demo2_workbook

September 5, 2022

1 wk2 Demo - Intro to Hadoop Streaming

`MIDS w261: Machine Learning at Scale | UC Berkeley School of Information`

Last week you implemented your first MapReduce Algorithm using a bash script framework. We saw that adding a sorting component to our framework allowed us to write a more efficient reducer script and perform word counting in parallel. In this notebook, we'll introduce a new framework: Hadoop Streaming. Like before, you'll write mapper and reducer scripts in python then pass them to the framework which will stream over your input files, split them into chunks and sort to your specification. Although Hadoop Streaming is rarely used in production anymore it is the precursor to systems like Spark and as such is a useful way to illustrate key concepts in parallel computation. By the end of this live session you should be able to: * **… describe** the main components and default behavior of the Hadoop Streaming framework. * **… write** a Hadoop MapReduce job from scratch. * **… access** the Hadoop Streaming UI and use it in debugging your jobs. * **… design** Hadoop MapReduce implementations for simple tasks like counting and ordering. * **… explain** why sorting with multiple reducers requires some extra work (as opposed to sorting with a single reducer).

Note: Hadoop Streaming syntax is very particular. Make sure to test your python scripts before passing them to the Hadoop job and pay careful attention to the order in which Hadoop job parameters are specified.

1.0.1 Notebook Set-Up

[4]:
```
!hadoop version
```

```
Hadoop 3.2.3
Source code repository https://bigdataoss-
internal.googlesource.com/third_party/apache/hadoop -r
b85070eb738c0d1fbf983f438199bfab3558d7b0
Compiled by bigtop on 2022-06-29T08:08Z
Compiled with protoc 2.5.0
From source with checksum aa57b5f3392f84a8f2729b81819a65d4
This command was run using /usr/lib/hadoop/hadoop-common-3.2.3.jar
```

[5]:
```
# adjust to current directory
%cd /media/notebooks/LiveSessionMaterials/wk02Demo_IntroToHadoop
```

```
/media/notebooks/LiveSessionMaterials/wk02Demo_IntroToHadoop
```

1.1 Hadoop Streaming Docs

https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html
Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

[3]:
```
# reloads
%reload_ext autoreload
%autoreload 2
```

For convenience, let's set a few global variables for paths you'll use frequently. **NOTE:** *you may need to modify the jar file and HDFS (or local home) directory paths to match your environment. The paths below should work on the course Docker image. Refer to* this debugging FAQ *if you are unsure of the correct paths or encounter errors.*

[6]:
```
# path to java archive files for the hadoop streaming application on your␣
 ↪machine
JAR_FILE = "/usr/lib/hadoop/hadoop-streaming.jar"
```

[7]:
```
# hdfs directory where  we'll store files for this assignment
HDFS_DIR = "/user/root/demo2"
!hdfs dfs -mkdir -p {HDFS_DIR}
```

[8]:
```
!hdfs dfs -ls
```

```
Found 1 items
drwxr-xr-x   - root hadoop          0 2022-08-30 12:31 demo2
```

[9]:
```
# get path to notebook
PWD = !pwd
```

[10]:
```
PWD
```

[10]:
```
['/media/notebooks/LiveSessionMaterials/wk02Demo_IntroToHadoop']
```

[11]:
```
PWD = PWD[0]
```

[12]:
```
PWD
```

[12]:
```
'/media/notebooks/LiveSessionMaterials/wk02Demo_IntroToHadoop'
```

[13]:
```
# store notebook environment path
from os import environ
PATH  = environ['PATH']
# NOTE: we can pass this variable to our Hadoop Jobs using -cmdenv PATH={PATH}
# This will ensure that, among other things, Hadoop uses the right python␣
 ↪version.
```

```
# You should not *need* this until the very last question in part 1.
```

1.1.1 Load the Data

In this notebook, we'll continue working with the *Alice in Wonderland* text file from HW1 and the test file we created for debugging. Run the following cell to confirm that you have access to these files and save their location to a global variable to use in your Hadoop Streaming jobs.

[14]:
```
# make a data subfolder - RUN THIS CELL AS IS
!mkdir -p data/
!gsutil cp gs://w261-hw-data/main/Assignments/HW1/data/alice.txt data/alice.txt
```

```
Copying gs://w261-hw-data/main/Assignments/HW1/data/alice.txt…
/ [1 files][170.2 KiB/170.2 KiB]
Operation completed over 1 objects/170.2 KiB.
```

[14]:
```
# (Re)Download Alice Full text from Project Gutenberg - RUN THIS CELL AS IS
# NOTE: feel free to replace 'curl' with 'wget' or equivalent command of your␣
↪choice.
# !curl "http://www.gutenberg.org/files/11/11-0.txt" -o data/alice.txt
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  170k  100  170k    0     0   238k      0 --:--:-- --:--:-- --:--:--  238k
```

[15]:
```
%%writefile data/alice_test.txt
This is a small test file. This file is for a test.
This small test file has two small lines.
```

```
Overwriting data/alice_test.txt
```

[16]:
```
# save the paths - RUN THIS CELL AS IS (if Option 1 failed)
ALICE_TXT = PWD + "/data/alice.txt"
TEST_TXT = PWD + "/data/alice_test.txt"
```

[17]:
```
# confirm the files are there - RUN THIS CELL AS IS
!echo "######### alice.txt #########"
!head -n 6 {ALICE_TXT}
!echo "######### alice_test.txt #########"
!cat {TEST_TXT}
```

```
######### alice.txt #########
The Project Gutenberg eBook of Alice's Adventures in Wonderland, by Lewis
Carroll

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
```

```
of the Project Gutenberg License included with this eBook or online at
######### alice_test.txt #########
This is a small test file. This file is for a test.
This small test file has two small lines.
```

2 Content Overview: MapReduce Programming Paradigm

The week 2 reading from *Data Intensive Text Processing with Map Reduce* by Lin and Dyer gave a high level overview of the key issues faced by parallel computation frameworks. It also introduced the Hadoop MapReduce framework. Lets start by briefly reviewing some of the key concepts from this week's async:

> **DISCUSSION QUESTIONS:**
> * What is MapReduce? How does it differ from Hadoop?
> * What are the main ideas of the functional programming paradigm and how does MapReduce exemplify these ideas? * What is the basic data structure used in Hadoop MapReduce? * What does 'data/code co-location' mean? How does this principle contribute to the efficiency of a distributed computation? * What is a race condition in the context of parallel computation? Give an example. * What kind of *synchronization* does Hadoop MapReduce perform by default? * What aspect of the synchronization process is computationally costly? * What does Hadoop sort by? What is Hadoop's default sort order? * Throughout this course we'll emphasize the goal of writing 'stateless' implementations? What does that mean?

2.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS * What is MapReduce? How does it differ from Hadoop? > MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. It has two phases: 'map' which applies a transformation(or filter) to all elements of a collection, and 'reduce' which aggregates/folds the elements of a collection. The term "MapReduce" is also commonly used to refer to an execution framework built around this paradigm. Hadoop MapReduce is one such popular open-source framework.

- What are the main ideas of the functional programming paradigm and how does MapReduce exemplify these ideas? > Hadoop MapReduce is based on the functional programing paradigm. It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function. Key ideas in this paradigm include avoiding state changes/mutable data, and using higher-order functions (functions which take other functions as inputs. Map and reduce are examples of higher-order functions.
- What is the basic data structure used in Hadoop MapReduce? >Hadoop MapReduce processes data in key-value pairs.
- What does 'data/code co-location' mean? How does this principle contribute to the efficiency of a distributed computation? > Data/code "colocation" is a principle in distributed processing by which we send the code to the node where a chunk of data is stored and perform the computation on that node. This is efficient because it limits network traffic.
- What is a race condition in the context of parallel computation? Give an example. >

A race condition happens when multiple execution entities (e.g. threads, processes, etc) are accessing or modifying the same resource at the same time, and the order of access can impact the result. For example, suppose two concurrent threads or processes A and B attempt to increment the same memory location. Without proper synchronization, chances are that the following sequence will occur: A reads, B reads, A writes, B writes - with the final value being incremented only once. To overcome this conditon, the global/count variable must be locked during the update; otherwise a thread can read the count while other threads are updating it, and the final accumulation will be less than the true value.

- What kind of *synchronization* does Hadoop MapReduce perform by default? > Hadoop MapReduce performs synchronization using a barrier between the map and reduce phases. The reduce phase will not start until all mappers are complete. Then Hadoop reorganizes (a.k.a. 'shuffles') the mapper output which will be input to the reduce phase which will receive records sorted by key.
- What aspect of the synchronization process is computationally costly? > This 'shuffle' is computationally costly because of the sorting and network transfer involved.
- What does Hadoop sort by? What is Hadoop's default sort order? > By default, hadoop sorts by key in alpha-numeric order.
- Throughout this course we'll emphasize the goal of writing 'stateless' implementations? What does that mean? > We don't want our implementations to depend on any shared mutable information. This is related to the idea of avoiding race conditions & the optimality of 'embarassingly parallel' implementations. Note that we also like to limit the use of memory given that we are likely using a cluster of cheap machines that may have limited space.... but that in-memory aggregation on individual nodes is not a violation of the principle of 'statelessness' (though it might seem like we are storing a 'state' in that scenario, the 'statefulness' that we want to avoid is any shared information that needs to be updated across multiple nodes.)

3 Preview: Hadoop Streaming Syntax

A basic Hadoop MapReduce job consists of three components: a mapper script, a reducer script and a line of code in which you pass these scripts to the Hadoop streaming application/framework. The mapper and reducer can be any executable that will read from `stdin` and write to `stdout`, including a bash executable like`/bin/cat` which simply passes the lines of your file unchanged, or python scripts like the ones you wrote in HW1.

To run your hadoop streaming job, you'll need an HDFS filepath for the input data. We can use the following line of code to load a local file into HDFS:

[18]:
```
# put the alice test file into HDFS - RUN THIS CELL AS IS
!hdfs dfs -put {TEST_TXT} {HDFS_DIR}
```

TIP: *Recall that we set the global variable* `HDFS_DIR` *above to point to a directory called* `demo2` *inside* `/user/root/`*, you confirm that we've successfully loaded the data into HDFS using the following line. You can learn more about this command and others like it by reading the Hadoop File System Shell Guide.*

[19]:
```
# confirm the data was loaded - RUN THIS CELL AS IS
!hdfs dfs -ls {HDFS_DIR}
```

```
Found 1 items
-rw-r--r--   1 root supergroup         94 2021-01-16 18:54
/user/root/demo2/alice_test.txt
```

Ok, with that little bit of preparation, we can now run the most basic Hadoop Streaming Job.

> **DISCUSSION QUESTIONS** (*before you run the next cell …*)
> * Read the 6 lines below, what do each of them tell the framework to do? * What to the forward slashes indicate? * What do you expect to happen when we run this cell? (what should the output be? will it be printed to the console?)

[19]:
```
# first hadoop streaming job - RUN THIS CELL AS IS
!hdfs dfs -rm -r {HDFS_DIR}/test-output
```

```
rm: `/user/root/demo2/test-output': No such file or directory
```

[20]:
```
!hdfs dfs -rm -r {HDFS_DIR}/test-output
!hadoop jar {JAR_FILE} \
  -mapper /bin/cat \
  -reducer /bin/cat \
  -input {HDFS_DIR}/alice_test.txt \
  -output {HDFS_DIR}/test-output \
  -cmdenv PATH={PATH}
```

```
rm: `/user/root/demo2/test-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob7411907185031742716.jar tmpDir=null
2022-08-30 12:34:28,620 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:34:28,851 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:34:29,465 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:34:29,465 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:34:30,147 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0001
2022-08-30 12:34:30,475 INFO mapred.FileInputFormat: Total input files to
process : 1
2022-08-30 12:34:30,578 INFO mapreduce.JobSubmitter: number of splits:10
2022-08-30 12:34:31,214 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0001
2022-08-30 12:34:31,215 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 12:34:31,412 INFO conf.Configuration: resource-types.xml not found
2022-08-30 12:34:31,413 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
```

```
2022-08-30 12:34:31,937 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0001
2022-08-30 12:34:32,022 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0001/
2022-08-30 12:34:32,025 INFO mapreduce.Job: Running job: job_1661861832644_0001
2022-08-30 12:34:44,323 INFO mapreduce.Job: Job job_1661861832644_0001 running
in uber mode : false
2022-08-30 12:34:44,325 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 12:34:53,437 INFO mapreduce.Job:  map 30% reduce 0%
2022-08-30 12:35:00,503 INFO mapreduce.Job:  map 50% reduce 0%
2022-08-30 12:35:01,509 INFO mapreduce.Job:  map 60% reduce 0%
2022-08-30 12:35:08,569 INFO mapreduce.Job:  map 70% reduce 0%
2022-08-30 12:35:09,574 INFO mapreduce.Job:  map 90% reduce 0%
2022-08-30 12:35:13,595 INFO mapreduce.Job:  map 100% reduce 0%
2022-08-30 12:35:21,672 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 12:35:22,680 INFO mapreduce.Job:  map 100% reduce 67%
2022-08-30 12:35:23,690 INFO mapreduce.Job:  map 100% reduce 100%
2022-08-30 12:35:25,709 INFO mapreduce.Job: Job job_1661861832644_0001 completed
successfully
2022-08-30 12:35:25,798 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=118
                FILE: Number of bytes written=3233358
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1450
                HDFS: Number of bytes written=96
                HDFS: Number of read operations=45
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=10
                Launched reduce tasks=3
                Data-local map tasks=10
                Total time spent by all maps in occupied slots (ms)=207387072
                Total time spent by all reduces in occupied slots (ms)=54785004
                Total time spent by all map tasks (ms)=65712
                Total time spent by all reduce tasks (ms)=17359
                Total vcore-milliseconds taken by all map tasks=65712
                Total vcore-milliseconds taken by all reduce tasks=17359
                Total megabyte-milliseconds taken by all map tasks=207387072
                Total megabyte-milliseconds taken by all reduce tasks=54785004
        Map-Reduce Framework
                Map input records=2
                Map output records=2
                Map output bytes=96
                Map output materialized bytes=280
                Input split bytes=960
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=280
                Reduce input records=2
                Reduce output records=2
                Spilled Records=4
                Shuffled Maps =30
                Failed Shuffles=0
                Merged Map outputs=30
                GC time elapsed (ms)=2162
                CPU time spent (ms)=16070
                Physical memory (bytes) snapshot=6691561472
                Virtual memory (bytes) snapshot=57763381248
                Total committed heap usage (bytes)=5576327168
                Peak Map Physical memory (bytes)=597680128
                Peak Map Virtual memory (bytes)=4452065280
                Peak Reduce Physical memory (bytes)=307916800
                Peak Reduce Virtual memory (bytes)=4450656256
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=490
        File Output Format Counters
                Bytes Written=96
2022-08-30 12:35:25,798 INFO streaming.StreamJob: Output directory:
/user/root/demo2/test-output
```

Note that running the cell above doesn't actually show us the results. That's because the results get written directly to the output directory in HDFS. You can view the results using an `hdfs` command. (**`Note:`** `test-output` *is an HDFS directory that was created by the 5th line in the last cell, we could have named it anything we wanted*):

[21]:
```
# view the contents of the result directory - RUN THIS CELL AS IS
!hdfs dfs -ls {HDFS_DIR}/test-output/
```

```
Found 4 items
-rw-r--r--   1 root hadoop          0 2022-08-30 12:35 /user/root/demo2/test-
output/_SUCCESS
-rw-r--r--   1 root hadoop         53 2022-08-30 12:35 /user/root/demo2/test-
output/part-00000
```

```
-rw-r--r--   1 root hadoop          0 2022-08-30 12:35 /user/root/demo2/test-
output/part-00001
-rw-r--r--   1 root hadoop         43 2022-08-30 12:35 /user/root/demo2/test-
output/part-00002
```

[22]:
```
# view the results themselves - RUN THIS CELL AS IS
!hdfs dfs -cat {HDFS_DIR}/test-output/part-00000 | head
```

```
This is a small test file. This file is for a test.
```

> `DISCUSSION QUESTIONS` (*after running the Hadoop Job.*)
> * Do the results match your expectations? * Scan the Hadoop Job logging, what information stands out to you? * When would it be a bad idea to print the full results of a job to the console? what could we do instead? * What goes wrong if you try re-running the Hadoop Streaming job? Discuss two potential solutions to this problem.

3.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS * When discussing results point out the /bin/cat again… these are the 'unity' mapper and reducer… they just return the lines they receive.

- When discussing the loggings goal here is just to get students familiar with how this looks… they may notice information about time & memory usage, or the count of mappers and reducers. We'll get into the Hadoop UI explicilty a bit later but you may choose to point this out now.
- When would it be a bad idea to print the full results of a job to the console? what could we do instead? > For large output this is a waste of space. For really large output this could cause your notebook to crash. HDFS has a head command.
- What goes wrong if you try re-running the Hadoop Streaming job? Discuss two potential solutions to this problem. > Goal here is for students to start to be comfortable receiving an error message. In this case, the Hadoop Job won't run if the output directory you specify is not empty. Two solutions 1) rename the output directory when you re-run the command (this will duplicate results) OR better still 2) use a Hadoop command like the one below to empty the output directory each time you re-run a job.

[24]:
```
# <--- SOLUTION --->
# clear the output directory before re-running a job. Eg.
!hdfs dfs -rm -r {HDFS_DIR}/test-output
```

```
Deleted /user/root/demo2/test-output
```

4 Breakout 1: WordCount in Hadoop MapReduce

For most of the Hadoop jobs you write this week, you will use python scripts similar to those you wrote in HW1 to serve as your mapper and reducer. Since the Hadoop Streaming framework implements the principle of data/code co-location, we'll need to provide the paths to these python scripts so that Hadoop can ship them to the nodes where the code will get run. We do this by adding an additional flag to the Hadoop streaming command: the `-files` parameter. This will be the first of a number of additional flags that you can added to your Hadoop Streaming jobs to specify how the framework should handle your data. `TIP:` *Hadoop can be very particular about the order in which you specify optional configuration fields. As a good debugging practice we recommend that you always maintain working code by starting with a basic job like the one we provided above and then testing the command as you add or modify parameters one by one.*

For your first breakout activity we've provided an example of a Hadoop MapReduce job that performs word counting on an input file of your choice. The mapper and reducer are python scripts provided at `WordCount/mapper.py` and `WordCount/reducer.py`. The Hadoop Streaming command is in a cell below. As you read through the example and go on to write your own Hadoop MapReduce jobs you may want to refer to Michael Noll's blogpost on writing an MapReduce job and/or the Hadoop Streaming documentation.

4.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (before breakout 1) * Breakouts can be awkward, you should jump in as quickly as possible you'll have just 10min for this one.

- Note: for a lot of these tasks you'll be reading & modifying the python scripts in folders inside the current directory… highly recommend Jupyter lab's split screen for this.

4.0.2 Breakout 1 Tasks:

- **a) read scripts & docstrings:** Read through `WordCount/mapper.py` and `WordCount/reducer.py` scripts and pay attention to the docstrings. Note that they (briefly) explain what the script does and the expected input/output record formats. [`HINT:` *docstrings are a way to record information to help your reader (future-self/collaborator/grader) quickly orient to a piece of code. They should describe* **what** (*not* how) *is being done. For more information refer to the PEP 8 Style Guide for Python*] The use of docstrings is recommended in all code that is written for this class.
- **b) discuss:** What are the 'keys' and what are the 'values' in this MapReduce job? What delimiter separates them when we write to standard output? How will you expect Hadoop to sort the records emitted by the mapper script? Why is this order important given how the reducer script is written?
- **c) run provided code:** Run the cells provided to make sure that your mapper and reducer scripts are executable, load the input files into HDFS, and clear the HDFS directory where the job will write its output. You will need to do these preparation steps for all future Hadoop MapReduce jobs.
- **d) unit test:** A good habit when writing Hadoop streaming jobs is to test your mappers and reducers locally before passing them to a Hadoop streaming command. An easy way to do this is to pipe in a small line of text. We've provided the unix code to do so and added a unix sort to mimic Hadoop's default sorting. Run these cells to confirm that our mapper and reducer work properly. (Observe how the reducer doesn't work without the sort).
- **e) code:** We've provided the code to run your Hadoop streaming command on the test file. Read through this command and be sure you understand each parameter that we're passing in, then run it and confirm that the output performs word counting correctly. Finally, modify the code provided to run the Hadoop MapReduce job on the *Alice and Wonderland* text

instead of the test file. Remember that the input path you pass to the Hadoop streaming command should be a location in HDFS not a local path. Take a look at the output and confirm you get the same count for 'alice' as in HW1. Food for thought: *does the sorting match what you expected?*

part c Prep for Hadoop Streaming Job

```
[23]: # part c - make sure the mapper and reducer are executable (RUN THIS CELL AS IS)
!chmod a+x WordCount/mapper.py
!chmod a+x WordCount/reducer.py
```

```
[24]: # part c - load the input files into HDFS (RUN THIS CELL AS IS)
!hdfs dfs -copyFromLocal {TEST_TXT} {HDFS_DIR}
!hdfs dfs -copyFromLocal {ALICE_TXT} {HDFS_DIR}
```

```
copyFromLocal: `/user/root/demo2/alice_test.txt': File exists
```

```
[25]: # part c - clear the output directory (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/wordcount-output
# NOTE: this directory won't exist unless you are re-running a job, that's fine.
```

```
rm: `/user/root/demo2/wordcount-output': No such file or directory
```

part d Unit test your scripts.

```
[26]: # part d - unit test mapper script
!echo "foo foo quux labs foo bar quux" | WordCount/mapper.py
```

```
foo     1
foo     1
quux    1
labs    1
foo     1
bar     1
quux    1
```

```
[27]: # part d - unit test reducer script
!echo␣
 ↪"foo         1\nfoo          1\nquux          1\nlabs          1\nfoo          1\nbar          1\nquux
 ↪| WordCount/reducer.py
```

```
foo     2
quux    1
labs    1
foo     1
bar     1
quux    1
```

```
[28]: # part d - systems text mapper and reducer together
      !echo "foo foo quux labs foo bar quux" | WordCount/mapper.py | WordCount/
       ↪reducer.py
```

```
foo     2
quux    1
labs    1
foo     1
bar     1
quux    1
```

```
[29]: # part d - systems text mapper and reducer together with sort (RUN THIS CELL AS␣
       ↪IS)
      !echo "foo foo quux labs foo bar quux" | WordCount/mapper.py | sort -k1,1 |␣
       ↪WordCount/reducer.py
```

```
bar     1
foo     3
labs    1
quux    2
```

part e Hadoop streaming command. **NOTE:** *don't forget to clear the output directory before re-running this cell (see part b above)*

```
[30]: # part e - Hadoop streaming job (RUN THIS CELL AS IS FIRST, then make your␣
       ↪modification)
      !hadoop jar {JAR_FILE} \
        -files WordCount/reducer.py,WordCount/mapper.py \
        -mapper mapper.py \
        -reducer reducer.py \
        -input {HDFS_DIR}/alice_test.txt \
        -output {HDFS_DIR}/wordcount-output \
        -cmdenv PATH={PATH}
```

```
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob6312471413240690045.jar tmpDir=null
2022-08-30 12:38:17,215 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:38:17,470 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:38:17,986 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:38:17,986 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:38:18,201 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0002
2022-08-30 12:38:18,584 INFO mapred.FileInputFormat: Total input files to
process : 1
```

```
2022-08-30 12:38:18,660 INFO mapreduce.JobSubmitter: number of splits:10
2022-08-30 12:38:18,912 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0002
2022-08-30 12:38:18,915 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 12:38:19,141 INFO conf.Configuration: resource-types.xml not found
2022-08-30 12:38:19,141 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2022-08-30 12:38:19,222 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0002
2022-08-30 12:38:19,262 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0002/
2022-08-30 12:38:19,263 INFO mapreduce.Job: Running job: job_1661861832644_0002
2022-08-30 12:38:28,393 INFO mapreduce.Job: Job job_1661861832644_0002 running
in uber mode : false
2022-08-30 12:38:28,394 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 12:38:37,550 INFO mapreduce.Job:  map 30% reduce 0%
2022-08-30 12:38:44,635 INFO mapreduce.Job:  map 40% reduce 0%
2022-08-30 12:38:45,640 INFO mapreduce.Job:  map 60% reduce 0%
2022-08-30 12:38:52,706 INFO mapreduce.Job:  map 70% reduce 0%
2022-08-30 12:38:53,712 INFO mapreduce.Job:  map 90% reduce 0%
2022-08-30 12:38:56,737 INFO mapreduce.Job:  map 100% reduce 0%
2022-08-30 12:39:04,792 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 12:39:06,809 INFO mapreduce.Job:  map 100% reduce 100%
2022-08-30 12:39:08,826 INFO mapreduce.Job: Job job_1661861832644_0002 completed
successfully
2022-08-30 12:39:08,921 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=189
                FILE: Number of bytes written=3253923
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1450
                HDFS: Number of bytes written=64
                HDFS: Number of read operations=45
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=10
                Launched reduce tasks=3
                Data-local map tasks=10
                Total time spent by all maps in occupied slots (ms)=208425396
                Total time spent by all reduces in occupied slots (ms)=52746228
                Total time spent by all map tasks (ms)=66041
                Total time spent by all reduce tasks (ms)=16713
                Total vcore-milliseconds taken by all map tasks=66041
                Total vcore-milliseconds taken by all reduce tasks=16713
                Total megabyte-milliseconds taken by all map tasks=208425396
                Total megabyte-milliseconds taken by all reduce tasks=52746228
        Map-Reduce Framework
                Map input records=2
                Map output records=20
                Map output bytes=131
                Map output materialized bytes=351
                Input split bytes=960
                Combine input records=0
                Combine output records=0
                Reduce input groups=10
                Reduce shuffle bytes=351
                Reduce input records=20
                Reduce output records=10
                Spilled Records=40
                Shuffled Maps =30
                Failed Shuffles=0
                Merged Map outputs=30
                GC time elapsed (ms)=1748
                CPU time spent (ms)=18840
                Physical memory (bytes) snapshot=6852063232
                Virtual memory (bytes) snapshot=57740460032
                Total committed heap usage (bytes)=5848432640
                Peak Map Physical memory (bytes)=605978624
                Peak Map Virtual memory (bytes)=4443934720
                Peak Reduce Physical memory (bytes)=305381376
                Peak Reduce Virtual memory (bytes)=4447514624
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=490
        File Output Format Counters
                Bytes Written=64
2022-08-30 12:39:08,922 INFO streaming.StreamJob: Output directory:
/user/root/demo2/wordcount-output
```

[31]:
```
# <--- SOLUTION --->
# part e - Hadoop streaming job (RUN THIS CELL AS IS FIRST, then make your␣
↪modification)
!hdfs dfs -rm -r {HDFS_DIR}/wordcount-output
!hadoop jar {JAR_FILE} \
  -files WordCount/reducer.py,WordCount/mapper.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/wordcount-output \
  -cmdenv PATH={PATH}
```

```
Deleted /user/root/demo2/wordcount-output
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob7300702197210842496.jar tmpDir=null
2022-08-30 12:39:14,444 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:39:14,731 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:39:15,246 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:39:15,247 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:39:15,503 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0003
2022-08-30 12:39:15,934 INFO mapred.FileInputFormat: Total input files to
process : 1
2022-08-30 12:39:16,029 INFO mapreduce.JobSubmitter: number of splits:9
2022-08-30 12:39:16,284 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0003
2022-08-30 12:39:16,286 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 12:39:16,524 INFO conf.Configuration: resource-types.xml not found
2022-08-30 12:39:16,524 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2022-08-30 12:39:16,595 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0003
2022-08-30 12:39:16,633 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0003/
2022-08-30 12:39:16,635 INFO mapreduce.Job: Running job: job_1661861832644_0003
2022-08-30 12:39:25,771 INFO mapreduce.Job: Job job_1661861832644_0003 running
in uber mode : false
2022-08-30 12:39:25,772 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 12:39:34,905 INFO mapreduce.Job:  map 22% reduce 0%
2022-08-30 12:39:35,915 INFO mapreduce.Job:  map 33% reduce 0%
2022-08-30 12:39:41,982 INFO mapreduce.Job:  map 44% reduce 0%
2022-08-30 12:39:42,993 INFO mapreduce.Job:  map 56% reduce 0%
2022-08-30 12:39:44,002 INFO mapreduce.Job:  map 67% reduce 0%
2022-08-30 12:39:49,046 INFO mapreduce.Job:  map 78% reduce 0%
2022-08-30 12:39:50,052 INFO mapreduce.Job:  map 100% reduce 0%
2022-08-30 12:39:59,125 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 12:40:00,131 INFO mapreduce.Job:  map 100% reduce 67%
2022-08-30 12:40:01,138 INFO mapreduce.Job:  map 100% reduce 100%
```

```
2022-08-30 12:40:03,154 INFO mapreduce.Job: Job job_1661861832644_0003 completed
successfully
2022-08-30 12:40:03,245 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=276444
                FILE: Number of bytes written=3556059
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=28488
                HDFS: Number of read operations=42
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=9
                Launched reduce tasks=3
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=188918160
                Total time spent by all reduces in occupied slots (ms)=57120444
                Total time spent by all map tasks (ms)=59860
                Total time spent by all reduce tasks (ms)=18099
                Total vcore-milliseconds taken by all map tasks=59860
                Total vcore-milliseconds taken by all reduce tasks=18099
                Total megabyte-milliseconds taken by all map tasks=188918160
                Total megabyte-milliseconds taken by all reduce tasks=57120444
        Map-Reduce Framework
                Map input records=3761
                Map output records=30564
                Map output bytes=215298
                Map output materialized bytes=276588
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=3006
                Reduce shuffle bytes=276588
                Reduce input records=30564
                Reduce output records=3006
                Spilled Records=61128
                Shuffled Maps =27
                Failed Shuffles=0
                Merged Map outputs=27
                GC time elapsed (ms)=1989
                CPU time spent (ms)=18460
                Physical memory (bytes) snapshot=6148235264
                Virtual memory (bytes) snapshot=53300072448
                Total committed heap usage (bytes)=5204082688
                Peak Map Physical memory (bytes)=616775680
                Peak Map Virtual memory (bytes)=4451176448
                Peak Reduce Physical memory (bytes)=295714816
                Peak Reduce Virtual memory (bytes)=4443209728
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=28488
2022-08-30 12:40:03,245 INFO streaming.StreamJob: Output directory:
/user/root/demo2/wordcount-output
```

[32]:
```
# part e - retrieve results from HDFS & copy them into a local file (RUN THIS
↪CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/wordcount-output/part-0000* > WordCount/results.txt
# NOTE: we would never do this for a really large output file!
# (but it's convenient for illustration in this assignment)
```

[33]:
```
# part e - view results (RUN THIS CELL AS IS)
!head WordCount/results.txt
# NOTE: these words and counts should match your results in HW1
```

```
able    1
about   102
accepted        2
account 1
accounts        1
ache    1
actual  1
adjourn 1
advance 3
after   43
```

[34]:
```
# part e - check 'alice' count (RUN THIS CELL AS IS after running the job on
↪the full file)
!grep 'alice' WordCount/results.txt
# EXPECTED OUTPUT: 403
```

```
alice   403
```

DISCUSSION QUESTIONS (after breakout 1) * What are the 'keys' and what are the 'values' in this MapReduce job? What delimiter separates them when we write to standard output? How will you expect Hadoop to sort the records emitted by the mapper script? Why is this order important given how the reducer script is written? * What was the default sorting you saw? When do you think this sorting happens? * Why go to the trouble of writing the reducer this way? why not just use a dictionary to count the words?

4.0.3 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (after breakout 1) * What are the 'keys' and what are the 'values' in this MapReduce job? What delimiter separates them when we write to standard output? How will you expect Hadoop to sort the records emitted by the mapper script? Why is this order important given how the reducer script is written? > The keys are words the values are counts. The default delimiter in Hadoop streaming is a tab, however other delimiters (eg. comma, backslash, etc) could be specified. Hadoop will sort the records alphabetically by key (in our case, the word). This is important because the reducer will stream over the input records and emit the sum of consecutive counts (that belong to the same word). If the records were not alphabetized then the reducer might emit the same word twice without fully adding up its occurences.

- What was the default sorting you saw? When do you think this sorting happens? > Sort by key, this happens between mapper & reducer (reducer needs it in order to work properly!)
- Why go to the trouble of writing the reducer this way? why not just use a dictionary to count the words? > This goes back to the lesson from HW1… if we can't guarantee how much memory will be available in the nodes in our cluster of cheap comodity hardware, we want to try storing as little information as possible.

5 Breakout 2: Uppercase and Lowercase Counts

What if we didn't care about individual word counts but rather wanted to know how many of the words in the *Alice* file are upper and lower case? We could retrieve this information easily using a Hadoop streaming job with the same reducer script as in Section 2 (i.e. **`WordCount/reducer.py`**) but a slightly different mapper. In this question you'll design and write your own Hadoop streaming job to do just this.

> ___ DISCUSSION QUESTION:___
> * What should the keys be for this task? [`HINT:` *we'll need a key for each thing we want to count.*]

5.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (before breakout 2) * What should the keys be for this task? > We only need two keys: 'upper' and 'lower' (or something equivalent).

- NOTE: by 'upper case' we mean the first letter is uppercase, not the whole word.

5.0.2 Breakout 2 Tasks:

- **a) code:** Complete the docstring and code in the **`UpperLower/mapper.py`** to create a mapper that reads each input line, splits it into words and emits an appropriate key-value pair for each one. [`HINT:` *we're going to use this mapper in conjunction with the reducer from Breakout 1 so your key-value format should look very similar to the one in Breakout 1's mapper.*]

- **b) unit test:** Run the provided cells to make your new mapper executable and test that it works as you expect.
- **c) code:** We've provided the start of a Hadoop streaming command. Fill in the missing parameters following the example provided in Section 2. Run your Hadoop job on the test file to confirm that it works. When you are happy with the results replace the test filepath with the real *Alice in Wonderland* filepath and rerun the job. We'll compare results when we come back from breakouts.
- **d) discuss:** Like our bash script HW1, Hadoop automatically splits up your records to be processed in parallel on separate mapper and reducer "nodes" (called "tasks" by Hadoop). Judging from the jobs you've run so far, what are the default number of 'map tasks' and 'reduce tasks' that Hadoop uses? Does this framework allow us to directly control the number of mappers and reducers? [`HINTS`: *to answer the first part of this question, look at the "Job Counters" section in the logging from your Hadoop job; for the second part of this question refer back to Lin & Dyer p24 at the very bottom*]

[35]:
```
# part a - run this cell after completing your portion of the code
!chmod a+x UpperLower/mapper.py
```

[36]:
```
# part b - unit test your new mapper (RUN THIS CELL AS IS)
!echo "Foo foo Quux Labs foo bar quux" | UpperLower/mapper.py
```

```
upper   1
lower   1
upper   1
upper   1
lower   1
lower   1
lower   1
```

[37]:
```
# part b - systems test your new mapper with the reducer from question 2 (RUN␣
↪THIS CELL AS IS)
!echo "Foo foo Quux Labs foo bar quux" | UpperLower/mapper.py | sort -k1,1 |␣
↪WordCount/reducer.py
```

```
lower   4
upper   3
```

[38]:
```
# part c - clear output directory before (re)running your Hadoop Job (RUN THIS␣
↪CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/upperlower-output
```

```
rm: `/user/root/demo2/upperlower-output': No such file or directory
```

[ ]:
```
# part c - Hadoop streaming command (FILL IN MISSING ARGUMENTS)
!hadoop jar {JAR_FILE} \
  -files UpperLower/mapper.py,WordCount/reducer.py \
  -output {HDFS_DIR}/upperlower-output \
  -cmdenv PATH={PATH}
```

[39]:
```
# <--- SOLUTION --->
# part c - Hadoop streaming command (FILL IN MISSING ARGUMENTS)
!hadoop jar {JAR_FILE} \
  -files UpperLower/mapper.py,WordCount/reducer.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/upperlower-output \
  -cmdenv PATH={PATH}
```

```
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob4867320095547223097.jar tmpDir=null
2022-08-30 12:50:56,940 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:50:57,184 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:50:57,671 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:50:57,672 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:50:57,872 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0004
2022-08-30 12:50:58,606 INFO mapred.FileInputFormat: Total input files to
process : 1
2022-08-30 12:50:58,670 INFO mapreduce.JobSubmitter: number of splits:9
2022-08-30 12:50:58,843 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0004
2022-08-30 12:50:58,845 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 12:50:59,017 INFO conf.Configuration: resource-types.xml not found
2022-08-30 12:50:59,017 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2022-08-30 12:50:59,088 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0004
2022-08-30 12:50:59,123 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0004/
2022-08-30 12:50:59,125 INFO mapreduce.Job: Running job: job_1661861832644_0004
2022-08-30 12:51:07,232 INFO mapreduce.Job: Job job_1661861832644_0004 running
in uber mode : false
2022-08-30 12:51:07,233 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 12:51:15,367 INFO mapreduce.Job:  map 11% reduce 0%
2022-08-30 12:51:16,373 INFO mapreduce.Job:  map 33% reduce 0%
2022-08-30 12:51:23,435 INFO mapreduce.Job:  map 44% reduce 0%
```

```
2022-08-30 12:51:24,444 INFO mapreduce.Job:  map 67% reduce 0%
2022-08-30 12:51:30,507 INFO mapreduce.Job:  map 78% reduce 0%
2022-08-30 12:51:31,512 INFO mapreduce.Job:  map 100% reduce 0%
2022-08-30 12:51:38,568 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 12:51:40,581 INFO mapreduce.Job:  map 100% reduce 67%
2022-08-30 12:51:41,588 INFO mapreduce.Job:  map 100% reduce 100%
2022-08-30 12:51:42,598 INFO mapreduce.Job: Job job_1661861832644_0004 completed
successfully
2022-08-30 12:51:42,691 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=280338
                FILE: Number of bytes written=3563871
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=23
                HDFS: Number of read operations=42
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=9
                Launched reduce tasks=3
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=185894712
                Total time spent by all reduces in occupied slots (ms)=47684004
                Total time spent by all map tasks (ms)=58902
                Total time spent by all reduce tasks (ms)=15109
                Total vcore-milliseconds taken by all map tasks=58902
                Total vcore-milliseconds taken by all reduce tasks=15109
                Total megabyte-milliseconds taken by all map tasks=185894712
                Total megabyte-milliseconds taken by all reduce tasks=47684004
        Map-Reduce Framework
                Map input records=3761
                Map output records=28032
                Map output bytes=224256
                Map output materialized bytes=280482
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=280482
                Reduce input records=28032
                Reduce output records=2
                Spilled Records=56064
                Shuffled Maps =27
                Failed Shuffles=0
                Merged Map outputs=27
                GC time elapsed (ms)=1869
                CPU time spent (ms)=17740
                Physical memory (bytes) snapshot=6255329280
                Virtual memory (bytes) snapshot=53339373568
                Total committed heap usage (bytes)=5271715840
                Peak Map Physical memory (bytes)=612769792
                Peak Map Virtual memory (bytes)=4458713088
                Peak Reduce Physical memory (bytes)=315748352
                Peak Reduce Virtual memory (bytes)=4449312768
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=23
2022-08-30 12:51:42,692 INFO streaming.StreamJob: Output directory:
/user/root/demo2/upperlower-output
```

[40]:
```
# part c - results (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/upperlower-output/part-000* > UpperLower/results.txt
!cat UpperLower/results.txt
```

```
lower   25055
upper   2977
```

DISCUSSION QUESTIONS (after breakout 2) * How many uppercase/lowercase words did you find? * How many map tasks? how many reduce tasks? Where did you find this information?

5.0.3 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (after breakout 2) * How many uppercase/lowercase words did you find? > `lower    25055 upper  2977`

- How many map tasks? how many reduce tasks? Where did you find this information? > By default Hadoop uses (at least) 2 mappers and 1 reducer. We can specify the number of reducers it should use and suggest a number of mappers, but we can't force it to use the number of mappers we suggest. Hadoop will make a determination based on the data size and location (and block size in HDFS).

6 Breakout 3: Number of Unique Words

Another variation on the simple word counting job would be to count the number of unique words in the book (instead of counting the unique occurrences of each word). Of course in reality the easiest way to get this information would be to count the number of lines in our word count output file… but since our goal here is to practice designing and writing Hadoop jobs, let's pretend for a moment that we don't have access to that file and instead think about how we'd do this from scratch. In this question we'll also introduce an important flag you can add to your Hadoop streaming jobs to control the degree of parallelization.

> **DISCUSSION QUESTIONS:**
> * What should our keys be for this task? * Does it make most sense to check for 'uniqueness' inside the mapper or inside the reducer? why? [`HINT:` *think about our discussion of memory constraints in HW1 and about the synchronization that Hadoop performs for us automatically between the map and reduce phases.*]

6.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (before breakout 3) * what should the keys/values be? > The keys should be words –> we need to sort these inorder to remove duplicates (& get the unique count)

- where should we check 'uniqueness'? > In the reducer after we've sorted (if we do it in mappers we could end up with duplicates). Filtering for unique words inside the mapper would require maintaining a list of 'words seen' which could be memory intensive. Even if we decided to do this local aggregation, there could be duplicate keys emitted by map tasks on different nodes which means we can't fully filter out non-unique words in the map phase alone (unless we were to guarantee a single map task). Since the shuffle phase automatically groups records for each key together it will be easy and effcient to handle deduplication in the reducer.

6.0.2 Breakout 3 Tasks:

- **a) code + unit test:** Since the mapper we wrote for `WordCount` already emits the right keys, let's simply reuse that mapper. Fill in the missing code in **`VocabSize/reducer.pyso` that this new reducer processes the records emitted by `WordCount/mapper.py`** and outputs the count of the number of unique words that appear in the input file. Run the provided unit test to confirm that your reducer works as you want it to.
- **b) code:** Write and run a Hadoop streaming job to calculate the number of unique words in *Alice and Wonderland* and record it in the space provided (`NOTE:` *for 'c' you'll modify this job and overwrite the original result which is why we'll ask you to record it in markdown.*)
- **c) code + discussion:** Add the flag `-numReduceTasks 3` to the very end of the Hadoop streaming command you wrote for `part c`. This flag tells Hadoop to use 3 separate reduce tasks, in other words, we'll make 3 'partitions' from the records emitted by your map phase and perform the reducing on each part. Rerun the job with this added flag and observe the result.
  - What do you notice about the contents of the HDFS output directory and the final output itself? How would we have to post process our results to get the answer we're

looking for?

- **d) Hadoop UI:** In addition to the logging that Hadoop prints to your notebook you can also access two UIs with more detailed information about your Hadoop streaming jobs. While your job is currently running you can track its progress on port `8088` (this will be especially helpful in latter assignments when we run jobs that may take a long time).The link to this 'Running Job Tracker' UI can be found near the top of the logging from your job. Look for a line that reads something like: >`The url to track the job: http://quickstart.cloudera:8088/proxy/application_########_#####/`

Once the job has completed, this link will redirect to the 'MapReduce Job History UI' on port `19888`. This is where you can access information about completed jobs (note the Job ID number will match the one printed in the URL above). > `localhost:19888/jobhistory/job/job_########_#####`

For `part d` Navigate to the MapReduce Job History UI (the one on port `19888`) and confirm that your job used 2 map tasks and 3 reduce tasks.

[41]:
```
# part b - write your code in the provided script first, then RUN THIS CELL AS␣
↪IS
!chmod a+x VocabSize/reducer.py
```

[42]:
```
# part b - unit test your new reducer (RUN THIS CELL AS IS)
!echo␣
↪"foo        1\nfoo        1\nquux        1\nlabs        1\nfoo        1\nbar        1\nquux
↪| sort -k1,1 | VocabSize/reducer.py
```

```
NumUniqueWords  4
```

[43]:
```
# part c - clear output directory before (re)running your Hadoop Job (RUN THIS␣
↪CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/vocabsize-output
```

```
rm: `/user/root/demo2/vocabsize-output': No such file or directory
```

`TIPS:` *When writing your job below make sure that you have the correct paths to your input file, output directory and mapper/reducer script. Don't forget the* `-files` *option, and* DO NOT *put spaces between the paths that you pass to this option.*

[ ]:
```
# parts b/c - write/modify your Hadoop streaming command here:
```

[44]:
```
# <--- SOLUTION --->
# parts b/c - write/modify your Hadoop streaming command here:
!hdfs dfs -rm -r {HDFS_DIR}/vocabsize-output
!hadoop jar {JAR_FILE} \
  -files WordCount/mapper.py,VocabSize/reducer.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/vocabsize-output \
  -numReduceTasks 3 \
  -cmdenv PATH={PATH}
```

```
rm: `/user/root/demo2/vocabsize-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob8922807512985194972.jar tmpDir=null
2022-08-30 12:54:09,945 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:54:10,183 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:54:10,665 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 12:54:10,666 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 12:54:10,874 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0005
2022-08-30 12:54:11,674 INFO mapred.FileInputFormat: Total input files to
process : 1
2022-08-30 12:54:11,734 INFO mapreduce.JobSubmitter: number of splits:9
2022-08-30 12:54:11,954 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0005
2022-08-30 12:54:11,956 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 12:54:12,137 INFO conf.Configuration: resource-types.xml not found
2022-08-30 12:54:12,137 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2022-08-30 12:54:12,206 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0005
2022-08-30 12:54:12,248 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0005/
2022-08-30 12:54:12,250 INFO mapreduce.Job: Running job: job_1661861832644_0005
2022-08-30 12:54:21,360 INFO mapreduce.Job: Job job_1661861832644_0005 running
in uber mode : false
2022-08-30 12:54:21,361 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 12:54:30,478 INFO mapreduce.Job:  map 33% reduce 0%
2022-08-30 12:54:37,568 INFO mapreduce.Job:  map 56% reduce 0%
2022-08-30 12:54:38,573 INFO mapreduce.Job:  map 67% reduce 0%
2022-08-30 12:54:45,631 INFO mapreduce.Job:  map 78% reduce 0%
2022-08-30 12:54:46,636 INFO mapreduce.Job:  map 100% reduce 0%
2022-08-30 12:54:54,687 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 12:54:56,699 INFO mapreduce.Job:  map 100% reduce 67%
2022-08-30 12:54:57,704 INFO mapreduce.Job:  map 100% reduce 100%
2022-08-30 12:54:58,716 INFO mapreduce.Job: Job job_1661861832644_0005 completed
successfully
2022-08-30 12:54:58,803 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=276444
                FILE: Number of bytes written=3555543
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=59
                HDFS: Number of read operations=42
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed reduce tasks=1
                Launched map tasks=9
                Launched reduce tasks=3
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=196139088
                Total time spent by all reduces in occupied slots (ms)=55649748
                Total time spent by all map tasks (ms)=62148
                Total time spent by all reduce tasks (ms)=17633
                Total vcore-milliseconds taken by all map tasks=62148
                Total vcore-milliseconds taken by all reduce tasks=17633
                Total megabyte-milliseconds taken by all map tasks=196139088
                Total megabyte-milliseconds taken by all reduce tasks=55649748
        Map-Reduce Framework
                Map input records=3761
                Map output records=30564
                Map output bytes=215298
                Map output materialized bytes=276588
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=3006
                Reduce shuffle bytes=276588
                Reduce input records=30564
                Reduce output records=3
                Spilled Records=61128
                Shuffled Maps =27
                Failed Shuffles=0
                Merged Map outputs=27
                GC time elapsed (ms)=1755
                CPU time spent (ms)=21630
                Physical memory (bytes) snapshot=6360526848
                Virtual memory (bytes) snapshot=53441662976
                Total committed heap usage (bytes)=5123866624
                Peak Map Physical memory (bytes)=627888128
                Peak Map Virtual memory (bytes)=4516925440
                Peak Reduce Physical memory (bytes)=326762496
                Peak Reduce Virtual memory (bytes)=4473929728
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=59
2022-08-30 12:54:58,803 INFO streaming.StreamJob: Output directory:
/user/root/demo2/vocabsize-output
```

[45]:
```
# parts b/c - take a look at the output directory in HDFS (RUN THIS CELL AS IS)
!hdfs dfs -ls {HDFS_DIR}/vocabsize-output/
```

```
Found 4 items
-rw-r--r--   1 root hadoop          0 2022-08-30 12:54
/user/root/demo2/vocabsize-output/_SUCCESS
-rw-r--r--   1 root hadoop         20 2022-08-30 12:54
/user/root/demo2/vocabsize-output/part-00000
-rw-r--r--   1 root hadoop         20 2022-08-30 12:54
/user/root/demo2/vocabsize-output/part-00001
-rw-r--r--   1 root hadoop         19 2022-08-30 12:54
/user/root/demo2/vocabsize-output/part-00002
```

[46]:
```
# parts b/c - view results (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/vocabsize-output/*
```

```
NumUniqueWords  1023
NumUniqueWords  1020
NumUniqueWords  963
```

> **DISCUSSION QUESTIONS (after breakout 3)** * How many unique words were there? * What happened when 3 reduce tasks were specified?

6.0.3 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (after breakout 3)

- How many unique words were there? > There are 3006 unique words in this text. (NOTE: the count of unique words is dependent on the tokenizer, this count assumes the provided tokenizer was used.)
- What happened when 3 reduce tasks were specified?
  > Specifying three reduce tasks causes our result file to include 3 sub-counts instead of one total count. We'd need to add these three numbers to get the true total.

7 Breakout 4: Secondary Sort

In breakout 1 we talked a little bit about the default sorting that Hadoop observes. However we'll often want to sort not just by the key but also by value. For example, we might want to sort the words by their count to find the most frequent words but then break ties by the word in alphabetical order. This is called a 'secondary sort'. In this question we'll learn about specifying parameters for sorting in Hadoop jobs. In particular you'll add three new parameters to your Hadoop Streaming command:

`-D stream.num.map.output.key.fields=2` : tells Hadoop to treat both the first and the second (tab separated) fields as a composite key.

`-D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedCompara`
: tells Hadoop that we want to make comparisons (for sorting) based on the fields in this composite key

`-D mapreduce.partition.keycomparator.options="-k2,2nr -k1,1"`: Tells Hadoop to perform a reverse numerical sort on the second field in the composite key and then break ties by sorting (alphabetically) on the first field in the composite key.

To find the top words in the *Alice in Wonderland* text we'll use the output of our word counting job as the input for this new sorting task. Recall that this output is a file in alphabetical order whose lines are of the format `word \t count`. Also recall that this file is already available in HDFS at the path `{HDFS_DIR}/wordcount-output`. You can simply pass this directory path in to the Hadoop streaming input parameter and it will understand that it should read in the directory's contents. `IMPORTANT:` *please use a single reduce task for parts a and b.*

Documentation: * https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html#Streaming_Command_Options
* https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html#Generic_Command_Options

> **DISCUSSION QUESTIONS (before breakout 4):**
> * Before we get to the full secondary sort it's worth noting that there is a really easy way to get Hadoop to sort our file by count: we could just switch the order of the count and the word when we print to standard output in our mapper. Why does this work?
> * In the Hadoop job below we're using `/bin/cat` as our reducer… what is that?

7.0.1 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (before breakout 4) * Why would switching the order of the count & word achieve a secondary sort? > This would work because Hadoop sorts on the key by default, so if the key is the count then we should get a list of words by frequency.

- In the Hadoop job below we're using `/bin/cat` as our reducer… what is that? > remember that we can use any executable as a mapper or reducer. This is the bash command (binary excecutable) that just reads/passses whatever we give it.

7.0.2 Breakout 4 Tasks:

- **a) code + discussion:** Complete the code in `TopWords/mapper.py` so that it performs the switch described above. For debugging purposes we'll first run this job using a test file of word counts instead of the full *Alice* file. Run the provided Hadoop streaming command to confirm that our sneaky solution works.

  - Notice that there is a problem with this result. Briefly discuss in groups what the problem is and why it happens.

- **b) code:** Ok, for obvious reasons our 'sneaky' solution didn't quite give us the output we wanted. So let's do a secondary sort properly this time. To do this, add the three new Hadoop options described in the intro to this question. Run your job with these new specifications on the dummy count file. When you are satisfied that your job works, change the input path to specify the alice count output that is already in HDFS. Your list of top words should match the result you got in HW1.
  - **`Two important warnings:`** 1) Parameters starting with the `-D` flag must come immediately after the line where you specify the jar file and before the parameters **`-files`**, **`-mapper`**, etc; 2) The options we provided you above specify a reverse numeric sort on the second field and tie breaking using the first field... but the mapper you wrote in part a switched the order of the words and the counts. You will need to make a small adjustment to the options we provided so that it instead reverse numerically sorts by the first field and breaks ties on the second.
- **c) code + discussion:** Run your Hadoop job one more time but this time add the parameter to specify that the job should use 2 reduce tasks instead of 1. For convenience of illustration, you should do this using the sample counts file instead of the full *Alice* text.
  - Something is wrong with the results. Use the provided code to look at the output of each partition independently. Discuss why our results are off.

  - Imagine we had a really large file and performed a sort using 100 partitions, what post processing would we have to do to get a fully ordered list (Total Order Sort)? Compare the computational cost of this postprocessing to the postprocessing we discussed in the VocabSize job.

`NOTE:` The cell below will create a short file of word counts that we can load into HDFS and use to test our Hadoop MapReduce job. Take a moment to read this sample file and figure out what a reverse numerical sort (with alphabetical tie breaking) should yield. Then go on to complete your tasks as described above.

[59]:
```
%%writefile TopWords/sample.txt
foo         5
quux         9
labs         100
bar         5
qi         1
```

```
Overwriting TopWords/sample.txt
```

[60]:
```
# load sample file into HDFS (RUN THIS CELL AS IS)
!hdfs dfs -copyFromLocal TopWords/sample.txt {HDFS_DIR}
```

```
copyFromLocal: `/user/root/demo2/sample.txt': File exists
```

[61]:
```
# part a - complete your work above then RUN THIS CELL AS IS
!chmod a+x TopWords/mapper.py
```

[62]:
```
# parts a/b/c - clear output directory (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/topwords-output
```

```
Deleted /user/root/demo2/topwords-output
```

[63]:
```
# part a/b/c - Hadoop streaming command
!hadoop jar {JAR_FILE} \
  -files TopWords/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \
  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/topwords-output \
  -numReduceTasks 1 \
  -cmdenv PATH={PATH}
```

```
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob7658509421783337419.jar tmpDir=null
2022-08-30 13:08:47,286 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 13:08:47,495 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 13:08:47,947 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 13:08:47,948 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 13:08:48,160 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0007
2022-08-30 13:08:48,511 INFO mapred.FileInputFormat: Total input files to
process : 1
^C
2022-08-30 13:08:48,957 INFO mapreduce.JobSubmitter: Cleaning up the staging
area /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0007
2022-08-30 13:08:48,958 ERROR streaming.StreamJob: Error Launching job :
Filesystem closed
Streaming Command Failed!
```

[64]:
```
# <--- SOLUTION --->
# part a/b/c - Hadoop streaming command
!hdfs dfs -rm -r {HDFS_DIR}/topwords-output
!hadoop jar {JAR_FILE} \
  -D stream.num.map.output.key.fields=2 \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.
↪KeyFieldBasedComparator \
  -D mapreduce.partition.keycomparator.options="-k1,1nr -k2,2" \
  -files TopWords/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \
  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/topwords-output \
  -numReduceTasks 3 \
  -cmdenv PATH={PATH}
```

```
rm: `/user/root/demo2/topwords-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar]
/tmp/streamjob2326149892501135834.jar tmpDir=null
2022-08-30 13:08:56,315 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 13:08:56,556 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 13:08:57,009 INFO client.RMProxy: Connecting to ResourceManager at
w261-m/10.128.0.8:8032
2022-08-30 13:08:57,009 INFO client.AHSProxy: Connecting to Application History
server at w261-m/10.128.0.8:10200
2022-08-30 13:08:57,214 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1661861832644_0008
2022-08-30 13:08:57,520 INFO mapred.FileInputFormat: Total input files to
process : 1
2022-08-30 13:08:57,608 INFO mapreduce.JobSubmitter: number of splits:11
2022-08-30 13:08:57,810 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1661861832644_0008
2022-08-30 13:08:57,812 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-08-30 13:08:57,976 INFO conf.Configuration: resource-types.xml not found
2022-08-30 13:08:57,976 INFO resource.ResourceUtils: Unable to find 'resource-
types.xml'.
2022-08-30 13:08:58,039 INFO impl.YarnClientImpl: Submitted application
application_1661861832644_0008
2022-08-30 13:08:58,086 INFO mapreduce.Job: The url to track the job:
http://w261-m:8088/proxy/application_1661861832644_0008/
2022-08-30 13:08:58,088 INFO mapreduce.Job: Running job: job_1661861832644_0008
2022-08-30 13:09:06,189 INFO mapreduce.Job: Job job_1661861832644_0008 running
in uber mode : false
2022-08-30 13:09:06,190 INFO mapreduce.Job:  map 0% reduce 0%
2022-08-30 13:09:14,302 INFO mapreduce.Job:  map 9% reduce 0%
2022-08-30 13:09:15,317 INFO mapreduce.Job:  map 27% reduce 0%
2022-08-30 13:09:20,365 INFO mapreduce.Job:  map 36% reduce 0%
2022-08-30 13:09:22,385 INFO mapreduce.Job:  map 55% reduce 0%
2022-08-30 13:09:26,429 INFO mapreduce.Job:  map 64% reduce 0%
2022-08-30 13:09:29,452 INFO mapreduce.Job:  map 73% reduce 0%
2022-08-30 13:09:30,458 INFO mapreduce.Job:  map 82% reduce 0%
2022-08-30 13:09:32,471 INFO mapreduce.Job:  map 91% reduce 0%
2022-08-30 13:09:34,484 INFO mapreduce.Job:  map 100% reduce 0%
```

```
2022-08-30 13:09:40,529 INFO mapreduce.Job:  map 100% reduce 33%
2022-08-30 13:09:42,541 INFO mapreduce.Job:  map 100% reduce 67%
2022-08-30 13:09:43,545 INFO mapreduce.Job:  map 100% reduce 100%
2022-08-30 13:09:44,556 INFO mapreduce.Job: Job job_1661861832644_0008 completed
successfully
2022-08-30 13:09:44,640 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=66
                FILE: Number of bytes written=3506780
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1210
                HDFS: Number of bytes written=38
                HDFS: Number of read operations=48
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed reduce tasks=1
                Launched map tasks=11
                Launched reduce tasks=3
                Data-local map tasks=11
                Total time spent by all maps in occupied slots (ms)=198632328
                Total time spent by all reduces in occupied slots (ms)=44155596
                Total time spent by all map tasks (ms)=62938
                Total time spent by all reduce tasks (ms)=13991
                Total vcore-milliseconds taken by all map tasks=62938
                Total vcore-milliseconds taken by all reduce tasks=13991
                Total megabyte-milliseconds taken by all map tasks=198632328
                Total megabyte-milliseconds taken by all reduce tasks=44155596
        Map-Reduce Framework
                Map input records=5
                Map output records=5
                Map output bytes=38
                Map output materialized bytes=246
                Input split bytes=1012
                Combine input records=0
                Combine output records=0
                Reduce input groups=5
                Reduce shuffle bytes=246
                Reduce input records=5
                Reduce output records=5
                Spilled Records=10
                Shuffled Maps =33
                Failed Shuffles=0
                Merged Map outputs=33
                GC time elapsed (ms)=1906
                CPU time spent (ms)=16850
                Physical memory (bytes) snapshot=7361687552
                Virtual memory (bytes) snapshot=62238027776
                Total committed heap usage (bytes)=6236405760
                Peak Map Physical memory (bytes)=614694912
                Peak Map Virtual memory (bytes)=4466851840
                Peak Reduce Physical memory (bytes)=314245120
                Peak Reduce Virtual memory (bytes)=4452638720
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=198
        File Output Format Counters
                Bytes Written=38
2022-08-30 13:09:44,640 INFO streaming.StreamJob: Output directory:
/user/root/demo2/topwords-output
```

[65]:
```
# part a/b/c - Save results locally (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/topwords-output/part-0000* > TopWords/results.txt
```

[66]:
```
# part a/b/c - view results (RUN THIS CELL AS IS)
!head TopWords/results.txt
```

```
5       bar
5       foo
100     labs
9       quux
1       qi
```

[67]:
```
# part c - look at first partition (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/topwords-output/part-00000
```

```
5       bar
5       foo
```

[68]:
```
# part c - look at second partition (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/topwords-output/part-00001
```

```
100     labs
9       quux
1       qi
```

Expected Results:

part a

part b

part c

> **DISCUSSION QUESTIONS**: * What was the problem with the results in part a? * How do you know the secondary sort worked in part b? * What was the problem with the results in part c? * How could we post process these files (part c partitions) to produce a total order sort of them? Why would this be computationally expensive?

7.0.3 <— SOLUTION —>

INSTRUCTOR TALKING POINTS (after breakout 4)

- What was the problem with the results in part a? > There is a big and a little problem here. The little problem is that we've sorted from smallest to largest when we'd probably have the opposite order if we're looking for a list of most frequent words. The bigger problem is that 'labs 100' isn't ordered correctly. It should be last but instead its second. This occurs because Hadoop is treating the counts as text values not as a numbers so they're sorted alphabetically not by value.
- How do you know the secondary sort worked in part b? > there are two records with the same count (5) and the 'bar' comes before the 'foo'
- What was wrong with the results in part c? > The results are not sorted properly because when we specify 2 reducers we end up with two sorted files concatenated together.
- How could we post-process these files to produce a total order sort of them? Why would this be computationally expensive? > To get a single list of words by frequency we'd need to merge sort these files (partitions) together. This is a much more computationally challenging task than simply adding the result of each mapper. In particular if we had lots of partitions (eg. 100) this mergesort would be O(100∗|V|).

8 Breakout 5: Tracking Down Errors in Python Code

You've now seen most of the basic functionality of writing and running Hadoop streaming jobs. In this week's homework and over the course of the next few weeks we'll explore additional options and tricks to add to our jobs. As we do this you will want to be able to quickly distinguish between errors that occur due to a mistake in your algorithm design or Hadoop streaming command and errors that are rooted in your Python code. Unfortunately the error logs printed to console do not always make this distinction obvious. Luckily, the Hadoop UI logs do make it very easy to identify Python coding errors. Before you move on to the homework (even if there isn't time in class) we'd like to make sure you know where to find these logs and how to fix two common mistakes. **Below, we provided code that contains two common errors for you to debug. Your job is to:**
1. **Run the provided code as is, it will throw an error.**

2. **Navigate to the Hadoop UI and find the relevant logs explaining your error.**

- Under `Task Type`, click `Map` > `task_XXXXXX` >`logs`

3. **Fix the error(s) and re-run the job**.

NOTE 1: There are two different kinds of errors in the mapper code. See the inline comments for specific fixes: one involves adding a parameter to your Hadoop job the other two you must fix in the mapper code (re-run that cell to overwrite the old mapper). I'd recommend fixing them one at a time so that you can see how the logs and error messages change depending on the type of error.

NOTE 2: If you do not get to this section in class, you still must complete it before beginning the homework!

[74]:
```
%%writefile TopWords/sample.txt
foo        5
quux        9
labs        100
bar        5
qi        1
```

```
Overwriting TopWords/sample.txt
```

[75]:
```
# run the following cells to create the demo mapper
!mkdir demo
```

```
mkdir: cannot create directory `demo': File exists
```

The next cell uses a little Jupyter Magic to create a python script on the fly, this is a useful technique that you may want to use in the future to create files for unit testing, etc.

[76]:
```
%%writefile demo/mapper.py
#!/usr/bin/env python
"""
This is a silly mapper to demonstrate some errors.
"""
import sys
import numpy as np  # To use numpy add -cmdenv PATH={PATH} to your Hadoop Job

for line in sys.stdin:
    msg = ("a message"   # missing a parenthesis here
    print(1/0)            # dividing by zero is a no-go
```

```
Overwriting demo/mapper.py
```

[77]:
```
# clear HDFS output directory when you re-run the job
!hdfs dfs -rm -r {HDFS_DIR}/demo-output
```

```
rm: `/user/root/demo2/demo-output': No such file or directory
```

[78]:
```
# Hadoop streaming command
!hdfs dfs -rm -r {HDFS_DIR}/demo-output
!hadoop jar {JAR_FILE} \
  -files demo/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \
  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/demo-output \
  -cmdenv PATH={PATH}
```

```
rm: `/user/root/demo2/demo-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-
streaming-2.6.0-cdh5.16.2.jar] /tmp/streamjob9580416457511258.jar tmpDir=null
21/01/16 20:53:55 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
21/01/16 20:53:56 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
21/01/16 20:53:56 INFO mapred.FileInputFormat: Total input paths to process : 1
21/01/16 20:53:56 INFO mapreduce.JobSubmitter: number of splits:2
21/01/16 20:53:56 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1610822007109_0010
21/01/16 20:53:56 INFO impl.YarnClientImpl: Submitted application
application_1610822007109_0010
21/01/16 20:53:57 INFO mapreduce.Job: The url to track the job:
http://docker.w261:8088/proxy/application_1610822007109_0010/
21/01/16 20:53:57 INFO mapreduce.Job: Running job: job_1610822007109_0010
21/01/16 20:54:03 INFO mapreduce.Job: Job job_1610822007109_0010 running in uber
mode : false
21/01/16 20:54:03 INFO mapreduce.Job:  map 0% reduce 0%
21/01/16 20:54:07 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000000_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

21/01/16 20:54:08 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000001_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
```

```
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

21/01/16 20:54:11 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000000_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

21/01/16 20:54:12 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000001_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

21/01/16 20:54:16 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000000_2, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

21/01/16 20:54:17 INFO mapreduce.Job: Task Id :
attempt_1610822007109_0010_m_000001_2, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess
failed with code 1
        at
org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:325)
        at
org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:538)
        at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
        at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
        at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
        at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:459)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInforma
tion.java:1924)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
```

```
21/01/16 20:54:21 INFO mapreduce.Job:  map 100% reduce 100%
21/01/16 20:54:21 INFO mapreduce.Job: Job job_1610822007109_0010 failed with
state FAILED due to: Task failed task_1610822007109_0010_m_000000
Job failed as tasks failed. failedMaps:1 failedReduces:0

21/01/16 20:54:21 INFO mapreduce.Job: Counters: 14
        Job Counters
                Failed map tasks=7
                Killed map tasks=1
                Killed reduce tasks=1
                Launched map tasks=8
                Other local map tasks=6
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=21997
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=21997
                Total vcore-milliseconds taken by all map tasks=21997
                Total megabyte-milliseconds taken by all map tasks=22524928
        Map-Reduce Framework
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
21/01/16 20:54:21 ERROR streaming.StreamJob: Job not successful!
Streaming Command Failed!
```

[79]:
```
# <--- SOLUTION --->
# Hadoop streaming command
!hadoop jar {JAR_FILE} \
  -files demo/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \
  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/demo-output \
  -cmdenv PATH={PATH}
```

```
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-
streaming-2.6.0-cdh5.16.2.jar] /tmp/streamjob2110567459188237115.jar tmpDir=null
21/01/16 20:54:23 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
21/01/16 20:54:23 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
21/01/16 20:54:23 WARN security.UserGroupInformation: PriviledgedActionException
as:root (auth:SIMPLE) cause:org.apache.hadoop.mapred.FileAlreadyExistsException:
Output directory hdfs://quickstart.cloudera:8020/user/root/demo2/demo-output
already exists
21/01/16 20:54:23 WARN security.UserGroupInformation: PriviledgedActionException
as:root (auth:SIMPLE) cause:org.apache.hadoop.mapred.FileAlreadyExistsException:
```

```
Output directory hdfs://quickstart.cloudera:8020/user/root/demo2/demo-output
already exists
21/01/16 20:54:23 ERROR streaming.StreamJob: Error Launching job : Output
directory hdfs://quickstart.cloudera:8020/user/root/demo2/demo-output already
exists
Streaming Command Failed!
```

8.0.1 Follow-Up:

- The below images should be similar to what was found during the above exercise. Before beginning homework 2, make sure you can find these errors through the Hadoop UI.
  - If you are unable to find these errors in the Hadoop UI logs, **seek the help of an instructor or TA immediately**.

[ ]:

```
```