CC LAB PRGRMS

2.Write a C program for storing data in a simulated cloud storage environment using a local file.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char data[1024];

    // Open a file in write mode. Simulating a cloud storage by using a local file.
    FILE *fptr = fopen("cloud_storage.txt", "w");
    if (fptr == NULL) {
        printf("Error opening the file!\n");
        exit(1);
    }

    // Get user input
    printf("Enter text to store in the cloud: ");
    fgets(data, sizeof(data), stdin);

    // Write data to the file
    fprintf(fptr, "%s", data);

    // Close the file
    fclose(fptr);

    printf("Data successfully saved to 'cloud_storage.txt'\n");

    return 0;
}
```

3.Write a C-Program for CPU usage Monitoring and Logging on Cloud-BasedUbuntu Server.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

// Function to read the current CPU usage from the /proc/stat file
float get_cpu_usage() {
    long double a[4], b[4], loadavg;
    FILE *fp;

    // Read first set of CPU statistics
```

```
    fp = fopen("/proc/stat","r");
    if (fp == NULL) {
        perror("Failed to open /proc/stat");
        exit(EXIT_FAILURE);
    }
    fscanf(fp, "%*s %Lf %Lf %Lf %Lf", &a[0], &a[1], &a[2], &a[3]);
    fclose(fp);

    // Sleep for a second to get the second set of CPU statistics
    sleep(1);

    // Read second set of CPU statistics
    fp = fopen("/proc/stat","r");
    if (fp == NULL) {
        perror("Failed to open /proc/stat");
        exit(EXIT_FAILURE);
    }
    fscanf(fp, "%*s %Lf %Lf %Lf %Lf", &b[0], &b[1], &b[2], &b[3]);
    fclose(fp);

    // Calculate the CPU usage
    loadavg = ((b[0]+b[1]+b[2]) - (a[0]+a[1]+a[2])) /
            ((b[0]+b[1]+b[2]+b[3]) - (a[0]+a[1]+a[2]+a[3]));

    return loadavg;
}

int main() {
    FILE *log_file;
    char *filename = "cpu_usage.log";

    // Open log file for writing
    log_file = fopen(filename, "w");
    if (log_file == NULL) {
        perror("Failed to open log file");
        return EXIT_FAILURE;
    }

    // Loop to record CPU usage
    for (int i = 0; i < 10; ++i) {
        float usage = get_cpu_usage();
        fprintf(log_file, "CPU Usage: %.2f%%\n", usage * 100);
        printf("Logged CPU Usage: %.2f%%\n", usage * 100);
```

```
        // Sleep for 5 seconds
        sleep(5);
    }

    // Close log file
    fclose(log_file);
    printf("CPU usage logging completed.\n");

    return 0;
}
```

4.Use Google App Engine Launcher to launch the web applications.

Step 1: Install Google App Engine

1. **Install Google App Engine 1.9.62**:

   Go To: https://www.npackd.org/p/com.google.AppEnginePythonSDK/1. 9.62 and install the 1.9.62 version.

   - Go to npackd.org for Google App Engine Python SDK 1.9.62 and download the 1.9.62 version.
   - Ensure Python 2.7 is installed on your system. If not, download and install it from python.org.

Step 2: Set Up Google App Engine Launcher

1. **Create a New Folder on Desktop**:
   - Right-click on the desktop, select `New` -> `Folder`, and name it appropriately (e.g., `MyGAEApp`).
2. **Configure Preferences**:
   - Open Google App Engine Launcher.
   - Go to `Edit` -> `Preferences`.
   - Set the path for Google App Engine and Python.
     - **Google App Engine Path**: This should be the installation directory of the Google App Engine SDK.
     - **Python Path**: This should be the path where Python 2.7 is installed (typically `C:\Python27` on Windows).

Step 3: Create a New Application

1. **Create a New Application**:
   - In Google App Engine Launcher, go to `File` -> `Create New Application`.

    - Select the path of the folder created earlier (e.g., `MyGAEApp`).
2. **Configure the Application Folder**:
    - Open the created folder. You should see an automatically created sub-folder (e.g., `MyGAEApp`).
    - Ensure the project directory contains an `app.yaml` file. This file provides instructions for Google App Engine to provision resources for your app.

Example `app.yaml` File

Create an `app.yaml` file in the project directory with the following content:

Example Python Application (`main.py`)

Create a `main.py` file in the project directory with the following content:

```
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.write('Hello, Google App Engine!')

app = webapp2.WSGIApplication([
    ('/', MainPage),
], debug=True)
```

Step 4: Run the Application

1. **Launch the Application**:
    - In Google App Engine Launcher, select the created application.
    - Click the `Run` button.
    - The application will start running locally, and you should see output indicating that the server is running.
2. **Access the Application**:
    - Open a web browser and go to `http://localhost:8080`.
    - You should see the message `Hello, Google App Engine!`.

8. Fifa
Import Necessary Libraries:

- Import the required PySpark modules for creating a Spark session and defining the schema.

Initialize SparkSession:

- Create a Spark session with the application name "FIFADataFrameExample".

Define the Schema for the FIFA Dataset:

- Define a schema using `StructType` and `StructField` to specify the data types for each column in the dataset.

Load FIFA Dataset from CSV:

- Read the CSV file into a DataFrame using the defined schema.

Show DataFrame:

- Display the contents of the DataFrame using `show()`.

Select and Show Specific Columns:

- Select the `previous_points` column from the DataFrame and display it.

Stop SparkSession:

- Stop the Spark session to free up resources.

```
# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
FloatType

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("FIFADataFrameExample") \
    .getOrCreate()

# Define the schema for the FIFA dataset
schema = StructType([
    StructField("team", StringType(), True),
    StructField("team_code", StringType(), True),
    StructField("association", StringType(), True),
    StructField("rank", IntegerType(), True),
    StructField("previous_rank", IntegerType(), True),
```

```
    StructField("points", FloatType(), True),
    StructField("previous_points", FloatType(), True)
])

# Load FIFA dataset from CSV
file_path = "/content/fifa_ranking_2022-10-06.csv"
fifa_df = spark.read.csv(file_path, header=True, schema=schema)

# Show DataFrame
fifa_df.show()

# Select and show specific columns
fifa_df.select("previous_points").show()

# Stop SparkSession
spark.stop()
```

5.Setting up a single-node Hadoop cluster involves several steps. Below is a structured and detailed guide with explanations and corrections to ensure clarity and completeness:

Step-by-Step Guide to Setting Up a Single-Node Hadoop Cluster

1. Update and Install Necessary Packages
```sh
sudo apt update
sudo apt install ssh pdsh
```

2. Configure SSH for Passwordless Login
```sh
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
ssh localhost
```

3. Install Java (Hadoop Requires Java)
```sh
sudo apt install openjdk-8-jdk
java -version
```

4. Download and Extract Hadoop
```sh
```

```
sudo wget -P ~
https://archive.apache.org/dist/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
tar xzf ~/hadoop-3.2.1.tar.gz
mv ~/hadoop-3.2.1 ~/hadoop
```

5. Configure Hadoop Environment Variables
Edit `hadoop-env.sh` to set the `JAVA_HOME` path:
```sh
nano ~/hadoop/etc/hadoop/hadoop-env.sh
```
Add the following line:
```sh
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

6. Move Hadoop Directory to `/usr/local` and Set Permissions
```sh
sudo mv ~/hadoop /usr/local/hadoop
sudo chown -R $(whoami):$(whoami) /usr/local/hadoop
sudo chmod -R 755 /usr/local/hadoop
```

7. Update System Environment Variables
Edit `/etc/environment`:
```sh
sudo nano /etc/environment
```
Add the following lines:
```sh
HADOOP_HOME="/usr/local/hadoop"
PATH="$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin"
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```
Reload the environment variables:
```sh
source /etc/environment
```

8. Create a Hadoop User (Optional)
```sh
sudo adduser hadoopuser
sudo usermod -aG sudo hadoopuser
```

9. Configure Hadoop Core and HDFS
Edit `core-site.xml`:
```sh
sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
```
Add the following configuration:
```xml
<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
   </property>
   <property>
      <name>hadoop.tmp.dir</name>
      <value>/usr/local/hadoop/tmp</value>
   </property>
</configuration>
```

Edit `hdfs-site.xml`:
```sh
sudo nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```
Add the following configuration:
```xml
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
</configuration>
```

10. Configure MapReduce and YARN
Edit `mapred-site.xml`:
```sh
sudo nano /usr/local/hadoop/etc/hadoop/mapred-site.xml
```
Add the following configuration:
```xml
<configuration>
   <property>
      <name>mapreduce.framework.name</name>
```

```
        <value>yarn</value>
    </property>
</configuration>
```

Edit `yarn-site.xml`:
```sh
sudo nano /usr/local/hadoop/etc/hadoop/yarn-site.xml
```
Add the following configuration:
```xml
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

11. Format the HDFS Filesystem
```sh
hdfs namenode -format
```

12. Start Hadoop Services
```sh
start-dfs.sh
start-yarn.sh
```

13. Verify Hadoop Installation
You can check the Hadoop services running by accessing the web UIs:
- NameNode: http://localhost:9870
- ResourceManager: http://localhost:8088

14. Clean Up (Optional)
Remove `pdsh` if no longer needed:
```sh
sudo apt remove pdsh
```

6.Hadoop Programming: Word Count Map Reduce Program Using Eclipse.

1. Driver Code (`WCDriver.java`)
The driver code configures the Hadoop job and sets up the mapper and reducer classes.

```java
import java.io.IOException;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool {
   public int run(String[] args) throws IOException {
      if (args.length < 2) {
         System.out.println("Please provide valid inputs");
         return -1;
      }
      JobConf conf = new JobConf(WCDriver.class);
      FileInputFormat.setInputPaths(conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new Path(args[1]));
      conf.setMapperClass(WCMapper.class);
      conf.setReducerClass(WCReducer.class);
      conf.setMapOutputKeyClass(Text.class);
      conf.setMapOutputValueClass(IntWritable.class);
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      JobClient.runJob(conf);
      return 0;
   }

   public static void main(String[] args) throws Exception {
      int exitCode = ToolRunner.run(new WCDriver(), args);
      System.out.println(exitCode);
   }
}
```

2. Mapper Code (`WCMapper.java`)
The mapper code processes each line of input and emits key-value pairs for each word.

```java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {
   public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
output, Reporter rep) throws IOException {
      String line = value.toString();
      // Splitting the line on spaces
      for (String word : line.split(" ")) {
         if (word.length() > 0) {
            output.collect(new Text(word), new IntWritable(1));
         }
      }
   }
}
```

3. Reducer Code (`WCReducer.java`)
The reducer code aggregates the counts for each word and emits the final count.

```java
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
```

```
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter rep) throws IOException {
        int count = 0;
        // Counting the frequency of each word
        while (values.hasNext()) {
            count += values.next().get();
        }
        output.collect(key, new IntWritable(count));
    }
}
```

Compiling and Running the MapReduce Job

1. **Compile the Java Code**:
    - Save the three Java files (`WCDriver.java`, `WCMapper.java`, `WCReducer.java`) in the same directory.
    - Open a terminal and navigate to the directory containing these files.
    - Compile the code using the following commands:

    ```sh
    javac -classpath $(hadoop classpath) -d . WCDriver.java WCMapper.java
WCReducer.java
    ```

2. **Create a JAR File**:
    - Package the compiled classes into a JAR file:

    ```sh
    jar cvf hadoop-mapreduce-example.jar *.class
    ```

3. **Run the MapReduce Job**:
    - Ensure your Hadoop cluster is running.
    - Copy your input data to HDFS:

    ```sh
    hdfs dfs -mkdir /sample
    hdfs dfs -mkdir /sample/input
    hdfs dfs -put /path/to/your/input/file /sample/input
    ```

    - Run the Hadoop job using the following command:

```sh
hadoop jar hadoop-mapreduce-example.jar WCDriver /sample/input /sample/output
```

4. **Check the Output**:
   - After the job completes, check the output in HDFS:

```sh
hdfs dfs -cat /sample/output/part-00000
```

7.File Management tasks in Hadoop using HDFS commands
a. Adding files to HDFS
b. Retrieving files from HDFS
c. Deleting files from HDFS

File Management Tasks in Hadoop Using HDFS Commands
1. Create a Directory in HDFS

Usage:
```sh
hadoop fs -mkdir <paths>
```

Example:
```sh
hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2
```

2. List the Contents of a Directory

Usage:
```sh
hadoop fs -ls <args>
```

Example:
```sh
hadoop fs -ls /user/saurzcode
```

3. Upload and Download a File in HDFS

a. Upload a File to HDFS

Usage:
```sh
hadoop fs -put <localsrc> ... <HDFS_dest_Path>
```

Example:
```sh
hadoop fs -put /home/saurzcode/Samplefile.txt /user/saurzcode/dir3/
```

b. Download a File from HDFS

Usage:
```sh
hadoop fs -get <hdfs_src> <localdst>
```

Example:
```sh
hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/
```

4. See the Contents of a File

Usage:
```sh
hadoop fs -cat <path[filename]>
```

Example:
```sh
hadoop fs -cat /user/saurzcode/dir1/abc.txt
```

5. Remove a File or Directory in HDFS

Usage:
```sh
hadoop fs -rm <arg>
```

Example:
```sh
```

```
hadoop fs -rm /user/saurzcode/dir1/abc.txt
```

6. Display Last Few Lines of a File

Usage:
```sh
hadoop fs -tail <path[filename]>
```

Example:
```sh
hadoop fs -tail /user/saurzcode/dir/abc.txt
```

Summary of Commands

Task	Command
Create a directory	`hadoop fs -mkdir <paths>`
List directory contents	`hadoop fs -ls <args>`
Upload a file to HDFS	`hadoop fs -put <localsrc> ... <HDFS_dest_Path>`
Download a file from HDFS	`hadoop fs -get <hdfs_src> <localdst>`
See contents of a file	`hadoop fs -cat <path[filename]>`
Remove a file or directory	`hadoop fs -rm <arg>`
Display last few lines of a file	`hadoop fs -tail <path[filename]>`