Exp.No: 2

Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm

AIM:

To run a basic Word Count MapReduce program.

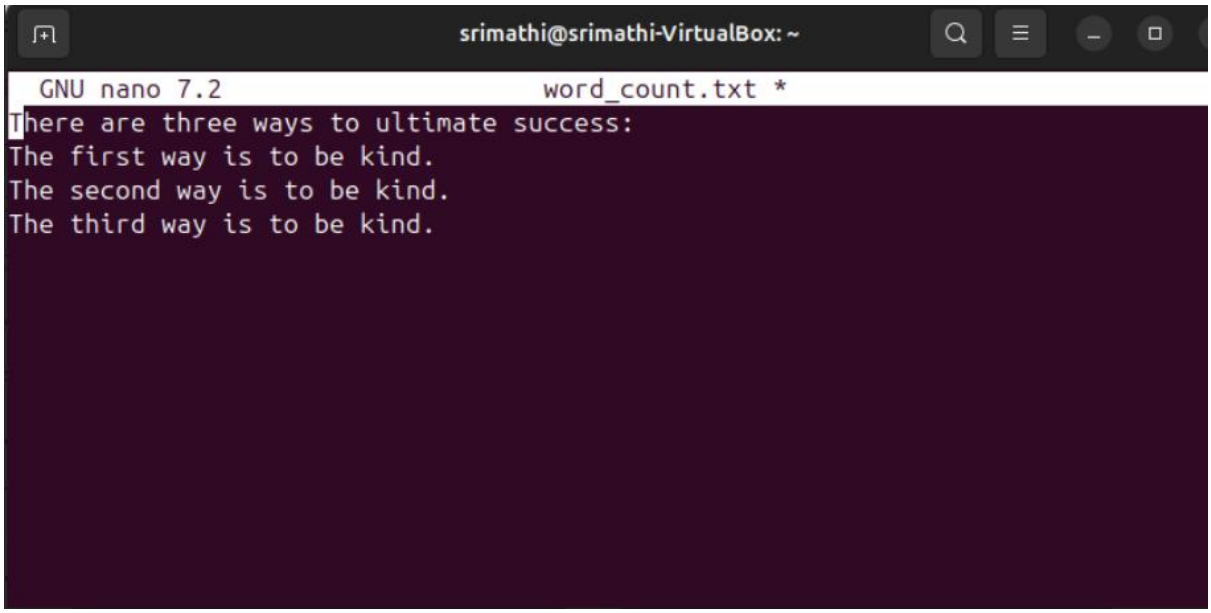
Procedure:

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
```

```
srimathi@srimathi-VirtualBox:~$ hadoop jar /home/srimathi/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.0.jar   -input /word_count_in_pyth
on/word_count.txt   -output /word_count_in_python/output   -mapper /home/srimathi/mapper.py   -reducer /home/srimathi/reducer.py
2024-09-18 10:27:55,954 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2024-09-18 10:27:56,142 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2024-09-18 10:27:56,144 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2024-09-18 10:27:56,170 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2024-09-18 10:27:56,784 INFO mapred.FileInputFormat: Total input files to process : 1
2024-09-18 10:27:56,915 INFO mapreduce.JobSubmitter: number of splits:1
2024-09-18 10:27:57,166 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1170730732_0001
2024-09-18 10:27:57,166 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-09-18 10:27:57,405 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2024-09-18 10:27:57,409 INFO mapreduce.Job: Running job: job_local1170730732_0001
2024-09-18 10:27:57,417 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2024-09-18 10:27:57,432 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
2024-09-18 10:27:57,473 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-09-18 10:27:57,473 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, igno
re cleanup failures: false
2024-09-18 10:27:57,547 INFO mapred.LocalJobRunner: Waiting for map tasks
2024-09-18 10:27:57,561 INFO mapred.LocalJobRunner: Starting task: attempt_local1170730732_0001_m_000000_0
2024-09-18 10:27:57,610 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-09-18 10:27:57,613 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, igno
re cleanup failures: false
2024-09-18 10:27:57,640 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
2024-09-18 10:27:57,659 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/word_count_in_python/word_count.txt:0+133
2024-09-18 10:27:57,685 INFO mapred.MapTask: numReduceTasks: 1
2024-09-18 10:27:57,771 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2024-09-18 10:27:57,771 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2024-09-18 10:27:57,771 INFO mapred.MapTask: soft limit at 83886080
2024-09-18 10:27:57,771 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2024-09-18 10:27:57,771 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2024-09-18 10:27:57,779 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
```

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
```

```
srimathi@srimathi-VirtualBox:~$ hdfs dfs -cat /word_count_in_python/output/part-00000
The     3
There   1
are     1
be      3
first   1
is      3
kind.   3
second  1
success:        1
third   1
three   1
to      4
ultimate        1
way     3
ways    1
srimathi@srimathi-VirtualBox:~$
```

Result:

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