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.

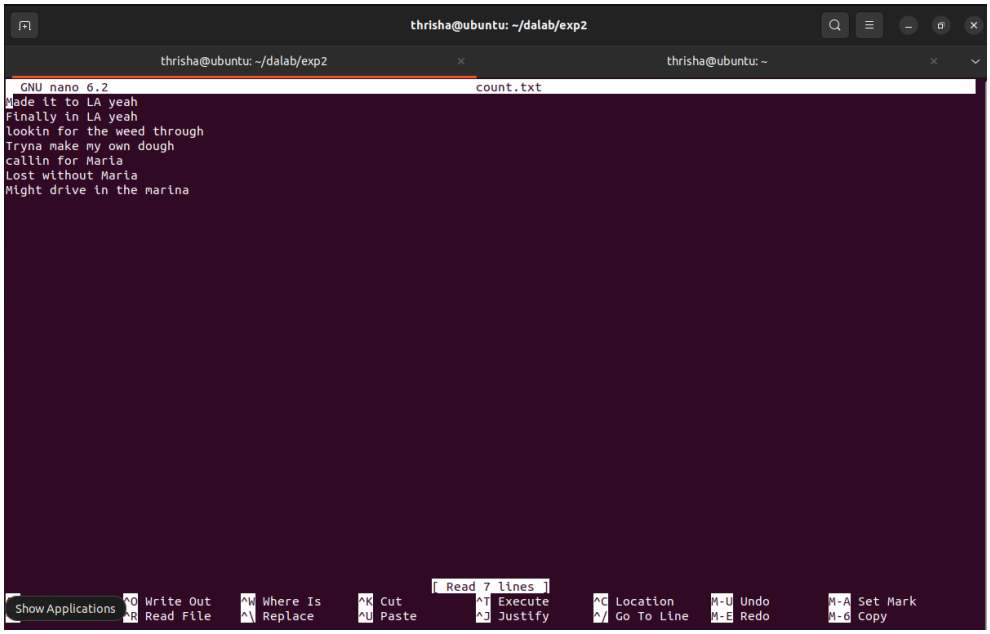
Procedure:

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
```

```
thrisha@ubuntu:~$ hadoop jar $HADOOP_STREAMING     -input /word_count_in_python/count.txt     -output /word_count_in_python/output
   -mapper ~/dalab/exp2/mapper.py     -reducer ~/dalab/exp2/reducer.py
packageJobJar: [/tmp/hadoop-unjar6415991681819250656/] [] /tmp/streamjob7622807675503802489.jar tmpDir=null
2024-09-03 10:55:46,747 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-09-03 10:55:47,106 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-09-03 10:55:47,561 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/thrisha/.sta
ging/job_1725336950875_0008
2024-09-03 10:55:48,148 INFO mapred.FileInputFormat: Total input files to process : 1
2024-09-03 10:55:48,715 INFO mapreduce.JobSubmitter: number of splits:2
2024-09-03 10:55:48,983 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1725336950875_0008
2024-09-03 10:55:48,984 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-09-03 10:55:49,341 INFO conf.Configuration: found resource resource-types.xml at file:/home/thrisha/hadoop-3.4.0/etc/hadoop/res
ource-types.xml
2024-09-03 10:55:49,481 INFO impl.YarnClientImpl: Submitted application application_1725336950875_0008
2024-09-03 10:55:49,564 INFO mapreduce.Job: The url to track the job: http://ubuntu.myguest.virtualbox.org:8088/proxy/application_17
25336950875_0008/
2024-09-03 10:55:49,567 INFO mapreduce.Job: Running job: job_1725336950875_0008
2024-09-03 10:56:02,989 INFO mapreduce.Job: Job job_1725336950875_0008 running in uber mode : false
2024-09-03 10:56:03,167 INFO mapreduce.Job:  map 0% reduce 0%
2024-09-03 10:56:15,872 INFO mapreduce.Job:  map 100% reduce 0%
2024-09-03 10:56:26,166 INFO mapreduce.Job:  map 100% reduce 100%
2024-09-03 10:56:27,246 INFO mapreduce.Job: Job job_1725336950875_0008 completed successfully
2024-09-03 10:56:28,515 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=278
                FILE: Number of bytes written=934851
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=439
                HDFS: Number of bytes written=175
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
```

```
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=19380
                Total time spent by all reduces in occupied slots (ms)=7763
                Total time spent by all map tasks (ms)=19380
                Total time spent by all reduce tasks (ms)=7763
                Total vcore-milliseconds taken by all map tasks=19380
                Total vcore-milliseconds taken by all reduce tasks=7763
                Total megabyte-milliseconds taken by all map tasks=19845120
                Total megabyte-milliseconds taken by all reduce tasks=7949312
        Map-Reduce Framework
                Map input records=8
                Map output records=30
                Map output bytes=212
                Map output materialized bytes=284
                Input split bytes=208
                Combine input records=0
                Combine output records=0
                Reduce input groups=24
                Reduce shuffle bytes=284
                Reduce input records=30
                Reduce output records=24
                Spilled Records=60
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=369
                CPU time spent (ms)=3450
                Physical memory (bytes) snapshot=860766208
                Virtual memory (bytes) snapshot=7603970048
                Total committed heap usage (bytes)=635437056
                Peak Map Physical memory (bytes)=322080768
                Peak Map Virtual memory (bytes)=2533076992
                Peak Reduce Physical memory (bytes)=216637440
                Peak Reduce Virtual memory (bytes)=2538213376
```

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

```
                Peak Reduce Virtual memory (bytes)=2538213376
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=231
        File Output Format Counters
                Bytes Written=175
2024-09-03 10:56:28,520 INFO streaming.StreamJob: Output directory: /word_count_in_python/output
thrisha@ubuntu:~$ hdfs dfs -cat /word_count_in_python/output/part-* | more
Finally 1
LA      2
Lost    1
Made    1
Maria   2
Might   1
Tryna   1
callin  1
dough   1
drive   1
for     2
in      2
it      1
lookin  1
make    1
marina  1
my      1
own     1
the     2
through 1
to      1
weed    1
without 1
yeah    2
thrisha@ubuntu:~$ S
```

Result:

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