95-702 Distributed Systems Due Friday April 18, 11:59PM

Project 5 Programming Map Reduce

The purpose of this project is to introduce the student to Big Data, Hadoop (HDFS), and MapReduce.

The code and example data for the MaxTemperature application is from "Hadoop: The Definitive Guide, Second Edition, by Tom White. Copyright 2011 Tom White, 978-1-449-38973-4."

We will be running Linux, Centos 6.3, on a Hadoop cluster of 24 physical machines on a single rack at Heinz College.

This file is in plain text so that you can easily copy lines from this file into the Linux shell command line.

To reach the hadoop cluster, you will need to ssh into heinz-jumbo.heinz.cmu.local.

If you want to work from home, first run the Cisco AnyConnect Secure Mobility Client.
This tool may be downloaded from here:

https://www.cmu.edu/computing/software/all/cisco-anyconnect/index.html

You should spend a little time learning vi. This is an editor that you will find useful when working on Linux. Some useful vi commands are:

To start vi, enter vi filename.
To quit vi without saving esc : q!
To quit vi with saving esc: wq
To save the current file without quitting esc : w

To move the curser down use j.
To move the curser up use k.
To move the curser left use h.
To move the curser right use l.

To append after the curser use a.
To insert before the curser use i.
To exit insert mode use esc :.
To perform a global replacement in vi, use esc : %s/xyz/abc/g.

Your home directory is /home/userID.

The hadoop jars and binaries are at /usr/local/hadoop.
Your HDFS file system directories are at /user/userID/output.

Note that one is "usr" and the other is "user".

It is assumed that students with Windows machines will be running putty.
It is also assumed that each student has a user-ID and password for jumbo.

You should have established a new password when you completed the Hadoop lab. Your user-ID should be on the grade book on Blackboard. You should communicate your password to the TA's by taking the quiz which asks for your password.

If you need to transfer data from your machine to jumbo, be sure to use sftp. The "copy and paste" approach may introduce hidden characters in your file. You will need to sftp your file to jumbo. Here is an example execution of sftp.

```
$sftp userID@heinz-jumbo.heinz.cmu.local
sftp>put MaxTemperature.java
sftp>get MaxTemperature.java
```

Basic commands that you will need to study before attempting to work on the cluster:
==========

The user would like to see a list of files on the Hadoop distributed file system under /user/userID/input.

Note the use of "user" and not "usr". "user" is a reference to the directory in HDFS. "usr" is a reference to your local directory.

```
$hadoop dfs -ls /user/userID/input/
```

An input file needs to be placed under HDFS. This file will be used for subsequent map reduce processing.
The HDFS file and directory are created. If they already exist, then this command will return an 'already exists' message.

```
$hadoop dfs -copyFromLocal /home/userID/input/1902.txt /user/userID/input/1902.txt
```

Remove a directory of Java classes that may contain Java packages.

```
$rm -r temperature_classes
```

Create a directory for Java classes.

```
$mkdir temperature_classes
```

Compile three Java classes using a library of Hadoop classes stored in a jar. The classes directory (./temperature_classes) is also consulted during the compile. The classes directory (./temperature_classes) is the target of the compile and may be populated with a a directory structure that corresponds to Java packages.

These commands assume that you are in a directory just above temperature_classes.

```
$javac -classpath  /usr/local/hadoop/hadoop-core-1.1.2.jar:./temperature_classes -d temperature_classes MaxTemperatureMapper.java

$javac -classpath  /usr/local/hadoop/hadoop-core-1.1.2.jar:./temperature_classes -d temperature_classes MaxTemperatureReducer.java

$javac -classpath  /usr/local/hadoop/hadoop-core-1.1.2.jar:./temperature_classes -d temperature_classes MaxTemperature.java
```

Remove an existing jar file.

```
$rm temperature.jar
```

Create a new jar file called temperature.jar. It will include all of the classes found in temperature_classes.

```
$jar -cvf temperature.jar -C  temperature_classes/  .
```

Remove the output directory from the distributed file system.

```
$hadoop dfs -rmr /user/userID/output
```

Copy a file from the Hadoop Distributed File system to the client.

```
hadoop dfs -getmerge /user/userID/output aCoolLocalDirectory
```

You may view what jobs are running on the cluster
with the command:

```
hadoop job -list
```

Kill a job that is not making progress:

```
bin/hadoop job -kill job_201310251241_0754
```

You will need your own Job ID.

Execute a map reduce job on the cluster of machines. The file temperature.jar holds the map reduce code. Next,
a path to the class with a main routine is provided. Then, the input and output is specified. The input is a file
that must exist on the distributed file system and the output is a directory that will be created. It must
not exist before running the command or you will receive an exception.

```
$ hadoop jar /home/userID/temperature.jar edu.cmu.andrew.mm6.MaxTemperature
/user/userID/input/combinedYears.txt /user/userID/output
```

We wish to get a copy of the content on the output directory.

```
$mkdir coolProjectOutput
```

```
$hadoop dfs -getmerge /user/userID/output ~/coolProjectOutput/
```

```
$cat ~/coolProjectOutput/output
```

Note that the code below is defined within Java packages. So, when you compile the source code
the compiled (.class files) will be placed within directories and sub directories.

Task 0
======

Compile and execute a MapReduce job developed from WordCount.java. The file WordCount.java is found under
/home/public/. The code is also available at the bottom of this document. Copy it to your home directory.
Compile it and generate a jar file called wordcount.jar. Deploy the jar file and test it against
/home/public/words.txt. The file words.txt will need to be copied to HDFS using hadoop's copyFromLocal option.
The output should be left in your /home/userID/Task0Output file.

Task 1
======

Build and deploy a MapReduce application called TotalWords.java and place it into a jar file called
total_words.jar. This program will compute the total number of words in words.txt. The output should be left
in your /home/userID/Task1Output directory. Note that CountWords.java uses a call to nextToken() on the
iterator. Without this call, the program will enter an infinite loop.

Task 2
======

Modify the code in Task 1 so that it reads the words.txt file and computes the number of words in
the file that contain the substring "ei". Note: we are counting words and not the number of ei
substrings.

Call this application TotalEISubstring.java and place it in the jar file called totaleisubstring.jar. The output should be left in your /home/userID/Task2Output directory.

Task 3
======

In this task, we will use a fairly large dataset from Tom White's book.
The data file, combinedYears.txt, contains thousands of temperature readings (in Celsius times 10)
in the USA by year from 1901 to 1902. The year is specified in positions 15-18. The temperature
is specified beginning at column 87 and begins with a plus or minus character. Four digits are
used for the temperature. Be sure to study the code to see how it references these locations.

Below are three files: MaxTemperature.java, MaxTemperatureMapper.java and MaxTemperatureReducer.java.
These files may be found at /home/public. Copy these files to your home directory.

Run this application against the data set under /home/public/combinedYears.txt. Your jar file will
be named maxtemperature.jar. The output should be left in your /home/userID/Task3Output directory.

Task 4
======

Modify the code from Task 3 and build a min temperature application. Note that the temperatures in this file
are degrees Celsius * 10. Your jar file will be named mintemperature.jar. The output should be left
in your /home/userID/Task4Output directory.

Task 5
======

Within the /home/public directory, there is a file called P1V.txt.

P1V.txt is a tab delimited text file with a individual offense incidents
from January 1990 through December 1999 for serious violent crimes (FBI Part 1) in Pittsburgh.

```
The first two columns (X,Y) represent State Plane (projected, rectilinear)
coordinates (measured
in feet) specifying the location of the crime.
The third column is the time.
The fourth column is a street address.
The fifth column is the type of offense (Robbery, Rape, Etc.)
The sixth column is the date.
The seventh column is the 2000 census tract.

Write a MapReduce application that finds the total number of each crime by offense.
That is, we want to know the total number of aggravated assaults, robberies, rapes
and so on.

Your jar file will be named pittsburghcrimestats.jar. The output should be left
in your /home/userID/Task5Output directory.

This Task requires that you document your code.

Task 6
======

Modify your solution to Task 5 so that it finds the total number of crimes that
occurred within
1000 feet of 3803 Forbes Avenue in Oakland. This location has the (X,Y) coordinates
of
(1354326.897,411447.7828). Use these coordinates and the Pythagorean theorem to
decide if a crime
occurred within 1000 feet of 3803 Forbes Avenue.

Your jar file will be named oaklandcrimestats.jar. The output should be left
in your /home/userID/Task6Output directory.

This Task requires that you document your code.

Summary
=======

Submit a paragraph to Blackboard that says that you have completed work on this
assignment.
After the submission of the paragraph, do not alter your files on the cluster. The
graders
will be looking at the timestamps. The graders will examine your /home/userID/
directory.
All of your source code (.java files) will be at this top level directory. There
will be
seven different 'classes' directories containing compiled code (and packages). There
will
be seven different output directories. The grader will look over your source code
and check
the output directories.

For Tasks 5 and 6, the grader will be looking for clear documentation.

In my solutions, I have found it best to use Java packages.

The commands provided above were all tested on jumbo.


// ====================== WordCount.java ========================================
package org.myorg;
```

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.*;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.*;
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;


public class WordCount extends Configured implements Tool {

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

                @Override
                public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException
                {
                        String line = value.toString();
                        StringTokenizer tokenizer = new StringTokenizer(line);
                        while(tokenizer.hasMoreTokens())
                        {
                                word.set(tokenizer.nextToken());
                                context.write(word, one);
                        }
                }
        }

        public static class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable>
        {
                public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException
                {
                        int sum = 0;
                        for(IntWritable value: values)
                        {
                                sum += value.get();
                        }
                        context.write(key, new IntWritable(sum));
                }

        }

        public int run(String[] args) throws Exception  {

                Job job = new Job(getConf());
                job.setJarByClass(WordCount.class);
                job.setJobName("wordcount");
```

```
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(WordCountMap.class);
                job.setCombinerClass(WordCountReducer.class);
                job.setReducerClass(WordCountReducer.class);


                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);


                FileInputFormat.setInputPaths(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                boolean success = job.waitForCompletion(true);
                return success ? 0: 1;
        }


        public static void main(String[] args) throws Exception {
                // TODO Auto-generated method stub
                int result = ToolRunner.run(new WordCount(), args);
                System.exit(result);
        }

}


// ============== MaxTemperature.java ================================
        package edu.cmu.andrew.mm6;
        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 MaxTemperatureMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
                private static final int MISSING = 9999;
                public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws                  IOException {

                        // Get line from input file. This was passed in by Hadoop as
value.
                        // We have no use for the key (file offset) so we are
ignoring it.

                        String line = value.toString();

                        // Get year when weather data was collected. The year is in
positions 15-18.
                        // This field is at a fixed position within a line.
                        String year = line.substring(15, 19);

                        // Get the temperature too.

                        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));
                        }

                        // Get quality of reading. If not missing and of good
quality then
                        // produce intermediate (year,temp).

                        String quality = line.substring(92, 93);
                        if (airTemperature != MISSING && quality.matches("[01459]"))
{

                             // for each year in input, reduce will be called with
                             // (year,[temp,temp,temp,â€¦])
                             // They key is year and the list of temps will be
placed in an iterator.

                                output.collect(new Text(year), new
IntWritable(airTemperature)); }
                        }
         }


========== MaxTemperatureReducer.java
==================================================
        package edu.cmu.andrew.mm6;
        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 MaxTemperatureReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {

                public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text,
                                        IntWritable> output, Reporter reporter) throws
IOException {

                        // from the list of values, find the maximum
                        int maxValue = Integer.MIN_VALUE;
                        while (values.hasNext()) {
                             maxValue = Math.max(maxValue, values.next().get());
                        }
                        // emit (key = year, value = maxTemp = max for year)
                        output.collect(key, new IntWritable(maxValue));
                }
        }

// ======= And, to get it all running and tied together: MaxTemperature.java
============

        package edu.cmu.andrew.mm6;
```

```
        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;

        public class MaxTemperature {
                public static void main(String[] args) throws IOException {
                        if (args.length != 2) {
                                System.err.println("Usage: MaxTemperature <input
path> <output path>");
                                System.exit(-1);
                        }
                        JobConf conf = new JobConf(MaxTemperature.class);
                        conf.setJobName("Max temperature");
                        FileInputFormat.addInputPath(conf, new Path(args[0]));
                        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
                        conf.setMapperClass(MaxTemperatureMapper.class);
                        conf.setReducerClass(MaxTemperatureReducer.class);
                        conf.setOutputKeyClass(Text.class);
                        conf.setOutputValueClass(IntWritable.class);
                        JobClient.runJob(conf);
                }
        }
```