

## **Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.**

### **Aim:**

To run a basic Word Count MapReduce program.

### **Procedure:**

#### **Step 1: Create Data File:**

Create a file named "word\_count\_data.txt" and populate it with text data that you wish to analyse.

Login with your hadoop user.

```
nano word_count.txt
```

Output: Type the below content in word\_count.txt

#### **Step 2: Mapper Logic - mapper.py:**

Create a file named "mapper.py" to implement the logic for the mapper. The mapper will read input data from STDIN, split lines into words, and output each word with its count.

```
nano mapper.py
```

```
# Copy and paste the mapper.py code
```

```
#!/usr/bin/env python3
```

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

```
#!/usr/bin/python3
```

```
import sys
```

```
for line in sys.stdin:
```

```
    line = line.strip() # remove leading and trailing whitespace
```

```
    words = line.split() # split the line into words
```

```
    for word in words:
```

```
print( '%s\t%s' % (word, 1))
```

.

### Step 3: Reducer Logic - reducer.py:

Create a file named "reducer.py" to implement the logic for the reducer. The reducer will aggregate the occurrences of each word and generate the final output.

```
nano reducer.py
```

```
# Copy and paste the reducer.py code
```

#### reducer.py

```
#!/usr/bin/python3
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 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))
```

### Step 4: Prepare Hadoop Environment:

Start the Hadoop daemons and create a directory in HDFS to store your data.

```
start-all.sh
```

```
hdfsdfs -mkdir /word_count_in_python
```

```
hdfsdfs -copyFromLocal /path/to/word_count.txt/word_count_in_python
```

### Step 6: Make Python Files Executable:

Give executable permissions to your mapper.py and reducer.py files.

```
chmod 777 mapper.py reducer.py
```

### Step 7: Run Word Count using Hadoop Streaming:

Download the latest hadoop-streaming jar file and place it in a location you can easily access.

Then run the Word Count program using Hadoop Streaming.

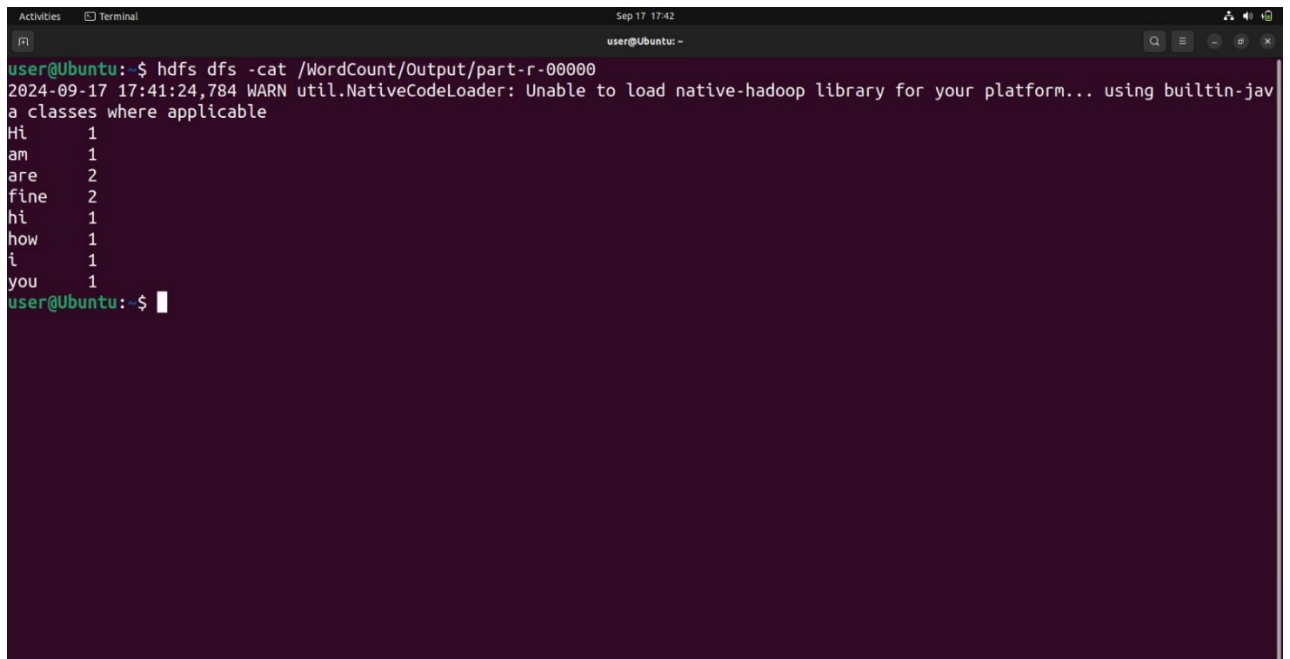
```
hadoop jar /path/to/hadoop-streaming-3.3.6.jar \  
-input /word_count_in_python/word_count_data.txt \  
-output /word_count_in_python/new_output \  
-mapper /path/to/mapper.py \  
-reducer /path/to/reducer.py
```

### Step 8: Check Output:

Check the output of the Word Count program in the specified HDFS output directory

```
.hdfs dfs -cat /word_count_in_python/new_output/part-00000
```

### OUTPUT:

A terminal window titled 'Terminal' with a dark background. The prompt is 'user@Ubuntu:~'. The command entered is 'hdfs dfs -cat /WordCount/Output/part-r-00000'. The output shows a warning message: '2024-09-17 17:41:24,784 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-jav a classes where applicable'. Below this, the word counts are listed: 'Hi 1', 'am 1', 'are 2', 'fine 2', 'hi 1', 'how 1', 'i 1', 'you 1'. The prompt 'user@Ubuntu:~\$' is visible at the bottom.

```
user@Ubuntu:~$ hdfs dfs -cat /WordCount/Output/part-r-00000  
2024-09-17 17:41:24,784 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-jav  
a classes where applicable  
Hi 1  
am 1  
are 2  
fine 2  
hi 1  
how 1  
i 1  
you 1  
user@Ubuntu:~$
```

### Result:

Thus, the program for basic Word Count Map Reduce has been executed successfully.