Using Amazon's EC2

Chris Paciorek

March 11, 2014

This document provides a tutorial on using Amazon's EC2 to do cloud computing, with a focus on statistical users. The syntax here is designed for SCF users, but should provide guidance for non-SCF users as well.

The basic idea of EC2 (and other cloud computing resources such as Google Compute Engine) is to allow you to start up one or more virtual computers that run on Amazon's servers. You can link together multiple virtual computers into your own computer cluster. Each virtual computer is called an 'instance' (in Amazon's terminology). I'll also refer to it as a 'node', mimicing the idea of having a cluster with multiple nodes. The advantage of all this is that you have access to as many computers as you want, but you only pay for them for as long as you use them.

The instructions below assume that you are logged in to an SCF Linux machine and have access to your SCF home directory.

1 Getting an account and setting up payment

1. For most users (those users affiliated with the SCREMS grant see below), you'll need to establish an account with Amazon Web Services that you either pay for directly yourself or link with a grant. See http://aws.amazon.com. To bill to a grant, please contact consult@stat.berkeley.edu and Jane Muirhead (statgrants@stat.berkeley.edu), and we'll work with you to set up the billing so that Amazon bills the university. Once you set things up, you'll be assigned an AWS_ACCESS_KEY and AWS_SECRET_KEY that will be your authentication, allowing you to start instances. You can log on to the AWS console to view your account details (such as billing) at http://console.aws.amazon.com/console. You can go to the EC2 console to monitor instances (and start and stop them via point and click) at http://console.aws.amazon.com/ec2.

2. If your work is affiliated with the SCREMS grant (SCREMS faculty PIs are Nolan, Huang, Kaufman, Purdom, Yu), get permission from the appropriate PI and then contact

consult@stat.berkeley.edu (cc'ing the PI). We'll arrange to have you added to the appropriate account with Amazon. We'll provide you with an AWS_ACCESS_KEY and AWS_SECRET_KEY that will be your authentication, allowing you to start instances, as well as the SCREMS AWS user ID. We'll also provide you with a username (your SCF email address) and a password that you can use to log on to a graphical interface that will allow you to monitor instances, as well as starting and stopping them. The URL is https://scf.signin.aws.amazon.com/console. Once logged on, click on the EC2 icon. Please be careful as all users have access to all instances. So please do not stop somebody else's instance.

IMPORTANT: You will be billed for the entire time an instance is running, regardless of whether you are running anything on it. So make sure to terminate the instance when you are done. See Section 5 for details.

2 Starting a compute instance in the cloud

EC2 cloud computing can be managed by a variety of tools:

1. The EC2 console website mentioned above
2. EC2 command line tools
3. the Boto Python package
4. the StarCluster tools

You can use any of these that you wish (the last three sets of tools are all installed on the SCF Linux machines), but this tutorial only covers the **StarCluster tools**. The EC2 command line tools are somewhat clunky and require one to parse strings when automating operations. Boto requires you to use Python, and you would have to either keep the Python session open or know how to save the appropriate information. Please contact consult@stat.berkeley.edu if you'd like more information about any of the tools.

Note that the EC2 command line tools are available in */usr/local/linux/ec2-api-tools/bin*. If you want to use one of the commands, you need to use the entire path, e.g.,

```
   ./usr/local/linux/ec2-api-tools/bin/ec2-run-instances ami-1aab222a
-k ec2new -t m1.small
```

or you need to have */usr/local/linux/ec2-api-tools/bin* in your path; you can do this by putting the following in your *.bashrc* file:

```
   export PATH=$PATH:/usr/local/linux/ec2-api-tools/bin
```

2.1 Getting set up with StarCluster

In general you can do the following to get help with StarCluster:

```
starcluster --help
```

There also online overview, http://star.mit.edu/cluster/docs/latest/overview.html, quickstart, http://star.mit.edu/cluster/docs/latest/quickstart.html, and user manual, http://star.mit.edu/cluster/docs/latest/manual/index.html, documents online.

First, you need to set up a configuration file. At the command line, do

```
starcluster help
```

and choose option #2. This will place a template configuration file in *~/.starcluster/config*.

Edit the file in a text editor and save the changes.

1. Add your AWS_ACCESS_KEY as the *aws_access_key_id* and your AWS_SECRET_KEY as the *aws_secret_access_key*.

2. Add your Amazon Web Services (AWS) user id as the *aws_user_id*.

Most users (i.e., non-SCREMS users) will now need to set up a public-private key pair to allow you to do password-less ssh:

```
starcluster createkey ec2star -o ~/.ssh/ec2star.rsa
```

The private key will be deposited in *~/.ssh/ec2star.rsa*, while Amazon will deposit the public key in all the EC2 instances that you create so you can ssh to them without a password. Once you start an instance, you could see that public key in the *.ssh* folder on the instance, but you shouldn't have any need for it. (You can use a different name than *ec2star* for the keyname or *ec2star.rsa* for the name of the file with the private key, if you prefer.)

However users affliliated with the SCREMS grant will receive a file from the SCF containing the private key, called *ec2star.rsa*. Save this file as *~/.ssh/ec2star.rsa*.

Note that if you want to access EC2 from a different machine, you'd need to copy the *~/starcluster/config* file and the private key over to that other machine.

Now add *~/.ssh/ec2star.rsa* as the value of *KEY_LOCATION* in the config file, naming the *section* for this key as *ec2star*.

```
[key ec2star]
KEY_LOCATION=~/.ssh/ec2star.rsa
```

EC2 has computers in different locations. We'll set things up so you use the Oregon location (us-west-2) - it's fairly close geographically and cheaper than machines in the Northern California location (us-west-1) [presumably this could change in the future]. Set the following in the config file:

```
AWS_REGION_NAME = us-west-2
```

```
AWS_REGION_HOST = ec2.us-west-2.amazonaws.com
```

StarCluster allows you to set settings for different kinds of clusters (different numbers of nodes, different machine capabilities (memory, CPU, etc.)). For now we'll just modify the *smallcluster* template. The template that is used by default when starting instances is defined by DEFAULT_TEMPLATE at the top of the config file.

Set the following in the config file:

```
keyname = ec2star
CLUSTER_SIZE = <size>
CLUSTER_USER = <yourSCFusername>
NODE_IMAGE_ID = ami-2e48c31e

NODE_INSTANCE_TYPE = m1.small
DISABLE_QUEUE=True
```

where <*size*> should be the number of nodes you want to start. The CLUSTER_USER variable results in setting up a user account on the instances with the same name your SCF user name. We set DISABLE_QUEUE as we don't need any scheduling software and installing it takes a bit of time. The NODE_IMAGE_ID specifies the operating system - the one above is Ubuntu 12.04, which is what we are using on the SCF. If you happen to use the us-west-1 region, the analogous NODE_IMAGE_ID is *ami-5caa8a19*. [Recent update - to use the base SCF image, which already contains standard computational software such as R, use *ami-0e4c2f3e* in the us-west-2 (Oregon) region.] For the NODE_INSTANCE_TYPE, you can choose amongst the types listed at http://aws.amazon.com/ec2/instance-types/, entering the "API name" that is the last row of information for each instance type. These vary by the CPU and number of cores on each virtual machine. Note that some types are not available in the us-west-2 region. You can look at pricing and which types are available by region at http://aws.amazon.com/ec2/pricing/. Make sure to click on the right region under "On-Demand Instances". Those with more cores are more expensive but you can do more calculations, providing your code uses parallelization.

For your particular job, you may want to experiment with types with different numbers of cores, different CPU capabilities, and optimized network performance for distributed memory calculations (the Cluster Compute Eight Extra Large instance, *cc2.8xlarge*). You'll need to know how much memory your job will take at its maximum memory use. It may be the case that starting many *m1.small* instances, which are cheap, is more cost-effective than starting fewer more expensive instances. Please contact consult@stat.berkeley.edu if you'd like to discuss these issues further. Note that you can use 750 hours per month of the *t1.micro* type for free (http://aws.amazon.com/free/), so that may be a good way to test out EC2, provided 613 Mb of memory is not a limitation. Also note that for *t1.micro* you need to use EBS storage (see below)

2.2 Starting a cluster and installing software

2.2.1 Starting a cluster

First, to simplify things below, set an environment variable with the user name specified as the CLUSTER_USER in the config file. Here we'll assume it's the same as your SCF user name, which is contained within the $USER environment variable.

```
scuser=$USER
```

Starting a cluster is as easy as

```
starcluster start mycluster
```

You can now ssh to the master node (the only node if you set CLUSTER_SIZE = 1 in the config file) as a regular user:

```
starcluster sshmaster -u $scuser mycluster
```

Note that if you want to login as root user (i.e., with administrator privileges), this is the default user, so just leave off the *-u $scuser* syntax. To ssh to a slave node:

```
starcluster sshnode -u $scuser mycluster <nodeName>
```

where *nodeName* follows the pattern *node001*, *node002*, etc.

You can get information about your cluster(s):

```
starcluster listclusters
```

Note that you can log off of the SCF machine and log back in and still access your cluster.

Finally we'll set up a variable containing the names of the nodes we created. This isn't necessary but will help in automating some of the steps below.

```
numNodes=`starcluster listclusters mycluster | grep "Total nodes"
| cut -d' ' -f3`
nodes="master `eval echo $(seq -f node%03g 1 $(($numNodes-1)))`"
```

Note that if you only have the master, do

```
nodes=master
```

2.2.2 Putting software on your instance

If you are using an image provided by SCF or an image you've already created, then you can skip this step, provided the image has all the software you need already.

If not, the next step is to put the software you want on your instances. In general, we'll use the *apt-get* mechanism of Ubuntu to install pre-existing Ubuntu packages, but you can also build and install software from scratch on the instances. If you need help with this, contact consult@stat.berkeley.edu.

We'll install R and Octave (an open source version of Matlab). It turns out the default StarCluster image already has Java, a threaded BLAS for linear algebra (openBLAS), and openMPI for distributed computing. (If you are using a different image, then add "`libopenblas-base openmpi-bin libopenmpi-dev openjdk-7-jre openjdk-7-jdk`" to the *apt-get install* line below to get these packages.

You can logon to each node and run the following commands. However, to do this automatically on multiple nodes, insert the following two lines in a bash script file, which I call *install.sh* here:

```
apt-get update
apt-get install -y r-base octave3.2
echo "DONE with Ubuntu package installation on $(hostname -s)."
```

Now run the script file on each node using the following commands:

```
starcluster put -u $scuser mycluster install.sh .
# first we'll install on the master
starcluster sshmaster mycluster "source /home/$scuser/install.sh >&
   /home/$scuser/install.log.master"
# answer 'yes', then hit Ctrl-Z to put the installation in the
#  background, then do the following to install on the nodes
for node in $nodes; do
   cmd="source /home/$scuser/install.sh >& /home/$scuser/install.log.$node"
   if [ "$node" != "master" ]; then
      starcluster sshmaster mycluster "ssh $node $cmd" &
   fi
done
```

You'll need to wait a few minutes for this to finish - to check on it, you can do the following, which will continually update itself. Just wait until all the nodes report as being DONE.

```
watch starcluster sshmaster mycluster "grep 'DONE'
    /home/$scuser/install.log.*"
```

If you need additional software to include in the list of packages indicated in *install.sh*, you can search amongst available Ubuntu packages (here e.g. for octave packages) with

```
apt-cache search octave
```

And you can see if a specific package is installed and what version would be installed if you did install it:

```
apt-cache policy octave3.2
```

Next we'll install some standard R packages for parallel computing. You should add whatever additional R packages you need to the list of packages below, or you can of course omit this if you won't be using R. Place the following line in a file; we'll call it *installRpkgs.R*, push it to the master, and invoke R and run *installRpkgs.R* on each node.

```
echo "install.packages(c('Rmpi', 'foreach', 'doMC', 'doParallel',
   'doMPI'), repos = 'http://cran.cnr.berkeley.edu');
    print(paste('DONE with R package installation on ',
    system('hostname -s', intern = TRUE), '.')" > installRpkgs.R
starcluster put -u $scuser mycluster installRpkgs.R .
for node in $nodes; do
cmd="R CMD BATCH --no-save /home/$scuser/Rpkgs.R
        /home/$scuser/install.Rpkgs.log.$node"
starcluster sshmaster mycluster "ssh $node $cmd" &
done
```

Again, you can query the cluster to see if the installation is done for all packages on each node:

```
watch starcluster sshmaster mycluster "grep 'DONE'
   /home/$scuser/install.Rpkgs.log.*"
```

2.3 Creating your own image

First you need to start an instance and install all the software you want on it (as we've done above). My experience suggests that your instance should not have any EBS volumes attached (see Section 3.2). Then do:

```
starcluster ebsimage <instance id> <image name>
```

where <*instance id*> is the ID of the currently running instance on which you want the image to be based (you can obtain this via `starcluster listinstances`) and <*image name*> is the name you want to assign to the image. The *ebsimage* syntax says to create an image that can have persistent storage associated with it.

Running the above command will print out an AMI ID that you can use in place of the default value of NODE_IMAGE_ID in the *.starcluster/config* file.

You can also create an image that is backed by instance store (i.e., is not able to have persistent storage associated with it). More details on EBS-backed vs. instance store-backed (also called S3-backed) images is available at `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html#storage-for-the-root-device`.

3 Getting files to and from EC2 and disk usage

Note that StarCluster sets up the slave nodes so that they have access to */home* (in particular */home/$scuser*) via a mechanism called NFS (the same mechanism is used on the SCF file system). So you only need to copy files to your home directory on the master node and they'll be accessible on the slaves. However, there is limited space on */home*, so you may need to use */mnt* (this is the "instance store" mentioned on the EC2 webpage describing the instance types, with 160Gb for m1.small), which is not shared across nodes. You can of course choose a different instance type if you need more disk space.

Here are queries to assess how much disk space is available.

```
starcluster sshmaster mycluster "df -h /home/$scuser"
starcluster sshmaster mycluster "df -h /tmp"
starcluster sshmaster mycluster "df -h /mnt"
```

3.1 Transferring files

To transfer files to and from the cluster, you can do the following:

```
   starcluster put -u $scuser mycluster path/to/file/on/SCF/filename
path/to/file/on/node/.
   starcluster get -u $scuser path/to/file/on/node/filename
path/to/file/on/SCF/.
```

If you've copied to the master node to somewhere outside of */home* (e.g., */mnt/$scuser* below) and want to copy files to the other nodes from the master, you can do the following:

```
for node in $(eval echo $(seq -f node%03g 1 $(($numNodes-1)))); do
   starcluster sshmaster -u $scuser mycluster "scp /mnt/$scuser/file
      $node:/mnt/$scuser/."
done
```

Hopefully this will save on data transfer costs, though I haven't investigated this.

3.2 Persistent disk space

The disk space available for your cluster is removed when the cluster is terminated, unless you attach an EBS volume to your cluster.

If you have an EBS volume (SCREMS users may have access to one through the grant, while other users could create a volume), you can attach it to your cluster as follows. Note that to see info about available volumes, you can click on *Volumes* under the AWS EC2 console webpage: https://console.aws.amazon.com/ec2. See below for information on creating volumes.

First, modify your StarCluster config file as follows. Uncomment the following lines and replace (if desired) *biodata* with whatever name you want to refer to the volume by. Replace vol-c999999 with Amazon's name for the EBS volume.

```
# [volume biodata]
# VOLUME_ID = vol-c999999
# MOUNT_PATH = /data
```

If you leave */data* as the MOUNT_PATH, the persistent disk space will be available at */data*. If you choose */home*, then */home* will be on the EBS volume, including your home directory, */home/$scuser*, which may or may not be desirable.

Now, in the cluster template area where your cluster is defined, uncomment the *VOLUMES* line and replace the default name with the name you chose for the volume above (by default *biodata* above).

The EBS volume will be automatically available to all the nodes. So you don't need to copy file between nodes. You can most easily access the files from the SCF by copying to and from the master.

Files changed and created in the volume will be accessible on the volume even after terminating the cluster. E.g., if you attach the volume to a new cluster, you'll see the files.

Creating volumes You can create a volume via the EC2 web interface by going to https://console.aws.amazon.com/ec2, clicking on *Volumes*, and then clicking on *Create Volume*.

You can also create volumes from the command line - see http://star.mit.edu/cluster/docs/0.93.3/manual/volumes.html. The basic syntax (here for a 20 Gb volume that we'll refer to as *data*) is

```
starcluster createvolume --name=data 20 us-west-2a
```

4 Running jobs

Note that once you use *starcluster sshmaster* or *starcluster sshnode* to ssh to an instance, you can ssh to another instance simply by using ssh and the name of the node, e.g. `ssh node001`.

4.1 Single node (shared memory) jobs

For the most part, the steps you need to follow are described in the SCF tutorial on shared memory parallel programming, also available on Chris Paciorek's website.

Suppose that you have an R code file, *testShared.R*, you want to run (the same basic approach should work for Matlab code or a shell script). Also suppose that the R code produces the *testShared.RData* file as output. We can run the job remotely and retrieve the results as follows:

```
starcluster put -u $scuser mycluster testShared.R .
starcluster sshmaster -u $scuser mycluster "R CMD BATCH --no-save
testShared.R testShared.Rout"
starcluster get -u $scuser mycluster testShared.Rout .
starcluster get -u $scuser mycluster testShared.RData .
```

4.2 Multiple node (distributed memory) jobs

For the most part, the steps you need to follow are described in the SCF tutorial in the SCF tutorial on distributed memory parallel programming, also available on Chris Paciorek's website.

First, you'll need a file that lists the names of the nodes on which the distributed job will be processed. You can automate creation of a hostfile (following our creation of the *$nodes* variable above), which I call *.hosts* here:

```
echo $nodes > .hosts; sed -i 's/ /\n/g' .hosts
```

To choose the number of processes to assign per node, you can do:

```
numProcsPerNode=2
for node in $nodes; do
    echo $node slots=$numProcsPerNode >> .hosts
done
```

What follows is an example for R, but analogous commands should work for non-R jobs.

Suppose you have a file of R code, *testDistrib.R*, that implements a distributed job, such as by using the Rmpi package. Assuming you set -np in the mpirun command below, you do not need to specify a specific number of slaves in the R file; just include the line `startMPIcluster()` in your R code file.

You'll need to push the R code file and the *.hosts* file to your master node.

```
starcluster put -u $scuser mycluster testDistrib.R .
starcluster put -u $scuser mycluster .hosts .
```

Now run the R job by invoking *mpirun* with the *.hosts* file so that Rmpi knows what nodes to distribute the work to. Set the -np flag based on how many cores you want to use, e.g., 4 here would be appropriate if you have two nodes with two cores each (you might choose to use fewer than the number of cores available per machine to allow for threaded calculations).

```
starcluster sshmaster -u $scuser mycluster "mpirun -hostfile .hosts
-np 4 R CMD BATCH --no-save testDistrib.R testDistrib.Rout"
```

Assuming that the R code in *testDistrib.R* writes output to *testDistrib.RData* (on the master only) and writes text to *testDistrib.Rlog,* we need to get these files back from the nodes.

```
starcluster get -u $scuser mycluster testDistrib.Rout .
starcluster get -u $scuser mycluster testDistrib.RData .
starcluster get -u $scuser mycluster testDistrib.Rlog .
```

Analogous commands should work for non-R jobs.

4.3 Threaded computations

Note that the installation above includes *openBLAS*, a threaded BLAS, as the default BLAS. Any code that uses the BLAS, including R linear algebra functionality, will use as many cores as possible on a given node to do the calculations. In some cases (e.g., small matrix problems), threaded BLAS calculations might actually take longer than unthreaded, and there are cases of conflicts between threaded BLAS and other software. So in some cases you may want to fix the number of threads used by threaded programs that use the *openMP* protocol, which includes *openBLAS*. You can do this by setting the OMP_NUM_THREADS environment variable, e.g., `export OMP_NUM_THREADS=1`, as described in more detail, in the links in the above subsections.

5 Terminating instances

You will be billed for the entire time an instance is running, regardless of whether you are running anything on it. So make sure to terminate the instance when you are done.

```
starcluster terminate -c mycluster
```

5.1 Automatically stopping a cluster when your job is done

If you want to do this automatically, you could write a shell script that submits your job via *starcluster sshmaster*, copies the output back and then terminates the instances. As an example with a single node (and using R in this example, though this should work more generally), put the following in a file, say, *testStop.sh* (and assuming you have a file of R code you want to run, *test.R*, that saves output in *test.RData*). The crux of the shell code below is to check if the output file, *test.RData*, exists on the SCF machine, as a test for whether the job has completed and written output files back to the SCF machine, in which case it should be safe to shut down the cluster.

```
# begin testStop.sh
starcluster put -u $scuser mycluster test.R . # copy R file to node
if [ -f test.RData ]; then # does test.RData exist already?
```

```
    # if so, move any existing file aside, since we'll test
    # for its existence to check that job completes ok
    # before deleting the instance
    mv test.RData test.RData-
fi
# run job in foreground, waiting until done
starcluster sshmaster -u $scuser mycluster "R CMD BATCH
     --no-save test.R test.Rout"
starcluster get -u $scuser mycluster test.Rout . # copy results back
starcluster get -u $scuser mycluster test.RData .
if [ -f test.RData ]; then # has key file been transferred back
    starcluster terminate -c mycluster # if so, delete instance
echo "test.RData successfully copied back from node(s);
         instances are being shut down."
   echo "test.RData successfully copied back from node(s);
      instances are being shut down." | mailx -s "job succeeded"
      $USER@stat.berkeley.edu # automatic email notification
else
    echo "test.RData not copied back from node(s);
              instances are NOT being shut down"
    echo "test.RData not copied back from node(s);
       instances are NOT being shut down." | mailx -s "job failed"
       $USER@stat.berkeley.edu # automatic email notification
fi
# end testStop.sh
```

Now do

```
chmod u+x testStop.sh
./testStop.sh >& job.out &
```

You can then monitor *job.out* to see the progress of your job. Note that *testStop.sh* runs in the background (note the & at the end of the line above) so you can log out of the SCF machine. But the contents of *testStop.sh* run “in the foreground” in the sense that they run sequentially, so the copying of results back should only occur after the computation has finished.

5.2 Stopping and starting an EBS-backed cluster

Alternatively, if the cluster uses EBS instances, you can use the *stop* command to suspend all nodes and put them into a 'stopped' state, preserving the EBS volumes backing the nodes:

```
starcluster stop mycluster
```

WARNING: Any data stored in ephemeral storage (usually */mnt*) will be lost!

You can activate a 'stopped' cluster by passing the -x option to the *start* command:

```
starcluster start -x mycluster
```

This will restart the suspended nodes and reconfigure the cluster.