2.7-minimal libpython2.7-stdlib python2
  python2-minimal python2.7 python2.7-minimal
0 upgraded, 7 newly installed, 0 to remove and 425 not upgraded.
Need to get 4,005 kB of archives.
After this operation, 16.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ma.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libpython
```

FIGURE 6 – Installation de Python2

À ce stade , nous avons Spark et il est utilisable.

Nous pouvons accéder au terminal Scala en accédant tout d'abord au répertoire spark (`$ cd /usr/local/spark/` puis nous executons le fichier **spark-shell** en tapant :
`$./bin/spark-shell`

FIGURE 7 – Le terminal Scala

Nous pouvons également accéder au terminal Python en executant le fichier **pyspark** en tapant : **\$. ./bin/pyspark**

```

hduser@labaallioubelkas-VirtualBox: ~
hduser@labaallioubelkas-VirtualBox: ~ cd /usr/local
hduser@labaallioubelkas-VirtualBox: /usr/local$ cd /spark
bash: cd: /spark: No such file or directory
hduser@labaallioubelkas-VirtualBox: /usr/local$ cd /usr/local/spark
hduser@labaallioubelkas-VirtualBox: /usr/local/spark$ ./bin/pyspark
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
23/10/24 13:20:38 WARN Utils: Your hostname, labaallioubelkas-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
)
23/10/24 13:20:38 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
23/10/24 13:20:42 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
23/10/24 13:20:45 WARN Utils: Service 'SparkUI' could not bind on port 4040. Att
empting port 4041.
Welcome to
   ____          _ _ _ _ 
  / \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
 /_ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
  / \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
 /_ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
  / \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
 /_ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
version 3.3.0

Using Python version 3.10.4 (main, Jun 29 2022 12:14:53)
Spark context Web UI available at http://10.0.2.15:4041
Spark context available as 'sc' (master = local[*], app id = local-1698150046032
).
SparkSession available as 'spark'.
>>> 
```

FIGURE 8 – Le terminal pyspark

3 Manipulation des RDDs en utilisant le terminal pyspark

Dans cette section, nous allons appliquer les différents exemples vus dans le cours.

3.1 Enregistrement et chargement des Textfile

Créons un fichier **ValeursINPT.txt** dans le répertoire **usr/local/spark** comme suit :

```

hduser@labaallioubelkas-VirtualBox: /usr/local/spark
Nos valeurs à l'INPT sont :
Numérique par nature.
Renouvellement permanent.
Innovation et entrepreneuriat.
Ouverture sur l'écosystème.

"ValeursINPT.txt" 5L, 143B      5.31-27      All 
```

FIGURE 9 – Le fichier **ValeursINPT.txt**

Accédons de nouveau au terminal pyspark (`$./bin/pyspark`) et tapons les commandes suivantes :

En premier temps, on récupère le contenu du fichier **ValeursINPT.txt** comme suit :

```
» mydata = sc.textFile("file :/usr/local/spark/ValeursINPT.txt")
```

Puis, on affiche le contenu en utilisant :

```
» for line in mydata.collect():
... print(line)
```

```
>>>
>>>
>>> for line in mydata.collect():
...     print(line)
...
Nos valeurs à l'INPT sont :
Numérique par nature.
Renouvellement permanent.
Innovation et entrepreneuriat.
Ouverture sur l'écosystème.
>>> █
```

FIGURE 10 – Affichage du contenu de **ValeursINPT.txt** en utilisant le terminal pyspark

Nous filtrons maintenant les lignes qui commencent par la lettre "N" avec les commandes suivantes :

```
» mydata_filt = mydata.filter(lambda s : s.startswith('N'))
» mydata_filt.saveAsTextFile("file :/usr/local/spark/values_starts_withN")
```

Finalement, nous présentons initialement le nombre total de lignes en utilisant :

```
» mydata.count()
```

Puis procémons au calcul du nombre de lignes débutant par la lettre "N" en tapant :

```
» mydata_filt.count()
```

```
Nos valeurs à l'INPT sont :
Numérique par nature.
Renouvellement permanent.
Innovation et entrepreneuriat.
Ouverture sur l'écosystème.
>>> mydata_filt = mydata.filter(lambda s: s.startswith('N'))
>>> mydata_filt.saveAsTextFile("file:/usr/local/spark/values_starts_withN")
>>> mydata.count()
5
>>> mydata_filt.count()
2
>>>
```

FIGURE 11 – Filtrage du contenu de **ValeursINPT.txt** en utilisant le terminal pyspark

Vérifions alors que dans le répertoire `/usr/local/spark` qu'un répertoire nommé `values_starts_withN` est bien créé, ce répertoire doit contenir deux fichiers : `part-00000` et `_SUCCESS`.

```
hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ ls
bin  examples  LICENSE  python  RELEASE  values_starts_withN
conf  jars  licenses  R  sbin  yarn
data  kubernetes  NOTICE  README.md  ValeursINPT.txt
hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ cat values_starts_withN/
cat: values_starts_withN/: Is a directory
hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ cd values_starts_withN/
hduser@labaallioubelkas-VirtualBox:/usr/local/spark/values_starts_withN$ ls
part-00000  part-00001  _SUCCESS
hduser@labaallioubelkas-VirtualBox:/usr/local/spark/values_starts_withN$
```

FIGURE 12 – Le contenu du répertoire `values_starts_withN`

3.2 Enregistrement et chargement des SequenceFiles

Maintenant, nous allons créer, sauvegarder et récupérer un fichier de séquence en suivant les étapes suivantes :

1. Création d'une RDD (Resilient Distributed Dataset) :
» `rdd = sc.parallelize(range(1, 4)).map(lambda x : (x, "a" * x))`
 2. Sauvegarde de la RDD en tant que fichier de séquence :
» `rdd.saveAsSequenceFile("file :/usr/local/spark/fileseq1")`
 3. Récupération et tri du fichier de séquence :
» `sorted(sc.sequenceFile("file :/usr/local/spark/fileseq1").collect())`

```
sparkSession.createDataFrame(sc.parallelize(range(1, 4)).map(lambda x: (x, "a" * x)))
>>> rdd.saveAsSequenceFile("file:/usr/local/spark/fileseq1")
>>> sorted(sc.sequenceFile("file:/usr/local/spark/fileseq1").collect())
[(1, 'a'), (2, 'aa'), (3, 'aaa')]
>>> 
```

FIGURE 13 – Enregistrement et chargement des SequenceFiles

Vérifions alors que dans le répertoire /usr/local/spark qu'un répertoire nommé fileseq1 est bien créé, ce répertoire doit contenir deux fichiers : part-00000 et SUCCESS.

```
hduser@labaallioubelkas-VirtualBox: ~
```

hduser@labaallioubelkas-VirtualBox: ~\$ cd /usr/local/spark
hduser@labaallioubelkas-VirtualBox: /usr/local/spark\$ ls
bin examples kubernetes NOTICE README.md ValeursINPT.txt
conf fileseq1 LICENSE python RELEASE values_starts_withN
data jars licenses R sbin yarn
hduser@labaallioubelkas-VirtualBox: /usr/local/spark\$ cat fileseq1/
cat: fileseq1/: Is a directory
hduser@labaallioubelkas-VirtualBox: /usr/local/spark\$ cd fileseq1/
hduser@labaallioubelkas-VirtualBox: /usr/local/spark/fileseq1\$ ls
part-00000 part-00001 _SUCCESS
hduser@labaallioubelkas-VirtualBox: /usr/local/spark/fileseq1\$

FIGURE 14 – Le contenu du répertoire /fileseq1

3.3 Utilisation d'une fonction nommée

Nous considérons l'exemple suivant :

1. Définition de la fonction toUpper :
» **def toUpper(s) :**
.. **return s.upper()**
..
 2. Lecture du fichier texte dans une RDD :
» **mydata = sc.textFile("file :/usr/local/spark/ValeursINPT.txt")**
 3. Application de la transformation map :
» **mydataupper = mydata.map(toUpper)**
 4. Affichage des résultats avec une boucle for :
» **for line in mydataupper.collect() :**
.. **print(line)**

```
>>> def toUpper(s):
...     return s.upper()
...
>>> mydata = sc.textFile("file:/usr/local/spark/ValeursINPT.txt")
>>> mydataupper = mydata.map(toUpper)
>>> for line in mydataupper.collect():
...     print(line)
...
NOS VALEURS À L'INPT SONT :
NUMÉRIQUE PAR NATURE.
RENOUVELLEMENT PERMANENT.
INNOVATION ET ENTREPRENEURIAT.
OUVERTURE SUR L'ÉCOSSYSTEME.
>>>
```

FIGURE 15 – Utilisation d'une fonction nommée

3.4 Utilisation d'une fonction anonyme

Déposons d'abord le fichier : « poeme.txt » dans /usr/local/spark.

1. Lecture du fichier texte dans une RDD :

```
» mydata = sc.textFile("file :/usr/local/spark/poeme.txt")
```

2. Application de la transformation map avec une fonction lambda :

```
» mydata_upper = mydata.map(lambda line : line.upper()).take(5)
```

3. Affichage des résultats avec une boucle for :

```
» for line in mydata_upper :
.. print(line)
```

```
>>> mydata = sc.textFile("file:/usr/local/spark/poeme.txt")
>>> mydata_upper = mydata.map(lambda line: line.upper()).take(5)
>>> for line in mydata_upper:
...     print(line)
...
CELUI QUI CROYAIT AU CIEL
CELUI QUI NY CROYAIT PAS
TOUS DEUX ADORAIENT LA BELLE
PRISONNIERE DES SOLDATS
LEQUEL MONTAIT A LECHELLE
>>> █
```

FIGURE 16 – Utilisation d'une fonction anonyme

3.5 Utilisation de « parallelize »

Dans cette partie, nous allons utiliser la fonction parallelize :

1. Crédation d'une RDD à partir d'une liste :

```
» data = [10, 20, 30, 40, 50, 100, 250]
» distData = sc.parallelize(data)
```

2. Réduction de la RDD avec une fonction lambda :

```
» total = distData.reduce(lambda a,b : a + b)
```

3. Affichage du résultat :

```
» print (total)
```

```
>>> data = [10, 20, 30, 40, 50, 100, 250]
>>> distData = sc.parallelize(data)
>>> total = distData.reduce(lambda a,b: a + b)
>>> print(total)
500
```

FIGURE 17 – Utilisation de « parallelize »

3.6 Utilisation de « wholeTextFiles »

Tout d'abord dans le répertoire /usr/local/spark, créons un répertoire nommé : **json_files** et dans ce répertoire, créons les deux fichiers json : **file1.json** et **file2.json** avec le contenu suivant :

The image shows two terminal windows side-by-side. The top window displays the contents of file1.json:

```
{"firstName": "Fred",
"lastName": "Flintstone",
"userid": "123"}
```

The bottom window displays the contents of file2.json:

```
{"firstName": "Barney",
"lastName": "Rubble",
"userid": "234"}
```

FIGURE 18 – Les fichiers **file1.json** et **file2.json**

Ensuite, nous tapons le code suivante :

1. Importation de la bibliothèque JSON :
» **import json**
2. Création d'une RDD à partir de fichiers texte complets :
» **myrdd1 = sc.wholeTextFiles("file :/usr/local/spark/json_files")**
3. Conversion des données JSON dans la RDD :
» **myrdd2 = myrdd1.map(lambda a : json.loads(a[1]))**
4. Affichage des valeurs associées à la clé "firstName" pour les deux premiers enregistrements :
» **for record in myrdd2.take(2) :**
.. **print(record.get("firstName",None))**

```
>>> data = [10, 20, 30, 40, 50, 100, 250]
>>> distData = sc.parallelize(data)
>>> total = distData.reduce(lambda a,b: a + b)
>>> print(total)
500
>>> import json
>>> myrdd1 = sc.wholeTextFiles("file:/usr/local/spark/json_files")
>>> myrdd2 = myrdd1.map(lambda a: json.loads(a[1]))
>>> for record in myrdd2.take(2):
...     print(record.get("firstName",None))
...
Fred
Barney
>>> 
```

FIGURE 19 – Utilisation de « wholeTextFiles »

3.7 Utilisation de « flatMap » et « distinct »

Pour utiliser flatMap et distinct, on suit les étapes suivantes :

1. Lecture du fichier texte dans une RDD :

```
» mydata = sc.textFile("file :/usr/local/spark/poeme.txt")
```
2. Application de la transformation flatMap avec une fonction lambda :

```
» mynewdata = mydata.flatMap(lambda line : line.split(' ')).distinct()
```
3. Affichage des résultats avec une boucle for :

```
» for line in mynewdata.collect() :
.. print(line)
```

```
hduser@labaalioubelkas-VirtualBox: /usr/local/spark
>>> mydata = sc.textFile("file:/usr/local/spark/poeme.txt")
>>> mynewdata = mydata.flatMap(lambda line: line.split(' ')).distinct()
>>> for line in mynewdata.collect():
...     print(line)
...
croyait
au
ny
pas
tous
deux
adoraient
prisonniere
des
montait
en
bas
sappelle
cette
sur
leur
que
lun
fut
les
du
quelle
vivra
verra
```

FIGURE 20 – Utilisation de « flatMap » et « distinct »

3.8 Utilisation de « subtract » et « zip »

En premier temps, nous allons créer deux RDD (rdd1 et rdd2) à partir de listes et appliquer ensuite la transformation subtract pour obtenir les éléments uniques de la première RDD (rdd1).

1. Création des RDDs à partir de listes :

```
» mydata = ["Chicago", "Boston", "Paris", "San Francisco", "Tokyo"]
» rdd1 = sc.parallelize(mydata)
» data = ["San Francisco", "Boston", "Amsterdam", "Mumbai", "Mc-Murdo Station"]
» rdd2 = sc.parallelize(data)
```
2. Application de la transformation subtract :

```
» newrdd = rdd1.subtract(rdd2)
```
3. Affichage des résultats avec une boucle for :

```
» for line in newrdd.collect() :
.. print(line)
```

```
>>> mydata = ["Chicago", "Boston", "Paris", "San Francisco", "Tokyo"]
>>> rdd1 = sc.parallelize(mydata)
>>> data = ["San Francisco", "Boston", "Amsterdam", "Mumbai", "McMurdo Station"]
>>> rdd2 = sc.parallelize(data)
>>> newrdd = rdd1.subtract(rdd2)
>>> for line in newrdd.collect():
...     print(line)
...
Tokyo
Chicago
Paris
>>> |
```

FIGURE 21 – Utilisation de « subtract »

Et en deuxième temps, nous allons utiliser la transformation « zip » :

1. Application de la transformation zip :
 » **ziprdd = rdd1.zip(rdd2)**
2. Affichage des résultats avec une boucle for :
 » **for line in ziprdd.collect() :**
 .. **print(line)**

```
>>>
>>> ziprdd = rdd1.zip(rdd2)
>>> for line in ziprdd.collect():
...     print(line)
...
('Chicago', 'San Francisco')
('Boston', 'Boston')
('Paris', 'Amsterdam')
('San Francisco', 'Mumbai')
('Tokyo', 'McMurdo Station')
>>> |
```

FIGURE 22 – Utilisation de « zip »

3.9 Utilisation de intersection et union

Dans cette partie, nous allons effectuer des opérations d'intersection et d'union entre deux RDD (rdd1 et rdd2).

1. Opération d'intersection (intersection) :
 » **interdd = rdd1.intersection(rdd2)**
 » **for line in interdd.collect() :**
 .. **print(line)**

```
>>> interdd = rdd1.intersection(rdd2)
>>> for line in interdd.collect():
...     print(line)
...
San Francisco
Boston
>>> |
```

FIGURE 23 – Utilisation de intersection

2. Opération d'union (union) :
 » **unionrdd = rdd1.union(rdd2)**
 » **for line in unionrdd.collect() :**
 .. **print(line)**

```
>>> unionrdd = rdd1.union(rdd2)
>>> for line in unionrdd.collect():
...     print(line)
...
Chicago
Boston
Paris
San Francisco
Tokyo
San Francisco
Boston
Amsterdam
Mumbai
McMurdo Station
```

FIGURE 24 – Utilisation de union

4 Connexion de Spark à une distribution de Hadoop

Spark peut utiliser les bibliothèques clientes Hadoop pour HDFS et YARN.

À partir de la version Spark 1.4, les packages de projet «Hadoop free» sont conçus pour nous permettre de connecter plus facilement un seul fichier binaire Spark à n'importe quelle version de Hadoop.

Pour utiliser ces packages , nous devons modifier **SPARK_DIST_CLASSPATH** afin d'inclure les fichiers jar relatifs à ces packages.

Pour ce faire, il est préférable d'ajouter une entrée dans **conf/spark-env.sh** :

```
hduser@labaalioubelkas-VirtualBox:/usr/local/spark$ cp conf/spark-env.sh.template spark-env.sh
hduser@labaalioubelkas-VirtualBox:/usr/local/spark/conf$ vim spark-env.sh
```

FIGURE 25 – Connexion de Spark à une distribution de Hadoop

```
t
### in conf/spark-env.sh ####
# If 'hadoop' binary is on your PATH
export SPARK_DIST_CLASSPATH=/usr/local/hadoop
# With explicit path to 'hadoop' binary
export SPARK_DIST_CLASSPATH=/usr/local/hadoop/bin
# Passing a Hadoop configuration directory
export SPARK_DIST_CLASSPATH=/usr/local/hadoop/etc/hadoop
C~
```

86,57

Bot

FIGURE 26 – Le contenu du fichier **conf/spark-env.sh**

5 Exécution du « Word Count » en utilisant le terminal scala et python

Une fois le noeud hadoop est bien configuré et spark bien connecté à hadoop. Nous pouvons déposer des données dans HDFS et les utiliser par spark.

5.1 Dépôt du poeme.txt dans le HDFS

Dans cette section, nous allons déposer le poeme.txt dans le HDFS :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/spark/conf$ cd /usr/local/hadoop/
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -put /home/hd
user/Documents/poeme.txt /
2023-10-24 19:23:09,077 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
put: '/home/hduser/Documents/poeme.txt': No such file or directory
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -put /home/hd
user/Documents/poeme.txt /
2023-10-24 19:30:53,514 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -ls /
2023-10-24 19:31:53,147 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 hduser supergroup      1669 2023-10-24 19:31 /poeme.txt
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$
```

FIGURE 27 – Dépôt du poeme.txt dans le HDFS

5.2 Exécution du code du « Word Count »

5.2.1 Exécution du code du « Word Count » en scala

Pour exécuter le traitement du « Word Count » en utilisant le terminal spark-shell, on tape le code scala suivant :

```
scala> val lines = sc.textFile("/poeme.txt")
lines: org.apache.spark.rdd.RDD[String] = /poeme.txt MapPartitionsRDD[1] at
File at <console>:23

scala> val words = lines.flatMap(_.split("\\s+"))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at
sole:23

scala> val wc = words.map(w => (w, 1)).reduceByKey(_ +_)
wc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey
console>:23

scala> wc.saveAsTextFile("file1.count")
scala>
```

FIGURE 28 – Exécution du code du « Word Count » en scala

Ensuite, nous suivant les étapes suivantes enfin d'afficher le résultat :

- Assurons-nous d'être dans le bon répertoire où Hadoop est installé :

\$ cd /usr/local/hadoop

- Copions le fichier 'file1.count' depuis HDFS vers le système de fichiers local :

\$ hadoop fs -get file1.count

- Listons les fichiers dans le répertoire courant de HDFS :

\$ bin/hdfs dfs -ls .

- Affichons le contenu du fichier 'part-00000' de 'file1.count' dans HDFS :

\$ bin/hdfs dfs -cat file1.count/part-00000

Et voilà, le résultat du programme Word Count :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -cat file1.co  
unt/part-00000  
2023-10-24 20:06:40,021 WARN util.NativeCodeLoader: Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
(jura,1)  
(ils,1)  
(bras,1)  
(des,5)  
(soldats,1)  
(celui,20)  
(leur,2)  
(bretagne,1)  
(cruelle,1)  
(fou,2)  
(quelle,1)  
(raisin,1)  
(les,4)  
(chapelle,1)  
(flute,1)  
(ruisselle,1)  
(combat,1)  
(mourra,1)  
(fideles,1)  
(au,11)  
(delicat,1)  
(ailes,1)  
(coule,2)  
(rouge,1)  
(verra,1)  
(songe,1)  
(murisse,1)  
(sappelle,1)  
(montait,1)
```

FIGURE 29 – Le résultat du programme Word Count

NB : Pour pouvoir écrire dans le hdfs, il faut donner tous les droits au répertoire : « /usr/local/hadoop_store/hdfs/datanode » pour cela on tape les commandes suivantes :
cd /usr/local/hadoop

```
# chmod 777 /usr/local/hadoop store/hdfs/datanode
```

5.2.2 Exécution du code du « Word Count » en pyspark

Pour exécuter le traitement du « Word Count » en utilisant le terminal Pyspark, on tape le code suivant pour lancer le terminal pyspark :

```
$ cd /usr/local/spark
```

```
$ ./bin/pyspark
```

Puis, on exécute le code suivant :

FIGURE 30 – Exécution du code du « Word Count » en pyspark

Enfin, nous pouvons afficher le résultat en utilisant hdfs :

```

hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ hadoop fs -get file2.count/
2023-10-24 20:57:31,750 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -ls
2023-10-24 20:57:44,043 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x - hduser supergroup 0 2023-10-24 20:01 file1.count
drwxr-xr-x - hduser supergroup 0 2023-10-24 20:51 file2.count
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -cat file2.count
//part-00000
2023-10-24 20:58:14,219 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
('croyait', 20)
('au', 11)
('ny', 10)
('pas', 11)
('tous', 3)
('deux', 6)
('adoraient', 1)
('prisonniere', 1)
('des', 5)
('montait', 1)
('en', 2)
('bas', 1)
('sappelle', 1)
('cette', 1)
('sur', 1)
('leur', 2)

```

FIGURE 31 – Le résultat du programme Word Count en pyspark

6 Exécution du « Word Count » en utilisant un script python

Créons d'abord un fichier « word_count.py » et insérons le code python suivant :

```

hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ 
import sys
from pyspark import SparkContext, SparkConf
if __name__ == "__main__":
    # create Spark context with necessary configuration
    sc = SparkContext("local", "PySpark Word Count Example")
    # read data from text file and split each line into words
    words = sc.textFile("/poeme.txt").flatMap(lambda line: line.split(" "))
    # count the occurrence of each word
    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a+b)
    # save the counts to output
    wordCounts.saveAsTextFile("file0.count")
~
```

FIGURE 32 – Le fichier « word_count.py »

Tapons ensuite la commande suivante pour exécuter le script python créé :
\$./bin/spark-submit word_count.py

```
examples$ hadoop fs -put poem.txt RELEASE word_count.py
hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ ./bin/spark-submit word_count.py
23/10/24 21:03:38 WARN Utils: Your hostname, labaallioubelkas-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
23/10/24 21:03:38 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
23/10/24 21:03:43 INFO SparkContext: Running Spark version 3.3.0
23/10/24 21:03:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
23/10/24 21:03:44 INFO ResourceUtils: =====
=====
23/10/24 21:03:44 INFO ResourceUtils: No custom resources configured for spark.driver.
23/10/24 21:03:44 INFO ResourceUtils: =====
=====
23/10/24 21:03:44 INFO SparkContext: Submitted application: PySpark Word Count Example
23/10/24 21:03:44 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
23/10/24 21:03:44 INFO ResourceProfile: Limiting resource is cpu
23/10/24 21:03:44 INFO ResourceProfileManager: Added ResourceProfile id: 0
23/10/24 21:03:45 INFO SecurityManager: Changing view acls to: hduser
```

FIGURE 33 – L'exécution du fichier « word_count.py »

Visualisons donc le résultat :

```
hduser@labaallioubelkas-VirtualBox:/usr/local$ cd hadoop
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ hadoop fs -get file0.count
2023-10-24 21:05:00,921 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -ls .
2023-10-24 21:05:15,757 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 3 items
drwxr-xr-x  - hduser supergroup          0 2023-10-24 21:04 file0.count
drwxr-xr-x  - hduser supergroup          0 2023-10-24 20:01 file1.count
drwxr-xr-x  - hduser supergroup          0 2023-10-24 20:51 file2.count
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ hadoop fs -cat file0.count/part-00000
2023-10-24 21:05:43,888 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
('celui', 20)
('qui', 25)
('croayait', 20)
('au', 11)
('ciel', 10)
('ny', 10)
('pas', 11)
('tous', 3)
('deux', 6)
('adoraient', 1)
('la', 8)
('belle', 1)
('prisonniere', 1)
('des', 5)
('soldats', 1)
('lequel', 5)
('montait', 1)
```

FIGURE 34 – Visualisation des résultats du fichier « word_count.py »

7 Exécution d'une application Spark Batch en Java

Dans cette section, nous allons créer une application Spark Batch en Java (un simple WordCount), le charger sur le nœud en local et le lancer.

7.1 Installation d'Apache Maven

Téléchargeons d'abord « apache-maven-3.5.0-bin.tar.gz » à partir du lien suivant : [Lien](#)
Déposons ce fichier dans notre répertoire /home/hduser/Documents/

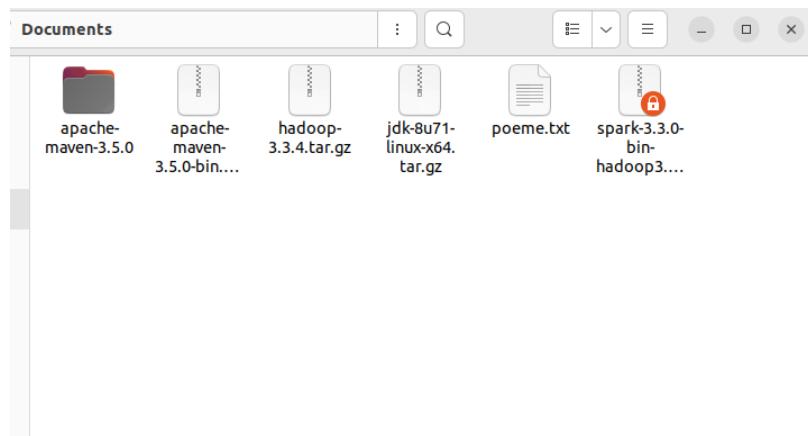


FIGURE 35 – Le fichier « apache-maven-3.5.0-bin.tar.gz »

Ensuite nous décompressons ce fichier en utilisant les commandes suivantes :

```
$ cd /home/hduser/Documents/
$ tar -zxvf apache-maven-3.5.0-bin.tar.gz
```

Et puis nous le plaçons dans /opt/ :

```
$ sudo -i
# mv /home/hduser/Documents/apache-maven- 3.5.0 /opt/
```

Pour mettre en place de manière permanente la variable d'environnement PATH pour tous les utilisateurs :

Ouvrons le fichier /etc/profile et modifions le PATH en ajoutant le chemin où se trouve le bin de maven dans export PATH :

```
export JAVA_HOME=/opt/java/jdk1.8.0_71/
export JRE_HOME=/opt/java/jdk1.8.0_71/jre
export PATH=$PATH:/opt/java/jdk1.8.0_71/bin:/opt/java/jdk1.8.0_71/jre/bin
export PATH=$PATH:/opt/java/jdk1.8.0_71/bin:/opt/java/jdk1.8.0_71/jre/bin:/opt/apache-maven-3.5.0/bin
```

FIGURE 36 – Le fichier /etc/profile

Après avoir enregistré le fichier profile, exécutons la commande source pour recharger le fichier dans la session hduser :

```
$ source /etc/profile
```

Testons la configuration de maven en tapant :**\$ mvn -v**

```
root@labaalioubelkas-VirtualBox:~# sudo vim /etc/profile
root@labaalioubelkas-VirtualBox:~# source /etc/pro
file
profile.d/ protocols
root@labaalioubelkas-VirtualBox:~# source /etc/profile
root@labaalioubelkas-VirtualBox:~# mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T20:39:06+01:00)
Maven home: /opt/apache-maven-3.5.0
Java version: 1.8.0_71, vendor: Oracle Corporation
Java home: /opt/java/jdk1.8.0_71/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.2.0-35-generic", arch: "amd64", family: "unix"
root@labaalioubelkas-VirtualBox:~#
```

FIGURE 37 – Teste de la configuration de maven

Créons après un projet Maven avec la commande suivante :

```
$ mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
-DarchetypeVersion=1.1
```

```
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.jar (6.2 kB at 82 kB/s)
Define value for property 'groupId': inpt.myapp
Define value for property 'artifactId': myapp
Define value for property 'version' 1.0-SNAPSHOT:
Define value for property 'package' inpt.myapp:
Confirm properties configuration:
groupId: inpt.myapp
artifactId: myapp
version: 1.0-SNAPSHOT
package: inpt.myapp
Y: : Y
```

FIGURE 38 – Les paramètres pour la création du projet maven

À la fin de la génération du projet maven, un message de « Build success » s'affiche :

```
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:1.1
[INFO] -----
[INFO] Parameter: basedir, Value: /home/hduser
[INFO] Parameter: package, Value: inpt.myapp
[INFO] Parameter: groupId, Value: inpt.myapp
[INFO] Parameter: artifactId, Value: myapp
[INFO] Parameter: packageName, Value: inpt.myapp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/hduser/myapp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:30 min
[INFO] Finished at: 2023-10-24T22:05:50+01:00
[INFO] Final Memory: 18M/283M
[INFO] -----
hduser@labaalloubelkas-VirtualBox:~$
```

FIGURE 39 – Message de « Build success »

Puis nous exécutons les commandes suivantes :

```
$ cd myapp/
$ mvn package
```

```
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/classworlds/classworlds-1.1-alpha-2.jar (38 kB at 104 kB/s)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.jar (58 kB at 161 kB/s)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.jar (60 kB at 134 kB/s)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar (184 kB at 231 kB/s)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar (226 kB at 318 kB/s)
Downloaded: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.jar (208 kB at 289 kB/s)
[INFO] Building jar: /home/hduser/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 44.729 s
[INFO] Finished at: 2023-10-24T22:08:07+01:00
[INFO] Final Memory: 19M/175M
[INFO] -----
hduser@labaalloubelkas-VirtualBox:~/myapp$
```

FIGURE 40 – mvn package

Pour voir toute l'arborescence du projet, nous pouvons installer la commande « tree » et ainsi visualiser toute l'arborescence du projet :

```
$ cd
$ sudo apt install tree
$ tree myapp/
```

```
hduser@labaalioubelkas-VirtualBox:~/myapp/
myapp/
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── inpt
    │           └── myapp
    │               └── App.java
    └── test
        └── java
            └── inpt
                └── myapp
                    └── AppTest.java
└── target
    ├── classes
    │   └── inpt
    │       └── myapp
    │           └── App.class
    ├── maven-archiver
    │   └── pom.properties
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   ├── default-compile
    │       │   │   └── createdFiles.lst
    │       │   │   └── inputFiles.lst
    │       │   └── testCompile
    │       │       ├── default-testCompile
    │       │       │   └── createdFiles.lst
    │       │       └── inputFiles.lst
    └── myapp-1.0-SNAPSHOT.jar
    └── surefire-reports
```

FIGURE 41 – L’arborescence du projet

Le projet maven créé correspond à un projet java « Hello World » sur lequel nous allons nous baser pour créer notre application java « WordCount ».

7.2 Reconfiguration du projet Maven

Rajoutons dans le fichier « pom.xml » les dépendances nécessaires, pour cela : Nous allons ajouter entre <properties>et </properties>, le contenu suivant :

```
hduser@labaalioubelkas-VirtualBox:~/myapp
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<name>myapp</name>
<url>http://maven.apache.org</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
    </dependency>
```

FIGURE 42 – Le contenu entre <properties>et </properties>

Et nous allons ajouter entre <dependencies> et </dependencies>, le contenu suivant :

```

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.13</artifactId>
        <version>3.2.0</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.22</version>
    </dependency>
    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>31.0.1-jre</version>
    </dependency>
</dependencies>
</project>
-- INSERT --

```

FIGURE 43 – Le contenu entre <dependencies> et </dependencies>

Dans le répertoire /home/hduser/myapp/src/main/java/DataEngineer/myapp nous allons :

- Renommer App.java en WordCountTask.java par :

```
$ cd /home/hduser/myapp/src/main/java/inpt/myapp
$ mv App.java WordCountTask.java
```

- Modifier le fichier WordCountTask.java en supprimant l'ancien contenu et copier le contenu du traitement « WordCountTask » ci-dessous dans « WordCountTask.java » :

```

package inpt.myapp;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import scala.Tuple2;

import java.util.Arrays;
import static com.google.common.base.Preconditions.checkNotNull;

public class WordCountTask {
    private static final Logger LOGGER = LoggerFactory.getLogger(WordCountTask.class);
    public static void main(String[] args) {
        checkArgument(args.length > 1, "Please provide the path of input file and output dir as parameters.");
        new WordCountTask().run(args[0], args[1]);
    }
    public void run(String inputFilePath, String outputDir) {
        SparkConf conf = new SparkConf()
            .setAppName(WordCountTask.class.getName());
        JavaSparkContext sc = new JavaSparkContext(conf);
        JavaRDD<String> textFile = sc.textFile(inputFilePath);
        JavaPairRDD<String, Integer> counts = textFile
            .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
            .mapToPair(word -> new Tuple2<>(word, 1))
            .reduceByKey((a, b) -> a + b);
        counts.saveAsTextFile(outputDir);
    }
}

```

FIGURE 44 – Le contenu du fichier WordCountTask.java

- Enregistrer le fichier WordCountTask.java, et lancer la commande suivante pour

compiler le code :

```
$ cd myapp/
$ mvn package
```

```
Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -- maven-jar-plugin:2.4:jar (default-jar) @ myapp --
[INFO] Building jar: /home/hduser/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:48 min
[INFO] Finished at: 2023-10-24T22:35:59+01:00
[INFO] Final Memory: 40M/286M
[INFO] -----
hduser@labaallioubelkas-VirtualBox:~/myapp$
```

FIGURE 45 – Compilation du code

7.3 Nettoyage et Formatage du nœud hadoop

Nous lançons les commandes shell sont utilisées pour effectuer certaines opérations de maintenance sur le stockage associé à Hadoop, principalement sur les répertoires de stockage des nœuds du système de fichiers distribué HDFS (Hadoop Distributed File System).

```
hduser@labaallioubelkas-VirtualBox:~/myapp$ cd /usr/local
hduser@labaallioubelkas-VirtualBox:/usr/local$ cd hadoop_store/
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ rm -rf *
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ mkdir -p /usr/local/
hadoop_store/hdfs/namenode
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ mkdir -p /usr/local/
hadoop_store/hdfs/datanode
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ chown -R hduser /usr
/local/hadoop_store/hdfs/namenode
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ chown -R hduser /usr
/local/hadoop_store/hdfs/datanode
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop_store$ cd /usr/local/hadoop
/etc/hadoop
```

FIGURE 46 – Nettoyage du nœud hadoop

Ensuite, nous relançons le formatage du noeud :

```
2023-10-24 22:40:04,774 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/hadoop_store/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 us
ing no compression
2023-10-24 22:40:05,332 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/hadoop_store/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 of size 4
01 bytes saved in 0 seconds .
2023-10-24 22:40:05,431 INFO namenode.NNStorageRetentionManager: Going to retain
1 images with txid >= 0
2023-10-24 22:40:05,490 INFO namenode.FSNamesystem: Stopping services started fo
r active state
2023-10-24 22:40:05,491 INFO namenode.FSNamesystem: Stopping services started fo
r standby state
2023-10-24 22:40:05,528 INFO namenode.FSImage: FSImageSaver clean checkpoint: tx
id=0 when meet shutdown.
2023-10-24 22:40:05,530 INFO namenode.NameNode: SHUTDOWN_MSG:
*****SHUTDOWN MSG! Shutting down NameNode at labaallioubelkas-VirtualBox/127.0.1.1*****
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop/etc/hadoop$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [labaallioubelkas-VirtualBox]
2023-10-24 22:40:42,744 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop/etc/hadoop$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop/etc/hadoop$
```

FIGURE 47 – Formatage du nœud hadoop

Finalement, nous testons si le noeud est disponible (live) :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop/etc/hadoop$ hdfs dfsadmin - report
2023-10-24 22:41:42,215 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Configured Capacity: 21312888832 (19.85 GB)
Present Capacity: 1900490752 (1.77 GB)
DFS Remaining: 1900466176 (1.77 GB)
DFS Used: 24576 (24 KB)
DFS Used%: 0.00%
Replicated Blocks:
    Under replicated blocks: 0
    Blocks with corrupt replicas: 0
    Missing blocks: 0
    Missing blocks (with replication factor 1): 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
Erasure Coded Block Groups:
    Low redundancy block groups: 0
    Block groups with corrupt internal blocks: 0
    Missing block groups: 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
-----
Live datanodes (1):
Name: 127.0.0.1:9866 (localhost)
Hostname: labaallioubelkas-VirtualBox
Decommission Status : Normal
Configured Capacity: 21312888832 (19.85 GB)
DFS Used: 24576 (24 KB)
```

FIGURE 48 – Test si le nœud est disponible

7.4 Dépôt du poeme.txt dans HDFS

Nous allons déposer le poeme.txt dans le HDFS comme précédemment :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -put /home/hduser/Documents/poeme.txt /
2023-10-24 22:43:44,511 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin java classes where applicable
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -ls /
2023-10-24 22:44:02,091 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin java classes where applicable
Found 1 items
-rw-r--r-- 1 hduser supergroup 1669 2023-10-24 22:43 /poeme.txt
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ █
```

FIGURE 49 – Dépôt du poeme.txt dans HDFS

Maintenant nous pouvons lancer la commande spark-submit pour exécuter le programme Word Count :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/spark$ spark-submit --class inpt.myapp.WordCountTask /home/hduser/myapp/target/myapp-1.0-SNAPSHOT.jar /poeme.txt /results
23/10/24 22:59:30 WARN Utils: Your hostname, labaallioubelkas-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
23/10/24 22:59:30 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
23/10/24 22:59:31 INFO SparkContext: Running Spark version 3.3.0
23/10/24 22:59:31 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
23/10/24 22:59:32 INFO ResourceUtils: ======
=====
23/10/24 22:59:32 INFO ResourceUtils: No custom resources configured for spark.driver.
23/10/24 22:59:32 INFO ResourceUtils: ======
=====
23/10/24 22:59:32 INFO SparkContext: Submitted application: inpt.myapp.WordCountTask
23/10/24 22:59:32 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpu -> name: cpus, amount: 1.0)
```

FIGURE 50 – spark-submit

Pour afficher le résultat, nous tapons les commandes :

```
hduser@labaallioubelkas-VirtualBox:/usr/local/hadoop$ bin/hdfs dfs -cat /results/part-00000
2023-10-24 23:02:11,194 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(jura,1)
(ils,1)
(bras,1)
(des,5)
(soldats,1)
(celui,20)
(leur,2)
(bretagne,1)
(cruelle,1)
(fou,2)
(quelle,1)
(raisin,1)
(les,4)
(chapelle,1)
(flute,1)
(ruisselle,1)
(combat,1)
(mourra,1)
(fideles,1)
(au,11)
(delicat,1)
(ailes,1)
(coule,2)
(rouge,1)
```

FIGURE 51 – Affichage des résultats

Conclusion

Au cours de ce TP, notre exploration d'Apache Spark en association avec Hadoop a été enrichissante. Nous avons acquis des compétences fondamentales en installant et en utilisant ces frameworks de traitement distribué. La mise en œuvre d'un programme de Word Count a été un excellent point de départ, nous permettant de comprendre les principes de base de la programmation distribuée avec Spark. Ce processus nous a permis de manipuler efficacement de grands ensembles de données tout en exploitant la puissance de traitement parallèle offerte par ces technologies. Ces connaissances constituent une base solide pour explorer davantage les fonctionnalités avancées d'Apache Spark et Hadoop dans des projets plus complexes et stimulants.