LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

Result Management System

There are 10000 students 6 subjects in a very big university. You need to develop a result management system for the university.

Basic: (For the capstone)

Generate 10000 students' profile

Generate 6 general subjects (Electronics, Programming, Database, Data Science, Mathematics and DSA}

Generate 6 subject marks for the 10000 students

Use spark and Hadoop framework map reduce etc... whatever you had learnt in the class to process the students' marks

Do basic analysis over the marks and come up with the statistics

Display in a dashboard with statistics

This is the code that we run in the Hadoop for the map reduce Function

Step1.Check if pip is installed correctly:

Using :pip3 –version

Step2.ensure you're using the correct version of pip:

pip3 install pandas faker openpyxl

Step3.Upgrade pip:

python3 -m pip install --upgrade pip

Step4. Check for Permissions:

sudo pip3 install pandas faker openpyxl

Step5.Try Installing Packages Individually:

pip3 install pandas

pip3 install faker

pip3 install openpyxl

Step6.Virtual Environment (Optional):

python3 -m venv myenv

source myenv/bin/activate

pip install pandas faker openpyxl

Step7.Create a Python Script: First, create a Python script that contains the code you want to execute. You can use any text editor to create a new Python file, such as nano or vim.

nano generate_data.py

Step8.Add Your Python Code

```
import pandas as pd
from faker import Faker
import random

# Initialize Faker instance
fake = Faker()

# Function to generate random marks
def generate_marks():
    return [random.randint(50, 100) for _ in range(5)]

# List to hold the generated data
data = []

# Generate data for 10,000 members
for _ in range(10000):
    name = fake.name()
    marks = generate_marks()
    data.append([name] + marks)

# Create DataFrame
columns = ['Name', 'Subject 1', 'Subject 2', 'Subject 3', 'Subject
4', 'Subject 5']
df = pd.DataFrame(data, columns=columns)
```

```
# Save the DataFrame to an Excel file
df.to_excel('random_names_and_marks.xlsx', index=False)
print("Excel file saved successfully!")
```

Step9.Save the File: In nano, after pasting the code, save the file by pressing CTRL + O (write out), then press Enter to confirm the filename. Exit nano by pressing CTRL + X.

Run the Python Script: Now that you've created the script, you can run it using:

```
python3 generate_data.py
```

Step10.Verify the Excel File: You should now see the file random_names_and_marks.xlsx in your current directory. You can confirm its existence by running:

```
ls
```

Step11.Open the Excel File: If you'd like to open the file and verify the data, you can use any spreadsheet software like LibreOffice, Excel, or an online viewer.

Step12.Use the Data in Hadoop: Now that you have the dataset, you can start working with it in your Hadoop environment.

Steps to Process Excel Data in Hadoop using MapReduce:

```
import pandas as pd


# Load the Excel file
df = pd.read_excel('random_names_and_marks.xlsx')
```

```
# Save it as a CSV file
df.to_csv('random_names_and_marks.csv', index=False)
```

Upload the CSV file to HDFS:

```
hdfs dfs -put random_names_and_marks.csv
/user/hadoop/input/
```

Step13.Create a MapReduce Job: Now that the file is in HDFS, you can create a MapReduce job to process the data.

A simple example MapReduce job could calculate the average marks for each subject. Here’s how you might structure your job:

Mapper: This will read the input CSV file, extract the marks, and output key-value pairs where the key is the subject and the value is the mark.

Reducer: The reducer will process the keys (subjects) and calculate the average marks.

Step14.Mapper Code (Java example): Below is a simple example of a Mapper class for this task:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
```

```
public class MarksMapper extends Mapper<Object, Text, Text,
IntWritable> {


    private final static IntWritable mark = new IntWritable();

    private Text subject = new Text();


    public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {

        String[] columns = value.toString().split(",");

        if (columns.length == 6) {

            for (int i = 1; i < 6; i++) { // Loop over subjects columns
(1 to 5)

                subject.set("Subject " + i);

                mark.set(Integer.parseInt(columns[i]));

                context.write(subject, mark);

            }

        }

    }

}
```

Step15.Reducer Code (Java example): Here’s a simple Reducer class to calculate the average marks per subject:

```
import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class MarksReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
        int sum = 0;
        int count = 0;
        for (IntWritable val : values) {
            sum += val.get();
            count++;
        }
        result.set(sum / count); // Calculate average
        context.write(key, result);
    }
}
```

Step16.Driver Code (Java example): Here's an example of the main driver class to configure and run the MapReduce job:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
```

```
}

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MarksAverage {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Marks Average");
        job.setJarByClass(MarksAverage.class);
        job.setMapperClass(MarksMapper.class);
        job.setReducerClass(MarksReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Step17.Compile and Run the MapReduce Job:

Using:Compile your Java MapReduce code into a .jar file.

Step18.Submit the job to Hadoop:

```
hadoop jar your-job.jar MarksAverage
/user/hadoop/input/random_names_and_marks.csv
/user/hadoop/output
```

Check the Output: Once the job completes, you can check the results:

```
hdfs dfs -cat /user/hadoop/output/part-r-00000
```

This Process can run In the google Collab Sheet

Spark Frame Work Code:

```
import pandas as pd

import numpy as np

import random

from pyspark.sql import SparkSession

from pyspark.sql import Row

from pyspark.sql import functions as F

import matplotlib.pyplot as plt

import seaborn as sns


# 1. Project Setup (Libraries imported)


# 2. Generate 10,000 Students' Profile

names = ["Ramesh", "Suresh", "Hitesh", "Mukesh", "Rajesh",
"Mahesh", "Pankaj", "Sanjay", "Vikas", "Amit", "Karan", "Arjun"]
```

```
num_students = 10000
students = pd.DataFrame({
    'Student_ID': range(1, num_students + 1),
    'Name': [random.choice(names) + " " +
random.choice(["Sharma", "Verma", "Patel", "Yadav", "Gupta"])
for _ in range(num_students)]
})
print(students.head())
spark =
SparkSession.builder.appName("ResultManagement").getOrCr
eate()
students_spark = spark.createDataFrame(students)

# 3. Generate 6 General Subjects
subjects = ["Electronics", "Programming", "Database", "Data
Science", "Mathematics", "DSA"]

# 4. Generate 6 Subject Marks for 10,000 Students
marks_data = []
for student in students_spark.collect():
    student_id = student.Student_ID
    marks_row = {"Student_ID": student_id}
    for subject in subjects:
        marks_row[subject] = random.randint(0, 100)
    marks_data.append(Row(**marks_row))
marks_df = spark.createDataFrame(marks_data)
```

```
# 5. Use Spark and Hadoop Framework
student_marks_df = students_spark.join(marks_df,
"Student_ID")
for subject in subjects:
    student_marks_df =
student_marks_df.withColumn(f"{subject}_Grade",
F.when(F.col(subject) >= 90, "A").when(F.col(subject) >= 80,
"B").when(F.col(subject) >= 70, "C").when(F.col(subject) >= 60,
"D").otherwise("F"))
average_marks =
student_marks_df.select([F.mean(col).alias(f"Average_{col}")
for col in subjects])


# 6. Do Basic Analysis and Statistics
average_marks.show()
statistics =
student_marks_df.select([F.stddev(col).alias(f"StdDev_{col}")
```

Visualizing the Results:

BarPlot:

```
average_marks_pd.plot(kind='bar', title='Average Marks')
plt.show()
```

Scatter Plot:

```
# Scatter Plot (Example: Electronics vs. Programming)
plt.figure(figsize=(8, 6))
```

```
sns.scatterplot(x='Electronics', y='Programming',
data=student_marks_pd, hue='Electronics_Grade')
plt.title('Scatter Plot: Electronics vs. Programming')
plt.show()

spark.stop()
```

Heat Map:

```
subjects_marks = student_marks_pd[subjects]
plt.figure(figsize=(10, 8))
sns.heatmap(subjects_marks.corr(), annot=True,
cmap='coolwarm', linewidths=.5)
plt.title('Correlation Heatmap of Subject Marks')
plt.show()
```

Bar Graph:

```
average_marks_pd.plot(kind='bar', title='Average Marks per
Subject')
plt.xlabel('Subjects')
plt.ylabel('Average Marks')
plt.xticks(range(len(subjects)), subjects, rotation=45)
plt.tight_layout()
plt.show()
```

Graph:

```
for subject in subjects:
    plt.figure(figsize=(6, 4))
    plt.hist(student_marks_pd[subject], bins=20,
edgecolor='black')
    plt.title(f'Distribution of Marks in {subject}')
    plt.xlabel('Marks')
    plt.ylabel('Frequency')
    plt.show()
```