Assignment – 1

for

CLOUD COMPUTING TECHNOLOGY
UEC634

Submitted By: Vishav Singla
Roll number: 102115204
Group: 3NC8

Submitted to

Dr. Mayank Agarwal

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB
INDIA

January – May
2024

MAPREDUCE

Software/Tools used : python3, mrjob, vscode

Theory:

MapReduce is a programming paradigm for big data processing, where data is partitioned into distributed chunks and processed by a series of transformations.

The MapReduce programming paradigm processes data in 2 operations: map() and then reduce(). map() is a user-defined function that maps each data record in the data collection. reduce() groups the output of map() with another user-defined function.

2. MapReduce Pipeline

MapReduce works on (key, value) pairs by performing the below steps:

INPUT: list of key-value pairs (k1, v1)

MAP: (k1, v1) >> [list of (k2, v2)]

SHUFFLE: combine (k2, v2) >> (k2, [list of v2])

REDUCE: (k2, [list of v2]) >> (k3, v3)

OUTPUT: list of (k3, v3)

Mrjob is a library that allows you to write Python programs that run on Hadoop. With mrjob, you can test your code locally without installing Hadoop or run it on a cluster of your choice. “If you don’t want to be a Hadoop expert but need the computing power of MapReduce, mrjob might be just the thing for you.”

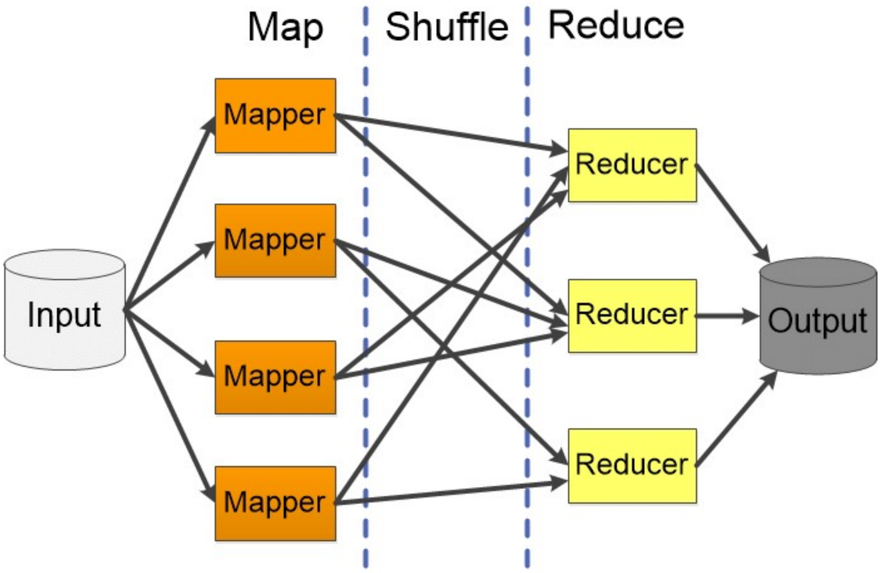

Fig 1.1 Map reduce flow chart

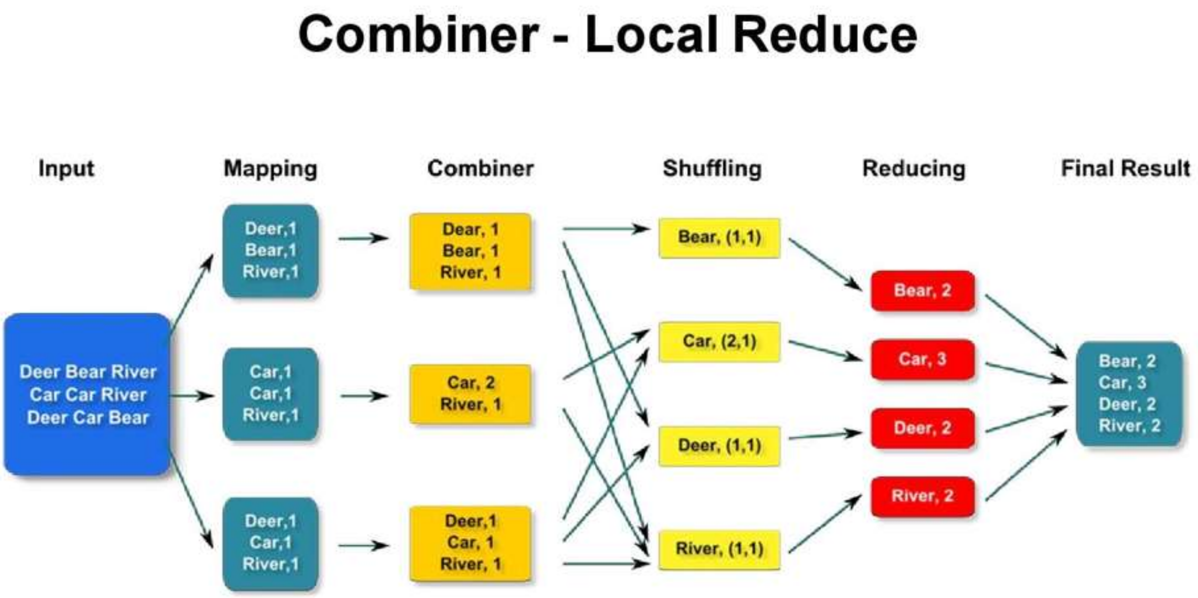

Fig 1.2 Example of Mapreduce in tabular form

Python CODE:

```
from mrjob.job import MRJob

class WordCount(MRJob):

    def mapper(self, _, line):
        # Split the line into words
        words = line.split()

        # Emit each word with a count of 1
        for word in words:
            yield word.lower(), 1

    def reducer(self, key, values):
        # Sum up the counts for each word
        yield key, sum(values)

WordCount.run()
```

Input.txt File

```
Hello world
This is a sample input file
It contains multiple lines
Each line has some words
multiple some words
```

Output for mapper function:.

```
"hello" 1
"world"         1
"this"  1
"is"    1
"a"     1
"sample"        1
"input"1
"file"  1
"it"    1
"contains"      1
"multiple"      1
"lines" 1
"each" 1
"line"  1
"has"   1
"some"          1
"words"         1
"multiple"      1
"some"          1
"words"         1
```

Output for mapper and reducer functions:

```
"a"       1
"contains"        1
"each"  1
"file"    1
"has"    1
"hello"  1
"input" 1
"is"      1
"it"       1
"line"   1
"lines"  1
"multiple"        2
"sample"          1
"some" 2
"this"   1
"words"           2
"world" 1
```

Conclusion:

This experiment explored MapReduce, a powerful tool for handling big data. We crunched a text file using mrjob, a handy Python library, and counted every word's appearances. The MapReduce pipeline neatly split the file, counted words, and finally presented the results. This confirmed MapReduce's potential for analyzing massive datasets, opening doors for tasks like website traffic analysis or recommending products based on customer preferences