Hadoop "Hello World" – WordCount example

Due date: September 23rd (by midnight).

The main goal for this assignment is to familiarize yourself with running Hadoop in several modes: standalone mode (using the local file system for input/output, no HDFS – mainly useful for development/debugging purposes), pseudo-distributed environment (a.k.a., single-node cluster) - on your own computers (where you need to install Hadoop), and also in a fully distributed mode on the Beocat cluster.

To get a Beocat account, please follow the link:
https://account.beocat.cis.ksu.edu

Information about Beocat's Hadoop cluster is available at:
http://support.beocat.cis.ksu.edu/BeocatDocs/index.php/Hadoop

There are many releases of Hadoop available, but at a high level, we can characterize these releases into Hadoop 1.x and Hadoop 2.x. Hadoop 1 was designed with large MapReduce batch jobs in mind (which is what we need for many information retrieval tasks). Hadoop 2 was designed to also allow running more interactive/specialized processing models (such as interactive querying and streaming data applications) simultaneously with MapReduce jobs. Two main technical advances in Hadoop 2 are the HDFS federation (multiple Namenode servers manage namespaces) and the resource manager YARN. YARN stands for Yet Another Resource Negotiator, and can be seen as a large-scale, distributed operating system for big data applications. You can read more about Hadoop 1 vs Hadoop 2 differences at:
http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html

Given that we will use Hadoop for traditional MapReduce batch jobs, both Hadoop 1 and Hadoop 2 releases will work well for our programming assignments. The easiest to use would be a Hadoop 1 release, e.g. the stable Hadoop 1.2.1 release. The version available on Beocat is Hadoop 2.0.0. The latest stable release is Hadoop 2.7.1. The MapReduce in Hadoop 2 (MRv2) is *mostly* backwards compatible with MapReduce in Hadoop 1 (MRv1). Some incompatibilities are described at:

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduce_Compatibility_Hadoop1_Hadoop2.html

We will use the WordCount example in what follows, and three sample text collections: the text of the Bible, the complete works of Shakespeare, and Bible+Shakespeare – the corresponding files are available on K-State online.

Please download your preferred version of Hadoop, and use the WordCount code that is distributed with that respective version. Several stable releases Hadoop can be found at each one of the mirrors listed at:

http://www.apache.org/dyn/closer.cgi/hadoop/common, for example at
http://www.motorlogy.com/apache/hadoop/common/

To install Hadoop on a windows machine, you will first need to install Ubuntu VM (make sure that the VM is provided enough RAM and cores for good performance. On a machine with 8GB, try setting RAM to 3GB.

For Hadoop 1, you can follow the tutorial at:
http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

For Hadoop 2, you can follow one of the tutorials at the following links:
http://happydaddytoday.blogspot.com/2014/04/hadoop-220-installation-steps-for.html
http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html

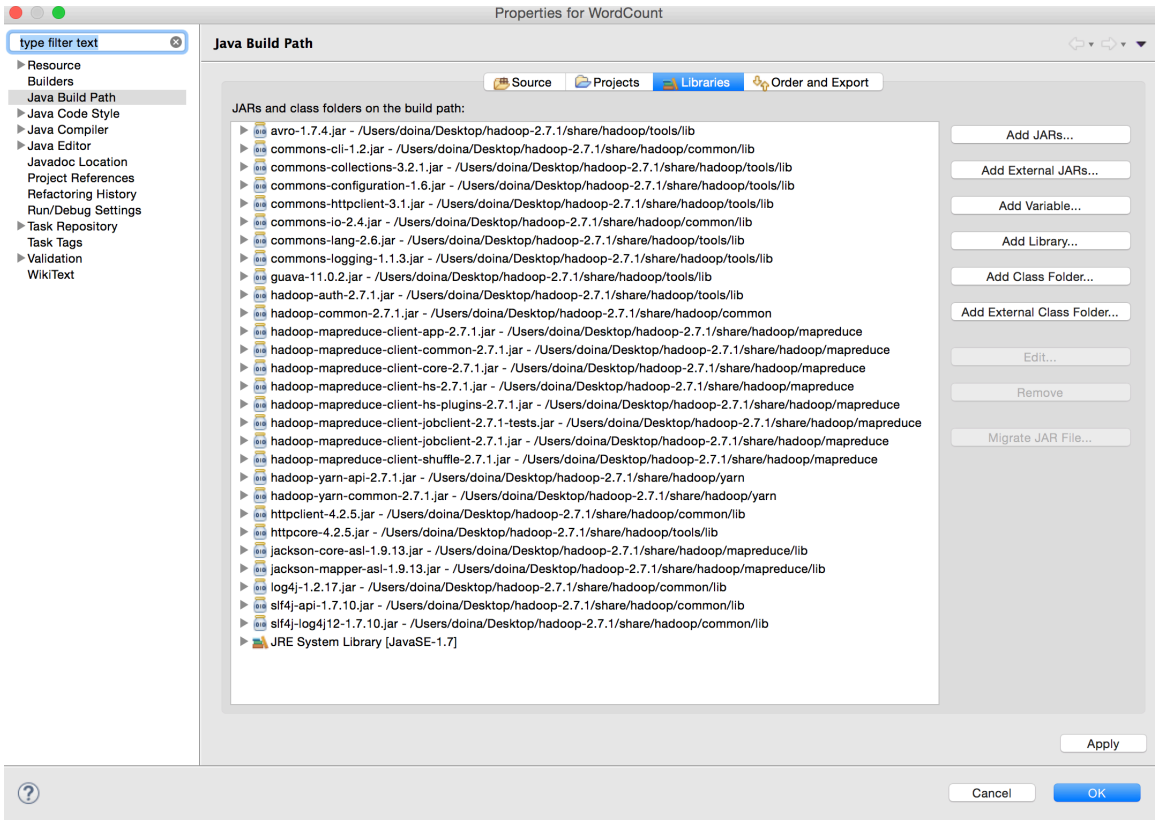
Alternatively, you can simply install the Cloudera QuickStart VM from:
http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-4-x.html

After installing the Ubuntu VM and Hadoop, install Eclipse (or your favorite IDE) in Ubuntu.

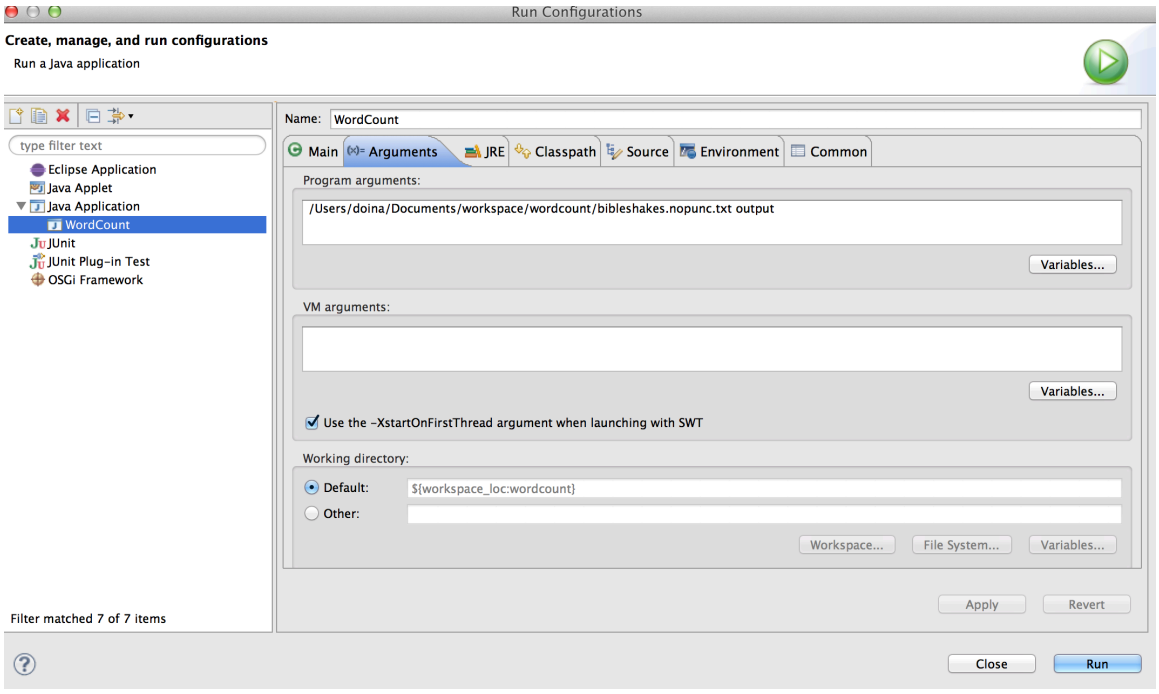
You will need to compile the Hadoop WordCount.java class into a JAR file (the instructions below assume that you are using the Eclipse IDE):

1. **Create a new Java Project.** Launch Eclipse, and from the File Menu select New, then use the Wizard to create a new Java Project. Enter a project name, e.g., WordCount. Click Finish.

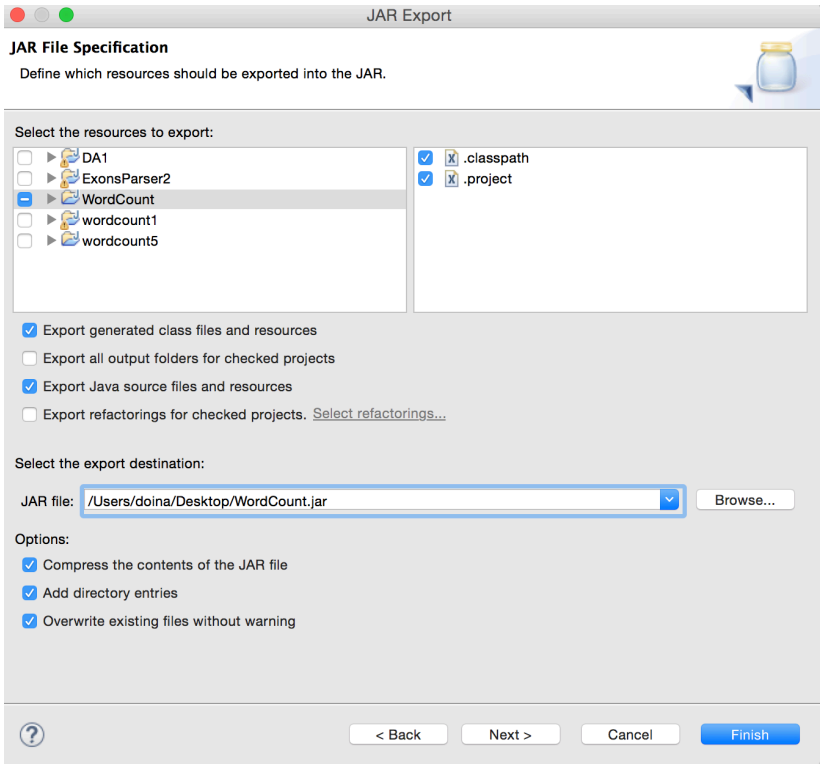
2. **Add Hadoop library to project.** In Eclipse, right-click (control-click), on your project, go to Configure Build Path then Add External JARs. Browse to the Hadoop folder and select the jars you want to import, then click Open. Please see the screenshot below for a list of jars that are needed for the WordCount example to compile and run.



3. **Add source code file.** Copy the WordCount.java code to the project source directory. Eclipse will compile the file as soon as you save it. At this point, you should be able to run the program in standalone mode from Eclipse, if you specify the input and output directories as shown below. The good thing about running Hadoop in standalone mode using Eclipse is that you can debug your code while developing it. This is useful especially when writing larger programs.



4. **Export JAR file.** From the File Menu, select Export. From under Java select Runnable JAR file, click Next. Make sure the `Extract required libraries into generated JAR` checkbox is checked. Select an export destination for your JAR file - you can use your Desktop, or some other directory. For simplicity, name the file WordCount.jar.



Once the .jar file is created, you can also run the WordCount from command line as shown below (in standalone mode, the local file system is used):

$ bin/hadoop jar WordCount.jar WordCount <in-dir> <out-dir>

Next, you will need run the WordCount program in a single-node/pseudo-distributed mode on the Bible and Shakespeare files, separately, and also on the combined Bible+Shakespeare file.

Make sure you start all the Hadoop daemons. Also, put the input data <in-dir> into the HDFS before you run Hadoop and, at the end, copy the output <out-dir> to your local file system.

To run the program, the command syntax is the same as the one for standalone mode, except that <in-dir> and <out-dir> are now HDFS locations:

$ bin/hadoop jar WordCount.jar WordCount <in-dir> <out-dir>

When you are done with the output files for a run, you should delete the output directory. Hadoop will not automatically do this for you, and it will throw an error if you run it while there is an old output directory. To do this, execute: $ hadoop dfs -rmr output

For more details on how to run the WordCount example, please see:

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v2.7.1

Question 1: Show screenshots for your file copying commands to and from HDSF and also for your WordCount.jar running command.

The WordCount program counts the total number of times each distinct word appears in the input set of files. So, the output is of the form (word, count). Please answer the following questions:

Question 2: Show the first 10 and the last 10 words, and their corresponding counts, in your output files.

Question 3: What is the most frequent word in the Bible collection? What is the most frequent word in the Shakespeare collection?

Next, you will run the WordCount program on Beocat on the combined Bible+Shakespeare file.

Question 4: Show screenshots for your file copying commands to and from HDSF and also for your WordCount.jar running command.

Question 5: What is the most frequent word in the combined collection?

At last, you should modify the WordCount program, so that it outputs the word count for each distinct word in each file. So, the output of this DocWordCount program should be of the form (word, filename, count). Compile the new class into a JAR file and run it with an input directory containing the Bible and Shakespeare files (you can do this on your local machines).

Question 6: Submit your source code (named DocWordCount.java).

Question 7: Show the first 10 and the last 10 words, together with their counts in each document, in the output file.

Question 8. How long did it take you to complete this assignment?

Other Hadoop resources:

Hadoop: http://hadoop.apache.org/
Hadoop wiki: http://wiki.apache.org/hadoop/
FileSystemShell: http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/FileSystemShell.html

Acknowledgements: This assignment is adapted from similar assignments by Lim (UMD) and Akella (UNCC) and also from a quick guide on Hadoop at Cornell.