## Embarrassingly Parallel

By now, you should have all of the available log data from Sparkify uploaded to an S3 bucket. While this isn't yet on the order that we would call "big data", it is big enough to benefit from a little parallel processing. We will return to the same query from last week: what are the top 10 songs and top 10 artists in descending order? However, this time, instead of using a DBMS to compute our query, we will do it in pure Python.

1. Let's start by seeing how much data we've accumulated using [aws s3 ls](http://docs.aws.amazon.com/cli/latest/reference/s3/ls.html)i.e.

aws s3 ls --summarize --human-readable --recursive s3://bucket-name/directory

1. Now that we know how much data we're dealing with, we'll launch a new EC2 instance that we will call our analytics machine.
   1. First we need to pick our AMI. You may use Amazon Linux AMI or Ubuntu Server as you wish at this point. I generally prefer Amazon Linux for infrastructure and Ubuntu for OLAP given that Amazon Linux is reputed to be more stable while Ubuntu more user friendly. As a matter of course, however, both will work for either and the choice is ultimately yours. (There is also much to be said for picking one distribution and sticking with it.)
   2. Next we need to choose an instance type. Our data should be small enough to fit in memory (i.e. RAM) so long as we pick a large enough instance type. You will also want more than 2 vCPUs (for reasons that will become clear soon). c4.xlarge should suffice though it's worth noting that (at the time of this writing) we could conceivably go all the way up to 1.9 TB (with 128 vCPUs) before we have to either spill out onto disk or distribute the load onto multiple machines!
   3. **Optional:** We haven't talked about requesting spot instances yet, but if you feel like going a bit further into EC2 land, now would be a good time. Requesting a spot instance allows you to bid on an instance rather than pay full price. It can save a lot of money, but at the risk of being outbid and having your instance shut down. It is not a good choice for servers you want to keep running indefinitely like the ones we've been using. (Besides, those were on the free tier anyway, so it didn't matter.) For an analytics instance, however, we don't need it most of the time anyway, so it makes sense to request a spot instance. Take a look [Amazon EC2 Spot Instances Pricing](https://aws.amazon.com/ec2/spot/pricing/) to see how much your instance type is going for. (At the time of this writing, the spot instance charge for m4.large was only 1/8 the normal cost.)
   4. Don't forget to configure your security group. Since we will be running Jupyter Notebook on our instance, we probably want to add a custom TCP rule on port 8888 from anywhere. (If you forget to do this, you can always add this rule after you launch your instance.)
2. Install IPython parallel on your EC2 instance.  
   While I suggest you avoid using sudo on your laptop if you can avoid it, when setting up your EC2 instance to run Jupyter and IPython parallel, you should use sudo for just about everything. If you don't, it is possible one application won't be able to see the other.
   1. ssh into your new EC2 instance
   2. Since we're starting a new instance, we'll need to update (apt or yum) and install the usual packages. It's not easy to remember which packages to install using pip and which to install using our Linux package manager. Just as with conda, always start with the Linux package manager. If you can't find it, then try pip. And even if you can find it, you might want to update with pip after the fact.
   3. sudo pip install pyyaml ipython jupyter ipyparallel pandas boto -U to install (and/or upgrade) IPython, Jupyter, IPython parallel, etc.
   4. sudo ipcluster nbextension enable to enable the IPython cluster Jupyter notebook extension. This allows us to create a virtual cluster from within the Jupyter notebook.
   5. sudo jupyter notebook --port=8888 --no-browser --ip=0.0.0.0 to run Jupyter notebook. Notice:
      * We have set the --no-brower flag. As you've no doubt noticed when running Jupyter notebook on your laptop, Jupyter will automatically try to load the notebook in your default browser. We do not want to run a browser on our EC2 instance, however. Instead, we will be connecting to it from a browser we are running on our laptops.
      * We have set --ip=0.0.0.0. This means that Jupyter will accept connections from any IP address (not just localhost). Note that this is separate from (and in addition to) the security group we set on the AWS console. Both need to allow connections from anywhere (0.0.0.0) on port 8888.
      * One of the status messages that prints out when you run this says something like The Jupyter Notebook is running at: http://0.0.0.0:8888/?token=16e47a6fd1c42f2823ad344427ec7218d7f4211c400186d3 (only the token will no doubt be different). Make a note of this as you will need it in the next step.
3. Get started with ipyparallel
   1. In your favorite web browser, navigate to port 8888 on your EC2 instance using the token generated in the last step above (replacing 0.0.0.0 with the URL of your EC2 instance). For example, if the public URL of your EC2 instance is ec2-52-90-129-59.compute-1.amazonaws.com and the token generated by Jupyter is 16e47a6fd1c42f2823ad344427ec7218d7f4211c400186d3 then you would navigate to <http://ec2-52-90-129-59.compute-1.amazonaws.com:8888/tree?token=16e47a6fd1c42f2823ad344427ec7218d7f4211c400186d3>
   2. You should see the Jupyter home screen that by now you are so familiar with. At the top there should be three tabs: Files, Running, and IPython Clusters (or perhaps just Clusters). Click Clusters.
   3. You should see a row with a text box under the column "# of engines". Enter a number and click "Start" to start an IPython cluster. (If you don't see this, it means that ipcluster is either not running or Jupyter is not able to connect to it. Come see me.)
      * Picking the number of engines to run depends partially on the number of cores in your system (if you used a c4.xlarge, that would be 4) and partially on the nature of the application. If it is processor bound (such as when you are doing many mathematical operations like gradient descent), you will probably want your number of engines to equal n or \_n-\_1 where n is the number of cores. If it is I/O bound (such as when you are getting a lot of data from an external system like Twitter or S3) then you will want it to be some multiple of n or n- 1. In this case, I found having around 2-3x as many engines as cores to provide the best performance, but you should experiment with this number and ask yourself why that would be and under what circumstances would you want more or fewer engines?
   4. Return to the "Files" tab and create a new Python 3 notebook.
   5. Test it:
   6. from ipyparallel import Client
   7. rc = Client()

len(rc)

The output should be 3 (the number of engines in your cluster). If it's 0, wait a few seconds and try again. If it outputs anything else, it means it can't find your cluster.

1. Divide and Conquer
   1. Scatter the keys
      * Using [boto](http://boto.cloudhackers.com/en/latest/ref/s3.html#boto.s3.bucket.Bucket.get_all_keys) or [boto3](http://boto3.readthedocs.io/en/latest/reference/services/s3.html#S3.Bucket.objects), collect the keys to all the objects in your bucket of Sparkifylog events.
      * [scatter](https://ipyparallel.readthedocs.io/en/latest/details.html#scatter-and-gather) the keys to the different engines
      * On each engine, where a portion of the keys reside:
        1. [split](https://docs.python.org/2/library/string.html#string.split) the data into [json](https://docs.python.org/2/library/json.html) objects to load the event logs in as Python dictionaries.
        2. [get](https://docs.python.org/2/library/stdtypes.html#dict.get) the song names and artist names out of [entities](https://dev.twitter.com/overview/api/entities).
        3. Update a [counter](https://docs.python.org/2/library/collections.html#counter-objects) with each song name and artist name that you find
   2. Gather and reduce the result
      * [gather](https://ipyparallel.readthedocs.io/en/latest/details.html#scatter-and-gather) the resulting counters
      * add them together using the [reduce](https://docs.python.org/2/library/functions.html#reduce) function to get a final count of all the song names and artist names.
      * take the [most\_common](https://docs.python.org/2/library/collections.html#collections.Counter.most_common) song names and artist names and load them into a [DataFrame](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html)

**Submission:** Submit your solution as a Jupyter Notebook (.ipynb) file.