**Problem Statement:** **Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.**

* **Find the number of occurrence of each word appearing in the input file(s)**
* **Performing a Map Reduce Job for word search count (look for specific keywords in a file)**

**Solution 1:**

from collections import defaultdict

import multiprocessing

# Mapper function

def mapper(chunk):

word\_count = defaultdict(int)

for word in chunk.split():

word\_count[word] += 1

return word\_count

# Reducer function

def reducer(results):

word\_count = defaultdict(int)

for wc in results:

for word, count in wc.items():

word\_count[word] += count

return word\_count

# MapReduce function

def mapreduce(data, mapper, reducer, num\_processes=2):

with multiprocessing.Pool(processes=num\_processes) as pool:

mapped\_data = pool.map(mapper, data)

reduced\_data = reducer(mapped\_data)

return reduced\_data

if \_\_name\_\_ == "\_\_main\_\_":

# Sample input data

data = [

"hello world",

"world is beautiful",

"hello beautiful world"

]

# Execute MapReduce

word\_counts = mapreduce(data, mapper, reducer)

# Output results

print("Word Counts:")

for word, count in word\_counts.items():

print(f"{word}: {count}")

**Solution 2:**

from collections import defaultdict

from multiprocessing import Pool

# Mapper function

def mapper(text):

word\_count = defaultdict(int)

for word in text.split():

word\_count[word.lower()] += 1

return word\_count

# Reducer function

def reducer(count\_dicts):

word\_count = defaultdict(int)

for count\_dict in count\_dicts:

for word, count in count\_dict.items():

word\_count[word] += count

return word\_count

# Main function to orchestrate MapReduce process

def main():

# Sample input data

data = [

"Hello world",

"World is beautiful",

"Hello beautiful world"

]

# Map phase

with Pool() as pool:

mapped\_data = pool.map(mapper, data)

# Reduce phase

reduced\_result = reducer(mapped\_data)

# Print the final word counts

print("Word Counts:")

for word, count in reduced\_result.items():

print(f"{word}: {count}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Mapreduce program in python using external dataset with excel file:**

**Solution**: implement a MapReduce program in Python using an external dataset stored in an Excel file, you can use libraries like pandas to read the data from the Excel file and then apply the MapReduce paradigm. Below is an example that demonstrates this:

import pandas as pd

import multiprocessing

from collections import defaultdict

def mapper(data):

word\_count = defaultdict(int)

for row in data.itertuples():

for word in str(row.text).split(): # Assuming the text column contains the text data

word\_count[word] += 1

return word\_count

def reducer(word\_counts):

final\_word\_count = defaultdict(int)

for word\_count in word\_counts:

for word, count in word\_count.items():

final\_word\_count[word] += count

return final\_word\_count

def mapreduce(data, num\_workers=2):

pool = multiprocessing.Pool(processes=num\_workers)

chunk\_size = len(data) // num\_workers

chunks = [data[i:i+chunk\_size] for i in range(0, len(data), chunk\_size)]

word\_counts = pool.map(mapper, chunks)

final\_word\_count = reducer(word\_counts)

pool.close()

pool.join()

return final\_word\_count

if \_\_name\_\_ == "\_\_main\_\_":

# Read data from Excel file

df = pd.read\_excel('your\_excel\_file.xlsx') # Replace 'your\_excel\_file.xlsx' with the path to your Excel file

result = mapreduce(df, num\_workers=2)

print(result)

In this example:

pd.read\_excel('your\_excel\_file.xlsx') reads the data from the Excel file into a Pandas DataFrame. Replace 'your\_excel\_file.xlsx' with the path to your Excel file.

The mapper function iterates over each row in the DataFrame and splits the text into words, incrementing the count for each word.

The reducer function combines the word counts from different chunks.

The mapreduce function distributes the data among worker processes using multiprocessing.Pool, applies the mapper function to each chunk, and then reduces the results using the reducer function.

The \_\_name\_\_ == "\_\_main\_\_" block ensures that the script is executed only if it is the main program.

Make sure that the Excel file contains a column with text data that you want to process. Adjust the column name ('text' in this example) in the mapper function if needed.

**Problem statement: Write a python code to understand the overall programming architecture using Map Reduce API**

**Solution: 1**

Implementation of the Map Reduce programming model using Python's multiprocessing module:

import multiprocessing

def mapper(data):

# Mapper function: takes input data and emits key-value pairs

key, value = data

# Example: Splitting a sentence into words and emitting (word, 1) pairs

for word in value.split():

yield (word, 1)

def reducer(data):

# Reducer function: takes key-value pairs and aggregates values by key

key, values = data

# Example: Summing up counts for each word

return (key, sum(values))

def mapreduce(data, mapper, reducer, num\_processes=2):

# Initialize multiprocessing pool

pool = multiprocessing.Pool(processes=num\_processes)

# Map phase: apply mapper function to data

mapped\_data = pool.map(mapper, data)

# Flatten mapped data

flattened\_data = [item for sublist in mapped\_data for item in sublist]

# Group mapped data by key

grouped\_data = {}

for key, value in flattened\_data:

grouped\_data.setdefault(key, []).append(value)

# Reduce phase: apply reducer function to grouped data

reduced\_data = pool.map(reducer, grouped\_data.items())

# Close the pool

pool.close()

pool.join()

return reduced\_data

if \_\_name\_\_ == "\_\_main\_\_":

# Example input data

input\_data = [

(1, "apple banana"),

(2, "banana orange"),

(3, "orange apple")

]

# Perform MapReduce

result = mapreduce(input\_data, mapper, reducer)

# Output the result

for item in result:

print(item)

This code demonstrates a basic implementation of the MapReduce programming model using Python's multiprocessing module. It defines a mapper function, a reducer function, and a mapreduce function to perform the MapReduce operation on the provided input data. The mapreduce function distributes the mapping and reducing tasks across multiple processes using Python's multiprocessing.Pool. Finally, it prints the result of the MapReduce operation.

**Solution 2:**

Creating a full-fledged MapReduce API in Python involves defining classes and methods that allow users to easily define mappers, reducers, and execute MapReduce jobs. Below is a simplified implementation of a MapReduce API in Python:

import multiprocessing

class MapReduce:

def \_\_init\_\_(self, num\_processes=2):

self.num\_processes = num\_processes

def mapper(self, data):

raise NotImplementedError("Subclasses must implement mapper method")

def reducer(self, data):

raise NotImplementedError("Subclasses must implement reducer method")

def \_map(self, data):

return [self.mapper(item) for item in data]

def \_reduce(self, data):

reduced\_data = {}

for key, value in data:

reduced\_data.setdefault(key, []).append(value)

return [self.reducer(item) for item in reduced\_data.items()]

def run(self, data):

pool = multiprocessing.Pool(processes=self.num\_processes)

mapped\_data = pool.map(self.\_map, data)

flattened\_data = [item for sublist in mapped\_data for item in sublist]

result = self.\_reduce(flattened\_data)

pool.close()

pool.join()

return result

# Example usage:

class WordCount(MapReduce):

def mapper(self, data):

key, value = data

for word in value.split():

yield (word, 1)

def reducer(self, data):

key, values = data

return (key, sum(values))

if \_\_name\_\_ == "\_\_main\_\_":

input\_data = [

(1, "apple banana"),

(2, "banana orange"),

(3, "orange apple")

]

# Create a WordCount MapReduce job

word\_count\_job = WordCount()

# Run the MapReduce job

result = word\_count\_job.run(input\_data)

# Output the result

for item in result:

print(item)

In this code:

MapReduce class provides the basic structure for a MapReduce job. Users can subclass it and implement their custom mapper and reducer methods.

The run method orchestrates the execution of the MapReduce job by distributing the mapping and reducing tasks across multiple processes.

The WordCount class is an example of a subclass that implements a specific MapReduce job for counting words in a collection of documents. It overrides the mapper and reducer methods to define the word counting logic.

Example usage demonstrates how to create an instance of the WordCount class and execute a MapReduce job to count words in a given dataset.

**Problem Statement:**

Stop word elimination problem: • Input: A large textual file containing one sentence per line A small file containing a set of stop words (One stop word per line) • Output: A textual file containing the same sentences of the large input file without the words appearing in the small file.

**Solution:**

To solve the stop word elimination problem, you can write a script in Python. Here's a sample code to achieve this:

def remove\_stopwords(sentence, stopwords):

"""

Function to remove stop words from a sentence.

"""

words = sentence.split()

filtered\_sentence = [word for word in words if word.lower() not in stopwords]

return ' '.join(filtered\_sentence)

def main(input\_file, stopwords\_file, output\_file):

"""

Main function to read input file, remove stopwords, and write to output file.

"""

# Read stop words from stopwords\_file

with open(stopwords\_file, 'r') as f:

stopwords = set([word.strip().lower() for word in f.readlines()])

# Process input file

with open(input\_file, 'r') as f:

sentences = f.readlines()

# Remove stop words from each sentence and write to output file

with open(output\_file, 'w') as f:

for sentence in sentences:

filtered\_sentence = remove\_stopwords(sentence, stopwords)

f.write(filtered\_sentence + '\n')

if \_\_name\_\_ == "\_\_main\_\_":

input\_file = "input.txt" # Path to input file containing one sentence per line

stopwords\_file = "stopwords.txt" # Path to file containing stop words, one per line

output\_file = "output.txt" # Path to output file

main(input\_file, stopwords\_file, output\_file)

This Python script reads input from a large file containing one sentence per line and another file containing stop words (one word per line). It then removes stop words from each sentence and writes the processed sentences to an output file.

Make sure to replace "input.txt", "stopwords.txt", and "output.txt" with the appropriate file paths for your files.

**Problem Statement**

Write a Map Reduce program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record- oriented. Data available at: https://github.com/tomwhite/hadoopbook/tree/master/input/ncdc/all. • Find average, max and min temperature for each year in NCDC data set? • Filter the readings of a set based on value of the measurement, Output the line of input files associated with a temperature value greater than 30.0 and store it in a separate file.

**Solution:**

**To implement a MapReduce program for mining weather data, we'll use Python's mrjob library, which simplifies the process of writing MapReduce jobs. Below are two MapReduce jobs, one for finding the average, maximum, and minimum temperature for each year in the NCDC dataset, and another for filtering readings based on a temperature value greater than 30.0 and storing them in a separate file.**

**Make sure you have mrjob installed. You can install it using pip install mrjob.**

**First, let's create a Python script for the MapReduce jobs:**

**from mrjob.job import MRJob**

**from mrjob.step import MRStep**

**class WeatherDataStats(MRJob):**

**def mapper(self, \_, line):**

**# Split the line into fields**

**fields = line.split(',')**

**if len(fields) == 3:**

**# Extract year and temperature from the line**

**year = fields[0].strip()**

**temperature = float(fields[1].strip())**

**# Emit the year and temperature**

**yield year, temperature**

**def reducer(self, year, temperatures):**

**# Calculate average, max, and min temperature for each year**

**temperatures = list(temperatures)**

**avg\_temp = sum(temperatures) / len(temperatures)**

**max\_temp = max(temperatures)**

**min\_temp = min(temperatures)**

**yield year, (avg\_temp, max\_temp, min\_temp)**

**class FilterTemperature(MRJob):**

**def mapper(self, \_, line):**

**# Split the line into fields**

**fields = line.split(',')**

**if len(fields) == 3:**

**# Extract year, temperature, and other data from the line**

**year = fields[0].strip()**

**temperature = float(fields[1].strip())**

**other\_data = fields[2].strip()**

**# Emit the line if temperature is greater than 30.0**

**if temperature > 30.0:**

**yield year, (temperature, other\_data)**

**def reducer(self, year, readings):**

**# Write the filtered readings to a file**

**with open(f"filtered\_readings\_{year}.txt", "w") as f:**

**for reading in readings:**

**f.write(f"{year}, {reading[0]}, {reading[1]}\n")**

**if \_\_name\_\_ == '\_\_main\_\_':**

**# Weather data statistics job**

**MRJobStats = WeatherDataStats(args=['ncdc\_data.txt'])**

**with MRJobStats.make\_runner() as runner:**

**runner.run()**

**# Filtering job**

**MRJobFilter = FilterTemperature(args=['ncdc\_data.txt'])**

**with MRJobFilter.make\_runner() as runner:**

**runner.run()**

**In this script:**

**WeatherDataStats class calculates the average, maximum, and minimum temperature for each year.**

**FilterTemperature class filters readings with a temperature greater than 30.0 and stores them in separate files.**

**Both jobs read from the same input file ncdc\_data.txt.**

**Make sure to replace 'ncdc\_data.txt' with the path to your actual weather data file. You can run this script using Python, and it will perform the MapReduce jobs as described.**

**Write python script to perform Store the basic information about students such as roll no, name, date of birth , and address of student using various collection types such as List, Set and Map**

**Solation:**

Below is a Python script that stores basic information about students such as roll number, name, date of birth, and address using various collection types such as list, set, and dictionary (map).

**class Student:**

**def \_\_init\_\_(self, roll\_no, name, dob, address):**

**self.roll\_no = roll\_no**

**self.name = name**

**self.dob = dob**

**self.address = address**

**# Using List to store student information**

**students\_list = [**

**Student(1, "John", "2000-01-01", "123 Main St"),**

**Student(2, "Alice", "2001-02-02", "456 Elm St"),**

**Student(3, "Bob", "1999-03-03", "789 Oak St")**

**]**

**# Using Set to store student information (assuming roll number is unique)**

**students\_set = {**

**Student(1, "John", "2000-01-01", "123 Main St"),**

**Student(2, "Alice", "2001-02-02", "456 Elm St"),**

**Student(3, "Bob", "1999-03-03", "789 Oak St")**

**}**

**# Using Dictionary (Map) to store student information with roll number as key**

**students\_dict = {**

**1: Student(1, "John", "2000-01-01", "123 Main St"),**

**2: Student(2, "Alice", "2001-02-02", "456 Elm St"),**

**3: Student(3, "Bob", "1999-03-03", "789 Oak St")**

**}**

**# Printing information using List**

**print("Students List:")**

**for student in students\_list:**

**print(f"Roll No: {student.roll\_no}, Name: {student.name}, DOB: {student.dob}, Address: {student.address}")**

**# Printing information using Set**

**print("\nStudents Set:")**

**for student in students\_set:**

**print(f"Roll No: {student.roll\_no}, Name: {student.name}, DOB: {student.dob}, Address: {student.address}")**

**# Printing information using Dictionary**

**print("\nStudents Dictionary:")**

**for roll\_no, student in students\_dict.items():**

**print(f"Roll No: {student.roll\_no}, Name: {student.name}, DOB: {student.dob}, Address: {student.address}")**

**In this script:**

**Student class represents each student with attributes roll number, name, date of birth, and address.**

**Student information is stored in a list, a set, and a dictionary.**

**The script then prints out the student information stored in each collection type.**

**This script demonstrates the usage of different collection types in Python to store student information.**

**Problem Statement:**

Basic CRUD operations in MongoDB

**Solution:**

To integrate Python with MongoDB, you can use the official MongoDB Python driver called PyMongo. PyMongo allows you to interact with MongoDB databases from your Python code.

Here's a basic guide to integrating Python with MongoDB using PyMongo:

1. Install PyMongo: You can install PyMongo using pip, Python's package installer.
   1. **pip install pymongo**
2. Connect to MongoDB:You need to establish a connection to your MongoDB database using PyMongo. You typically connect to a MongoDB server running on localhost, but you can connect to a remote server as well.

import pymongo

# Connect to the MongoDB server

client = pymongo.MongoClient("mongodb://localhost:27017/")

# Select the database

db = client["mydatabase"]

1. Create or Access a Collection:In MongoDB, data is stored in collections. You can create a new collection or access an existing one.

# Access an existing collection or create it if it doesn't exist

collection = db["students"]

1. Insert Documents:You can insert documents (Python dictionaries) into the collection.

# Insert a single document

student = {"name": "John", "age": 20, "grade": "A"}

collection.insert\_one(student)

# Insert multiple documents

students = [

{"name": "Alice", "age": 22, "grade": "B"},

{"name": "Bob", "age": 21, "grade": "C"}

]

collection.insert\_many(students)

1. Query Documents:You can query documents from the collection using various criteria.

# Find a single document

result = collection.find\_one({"name": "John"})

print(result)

# Find all documents that match a query

results = collection.find({"grade": "A"})

for result in results:

print(result)

1. Update Documents: You can update existing documents in the collection.

# Update a single document

collection.update\_one({"name": "John"}, {"$set": {"age": 21}})

# Update multiple documents

collection.update\_many({"grade": "B"}, {"$set": {"grade": "A"}})

1. Delete Documents:You can delete documents from the collection.

# Delete a single document

collection.delete\_one({"name": "John"})

# Delete multiple documents

collection.delete\_many({"grade": "C"})

Note for reference use given link: https://pymongo.readthedocs.io/en/stable/

**Problem Statement:** Retrieve various types of documents from students collection

Solution:

To retrieve various types of documents from a collection named students, you would typically use a database query. The specific query syntax depends on the database system you are using. Assuming you're using MongoDB, here are some common scenarios for retrieving documents from the students collection:

Retrieve all documents:

To retrieve all documents from the students collection, you can use the find() method without any parameters:

result = db.students.find()

Retrieve documents based on a specific condition:

If you want to retrieve documents based on certain conditions, you can pass a query filter to the find() method. For example, to retrieve documents where the age field is greater than 20:

result = db.students.find({"age": {"$gt": 20}})

Retrieve documents with specific fields:

If you only want to retrieve specific fields from the documents, you can pass a projection to the find() method. For example, to retrieve only the name and age fields:

result = db.students.find({}, {"name": 1, "age": 1})

Retrieve a single document:

If you want to retrieve a single document, you can use the find\_one() method. This method returns the first document that matches the query filter:

result = db.students.find\_one({"name": "John"})

Retrieve a limited number of documents:

You can limit the number of documents returned by using the limit() method:

result = db.students.find().limit(10)

Replace db with your actual MongoDB database object, and adjust the query filters, projections, and limits as needed for your specific requirements.

**Problem Statement:** To find documents from Students collection

**Solution:**

To find documents from the "Students" collection, assuming you're using MongoDB, you would typically use a MongoDB query. Below is an example of how you can find documents from the "Students" collection using Python and pymongo, the official MongoDB driver for Python:

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/') # Assuming MongoDB is running locally

# Select database and collection

db = client['your\_database\_name'] # Replace 'your\_database\_name' with your actual database name

collection = db['Students']

# Find all documents in the collection

cursor = collection.find()

# Iterate over the cursor to access the documents

for document in cursor:

print(document)

This code will connect to your MongoDB instance, select the specified database (your\_database\_name), and then select the "Students" collection. It then retrieves all documents from the collection using the find() method and iterates over the cursor to print each document.

Remember to replace 'your\_database\_name' with the actual name of your database. If authentication is enabled, you will need to provide the appropriate credentials as well.

Ensure that you have the pymongo library installed. You can install it via pip:

pip install pymongo

Adjust the code as necessary based on your specific requirements, such as adding query conditions or projections to filter or limit the returned documents.