Experiment 6

Shashwat Shah
60004220126
TY BTECH COMPSB

Aim: Implement matrix multiplication and word freqency count using MapReduce

Theory: MapReduce is one of the main features of Hadoop. It is a type of programming paradigm used in hadoop. The major features of map redue is to perform the distributed processing in parallel in hadoop cluster which makes the hadoop working so fast

Mapreduce has 2 functions
Map () - Takes input from disk as key, value pair processes them and produces another set of intermediate <key, values> pair as output.
Reduce ()- Also takes input as key value pairs. Output of mapping function.

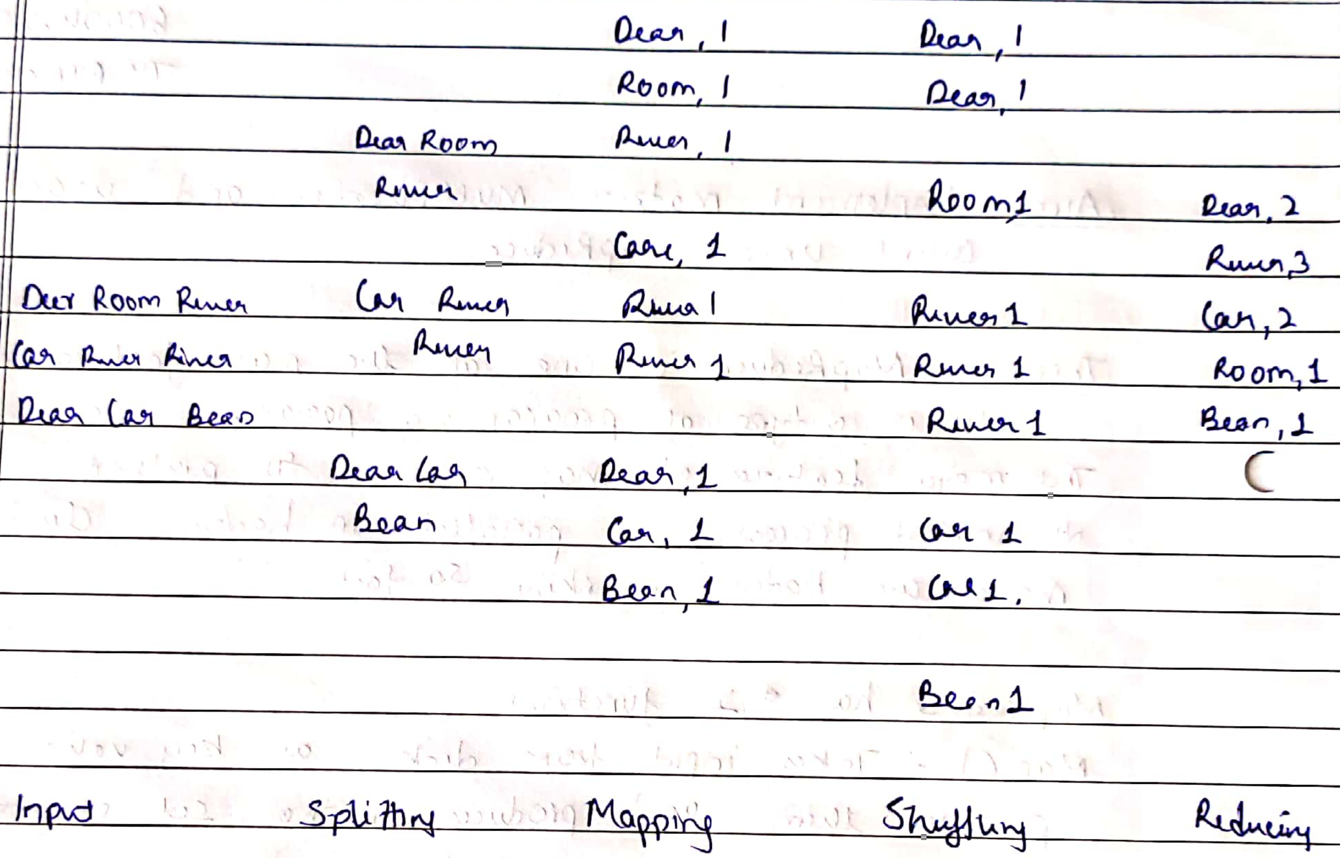
→ Matrix multiplication

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Result (R) $= i \times k = 2 \times 2$

Formula for mapping
Matrix A $(k, v) = (i, k)\ (A_{ij}, A_{ij})$ for all k
Matrix B $(k, v) = (i, k)\ (A, j, A_{ij})$ for all i

word count



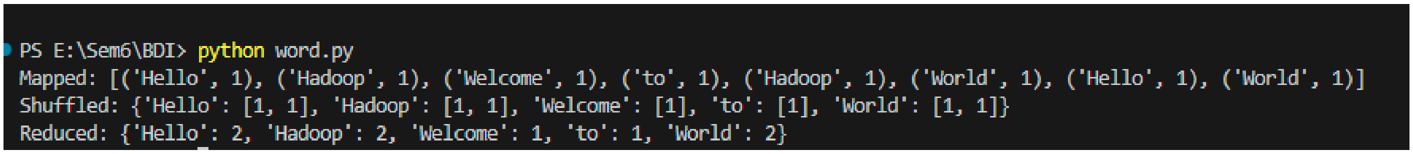
Conclusion: Thus, we have implemented matrix multiplicaton & word freauency count using map reduce.

WORDCOUNT:

CODE:

```
from collections import defaultdict

documents = [
    "Hello Hadoop",
    "Welcome to Hadoop World",
    "Hello World"
]

# Mapping Phase
def map_phase(documents):
    mapped = []
    for document in documents:
        for word in document.split():
            mapped.append((word, 1))
    return mapped

# Shuffling Phase
def shuffle_phase(mapped):
    shuffled = defaultdict(list)
    for key, value in mapped:
        shuffled[key].append(value)
    return shuffled

# Reducing Phase
def reduce_phase(shuffled):
    reduced = {}
    for key, values in shuffled.items():
        reduced[key] = sum(values)
    return reduced

# Driver code to simulate the MapReduce process
if__name__== "__main__":
    # Map Phase
    mapped = map_phase(documents)
    print(f"Mapped: {mapped}")

    # Shuffle Phase
    shuffled = shuffle_phase(mapped)
    print(f"Shuffled: {dict(shuffled)}")
    # Reduce Phase
    reduced = reduce_phase(shuffled)
    print(f"Reduced: {reduced}")
```

OUTPUT:



```
PS E:\Sem6\BDI> python word.py
Mapped: [('Hello', 1), ('Hadoop', 1), ('Welcome', 1), ('to', 1), ('Hadoop', 1), ('World', 1), ('Hello', 1), ('World', 1)]
Shuffled: {'Hello': [1, 1], 'Hadoop': [1, 1], 'Welcome': [1], 'to': [1], 'World': [1, 1]}
Reduced: {'Hello': 2, 'Hadoop': 2, 'Welcome': 1, 'to': 1, 'World': 2}
```

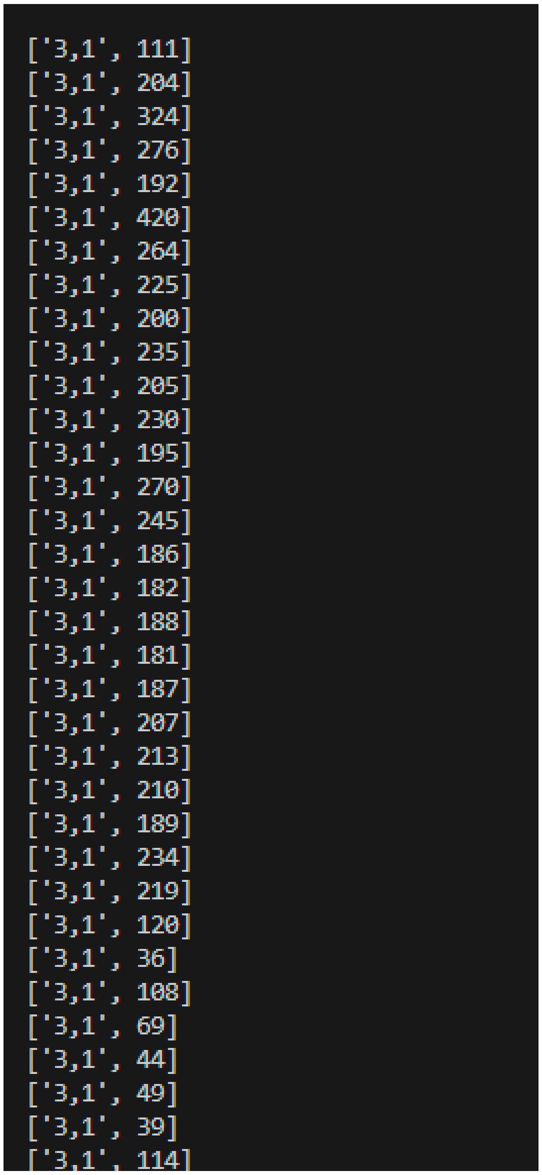
MATRIX MULTIPLICATION:

CODE:

```
with open("cache.txt") as cache_file:
    cache = cache_file.readline().split(",")
row_a, col_b = map(int, cache)
mapperOutput = open("mapperOutput.txt", "w")
for line in open("input.txt"):
    matrix_index, row, col, value = line.rstrip().split(",")
    if matrix_index == "A":
        for i in range(0, col_b):
            key = row + "," + str(i)
            mapperOutput.write("%s\t%s\t%s" % (key, col, value) + "\n")
    else:
        for j in range(0, row_a):
            key = str(j) + "," + col
            mapperOutput.write("%s\t%s\t%s" % (key, row, value) + "\n")
mapperOutput.close()

listMultiply1 = list()
listMultiply2 = list()
listAdd1 = list()
listAdd2 = list()
reducerTemp = list()
reducerOutput = list()

for line in open("mapperOutput.txt"):
    key, index, value = line.rstrip().split("\t")
    index, value = map(int, [index, value])
    listMultiply1.append((key, index, value))
```

```
listMultiply2 = listMultiply1

for i in listMultiply1:
    for j in listMultiply2:
        if i != j:
            if i[1] == j[1]:
                listAdd1.append([i[0], i[2] * j[2]])

for sublist in listAdd1:
    if sublist not in listAdd2:
        listAdd2.append(sublist)

listAdd1 = listAdd2

for i in listAdd1:
    for j in listAdd2:
        if i != j:
            if i[0] == j[0]:
                reducerTemp.append([i[0], i[1] + j[1]])

for sublist in reducerTemp:
    if sublist not in reducerOutput:
        reducerOutput.append(sublist)

# Print the result of this reducer
for i in reducerOutput:
    print(i)
```

OUTPUT:



```
['3,1', 111]
['3,1', 204]
['3,1', 324]
['3,1', 276]
['3,1', 192]
['3,1', 420]
['3,1', 264]
['3,1', 225]
['3,1', 200]
['3,1', 235]
['3,1', 205]
['3,1', 230]
['3,1', 195]
['3,1', 270]
['3,1', 245]
['3,1', 186]
['3,1', 182]
['3,1', 188]
['3,1', 181]
['3,1', 187]
['3,1', 207]
['3,1', 213]
['3,1', 210]
['3,1', 189]
['3,1', 234]
['3,1', 219]
['3,1', 120]
['3,1', 36]
['3,1', 108]
['3,1', 69]
['3,1', 44]
['3,1', 49]
['3,1', 39]
['3,1', 114]
```

Conclusion: Thus, we have implemented Matrix Multiplication and Word Frequency Count using MapReduce.