To help debug programs, here are some tips. The first 2 are an expansion of what was mentioned in the Join Assignment, the 3rd shows how get information during execution. The others are also helpful notes that students have suggested.

1. Try the following from the command line:

>cat testfile1 | ./<mapper.py> | sort

This will show you if the mapper script is working ok outside map/reduce, and not using what is in the HDFS file system.

2. Also try:

> cat testfile* | ./<mapper.py> | sort | ./<reducer.py>

If these commands work you should see the all the <word, total count>'s , and then you can try running in hadoop.

If these commands work but do not show the correct output, then check your code, test your logic, add in print statements to see what is happening, try making a new test file with just a few words.

If these commands work but then your map/reduce job fails, then it becomes more tricky to debug. Perhaps something about data being partitioned is throwing off your logic, I added a reading material, which shows how you can debug a map/reduce program. I suspect you won't need to go there for wordcount, but maybe for join assignment.

3. Here's is some code techniques for debuging Python streaming.

Programs using map/reduce is notoriously difficult to debug,

especially because Hadoop is managing the execution and standard error/output files

#

There are several approaches you can use, some of which depend on your Hadoop set up, like

finding the 'userlog' files and looking for standard error/output files.

Here is perhaps an old-fashioned approach, but one that can be very informative.

In your program you can print debugging information to a file that is in a shared location, which means

you have to name the file differently for each map or reduce program.

#Hopefully, if your program is crashing, this will help you pinpoint the problem.

#As always, when debugging, you might want to run tests with a little bit of data.

#Here are the steps using Python:

#1. Hadoop has some environmental variables you can use, such as task id.

Before your main loop enter these lines:

import os

myid=os.environ["mapred_tip_id"]

#NOTE: for testing a script in a unix pipe, just set this to a string value, such as myid='pipe_test'

#2. Open a file for writing. Cloudera VM lets tasks write to /tmp, other installations might have better place to write to

mylog=open("/tmp/mymaplog"+myid,"w")

#3. Somewhere in your code add some debug statements, this can help show how the program is doing and where it might be crashing, such as:

for line in sys.stdin: #sys.stdin call 'sys' to read a line from standard input

....

mylog.write(line) #instead of line, you could put any string or string + str(integer) in there.

...

mylog.write(key)

#4. At the end of the program close the file.

mylog.close()

#5 Now you can look at the /tmp/mylog* files, try

> ls -ldth /tmp/mymaplog*

and

> more /tmp/mymaplog__ #where the __ is the log file you want to see.

4. Some people have found that running dos2unix on the wordcount_*.py files helps ensure the text is converted properly. Unix vs Dos necessary newline characters can sometimes mess things up.

5. Someone found this article useful.

http://allthingshadoop.com/2011/12/16/simple-hadoop-streaming-tutorial-using-joins-and-keys-with-python/

6. Hue will make some map/reduce job log information available:

First, you open your browser and login into the Hue. If you use the cloudera-quickstart-vm, the link looks like this: http://quickstart.cloudera:8888/jobbrowser/

Second, you click one of the failed jobs ,select the logs and go to the stderr tab.