CMR Institute of Technology, Bangalore
DEPARTMENT OF MASTER OF COMPUTER APPLICATION
III - INTERNAL ASSESSMENT

Semester: 4-CBCS 2018 Date: 16 Jun 2021
Subject: BIG DATA ANALYTICS (18MCA454)
Faculty: Ms Gomathi T Max Marks: 50

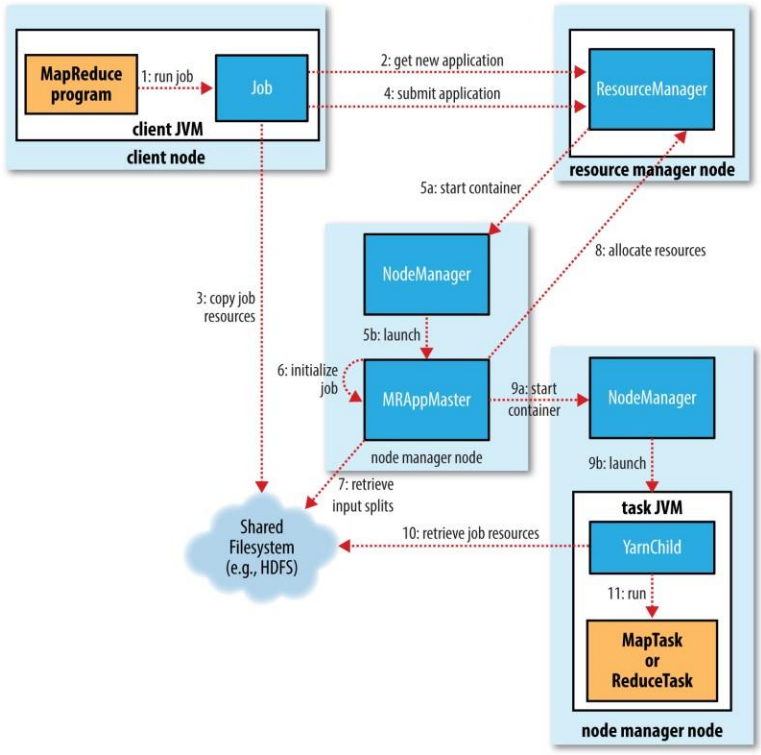
Answer Key

1.**What is Map Reduce? Sketch a neat diagram and explain the logical data flow in Map Reduce?**

Map Reduce:
MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a map procedure, which performs filtering and sorting, and a reduce method, which performs a summary operation.

- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits, because time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
  In the event of node failure before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.

- Reduce task don't work on the concept of data locality. Output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
- On this machine the output is merged and then passed to the user defined reduce function.
- Unlike to the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output



2**. Discuss the data format of Weather Dataset and write a Unix code to retrieve the maximum temperature.**

Data Format

- NCDC data (National Climatic Data Center)
- The data is stored using a line-oriented ASCII format, in which each line is a record
- Focus is on basic elements such as temperature
- Data files are organized by date and weather station

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

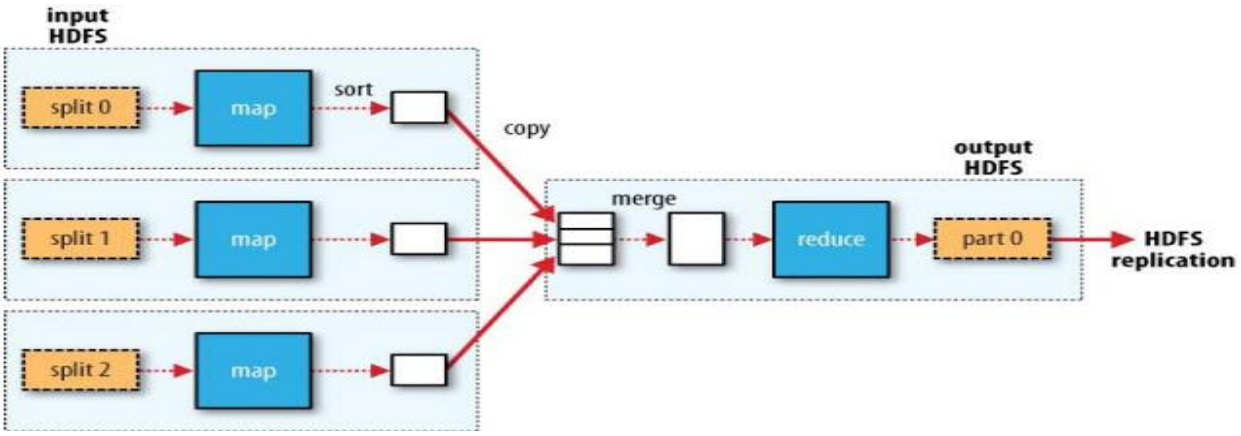
Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/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
```

3**. How does a Map reduce model works with a single reduce task? Explain with a neat diagram.**

A **Map Task** is a single instance of a MapReduce app. These tasks determine which records to process from a data block. The input data is split and analyzed, in parallel, on the assigned compute resources in a Hadoop cluster. This step of a MapReduce job prepares the <key, value> pair output for the reduce step.

A **Reduce Task** processes an output of a map task. Similar to the map stage, all reduce tasks occur at the same time, and they work independently. The data is aggregated and combined to deliver the desired output. The final result is a reduced set of <key, value> pairs which MapReduce, by default, stores in HDFS.



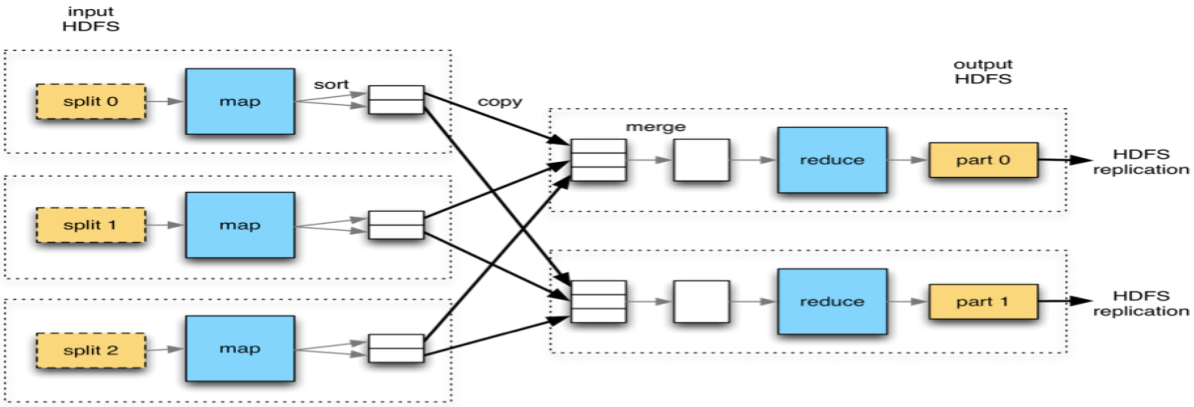
The Map and Reduce stages have two parts each.

The **Map** part first deals with the **splitting** of the input data that gets assigned to individual map tasks. Then, the **mapping** function creates the output in the form of intermediate key-value pairs.

The **Reduce** stage has a shuffle and a reduce step. **Shuffling** takes the map output and creates a list of related key-value-list pairs. Then, **reducing** aggregates the results of the shuffling to produce the final output that the MapReduce application requested.

4. How does a Map reduce model works with a multi- reduce tasks? Explain with a neat diagram

- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - Map stage − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - Reduce stage − This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



5. Explain in detail the steps involved in running the map reduce program in a cluster.

- Packaging
- Launching a job
- The map reduce web UI
- Retrieving the results
- Debugging a job

Packaging

- The program need not be modified to run on a cluster.
- The programs have to be packages as JAR files.
- It is done through Ant tool

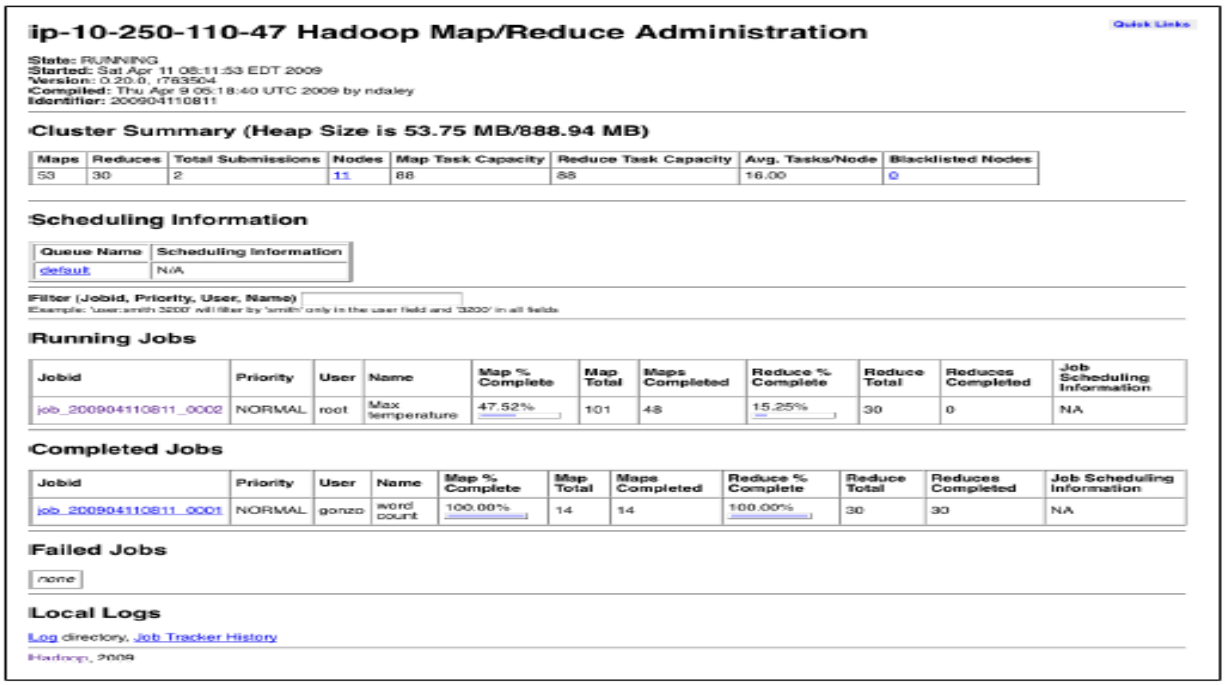
```
<jar
destfile="hadoop-examples.jar" basedir="${classes.dir}"/>
```

Launching a job

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
```

The map reduce web UI



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

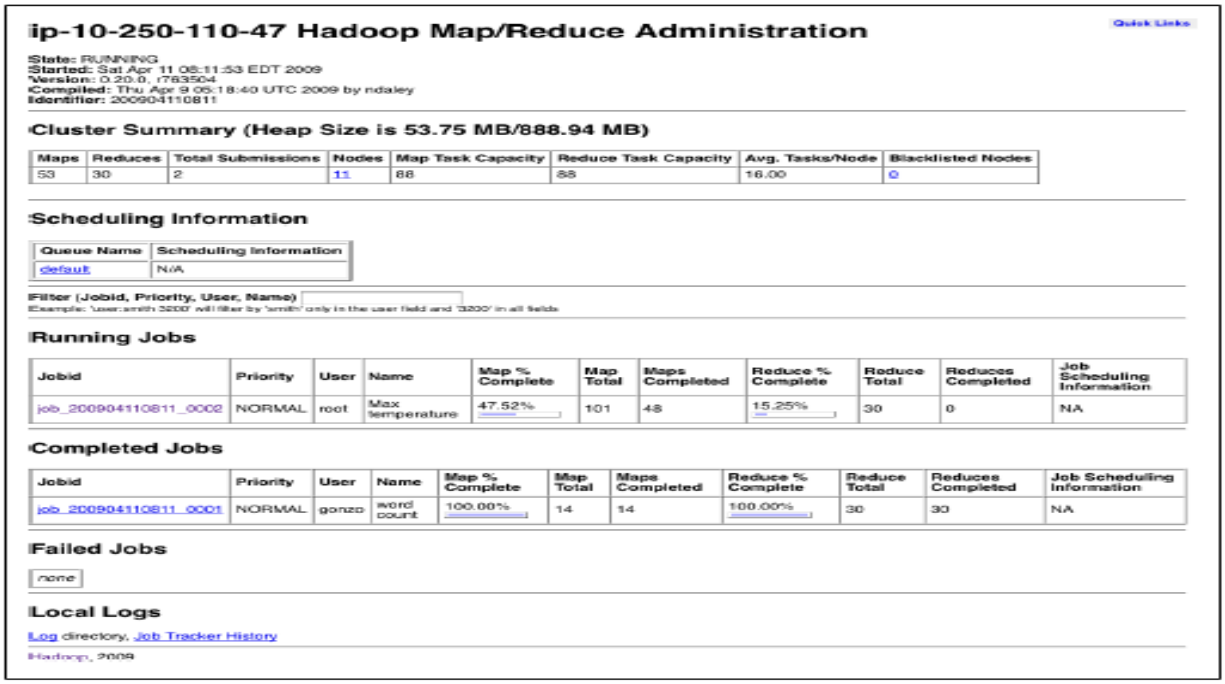
Debugging a Job:

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

  parser.parse(value);
  if (parser.isValidTemperature()) {
    int airTemperature = parser.getAirTemperature();
    if (airTemperature > 1000) {
      System.err.println("Temperature over 100 degrees for input: " + value);
      context.setStatus("Detected possibly corrupt record: see logs.");
      context.getCounter(Temperature.OVER_100).increment(1);
    }
    context.write(new Text(parser.getYear()), new IntWritable(airTemperature));
  }
```

6. Write a short note on: MapReduceUI and Hadoop Logs

MapReduceUI:



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, r783504
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

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.

7. Write a Java Map Reduce code to find maximum temperature from the weather data set

Example 2-3. 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));
    }
  }
}
```

Example 2-4. 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));
  }
}
```

Example 2-5. 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);
  }
}
```

8**. What are Hadoop pipes? Demonstrate the concept of Hadoop pipes with a program.**

Hadoop Pipes is the name of the C++ interface to Hadoop MapReduce. Unlike Streaming, which uses standard input and output to communicate with the map and reduce code, Pipes uses sockets as the channel over which the task tracker communicates with the process running the C++ map or reduce function. JNI is not used.

```
#include <algorithm>
#include <limits>
#include <stdint.h>
#include <string>
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"
class MaxTemperatureMapper : public HadoopPipes::Mapper {
public:
MaxTemperatureMapper(HadoopPipes::TaskContext& context) {
}
void map(HadoopPipes::MapContext& context) {
std::string line = context.getInputValue();
std::string year = line.substr(15, 4);
std::string airTemperature = line.substr(87, 5);
std::string q = line.substr(92, 1);
if (airTemperature != "+9999" &&
(q == "0" || q == "1" || q == "4" || q == "5" || q == "9")) {
context.emit(year, airTemperature);
}
}
};
class MapTemperatureReducer : public HadoopPipes::Reducer {
public:
```

```
MapTemperatureReducer(HadoopPipes::TaskContext& context) {
}
void reduce(HadoopPipes::ReduceContext& context) {
int maxValue = INT_MIN;
while (context.nextValue()) {
maxValue = std::max(maxValue, HadoopUtils::toInt(context.getInputValue()));
}
context.emit(context.getInputKey(), HadoopUtils::toString(maxValue));
}
};
int main(int argc, char *argv[]) {
return HadoopPipes::runTask(HadoopPipes::TemplateFactory<MaxTemperatureMapper,
MapTemperatureReducer>());
}
```

9. **Explain the concept of Hadoop streaming and write a python code to find the maximum temperature from the weather dataset.**

Example 2-10. 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)
```

Example 2-11. 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)
```

10. Explain in detail the configuration file, its APIs and access Properties.

- Configuration are specified by resources.
- A resource contains a set of name/ value pairs as xml data.
- Each resource is named by either a string or a path.
- If named as string, then the class path is examined for a file with that name.
- If named by a path, then the local file system is examined.
- Core-default.xml – Read-only defaults for hadoop.
- Core-site.xml – site specific configuration for a given hadoop installation.

Example 5-1. A simple configuration file, configuration-1.xml

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>color</name>
    <value>yellow</value>
    <description>Color</description>
  </property>

  <property>
    <name>size</name>
    <value>10</value>
    <description>Size</description>
  </property>

  <property>
    <name>weight</name>
    <value>heavy</value>
    <final>true</final>
    <description>Weight</description>
  </property>

  <property>
    <name>size-weight</name>
    <value>${size},${weight}</value>
    <description>Size and weight</description>
  </property>
</configuration>
```

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
assertThat(conf.get("color"), is("yellow"));
assertThat(conf.getInt("size", 0), is(10));
assertThat(conf.get("breadth", "wide"), is("wide"));
```