CS61C Summer 2015 Lab 13: MapReduce II

Goals

- Fit a real-world problem to the MapReduce paradigm.
- Run MapReduce on Amazon EC2

Setup

You should complete this lab on the hive machines in 330 Soda, which have the relevant tools and scripts for starting up a Spark cluster on EC2. If you are not sitting physically in front of a lab machine, you can access one (list of machine names) remotely by following these instructions. These directions rely on commands on the instructional machines. **You won't be able to do this from your desktop**. As a result, you'll use your course account to complete this lab.

Copy the contents of `~cs61c/labs/13` to a new lab13 directory in your home directory:

```
[ssh'd onto a hive machine]
$ mkdir lab13
$ cd lab13
$ cp -r ~cs61c/labs/13/* .
```

It will be especially helpful if inexperienced Python programmers partner with experienced Python programmers for this lab.

MapReduce is primarily designed to be used on large distributed clusters. And now, we're going to have you run on mid-size clusters on EC2. EC2, as was mentioned in lecture, is an Amazon service that lets you rent machines by the hour. CS61C has a grant from Amazon that allows you to use EC2 at no cost to you!

Background Information

In our last lab, we tried out some intro-level MapReduce problems. In this lab, we'll tackle a real-world problem and then run it on a real-world-sized dataset on EC2.

The MapReduce programming framework is primarily designed to be used on large distributed clusters. However, large, distributed jobs are harder to debug. For this lab, we'll be using Spark, an open source platform (developed at Berkeley!) which implements the MapReduce paradigm (among other things). First, we'll use Spark in "local mode", where your Map and Reduce routines are run entirely within one process. Once your program works, we'll try it out on a larger dataset on Amazon EC2!

Avoid Global Variables

One of the core tenets of MapReduce is that we want to avoid multiple machines working on a single, unpartitioned data set because of the associated overhead. As a result, your algorithms will very rarely need to use global variables. In the worst case, you may need to share one or two variables for configuration across machines. If this is necessary, we will indicate it to you specifically in the spec.

Useful classes/python tips (specifically for implementing `mutualfriends.py`)

- sets - Good for eliminating duplicates from a sequence and performing intersections (for example, of two friends lists).
- filter - You'll probably want to use this to non-destructively remove elements from a list.

More Useful classes, Hadoop Documentation, and Additional Resources

- The Python API documentation is on the web: https://docs.python.org/2/library/.
- The Spark Programming Guide is also useful: http://spark.apache.org/docs/latest/programming-guide.html. **Make sure you switch to the python tabs, NOT Scala or Java.** You may find the Transformations section particularly useful for understanding the skeleton.

Exercise Background

The following exercises use two different sample input files that can be found in `~cs61c/data/mrlab2` or on Amazon S3 (the filestore for EC2):

1. `small.seq` -- a very small social network, for debugging
2. `large.seq` -- a large social network, for testing out EC2 (only available on S3)

Notice the `.seq` extension, which signifies a Hadoop sequence file. These are NOT human-readable. For debugging purposes, ~cs61c/data/mrlab2/small.txt is provided (and is also shown as an example below).

We recommend deleting output directories when you have completed the lab, so you don't run out of your 500MB of disk quota. You can do this by running:

```
$ make clean
```

Please be careful with this command as it will delete all MapReduce outputs generated in this lab.

Exercise 1: Generating a Mutual Friends List on Social Network Data

For this exercise, you will fill in `mutualfriends.py`. We will be working locally (**NOT** on EC2) for this section, since debugging on a smaller dataset is much easier.

Suppose as an intern at a new social networking company, you're given the task of implementing the daily Mutual Friends List computation using MapReduce (a mutual friend of a person A and a person B is a person C that both A and B consider a friend). Unfortunately, the data has not been given to you in a very clean format. You're given key-value pairs of the form `(person_1, person_2)`, where such a pair indicates a friendship between `person_1` and `person_2`. Unfortunately, the data is messy, so our input may or may not have the corresponding key-value pair `(person_2, person_1)` (as you may have noticed, social networks using the concept of "friendship" are undirected graphs).

It is your job as the intern to process this data and output a list of mutual friendships for every pair of friends in the dataset. For the input:

```
1 2
2 3
1 4
1 3
2 1
```

Your output should look like:

```
(1, 2)
[3]
(1, 3)
[2]
(2, 3)
[1]
(1, 4)
[]
```

This indicates that:
3 is a mutual friend of 1 and 2,
2 is a mutual friend of 1 and 3,
1 is a mutual friend of 2 and 3,
1 and 4 have no mutual friends.

In order to maintain user privacy, all usernames have been encoded as `int`s, so in the above input, 1 is friends with 2, 2 is friends with 3, 1 is friends with 4, and 3 is friends with 1 (notice how order does not matter and we potentially have duplicates, like the

first and last input pairs in this example).

Here is our general plan of attack (also outlined in the mutualfriends function, which you should not have to modify):

1. Apply the `flatMap` transformation, with the function `flatmapper1` (which you will write).
2. Apply the `reduceByKey` transformation, with the function `reduce_to_friends_list` (which you will write).
3. Apply the `flatMap` transformation, with the function `flatmapper2` (which you will write).
4. Apply the `groupByKey` transformation, which performs the shuffle. This essentially combines all of the values together for a particular key, so that you can use one more `flatMap` in the next stage. (Think about how it would be cumbersome to use a `reduceByKey` here). Note that there is no function passed into `groupByKey`, so there is nothing for you to write in this step.
5. Apply the `flatMap` transformation, with the function `flatmapper3` (this is already written for you).

Fill in the indicated areas in `mutualfriends.py`. The skeleton is heavily commented and a thorough read through it will be essential to completing the lab. Once you've completed your implementation, test it out on the small input graph and confirm that we are in fact getting the correct output (the output shown above).

To run your code on the small dataset, do the following:

```
$ make run-local
```

The output file `part-00000` will be located in the `spark-mutualfriends-out` directory. Confirm that you are getting the output shown above before moving on.

Checkoff:

- Run MutualFriends locally on small.seq and show your TA the output.

Exercise 2 Setup

First, you'll need to run this command to configure your class account to work with ec2.

```
[assuming you are inside your lab13 directory on a hive machine]
$ cd ec2-scripts
$ make account
```

You should only need to run these commands once. It's okay if you see some failure messages (HTTP 404's and 403's), as long as the end of the output of the command looks like the following (note that the key at the end *will not* match exactly – but you should have a key after the countdown):

```
Checking for existing certs...
No existing certs found...
Adding new cert to IAM...
403 InvalidClientTokenId The security token included in the request is invalid
Done.
If you see a 403 error above this line, that's okay
If you see a 403 error below this line, that's a problem
Now pausing before running iam-useraddcert to allow the EC2
key management service to catch up, please wait
5
4
3
2
1
IV4HAPD6PBDAR5LEBHBIW6BMG2JHJGSH
```

In the unlikely event that you do not get this output, try running `make account` again. If another run also fails similarly, please talk to a TA.

Our Test Corpus and Accessing it With Hadoop

We have a larger dataset like the "small.seq" file you used to debug Exercise 1. We have stored this on Amazon's Simple Storage Service (S3), which provides storage that is located in the same datacenter as the EC2 virtual machines. You can access it from Spark jobs by specifying the "filename" `s3n://cs61cMR2` ("s3n" stands for S3 Native). The data is stored compressed, so you should expect output from a job run against it to be substantially bigger.

EC2 Billing

Amazon EC2 rents virtual machines by the hour, rounded up to the next hour. Amazon provides several price points of virtual machines. In this lab, you will use the "Large" cluster listed below. To save you time, we've also provided some information about the same mutualfriends code running on a "Small" cluster:

Cluster	Number of Machines	Node type	vCPUs per Node	RAM per Node	Cost per Node per Hour	Total vCPUs (excludes Master)	Total RAM (excludes Master)	Total cost to run cluster per hour	Time to run mutualfriends on large.seq
Small	4: 1 Master, 3 Workers	c3.xlarge	2	3.75 GiB	$0.12	6	11.25 GiB	$0.48	21.2 minutes
Large	6: 1 Master, 5 Workers	c3.8xlarge	32	60 GiB	$1.912	160	300 GiB	$11.472	???

Note that we are billed for all time when the machines are on, regardless of whether the machines are active, and we pay for at least one hour every time new machines are started. (So starting a machine for 5 minutes, terminating it, and starting an identical one for another 5 minutes causes us to be billed for *2 hours* of machine time.)

In addition to billing for virtual machine usage by the hour, Amazon also charges by usage for out-of-datacenter network bandwidth and long-term storage. Usually these costs are negligible compared to the virtual machine costs.

How we're able to use EC2 "For Free"

Amazon generously gives CS61C a grant every semester for a limited number of EC2 credits for our activities. Thus, you get to try out EC2 **at no cost to you!**

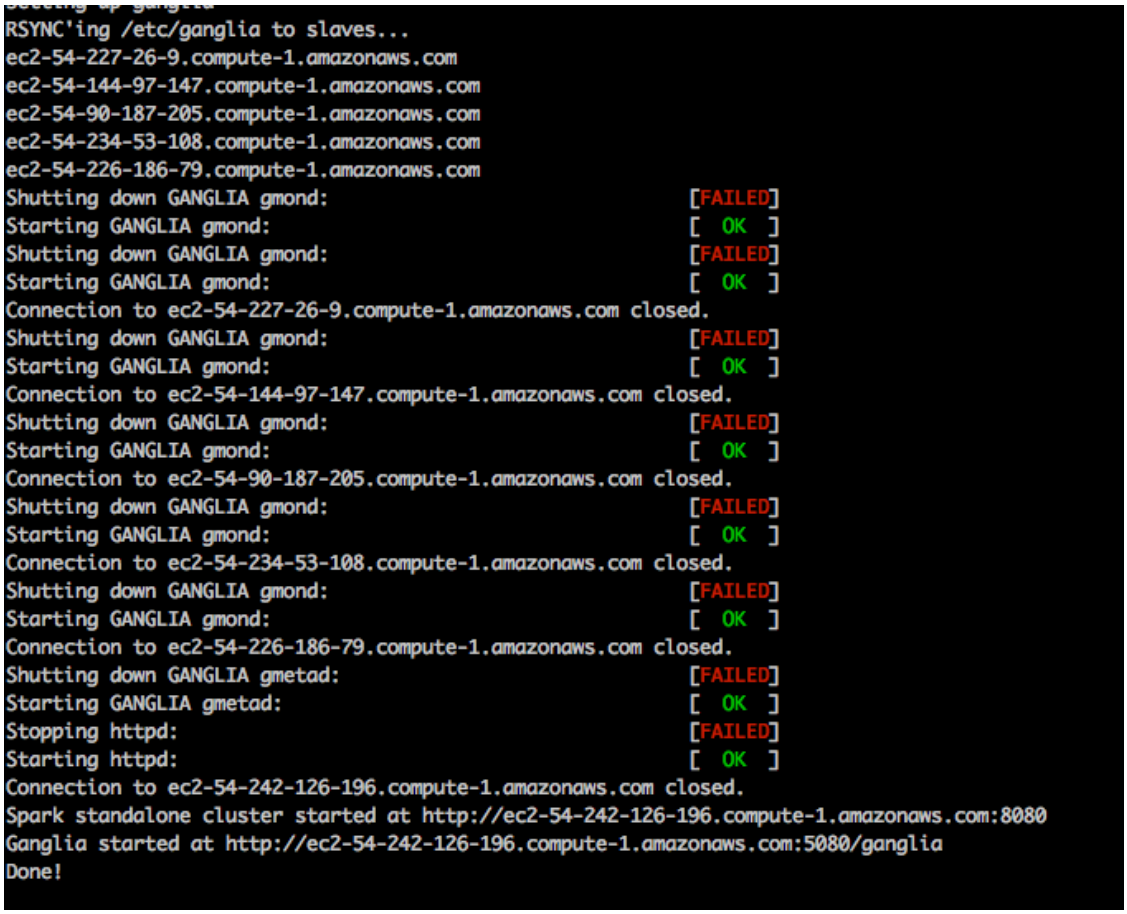
Exercise 2: Run MutualFriends on EC2 on a Large Cluster with a Large Dataset

Starting a Cluster

Go ahead and start a Hadoop cluster of 5 c3.8xlarge worker nodes and 1 c3.8xlarge master node by running the following command:

```
[from inside the lab13/ec2-scripts directory]
$ make launch-big
```

This command may take a couple minutes to complete. You may see some connection refused messages while the cluster is starting up, which you can safely ignore. You should see something similar to the following output when this command completes (note that the GANGLIA failures are expected):

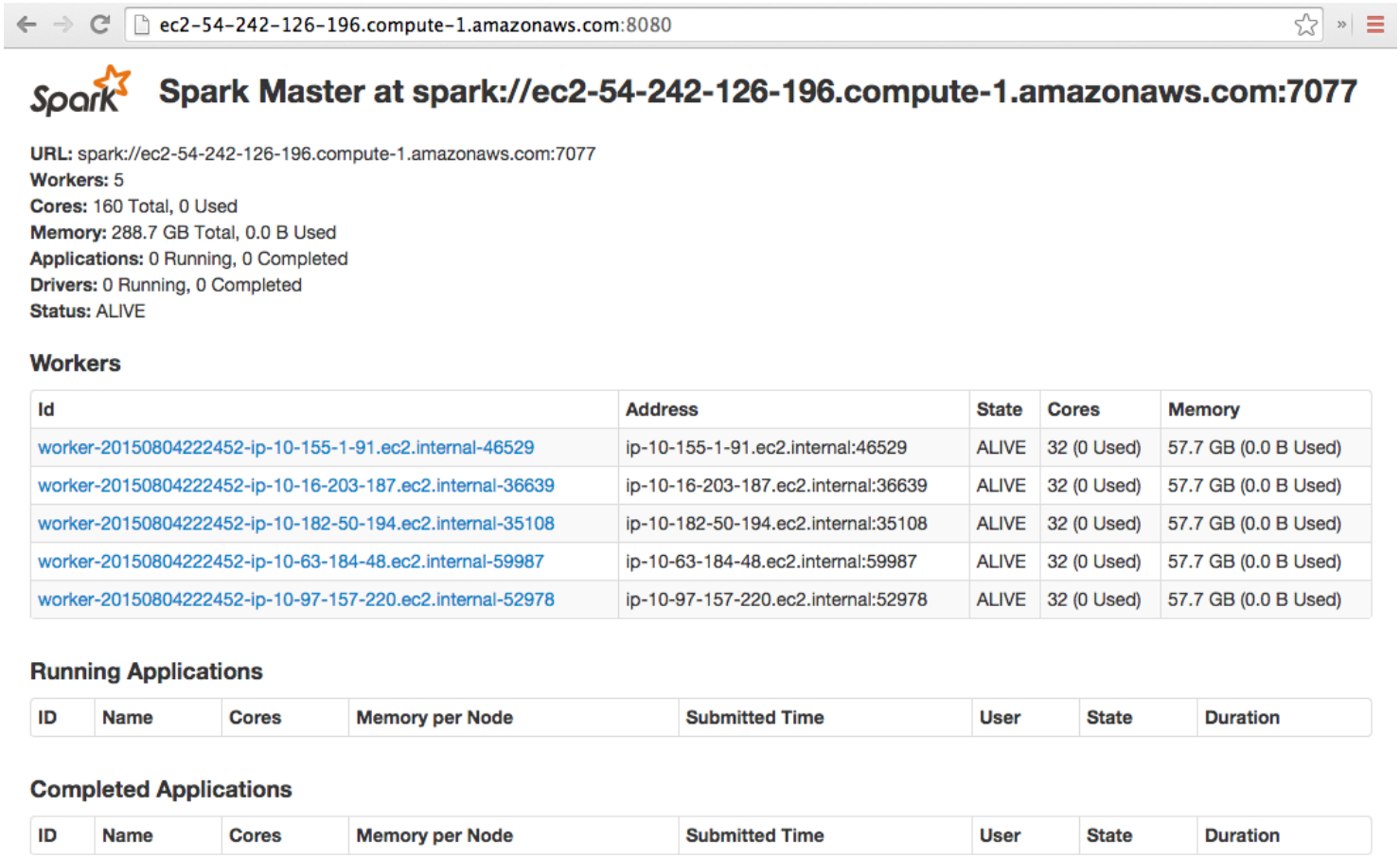


If you do not get output like this when the command has completed, try running `make resume`. If this also does not produce matching output, talk to a TA.

At this point, we are "on the clock" and being charged for the running cluster. You should not leave a cluster running for more than an hour. Once the command completes, we should be able to access a jobtracker webpage for our cluster. To get the url, run:

```
$ make master
```

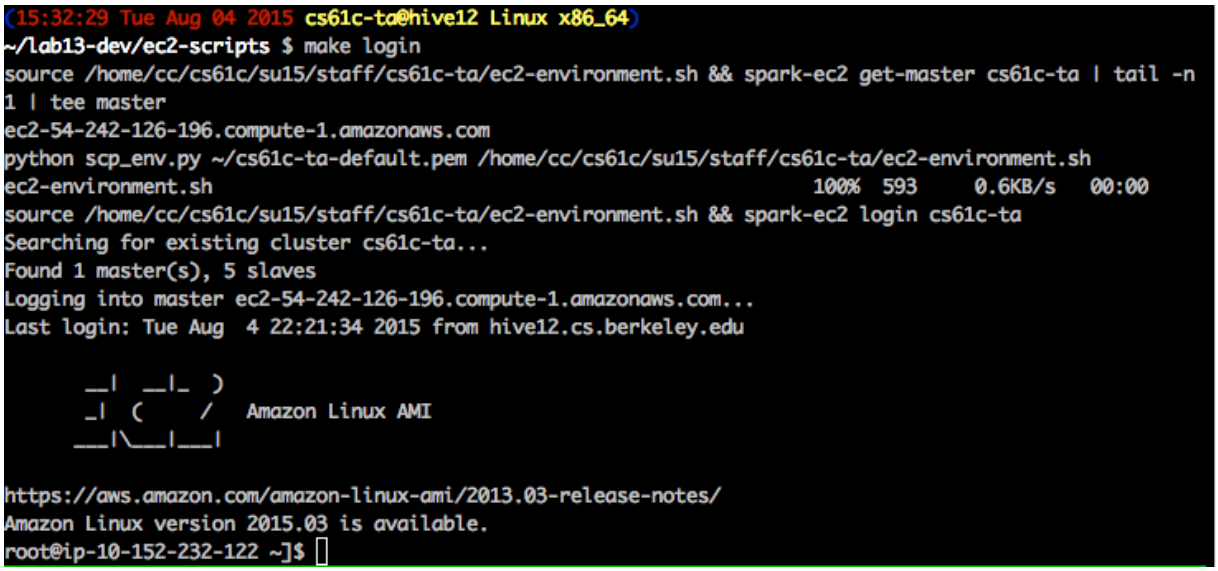
Take the output from this (something that looks like `ec2-54-242-126-196.compute-1.amazonaws.com`), add on ":8080" to the end, and paste it into your browser. You should be taken to a page that looks like this:



During this process, we have done all of the provisioning automatically so that your cluster is capable of running Spark programs. Now, we can transfer our code to the cluster and then login to the cluster over ssh:

```
[from inside the lab13/ec2-scripts directory]
make copy-code
make login
```

After doing this, we will be `ssh'd` into the master node of our ec2 cluster. It should look like below:



Now, if we run `ls`, we get the following output.

```
root@ip-10-234-72-164 ~]$ ls
ec2-environment.sh  ephemeral-hdfs  hadoop-native  lab13  mapreduce  persistent-hdfs  scala  shark  spark  spark-ec2  tachyon
```

You will notice that in the home directory of our master node, we have the code we've written in the lab ready to go in the `lab13` directory. Do the following to run your code:

```
[On the EC2 Node that make login ssh'd you into]
cd lab13
make run-ec2-large
```

You may notice a stacktrace early-on about Hadoop NativeIO - you can safely ignore this. If you return to the webpage we opened earlier, you will be able to track the progress of this job. On this page, right-click on "largemfriends" under "Running Applications" and click open in new tab (the page we are opening now will disappear when the job completes, so we want to be able to easily get back to the other page). On the page in the resulting tab, you can track the progress of your individual Spark job on the cluster.

Running your code on EC2 should take no longer than half an hour. If you're at the 30-minute mark and your code is still running, you should screenshot the job's status webpage to show to your TA and then terminate the cluster anyway (see instructions below) and write down "30 minutes" for your job duration (you won't lose any credit for this). You ***should not*** run your code on EC2 multiple times, nor should you spin up more than one cluster.

When the job finishes, you'll have control returned back to you in the terminal in which you're ssh'd into the master node. To view information about the completed job, you can return to the first tab we opened in the browser and click the corresponding job name under "Completed Applications." Take note of the duration of the job for checkoff.

In theory we could also retrieve the outputs from the hadoop distributed filesystem (HDFS) attached to each node. However, the output in this case is prohibitively large, so we'll just trust that our code produced the right output (this is why we debugged on a much smaller input earlier in the lab).

MOST IMPORTANT STEP!!!!!!!!!! Shutdown your cluster!!!!!!!

Once you've noted the amount of time it took your code to process the large input dataset, it's time to shutdown the cluster. First, we will exit from the ec2 master node by typing `exit` and then we will shutdown our cluster with `make destroy`. Run the following to shutdown the cluster:

```
[assuming you are currently on the ec2 master node]
$ exit
```

```
[now, you should be back in the ec2-scripts directory on the hive machine that you're using. from there, run:]
$ make destroy
```

It should produce the following output:

```
source /home/cc/cs61c/su15/staff/cs61c-ta/ec2-environment.sh && spark-ec2 destroy cs61c-ta
Are you sure you want to destroy the cluster cs61c-ta?
The following instances will be terminated:
Searching for existing cluster cs61c-ta...
Found 1 master(s), 5 slaves
> ec2-54-242-126-196.compute-1.amazonaws.com
> ec2-54-227-26-9.compute-1.amazonaws.com
> ec2-54-144-97-147.compute-1.amazonaws.com
> ec2-54-90-187-205.compute-1.amazonaws.com
> ec2-54-234-53-108.compute-1.amazonaws.com
> ec2-54-226-186-79.compute-1.amazonaws.com
ALL DATA ON ALL NODES WILL BE LOST!!
Destroy cluster cs61c-ta (y/N): y
Terminating master...
Terminating slaves...
```

Make sure that you hit "y" when you're prompted. Once this has completed, run `make destroy` one more time to confirm that no more nodes are running:

```
$ make destroy
```

This time, you should get the following output:

```
source /home/cc/cs61c/su15/staff/cs61c-ta/ec2-environment.sh && spark-ec2 destroy cs61c-ta
Are you sure you want to destroy the cluster cs61c-ta?
The following instances will be terminated:
Searching for existing cluster cs61c-ta...
ALL DATA ON ALL NODES WILL BE LOST!!
Destroy cluster cs61c-ta (y/N): y
Terminating master...
Terminating slaves...
```

Notice now that the list of nodes is empty, so we're good to go.

Finally, answer the following questions and record your answers in a file:

a. How many times faster was the larger cluster than the smaller one? (see the above table for the job length on the small cluster)
b. What kind of scaling does this problem exhibit (strong or weak)?
c. How much money did you spend on EC2? (See the "EC2 Billing" section for rates).

Checkoff:

First, ensure that all clusters are terminated by running:

```
$ make destroy
```

Before you leave:

- Be sure to turn off your instances, via `make destroy`

Checkoff:

- Run MutualFriends locally on small.seq and show your TA the output (Exercise 1).
- Show your TA that you don't have any clusters running by running `make destroy`.
- Show your TA your file where you answered all the questions asked in this lab.