Running Hadoop Jobs at Edinburgh

Miles Osborne

July 17, 2011

Abstract

These notes explain how to run Hadoop jobs on the Edinburgh Hadoop cluster. They do not explain how Map Reduce or Unix works.

1 Getting ready

1.1 Paths etc

Edinburgh has a local version of Linux called *DICE*. There are minor differences, but it is very similar to Ubuntu etc. You will need to *renew* your filesystem credentials when you login. Do this (ie enter your password):

```
[burbank]miles: renc
Password:
```

You also need to do something similar for printing:

```
[burbank]miles: kinit
Password for miles@INF.ED.AC.UK:
[burbank]miles:
```

Finally, you should setup your path so that you can see the Hadoop binary. DICE uses a different version of bash:

`http://www.inf.ed.ac.uk/systems/support/FAQ/#J0`

so you will need to change **.benv** instead of **..bashrc**: Using your favourite editor (I prefer emacs), at the top level, edit **.benv** and add the following line:

```
PATH=$PATH:/opt/hadoop/hadoop-0.20.2/bin/
```

1.2 Logging into Hadoop

Hadoop has a special machine where you launch jobs. You need to login to it before you can do anything with Hadoop: The main one is called **namenode**:

```
ssh namenode
```

There are backup machines called **hackathon1 . . . hackathin19** which you can use if the main machine is too slow. Note these are also **slaves**.

2 DFS

Hadoop consists of two major parts: a distributed file system (DFS) and a method for running jobs. DFS stores files over the network and files are replicated in case of failure. It is very important to note that DFS is not the same as the normal Unix filesystem: you cannot use **ls** etc to use it. Instead, you use special Hadoop commands.

Login to a namenode and run the following commands:

```
hadoop dfs -ls /user/miles
```

This lists the files for user **miles**. You should see this:

```
Found 3 items
drwxr-xr-x   - miles miles          0 2011-02-22 19:20 /user/miles/clueweb
drwxr-xr-x   - miles miles          0 2010-12-02 17:55 /user/miles/data
drwxr-xr-x   - miles miles          0 2011-07-13 11:28 /user/miles/twitter
```

List the files in your area. (It should be empty):

```
hadoop dfs -ls
```

To create a directory on Hadoop, run this command:

```
hadoop dfs -mkdir test
```

Verify that it is there using the Hadoop version of **ls**.

To upload data from Unix to DFS, you use the **put** command. First create a file (on Unix) as follows:

```
ls / > temp.txt
```

Now, upload it to DFS:

```
hadoop dfs -put temp.txt test/temp.txt
```

Verify that it is there using the DFS **ls**. To download data from DFS into Unix:

```
hadoop dfs -cat  test/temp.txt > temp.txt
```

This should be the same as the version on Unix.

Hadoop also supports wildcards, so you can do this:

```
hadoop dfs -cat  test/t*
```

To rename a file:

```
hadoop dfs -mv  test/temp.txt test/fred.txt
```

Files can be deleted using **rm**. You can delete entire directories using **rmr**:

```
hadoop dfs -rmr  test
```

3 Running Map Reduce Jobs

We will be using the *streaming interface* to Hadoop. This allows you to run mappers and reducers written in any programming language. Here we will use a mapper and a reducer written in C++.

Download all of the scripts stored in DFS in **/user/miles/scripts** and store them in Unix. The file **m-compute-ngram-counts-batch** extracts ngrams (defaults to trigrams) from a file of raw sentences. This is a *mapper*. The file **r-compute-ngram-counts** is a *reducer*. It collects together the output of the mapper and produces a list of counted ngrams. All data is stored in DFS.The two files **runStreaming.sh** and **runStreamingCompInCompOut.sh** are front-ends to the streaming interface. (The file **runStreaming.sh** assumes data is not compressed in DFS and **runStreamingCompInCompOut.sh** assumes that data is compressed.)

runStreaming.sh takes five arguments:

1. The name of the mapper: (a Unix path)
2. The name of the reducer (a Unix path)
3. The input directory (a DFS directory)
4. The output directory (a DFS directory)
5. The job name.

Here is an example MR job and we will count trigrams from a file data.

Run the following commands:

```
chmod +x runStreaming.sh
./runStreaming.sh  m-compute-ngram-counts-batch r-compute-ngram-counts
        \ /user/miles/data/small.txt temp2 ngram
```

Ignore the warnings about depreciated flags. Once this is done you can inspect the results:

```
hadoop dfs -ls temp2
Found 11 items
drwxr-xr-x   - miles miles          0 2011-07-16 16:51 /user/miles/temp2/_logs
-rw-r--r--   4 miles miles     441054 2011-07-16 16:51 /user/miles/temp2/part-00000
-rw-r--r--   4 miles miles     438993 2011-07-16 16:51 /user/miles/temp2/part-00001

...
```

Hadoop stores the results in *shards* (for example **part-00000**). Have a look at it.

When developing and debugging MR jobs it can be useful to do this under Unix first. To simulate Hadoop in Unix, do the following:

```
chmod +x m-compute-ngram-counts-batch
ls / | ./m-compute-ngram-counts-batch > results1
```

Look at the results file. Now run it through the reducer:

```
chmod +x r-compute-ngram-counts
sort +0 -1 result1 | ./r-compute-ngram-counts > results2
```

Again, look at the new results. This is not that exciting as there are no repeated grams and so all of the counts are one.

Tracking Jobs

You can track the progress of your job using a web front end:

```
http://hcrc1425n32.inf.ed.ac.uk:50030/jobtracker.jsp
```

Jobs can be listed and killed as follows:

```
hcrc1425n30]miles: hadoop job -list
1 jobs currently running
JobId   State   StartTime       UserName        Priority        SchedulingInfo
job_201107131350_0707   1       1310830768287   s1053654        NORMAL  NA

hcrc1425n30]miles: hadoop job  -Dmapred.job.tracker=129.215.18.32:8021 -kill job_20 ...
```

(You might wish to alias the killing command in your **.benv** file)

4 Your own Hadoop Job

Here you will write a program to do work counting. We will use the Unix **wc** command to do most of the work.

- Count the number of words in the **large** file:

  ```
  hadoop dfs -cat /user/miles/data/large.txt | wc
  ```

- Now, run **wc** as a mapper and use **cat** as a reducer. Note you need to give the actual Unix paths to **wc** and **cat** (use **which wc** etc) when running **runStreaming.sh**. Here, the reducer simply copies the output.

- Inspect the output. Any idea why some of the shards are empty? For the shards that are not empty, what does each line mean?

Write a reducer that sums together the input it receives and produces for each shard a single line of output. For example, the input shard:

```
1 2 3
4 5 6
```

Would instead produce a shard with one line:

```
5 7 9
```

The results are on DFS and Unix. Can you produce a single output (in Unix)?