ASSIGNMENT 9

PROBLEM STATEMENT: -
Design and develop a distributed application to find the coolest/hottest year from the available weather data. Use weather data from the Internet and process it using Map Reduce.

OBJECTIVE:
1.Students should be able to design the Map function to extract year and temperature data from each record.

2.Students should be able to Implement the Reduce function to aggregate temperature readings for each year and calculate the average temperature.

THEORY:

To design and develop a distributed application to find the coolest/hottest year from available weather data using Map Reduce, we need to break down the process into several steps. Let's outline the theory behind this process:

1. **Data Collection:**
   - Obtain weather data from reliable sources available on the internet. This data can be historical weather data collected over several years.
2. **Data Preprocessing:**
   - Clean the raw weather data to remove any inconsistencies or errors.
   - Organize the data into a suitable format for processing, such as CSV or JSON.
3. **Map Reduce Programming Model:**
   - Map Reduce is a programming model for processing and generating large datasets in a distributed environment.
   - In Map Reduce, computations are divided into two phases: the map phase and the reduce phase.
   - The map phase involves processing individual data elements and emitting intermediate key-value pairs.
   - The reduce phase aggregates and processes intermediate key-value pairs to produce the final output.
4. **Map Function:**
   - In our case, the map function will read each weather data record and extract the year as the key and the temperature as the value.
   - It will emit key-value pairs where the key is the year and the value is the temperature.
5. **Partitioning:**
   - The Map Reduce framework partitions the intermediate key-value pairs based on the keys.
   - Each partition is processed independently by a reduce task.
6. **Reduce Function:**
   - The reduce function receives all intermediate key-value pairs for a particular key (year) from different map tasks.
   - It computes the average temperature for each year.

- Finally, it identifies the year with the highest or lowest average temperature based on the requirement.

7. **Combining Multiple Jobs:**
   - In some cases, multiple Map Reduce jobs might be required to achieve the desired result.
   - For example, one job could compute the average temperature for each year, and another job could find the hottest or coolest year based on the output of the first job.
8. **Output:**
   - The output of the Map Reduce job will contain the year with the hottest or coolest temperature, along with the corresponding temperature value.
9. **Scalability and Fault Tolerance:**
   - Map Reduce is designed to scale horizontally, allowing the processing of large datasets across multiple nodes in a distributed cluster.
   - The framework provides fault tolerance by automatically restarting failed tasks on other nodes.
10. **Implementation:**
    - Implement the Map Reduce job using a suitable programming framework, such as Apache Hadoop or Apache Spark.
    - Configure the cluster environment to distribute the computation across multiple nodes.
    - Monitor the job execution and optimize performance as needed.

By following this approach, we can design and develop a distributed application to find the coolest/hottest year from available weather data using the Map Reduce programming model. This approach enables efficient processing of large datasets in a distributed environment, making it suitable for big data analytics tasks like weather data analysis.

Hadoop Installation

1. Java Installation

- sudo apt update
- Install JDK

https://download.oracle.com/otn/java/jdk/9.0.1+11/jdk-9.0.1_linux-x64_bin.tar.gz?AuthParam=1683886421_1603b66108615d82845d5ab9e22ec42e
OR
https://kenfavors.com/code/how-to-manually-install-oracle-java-9-on-ubuntu-16-04/

Step to Retrieve Java Path

- dirname $(dirname $(readlink -f $(which java)))

/usr/lib/jvm/java-11-openjdk-amd64

- **Change java**

$ update-alternatives --config java

2. SSH Installation:

- ssh-keygen -t rsa

- cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
- chmod 640 ~/.ssh/authorized_keys
- sudo apt-get install openssh-server
- ssh localhost

3. Hadoop Configuration:

- wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
- tar xzvf hadoop-3.3.4.tar.gz
- Rename hadoop3.3.4 to Hadoop
- gedit ~/.bashrc

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop path of hadoop folder
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

- source ~/.bashrc
- Switch to Hadoop Directory
  /hadoop/etc/hadoop
- Edit Core-site.xml: gedit core-site.xml

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

- Edit mapred-site xml :gedit mapred-site xml

```
<configuration>
<property>
  <name>mapreduce.job.tracker</name>
  <value>localhost:9870</value>
</property>
</configuration>
```

- Edit mapred-site xml :gedit hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

- Edit mapred-site xml :gedit Hdfs-site.xml

Hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
```

</configuration>

4. Switch to root

hdfs namenode -format

5. Cd hadoop

Cd hadoop/sbin

./start-all.sh

```
2023-05-24 10:17:09,336 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2023-05-24 10:17:09,336 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2023-05-24 10:17:09,339 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2023-05-24 10:17:09,339 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry ca
che entry expiry time is 600000 millis
2023-05-24 10:17:09,340 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2023-05-24 10:17:09,340 INFO util.GSet: VM type       = 64-bit
2023-05-24 10:17:09,340 INFO util.GSet: 0.029999999329447746% max memory 1.9 GB = 602.1 KB
2023-05-24 10:17:09,340 INFO util.GSet: capacity      = 2^16 = 65536 entries
2023-05-24 10:17:09,355 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1290038774-127.0.1.1-168490
3629350
2023-05-24 10:17:09,383 INFO common.Storage: Storage directory /tmp/hadoop-gurukul/dfs/name has been suc
cessfully formatted.
2023-05-24 10:17:09,403 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-gurukul/dfs/n
ame/current/fsimage.ckpt_0000000000000000000 using no compression
2023-05-24 10:17:09,475 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-gurukul/dfs/name/cur
rent/fsimage.ckpt_0000000000000000000 of size 402 bytes saved in 0 seconds .
2023-05-24 10:17:09,485 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-24 10:17:09,519 INFO namenode.FSNamesystem: Stopping services started for active state
2023-05-24 10:17:09,520 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-05-24 10:17:09,523 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-05-24 10:17:09,523 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at gurukul-ThinkCentre-M800/127.0.1.1
************************************************************/
gurukul@gurukul-ThinkCentre-M800:~$
```

jps

./stop-all.sh

Following Steps need to perform incase of any system Error:

Error localhost: rcmd: socket: Permission denied

https://tecadmin.net/how-to-install-apache-hadoop-on-ubuntu-22-04/

I also encountered the same thing, I did so I found that my pdsh default rcmd is rsh, not ssh, rsh and ssh remote login authentication is not the same, when installing hadoop I configured ssh localhost password-free login, but rsh is not possible.

so, try :

1.check your pdsh default rcmd rsh

pdsh -q -w localhost

See what your pdsh default rcmd is.

2.Modify pdsh's default rcmd to ssh

export PDSH_RCMD_TYPE=ssh

you can be added to ~/.bashrc, and source ~/.bashrc

3.sbin / start-dfs.sh

Steps to Check Hadoop is Properly Installed or Not

- Switch to hadoop bin
- hadoop fs -mkdir -p /user/gurukul/input
- http://localhost:9870

Steps to Compiler and Run the application

- Command to run Weathercode Mapreduce

  javac -d . WeatherDriver.java WeatherMapper.java WeatherReducer.java -cp "$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.3.4.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-3.3.4.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-3.3.4.jar:~/WeatherMapReduce/*:$HADOOP_HOME/lib/*"

  hadoop fs -put /home/gurukul/WeatherMapReduce/sample_weather.txt /user/gurukul

  hadoop jar /home/gurukul/WeatherMapReduce/weather.jar WeatherDriver input out

Mapper class

```
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.mapreduce.Mapper;
public class MapperClass extends Mapper<LongWritable, Text, Text, LongWritable>{
        @Override
        protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
String w[] =value.toString().split(" ");
for (String word:w)
{
context.write(new Text(word), new LongWritable(1));
```

```
}
}
}
```

Reducer Class

```
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.mapreduce.Reducer;
public class ReducerClass extends Reducer<Text, LongWritable, Text, IntWritable> {
@Override
protected void reduce(Text key, Iterable<LongWritable> value,Context context)
throws IOException, InterruptedException {
int cnt=0;
for(LongWritable i:value)
{
cnt=cnt+1;
}
context.write(key, new IntWritable(cnt));
}
}
```

Driver Class

```
import org.apache.hadoop.conf.Configured;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class DriverClass extends Configured implements Tool {
@Override
public int run(String[] arg0) throws Exception {
Job job= new Job(getConf(),"KRN");
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setMapperClass(MapperClass.class);
job.setReducerClass(ReducerClass.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(LongWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path("input"));
```

```
FileOutputFormat.setOutputPath(job, new Path("out"));
job.setJarByClass(DriverClass.class);// to Run on hadoop
job.waitForCompletion(true);//Logs Display
return 0;
}
public static void main(String[] args) throws Exception {
ToolRunner.run(new DriverClass(), args);
}
}
```

CONCLUSION:

In this way we have by design a distributed application to find the coolest/hottest year from the available weather data.

ORAL QUESTION:

1. Can you briefly explain what MapReduce is and how it works?

2. What are the advantages of using MapReduce for processing large-scale data?

3. How does data shuffling occur in MapReduce, and why is it important?

Code:

```
import csv
from functools import reduce
from collections import defaultdict

# Define mapper function to emit (year, temperature) pairs
def mapper(row):
    year = row["Date/Time"].split("-")[0]  # Extract year from
"Date/Time" column
    temperature = float(row["Temp_C"])  # Convert temperature to
float
    return (year, temperature)

# Define reducer function to calculate sum and count of temperatures
for each year
def reducer(accumulated, current):
    accumulated[current[0]][0] += current[1]
    accumulated[current[0]][1] += 1
    return accumulated

# Read the weather dataset
weather_data = []
with open("weather_data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        weather_data.append(row)

# Map phase
mapped_data = map(mapper, weather_data)

# Reduce phase
reduced_data = reduce(reducer, mapped_data, defaultdict(lambda: [0,
0]))

# Calculate average temperature for each year
avg_temp_per_year = {year: total_temp / count for year, (total_temp,
count) in reduced_data.items()}

# Find coolest and hottest year
coolest_year = min(avg_temp_per_year.items(), key=lambda x: x[1])
hottest_year = max(avg_temp_per_year.items(), key=lambda x: x[1])

print("Coolest Year:", coolest_year[0], "Average Temperature:",
coolest_year[1])
```

```
print("Hottest Year:", hottest_year[0], "Average Temperature:",
hottest_year[1])
```

Output: