18.337/6.338 Homework 2
Fall 2011

Due: Wednesday, October 12, 2011
You can email your work to the TA, or submit it in class, whichever you prefer.

In this assignment you will implement a statistical analysis using the MapReduce paradigm with Hadoop, and run your analysis at a reasonable scale on a cluster of Amazon EC2 instances.

The first two sections of this document explain how to use the tools you will need, and the second two sections have tasks to perform.

1 Running Hadoop clusters

To use Hadoop, first log in to the class AWS Management Console. You will see a running instance named `hadoop-launcher`. You can connect to this machine the same way you connected to your instances in homework 1. From there, you can run a script to create your very own Hadoop cluster, as follows:

```
$ cd ~/hadoop/ec2
$ bin/hadoop-ec2 launch-cluster NAME NNODES
```

Replace `NAME` with a unique name, and `NNODES` with the number of nodes you want. For developing and debugging, 1 node is sufficient. From there, you can scale to perhaps 16 nodes without worrying too much. With 32 nodes or more, you will need to watch the clock and make sure you don't leave instances running. At that scale, you may also begin to encounter problems.

The `launch-cluster` command takes a couple minutes. During the process, it will print out the address of the master node, which you will want to take note of. When the cluster is ready, you can connect to the master node using

```
$ bin/hadoop-ec2 login NAME
```

The master node is used to start jobs.

When you're done working for the day, shut down your cluster using

```
$ bin/hadoop-ec2 terminate-cluster NAME
```

WARNING: Hadoop cluster instances are not EBS-backed, so terminating them discards all data on the master and worker nodes. Be sure to keep a copy of your source code and any data you need somewhere else.

Ground rules: `hadoop-launcher` will be on through the due date. Do not stop or terminate it. If you want to store data on it (slightly risky), make a directory for yourself. Do not connect to other people's clusters.

2 Running MapReduce jobs

There are two good options for how to write your MapReduce jobs: in Java using Hadoop's (native) Java API, or in any programming language that is able to read from standard in and write to standard out, using Hadoop Streaming. We will provide an example in Python, but you may use any language.

On the master node, Hadoop is located in `/usr/local/hadoop-0.19.0`. This path is also available as the `HADOOP_HOME` environment variable.

2.1 Using Java

Download the `WordCount.java` example from the homework page of the class website. Move the file into the home directory of your master node using `scp`, or by running `cat > WordCount.java`, pasting the contents of the file, and hitting `control-D`.

To compile, run the following commands from the home directory on the master node:

```
mkdir wc_classes

javac -classpath $HADOOP_HOME/hadoop-0.19.0-core.jar -d wc_classes WordCount.java

jar cvf wc.jar -C wc_classes .
```

To run a job, we need to copy input data into Hadoop's file system (HDFS). The input consists of a directory of text files, which you can create as follows:

```
$HADOOP_HOME/bin/hadoop dfs -mkdir /input
$HADOOP_HOME/bin/hadoop dfs -put file01 file02 ... /input
...
```

`file01` and `file02` are paths to files in the ordinary local filesystem of the machine.

Now we can actually run the job:

```
$HADOOP_HOME/bin/hadoop jar wc.jar org.myorg.WordCount /input /output
```

The last two arguments are the input path in HDFS, and the output path in HDFS. The output directory must not exist yet.

You can use the `dfs -ls` and `dfs -cat` commands to examine files in HDFS. Doing a web search for "HDFS shell guide" will provide more information on HDFS commands.

2.2 Using other languages

Download the `mapper.py` and `reducer.py` example source from the homework page of the class website. These programs solve the same word counting problem as `WordCount.java`. Run `chmod +x filename` on each to make them executable.

Copy data into HDFS exactly as in the previous section.

This command will run the job:

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-0.19.0-streaming.jar
    -D mapred.output.compress=false -file ~/mapper.py -mapper ~/mapper.py
    -file ~/reducer.py -reducer ~/reducer.py -input /input -output /output
```

Although here it is broken onto multiple lines, this is actually *one* rather long command line. The `-D` option tells Hadoop not to compress the output, which makes it human-readable.

The command is the same for any language; simply replace the `.py` file names with the names of your executables.

We recommend glancing at Michael Noll's article "Writing An Hadoop MapReduce Program In Python", linked from the readings page of the class website. There he cleverly points out that MapReduce programs written this way can be tested locally on small data using the Unix shell pipeline

```
cat data | map | sort | reduce
```

2.3 Eye candy

You can monitor the status of jobs by visiting `http://MASTER:50030` with a web browser, where `MASTER` is the address of the master node.

3 Reading

Read sections 3.1 through 3.3 of the book "Data-Intensive Text Processing with MapReduce", by Jimmy Lin and Chris Dyer, available on the course website.

(Optional) Skim chapter 2 for more background information.

4 Analyzing bigrams

This exercise is adapted from material accompanying Lin and Dyer's book. You are welcome to consult any material you want to help solve this problem. However, if you encounter a full solution to this particular problem on the internet, please do not use it.

Bigrams are sequences of two consecutive words. Understanding which bigrams occur in natural language is useful in a variety of search and textual analysis problems. We will use MapReduce first to count the unique bigrams in an input corpus, and then to compute relative frequencies (how likely you are to observe a word given the preceding word).

4.1 Counting bigrams

Modify the word count example code to count unique bigrams instead. How many unique bigrams are there? List the top ten most frequent bigrams and their counts.

4.2 Relative frequencies

Some bigrams may appear frequently simply because one of their words is especially common. We can obtain a more interesting statistic by dividing the number of occurrences of the bigram "A B" by the total number of occurrences of all bigrams starting with "A". This gives $P(B|A)$, the probability of seeing "B" given that it follows "A".

Pick a word, and show the relative frequencies of words following it. What pairs of words are especially strongly correlated, i.e. $P(B|A)$ is large?

Section 3.3 of the book discusses a technique for computing this result in a single MapReduce job, using fancy features of the framework. You do not have to use this technique, and may instead use multiple MapReduce jobs if you wish.

4.3 Obtaining input data

You can use the input data available on the homework page of the class website, or you may select your own corpus of input text. Project Gutenberg, at `http://www.gutenberg.org` is a good source of interesting texts. There is no particular requirement on the size of data to use, but it should be interestingly large, e.g. the complete novels of Charles Dickens. To make the assignment more fun, entirely at your option, you may wish to compare statistics of different authors, time periods, genres, etc.