Setup batchtools SSH cluster with Boto and Paramiko

I. Make AMI

The AMI is build on Ubuntu 16.04, and has R, sampoll/batchtools and NFS installed.

sampoll/batchtools is different from batchtools only in one line in Worker.R. The worker calls `system.file('/bin/linux-helper', 'batchtools')` on the remote node to find the location of the shell script that executes the actually commands. But in the multiple quoting something is not working. (`/bin/linux-helper` is quoted, the code argument to `Rscript` is quoted, then the entire command is quoted for remote execution.) For this particular case it seemed reasonable to just assume the directory structure is the same on the remote machine and the local (head) node. In general, it is the same, because both nodes are instantiated from the same AMI.

To install from github, we need package `devtools`, and `devtools` needs some special SSL packages.

AMI: ami-1480e86e

```
# Build AMI
sudo apt-get update
sudo apt install r-base-core
sudo apt install nfs-kernel-server
sudo apt-get install libcurl4-openssl-dev
sudo apt-get install libssl-dev  # devtools suggests this
sudo R
> install.packages("devtools", repos="http://lib.stat.cmu.edu/R/CRAN")
> install_github("sampoll/batchtools")
```

II. Create cluster with run1.py

The python script run1.py creates a security group which allows (1) SSH from anywhere and (2) NFS connection among all group members. Then it creates two EC2 instances and waits for them to come up.

```
import boto3
from botocore.exceptions import ClientError
import pickle

def create_cluster():
```

```
6
7   """ Cluster configuration is hard-coded into script. Eventually,
8       it should be read from a configuration file. """
9
10  cluster = {
11    'name'  : 'cluster000',
12    'vpc'   : 'vpc-xxxxxxxx',     # usually the user's default VPC
13    'nnode' : 2,
14    'key'   : 'XXXXXXXX',         # private key is assumed to be in working directory
15    'ami'   : 'ami-1480e86e',
16    'type'  : 't2.micro',
17  }
18  (sgid,sgn) = create_security_group(cluster)
19  cluster['sgid'] = sgid
20  cluster['sgname'] = sgn
21  ids = create_instances(cluster)
22  cluster['iids'] = ids
23  return cluster
24
25 def save_cluster(cluster):
26
27  """ Pickle cluster for run2.py """
28
29  file = open('cluster.obj', 'wb')
30  pickle.dump(cluster, file)
31  file.close()
32
33 def create_security_group(cluster):
34
35  """ Create a security group for the cluster.
36      Open port 2049 (NFS) for communication within the cluster
37      and port 22 (SSH) to connection from anywhere
38
39      Returns: hash with group name and id"""
40
41  sgn = cluster['name'] + '-sg'
42  client = boto3.client('ec2')
43
44  # If group already exists, delete it. (Note: this will fail if there
45  # is an instance using it.) clusters should be created with
46  # different names.
47
48  sgexists = True
49  try:
50    client.describe_security_groups(GroupNames=[sgn])
51  except ClientError as e:
52    sgexists = False
53  if sgexists:
54    client.delete_security_group(GroupName=sgn)
55
56  security_group_dict = client.create_security_group(
```

```
    Description='Security Group for SSH Cluster',
    GroupName=sgn, VpcId=cluster['vpc'])

  sgid = security_group_dict['GroupId']
  sg = boto3.resource('ec2').SecurityGroup(sgid)
  p1 = { 'FromPort' : 22, 'ToPort' : 22, 'IpProtocol' : 'tcp' ,
      'IpRanges' : [ { 'CidrIp' : '0.0.0.0/0', 'Description' : 'Anywhere' } ] }
  p2 = { 'FromPort' : 2049, 'ToPort' : 2049, 'IpProtocol' : 'tcp' ,
      'UserIdGroupPairs' : [{ 'GroupId' : sgid } ] }
  res_auth = sg.authorize_ingress(IpPermissions=[p1, p2])
  return (sgid, sgn)

def create_instances(cluster):

  """ Create instances and wait for them to come up.
      Returns: list of Ids """

  n = cluster['nnode']
  res = boto3.client('ec2').run_instances(ImageId=cluster['ami'],
InstanceType=cluster['type'],
    KeyName=cluster['key'], MinCount=n, MaxCount=n,SecurityGroupIds=[cluster['sgid']])

  ids = [ inst['InstanceId'] for inst in res['Instances'] ]
  for id in ids:
    boto3.resource('ec2').Instance(id).wait_until_running()
  return ids


if __name__ == "__main__":
  cluster = create_cluster()
  save_cluster(cluster)
```

III. Configure cluster with run2.py

It is possible that run1.py and run2.py could be united into one program. The reason they are separate is that the instance returns true for `wait_until_running()` before the status checks are done. To be safe, I have been waiting until the status checks are complete in the EC2 console.

```
# run2.py
import boto3
from paramiko import client as pclient
import sys
import pickle

def init_cluster(cluster):

  """ Get the public and private IPs for each node.
      Arbitrarily designate one node the head node. """

  filters=[ {'Name': 'instance-id', 'Values': cluster['iids'] } ]
```

```
  res = boto3.client('ec2').describe_instances(Filters=filters)
  ips = ipaddresses(res)
  cluster['head'] = { 'public' : ips[0][1], 'private' : ips[0][0] , 'sg' : ips[0][2] }
  cluster['compute'] = [ { 'public' : ii[1], 'private' : ii[0] , 'sg' : ii[2] } for ii in
ips[1:] ]

# extract the IP addresses from the return structure
def ipaddresses(rrr):
  r = []
  for rr in rrr['Reservations']:
    instances = rr['Instances']
    for inst in instances:
      pvt = inst['PrivateIpAddress']
      pub = inst['PublicIpAddress']
      sg = inst['SecurityGroups'][0]['GroupId']
      r.append( (pvt,pub, sg) )
  return r

# NFS export config to be added to /etc/exports on the head node
def write_exports(compute):
  file = open("exports", "w")
  for c in compute:
    ss = ("/scratch %s(rw,sync,no_root_squash,no_subtree_check)\n" % (c['private']))
    file.write(ss)
  file.close()

# NFS mount config to be added to /etc/fstab on the compute nodes
def write_fstab(head):
  file = open("fstab", "w")
  ss = ("%s:/scratch /scratch nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0\n" %
head['private'])
  file.write(ss)

# batchtools.conf.R for SSH requires IP addresses of compute nodes
def write_batchtools_config(compute):
  file = open("batchtools.conf.R", "w")
  ss = 'workers = list('
  for c in compute:
    ss = ss + 'Worker$new("' + c['private'] + '", ncpus=1)'
    if c != compute[-1]:
      ss = ss + ','
  ss = ss + ')\n'
  file.write(ss)
  file.write('cluster.functions = makeClusterFunctionsSSH(workers)\n')
  file.close()

# Execute SSH command remotely
def exssh(cl, cmd):
  print(cmd)
  (stdin, stdout, stderr) = cl.exec_command(cmd)
  try:
```

```
    ex = stdout.channel.recv_exit_status()
    if ex != 0:
      print('  !!! Warning: ssh returned non-zero exit status %d ' % (ex))
  except SSHException:
    print('SSHException %s' % (cmd))

def setup_cluster(cluster):

  """ Configure the cluster for use with batchtools SSH option:

    1. Set up passwordless SSH from head node to compute nodes
    2. Setup up NFS share directory /scratch on all nodes
    3. Configure /etc/exports on host and /etc/fstab on compute nodes
       for NFS sharing of /scratch
    4. Upload batchtools.conf.R to NFS shared directory

  """

  # write IP address-dependent file text in working directory for sftp to cluster nodes
  write_exports(cluster['compute'])
  write_fstab(cluster['head'])
  write_batchtools_config(cluster['compute'])

  # Initiate SSH and SFTP connection to head node
  headclient = pclient.SSHClient()
  headclient.set_missing_host_key_policy(pclient.AutoAddPolicy())
  hip = cluster['head']['public']
  kfn = cluster['key'] + '.pem'
  headclient.connect(hip, username='ubuntu', key_filename=kfn)
  headfclient = headclient.open_sftp()

  # Generate key pair and fetch the public key
  exssh(headclient, 'sudo rm -f /home/ubuntu/.ssh/id_rsa*')
  exssh(headclient, 'sudo ssh-keygen -t rsa -f /home/ubuntu/.ssh/id_rsa -q -N ""')
  exssh(headclient, 'sudo chmod 0400 /home/ubuntu/.ssh/id_rsa')
  exssh(headclient, 'sudo chown ubuntu:ubuntu /home/ubuntu/.ssh/id_rsa')
  headfclient.get('/home/ubuntu/.ssh/id_rsa.pub', 'id_rsa.pub')

  # Set up /etc/exports on head node
  headfclient.put('exports', '/home/ubuntu/exports')
  exssh(headclient, 'sudo cat /etc/exports /home/ubuntu/exports >/home/ubuntu/tmp')
  exssh(headclient, 'sudo mv /home/ubuntu/tmp /etc/exports')
  exssh(headclient, 'sudo rm -f /home/ubuntu/exports')
  exssh(headclient, 'sudo rm -f /home/ubuntu/tmp')

  # Make /scratch directory on head node
  exssh(headclient, 'sudo mkdir -p /scratch')
  exssh(headclient, 'sudo chown nobody:nogroup /scratch')
  exssh(headclient, 'sudo chmod -R 777 /scratch')

  # Push batchtools.conf.R to /scratch
```

```
  headfclient.put('batchtools.conf.R', '/scratch/batchtools.conf.R')

  # Restart NFS server on head node
  exssh(headclient, 'sudo systemctl restart nfs-kernel-server')

  # delete allocated ssh structures
  headfclient.close()
  headclient.close()

  for c in cluster['compute']:

    # Initiate SSH and SFTP connection to compute node
    cip = c['public']
    nodeclient = pclient.SSHClient()
    nodeclient.set_missing_host_key_policy(pclient.AutoAddPolicy())
    nodeclient.connect(cip, username='ubuntu', key_filename=kfn)
    nodefclient = nodeclient.open_sftp()

    # Propagate public key to compute node
    nodefclient.put('id_rsa.pub', '/home/ubuntu/.ssh/id_rsa.pub')
    exssh(nodeclient, 'sudo cat /home/ubuntu/.ssh/id_rsa.pub
>>/home/ubuntu/.ssh/authorized_keys')
    exssh(nodeclient, 'sudo rm /home/ubuntu/ssh/id_rsa.pub')

    # Make /scratch directory on compute nodes
    exssh(nodeclient, 'sudo mkdir -p /scratch')
    exssh(nodeclient, 'sudo chmod -R 777 /scratch')

    # Append mount to /etc/fstab
    nodefclient.put('fstab', '/home/ubuntu/fstab')
    exssh(nodeclient, 'sudo cat /etc/fstab /home/ubuntu/fstab >/home/ubuntu/tmp')
    exssh(nodeclient, 'sudo mv /home/ubuntu/tmp /etc/fstab')
    exssh(nodeclient, 'sudo rm -f /home/ubuntu/fstab')
    exssh(nodeclient, 'sudo rm -f /home/ubuntu/tmp')

    # Mount NFS export (Note: /etc/fstab is for automatic mounting on reboot)
    exssh(nodeclient, 'sudo mount ' + cluster['head']['private'] + ':/scratch /scratch')
    nodefclient.close()
    nodeclient.close()

# Load data from pickle made by run1.py
def load_cluster():
  file = open('cluster.obj', 'rb')
  cluster = pickle.load(file)
  file.close()
  return cluster

# Pickle cluster for del.py
def save_cluster(cluster):
  file = open('cluster.obj', 'wb')
  pickle.dump(cluster, file)
```

```
  file.close()

if __name__ == '__main__':
  cluster = load_cluster()
  init_cluster(cluster)
  setup_cluster(cluster)
  save_cluster(cluster)
```

IV. Run tiny example on cluster

ssh to head node

```
cd /scratch
R
R> piApprox = function(n)  {
+   nums = matrix(runif(2 * n), ncol = 2)
+   d = sqrt(nums[,1]^2 + nums[,2]^2)
+   4 * mean(d <= 1)
+ }
R> library(batchtools)
R> reg <- makeRegistry()      # should indicate registry made with SSH cluster functions
R> batchMap(fun = piApprox, n = rep(1e6, 3))
R> submitJobs()
R> waitForJobs()
R> reduceResults(function(x,y) x+y)/3     # 3.1398>
```

V. Delete cluster

Run delete.py

```
# delete.py
import boto3
import pickle

# load cluater data from pickle
file = open('cluster.obj', 'rb')
cluster = pickle.load(file)
file.close()

# delete instances and wait for them to terminate
client = boto3.client('ec2')
res = client.terminate_instances(InstanceIds=cluster['iids'])

# once instances are terminated, security group can be deleted
for id in cluster['iids']:
  boto3.resource('ec2').Instance(id).wait_until_terminated()
client.delete_security_group(GroupName=cluster['sgname'])
```