ASSIGNMENT NO: 6

Title: Integrate Python and Hadoop and perform the following operations on forest fire dataset

a. Data analysis using the Map Reduce in PyHadoop

b. Data mining in Hive

Software and Hardware Requirement: PC with Ubuntu

Theory:

Hadoop Streaming is a feature that comes with Hadoop and allows users or developers to use various different languages for writing MapReduce programs like Python, C++, Ruby, etc. It supports all the languages that can read from standard input and write to standard output. We will be implementing Python with Hadoop Streaming and will observe how it works. We will implement the word count problem in python to understand Hadoop Streaming. We will be creating mapper.py and reducer.py to perform map and reduce tasks.

Let’s create one file which contains multiple words that we can count.

Step 1: Create a file with the name word_count_data.txt and add some data to it.

```
cd Documents/                          # to change the directory to /Documents

touch word_count_data.txt                  # touch is used to create an empty file

nano word_count_data.txt                  # nano is a command line editor to edit the file

cat word_count_data.txt                  # cat is used to see the content of the file
```

Step 2: Create a mapper.py file that implements the mapper logic. It will read the data from STDIN and will split the lines into words, and will generate an output of each word with its individual count.

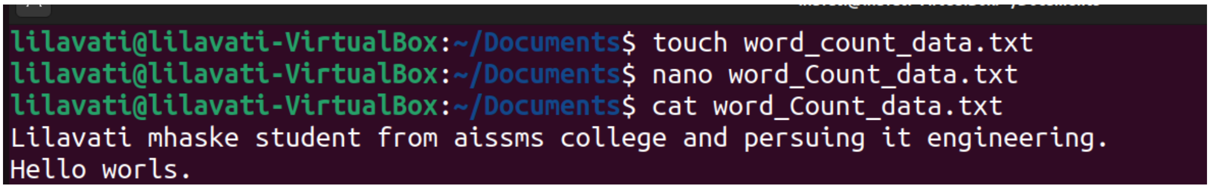
```
cd Documents/                          # to change the directory to /Documents

touch mapper.py                  # touch is used to create an empty file

cat mapper.py                  # cat is used to see the content of the file
```

mapper.py :

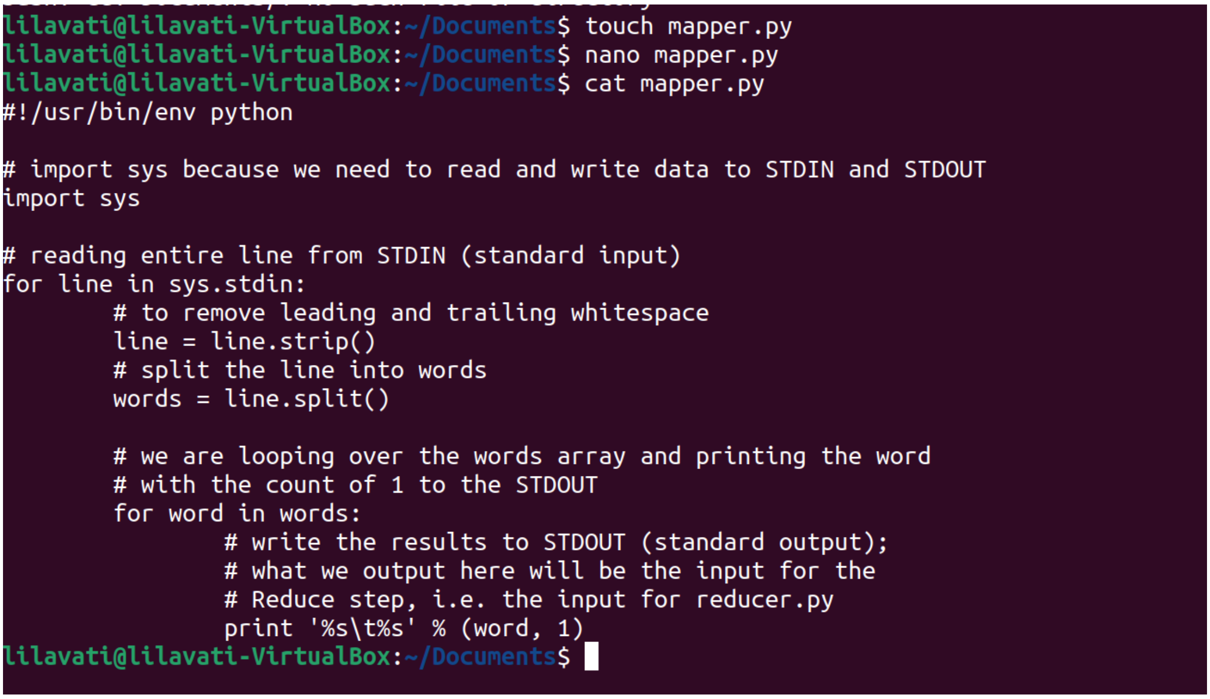
```
#!/usr/bin/python2.7

import sys
```

```
#Word Count Example

# input comes from standard input STDIN

for line in sys.stdin:

    line = line.strip() #remove leading and trailing whitespaces

    words = line.split() #split the line into words and returns as a list

    for word in words:

       # print '%s\t%s' % (word,1) #Emit the word

        print '%s  %s' % (word,1)
```

Here in the above program #! is known as shebang and used for interpreting the script. The file will be run using the command we are specifying.



```
lilavati@lilavati-VirtualBox:~/Documents$ touch word_count_data.txt
lilavati@lilavati-VirtualBox:~/Documents$ nano word_Count_data.txt
lilavati@lilavati-VirtualBox:~/Documents$ cat word_Count_data.txt
Lilavati mhaske student from aissms college and persuing it engineering.
Hello worls.
```



```
lilavati@lilavati-VirtualBox:~/Documents$ touch mapper.py
lilavati@lilavati-VirtualBox:~/Documents$ nano mapper.py
lilavati@lilavati-VirtualBox:~/Documents$ cat mapper.py
#!/usr/bin/env python

# import sys because we need to read and write data to STDIN and STDOUT
import sys

# reading entire line from STDIN (standard input)
for line in sys.stdin:
        # to remove leading and trailing whitespace
        line = line.strip()
        # split the line into words
        words = line.split()

        # we are looping over the words array and printing the word
        # with the count of 1 to the STDOUT
        for word in words:
                # write the results to STDOUT (standard output);
                # what we output here will be the input for the
                # Reduce step, i.e. the input for reducer.py
                print '%s\t%s' % (word, 1)
lilavati@lilavati-VirtualBox:~/Documents$ 
```

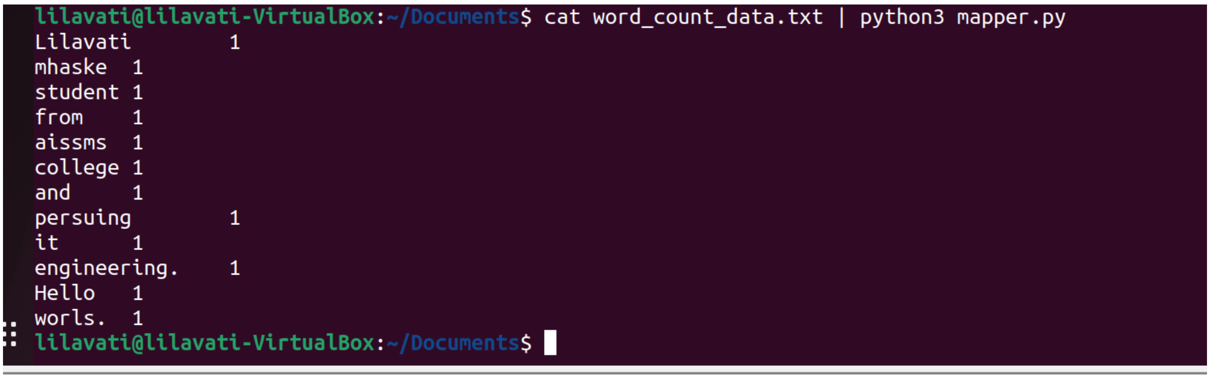
Let’s test our mapper.py locally that it is working fine or not.

cat <text_data_file> | python <mapper_code_python_file>

Command(in my case)

```
cat word_count_data.txt | python mapper.py
```

The output of the mapper is shown below.



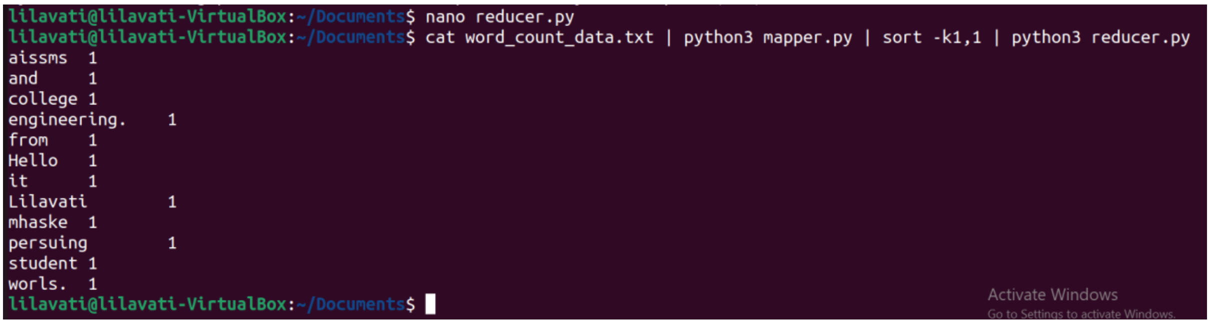
```
lilavati@lilavati-VirtualBox:~/Documents$ cat word_count_data.txt | python3 mapper.py
Lilavati        1
mhaske  1
student 1
from    1
aissms  1
college 1
and     1
persuing        1
it      1
engineering.    1
Hello   1
worls.  1
lilavati@lilavati-VirtualBox:~/Documents$ 
```

Step 3: Create a reducer.py file that implements the reducer logic. It will read the output of mapper.py from STDIN(standard input) and will aggregate the occurrence of each word and will write the final output to STDOUT.

```
cd Documents/                        # to change the directory to /Documents

touch reducer.py                # touch is used to create an empty file
```

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


from operator import itemgetter

import sys

current_word = None

current_count = 0

word = None


# read the entire line from STDIN

for line in sys.stdin:

        # remove leading and trailing whitespace

        line = line.strip()

        # splitting the data on the basis of tab we have provided in mapper.py
        word, count = line.split('\t', 1)
        # convert count (currently a string) to int
        try:
                count = int(count)
        except ValueError:
                # count was not a number, so silently
                # ignore/discard this line
                continue

        # this IF-switch only works because Hadoop sorts map output
        # by key (here: word) before it is passed to the reducer
        if current_word == word:
                current_count += count
        else:
                if current_word:
                        # write result to STDOUT
                        print '%s\t%s' % (current_word, current_count)
                current_count = count
                current_word = word

# do not forget to output the last word if needed!
if current_word == word:
        print '%s\t%s' % (current_word, current_count)
```

Now let’s check our reducer code reducer.py with mapper.py is it working properly or not with the help of the below command.

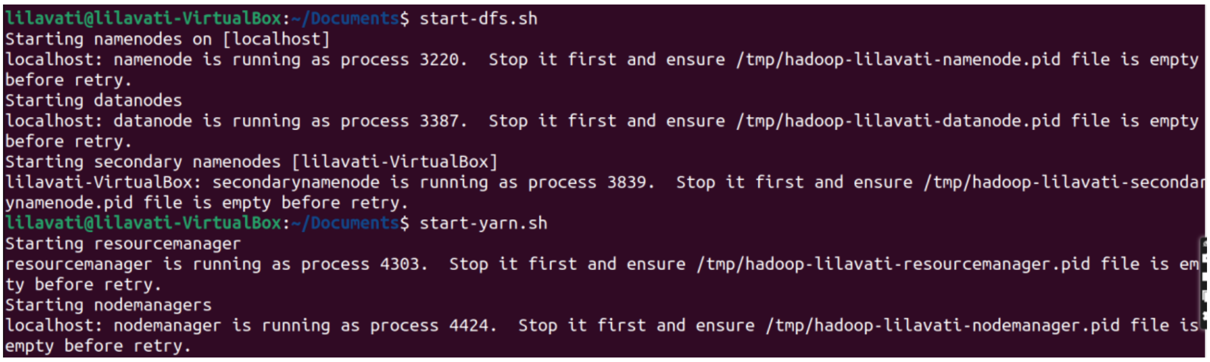


```
lilavati@lilavati-VirtualBox:~/Documents$ nano reducer.py
lilavati@lilavati-VirtualBox:~/Documents$ cat word_count_data.txt | python3 mapper.py | sort -k1,1 | python3 reducer.py
aissms  1
and     1
college 1
engineering.    1
from    1
Hello   1
it      1
Lilavati        1
mhaske  1
persuing        1
student 1
worls.  1
lilavati@lilavati-VirtualBox:~/Documents$ 
```

We can see that our reducer is also working fine in our local system.

Step 4: Now let’s start all our Hadoop daemons with the below command.

- start-dfs.sh
- start-yarn.sh



```
lilavati@lilavati-VirtualBox:~/Documents$ start-dfs.sh
Starting namenodes on [localhost]
localhost: namenode is running as process 3220.  Stop it first and ensure /tmp/hadoop-lilavati-namenode.pid file is empty
before retry.
Starting datanodes
localhost: datanode is running as process 3387.  Stop it first and ensure /tmp/hadoop-lilavati-datanode.pid file is empty
before retry.
Starting secondary namenodes [lilavati-VirtualBox]
lilavati-VirtualBox: secondarynamenode is running as process 3839.  Stop it first and ensure /tmp/hadoop-lilavati-secondar
ynamenode.pid file is empty before retry.
lilavati@lilavati-VirtualBox:~/Documents$ start-yarn.sh
Starting resourcemanager
resourcemanager is running as process 4303.  Stop it first and ensure /tmp/hadoop-lilavati-resourcemanager.pid file is em
ty before retry.
Starting nodemanagers
localhost: nodemanager is running as process 4424.  Stop it first and ensure /tmp/hadoop-lilavati-nodemanager.pid file is
empty before retry.
```

Now make a directory word_count_in_python in our HDFS in the root directory that will store our word_count_data.txt file with the below command.

hdfs dfs -mkdir /word_count_in_python

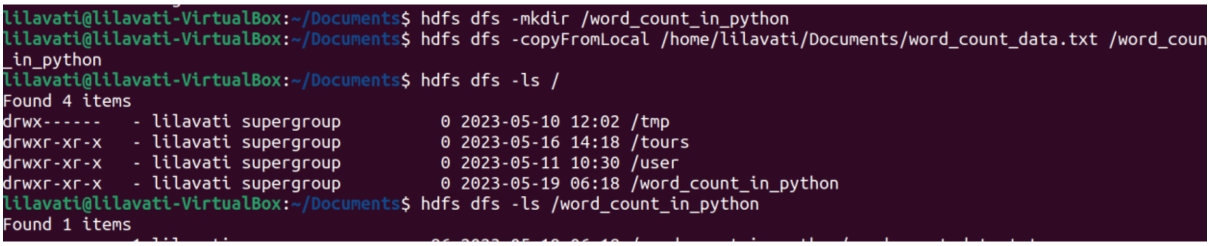
Copy word_count_data.txt to this folder in our HDFS with help of copyFromLocal command.

Syntax to copy a file from your local file system to the HDFS is given below:

hdfs dfs -copyFromLocal /path 1 /path 2 .... /path n /destination

Actual command(in my case)

hdfs dfs -copyFromLocal /home/lilavati/Documents/word_count_data.txt /word_count_in_python

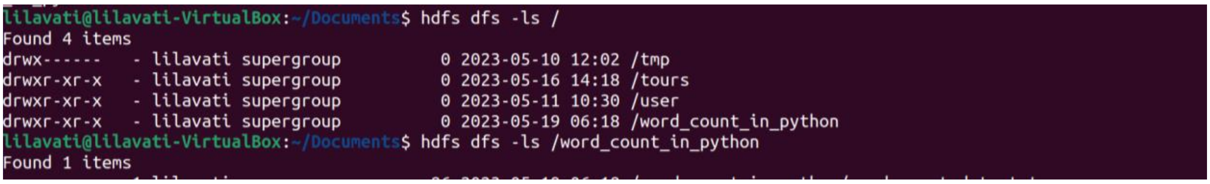


```
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -mkdir /word_count_in_python
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -copyFromLocal /home/lilavati/Documents/word_count_data.txt /word_coun
_in_python
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -ls /
Found 4 items
drwx------   - lilavati supergroup          0 2023-05-10 12:02 /tmp
drwxr-xr-x   - lilavati supergroup          0 2023-05-16 14:18 /tours
drwxr-xr-x   - lilavati supergroup          0 2023-05-11 10:30 /user
drwxr-xr-x   - lilavati supergroup          0 2023-05-19 06:18 /word_count_in_python
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -ls /word_count_in_python
Found 1 items
```

Now our data file has been sent to HDFS successfully. we can check whether it sends or not by using the below command or by manually visiting our HDFS.

hdfs dfs -ls /   # list down content of the root directory

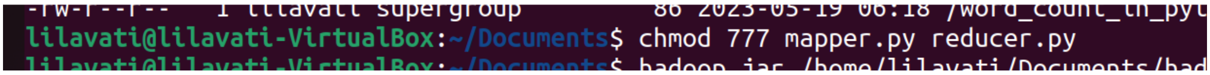
hdfs dfs -ls /word_count_in_python   # list down content of /word_count_in_python directory



```
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -ls /
Found 4 items
drwx------   - lilavati supergroup          0 2023-05-10 12:02 /tmp
drwxr-xr-x   - lilavati supergroup          0 2023-05-16 14:18 /tours
drwxr-xr-x   - lilavati supergroup          0 2023-05-11 10:30 /user
drwxr-xr-x   - lilavati supergroup          0 2023-05-19 06:18 /word_count_in_python
lilavati@lilavati-VirtualBox:~/Documents$ hdfs dfs -ls /word_count_in_python
Found 1 items
```

Let’s give executable permission to our **mapper.py** and **reducer.py** with the help of below command.
cd Documents/

chmod 777 mapper.py reducer.py   # changing the permission to read, write, execute for user, group and others

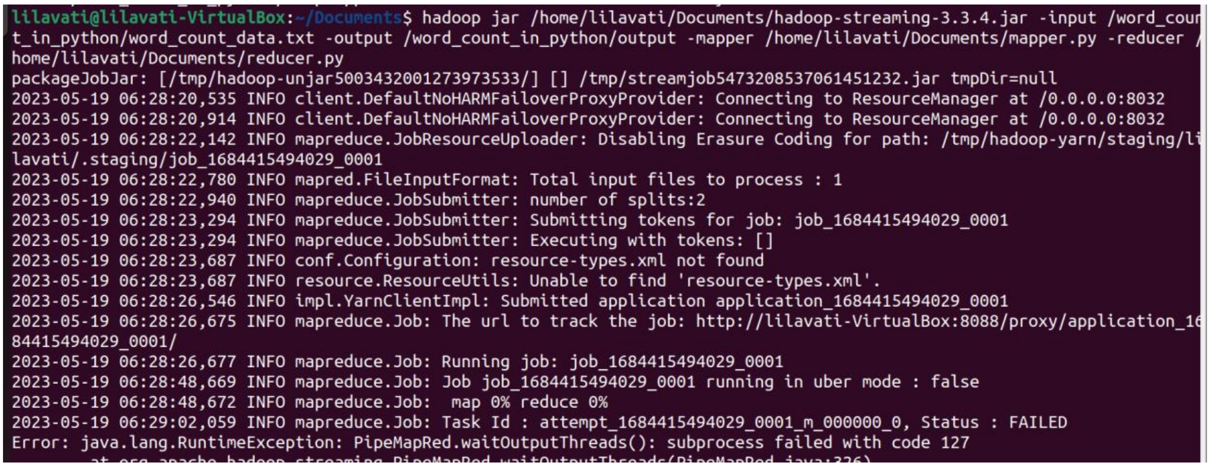


```
-rw-r--r--   1 lilavati supergroup         86 2023-05-19 06:18 /word_count_in_pyt
lilavati@lilavati-VirtualBox:~/Documents$ chmod 777 mapper.py reducer.py
lilavati@lilavati-VirtualBox:~/Documents$ hadoop jar /home/lilavati/Documents/had
```

Step 5: **Now download the latest** hadoop-streaming jar **file from** this Link**.** Then place, this Hadoop,-streaming jar file to a place from you can easily access it. In my case, I am placing it to ***/Documents*** folder where **mapper.py** and **reducer.py** file is present.
Now let’s run our python files with the help of the Hadoop streaming utility as shown below.

hadoop jar /home/dikshant/Documents/hadoop-streaming-2.7.3.jar \

```
> -input /word_count_in_python/word_count_data.txt \

> -output /word_count_in_python/output \

> -mapper /home/dikshant/Documents/mapper.py \

> -reducer /home/dikshant/Documents/reducer.py
```



```
lilavati@lilavati-VirtualBox:~/Documents$ hadoop jar /home/lilavati/Documents/hadoop-streaming-3.3.4.jar -input /word_coun
t_in_python/word_count_data.txt -output /word_count_in_python/output -mapper /home/lilavati/Documents/mapper.py -reducer /
home/lilavati/Documents/reducer.py
packageJobJar: [/tmp/hadoop-unjar5003432001273973533/] [] /tmp/streamjob5473208537061451232.jar tmpDir=null
2023-05-19 06:28:20,535 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-19 06:28:20,914 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-05-19 06:28:22,142 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/li
lavati/.staging/job_1684415494029_0001
2023-05-19 06:28:22,780 INFO mapred.FileInputFormat: Total input files to process : 1
2023-05-19 06:28:22,940 INFO mapreduce.JobSubmitter: number of splits:2
2023-05-19 06:28:23,294 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1684415494029_0001
2023-05-19 06:28:23,294 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-05-19 06:28:23,687 INFO conf.Configuration: resource-types.xml not found
2023-05-19 06:28:23,687 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-05-19 06:28:26,546 INFO impl.YarnClientImpl: Submitted application application_1684415494029_0001
2023-05-19 06:28:26,675 INFO mapreduce.Job: The url to track the job: http://lilavati-VirtualBox:8088/proxy/application_16
84415494029_0001/
2023-05-19 06:28:26,677 INFO mapreduce.Job: Running job: job_1684415494029_0001
2023-05-19 06:28:48,669 INFO mapreduce.Job: Job job_1684415494029_0001 running in uber mode : false
2023-05-19 06:28:48,672 INFO mapreduce.Job:  map 0% reduce 0%
2023-05-19 06:29:02,059 INFO mapreduce.Job: Task Id : attempt_1684415494029_0001_m_000000_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 127
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
```

In the above command in -output, we will specify the location in HDFS where we want our output to be stored. So let’s check our output in output file at location /word_count_in_python/output/part-00000 in my case.

We can check results by manually vising the location in HDFS or with the help of cat command as shown below.

```
hdfs dfs -cat /word_count_in_python/output/part-00000
```