EXPERIMENT - 1

AIM:

To Develop a MapReduce program to calculate the frequency of a given word in a given file

OBJECTIVE:

To develop a MapReduce program using Python to calculate the frequency of occurrence of each word in a given input file. The program will utilize the MapReduce paradigm to efficiently distribute the task of counting word frequencies across multiple nodes in a distributed computing environment. The objective is to achieve scalable and parallel processing of large datasets, enabling effective analysis of text data for various applications such as natural language processing, text mining, and sentiment analysis.

THEORY:

In the MapReduce paradigm, the map function plays a crucial role in transforming a set of input data into a series of key-value pairs, typically referred to as tuples. Each element of the input data is broken down and associated with a key-value pair, where the key represents a unique identifier and the value signifies the occurrence or relevant information about the element. For instance, in the classic Word Count example, the input data might consist of a series of words. The map function processes each word and emits a key-value pair where the word itself serves as the key, and the value is set to 1 to indicate its occurrence.

For Example -

Input: "Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN, BUS, buS, caR, CAR, car, BUS, TRAIN"

Output: (Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1), (BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Following the map phase, the reduce function aggregates and combines the intermediate key-value pairs generated by the map function. Its primary role is to process and condense the data into a smaller set of tuples, typically by summing or applying other operations to the values associated with each unique key. In the Word Count example, the reduce function would receive a collection of key-value pairs where each key corresponds to a word, and the values represent the occurrences of that word across the input data. By consolidating these pairs, the reduce function produces a final output consisting of key-value pairs where the key is the word, and the value is the total count of occurrences.

For Example -

Input: (Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1), (BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Output: (BUS,7), (CAR,7), (TRAIN,4)

In summary, the map function facilitates the transformation of input data into key-value pairs, while the reduce function consolidates and summarizes these pairs to produce meaningful

results. Together, they form the core components of the MapReduce framework, enabling efficient processing of large-scale data tasks.

WORK FLOW DIAGRAM -

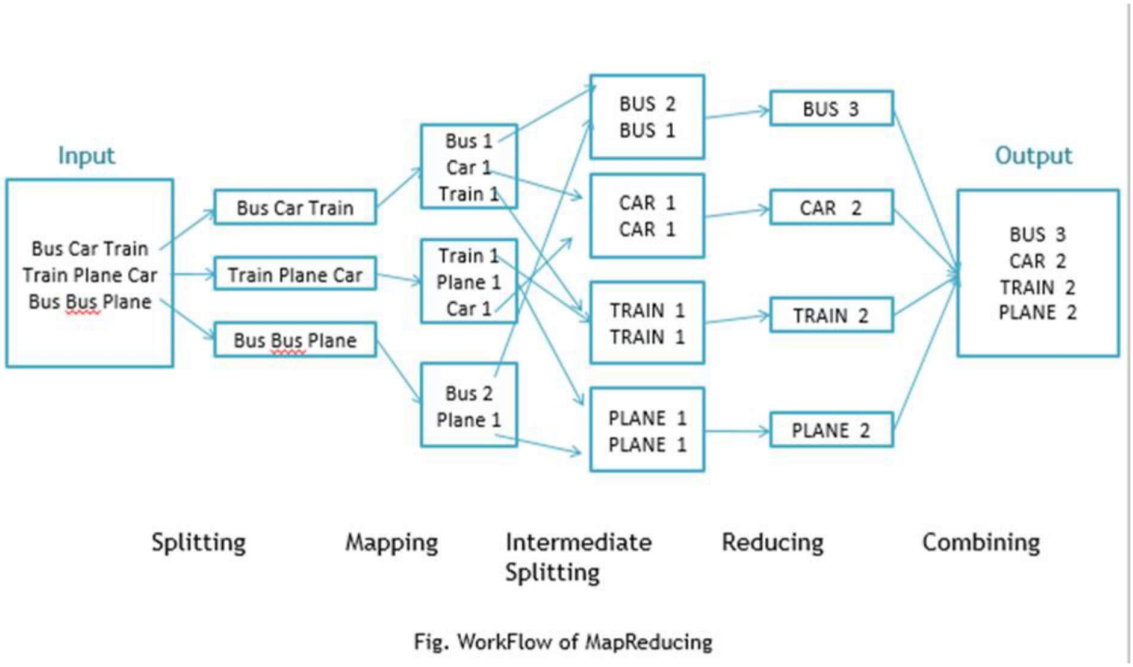


Fig. WorkFlow of MapReducing

CODE:

```
from mrjob.job import MRJob
import re


class WordCount(MRJob):


    def mapper(self, _, line):
        words = re.findall(r'\w+', line)
        for word in words:
            yield word.lower(), 1


    def reducer(self, word, counts):
        yield word, sum(counts)


if __name__ == '__main__':
```

WordCount.run()

OUTPUT:

```
PS C:\Users\rohan> python prac2.py "C:\Users\rohan\Desktop\abc.txt"
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory C:\Users\rohan\AppData\Local\Temp\prac2.rohan.20240419.125209.306732
Running step 1 of 1...
job output is in C:\Users\rohan\AppData\Local\Temp\prac2.rohan.20240419.125209.306732\output
Streaming final output from C:\Users\rohan\AppData\Local\Temp\prac2.rohan.20240419.125209.306732\output...
"bus"   7
"car"   7
"train" 4
Removing temp directory C:\Users\rohan\AppData\Local\Temp\prac2.rohan.20240419.125209.306732...
```

CONCLUSION:

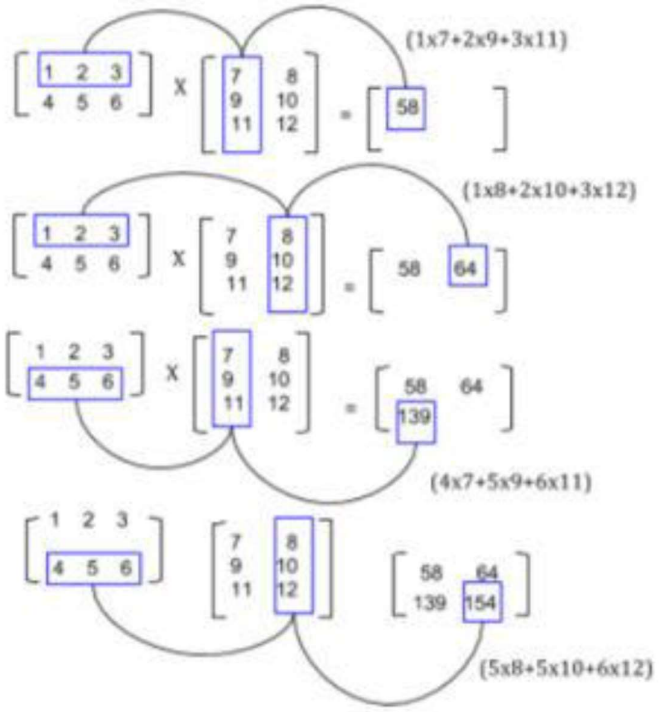
The MapReduce experiment demonstrates an effective approach for processing large datasets by employing the map and reduce functions. Through key-value pair transformations and aggregation, it efficiently computes word frequencies, showcasing the scalability and utility of the MapReduce paradigm.

EXPERIMENT - 2

AIM: Implement Matrix Multiplication using Map-Reduce

OBJECTIVE: The objective of this experiment is to implement matrix multiplication using the MapReduce paradigm in order to efficiently compute the product of two large matrices. By distributing the computation across multiple nodes in a distributed computing environment, we aim to achieve parallel processing and reduce the overall computation time required for matrix multiplication. This experiment demonstrates the scalability and effectiveness of MapReduce for handling complex mathematical operations on large-scale datasets, such as matrix multiplication, which is a fundamental operation in various fields including machine learning, scientific computing, and data analysis.

THEORY: In mathematics, matrix multiplication or the matrix product is a binary operation that produces a matrix from two matrices. The definition is motivated by linear equations and linear transformations on vectors, which have numerous applications in applied mathematics, physics, and engineering. In more detail, if A is an n × m matrix and B is an m × p matrix, their matrix product AB is an n × p matrix, in which the m entries across a row of A are multiplied with the m entries down a column of B and summed to produce an entry of AB. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.



CODE:

```
from collections import defaultdict

# Map function
def map_func(matrix, row, col, is_matrix_A):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if is_matrix_A:
                yield ((row, j), (matrix[i][j], 'A'))
            else:
                yield ((i, col), (matrix[i][j], 'B'))

# Reduce function
def reduce_func(indexes, value_pairs):
    matrix_A_values = []
    matrix_B_values = []
    result = 0

    for value, matrix_type in value_pairs:
        if matrix_type == 'A':
            matrix_A_values.append(value)
        else:
            matrix_B_values.append(value)

    for a_value in matrix_A_values:
        for b_value in matrix_B_values:
            result += a_value * b_value

    row, col = indexes
    yield (row, col), result
```

```
# Main function
def main():
    # Get matrix input from the user
    rows_A = int(input("Enter the number of rows for matrix A: "))
    cols_A = int(input("Enter the number of columns for matrix A: "))
    matrix_A = []
    print("Enter the elements of matrix A row-wise:")
    for i in range(rows_A):
        row = [int(x) for x in input().split()]
        matrix_A.append(row)

    cols_B = int(input("Enter the number of columns for matrix B: "))
    matrix_B = []
    print("Enter the elements of matrix B row-wise:")
    for i in range(cols_A):
        row = [int(x) for x in input().split()]
        matrix_B.append(row)

    # Map phase
    mapped_data = defaultdict(list)
    for i in range(rows_A):
        for j in range(cols_B):
            for map_result in map_func(matrix_A, i, j, True):
                mapped_data[map_result[0]].append(map_result[1])
            for map_result in map_func(matrix_B, i, j, False):
                mapped_data[map_result[0]].append(map_result[1])

    # Reduce phase
    result_matrix = []
    for key, value_pairs in mapped_data.items():
```

```
        for result in reduce_func(key, value_pairs):
            row, col = result[0]
            value = result[1]
            if len(result_matrix) <= row:
                result_matrix.append([])
            result_matrix[row].append(value)

    # Print the result
    print("\nMatrix A:")
    for row in matrix_A:
        print(row)

    print("\nMatrix B:")
    for row in matrix_B:
        print(row)

    print("\nResult Matrix (A x B):")
    for row in result_matrix:
        print(row)

if __name__ == '__main__':
    main()
```

OUTPUT:

```
PS C:\Users\rohan> python prac2.py
Enter the number of rows for matrix A: 2
Enter the number of columns for matrix A: 2
Enter the elements of matrix A row-wise:
1 2
3 4
Enter the number of columns for matrix B: 2
Enter the elements of matrix B row-wise:
5 6
7 8

Matrix A:
[1, 2]
[3, 4]

Matrix B:
[5, 6]
[7, 8]

Result Matrix (A x B):
[176, 264]
[240, 360]
PS C:\Users\rohan>
```

CONCLUSION: In conclusion, the implementation of matrix multiplication using the MapReduce paradigm demonstrates its effectiveness in efficiently computing the product of large matrices. By leveraging the parallel processing capabilities of MapReduce, we are able to distribute the computation across multiple nodes, thereby reducing the overall computation time and enabling the handling of large-scale datasets. This experiment underscores the scalability and utility of MapReduce for performing complex mathematical operations, highlighting its significance in various domains requiring efficient data processing and analysis.

EXPERIMENT - 3

AIM:

To Develop a MapReduce program to find the grades of students.

OBJECTIVE:

To develop a MapReduce program using Python and the mrjob library to determine the grades of students based on their marks in multiple subjects. The program utilizes the MapReduce paradigm to efficiently process large datasets and calculate the average marks for each student, subsequently assigning grades according to predefined criteria.

THEORY:

What is MapReduce?

MapReduce is a programming model and an associated implementation for processing and generating large datasets that are parallelizable. It consists of two main phases: the Map phase and the Reduce phase.

Map Phase:

- In this phase, the input data is divided into chunks, and each chunk is processed independently by a map function.
- The map function processes the input data and generates a set of intermediate key-value pairs.

Shuffle and Sort:

- This phase handles the distribution of intermediate key-value pairs across different nodes in the cluster.
- It ensures that all values associated with a particular key are grouped together.

Reduce Phase:

- In this phase, the intermediate key-value pairs are processed to produce the final output.
- The reduce function takes all values associated with a particular key and performs some aggregation or computation on them.

CODE:

```
from collections import defaultdict


# Map function

def map_func(student_record):

    student_id, marks = student_record.split(':')

    marks = [int(m) for m in marks.split(',')]

    for subject, mark in enumerate(marks):
```

```
        yield (student_id, (subject, mark))

# Reduce function
def reduce_func(student_id, subject_marks):
    total_marks = sum(mark for subject, mark in subject_marks)
    num_subjects = len(subject_marks)
    avg_marks = total_marks / num_subjects
    if avg_marks >= 80:
        grade = 'A'
    elif avg_marks >= 60:
        grade = 'B'
    elif avg_marks >= 40:
        grade = 'C'
    else:
        grade = 'D'
    yield (student_id, (total_marks, num_subjects, avg_marks, grade))

# Main function
def main():
    # Input data (student records)
    student_records = [
        '1001:80,75,90,92,88,75',
        '1002:62,67,55,68,72,61',
        '1003:38,42,51,25,32,27',
        '1004:91,87,75,62,72,63'
    ]

    # Map phase
    mapped_data = defaultdict(list)
    for student_record in student_records:
```

```
    for student_id, subject_mark in map_func(student_record):
        mapped_data[student_id].append(subject_mark)

# Reduce phase
for student_id, subject_marks in mapped_data.items():
    for result in reduce_func(student_id, subject_marks):
        print(f"Student ID: {result[0]}, Total Marks: {result[1][0]}, Number of Subjects: {result[1][1]}, Average Marks: {result[1][2]}, Grade: {result[1][3]}")

if __name__ == '__main__':
    main()
```

OUTPUT:

```
PS C:\Users\rohan> python prac2.py
Student ID: 1001, Total Marks: 500, Number of Subjects: 6, Average Marks: 83.33333333333333, Grade: A
Student ID: 1002, Total Marks: 385, Number of Subjects: 6, Average Marks: 64.16666666666667, Grade: B
Student ID: 1003, Total Marks: 215, Number of Subjects: 6, Average Marks: 35.833333333333336, Grade: D
Student ID: 1004, Total Marks: 450, Number of Subjects: 6, Average Marks: 75.0, Grade: B
```

CONCLUSION: In conclusion, the development of a MapReduce program to determine the grades of students has showcased the versatility and scalability of the MapReduce paradigm in handling educational data processing tasks. By efficiently distributing the computation across multiple nodes, the program efficiently calculates the average marks and assigns grades based on predefined criteria. This experiment underscores the effectiveness of MapReduce in processing large volumes of student data, demonstrating its applicability in educational analytics and other domains requiring scalable data processing solutions.

EXPERIMENT - 4

AIM: Mongo DB: Installation and Creation of database and Collection CRUD Document: Insert, Query, Update and Delete Document.

OBJECTIVE:

1. To install MongoDB on the local system.
2. To create a new database and collection.
3. To insert documents into the collection.
4. To query documents from the collection.
5. To update existing documents.
6. To delete documents from the collection.

THEORY:

MongoDB is a popular NoSQL database that stores data in a flexible, JSON-like format called BSON (Binary JSON). It is known for its scalability, flexibility, and performance. In this experiment, we'll focus on installing MongoDB, creating a new database and collection, and performing basic CRUD operations.

Steps:

- Install MongoDB:
  - Download MongoDB from the official website and follow the installation instructions for your operating system.
- Start MongoDB Service:
  - After installation, start the MongoDB service.
- Access MongoDB Shell:
  - Open a terminal or command prompt and access the MongoDB shell by running the command mongo.
- Create Database and Collection:
  - Use the use command to switch to a new or existing database.
  - Use the db.createCollection() command to create a new collection within the selected database.
- CRUD Operations:
  - Insert Document:
    - Use the db.collection.insertOne() or db.collection.insertMany() command to insert documents into the collection.
  - Query Document:
    - Use the db.collection.find() command to query documents from the collection.
  - Update Document:
    - Use the db.collection.updateOne() or db.collection.updateMany() command to update existing documents in the collection.
  - Delete Document:

- Use the db.collection.deleteOne() or db.collection.deleteMany() command to delete documents from the collection.

CODE:

```
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient('localhost', 27017)

# Creating a new database (Optional)
client.drop_database('mydatabase')  # Dropping if the database already exists
db = client['mydatabase']

# Accessing the collection
collection = db['mycollection']

# Insert Documents
documents = [
    {"name": "John", "age": 30},
    {"name": "Alice", "age": 25},
    {"name": "Bob", "age": 35}
]
collection.insert_many(documents)

# Query Documents
query_result = collection.find()
print("Query Result:")
for document in query_result:
    print(document)

# Update Document
collection.update_one({"name": "John"}, {"$set": {"age": 31}})
```

```
updated_document = collection.find_one({"name": "John"})
print("\nUpdated Document:")
print(updated_document)

# Delete Document
collection.delete_one({"name": "John"})
deleted_document = collection.find_one({"name": "John"})
print("\nDeleted Document:")
print(deleted_document)
```

OUTPUT:

```
[Running] python -u "c:\Users\rohan\prac2.py"
Query Result:
{'_id': ObjectId('6622700ac3c26fb6ed6ae02e'), 'name': 'John', 'age': 30}
{'_id': ObjectId('6622700ac3c26fb6ed6ae02f'), 'name': 'Alice', 'age': 25}
{'_id': ObjectId('6622700ac3c26fb6ed6ae030'), 'name': 'Bob', 'age': 35}

Updated Document:
{'_id': ObjectId('6622700ac3c26fb6ed6ae02e'), 'name': 'John', 'age': 31}

Deleted Document:
None

[Done] exited with code=0 in 0.843 seconds
```

CONCLUSION:

This practical experiment provided hands-on experience with MongoDB, including installation, database, and collection creation, along with performing CRUD operations on documents. Understanding these fundamental operations is crucial for working with MongoDB databases effectively.

EXPERIMENT - 5

AIM: Visualization: Connect to data, Build Charts and Analyze Data, Create Dashboard, Create Stories using Tableau/PowerBI.

OBJECTIVE:

1. To connect to a data source using Tableau or PowerBI.
2. To build different types of charts and visualizations to analyze data effectively.
3. To create an interactive dashboard with multiple visualizations.
4. To develop a data story that communicates insights derived from the visualizations.

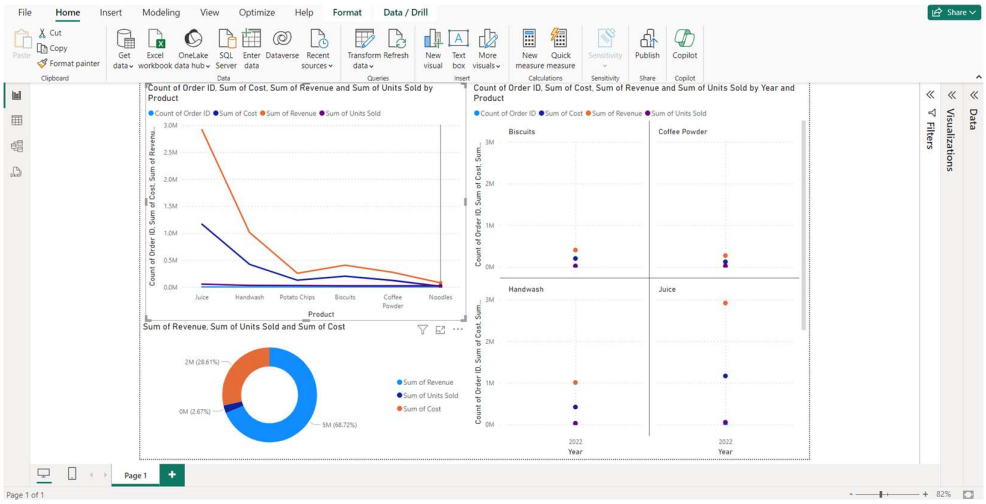
THEORY:

Tableau and PowerBI are powerful data visualization tools that allow users to connect to various data sources, transform raw data into insightful visualizations, and create interactive dashboards and stories. These tools provide a user-friendly interface for users to explore data visually and communicate insights effectively.

Steps:

1. Connect to Data Source:
    - Launch Tableau or PowerBI and connect to the desired data source such as Excel, CSV, SQL database, or online sources like Google Analytics.
2. Build Charts and Analyze Data:
    - Utilize the drag-and-drop interface to create different types of charts like bar charts, line charts, scatter plots, etc., to analyze the data.
    - Apply filters, groupings, and calculations to explore data trends and patterns.
3. Create Dashboard:
    - Combine multiple visualizations onto a single dashboard to provide an overview of key insights.
    - Add interactivity by linking filters and actions between visualizations for a seamless user experience.
4. Create Stories:
    - Develop a narrative around the data by creating a story that guides viewers through the insights derived from the visualizations.
    - Incorporate text, images, and annotations to provide context and highlight key findings.

OUTPUT:



CONCLUSION:

This practical experiment provided hands-on experience with Tableau or PowerBI for data visualization and analysis. By connecting to data sources, building various charts, creating interactive dashboards, and crafting data stories, users can effectively communicate insights derived from the data to stakeholders and make informed decisions. Mastery of these tools empowers users to unlock the full potential of their data and drive actionable insights.