Tutorial:
Introduction to Hadoop MapReduce

Sayed Hadi Hashemi
Last update: August 31, 2015

1 Overview

Welcome

Welcome to the MapReduce tutorial. This tutorial covers the critical skills needed to develop a Hadoop MapReduce application. It starts with an already deployed Hadoop environment where students will execute a series of hands-on labs.

Objectives

Upon completing this tutorial, students will be able to:

- Perform basic operations on HDFS
- Develop a simple MapReduce application (Word Count)
- Compile MapReduce applications
- Run MapReduce applications and examine the output
- Make modifications in MapReduce applications

Structure

This guide is designed as a set of step-by-step instructions for hands-on exercises. It teaches you how to operate the MapReduce service in a functional test environment through a series of examples. Exercises should be completed in the order described in the following steps:

1. Download some input datasets
2. Prepare a HDFS and upload dataset on it
3. Develop a simple "**Word Count**" application and build it
4. Run the application on Hadoop and examine the output
5. Modify the **"Word Count"** application to discard common words
6. Rebuilt, and rerun the modified application
7. Do it yourself modification

2 Requirements

This tutorial is designed to work on the **Hortonworks Sandbox 2.3** virtual machine. You need to have a working HortonWorks Sandbox machine either locally or on the Amazon Web Services.

Also, all assignments are designed based on **JDK 7** (included in the virtual machine).

Please refer to “Tutorial: Run HortonWorks Sandbox 2.3 Locally” or “Tutorial: Run HortonWorks Sandbox 2.3 on AWS” for more information.

3 Setup Hadoop Virtual Machine

Step 1: Start the virtual machine; then connect to it through the SSH.

Step 2: After successfully logging in, you should see a prompt similar to the following:

```
[root@sandbox ~]#
```

Step 3: Create a new folder for this tutorial:

```
# mkdir mapreduce-tutorial
```

Step 4: Change the current folder:

```
# cd mapreduce-tutorial
```

4 Prepare the HDFS

MapReduce needs all the input and output files to be located on HDFS. In this section, we will first download some text file as the input dataset. Then we create necessary folders on HDFS. Lastly, we copy the dataset to the HDFS.
For the input dataset some masterpieces by William Shakespeare (Hamlet, Macbeth, Othello, and Romeo & Juliet) will be downloaded directly from the Gutenberg Project.

Step 1: Create a folder for the input datasets:

```
# mkdir dataset
```

Step 2: Download the following Shakespeare books:

```
# wget -c http://www.gutenberg.lib.md.us/2/2/6/2264/2264.txt -P dataset
# wget -c http://www.gutenberg.lib.md.us/2/2/6/2265/2265.txt -P dataset
# wget -c http://www.gutenberg.lib.md.us/2/2/6/2266/2266.txt -P dataset
# wget -c http://www.gutenberg.lib.md.us/2/2/6/2267/2267.txt -P dataset
```

Step 3: Create some folders on HDFS:

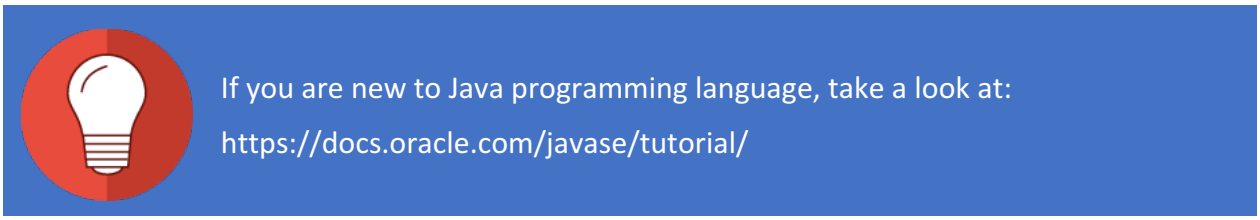
```
# hadoop fs -mkdir -p /tutorial/input
```

Step 4: Upload the Dataset to HDFS

```
# hadoop fs -put ./dataset/* /tutorial/input
```

5 Hello World (Word Count)

The Word Count application is the canonical example in MapReduce. It is a straightforward application of MapReduce, and MapReduce can handle word count extremely efficiently. In this particular example, we will be doing a word count within Shakespeare's books.



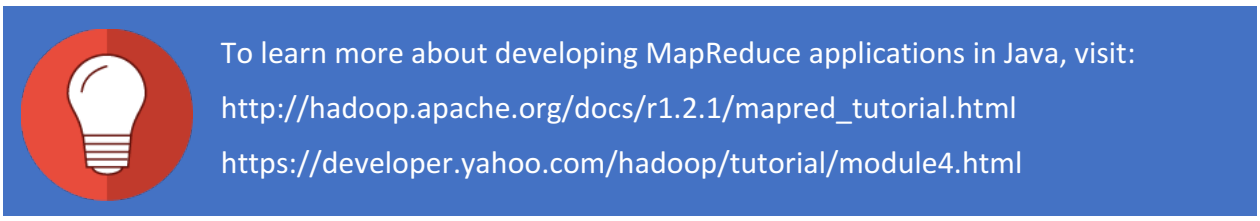
Step 1: Create a new Java source code file using the following command:

```
# nano WordCount.java
```

Step 2: Type (or copy/paste) the Java code of **"Appendix A"** in the editor, and then quit **nano** and save the file.

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget
https://github.com/xldrx/mapreduce_examples/raw/master/tutorial/first_exam
ple/WordCount.java
```



Step 3: Create a temporary folder for compiling the code:

```
# mkdir build
```

Step 4: Compile the source code using the following commands.

```
# export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
# hadoop com.sun.tools.javac.Main WordCount.java -d build
# jar -cvf WordCount.jar -C build/ ./
```

Step 5 (Optional: Just in case of compile errors): If you encounter any compilation error (possibly due to typos), you need to first edit the source file and fix the problems. Then clean the build folder, and finally, compile the code again using the direction above. In summary, these are the commands:

```
# nano WordCount.java
# rm –rf ./build/* ./WordCount.jar
# hadoop com.sun.tools.javac.Main WordCount.java -d build
# jar -cvf WordCount.jar -C build/ ./
```

Step 6: Run the application using the following command and wait for it to be done:

```
# hadoop jar WordCount.jar WordCount /tutorial/input /tutorial/output
```

Step 7: The output of the application will be on the HDFS. To see the output, you may use the following command:

```
# hadoop fs -cat /tutorial/output/part*
```

Step 8 (Optional): The output files can be downloaded from HDFS using the following command:

```
# hadoop fs -get /tutorial/output/part* .
```

Step 9 (Optional: Just in case of runtime errors): If you encounter any compilation error (possibly due to typos), you need to first edit the source file to fix the problems, and then compile the code again using the direction above.

Additionally, the output folder on HDFS should be cleaned before a new run using following command:

```
# hadoop fs -rm -r -f /tutorial/output
```

6 Makes Modifications

Let's make some changes to the Word Count application. As you might have already noticed, words are not correctly tokenized. This happens because the default function for tokenization in Java only uses a limited number of splitter characters. Furthermore, there are a few common English words that are not interesting enough to be included in the output.
In this section, we will fix these two problems.

Step 1: Edit the source code:

```
# nano WordCount.java
```

Step 2: Edit the WordCountMap class to match with following code:

```
public static class WordCountMap extends Mapper<Object, Text, Text,
IntWritable> {
    List<String> commonWords = Arrays.asList("the", "a", "an", "and",
"of", "to", "in", "am", "is", "are", "at", "not");
    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line, " \t,;.?!-
:@[](){}_*/");
        while (tokenizer.hasMoreTokens()) {
            String nextToken = tokenizer.nextToken();
            if (!commonWords.contains(nextToken.trim().toLowerCase())) {
                context.write(new Text(nextToken), new IntWritable(1));
            }
        }
    }
}
```

Step 2 (Alternative): Quit **nano**. Download the following source from the github:

```
# wget
https://github.com/xldrx/mapreduce_examples/raw/master/tutorial/second_exa
mple/WordCount.java
```

Step 3: Rebuild the code:

```
# rm –rf ./build/* ./WordCount.jar
# export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
# hadoop com.sun.tools.javac.Main WordCount.java -d build
# jar -cvf WordCount.jar -C build/ ./
```

Step 4: Clear the output folder:

```
# hadoop fs -rm -r /tutorial/output
```

Step 5: Run the application:

```
# hadoop jar WordCount.jar WordCount /tutorial/input /tutorial/output
```

Step 6: Check the results:

```
# hadoop fs -cat /tutorial/output/part*
```

7 Do it yourself

One other problem with the current output is that it is too long. In this section we will develop a new application which only shows the top 10 most commonly used words.

Step 1: Clean up the previous build file:

```
# rm -rf ./build/* ./*.jar
```

Step 2: Create a new Java source code file using the following command:

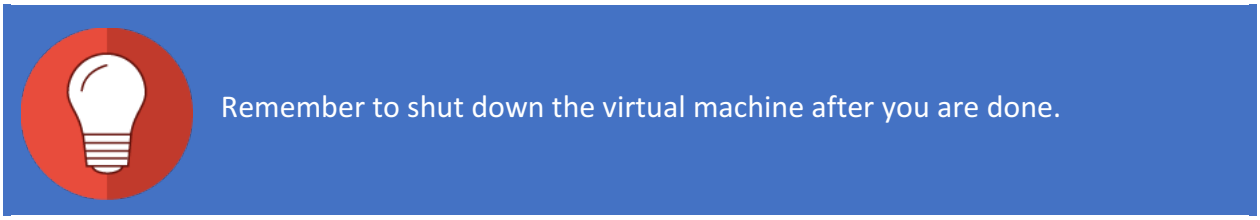
```
# nano TopWords.java
```

Step 3: Type (or copy/paste) the Java code of **"Appendix C"** in the editor, and then quit **nano** and save the file.

Step 3 (Alternative): Quit **nano**. Download the following source from the github:

```
# wget
https://github.com/xldrx/mapreduce_examples/raw/master/tutorial/third_exam
ple/TopWords.java
```

Step 4: Try to compile and run this application yourself.



8 Questions to Think About

1. Why data should be on HDFS?
2. Why should the "output" folder be removed each time?
3. In the "Top Word" example, can you develop a new code which reuses (not copy-paste) the "Word Count" code?

Appendix A: Word Count v1 Source Code

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

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {
    public static class WordCountMap extends Mapper<Object,
Text, Text, IntWritable> {
        @Override
      public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
            String nextToken = tokenizer.nextToken();
            context.write(new Text(nextToken), new
IntWritable(1));
         }
      }
   }

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

```
    }

    public static void main(String[] args) throws Exception {

          Job job = Job.getInstance(new Configuration(),
"wordcount");
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(IntWritable.class);

          job.setMapperClass(WordCountMap.class);
          job.setReducerClass(WordCountReduce.class);

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

          job.setJarByClass(WordCount.class);
       System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Appendix B: Word Count v2 Source Code

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

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {
    public static class WordCountMap extends Mapper<Object,
Text, Text, IntWritable> {
        List<String> commonWords = Arrays.asList("the", "a",
"an", "and", "of", "to", "in", "am", "is", "are", "at", "not");
        @Override
        public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new
StringTokenizer(line, " \t,;.?!-:@[](){}_*/");
            while (tokenizer.hasMoreTokens()) {
                String nextToken = tokenizer.nextToken();
                if
(!commonWords.contains(nextToken.trim().toLowerCase())) {
                    context.write(new Text(nextToken), new
IntWritable(1));
                }
            }
        }
    }

    public static class WordCountReduce extends Reducer<Text,
IntWritable, Text, IntWritable> {
        @Override
        public void reduce(Text key, Iterable<IntWritable>
values, Context context) throws IOException,
InterruptedException {
```

```
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {

        Job job = Job.getInstance(new Configuration(),
"wordcount");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(WordCountMap.class);
        job.setReducerClass(WordCountReduce.class);

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

        job.setJarByClass(WordCount.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Appendix C: Top Words Source Code

```
import java.io.IOException;
import java.lang.Integer;
import java.util.*;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class TopWords {
    public static class TextArrayWritable extends ArrayWritable
{
        public TextArrayWritable() {
            super(Text.class);
        }

        public TextArrayWritable(String[] strings) {
            super(Text.class);
            Text[] texts = new Text[strings.length];
            for (int i = 0; i < strings.length; i++) {
                texts[i] = new Text(strings[i]);
            }
            set(texts);
        }
    }

    public static class WordCountMap extends Mapper<Object,
Text, Text, IntWritable> {
        List<String> commonWords = Arrays.asList("the", "a",
"an", "and", "of", "to", "in", "am", "is", "are", "at", "not");
        @Override
        public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
            String line = value.toString();
```

```
            StringTokenizer tokenizer = new
StringTokenizer(line, " \t,;.?!-:@[](){}_*/");
            while (tokenizer.hasMoreTokens()) {
                String nextToken = tokenizer.nextToken();
                if
(!commonWords.contains(nextToken.trim().toLowerCase())) {
                    context.write(new Text(nextToken), new
IntWritable(1));
                }
            }
        }
    }

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

    public static class TopWordsMap extends Mapper<Text, Text,
NullWritable, TextArrayWritable> {
        private TreeSet<Pair<Integer, String>> countToWordMap =
new TreeSet<Pair<Integer, String>>();

        @Override
        public void map(Text key, Text value, Context context)
throws IOException, InterruptedException {
            Integer count = Integer.parseInt(value.toString());
            String word = key.toString();

            countToWordMap.add(new Pair<Integer, String>(count,
word));

            if (countToWordMap.size() > 10) {
                countToWordMap.remove(countToWordMap.first());
            }
        }
```

```
        @Override
        protected void cleanup(Context context) throws
IOException, InterruptedException {
            for (Pair<Integer, String> item : countToWordMap) {
                String[] strings = {item.second,
item.first.toString()};
                TextArrayWritable val = new
TextArrayWritable(strings);
                context.write(NullWritable.get(), val);
            }
        }
    }

    public static class TopWordsReduce extends
Reducer<NullWritable, TextArrayWritable, Text, IntWritable> {
        private TreeSet<Pair<Integer, String>> countToWordMap =
new TreeSet<Pair<Integer, String>>();

        @Override
        public void reduce(NullWritable key,
Iterable<TextArrayWritable> values, Context context) throws
IOException, InterruptedException {
            for (TextArrayWritable val: values) {
                Text[] pair= (Text[]) val.toArray();

                String word = pair[0].toString();
                Integer count =
Integer.parseInt(pair[1].toString());

                countToWordMap.add(new Pair<Integer,
String>(count, word));

                if (countToWordMap.size() > 10) {

countToWordMap.remove(countToWordMap.first());
                }
            }

            for (Pair<Integer, String> item: countToWordMap) {
                Text word = new Text(item.second);
                IntWritable value = new IntWritable(item.first);
                context.write(word, value);
            }
        }
    }
```

```
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        Path tmpPath = new Path("/w1/tmp");
        fs.delete(tmpPath, true);


        Job jobA = Job.getInstance(conf, "wordcount");
        jobA.setOutputKeyClass(Text.class);
        jobA.setOutputValueClass(IntWritable.class);

        jobA.setMapperClass(WordCountMap.class);
        jobA.setReducerClass(WordCountReduce.class);

        FileInputFormat.setInputPaths(jobA, new Path(args[0]));
        FileOutputFormat.setOutputPath(jobA, tmpPath);

        jobA.setJarByClass(TopWords.class);
        jobA.waitForCompletion(true);

        Job jobB = Job.getInstance(conf, "Top Words");
        jobB.setOutputKeyClass(Text.class);
        jobB.setOutputValueClass(IntWritable.class);

        jobB.setMapOutputKeyClass(NullWritable.class);
        jobB.setMapOutputValueClass(TextArrayWritable.class);

        jobB.setMapperClass(TopWordsMap.class);
        jobB.setReducerClass(TopWordsReduce.class);
        jobB.setNumReduceTasks(1);

        FileInputFormat.setInputPaths(jobB, tmpPath);
        FileOutputFormat.setOutputPath(jobB, new Path(args[1]));

        jobB.setInputFormatClass(KeyValueTextInputFormat.class);
        jobB.setOutputFormatClass(TextOutputFormat.class);

        jobB.setJarByClass(TopWords.class);
        System.exit(jobB.waitForCompletion(true) ? 0 : 1);
    }
}

class Pair<A extends Comparable<? super A>,
        B extends Comparable<? super B>>
        implements Comparable<Pair<A, B>> {
```

```
    public final A first;
    public final B second;

    public Pair(A first, B second) {
        this.first = first;
        this.second = second;
    }

    public static <A extends Comparable<? super A>,
            B extends Comparable<? super B>>
    Pair<A, B> of(A first, B second) {
        return new Pair<A, B>(first, second);
    }

    @Override
    public int compareTo(Pair<A, B> o) {
        int cmp = o == null ? 1 :
(this.first).compareTo(o.first);
        return cmp == 0 ? (this.second).compareTo(o.second) :
cmp;
    }

    @Override
    public int hashCode() {
        return 31 * hashcode(first) + hashcode(second);
    }

    private static int hashcode(Object o) {
        return o == null ? 0 : o.hashCode();
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Pair))
            return false;
        if (this == obj)
            return true;
        return equal(first, ((Pair<?, ?>) obj).first)
                && equal(second, ((Pair<?, ?>) obj).second);
    }

    private boolean equal(Object o1, Object o2) {
        return o1 == o2 || (o1 != null && o1.equals(o2));
    }
```

```
    @Override
    public String toString() {
        return "(" + first + ", " + second + ')';
    }
}
```