Brian Davis
tug75085
915409443
10 April 2020

Homework 3: MapReduce

For problems 1 and 2, I summarized the deliverables in the tables below. I extensively detailed (with screenshots) on all of my work to get these deliverables in pages 3 through 12.

The source code and reducer outputs can be found in /RawTextReducers/ and /RegularExpressionReducers/ for problems 1 and 2, respectively. Each folder has its own WordCount.java MapReduce program and is documented in this report.

Problem 1: Word Count of *Pride and Prejudice*

Number of Unique Words	5th to Last Word, Number of Occurrences	1st word, Number of Occurrences
13777	yourselves, 1	“’Tis, 2

Problem 2: Count Words with Letters in *Pride and Prejudice*

Number of Unique Words	5th to Last Word, Number of Occurrences	1st word, Number of Occurrences
5999	yours, 2	A, 41

Problem 3

HDFS is a block-structured file system, which divides and stores files into blocks of predetermined size, usually 128MB. HDFS replicates each file across 3 DataNodes by default. Since there are only 3 DataNodes in this example, each DataNode will contain blocks A and B from sample.dat. The underlying process of DataNode replications commences when a client creates a file by calling create() on the distributed file system. The distributed file system will then make an RPC call to the NameNode to create a new file in the file system’s namespace, with 0 blocks initially assigned to it. The NameNode throws an IOException if it finds permission or file existence errors.

Upon successful file creation, a FSDataOutputStream is returned to the client for writing data. The data written by the client is divided into packets and placed in a data queue. The Data Streamer is responsible for asking the NameNode to allocate new blocks by picking a list of suitable DataNodes to store the replicas. DataNodes are chosen first for block A, then again for block B in this fashion. The NameNode will respond with the same 3 DataNodes each time since there are only 3 DataNodes in this HDFS.

The data streamer will stream the packets to the first DataNode in the pipeline. The DFSOutputStream will use an ack queue that is responsible for holding all of the packets that are waiting to be acknowledged by DataNodes in the pipeline. The client calls close on the stream when all file data has been written and all packets have been sent to the DataNodes. The NameNode then receives a message that the file writing process has completed. Thus, the file sample.dat will be successfully stored amongst the 3 DataNodes in blocks A and B.

Problem 5

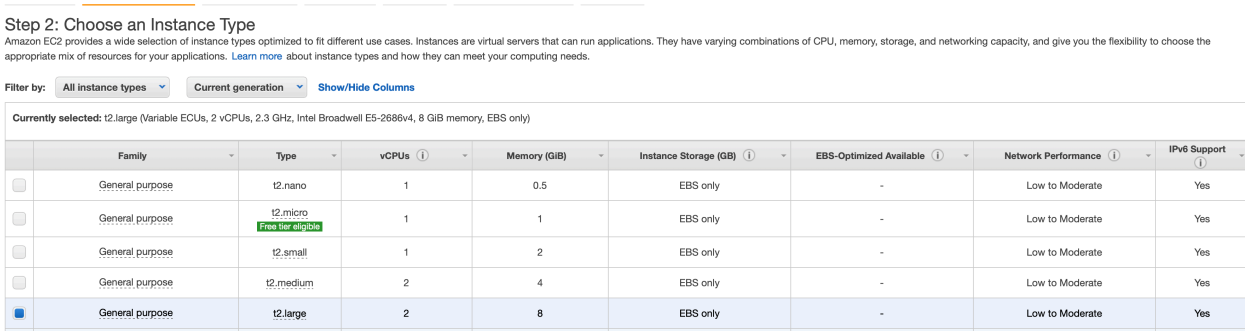
In an instance of the Quorum Consensus algorithm with a group of 5 replicas, a read quorum of size R, and a write quorum of size W:

1) If $R = 5$ and $W = 1$, then the performance for writes is improved and the replica with the highest version number is guaranteed to be selected, however this increase in performance of writes comes at the cost of a decrease in performance of reads. Reads are generally more common than writes, therefore decreases in read performance will severely decrease runtime performance. Also, this allocation of quorums is not desirable because a read cannot occur if a single replica fails, given that there are 5 total replicas and a read quorum of size $R = 5$. In addition, this allocation of quorums is not desirable because a write could occur on a single replica that fails, therefore forever losing the non-persistent state and content of the write. In order to avoid costly losses of information in the presence of replica failure, a write quorum of size $W > 1$ is preferred.
2) If $R = 1$ and $W = 5$, then the performance for reads is improved and all replicas are guaranteed to have the highest version number, however this increase in performance of reads comes at the cost of a decrease in performance of writes. Reads are generally more common than writes, therefore increases in read performance will severely increase runtime performance. However, this allocation of quorums is not desirable because a write cannot occur if a single replica fails, given that there are 5 total replicas and a write quorum of size $W = 5$. In order to avoid costly losses of information in the presence of replica failure, a write quorum of size less than the total number of replicas is preferred.
3) If $R = 3$ and $W = 3$, then the performance for reads is balanced with the performance for writes. Reads are generally more common than writes, therefore increases in read performance will severely increase runtime performance. However, this allocation of quorums is desirable because it decreases the performance for reads in order to provide reasonable availability for writes. Since the size of readers and size writers are both greater than half of the number of quorums, then reads are guaranteed to have the highest version number. This tradeoff is also necessary in order to ensure availability through replica failures. Even if 2 out of 5 replicas fail, the consensus algorithm will prevail, which makes this network robust to failures. In comparison to the $R = 1$ and $W = 5$ option above, this allocation of quorums is superior because it only decreases the read performance by the addition of 2 replicas, while increasing the write performance by the subtraction of 2 replicas and ensuring availability through 2 replica failures. In comparison to the $R = 5$ and $W = 1$ option above, this allocation of quorums is superior because it only decreases the write performance by the addition of 2 replicas, while increasing the read performance by the subtraction of 2 replicas and ensuring availability through 2 replica failures.

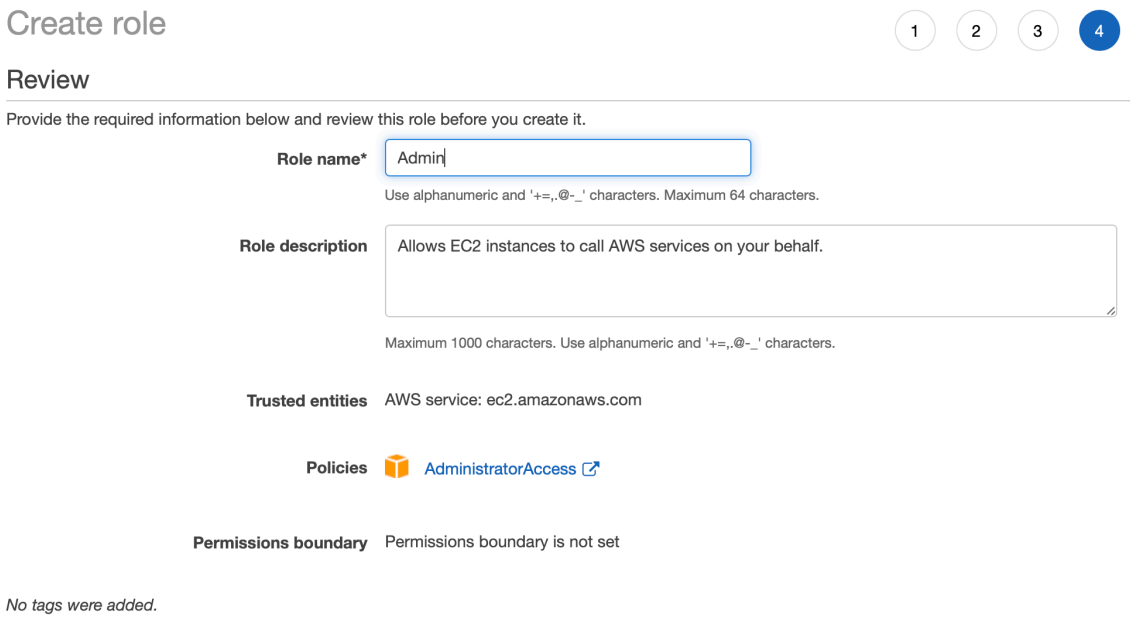
EC2 Creation

First, I chose the AMI type Ubuntu Server because I was able to find success setting up Hadoop on a local virtual machine running Ubuntu previously. The AMI type is seen below,

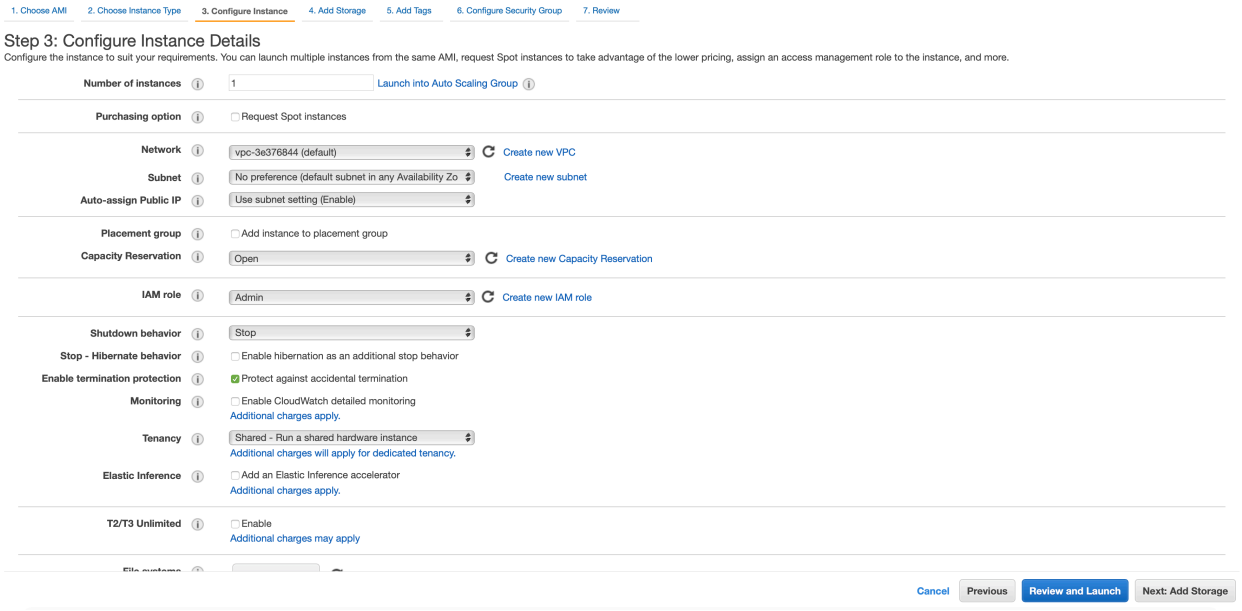
Next, I chose the instance type t2.large, which has 8 GiB of memory, in comparison to 2 GiB on the t2.micro. This is necessary in order to ensure Hadoop and MapReduce can run smoothly, as I had limited memory issues on my local environment previously. The instance type is seen below,



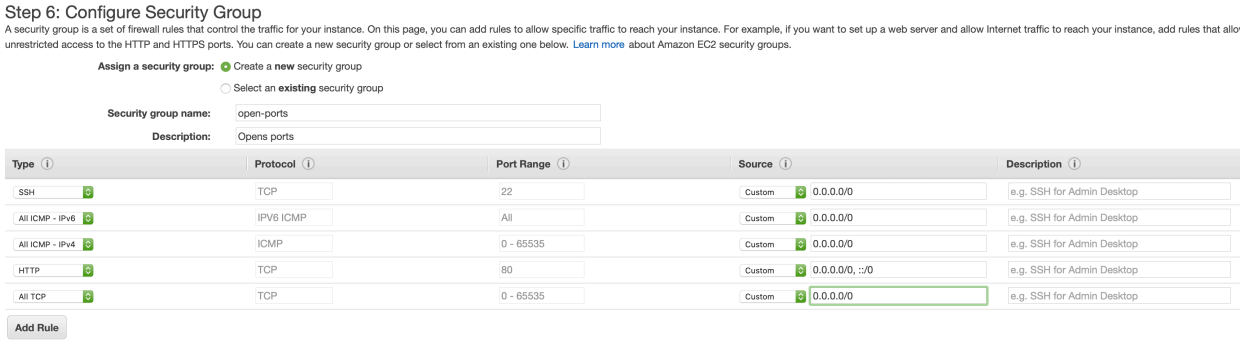
I created an IAM Role that allows Administrator Access to all AWS services, as seen below,



In the Configure Instances tab of the EC2 creation, I selected Administrator Access to all AWS services and to protect against accidental termination, as shown below,



In the Configure Security Group tab of the EC2 creation, I selected all ICMP and TCP ports so that the Hadoop webserver can work properly with limited flaws, as seen below,



Setting up Hadoop on Ubuntu EC2 Instance

First, we must install Java and Hadoop. We gather the tar files and untar them as seen below,

```
[[ec2-user@ip-172-31-84-98 ~]$ ls
 hadoop-2.7.3  hadoop-2.7.3.tar.gz  jdk1.8.0_131  jdk-8u131-linux-x64.tar.gz
```

First, we have to set environment variables in the ~/.bashrc, as seen below,

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions

# Set HADOOP HOME

export HADOOP_HOME=$HOME/hadoop-2.7.3
export HADOOP_CONF_DIR=$HOME/hadoop-2.7.3/etc/hadoop
export HADOOP_MAPPED_HOME=$HOME/hadoop-2.7.3
export HADOOP_COMMON_HOME=$HOME/hadoop-2.7.3
export HADOOP_HDFS_HOME=$HOME/hadoop-2.7.3
export YARN_HOME=$HOME/hadoop-2.7.3
export PATH=$PATH:$HOME/hadoop-2.7.3/bin

# Set JAVA HOME

export JAVA_HOME=/home/ec2-user/jdk1.8.0_131
export PATH=/home/ec2-user/jdk1.8.0_131/bin:$PATH
```

Next we activate the environment,

```
[[ec2-user@ip-172-31-84-98 ~]$ source ~/.bashrc
```

Then we ensure that ssh permissions for localhost are authorized,

```
root@ip-172-31-89-187:/home/ubuntu# ssh-keygen -t rsa -P ''
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Cl6x6QXYhjSWgfV2JxVeP/U37XOK8BclgHrKLExRXps root@ip-172-31-89-187
The key's randomart image is:
+---[RSA 2048]----+
|   o*o .. +oo    .|
|  .o.*.. +.+ o .o|
|    o B.+.E   +.=|
|     o.*.o.    =o|
|    .o+oSo.   ..o|
|   . +oo+  o . oo|
|    . o.    o o  |
|             .   |
|                 |
+----[SHA256]-----+
root@ip-172-31-89-187:/home/ubuntu# cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Example of ssh-ing into localhost

```
root@ip-172-31-89-187:/home/ubuntu# ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:HZVtauvI0ES3SXLtCDSAwUG0McW18aJ7Qj3F+o0DBV4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1057-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Apr  7 22:15:57 UTC 2020

  System load:  0.0               Processes:           108
  Usage of /:   31.1% of 7.69GB   Users logged in:     1
  Memory usage: 3%                IP address for eth0: 172.31.89.187
  Swap usage:   0%


76 packages can be updated.
50 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@ip-172-31-89-187:~#
```

Logging out from localhost

```
root@ip-172-31-89-187:~# exit
logout
Connection to localhost closed.
```

Next, we add specifications to the XML files for the Hadoop file system. This includes for running the dfs, as well as yarn.

```
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/etc/hadoop# vi mapred-site.xml
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/etc/hadoop# vi core-site.xml
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/etc/hadoop# vi yarn-site.xml
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/etc/hadoop# vi core-site.xml
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/etc/hadoop# vi hdfs-site.xml
```

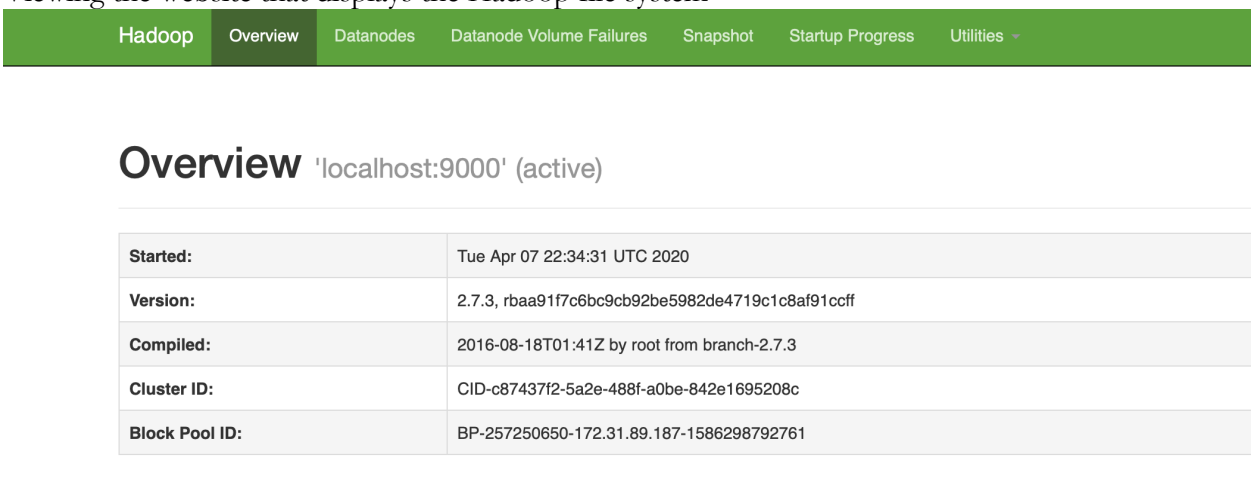
Next, we format the Hadoop filesystem,

```
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3# hdfs namenode -format
20/04/07 22:33:11 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = ip-172-31-89-187.ec2.internal/172.31.89.187
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.3
STARTUP_MSG:   classpath = /home/ubuntu/hadoop-2.7.3/etc/hadoop:/home/ubuntu/had
```

Starting the NameNode, DataNode, SecondaryNameNode, and ResourceManager,

```
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# ./start-dfs.sh
20/04/07 22:34:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-root-namenode-ip-172-31-89-187.out
localhost: starting datanode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-root-datanode-ip-172-31-89-187.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:HZVtauvI0ES3SXLtCDSAwUG0McW18aJ7Qj3F+o0DBV4.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /home/ubuntu/hadoop-2.7.3/logs/hadoop-root-secondarynamenode-ip-172-31-89-187.out
20/04/07 22:34:48 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# jps
5863 SecondaryNameNode
5465 NameNode
5978 Jps
5646 DataNode
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# ./start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/ubuntu/hadoop-2.7.3/logs/yarn-root-resourcemanager-ip-172-31-89-187.out
localhost: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.3/logs/yarn-root-nodemanager-ip-172-31-89-187.out
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# jps
5863 SecondaryNameNode
5465 NameNode
6473 Jps
6026 ResourceManager
6363 NodeManager
5646 DataNode
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin#
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /home/ubuntu/hadoop-2.7.3/logs/mapred-root-historyserver-ip-172-31-89-187.out
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# yarn-daemon.sh start nodemanager
nodemanager running as process 6363. Stop it first.
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin# jps
6610 Jps
5863 SecondaryNameNode
5465 NameNode
6026 ResourceManager
6363 NodeManager
6523 JobHistoryServer
5646 DataNode
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3/sbin#
```

Viewing the website that displays the Hadoop file system



Making the input directory and copying the *Pride and Prejudice* text file to the Hadoop file system.

```
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3# hdfs dfs -ls /
20/04/07 22:56:09 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
drwxrwx---   - root supergroup          0 2020-04-07 22:39 /tmp
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3# hdfs dfs -mkdir /input
20/04/07 22:56:22 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@ip-172-31-89-187:/home/ubuntu/hadoop-2.7.3# cd ..
root@ip-172-31-89-187:/home/ubuntu# hdfs dfs -copyFromLocal Hamlet.txt /input
20/04/07 22:56:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@ip-172-31-89-187:/home/ubuntu#
```

MapReduce Modifications

Modifying the Mapper of the WordCount.java script in order to suit our needs to count only words with letters. The regex expression "^[a-zA-Z]+$" ensures that only non-null strings with alphabetic characters are passed to the mapper. I used Java's Pattern class to compile the regex expression and Java's Matcher classes to identify matches in the strings tokens.

```
public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  // Construct the alphabet for the Matcher
  private static final Pattern alphabet = Pattern.compile("^[a-zA-Z]+$");

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      // Get a token
      String token = tokenizer.nextToken();
      // Determine if the token matches the alphabet Pattern
      Matcher wordMatcher = alphabet.matcher(token);
      // If the token matches the alphabet Pattern, then add it
      if (wordMatcher.matches()) {
        word.set(token);
        output.collect(word, one);
      }
    }
  }
}
```

In addition, I set the number of reducers to 5 programmatically for our tests, as seen below,

```
public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  // Set the number of reducer tasks to 5
  conf.setNumReduceTasks(5);

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

  JobClient.runJob(conf);
}
```

The main portion of this program is counting the unique words from the *Pride and Prejudice* text file. In order to do this, we need to first compile out MapReduce program so that it can be run on Hadoop. Below, we can see the compilation of the WordCount.java MapReduce program, and the subsequent compilation of the WordCount classes into WordCount.jar, all seen below,

```
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# javac -classpath /home/ubuntu/hadoop-2.7.3/etc/hadoop:/home/ubuntu/hadoop-2.7.3/share/hadoop/common/lib/*:/hom
e/ubuntu/hadoop-2.7.3/share/hadoop/common/*:/home/ubuntu/hadoop-2.7.3/share/hadoop/hdfs:/home/ubuntu/hadoop-2.7.3/share/hadoop/hdfs/lib/*:/home/ubuntu/hadoop-2.7.3/share/ha
doop/hdfs/*:/home/ubuntu/hadoop-2.7.3/share/hadoop/yarn/lib/*:/home/ubuntu/hadoop-2.7.3/share/hadoop/yarn/*:/home/ubuntu/hadoop-2.7.3/share/hadoop/mapreduce/lib/*:/home/ubu
ntu/hadoop-2.7.3/share/hadoop/mapreduce/*:/home/ubuntu/hadoop-2.7.3/contrib/capacity-scheduler/*.jar -d WordCountClasses/ WordCount.java
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# jar -cvf WordCount.jar -C WordCountClasses/ .
added manifest
adding: WordCount$Map.class(in = 2263) (out= 972)(deflated 57%)
adding: WordCount.class(in = 1550) (out= 772)(deflated 50%)
adding: WordCount$Reduce.class(in = 1591) (out= 642)(deflated 59%)
```

Next, we run the WordCount.jar on the Hadoop file system, as seen below,

```
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# hadoop jar WordCount.jar WordCount /input /output5ReducersWords
20/04/09 02:04:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20/04/09 02:04:46 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
20/04/09 02:04:46 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
20/04/09 02:04:46 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
20/04/09 02:04:46 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolR
unner to remedy this.
20/04/09 02:04:47 INFO mapred.FileInputFormat: Total input paths to process : 1
20/04/09 02:04:47 INFO mapreduce.JobSubmitter: number of splits:1
20/04/09 02:04:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1054084569_0001
20/04/09 02:04:47 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
20/04/09 02:04:47 INFO mapreduce.Job: Running job: job_local1054084569_0001
20/04/09 02:04:47 INFO mapred.LocalJobRunner: OutputCommitter set in config null
20/04/09 02:04:47 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
20/04/09 02:04:47 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
20/04/09 02:04:48 INFO mapred.LocalJobRunner: Waiting for map tasks
```

We were able to map and reduce the program!

```
INFO mapreduce.Job:  map 100% reduce 100%
```

We can see the 5 output files in the Hadoop file system on the website, as seen below,

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/output5ReducersWords Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	4/8/2020, 8:27:01 PM	1	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	0 B	4/8/2020, 8:27:01 PM	1	128 MB	part-00000
-rw-r--r--	root	supergroup	0 B	4/8/2020, 8:27:01 PM	1	128 MB	part-00001
-rw-r--r--	root	supergroup	0 B	4/8/2020, 8:27:01 PM	1	128 MB	part-00002
-rw-r--r--	root	supergroup	0 B	4/8/2020, 8:27:01 PM	1	128 MB	part-00003
-rw-r--r--	root	supergroup	20 B	4/8/2020, 8:27:01 PM	1	128 MB	part-00004

Next we get the output from the 5 reducers and store it locally, as seen below,

```
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# hdfs dfs -get /output5ReducersWords
20/04/09 02:05:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# ls
WordCount.jar  WordCount.java  WordCountClasses  commands.sh  old.java  output5ReducersWords
```

Obtaining the Deliverables

I wrote a bash script to determine the unique word count, the fifth to last word and its number of occurrences, and the first word and its number of occurrences, as seen below. The 5 output files from the reducers are all in the output5ReducersWords directory. This bash script concatenates them and pipes them into their respective functions.

```
#!/bin/bash

echo "Number of Unique Items"
cat output5ReducersWords/* | wc -l
echo "Fifth to Last Term and Number of Occurences of the Sorted List"
cat output5ReducersWords/* | sort | tail -5 | head -1
echo "First Term and Number of Occurences of the Sorted List"
cat output5ReducersWords/* | sort | head -1
```

When we run the bash script, it determines the unique word count, the fifth to last word and its number of occurrences, and the first word and its number of occurrences, as seen below,

When we use all text in *Pride and Prejudice*, the answers are,

```
root@ip-172-31-89-187:/home/ubuntu/RawTextReducers# ./commands.sh
Number of Unique Items
13777
Fifth to Last Term and Number of Occurences of the Sorted List
yourselves      1
First Term and Number of Occurences of the Sorted List
"'Tis   2
root@ip-172-31-89-187:/home/ubuntu/RawTextReducers#
```

When we only map and reduce the words in *Pride and Prejudice*, the answers are

```
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers# ./commands.sh
Number of Unique Items
5999
Fifth to Last Term and Number of Occurences of the Sorted List
yours   2
First Term and Number of Occurences of the Sorted List
A       41
root@ip-172-31-89-187:/home/ubuntu/RegularExpressionReducers#
```

Finally, we are done and we can stop the Hadoop filesystem,