TP MapReduce

Section 1: Introduction to Hadoop

Hadoop is a software framework for the distributed storage and processing of very large datasets standard hardware clusters. It is designed to scale from a few servers to thousands of machines, each local computing and storage capacity.

Section 2: The Hadoop File System (HDFS)

HDFS is Hadoop's distributed file system. It stores data on multiple machines to ensure redundancy and reliability. Files are split into blocks (by default, 128 MB or 256 MB) and each block is stored on several cluster nodes to ensure fault tolerance. The NameNode manages the namespace of the entire file system. It maintains the map of all files in the file system and which block belongs to which file. DataNodes store the actual data.

Section 3: MapReduce

MapReduce is a programming model for processing and generating large datasets with a parallel algorithm, distributed on a cluster. The process breaks down into two tasks: Map (processing each data element independently) and Reduce (consolidating the results to form a reduced output). Example: Calculating word redundancy in a text dataset.

Preparing the Docker and HDFS environment

1. **Pull the Docker image of Hadoop**:

   docker pull liliasfaxi/hadoop-cluster:latest

   This command downloads the Docker image containing a preconfigured installation of Hadoop and Spark.

2. **Create a Docker network**:

   docker network create --driver=bridge hadoop

   This creates an isolated network for your Hadoop containers, enabling nodes to communicate with each other.

3. **Start Hadoop containers:**
   Master:

   docker run -itd --net=hadoop -p 50070:50070 -p 8088:8088 - p 7077:7077 -p 16010:16010 --name hadoop-master -- hostname hadoop-master liliasfaxi/hadoop-cluster:latest

   Slave1:

docker run -itd -p 8040:8042 --net=hadoop --name hadoop-slave1 --hostname hadoop-slave1 liliasfaxi/hadoop-cluster:latest

   Slave2:

docker run -itd -p 8041:8042 --net=hadoop --name hadoop-slave2 --hostname hadoop-slave2 liliasfaxi/hadoop-cluster:latest

4. **Access the Master container**:
   This opens a Bash terminal in the Hadoop Master container.

   docker exec -it hadoop-master bash

5. **Enable services on hadoop and activate Hadoop with :**

   start-all.sh
   /root/start-hadoop.sh

Data preparation

1. **Create the input file** (already done with **input.txt** supplied).
2. **Copy input.txt file to hadoop-master (open CMD in working directory)**

   docker cp input.txt IDMASTER:/root/input.txt

3. **Place file in HDFS**:

   Creating a folder test in HDFS:

   hdfs dfs -mkdir /test

   Check that the folder has been created:

   hdfs dfs -ls /

   Put input.txt in the /test folder in HDFS:

   hdfs dfs -put input.txt /test

   Check the contents of the /test **folder:**

   hdfs dfs -ls /test

Map script (mapp.py)

Each line of **mapp.py** is explained below:

```
import sys    # Imports the sys module, which access to certain
variables used or maintained by the Python interpreter.

for line in sys.stdin:     # For each line read standard input (STDIN).
     line= line.strip()      # Removes white spaces at the beginning and
end of the line.
     words= line.split()      # Splits the line into words, using the space as a
separator.
     for word in words:   # For each word in the word list.
```

```
        print('%s\t%s' % (word, 1))     # Displays the word followed a tab
and the number 1, representing an initial count of 1 for this word.
```

Script Reduce (reducerr.py)

Each line of **reducerr.py** is explained below:

```
import sys   # Imports the sys module.

current_word=  None  #  Sets  current  word  to  None.
current_count= 0 # Sets current counter to 0. word= None #
Sets word to None.

for line in sys.stdin:      # For each line read from standard input. line=
    line.strip()            # Removes leading and trailing white space
of the line.
    word, count= line.split('\t', 1)    # Splits the line into word
and count, based on the first tab.
    try:
        count= int(count)      # Tries to convert count to integer. except
    ValueError:
        continue  # If conversion fails, ignore this line.

    if current_word== word:    # If the current word is the same as the word
lu.
        current_count+= count    # Increments the counter for this word.
    else:
        if current_word:  # If this is not the first time a word has been
treated.
            print('%s\t%s' % (current_word, current_count))  # Displays
the word and its total count.
        current_count= count    # Resets the counter with the count of the
new word.
        current_word= word    # Resets the current word with the new one
word.

if current_word== word:   # Displays the last word and its countdown
if necessary.
    print('%s\t%s' % (current_word, current_count))
```

Copy input.txt file to hadoop-master (open CMD in working directory)

docker cp mapp.py IDMASTER:/root/mapp.py

docker cp reducer.py IDMASTER:/root/reducer.py

Use the docker ps command to retrieve the IDMASTER

MapReduce Job execution

hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar -input /test/input.txt -output /output/results - file /root/mapp.py -mapper "python3 mapp.py" -file /root/reducerr.py -reducer "python3 reducerr.py" Note:

If an error occurs, run the following command to see the code errors in mapp.py

cat input.txt| python3 mapp.py

if error in mapp.py, check reducerr.py

cat input.txt| python3 mapp.py| python3 reducerr.py

once errors have been corrected, run the map reduce job again

Checking results

To see the contents of the output folder and the results :

hdfs dfs -ls /output/

hdfs dfs -ls /output/results

hdfs dfs -cat /output/results/part-00000