Running a MapReduce Job

|  |  |
| --- | --- |
| **Source Dir** | ~/workspace/wordcount/src/solution |
| **Java Files** | WordCount.java: A MapReduce driver class. WordMapper.java: A mapper class for the job. SumReducer.java: A reducer class for the job. |
| **JAR File** | wc.jar |

**In this exercise you will compile Java files, create a JAR, and run MapReduce jobs.**

In addition to manipulating files in HDFS, the wrapper program hadoop is used to launch MapReduce jobs. The code for a job is contained in a compiled JAR file. Hadoop loads the JAR into HDFS and distributes it to the worker nodes, where the individual tasks of the MapReduce job are executed.

One simple example of a MapReduce job is to count the number of occurrences of each word in a file or set of files. In this lab you will compile and submit a MapReduce job to count the number of occurrences of every word in the works of Shakespeare.

**Compiling and Submitting a MapReduce Job**

In a terminal window, change to the exercise source directory, and list the contents:

$ cd ~/workspace/wordcount/src

$ ls

This directory contains three “package” subdirectories: solution, fixme and hints. In this example we will be using the solution code, so list the files in the solution package directory:

$ ls solution

The package contains the following Java files:

WordCount.java: A simple MapReduce driver class.

WordMapper.java: A mapper class for the job.

SumReducer.java: A reducer class for the job.

Examine these files if you wish, but do not change them. Remain in this directory while you execute the following commands.

Before compiling, examine the classpath Hadoop is configured to use:

$ hadoop classpath

This shows lists the locations where the Hadoop core API classes are installed.

Compile the three Java classes:

$ javac -classpath `hadoop classpath` solution/\*.java

NOTE: In the command above, the quotes around hadoop classpath are backquotes. This runs the hadoop classpath command and uses its output as part of the javac command. The compiled (.class) files are placed in the solution directory.

1. Collect your compiled Java files into a JAR file:

$ jar cvf wc.jar solution/\*.class

Submit a MapReduce job to Hadoop using your JAR file to count the occurrences of each word in Shakespeare:

$ yarn jar wc.jar solution.WordCount shakespeare wordcounts

This yarn jar command names the JAR file to use (wc.jar), the class whose main method should be invoked (solution.WordCount), and the HDFS input and output directories to use for the MapReduce job.

Your job reads all the files in your HDFS shakespeare directory, and places its output in a new HDFS directory called wordcounts.

2. Try running this same command again without any change:

$ yarn jar wc.jar solution.WordCount shakespeare wordcounts

Your job halts right away with an exception, because Hadoop automatically fails if your job tries to write its output into an existing directory. This is by design; since the result of a MapReduce job may be expensive to reproduce, Hadoop prevents you from accidentally overwriting previously existing files.

3. Review the result of your MapReduce job:

$ hdfs dfs -ls wordcounts

This lists the output files for your job. (Your job ran with only one Reducer, so there should be one file, named part-r-00000, along with a \_SUCCESS file and a \_logs directory.)

1. View the contents of the output for your job:

$ hdfs dfs -cat wordcounts/part-r-00000 | less

You can page through a few screens to see words and their frequencies in the

works of Shakespeare. (The spacebar will scroll the output by one screen; the

letter 'q' will quit the less utility.) Note that you could have specified wordcounts/\* just as well in this command.

**Wildcards in HDFS file paths**

Take care when using wildcards (e.g. \*) when specifying HFDS filenames; because of how Linux works, the shell will attempt to expand the wildcard before invoking hadoop, and then pass incorrect references to local files instead of HDFS files. You can prevent this by enclosing the wildcarded HDFS filenames in single quotes, e.g. hdfs dfs –cat ‘wordcounts/\*'

1. Try running the WordCount job against a single file:

$ yarn jar wc.jar solution.WordCount shakespeare/poems words

When the job completes, inspect the contents of the pwords HDFS directory.

1. Clean up the output files produced by your job runs:

$ hdfs dfs -rm -r wordcounts pwords

**Stopping MapReduce Jobs**

It is important to be able to stop jobs that are already running. This is useful if, for example, you accidentally introduced an infinite loop into your Mapper. An important point to remember is that pressing ^C to kill the current process (which is displaying the MapReduce job's progress) does not actually stop the job itself.

A MapReduce job, once submitted to Hadoop, runs independently of the initiating process, so losing the connection to the initiating process does not kill the job. Instead, you need to tell the Hadoop JobTracker to stop the job.

5. Start another word count job like you did in the previous section:

$ yarn jar wc.jar solution.WordCount shakespeare count2

While this job is running, open another terminal window and enter:

$ mapred job -list

This lists the job ids of all running jobs. A job id looks something like: job\_200902131742\_0002

Copy the job id, and then kill the running job by entering:

$ mapred job -kill jobid

The JobTracker kills the job, and the program running in the original terminal completes.

**END**