

Map Reduce program to process a weather dataset.

Aim:

To implement MapReduce program to process a weather dataset

Procedure:

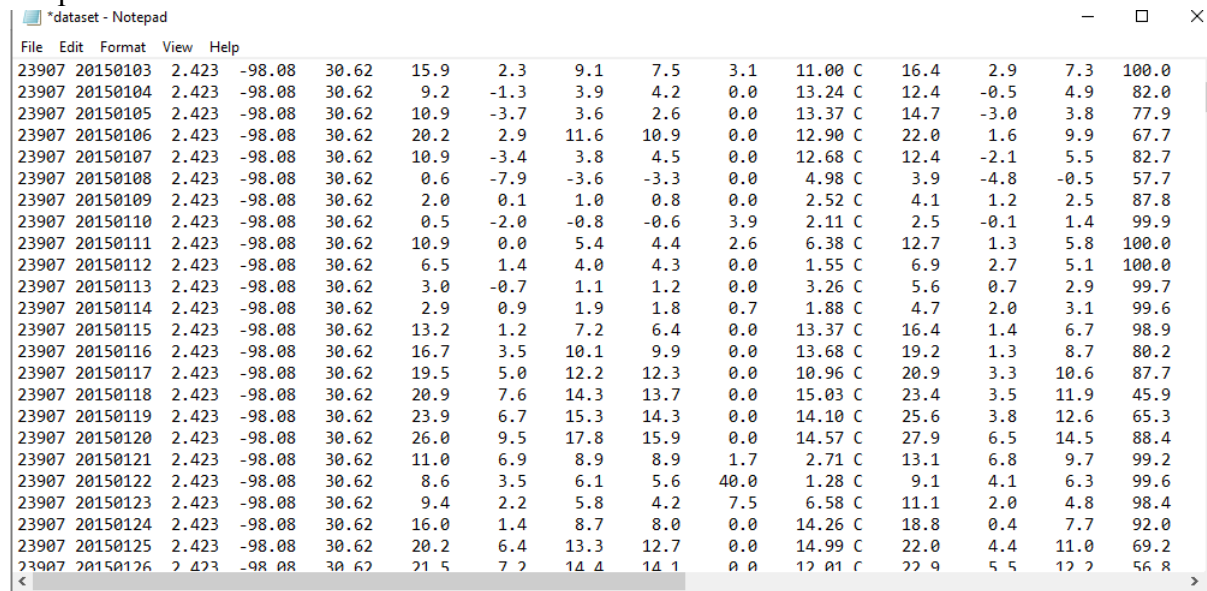
Step 1: Create Data File:

Create a file named "word_count_data.txt" and populate it with text data that you wish to analyse.

Login with your hadoop user.

Download the dataset (weather data)

Output:



| File | Edit | Format | View | Help |
|----------------|-------|--------|-------|------|
| 23907 20150103 | 2.423 | -98.08 | 30.62 | 15.9 |
| 23907 20150104 | 2.423 | -98.08 | 30.62 | 9.2 |
| 23907 20150105 | 2.423 | -98.08 | 30.62 | 10.9 |
| 23907 20150106 | 2.423 | -98.08 | 30.62 | 20.2 |
| 23907 20150107 | 2.423 | -98.08 | 30.62 | 10.9 |
| 23907 20150108 | 2.423 | -98.08 | 30.62 | 0.6 |
| 23907 20150109 | 2.423 | -98.08 | 30.62 | 2.0 |
| 23907 20150110 | 2.423 | -98.08 | 30.62 | 0.5 |
| 23907 20150111 | 2.423 | -98.08 | 30.62 | 10.9 |
| 23907 20150112 | 2.423 | -98.08 | 30.62 | 6.5 |
| 23907 20150113 | 2.423 | -98.08 | 30.62 | 3.0 |
| 23907 20150114 | 2.423 | -98.08 | 30.62 | 2.9 |
| 23907 20150115 | 2.423 | -98.08 | 30.62 | 13.2 |
| 23907 20150116 | 2.423 | -98.08 | 30.62 | 16.7 |
| 23907 20150117 | 2.423 | -98.08 | 30.62 | 19.5 |
| 23907 20150118 | 2.423 | -98.08 | 30.62 | 20.9 |
| 23907 20150119 | 2.423 | -98.08 | 30.62 | 23.9 |
| 23907 20150120 | 2.423 | -98.08 | 30.62 | 26.0 |
| 23907 20150121 | 2.423 | -98.08 | 30.62 | 11.0 |
| 23907 20150122 | 2.423 | -98.08 | 30.62 | 8.6 |
| 23907 20150123 | 2.423 | -98.08 | 30.62 | 9.4 |
| 23907 20150124 | 2.423 | -98.08 | 30.62 | 16.0 |
| 23907 20150125 | 2.423 | -98.08 | 30.62 | 20.2 |
| 23907 20150126 | 2.423 | -98.08 | 30.62 | 21.5 |

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 python
```

```
import sys
```

```
# input comes from STDIN (standard input)
```

```
# the mapper will get daily max temperature and group it by month. so output will be
(month,dailymax_temperature)
```

```

for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    #See the README hosted on the weather website which help us understand how each
position represents a column
    month = line[10:12]
    daily_max = line[38:45]
    daily_max = daily_max.strip()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be go through the shuffle process and then
        # be the input for the Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; month and daily max temperature as output
        print('%s\t%s' % (month, daily_max))
    .

```

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
```

reducer.py

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

```

from operator import itemgetter
import sys

#reducer will get the input from stdin which will be a collection of key, value(Key=month ,
value= daily max temperature)
#reducer logic: will get all the daily max temperature for a month and find max temperature
for the month
#shuffle will ensure that key are sorted(month)
current_month = None
current_max = 0
month = None

# input comes from STDIN
for line in sys.stdin:

```

```

# remove leading and trailing whitespace
line = line.strip()
# parse the input we got from mapper.py
month, daily_max = line.split('\t', 1)

# convert daily_max (currently a string) to float
try:
    daily_max = float(daily_max)
except ValueError:
    # daily_max was not a number, so silently
    # ignore/discard this line
    continue

# this IF-switch only works because Hadoop shuffle process sorts map output
# by key (here: month) before it is passed to the reducer
if current_month == month:
    if daily_max > current_max:
        current_max = daily_max
else:
    if current_month:
        # write result to STDOUT
        print ('%s\t%s' % (current_month, current_max))
    current_max = daily_max
    current_month = month

# output of the last month
if current_month == month:
    print ('%s\t%s' % (current_month, current_max))

```

Step 4: Prepare Hadoop Environment:

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

```
start-all.sh
```

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 the program using Hadoop Streaming:

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

Then run the program using

Hadoop Streaming. `hadoop fs`

`-mkdir -p /weatherdata`

`hadoop fs -copyFromLocal /home/sx/Downloads/`

`dataset.txt /weatherdata hdfs dfs -ls /weatherdata`

`hadoop jar /home/sx/hadoop-3.2.3/share/hadoop/tools/lib/hadoop-streaming-3.2.3.jar \`

`-input /weatherdata/dataset.txt \`

`-output /weatherdata/output \`

`-file "/home/sx/Downloads/mapper.py" \`

`-mapper "python3 mapper.py" \`

`-file "/home/sx/Downloads/reducer.py" \`

`-reducer "python3 reducer.py"`

`hdfs dfs -text /weatherdata/output/* > /home/sx/Downloads/outputfile.txt`

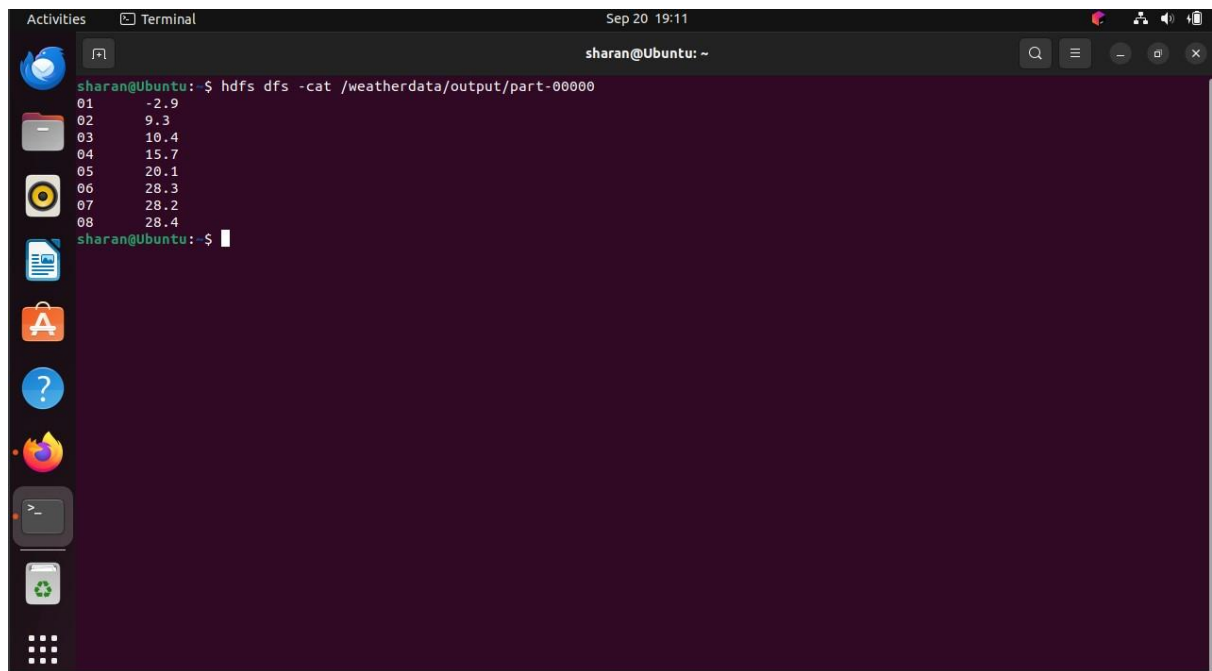
j
b
d
k
b

o
u
O
u
t
p
u
t:

Step 8: Check Output:

Check the output of the program in the specified HDFS output directory.

OUTPUT:



The image shows a terminal window on an Ubuntu system. The command `hdfs dfs -cat /weatherdata/output/part-00000` has been executed, resulting in the following output:

| | |
|----|------|
| 01 | -2.9 |
| 02 | 9.3 |
| 03 | 10.4 |
| 04 | 15.7 |
| 05 | 20.1 |
| 06 | 28.3 |
| 07 | 28.2 |
| 08 | 28.4 |

The terminal prompt is `sharan@Ubuntu: $`.

Result:

Thus, the program for weather dataset using Map Reduce has been executed successfully.