A Tutorial on Deploying and Using Amazon Elastic Cloud Compute Clusters

John Beieler
Pennsylvania State University
jub270@psu.edu

February 17, 2013

The Cloud and You

With the datasets analyzed by Political Scientists growing ever larger and analysis becoming more complex, it is often necessary to utilize more powerful computing resources. Research using large amounts of network data, event data, or textual data often pushes the limits of what an individual machine can accomplish in a reasonable period of time, if at all. The use of cloud resources allows tasks that take a large amount of time to be offloaded to a remote server in order to free up the user's local machine. Alternatively, for tasks larger than one computer can handle, one can divide and distribute a job across a cluster of servers. While many universities offer high-performance computing resources, it is often the case that the user does not have free reign over what software is installed; it can take hours, days, or even weeks to have a required piece of software installed by the server administrators. Additionally, jobs on university resources are typically restricted to a certain run length, such as 24 hours. The use of a remote server that you rent, for as little as two cents per hour, enables whatever software is necessary to be installed when desired, and for jobs to be run as long as required. Amazon's Elastic Compute Cloud (EC2) environment provides access to these cloud resources for the rental of a server or cluster of servers. Amazon EC2 allows for the creation of a cluster with up to 20 machines, each with multiple processing cores available. This is a large amount of computational power available on demand and at relatively low cost.

While EC2 offers quick and straightforward rental of computing resources, setting up and managing the servers comes with a rather steep learning curve. This article provides a brief introduction to the setup and use of EC2 resources. The focus is on the use of the `Starcluster` utility for creating and managing EC2 clusters. Following this, I provide a brief overview of using `R` and Python in parallel on a server cluster. Code and examples for the routines presented below are hosted on github.

Starcluster and EC2

Before starting an analysis on an EC2 server, it is necessary to follow a few steps to set up the server. There are two primary components of an EC2 server: the instance, which refers to the hardware used, and the Amazon Machine Instance (AMI), which refers to the software deployed on the machine such as the operating system and other packages. In order to ease the deployment of an EC2 instance, `Starcluster` was developed by the STAR program at MIT.

The following sections walk through the installation and configuration of `Starcluster`. This entails installation of `Starcluster`, the creation of an Amazon Web Services (AWS) account, the creation of an Elastic Backed Stores (EBS) volume for the storage of user data, and finally the installation of software for analysis, such as `R`, and the attendant libraries and packages, such as `joblib` in Python and `snow` in `R`.

Installing Starcluster

Since `Starcluster` is based on Python it is possible to easily install the utility using the `easy_install` method.[1] Unfortunately, `easy_install` does not come prepackaged with a Python distribution. To install `easy_install` in a Unix-like environment, download the appropriate Python egg from the Python Package Index, change into the directory that contains the egg, and run the shell script. Installing Starcluster requires the presence of a C compiler. On OS X, XCode must first be installed from the App Store. Within XCode, the command-line tools must be installed by selecting in the menubar `XCode -> Preferences -> Downloads` and installing the `Command Line Tools`. On other Unix-like systems, if a C compiler is not already installed, one can be loaded using the package-management method used on that particular distribution. This will then allow the installation of `Starcluster` on the local machine.

```
#Change to the directory where the egg was downloaded
cd ~/Downloads
#Execute the shell script
sh setuptools-0.6c11-py2.7.egg
#Install starcluster
sudo easy_install StarCluster
```

Following the install, typing `starcluster help` into the command-line will bring up dialogue asking the user to select an option for the configuration file. At this point type `2`, which will save the `config` file in the `~/.starcluster` directory. The following section describes the information the `config` file should contain.

Configuring Starcluster

With `Starcluster` installed, it is necessary to set up the configuration file for `Starcluster`. There are two basic parts to the configuration file: user information and instance information. The following sections provide guidance for adding the AWS user information to the `config` file, as well as adding the various templates and other options necessary to create an EC2 instance.

Configuring User Information

All configuration options for `Starcluster` are found in the `config` file located in the `~/.starcluster` directory. The default `config` provided by `Starcluster` has numerous comments and options. It is good to keep these in the file in order to see the available options, but in the interest of clarity I have provided a cleaner `config` with the configuration discussed in this article at the github link provided earlier.

The first step to configuring `Starcluster` is to create an Amazon AWS account. Once an account is created, navigate to the "Security Credentials" page, which can be found in

[1]This tutorial will assume the reader is working in a Unix-like environment such as Linux or OS X and has Python already installed. If on Windows, tutorials on installing Python and `easy_install` can be found here and here.

the drop-down menu entitled “My Account/Console” at the top right corner of the page displayed immediately after login. Once on the “Security Credentials” page, in the “Access Credentials” section there is the option to create a personal access key. First create an access key, and then copy down the Access Key ID, the Secret Access Key, and the Account Number, which can be found toward the top of the page under the name used to register the account. This information should be placed in the `config` file in the section entitled `AWS Credentials and Connection Settings`. To open and edit the `config` file, execute the following commands:[2]

```
  cd ~/.starcluster
  vim config
  #Or depending on your editor preferences
  emacs config
  #On Mac, the following will open the config file in TextEdit
  open config
```

Next, a pair of SSH keys must be created. SSH stands for secure shell, and is a method for “tunneling” securely from one computer to another. `Starcluster` uses SSH to allow to access to the EC2 instance. To create the key pair, execute the following commands:

```
  #If the ~/.ssh directory does not exist
  mkdir ~/.ssh
  starcluster createkey aws_key -o ~/.ssh/aws_key.rsa
```

This will create a key in the ~`/.ssh` directory on the local machine. This information should then be added to the `config` file in the `Defining EC2 Keypairs` section. The updated section should read as follows:

```
[key aws_key]
KEY_LOCATION=~/.ssh/aws_key.rsa
```

The next step is to create an Elastic Backed Storage (EBS) volume in order to store data in a persistent manner.[3] In the AWS management console, which is accessed by clicking the “My Account/Console” link at the top of the page after logging in to AWS, navigate to the EC2 section, followed by the “EBS Volumes” page under “My Resources.” Once on this page, create a new volume, with volume type of “standard” and the desired amount of storage[4], and copy down the Volume ID to add to the `Configuring EBS Volumes` section of the `config` file. For this example, the EBS volume will be named `data` and will be mounted on the cluster at `/root/data`. This gives the following configuration:

[2]It might be useful at this junction to point out that the use of a “programmer’s” editor will likely be necessary. When working on a remote server it often is not possible, or is very difficult, to use a graphical editor. Text-based editors such as vim or emacs come preloaded on almost every instance of Linux available. They are very powerful and useful, but come with a fairly steep learning curve.

[3]You can store data on the instance itself, but if you terminate the cluster the data is deleted. EBS storage allows you to terminate and restart clusters and keep the same data.

[4]For the examples used in this article a small amount of storage is necessary; 5 GiB should suffice.

```
[volume data]
VOLUME_ID = #INSERT ID
MOUNT_PATH = /root/data/
```

The final step is to uncomment, i.e., delete the "#" symbols around, the `ipcluster` plugin, which is located roughly around line 280 in the plugins section of the configuration file. After completing these steps the `config` file is properly setup with the basic user and configuration information. The next step is to define various server configurations that will be used.

Configuring Cluster Templates

There are three primary components to the setup of a server: the AMI used, the instance type, and the size. The individuals at the STAR program have generously provided public AMIs that have many of the components necessary for scientific research such as Python, OpenMPI, and the Sun Grid Engine, already present. The next section will cover creating your own AMI as an alternative to using the STAR AMIs. The second component necessary to create an EC2 instance is the instance type; Amazon offers numerous instance types with varying configurations and prices. For the purposes of this tutorial, the 64-bit `Starcluster` AMI will be used on the M1 Extra Large instance type. This leads to the following configuration, defined in the `Defining Cluster Templates` section:

```
[cluster base]
KEYNAME = aws_key
CLUSTER_SIZE = 1
CLUSTER_USER = john
CLUSTER_SHELL = bash
NODE_IMAGE_ID = ami-999d49f0
NODE_INSTANCE_TYPE = m1.xlarge
VOLUMES = data
PLUGINS = ipcluster
```

While this setup is sufficient for many use cases, there are other situations that might require more memory or more nodes in the cluster. `Starcluster` allows for the definition of further templates in the `Defining Additional Cluster Templates` section. The following code defines a small cluster with the same basic characteristics as the `base` configuration, but starting two nodes instead of one.

```
[cluster basecluster]
EXTENDS = base
CLUSTER_SIZE = 2
```

The final step is to head back up to the beginning of the `config` file and define the `DEFAULT_TEMPLATE` as `base`.

Using the Cluster

With all of the configuration options defined, the EC2 instance can finally be spun up and used. The following code will start a server named "mycluster" which can then be accessed using SSH. The first attempt to SSH into a server will be met with a long message about unknown hosts. This is nothing to worry about; just type "yes" and hit return.

```
starcluster start mycluster
starcluster sshmaster mycluster
```

Alternatively, a cluster following the `basecluster` configuration can be started by running:[5]

```
starcluster start mycluster -c basecluster
starcluster sshmaster mycluster
```

Users must issue the `starcluster terminate mycluster` command to shut the EC2 instance down. *It is important to note that if the instance is not explicitly terminated the instance will keep running and you will be charged for the uptime of the server.* Once connected to the cluster operation is the same as any Unix command-line interface.[6] In order to exit the server and terminate the instance, the following commands are used:

```
#This will exit the EC2 instance and returns to the local machine
exit

starcluster terminate mycluster
```

Adding Software and Creating AMIs

Creating a custom AMI is optional for the use of an EC2 instance. If one desires, software and packages can be loaded to the instance each time it is started. Having an AMI with all of the software pre-loaded, however, can save a large amount of time and repetitive action. In addition, having this AMI created will ensure that the same software is installed on all nodes within a cluster. Thus, the following example shows how to load the libraries and software needed for the rest of this tutorial, such as `R` and various Python packages, and how to create an AMI that will allow this same software configuration to be loaded repeatedly. This section assumes that the reader wishes to create a custom AMI to be reused. If not, the commands used to install software should be issued on the instance used to run analyses. The following commands start a special type of EC2 instance configured for the creation of AMIs and SSHs into the instance as usual.

[5]I advise against having multiple, different instances running at once. It is far too easy to forget how many are running, which can lead to a rather large (and unexpected) bill at the end of the month

[6]EC2 will close the connection if the session does not receive any input. This means that if a job or script is running, but nothing is typed into the terminal, EC2 will close the connection and the progress on the job will be lost. Thus, it is often a good idea to split jobs into GNU Screen sessions. This is done by typing `screen -S analysis`, which will create a screen session named `analysis`. To exit from a screen session simply use the command `Ctrl-a-d`, which indicates that the control key should be held down while pressing a then d. Then the screen session can be resumed using `screen -r analysis`.

```
starcluster start -o -s 1 -i m1.xlarge -n ami-999d49f0 imagehost
starcluster sshmaster imagehost
```

The AMI that the custom AMI is built upon runs on the operating system Ubuntu, which is a specific distribution of Linux. Ubuntu uses the command `apt-get install` to install programs.[7] Before installing packages, however, it is necessary to tell Ubuntu to look in a different location for the newest version of `R`. In the file `/etc/apt/sources.list` add the line `deb http://cran.mtu.edu/bin/linux/ubuntu oneiric/`. This will allow the newest version of `R` to be installed instead of version 2.13, which is usually installed by using `apt-get`.

```
apt-get update
apt-get install r-base-core

easy_install pip
pip install pandas
pip install joblib

R
```

Once inside the `R` session, it is necessary to install several packages that will be of use later when performing analyses in parallel. The specific packages used are `foreach` (Analytics 2012*c*), `doParallel` (Analytics 2012*a*), `snow` (Tierney, Rossini, Li and Sevcikova 2012), and `doSNOW` (Analytics 2012*b*). After these packages are installed, `R` can be exited, as can the instance itself.

```
install.packages('foreach')
install.packages('doParallel')
install.packages('snow')
install.packages('doSNOW')
q()

exit
```

Now back at the local machine's command line, it is time to create the custom AMI. Executing the following code will create an AMI with a unique AMI ID that can be placed in the `config` file in place of the STAR AMI that is currently in use. In other words, the new AMI ID would replace `ami-999d49f0` in the configuration.

```
#Note instance ID
starcluster listinstances
starcluster ebsimage <INSTANCE-ID> analysis-image
#Note the new AMI ID that prints out

starcluster terminate imagehost
```

[7]Reference is often seen to `sudo apt-get install`; the presence of `sudo` tells the machine to run the following command as the root user. This addition is unnecessary in this situation since the user is logged in to the instance as the root user already.

Performing Analyses on a Cluster

The server has now been configured and `R`, along with other libraries and packages, has been installed. At this point `R` can be used as it is on any other machine, but with the potential for much more computational power. The features of EC2 can be utilized for either cluster or multicore processing, depending on the problem. One potential situation that can arise when analyzing big data is that a long job needs to be offloaded onto a remote server. In this situation multicore processing, which is the use of multiple cores on a single CPU within the machine, may be advantageous. If a specific task is too large for a single machine to handle, due to issues such as RAM limitations, cluster computation, or using multiple machines to perform the computations, may be warranted. It is important to note that the use of a cluster is not always necessary; sometimes a machine with a large amount of RAM is sufficient for the task and will allow for greater simplicity (Rowstron, Narayanan, Donnelly, O'Shea and Douglas 2012). The various instance types available on EC2 allow for the selection of the proper setup for either situation.

Given these two different environments, multicore or cluster, different steps are required depending on which is utilized for a given task.[8] This section focuses on "embarrassingly parallel" situations of the type commonly encountered in basic data cleaning, data subsetting, or data simulations, e.g. Monte Carlo simulations. I define "embarrassingly parallel" problems as those that can commonly be approached using a for-loop in a computer program. The running example used is a function that generates 100 draws from the uniform distribution 100 times, which are then transformed to the exponential distribution from which a mean is calculated. This function is then called 100 times.[9]

Using R on EC2

Multicore

The following example makes use of the `foreach` library in `R`. The additions needed to run code in parallel in a multicore setting are to register a parallel backend, using `makeCluster()` and `registerDoParallel`, and the addition of `%dopar%` instead of `%do%` before the function being used, which will call the function in parallel instead of sequentially. In `R`,

```
library(foreach)
library(doParallel)

cl = makeCluster(3)
registerDoParallel(cl)

unif.trans = function(){
```

[8]An added level of complexity not covered is the combination of cluster and multicore computing. In short, one can create a job that shares work amongst nodes on a cluster, which is then further divided amongst multiple cores. To achieve this, one must simply combine the two different sets of code outlined in the multicore and cluster sections below.

[9]This is an admittedly trivial example, but it shows the basics of how a Monte Carlo simulation might proceed in a cluster or multicore environment.

```
    results = matrix(nrow=100,ncol=100)
    for(i in 1:100){
    results[i,] = runif(100)
    exponential = -log(results)
    }
    return(mean(exponential))
  }

x = foreach(i=1:100, .combine='c') %dopar% unif.trans()
mean(x)
```

Cluster

Performing analysis utilizing a cluster of computers uses a similar approach, but requires some communication between the various nodes within the cluster. `R` has some packages, such as `snow` and `snowfall`, that assist in this. The first step is to create a cluster with more than one node in it, such as defined by the `basecluster` template. The following code assumes that any other instances named `mycluster` have been terminated. Additionally, the following should be executed on the user's local machine in order to create the new EC2 instance.

```
  starcluster start mycluster -c basecluster
  #Note the IP Address of the instances in the cluster
  starcluster listclusters
```

The approach for analysis on a cluster environment is extremely similar to that for a multicore environment; the main difference lies in how the parallel backend is created. For the cluster setting, it is necessary to specify the IP addresses of the nodes included in the cluster.

```
library(snow)
library(doSNOW)
library(foreach)

#Replace MASTER and NODE001 with the appropriate IP Address
cl = makeCluster(c('MASTER.compute-1.amazonaws.com',
'NODE001.compute-1.amazonaws.com'), type='SOCK')
registerDoSNOW(cl)

unif.trans = function(){
    results = matrix(nrow=100,ncol=100)
    for(i in 1:100){
    results[i,] = runif(100)
    exponential = -log(results)
    }
    return(mean(exponential))
    }

x = foreach(i=1:100, .combine='c') %dopar% unif.trans()
mean(x)
```

`snow` also comes packaged with parallel and cluster versions of the `apply` family of functions. A more detailed discussion of these can be found in the `snow` documentation.

Using Python on EC2

Multicore

The easiest approach for implementing embarrassingly parallel for-loops in a multicore situation in Python is the `Parallel` functionality of the `joblib` package. The code below illustrates the use of `joblib` in the same toy simulation used in the `R` examples. The heart of the code below is the call to the `Parallel` function. The `n_jobs` argument tells the function how many cores to use; -1 indicates the use of all available cores. The `uniform_trans` function is then called 100 times, with these 100 calls split across all available cores.

```
starcluster sshmaster mycluster
ipython
```

```
from joblib import Parallel, delayed
import numpy as np
import scipy.stats as stats

def uniform_trans():
    results = list()
    for i in xrange(100):
        results.append(stats.uniform.rvs(size=100))
    results = np.asarray(results)
    exponential = -np.log(results)
    return np.mean(exponential)

means = Parallel(n_jobs=-1)(delayed(uniform_trans)()
                 for _ in xrange(100))
finalMean = np.mean(means)
```

Cluster

For analysis on a cluster using Python, the developers of `Starcluster` had the foresight to include the IPython (Pérez and Granger 2007) cluster plugin. This allows analysis to be easily forked to different nodes within a server. The main difference is that it is necessary to login to the EC2 instance with a different user than root, hence why the user `john` was defined in the `config` above. The basic functioning of the below code is to create a `Client` object, which contains the information about the available nodes in the cluster. The `nodes` object is then assigned all possible worker nodes. The asynchronous map function is then used to split the code between the nodes and collect the results in an asynchronous manner.[10] A final point of interest is that since the function is being sent to various worker nodes, it is necessary to import the appropriate packages within the function itself.

```
starcluster sshmaster mycluster -u john
ipython
```

```python
from IPython.parallel import Client
import numpy as np

cluster = Client(packer='pickle')
nodes = cluster[:]

def uniform_transform(z):
    import scipy.stats as stats
    import numpy as np
    results = list()
    for i in xrange(100):
        results.append(stats.uniform.rvs(size=100))
    results = np.asarray(results)
    exponential = -np.log(results)
    return np.mean(exponential)

gather = nodes.map_async(uniform_transform, xrange(100))
means = gather.get()
mean = np.mean(means)
```

Resources and Final Thoughts

While this article serves as an extremely brief introduction, there are many resources available for exploring parallel computation in R and Python in greater depth. Before explicating these resources, however, a final note on using parallel processing is in order. For small examples like the one used in this article, it is sometimes *slower* to run the process in parallel due to the scheduling and recombining of results. It is important to identify the bottleneck in your workflow. If you are trying to fit a model to a large amount of data and hitting memory limits, it is likely easier to use the high memory EC2 instance with 68 GiB of memory.[11] On the other hand, if your data subsetting script is taking an hour or more to run, a cluster or multicore solution might be useful. In addition, some problems are not

[10]Further explanation of `map_async` is located in the IPython Documentation.

[11]In fact, as I was writing this I received an email from Amazon announcing new types of instances that have 244 GiB of RAM and two Intel Xeon processors, which each have 8 cores for 16 total physical cores and 32 total threads. In reality this instance should be more than enough firepower for nearly any application that could arise in political science research.

easily parallelized while others are not parallelizable at all. The type of algorithms that are easily parallelized, however, could serve as the subject of an entirely different article.[12]

With that said, if multicore or cluster computing is the best way forward for a given problem there has been a copious amount of (digital) ink spilled outlining the various options available for parallel computation in both `R` and Python. In `R` there are the `foreach`, `snow`, and `snowfall` packages discussed in this article, in addition to the various implementations of `apply`.[13] There are also explicit implementations of MPI in `R` such as Rmpi, a good example of which, along with a less trivial usage of parallel processing than presented in this article, can be seen here. In Python, MPI is also available, as is the `multithread` package. The easiest and most straightforward approach, however, is to make use of IPython and `joblib`. These two should cover almost any imaginable scenario. With this in mind, the aim of this article was not to provide an exhaustive tutorial on parallel computation; in reality this would devolve into a repetition of the documentation for the various implementations mentioned above. Rather, the hope is that this article has provided the reader with a working understanding of, and a quick-start guide for, 1) initiating and running an AWS EC2 instance and 2) utilizing an EC2 instance for the purposes of parallel computing in `R` and Python.

[12]In short, if an algorithm contains the summation of results it is probably possible to run it in parallel.

[13]A good resource for a high level overview for some of these commands is Ryan Rosario's presentation on parallelizing R, available here.

References

Analytics, Revolution. 2012*a*. *doParallel: Foreach parallel adaptor for the parallel package*. R package version 1.0.1.
URL: *http://CRAN.R-project.org/package=doParallel*

Analytics, Revolution. 2012*b*. *doSNOW: Foreach parallel adaptor for the snow package*. R package version 1.0.6.
URL: *http://CRAN.R-project.org/package=doSNOW*

Analytics, Revolution. 2012*c*. *foreach: Foreach looping construct for R*. R package version 1.4.0.
URL: *http://CRAN.R-project.org/package=foreach*

Pérez, Fernando and Brian E. Granger. 2007. "IPython: a System for Interactive Scientific Computing." *Comput. Sci. Eng.* 9(3):21–29.
URL: *http://ipython.org*

Rowstron, Antony, Dushyanth Narayanan, Austin Donnelly, Greg O'Shea and Andrew Douglas. 2012. Nobody ever got fired for using Hadoop on a cluster. In *HotCDP 2012 - 1st International Workshop on Hot Topics in Cloud Data Processing*. Bern, Switzerland: https://research.microsoft.com/pubs/163083/hotcbp12%20final.pdf.

Tierney, Luke, A. J. Rossini, Na Li and H. Sevcikova. 2012. *snow: Simple Network of Workstations*. R package version 0.3-10.
URL: *http://CRAN.R-project.org/package=snow*