Distributed Computing

Signal Data Science

This lesson focuses on using distributed computing to parallelize algorithms for a significant reduction in the amount of required computatoin time. We will be using Amazon Web Services, although a variety of other cloud computing platforms are also available, most notably Google Cloud Platform.

Many technical terms in this assignment are linked to expository pages; if you are unfamiliar with a linked term, we strongly urge you to spend a minute or two reading the corresponding page. Taking the time to do so will substantially improve your understanding of GNU/Linux and relatedtopics.

Getting started with Amazon Web Services

- Register an account on the AWS website. You'll have to enter your billing info, but we'll be working with a cheap (under $0.02/hour) instance throughout this demonstration for which Amazon provides 31 free days of usage each month.
- At the top-right of AWS, make sure that the "US West (Oregon)" region is selected. You'll have the fastest connection to the N. California server, but speeds to Oregon are unnoticeably slower and AWS is slightly cheaper in that region due to high demand for N. California servers.
- Click on "Services" in the top left corner and go to "Console Home", where you'll see an overview of all the services which AWS offers. Click on the top option in the "Compute" section, labeled "EC2". This stands for "Elastic Compute Cloud", AWS's main service.[ˆamaz]
- Click on "Launch Instance" to make your first EC2 instance! Follow these steps to configure the instance:
  - *Step 1:* You can choose from a wide variety of preconfigured operating systems. Scroll through the list to get a sense of the options available to you, then scroll back to the top and select "Amazon Linux AMI", which comes configured with a wide variety of programming-related tools.

- *Step 2:* Select the `t2.micro` instance type, which should be selected by default and is labeled as "Free tier eligible".
- *Step 3:* Read the information tooltip (hover over the "i" icon) for each option. For the "Subnet" option, select `us-west-2a`.[1]
- *Step 4:* Briefly read about the different types of EBS storage available. (You can think of Amazon's EBS service as being analogous to a choice of hard drive.) Change the size of the "Root" volume, where your operating system files will reside, to "10 GiB", and select the General Purpose SSD (`gp2`) type.
- *Step 5:* If you have a lot of instances, you can add tags to them for organization. Skip this step.
- *Step 6:* Configuring the security group is important; otherwise, you won't be able to connect to your server properly. Select "Create a **new** security group" and give it any name or description you want. The default rule for "SSH" allows you to connect to your server via the SSH protocol from any IP address (because the "Source" is set to "Anywhere"). We would like to allow web browsers to connect to the server, so click "Add Rule", select "[HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol" for "Type", and select "Anywhere" for "Source".
- *Step 7:* Review the information and click "Launch". If you haven't used Amazon Web Services before, it will prompt you to create an authentication key. Do so and download the key file.

Congratulations: you've launched your first AWS server![2]

Amazon EC2 instances are billed on a *per-hour* basis, with different types of servers having different costs. You can turn instances on and off from the AWS web interface. If you'll be using an AWS instance consistently over a long period of time, it's possible to buy a reserved instance, where you pay a large upfront cost for a lower overall cost.[3] However, we won't be doing that at the moment, although you can spend a couple minutes looking around in the "Reserved Instances" tab to get a sense for how it works.

[1] The subnet labels are actually randomized for each user, so that the one labeled `us-west-2a` doesn't get a disproportionate number of people.

[2] The reservation is *automatically applied* to any eligible instances and is not its own separate instance. As soon as you buy one, no additional action is required; it's purely a change in how your running instances are billed.

[3] The reservation is *automatically applied* to any eligible instances and is not its own separate instance. As soon as you buy one, no additional action is required; it's purely a change in how your running instances are billed.

Logging on to your server

- Follow Amazon's official guide on how to connect to your server. You will need the key file which you previously created and downloaded.
- Following the same instructions, transfer a small test file (anything will do) to your home directory, located at /home/ec2-user/.

Next, we'll set up the server with some standard programming utilities.

- Software can be installed with the yum command. To search for packages, type yum search "x", where x is the string to search for in package names and descriptions. (For example, try yum search "emacs".) The part of the package name before the period (".") is the *name* of the package, and you can install a package with the command sudo yum install <pkg> (where <pkg> is the name of the package and does not include the <> brackets).
- Python comes preinstalled on the Amazon Linux AMI. Type python --version to find out which version of Python is installed.
- We would like to install Python 3 and use it instead of Python 2. Search for packages with python3 in their name and install the main Python 3 package as well as the -devel addon package. To figure out the command associated with Python 3, type python and then press Tab to see the autocomplete options. Run the correct command with the --version flag to verify that the correct version of Python has been installed.
- Briefly read about virtual environments in Python. It's easiest to work with Python on a server within a virtual environment. To create one, first determine the *path* of your Python 3 executable with the which command (try which python for an illustration of how it works). Next, type virtualenv -p <path> ~/venv, where <path> is the path to your Python 3 executable.
- Now that your virtual environment is set up, you can access the Python executable associated with your virtual environment by typing ~/venv/bin/python. Call it with the --version flag to verify that you successfully created a virtual environment with Python 3. Next, Python packages can be installed with by calling ~/venv/bin/pip install <pkg>, where <pkg> is the name of the Python package to install.

You can run Python scripts by calling <exec> script.py, where <exec> denotes the path to a Python executable, but you can also play around in the Python interpreter itself by just running <exec>.

- If you haven't tried using the Python interpreter before, do so. You can exit the interpreter with quit().

Using Python with AWS

We will use Python to perform some natural language processing on a dataset of Amazon reviews. Although the specific dataset we will work with is not very large, these methods generalize well to much larger datasets.

- Read the description of the Amazon review dataset, including the definition of a "5-core" subset of the data.
- Change the directory to `/home/ec2-user/` with the `cd` command and download the 5-core subset of video game review data by running `wget url_of_data`. (*Note:* You can usually paste into a Linux command line with `Shift+Insert`.) After it's done, you can display the contents of the directory with `ls` to verify that it downloaded correctly.
- The data comes in a compressed `.gz` format. Use `gunzip <file>` on the file to decompress it, and then remove the compressed file with `rm <file>`. The extracted data is in a `.json` format. Use `head -n 1 <file>` to view the first line of the file and see what the format looks like.

Getting a subset of the data

When you work with extremely large datasets, it's often the case that you want to take a *subset* of the data when performing exploratory analyses (so you get results faster), and only run full analyses on the entire dataset when you have a better idea of what methods work best.

We'll take a random 10% subset of Amazon video game reviews.

- Use `wc -l <file>` to count the number of lines in the file.

You can use `nano <file>` to open a simple text editor suitable for writing a short Python script. If you prefer, you can install and use Emacs, Vim, etc. In the nano editor, you can press `Ctrl+O` to save a a file, `Ctrl+W` to search, and `Ctrl+X` to exit.

- Write a Python script following these specifications:
  - Use `random.sample()` from the `random` library to sample 10% numbers from 0 to $L-1$, where L is the number of lines in the JSON file.
  - You can use the following Python code to iterate over the lines of a file without reading the entire file into memory simultaneously:

    ```python
    f = open('file.json')
    for i, line in enumerate(f):
        if i == 25:
            # This is the 26th line
    ```

```
    elif i == 30:
        # This is the 31st line
    elif i > 100:
        # Exit the loop
        break
f.close()
```

Referring to the Python documentation, especially the section on file input and output, write code which iterates through the lines of the data file.

- For the ith line, check if $i - 1$ is among the subset of line numbers previously generated; if so, write the ith line to `subset.json`. (You should open `subset.json` once at the beginning of the file and close it at the end instead of reopening the file every time you want to write a line to it.) Make sure to separate each written line from the next with a `'\n'` (newline character).
- The code may take some time to run. If you like, you can print out the line number every 1000 lines for an indication of the script's progress.

When calling the Python script, remember to call the executable corresponding to your installation of Python 3 in the virtual environment.

Next, we'll extract just the text of the reviews in preparation for doing some natural language processing with Python.

- Write a Python script which reads through the lines of `subset.json`. For each line, call `json.loads()` on it (from the `json` library) to turn the line into a dictionary, and then write the `'reviewText'` value of the dictionary to `vidreviews.txt`. (Make sure to separate each line from the next with a space or newline.)

  For an indication of what's happening, you can run the following Python code directly in the console:

  ```
  f = open('subset.json')
  l = f.readline()
  f.close()
  import json
  d = json.loads(l)
  print(d)
  print(d['reviewText'])
  ```

Running vec2pca

Finally, you will be running `vec2pca` on your server to analyze the subset of Amazon reviews.

- Read this description of `vec2pca`.

In order to compile some of the Python packages necessary, you will need to install several programming and numerical computing-related dependencies.

- `yum install` the following packages: `gcc`, `gcc-c++`, `blas-devel`, `lapack-devel`, `atlas-devel`, and `gfortran` packages. While waiting for them to download and install, read the introductions to their linked Wikipedia pages so you know what you're installing![4]

Since the `t2.micro` EC2 instance doesn't have enough RAM to properly install the Python packages we want, you'll have to enable swap space, which uses hard drive storage space as a surrogate for RAM.

- Run these commands to allocate 1 GB of swap space:

  ```
  sudo /bin/dd if=/dev/zero of=/var/swap.1 bs=1M count=1024
  sudo /sbin/mkswap /var/swap.1
  sudo chmod 0600 /var/swap.1
  sudo /sbin/swapon /var/swap.1
  ```

- Using the version of `pip` associated with your virtual environment, install the following libraries: `numpy`, `scipy`, `sklearn`, `gensim`, `nltk`, `beautifulsoup4`, `plac`, and `pandas`. You will need to run three separate calls to `pip`: one to install `numpy`, another to install `scipy`, and a last one to install everything else (you can include them all in the same line). This is because some libraries are dependent upon others.

  These packages may take some time to install. This is a good time to go back and look at any explanatory pages about various packages and utilities which you would like to learn more about.

- The `nltk` library will need some configuration. Open a command-line session of Python, `import` the `nltk` library, and run `nltk.download()`. Enter `d` (for "Download") and then enter `punkt`. Afterwards, type `q` followed by `quit()` to return to the shell.

- Install the `git` package (using `yum`). Go to the Github page for `vec2pca` and click on the green "Clone or download" button. Copy the URL and run `git clone <url>` to download the Github repository's files.

- Finally, `cd` into the `vec2pca` directory and run `~/venv/bin/python vec2pca.py <input> <output>`, where `<input>` denotes your input file of review text and `<output>` is a file to which the algorithm's output will be written. Give the output file a `.html` extension.

[4] In the future, you can run `sudo yum groupinstall "Development Tools"`, which will install a lot of unnecessary packages but will probably give you all the general-purpose compilation-related packages you'll need.

Viewing the results of `vec2pca`

`vec2pca` gives us results in HTML format, which web browsers know how to handle. We'll set up a webserver on our EC2 instance so we can see the results online.

- Install `httpd`, which is a standard, commonly-used web server software package. To start the HTTP server, run `sudo /etc/init.d/httpd start`. We want the HTTP server to automatically start whenever the server is rebooted, so run `sudo chkconfig --levels 3 httpd on`.
- In the Amazon AWS EC2 Instances page, click on your instance. In the "Description" tab below, copy the "Public IP" and paste it into your web browser's URL bar.

Each time we restart the server, its public IP will change, which isn't good for a website! Fortunately, Amazon's Elastic IP service allows us to associate a specific, unchanging IP with our server.

- In the EC2 overview page, click on "Elastic IPs" on the left (under "Network & Security"). Click on "Allocate New Address" and then "Yes, Allocate". You'll see a single entry show up in the table of Elastic IPs. Select it, click "Actions", and select "Associate Address". In the popup window, select your EC2 instance and then click "Associate". When you return to the "Instances" page, you'll see that the "Elastic IP" field, directly underneath "Public IP", is now filled in. Verify that it works by navigating to the Elastic IP of your instance in your web browser.
- The webserver's files should go in `/var/www/html`. Use the `cp` utility to copy the output of `vec2pca` into a file called `index.html` located in the aforementioned directory. (You may need to prefix `cp` with `sudo`, because the `/var` directory requires elevated privileges to modify.)
- Navigate to the Elastic IP and view the results! (One extreme end of each principal component is located at the top of the table and the other end is at the bottom.) Compare your results with the results of those around you, since you all took different subsets of the same dataset. How much variation is there?

Using RStudio in a web browser

To easily develop and run R scripts on an AWS server, it's possible to install a *server-side* version of RStudio which you can access via your web browser. Running RStudio on your `t2.micro` instance will not be much of an improvement over your own computer, but the setup process is identical for more powerful servers.

- Install R (in the `R` package).
- Run the two commands RStudio Server installation instructions for "Red-Hat/CentOS 6 and 7" under the "64bit" section.

You can now use the `rstudio-server` command to start and stop RStudio Server.

- Type `rstudio-server` on its own to view a list of commands.
- Use `rstudio-server status` to check the status of RStudio Server.
- Use `sudo rstudio-server start` and `sudo rstudio-server stop` to start and stop RStudio Server, verifying that its status is changing correctly with the `status` command. Can you start and stop the server without elevating privileges with `sudo`?
- Read RStudio Server: Getting Started and try to nagivate to `http://<Elastic IP>:8787`. Oops – we forgot to allow connections through port 8787 in our security group! In the EC2 dashboard, select "Security Groups" on the left, select the security group associated with your EC2 instance, click "Actions" and then "Edit inbound rules", and add a new rule of type "Custom TCP Rule". Set the "Port Range" to 8787 and the "Source" to "Anywhere".
- Try accessing port 8787 again through your web browser. Oops – we have to log in, but how? Create a new user with `sudo adduser <username>` and set its password with `sudo passwd <username>`, and then try logging into RStudio Server.
- Play around with the web interface a little bit and compare its performance to the speed of RStudio on your own computer.

Shutting off your AWS server

In order to avoid incurring any costs, stop the instance which you created ("Instances" → "Actions"→ "Instance State" → "Stop") and release the Elastic IP you allocated for that instance ("Elastic IPs" → "Actions" → "Disassociate Address" → "Release Addresses").

If you don't think you'll use it again, delete the EC2 instance ("Instances" → "Actions" → "Instance State" → "Terminate") as well as any EBS volumes which you created ("Volumes" → "Actions" → "Delete Volume"). You may need to detach EBS volumes before deleting them.

Parallelizing gradient boosted trees

You should have received as part of this assignment a document with information about how to log in to a more powerful AWS instance's RStudio Server. Follow the instructions!

We'll explore the Wine Quality Dataset which we looked at earlier, using random forests to illustrate how parallelization works in R. Since each constituent tree of a random forest is independent of the other ones, we can train lots of different trees on different processor cores and then combine them all together at the end.

- SSH into the server and download the Wine Quality Dataset CSVs.
- Install and load the `doMC`, `foreach`, and `tictoc` packages.
- Load the CSVs for the red and white wine data with `read.delim()` (don't forget `header=TRUE`). Consolidate them into a single dataframe with which you'll be making predictions for wine quality.
- Using `randomForest()` in the `randomForest` package, regress wine quality against the other variables with `ntree=2000`. Measure and record the elapsed time with `tic()` and `toc()`.
- Run `detectCores()` to check the number of processing cores on the server, and then run `registerDoMC(detectCores())` to tell the parallelization package to use all 8 cores. Call `getDoParWorkers()` to verify that all 8 cores have been successfully registered.

Here is an example of using `foreach()` for parallelization:

```
foreach(i = 1:3) %dopar% {
  sqrt(i)
}
```

Of course, this example is too simplistic to be worth parallelizing, but it illustrates how to properly structure the function calls. Each iteration of the code block after `%dopar%` is given a different value of `i` and forked off to its own processor core. By default, each of the individual results is returned, but the parameters of `foreach()` can be set so that it automatically combines the individual results into a single final result.

- Read the documentation for `foreach()`. The random forests algorithm can be parallelized by calling `foreach()` as follows:
  - `ntree` should be a vector where each entry corresponds to the number of trees for that specific core. In order to properly compare timing results, set `ntree` so that we end up fitting the same total number of trees as we did earlier.
  - `.combine` should be set to `combine`, because the `combine()` function is used to combine different random forests together.

  - Read the documentation for `combine()` and set the `.multicombine` parameter appropriately.

- Run your code for parallelized random forests, measuring the total elapsed time with `tic()` and `toc()`. Compare the elapsed time with the time required for unparallelized random forests. (If you need more clarification on how to use `foreach()`, refer to the vignette for `doMC`.)

Some packages will automatically handle the parallelization for you. In particular, the `xgboost` package is a very popular and well-designed package for parallelized gradient boosted trees.[5]

- Read through the vignette for `xgboost`.
- Using the optimal hyperparameters for `gbm()` which you found in the nonlinear techniques assignment, compare the time it takes to use `gbm()` versus `xgboost` for the task of training a gradient boosted tree to predict wine quality.

Advanced topics

There are many things you can do with AWS, and we can only hope to cover but the minutest fraction of them. Here, we'll point you to the right place to look for certain relatively common tasks for which you might want to use AWS.

Even if you don't have immediate need for this information, skim through it regardless so you have a better sense of what's possible with AWS.

Storing large amounts of data

It's the world of *big data*, after all!

- You can create dedicated data storage drives with Amazon's Elastic Block Storage system for data that you're using on a day-to-day basis. EBS volumes cost the same amount of money no matter how much of the preallocated space you're using, so you don't want to make volumes that are too large or retain too much old data on EBS drives.
- This StackOverflow question has a broad overview and comparison of Amazon's data storage methods. S3 is a good choice for a very large amount of data which you need to access, and Glacier is good for very long-term storage.

[5] You may ask: since gradient boosted trees are *sequential*, how can they be parallelized? Well, they aren't parallelized in the same way as random forests; the parallelization takes place *within* each tree, with different branches being dealt with by different processor cores.

- Amazon's AWS blog has a detailed blog post with instructions on how to read data from Amazon S3 into R.

Scraping the web with multiple IPs

Very often, web scraping is limited by the number of IPs you have access to: APIs may have a restriction on the total number of requests you can make per IP per day, web servers may limit the amount of download speed available to each IP, and so on and so forth.

Amazon's Virtual Private Cloud can be combined with Elastic IPs in order to easily access multiple public IPs from the same EC2 instance. Amazon has instructions on how to do this.

In practice, what this means is:

- You go to the "Network Interfaces" tab and add lots of private IPs to your instance, and then assign an Elastic IP to each one. (There is, unfortunately, a preset limit on the number of private IP addresses you can have for each instance.)
- Refresh the network with `sudo service network restart` and view the list of IP addresses associated with the instance with `ip addr li`.
- Test whether or not the multiple IP address setup works by running `curl --interface <private-ip> http://checkip.amazonaws.com`, which should return the (public) Elastic IP associated with that specific private IP address.
- If you are scraping data with a Python library (`urllib`, `requests`, etc.), you can run the following code[6] to make all subsequent outbound network connections go through the network interface associated with the private IP `sourceIP`:

  ```
  import socket
  true_socket = socket.socket
  def bound_socket(*a, **k):
      sock = true_socket(*a, **k)
      sock.bind((sourceIP, 0))
      return sock
  socket.socket = bound_socket
  ```

  For ease of usage, you can also configure a script so that it accepts the private interface IP as a command-line argument.

Tangentially: Michael Nielsen's post on How to crawl a quarter billion webpages in 40 hours is fairly illustrative.

[6]From this StackOverflow answer

Parallelizing more operations in R

Refer to CRAN Task View: High-Performance and Parallel Computing with R.