CMR
INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test III – July 2024

Sub:	Big Data Analytics							Sub Code:	22MCA412
Date:	27/07/2024	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	With a neat diagram, explain the working of map reduce in DFS. **OR**	10	CO2	L2
2	Explain mapper and reducer in java with a program example	10	CO2	L2
	PART II			
3	Write a note on hadoop streaming for the following: a)Ruby b) Python **OR**	5+5	CO2	L3
4	Write on note on configuration API	10	CO3	L3
	PART III			
5	Explain writing a unit test for mapper and reducer function. **OR**	10	CO2	L3
6	Explain hadoop pipes with a program	10	CO3	L3
	PART IV			
7	Write a note on running a job locally on test data **OR**	10	CO2	L2
8	Explain how to run a job on a cluster.	10	CO2	L2
	PART V			
9	Write a program for the following: a) Copying a local file to a Hadoop filesystem b) fsData Output Stream **OR**	5+5	CO3	L4
10	Write a program for the following: a) Debugging a job b) Application to find the maximum temperature, using a combiner function for efficiency	5+5	CO3	L4

Sure, let's go through each question one by one based on the provided PDF.

PART I

1. With a neat diagram, explain the working of MapReduce in DFS.
Answer:
MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

Diagram and Explanation:

1. **Input Data Splitting:**
   - The input data is split into fixed-size pieces called "splits".
   - Each split is processed by a separate "Map" task.

2. **Mapping:**
   - The map function processes each split, producing intermediate key-value pairs.
   - Example: For a word count problem, the input split might be a single line of text, and the map function would output a key-value pair for each word, such as (word, 1).

3. **Shuffling and Sorting:**
   - The intermediate key-value pairs are shuffled and sorted by the key.
   - This step groups all values associated with the same key together.

4. **Reducing:**
   - The reduce function processes each group of key-value pairs to produce the final output.
   - Example: The reduce function might sum the values for each key to produce a final word count.

5. **Output:**
   - The final output is written to HDFS.

Diagram:

```plaintext
   Input Data
       |
     Split
       |
   ---------
   |       |
 Map       Map
   |       |
Intermediates
   |       |
Shuffle and Sort
   |       |
 Reduce
   |
 Output Data
```

---

2. Explain mapper and reducer in Java with a program example.
Answer:
In Hadoop, the Mapper and Reducer are two key components of the MapReduce programming model.

Mapper Example:

```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.mapreduce.Mapper;

public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String[] words = value.toString().split("\\s+");
      for (String str : words) {
         word.set(str);
         context.write(word, one);
      }
   }
}
```

Reducer Example:

```java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class IntSumReducer 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;
      for (IntWritable val : values) {
         sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
   }
}
```

Explanation:
- **Mapper:** The TokenizerMapper splits the input text into words and emits each word with a count of one.
- **Reducer:** The IntSumReducer sums the counts for each word and emits the word with its total count.

PART II

3. Write a note on Hadoop streaming for the following:
a) Ruby
b) Python

Answer:
Hadoop Streaming is a utility that comes with the Hadoop distribution. It allows users to create and run MapReduce jobs with any executable or script as the mapper and/or the reducer.

a) Ruby:
- **Example Script:**
  ```ruby
  # Mapper
  ARGF.each do |line|
    line.split.each do |word|
      puts "#{word}\t1"
    end
  end
  ```
  ```ruby
  # Reducer
  current_word = nil
  current_count = 0
  ARGF.each do |line|
    word, count = line.split("\t")
    count = count.to_i
    if current_word == word
      current_count += count
    else
      if current_word
        puts "#{current_word}\t#{current_count}"
      end
      current_word = word
      current_count = count
    end
  end
  puts "#{current_word}\t#{current_count}" if current_word
  ```

b) Python:
- **Example Script:**
  ```python
  # Mapper
  import sys
  for line in sys.stdin:
      for word in line.strip().split():
          print(f"{word}\t1")
  ```

  ```python
  # Reducer
  ```

```
import sys
from collections import defaultdict

word_count = defaultdict(int)
for line in sys.stdin:
    word, count = line.strip().split('\t')
    word_count[word] += int(count)

for word, count in word_count.items():
    print(f"{word}\t{count}")
```

Explanation:
- These scripts read input from standard input, process the data, and write output to standard output.
- Hadoop Streaming passes the data from HDFS to the scripts and captures the output back into HDFS.

---

4. Write a note on configuration API.
Answer:
The Hadoop Configuration API is used to configure the Hadoop framework and its applications. It manages the configuration parameters for Hadoop's runtime behavior.

- **Key Classes and Methods:**
  - **Configuration Class:** This class manages a set of configuration parameters and their values.
    ```java
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "hdfs://namenode:8020");
    ```
  - **Reading Configuration:**
    ```java
    String fsDefaultName = conf.get("fs.defaultFS");
    ```
  - **Loading from XML:**
    ```java
    conf.addResource(new Path("/path/to/core-site.xml"));
    conf.addResource(new Path("/path/to/hdfs-site.xml"));
    ```
  - **Configuration Files:** `core-site.xml`, `hdfs-site.xml`, and `mapred-site.xml` are the primary files for configuring Hadoop.

Explanation:
- The Configuration API allows users to programmatically configure Hadoop and manage the parameters required for the execution of the Hadoop cluster and MapReduce jobs.

PART III

5. Explain writing a unit test for mapper and reducer function.
Answer:
Unit testing for Hadoop MapReduce jobs involves testing the Mapper and Reducer classes individually to ensure they perform the expected transformations.

Example:

Using the JUnit framework for testing a word count mapper and reducer.

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.apache.hadoop.mrunit.mapreduce.ReduceDriver;
import org.junit.Before;
import org.junit.Test;
import java.util.Arrays;

public class WordCountTest {
   private MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;
   private ReduceDriver<Text, IntWritable, Text, IntWritable> reduceDriver;

   @Before
   public void setUp() {
      TokenizerMapper mapper = new TokenizerMapper();
      IntSumReducer reducer = new IntSumReducer();
      mapDriver = MapDriver.newMapDriver(mapper);
      reduceDriver = ReduceDriver.newReduceDriver(reducer);
   }

   @Test
   public void testMapper() {
      mapDriver.withInput(new LongWritable(1), new Text("hello world"))
            .withOutput(new Text("hello"), new IntWritable(1))
            .withOutput(new Text("world"), new IntWritable(1))
            .runTest();
   }

   @Test
   public void testReducer() {
      reduceDriver.withInput(new Text("hello"), Arrays.asList(new IntWritable(1), new IntWritable(1)))
               .withOutput(new Text("hello"), new IntWritable(2))
               .runTest();
   }
}
```

Explanation:
- The `MapDriver` and `ReduceDriver` classes from the `mrunit` library are used to simulate and test the Mapper and Reducer behavior.
- The `withInput` and `withOutput` methods are used to define the expected input and output for the Mapper and Reducer.

---

6. Explain Hadoop pipes with a program.
Answer:
Hadoop Pipes is a C++ API to implement MapReduce applications for Hadoop. It provides a way to write MapReduce programs in C++ and interact with Hadoop's native Java APIs.

Example:

A simple word count example in C++ using Hadoop Pipes.

Mapper:
```cpp
#include <iostream>
#include <hadoop/Pipes.hh>
#include <hadoop/TemplateFactory.hh>
#include <hadoop/StringUtils.hh>

class WordCountMapper : public HadoopPipes::Mapper {
public:
   void map(HadoopPipes::MapContext& context) {
      std::string line = context.getInputValue();
      std::vector<std::string> words = HadoopUtils::splitString(line, " ");
      for (size_t i = 0; i < words.size(); ++i) {
         context.emit(words[i], "1");
      }
   }
};
```

Reducer:
```cpp
class WordCountReducer : public HadoopPipes::Reducer {
public:
   void reduce(HadoopPipes::ReduceContext& context) {
      int count = 0;
      while (context.nextValue()) {
         count += HadoopUtils::toInt(context.getInputValue());
      }
      context.emit(context.getInputKey(), HadoopUtils::toString(count));
   }
};
```

Main:
```cpp


int main(int argc, char *argv[]) {
   return HadoopPipes::runTask(HadoopPipes::TemplateFactory<WordCountMapper,
WordCountReducer>());
}
```

Explanation:
- **WordCountMapper:** The mapper reads each line, splits it into words, and emits each word with a count of 1.
- **WordCountReducer:** The reducer sums the counts for each word and emits the final count.
- **Main:** The main function sets up the Pipes job by running the task with the specified Mapper and Reducer.

PART IV

7. Write a note on running a job locally on test data.
Answer:
Running a Hadoop job locally is useful for testing and debugging purposes. It allows developers to test their MapReduce jobs without the need for a full Hadoop cluster.

Steps to Run a Job Locally:

1. **Set Up Local Job Runner:**
   - In the job configuration, set the `mapreduce.framework.name` property to `local`.
   ```java
   Configuration conf = new Configuration();
   conf.set("mapreduce.framework.name", "local");
   ```

2. **Prepare Input Data:**
   - Place the input data in the local file system.
   ```plaintext
   input/
     file01.txt
     file02.txt
   ```

3. **Run the Job:**
   - Configure and run the MapReduce job as usual.
   ```java
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   FileInputFormat.addInputPath(job, new Path("input"));
   FileOutputFormat.setOutputPath(job, new Path("output"));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
   ```

Explanation:
- Setting `mapreduce.framework.name` to `local` ensures the job runs in the local JVM.
- Input and output paths should be in the local file system.
- This setup allows for quick iteration and debugging of MapReduce jobs without the overhead of a Hadoop cluster.

---

8. Explain how to run a job on a cluster.
Answer:
Running a Hadoop job on a cluster involves submitting the job to the Hadoop JobTracker or ResourceManager, which manages the job execution across the distributed cluster.

Steps to Run a Job on a Cluster:

1. **Set Up Cluster Configuration:**
   - Ensure the Hadoop cluster is configured and running.

- Distribute the necessary configuration files (`core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`) across all nodes.

2. **Prepare Input Data:**
   - Place the input data in HDFS.
   ```bash
   hdfs dfs -mkdir /user/hadoop/input
   hdfs dfs -put local_input_data /user/hadoop/input
   ```

3. **Submit the Job:**
   - Configure and submit the job using the Hadoop command or API.
   ```java
   Configuration conf = new Configuration();
   Job job = Job.getInstance(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   FileInputFormat.addInputPath(job, new Path("/user/hadoop/input"));
   FileOutputFormat.setOutputPath(job, new Path("/user/hadoop/output"));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
   ```

4. **Monitor Job Execution:**
   - Use the Hadoop web interface or CLI to monitor job progress.
   ```bash
   hadoop job -status <job-id>
   ```

Explanation:
- The Hadoop job configuration points to the HDFS paths for input and output data.
- The job is submitted to the cluster, where it is divided into tasks and executed across multiple nodes.
- Job progress and status can be monitored through the Hadoop web UI or command-line tools.

PART V

9. Write a program for the following:
a) Copying a local file to a Hadoop filesystem
b) fsData Output Stream

Answer:
a) Copying a Local File to HDFS:
```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class LocalToHdfsCopy {
```

```
    public static void main(String[] args) throws IOException {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        Path srcPath = new Path("localfile.txt");
        Path dstPath = new Path("/user/hadoop/hdfsfile.txt");

        fs.copyFromLocalFile(srcPath, dstPath);

        System.out.println("File copied to HDFS successfully");
    }
}
```

b) fsData Output Stream:
```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import java.io.IOException;

public class HdfsWriteExample {
    public static void main(String[] args) throws IOException {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        Path path = new Path("/user/hadoop/hdfsoutput.txt");
        FSDataOutputStream outputStream = fs.create(path);

        String data = "This is an example of writing data to HDFS";
        outputStream.writeBytes(data);
        outputStream.close();

        System.out.println("Data written to HDFS successfully");
    }
}
```

Explanation:
- **Copying a Local File to HDFS:** The program initializes a Hadoop `Configuration` object and uses the `FileSystem` class to copy a file from the local filesystem to HDFS.
- **fsData Output Stream:** The program demonstrates how to write data to a file in HDFS using the `FSDataOutputStream` class.

---

10. Write a program for the following:
a) Debugging a job
b) Application to find the maximum temperature, using a combiner function for efficiency

Answer:
a) Debugging a Job:

Debugging Hadoop jobs typically involves checking logs and using counters or setting breakpoints if running locally.

Example with Counters:
```java
import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DebugJob {
   public static class DebugMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
      public static enum COUNTERS {
         NUM_LINES
      }

      public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
         context.getCounter(COUNTERS.NUM_LINES).increment(1);
         // Map logic here
      }
   }

   public static class DebugReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
      public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
         // Reduce logic here
      }
   }

   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Debug Job");
      job.setJarByClass(DebugJob.class);
      job.setMapperClass(DebugMapper.class);
      job.setReducerClass(DebugReducer.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);
   }
}
```

Explanation:
- The `DebugMapper` class uses counters to count the number of lines processed.

- The counters help in understanding how many lines are being processed and can be useful for debugging.

b) Application to Find Maximum Temperature using a Combiner Function:
Mapper:
```java
public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private static final int MISSING = 9999;

   public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String line = value.toString();
      String year = line.substring(15, 19);
      int airTemperature;
      if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
         airTemperature = Integer.parseInt(line.substring(88, 92));
      } else {
         airTemperature = Integer.parseInt(line.substring(87, 92));
      }
      String quality = line.substring(92, 93);
      if (airTemperature != MISSING && quality.matches("[01459]")) {
         context.write(new Text(year), new IntWritable(airTemperature));
      }
   }
}
```

Combiner:
```java
public class MaxTemperatureCombiner extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
      int maxValue = Integer.MIN_VALUE;
      for (IntWritable value : values

) {
         maxValue = Math.max(maxValue, value.get());
      }
      context.write(key, new IntWritable(maxValue));
   }
}
```

Reducer:
```java
public class MaxTemperatureReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
      int maxValue = Integer.MIN_VALUE;
      for (IntWritable value : values) {
         maxValue = Math.max(maxValue, value.get());
      }
      context.write(key, new IntWritable(maxValue));
   }
}
```

Driver:
```java
public class MaxTemperature {
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Max Temperature");
      job.setJarByClass(MaxTemperature.class);
      job.setMapperClass(MaxTemperatureMapper.class);
      job.setCombinerClass(MaxTemperatureCombiner.class);
      job.setReducerClass(MaxTemperatureReducer.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);
   }
}
```

Explanation:
- The `MaxTemperatureMapper` reads and extracts the year and temperature.
- The `MaxTemperatureCombiner` performs a local reduction on the mapper output to find the maximum temperature for each year.
- The `MaxTemperatureReducer` performs the final reduction to determine the maximum temperature for each year.

Feel free to ask for further clarifications or additional questions!