EXP 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

```
mapper.py  reducer.py  s.txt
hayagriv@fedora:~/exp2$ cat s.txt
hello world
my name is haya
mathesh is ugly
bye bye bye
deadpool
hayagriv@fedora:~/exp2$
```

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
```

```
2024-10-19 07:44:44,221 INFO mapred.FileInputFormat: Total input files to process : 1
2024-10-19 07:44:45,102 INFO mapreduce.JobSubmitter: number of splits:2
2024-10-19 07:44:45,254 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1729322327179_0001
2024-10-19 07:44:45,254 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-19 07:44:45,367 INFO conf.Configuration: resource-types.xml not found
2024-10-19 07:44:45,367 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-10-19 07:44:45,566 INFO impl.YarnClientImpl: Submitted application application_1729322327179_0001
2024-10-19 07:44:45,609 INFO mapreduce.Job: The url to track the job: http://fedora:8088/proxy/application_1729322327179_0001/
2024-10-19 07:44:45,610 INFO mapreduce.Job: Running job: job_1729322327179_0001
2024-10-19 07:44:52,736 INFO mapreduce.Job: Job job_1729322327179_0001 running in uber mode : false
2024-10-19 07:44:52,737 INFO mapreduce.Job:  map 0% reduce 0%
2024-10-19 07:44:56,833 INFO mapreduce.Job:  map 100% reduce 0%
2024-10-19 07:45:01,862 INFO mapreduce.Job:  map 100% reduce 100%
2024-10-19 07:45:02,874 INFO mapreduce.Job: Job job_1729322327179_0001 completed successfully
2024-10-19 07:45:02,952 INFO mapreduce.Job: Counters: 54
```

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
```

```
Generic options supported are:
-conf <configuration file>        specify an application configuration file
-D <property=value>               define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides 'fs.defaultFS' property from configurations.
-jt <local|resourcemanager:port>  specify a ResourceManager
-files <file1,...>                specify a comma-separated list of files to be copied to the map reduce cluster
-libjars <jar1,...>               specify a comma-separated list of jar files to be included in the classpath
-archives <archive1,...>          specify a comma-separated list of archives to be unarchived on the compute machines

The general command line syntax is:
command [genericOptions] [commandOptions]

hayagriv@fedora:~$ hadoop fs -cat /exp2/output/part-00000
bye     3
deadpool        1
haya    1
hello   1
is      2
mathesh 1
my      1
name    1
ugly    1
world   1
hayagriv@fedora:~$
```