Bases de données NOSQL et Big Data

_TP2 : Le traitement Batch avec Hadoop Streaming _

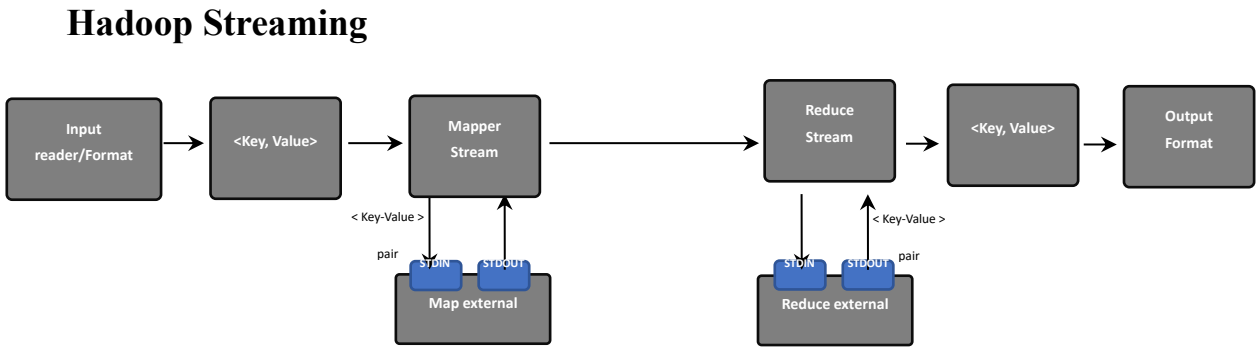
Objectifs du TP :

Ce TP vise à vous familiariser avec l'usage d'Hadoop Streaming pour la mise en oeuvre de tâches de traitement de données via MapReduce en utilisant Python. Vous apprendrez à développer des programmes de Map et de Reduce en Python, puis à exécuter des job MapReduce avecHadoop Straming.

Partie 1 : Comprendre Hadoop Streaming

Hadoop Streaming est un outil qui permet d'exécuter des tâches MapReduce avec des programmes personnalisés en utilisant des langages autre que Java (commen Python, Perl, Ruby, etc). C'est une approcje simple dans laquelle Hadoop invoque un exécutable en taant que partie d'une tâche plutôt que d'hébergerr un fichier JAR dans un environnement Java.

Il fonctionne en prenant des flux d'entrée et de sortie dequis et vers les tâches mappage (mapper) et réduction (reducer) sous forme de lignes textuelles. Les programmes personnalisés lisent dequis l'entré standard (STDIN) et écrivent vers la sortie standard (STDOUT). Comprendre comment STDIN et STDOUT sont utilisés n Python est essentiel pour utiliser l'API Hadoop Streaming.



Le programme qui ex&cute la fonction de mappage est appelé le «mapper». Le programme qui exécute la fonction de réduction est appelé le «réducteur». Les lignes de la sortie standard du processus de mappage sont converties en paires clé/valeur en les séparant sur le premier caractére de tabulation('\t', qui est le caractére de séparation par défaut mais peut être modifié). Les paires clé/valeur sont ensuite envoyées à l'entrée standard du réducteur pour un traitement ultérieur. Enfin, le réducteur écrit sur la sortie standard, qui est la sortie finale du programme.

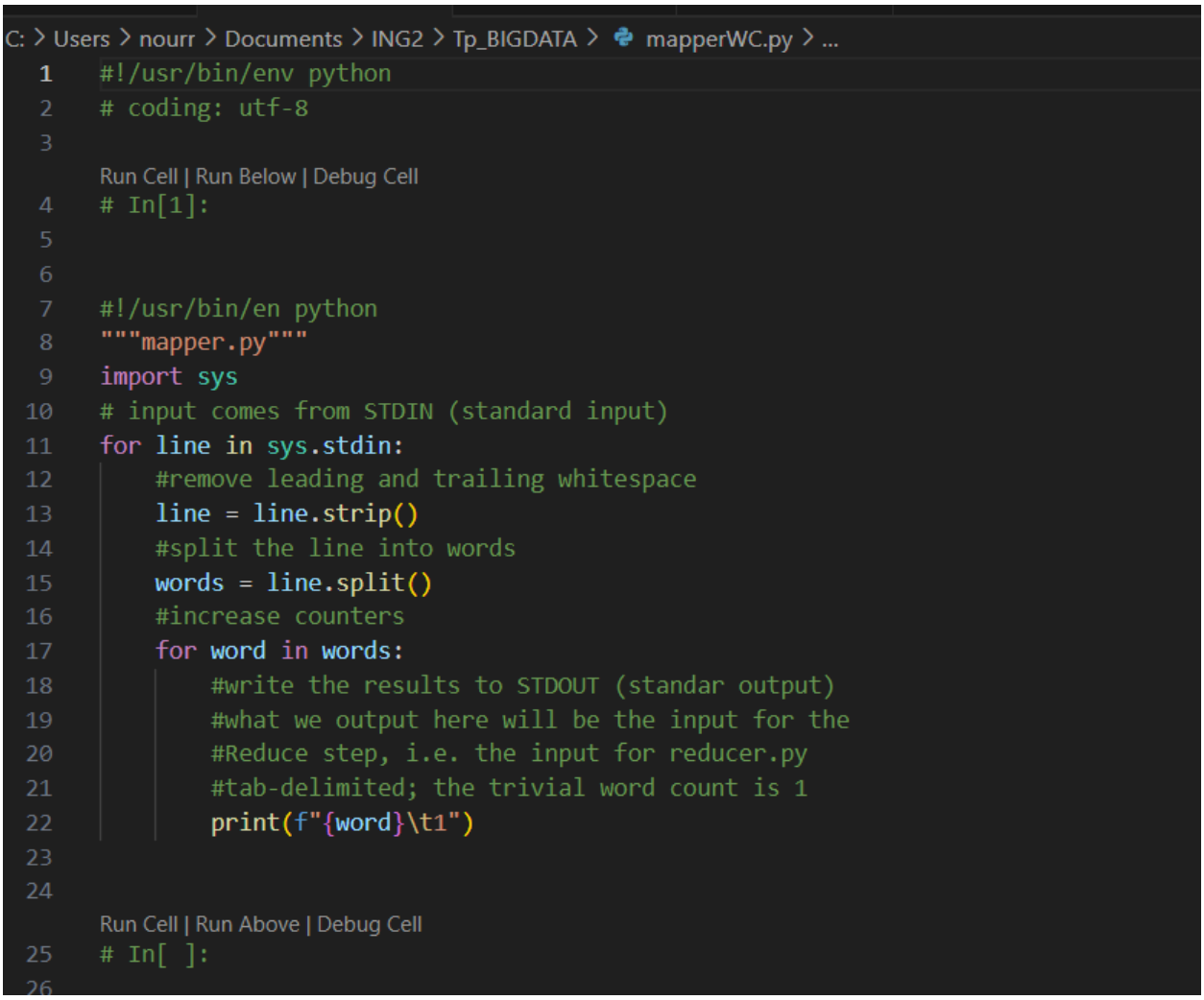
Partie 2 : Exécution d'un job avec Hadoop Streaming

Dans cet exemple, nous allons voir le code du **Word Count** pour compter combien de fois chaque mot apparaît dans un ensemble de fichiers texte. Le programme se compose d'une fonction de mappage (mapper) et d'une fonction de réduction (reducer). Initialement, nous allons tester le code localement sur de petits fichiers avant de l'utiliser dans une tâche de streaming MapReduce. Il est beaucoup plus facile de déboguer les erreurs de codage sur votre ordinateur personnel que sur le cluster.

Mapper : mapperWC.py

Le programme mapper lit chaque ligne d'entrée, découpe les mots et émet un couple clé-valeur pour chaque mot, ou la clé est le mot et la valeur est 1. Le programme renvoie (écrit) ces paires clé-valeur sur STDOUT. Un délimiteur (par défaut, '\t' est le délimiteur) doit être utiliser entre la clé et la valeur pour le traitement correct d'une paire clé-valeur lors de l'étape de Shuffle.

En utilisant votre environnement de développement Python, créez un fichier mapperWC.py. Il est préférable de saisir ce code manuellement plutôt que de le copier/coller pour mieux comprendre ce que fait le code.



Reducer: reducerWC.py

Le programme reducer agrège les paires clé-valeur en additionnant le nombre de fois que chaque mot apparaît. Il renvoie le résultat sur STDOUT. Créez un fichier reducerWC.py et saisir le code suivant:

C: > Users > nourr > Documents > ING2 > Tp_BIGDATA > reducerWC.py > ...

```
Run Cell | Run Below | Debug Cell
# In[2]:



#!/usr/bin/env python
import sys
#Initialize variables
current_word = None
current_count = 0
word = None
#Iterate through input lines, which are sorted by key (word) in ascending order
for line in sys.stdin:
    #remove leading and trailing whitespace
    line = line.strip()
    #split the key (word) and value (count) by a tab character
    word, count = line.split('\t',1)
    #convert the count to an integer
    try :
        count =  int(count)
    except ValueError:
        #if the conversion fails, skip this line
        continue
    #if the current word is the same as the previous word, increment the count
    if current_word == word:
        current_count +=count
    else:
        #if the word changes, print the result for the previous word
        if current_word :
            print('{}\t{}'.format(current_word,current_count))
        #Reset the variables for the new word .
        current_word = word
        current_count = count
#print the result for the last word
if current_word == word :
    print('{}\t{}'.format(current_word,current_count))
```

Execution du programme en local:

Il est judicieux de tester le code localemant sur de petits fichiers avant de l'utiliser dans une tâche de streaming MapReduce. Vous allez effectuer des tests locaux conformément à la méthode de pipeline typique de style Unix. Pour ce faire, vous allez combiner l'entrée standard (stdin) et la sortie standard (stdout) de votr script Python (mapperWC.py) en utilisant des commandes de redirection de flux. Vous devez avoir un fichier(ici «input.txt») pour tester vos codes.

Ouvrez PowerShell (ou un autre terminal si vous itilisez un systéme d'exploitation différent) et tapez la commande suivante : **cat input.txt | python mapperWC.py | python reducerWC.py**

```
PS C:\Users\nourr\Documents\ING2\Tp_BIGDATA> cat input.txt | python mapperWC.py | sort | python reducerWC.py
bonjour 2
cry     1
fatigu??e       1
I       1
je      1
la      1
suis    1
tellement       1
vie     1
will    1
PS C:\Users\nourr\Documents\ING2\Tp_BIGDATA>
```

Vous pouvez faire le même test sur votre container namenode.

Execution du programme en local:

Après avoir exécuté le code avec succés localement, l'étape suivante consiste à l'éxécuter sur le cluser avec Hadoop Streaming. Pour cela, il faut suivre les étapes suivantes :

Transférer les scripts dans le cluster Hadoop:

docker cp mapperWC.py namenode:/mapperWC.py
docker cp reducerWC.py namenode:/reducerWC.py

```
PS C:\Users\nourr\Documents\ING2\Tp_BIGDATA> docker cp mapperWC.py namenode:/mapperWC.py
Successfully copied 2.56kB to namenode:/mapperWC.py
PS C:\Users\nourr\Documents\ING2\Tp_BIGDATA> docker cp reducerWC.py namenode:/reducerWC.py
Successfully copied 3.07kB to namenode:/reducerWC.py
```

Entrer dans le container du namenode :

docker exec -it namenode bash

Créer des fichiers textes et les transferer dans HDFS :
En premier lien, vous allez tester votre code sur les petits fichiers suivants. Vous pouvez par la suite le tester sur les fichiers du dossier **Data** (créé dans le TP1).
mkdir input

```
echo "Hello World" > input/f1.txt
echo "Hello Docker" > input/f2.txt
echo "Hello Hadoop" > input/f3.txt
echo "Hello MapReduce" > input/f4.txt
hadoop fs -mkdir -p
hdfs dfs -put ./input/* input
```

```
PS C:\Users\nourr\Documents\ING2\Tp_BIGDATA> docker exec -it namenode bash
root@1a592075eb98:/# mkdir input
root@1a592075eb98:/# echo "hello word" > inpt/f1.txt
bash: inpt/f1.txt: No such file or directory
root@1a592075eb98:/# echo "hello word" > input/f1.txt
root@1a592075eb98:/# echo "hello word" > input/f2.txt
root@1a592075eb98:/# echo "hello word" > input/f3.txt
root@1a592075eb98:/# echo "hello word" > input/f4.txt
root@1a592075eb98:/# hadoop fs -mkdir -p input
root@1a592075eb98:/# hdfs dfs -put ./input/* input
2023-10-17 15:08:32,336 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-10-17 15:08:32,541 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-10-17 15:08:32,585 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-10-17 15:08:32,626 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```