CMR INSTITUTE OF TECHNOLOGY

USN | 1 | C | | | | | | | | |

Internal Assessment Test 3 – May 2019(Answer Key)

Sub:	Big Data Analytics							Code:	17MCA452
Date:	15-05-19	Duration:	90 mins	Max Marks:	50	Sem:	IV A & B	Branch:	MCA

Note: Answer any 5 questions. All questions carry equal marks. Total marks: 50

Marks

1. a **Explain the weather Data Set and in detail discuss the data format available in NCDC.** **5**

A Weather Dataset
For our example, we will write a program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semistructured and record-oriented.

Data Format
The data we will use is from the National Climatic Data Center (NCDC, http://www .ncdc.noaa.gov/). The data is stored using a line-oriented ASCII format, in which each line is a record. The format supports a rich set of meteorological elements, many of which are optional or with variable data lengths. For simplicity, we shall focus on the basic elements, such as temperature, which are always present and are of fixed width.

b **Write a Unix program to find the maximum recorded temperature by year from NCDC weather records** **5**

A program for finding the maximum recorded temperature by year from NCDC weather Records:

```
#!/usr/bin/env bash
for year in all/*
do
echo -ne `basename $year .gz`"\t"
gunzip -c $year | \
awk '{ temp = substr($0, 88, 5) + 0;
q = substr($0, 93, 1);
if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
END { print max }'
done
```

2. a **Write a Map Reduce full program to find the maximum recorded temperature by year from NCDC weather records using Java.** **10**

Mapper for maximum temperature example:

```
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 MaxTemperatureMapper
extends Mapper<LongWritable, Text, Text, IntWritable> {
private static final int MISSING = 9999;
```

```
@Override
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));
}
}
}
```

Reducer for maximum temperature example:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer
extends Reducer<Text, IntWritable, Text, IntWritable> {
@Override
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));
}
}
```

Application to find the maximum temperature in the weather dataset

```
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 MaxTemperature {
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: MaxTemperature <input path> <output path>");
System.exit(-1);
}
Job job = new Job();
job.setJarByClass(MaxTemperature.class);
job.setJobName("Max temperature");
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setMapperClass(MaxTemperatureMapper.class);
```

```
job.setReducerClass(MaxTemperatureReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

3. a **Write a mapper and reduce function for finding maximum temperature in ruby.** **5**

Map function for maximum temperature in Ruby

```
#!/usr/bin/env ruby
STDIN.each_line do |line|
val = line
year, temp, q = val[15,4], val[87,5], val[92,1]
puts "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
endputs "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
end
```

Reducer function for maximum temperature in Ruby

```
#!/usr/bin/env ruby
last_key, max_val = nil, 0
STDIN.each_line do |line|
key, val = line.split("\t")
if last_key && last_key != key
puts "#{last_key}\t#{max_val}"
last_key, max_val = key, val.to_i
else
last_key, max_val = key, [max_val, val.to_i].max
end
end
puts "#{last_key}\t#{max_val}" if last_key
```

b **Write a mapper and reduce function for finding maximum temperature in Python** **5**

Map function for maximum temperature in Python

```
#!/usr/bin/env python
import re
import sys
for line in sys.stdin:
val = line.strip()
(year, temp, q) = (val[15:19], val[87:92], val[92:93])
if (temp != "+9999" and re.match("[01459]", q)):
print "%s\t%s" % (year, temp)
```

Reduce function for maximum temperature in Python

```
#!/usr/bin/env python
import sys
(last_key, max_val) = (None, 0)
for line in sys.stdin:
(key, val) = line.strip().split("\t")
if last_key and last_key != key:
print "%s\t%s" % (last_key, max_val)
(last_key, max_val) = (key, int(val))
else:
(last_key, max_val) = (key, max(max_val, int(val)))
if last_key:
```

```
print "%s\t%s" % (last_key, max_val)
```

4. a **Explain the steps involved in setting up a Client Class Path and Task Class path while packaging a job.** **5**

The Client Class Path:

The users client-side classpath set by hadoop jar <jar> is made up of

- The job JAR file
- Any JAR files in the lib directory and the classes directory
- The class path defined by Hadoop-Classpath

The task class path;
On a cluster, map and reduce tasks run in separate JVMs and their class paths are not controlled by hadoop classpaths.

The users classpath is comprised of the following:

- The Job JAR file
- Any JAR files contained in the lib directory of the jobJAR files.
- Any files added to the distributed cache, using the –libjars or addFileToClassPath()

b **How will you Launch a job and retrieve the results while packaging a job.** **5**

To launch the job, we need to run the driver, specifying the cluster that we want to run the job on with the -conf option (we could equally have used the -fs and -jt options):
% hadoop jar hadoop-examples.jar v3.MaxTemperatureDriver -conf conf/hadoop-cluster.xml \
input/ncdc/all max-temp

Retrieving the Results
Once the job is finished, there are various ways to retrieve the results. Each reducer produces one output file, so there are 30 part files named part-r-00000 to partr-00029 in the max-temp directory. This job produces a very small amount of output, so it is convenient to copy it from HDFS to our development machine. The -getmerge option to the hadoop fs command is useful here, as it gets all the files in the directory specified in the source pattern and merges them into a single file on the local filesystem:

5. a **Explain the MapReduce web UI.** **10**

The MapReduce Web UI

Hadoop comes with a web UI for viewing information about your jobs. It is useful for following a job's progress while it is running, as well as finding job statistics and logs after the job has completed. You can find the UI at *http://resource-manager-host:8088/*.

ip-10-250-110-47 Hadoop Map/Reduce Administration

Quick Links

State: RUNNING
Started: Sat Apr 11 08:11:53 EDT 2009
Version: 0.20.0, r763504
Compiled: Thu Apr 9 05:18:40 UTC 2009 by ndaley
Identifier: 200904110811

Cluster Summary (Heap Size is 53.75 MB/888.94 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
53	30	2	11	88	88	16.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0002	NORMAL	root	Max temperature	47.52%	101	48	15.25%	30	0	NA

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0001	NORMAL	gonzo	word count	100.00%	14	14	100.00%	30	30	NA

Failed Jobs

none

Local Logs

Log directory, Job Tracker History

Hadoop, 2009.

The MapReduce job page

Clicking on the link for the "Tracking UI" takes us to the application master's web UI (or to the history page if the application has completed). In the case of MapReduce, this takes us to the job page, illustrated in Figure 6-2.

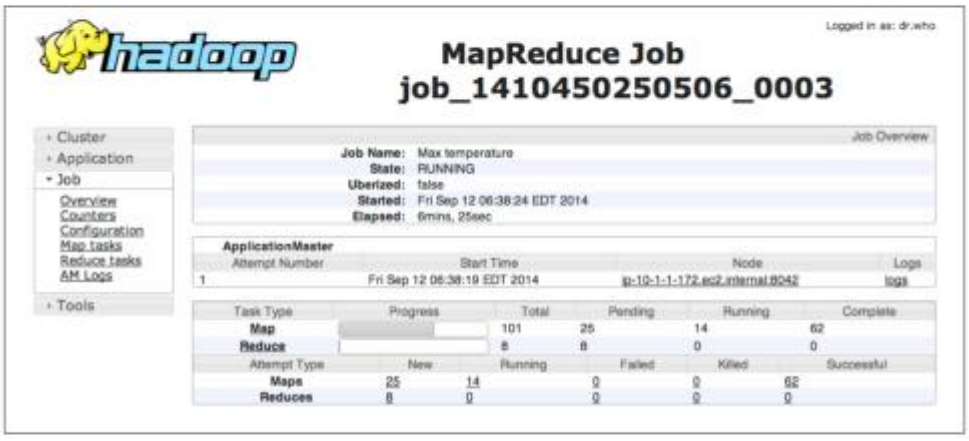

Figure 6-2. Screenshot of the job page

While the job is running, you can monitor its progress on this page. The table at the bottom shows the map progress and the reduce progress. "Total" shows the total number of map and reduce tasks for this job (a row for each). The other columns then show the state of these tasks: "Pending" (waiting to run), "Running," or "Complete" (successfully run).

6 a **Explain the Hadoop Logs with primary audience and description** 5

Table 5-2. Types of Hadoop logs

Logs	Primary audience	Description	Further information
System daemon logs	Administrators	Each Hadoop daemon produces a logfile (using log4j) and another file that combines standard out and error. Written in the directory defined by the HADOOP_LOG_DIR environment variable.	"System logfiles" on page 307 and "Logging" on page 349.
HDFS audit logs	Administrators	A log of all HDFS requests, turned off by default. Written to the namenode's log, although this is configurable.	"Audit Logging" on page 344.

Logs	Primary audience	Description	Further information
MapReduce job history logs	Users	A log of the events (such as task completion) that occur in the course of running a job. Saved centrally on the jobtracker, and in the job's output directory in a *_logs/history* subdirectory.	"Job History" on page 166.
MapReduce task logs	Users	Each tasktracker child process produces a logfile using log4j (called *syslog*), a file for data sent to standard out (*stdout*), and a file for standard error (*stderr*). Written in the *userlogs* subdirectory of the directory defined by the HADOOP_LOG_DIR environment variable.	This section.

b **Explain remote debugging with various options.** **5**

`mapred.child.java.opts`. These are explained in more detail in "Memory" on page 305.

Use task profiling

Java profilers give a lot of insight into the JVM, and Hadoop provides a mechanism to profile a subset of the tasks in a job. See "Profiling Tasks" on page 177.

Use `IsolationRunner`

Older versions of Hadoop provided a special task runner called `IsolationRunner` that could rerun failed tasks in situ on the cluster. Unfortunately, it is no longer available in recent versions, but you can track its replacement at *https://issues.apache.org/jira/browse/MAPREDUCE-2637*.

Reproduce the failure locally

Often the failing task fails consistently on a particular input. You can try to reproduce the problem locally by downloading the file that the task is failing on and running the job locally, possibly using a debugger such as Java's VisualVM.

Use JVM debugging options

A common cause of failure is a Java out of memory error in the task JVM. You can set `mapred.child.java.opts` to include `-XX:-HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/path/to/dumps` to produce a heap dump which can be examined afterwards with tools like *jhat* or the Eclipse Memory Analyzer. Note that the JVM options should be added to the existing memory settings specified by

7. a **Explain the difference between old and New APIs used in Java Map reduce** **10**

There are several notable differences between the two APIs:
• The new API favors abstract classes over interfaces, since these are easier to evolve.
• The new API is in the org.apache.hadoop.mapreduce package (and subpackages). The old API can still be found in org.apache.hadoop.mapred.
• The new API makes extensive use of context objects that allow the user code to communicate with the MapReduce system. The new Context, for example, essentially unifies the role of the JobConf, the OutputCollector, and the Reporter from the old API.
• In both APIs, key-value record pairs are pushed to the mapper and reducer, but in addition, the new API allows both mappers and reducers to control the execution flow by overriding the run() method. For example, records can be processed in batches, or the execution can be terminated before all the records have been processed. In the old API this is possible for mappers by writing a MapRunnable, but no equivalent exists for reducers.
• Configuration has been unified. The old API has a special JobConf object for job configuration, which is an extension of Hadoop's vanilla Configuration object (used for configuring daemons; see "The Configuration API" on page 146). In the new API, this distinction is dropped, so job configuration is done through a Configuration.
• Job control is performed through the Job class in the new API, rather than the old

JobClient, which no longer exists in the new API.
Output files are named slightly differently: in the old API both map and reduce outputs are named part-nnnnn, while in the new API map outputs are named partm-nnnnn, and reduce outputs are named part-r-nnnnn (where nnnnn is an integer designating the part number, starting from zero).
• User-overridable methods in the new API are declared to throw java.lang.InterruptedException. What this means is that you can write your code to be reponsive to interupts so that the framework can gracefully cancel long-running operations if it needs to3.
• In the new API the reduce() method passes values as a java.lang.Iterable, rather than a java.lang.Iterator (as the old API does). This change makes it easier to iterate over the values using Java's for-each loop construct:
for (VALUEIN value : values) { ... }

8. a **Write a Unit Test framework for Mapper class.** **5**

```
@Test
public void ignoresMissingTemperatureRecord() throws IOException,
    InterruptedException {
  MaxTemperatureMapper mapper = new MaxTemperatureMapper();

  Text value = new Text("0043011990999991950051518004+68750+023550FM-12+0382" +
                        // Year ^^^^
      "99999V0203201N00261220001CN9999999N9+99991+99999999999");
                      // Temperature ^^^^^
  MaxTemperatureMapper.Context context =
    mock(MaxTemperatureMapper.Context.class);
```

Example 5-4. Unit test for MaxTemperatureMapper

```
import static org.mockito.Mockito.*;

import java.io.IOException;
import org.apache.hadoop.io.*;
import org.junit.*;

public class MaxTemperatureMapperTest {

  @Test
  public void processesValidRecord() throws IOException, InterruptedException {
    MaxTemperatureMapper mapper = new MaxTemperatureMapper();

    Text value = new Text("0043011990999991950051518004+68750+023550FM-12+0382" +
                          // Year ^^^^
        "99999V0203201N00261220001CN9999999N9-00111+99999999999");
                        // Temperature ^^^^^
    MaxTemperatureMapper.Context context =
      mock(MaxTemperatureMapper.Context.class);

    mapper.map(null, value, context);

mapper.map(null, value, context);

verify(context, never()).write(any(Text.class), any(IntWritable.class));
}
```

b **How will you manage configuration in Map Reduce? Explain it with program.** **5**

The *hadoop-local.xml* file contains the default Hadoop configuration for the default filesystem and the jobtracker:

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.default.name</name>
    <value>file:///</value>
  </property>

  <property>
    <name>mapred.job.tracker</name>
    <value>local</value>
  </property>

</configuration>
```

The settings in *hadoop-localhost.xml* point to a namenode and a jobtracker both running on localhost:

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost/</value>
  </property>

  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>

</configuration>
```