Cloud Computing with HTCondor

Background

At the University of Liverpool, the Advanced Research Computing team support a variety of research computing facilities including two HPC clusters and a high throughput HTCondor service. The clusters are aimed primarily at parallel MPI/shared memory applications, although a number of users run multiple serial codes as array jobs.

The HTCondor pool makes use of spare computing cycles on the University's teaching and learning centre PCs and can run up to around 1600-2000 jobs concurrently. The PCs comprising the pool all run the central computing service's Managed Windows Service (MWS) and typically have four cores and 16 GB of memory. Access to the pool is via a single central manager/submit host. The vast majority of applications which run on the HTCondor pool make use of either MATLAB or the R statistics language.

Although the HTCondor service has been extremely successful in supporting researchers across a wide variety of disciplines, the range of applications it can support is limited by a number of factors, notably:

- applications must run under Windows (currently version 10)
- third party software that user applications require is limited to software that is either preinstalled on the PCs (e.g. MATLAB) or software that can be installed easily on-the-fly (e.g. R)
- since the pool comprises only commodity PCs, resources are limited to around 4 GB of memory per job and around 2 GB of disk storage
- jobs can usually only run for a maximum of around 8-10 hours before being interrupted by ordinary logged-in users and there is a hard limit of 24 hours since PCs are rebooted daily (in practice run times of 20-30 minutes seem to work best)

Making use of AWS cloud instances to run jobs overcomes these limitations since:

- jobs are not subject to interruption (at least for dedicated instances) and can run pretty much indefinitely
- a huge variety of hardware specifications are available so applications are not limited by memory, compute or storage and can make use of non-traditional resources such as GPUs
- many different operating systems are available including Windows and various UNIX 'flavours'
- software can be pre-installed and saved within snapshots or can be installed on-the-fly with administrator/root permission if need be
- data protection is less of an issue as jobs run on isolated virtual machines rather than on shared PCs as in the Windows HTCondor pool

The main disadvantage is cost. Our MWS HTCondor pool users have clocked up around a million core-hours of combined usage in some months and this amount computing would be prohibitively expensive on AWS. However we do have users with a need to run modest numbers of jobs (say thousands) which cannot easily be accommodated on our current pool because of either hardware or software limitations. A key advantage of using the AWS cloud to provide an additional HTCondor pool is that users can easily switch to AWS without

having to change their way of working as the implementation details are mostly hidden from them.

In our AWS HTCondor pool, the existing HTCondor server is used as submit host by running a HTCondor scheduler on it. A central manager is created on the AWS cloud as well as dynamically provisioned execute hosts. This does make the manager a single point of failure but it is straightforward to configure a fail-over manager. In HTCondor, the central manager performs only light work and can be run on a fairly low specification instance.

There are two main hurdles to overcome with this configuration. First, there are network problems to resolve since the campus firewall does not permit incoming connections to the HTCondor scheduler (outgoing connections are however unrestricted) and the central manager and execute hosts run inside a single virtual private cloud (VPC) - see Fig. 1. Secondly there is the problem of how to provision/decommission execute hosts. We only want execute hosts to be active where there is a demand for them to avoid wasting money and instances should shut down automatically when they are no longer needed.

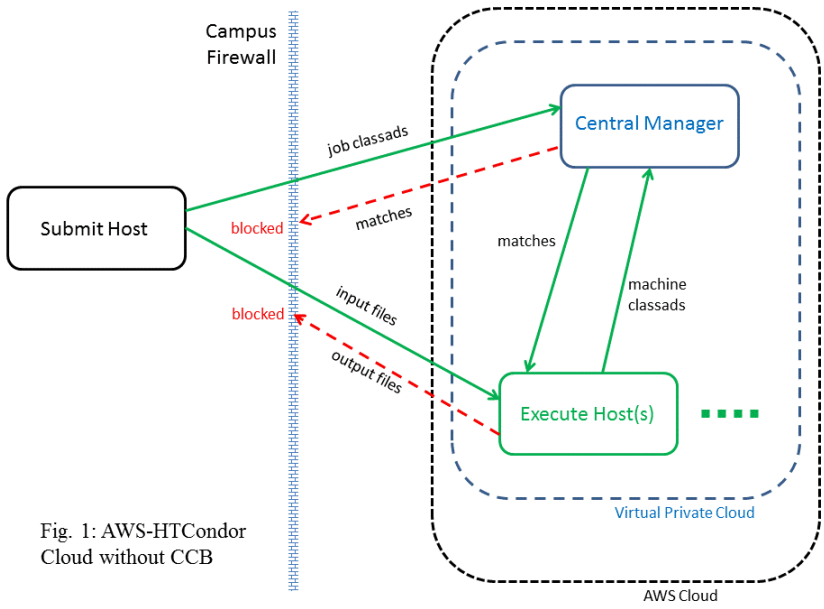

Fig. 1: AWS-HTCondor Cloud without CCB

HTCondor central manager/ connection broker

The central manager resides in a VPC along with the execute hosts and has a fixed (elastic) public IP address and dynamically assigned private IP address. A Condor Connection Broker (CCB) is used by the central manager to effectively reverse the direction in which connections are made to the submit host (see Fig. 2). This ensures that all communication between submit host <-> central manager and submit host <-> execute host originates on the submit host and can pass through the campus firewall. Traffic can flow in the opposite direction since the firewall recognises that this is part of the same duplex connection instantiated by the submit host (think of contacting a web server - the browser makes an outgoing connection through the firewall to the server in the outside world which then sends data back to the browser through the firewall).

Incoming connections to the central manager from the submit host are made using its public IP address whereas incoming connections from the execute hosts make use of the manager's private IP address. The central manager daemons attempt to communicate with

each other using the public IP address which is translated back to the private IP address using network address translation (NAT) so that this effectively becomes a loopback address (non-private addresses are routed to the VPC gateway by default).

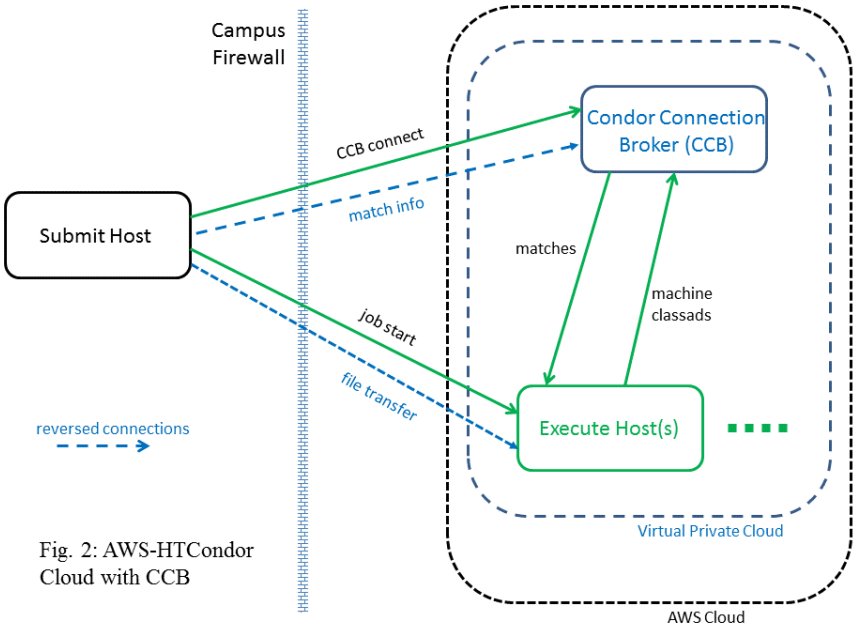

Fig. 2: AWS-HTCondor Cloud with CCB

Currently the central manager is implemented on a t2.small instance running the Amazon Linux operating system. This was first configured manually then snapshotted and the resulting Amazon machine image (AMI) saved for future use. Configuration commands were saved in a script to allow the process to be automated later.

The first stage of the configuration process is to gather all of the required information from the AWS instance metadata - this includes its public and private IP addresses and subnet address. The latest HTCondor release is then installed after being downloaded from the University of Wisconsin HTCondor website. A previously created condor configuration file is copied into the appropriate location as well as a local configuration file (`condor_config.local`) created using the previously obtained metadata and the "hard coded" hostname of HTCondor submit host. Settings in local configuration file take precedence over the main configuration file and are used for parameters that are subject to change (e.g. network addresses)

Next, the network is configured using `iptables` to give a loopback for the HTCondor daemons using NAT. This essentially completes the configuration process however it is necessary to ensure that the AMI will be configured with the correct network addresses when launched/rebooted as the private IP and subnet addresses are subject to change. This is achieved by installing a small configuration script as one of the Linux startup files (i.e. those in `/etc/init.d`). The configuration process is now complete and the instance can be snapshotted and saved for future use.

To create another central manager it not necessary to manually repeat the process. A script can be provided as AWS User Data when launching an instance which automatically configures the instance. This fetches the required HTCondor configuration files, the full configuration script and the startup reconfiguration script from our HTCondor web server.

Thus it is not necessary to go through the whole manual setup process again to provision a new central manager.

HTCondor execute hosts

The execute hosts are created using a similar process to the central manager. Again the Amazon Linux operating system was employed and a t2.small instance used to begin with. Note that the underlying hardware can be changed later to give more powerful execute hosts once we have an execute host AMI created and saved.

The first step in the manual configuration is again to gather the required host information (e.g. network addresses) as AWS instance metadata. The latest version of HTCondor is then downloaded from the University of Wisconsin HTCondor website and installed. A previously set up condor configuration file is copied into the appropriate location and and a local configuration file (`condor_config.local`) created using the previously obtained metadata and the "hard coded" hostname of HTCondor submit host.

The execute hosts are configured to always run jobs and never suspend/kill them (since the only load on the instance is from HTCondor jobs) or preempt them (as checkpointing is not possible in a CCB environment). Connections to the central manager are made using it's private IP address which is currently "hard coded" into the condor configuration. It should be possible in future to provide a failover central manager as well so this is not a single point of failure.

The next step is to configure the network using `iptables` to give a loopback for HTCondor daemons using NAT. The execute hosts are configured to power off automatically when not in use to avoid wasting money (this also requires that instances are launched with the shutdown behaviour set to 'terminate' to remove them from the cloud). This is achieved by installing a `cron` script which periodically checks whether there is any activity on the instance. The shutdown condition is fairly strict and requires that the execute host is in the "Unclaimed" state according to HTCondor, there has been no HTCondor activity for an hour, there are no logged in users and the (15 minute) load average is less than 0.1. The inactivity period has been chosen to make debugging easier but could easily be reduced in future to avoid wasted uptime where there are numerous execute hosts.

This essentially completes the configuration process however it necessary to ensure that the instance will be correctly configured with the correct network information when launched/rebooted as the IP addresses (public and private) and subnet addresses are subject to change on power up/reboot. This is achieved by installing a small configuration script as one of the Linux startup files (i.e. those in `/etc/init.d`).

As most of our existing HTCondor users make use of MATLAB and R, the MATLAB Runtime Component is installed (allowing precompiled MATLAB code to be run) and the latest version of R (plus most of the packages in our Windows version of R). This is done manually and does not form part of the automated configuration so that other execute hosts can be created without the rather bulky MATLAB runtime. However scripts can be used to install MATLAB and R plus the extra packages.

The configuration process is now complete and the instance can be snapshotted and saved for future use. To create another execute host from scratch it is not necessary to manually repeat the process. A script can be provided as AWS User Data when launching an instance which automatically configures the instance. This fetches the required condor configuration files, the full configuration script plus the crontab file and the startup reconfiguration script

from our HTCondor web server. Thus it is not necessary to go through the whole manual set up process again to provision a new execute host from scratch.

HTCondor submit host

The HTCondor scheduler for the AWS pool is installed on the MWS HTCondor submit host so there is a single access point to both the MWS and AWS pools and users can easily switch between them. The scheduler need only be configured with the IP address of the central manager (including the CCB). Rather than having a static pool of execute hosts which sit idle when not in use, execute hosts are spun up in response to users submitting jobs by the `condor_rooster` daemon. This was originally designed to wake up hibernating machines via wake-on-LAN and is used on our MWS HTCondor pool during periods when most of our classroom PCs are hibernating (e.g. during vacations). However it works equally well for spinning up AWS instances.

Offline machine classads are advertised via the central manager representing available execute hosts which the HTCondor negotiator matches against job classads and signals to `condor_rooster` that new execute hosts need to be created. These offline classads would normally represent real hibernating machines in a more typical wake-on-LAN setup. A limit is placed on the maximum number of execute hosts in the pool at one time so that we do not carry on spinning up instances ad infinitum. In practice there is a hard limit of around 250 hosts imposed by the number of addresses available in the VPC subnet.

User environment

The AWS command line interface whilst flexible is not something that most users would want to grapple with regularly and so we have hidden this behind a number of simple shell scripts e.g. `aws_start`, `aws_stop`, `aws_run`, `aws_spot_run`, `aws_terminate` and `aws_list`. These are mainly to be used by administrators or in other scripts and ordinary users will generally work with the AWS HTCondor pool using commands that are the equivalent of there ordinary MWS HTCondor pool commands e.g. `aws_submit`, `aws_status`, `aws_q`, `aws_rm` etc instead of `condor_submit`, `condor_status`, `condor_q`, `condor_rm` etc.

Our MWS HTCondor users make extensive use of simplified local job submission files built on top of `condor_submit` which hide most of the low level details. Thus we have `r_submit` for R based applications and `matlab_submit` for MATLAB applications. A typical job submission file might look like:

```
R_script = analysis.R
indexed_input_files = dataset_A.dat, dataset_B.dat
input_files = common_data.dat
indexed_output_files = output.dat
total_jobs = 1000
```

(indexed files are those that are different for each job task/job process having names that are parameterised with a task ID index). For the AWS HTCondor pool the equivalent commands are `aws_r_submit` and `aws_matlab_submit` and exactly the same job submission file can be used so that the AWS implementation details are *completely transparent* to the user.

An example application

A researcher had an urgent need to perform 100,000 simulations in order to address the requests of reviewers in an important paper on the environmental impacts in Britain on the spread of bluetongue disease in ruminants over 2006-2007. In order to work, a large number of possible parameter sets and scenarios had to be investigated, running a a minimum of 100 simulations each time. The model was computationally intensive and so required a significant amount of computing time. Running the 100 jobs (with 1000 simulations each) via the HTCondor interface on the AWS spot market took 7h 21m at a cost of just $51.16. Running the job serially would have taken 98 days.

Ian C. Smith
Advanced Research Computing
University of Liverpool
October 2018

Installation and Configuration Details

Central Manager/CCB

Full installation script

```
#!/bin/bash

#
# get various instance data from AWS metadata server
#

local_ipv4_address=`curl http://169.254.169.254/latest/meta-data/local-ipv4
2>/dev/null`
local_hostname=`curl http://169.254.169.254/latest/meta-data/hostname
2>/dev/null`
public_ipv4_address=`curl http://169.254.169.254/latest/meta-data/public-
ipv4 2>/dev/null`
public_hostname=`curl http://169.254.169.254/latest/meta-data/public-
hostname 2>/dev/null`
mac=`curl http://169.254.169.254/latest/meta-data/network/interfaces/macs/
2>/dev/null`
cidr_block=`curl http://169.254.169.254/latest/meta-
data/network/interfaces/macs/${mac}subnet-ipv4-cidr-block 2>/dev/null`

#
# get rid of all aliases - don't want things like cp -i causing prompts
#

unalias -a

#
# add the default condor user
#

groupadd condor
useradd -g condor condor

#
# set up Condor installation repo
#

cd /etc/yum.repos.d/
yum -y install wget
wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-
rhel7.repo
wget http://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor
rpm --import RPM-GPG-KEY-HTCondor

#
# install latest version of Condor
#

yum -y install condor.x86_64

#
# set up the Condor config files
#
```

```
cp -f ~ec2-user/condor_config /etc/condor/condor_config
cat <<END >/etc/condor/condor_config.local
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = $public_ipv4_address
MY_PUBLIC_FQDN = $public_hostname
MY_SUBNET = $cidr_block
END

#
# set up the NAT using iptables, clear out existing rules (chains first)
#

yum -y install iptables-services

iptables -F
iptables -X
iptables -F -t nat
iptables -X -t nat

iptables -t nat -A OUTPUT -d $public_ipv4_address -j DNAT --to-destination
$local_ipv4_address

service iptables save

#
# start and enable iptables and Condor services
#

systemctl start iptables
systemctl enable iptables
systemctl start condor
systemctl enable condor

#
# Copy the startup into /etc/init.d to ensure it runs on each reboot
#

cp ~ec2-user/startup_configure /etc/init.d/condor_configure
ln -s ../init.d/condor_configure /etc/rc2.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc3.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc4.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc5.d/S11condor_configure

yum -y install condor.x86_64
condor_config (part)
condor_config.local
```

Startup configuration script

```
#!/bin/bash

local_ipv4_address=`curl http://169.254.169.254/latest/meta-data/local-ipv4
2>/dev/null`
public_ipv4_address=`curl http://169.254.169.254/latest/meta-data/public-
ipv4 2>/dev/null`
public_hostname=`curl http://169.254.169.254/latest/meta-data/public-
hostname 2>/dev/null`
mac=`curl http://169.254.169.254/latest/meta-data/network/interfaces/macs/
2>/dev/null`
```

```
cidr_block=`curl http://169.254.169.254/latest/meta-
data/network/interfaces/macs/$mac/subnet-ipv4-cidr-block 2>/dev/null`
#
# Adverstise this host using its public IP/hostname via Condor
#
cat <<END >/etc/condor/condor_config.local
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = $public_ipv4_address
MY_PUBLIC_FQDN = $public_hostname
MY_SUBNET = $cidr_block
END
#
# Set up the NAT using iptables to give "loopback" on public IP for Condor.
# Flush existing rules first with -X, -F.
#
iptables -X -t nat
iptables -F -t nat
iptables -t nat -A OUTPUT -d $public_ipv4_address -j DNAT --to-destination
$local_ipv4_address
service iptables save
#
# Restart Condor to make sure we purge any old settings
#
systemctl stop condor
systemctl start condor
```

AWS User Data

```
#!/bin/bash

echo "******************** THIS IS THE STARTUP INSTALLER ************ "
yum -y install wget
cd ~ec2-user
wget http://condor.liv.ac.uk/aws/central_manager/full_installation
wget http://condor.liv.ac.uk/aws/central_manager/startup_configure
wget http://condor.liv.ac.uk/aws/central_manager/condor_config
chmod a+rx full_installation startup_configure
chown ec2-user full_installation startup_configure condor_config
~ec2-user/full_installation
echo "******************** STARTUP INSTALLER FINISHED  ************ "
```

condor_config (excerpt)

```
CONDOR_HOST = $(FULL_HOSTNAME)
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD
PRIVATE_NETWORK_NAME = aws.amazon.com
TCP_FORWARDING_HOST = $(MY_PUBLIC_IP)
NETWORK_HOST = $(MY_PUBLIC_FQDN)
```

condor_config.local (example)

```
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = 34.240.27.31
MY_PUBLIC_FQDN = ec2-34-240-27-31.eu-west-1.compute.amazonaws.com
MY_SUBNET = 173.31.3.0/24
```

Execute Hosts

Full installation script

```
#!/bin/bash

#
# get various instance data from AWS metadata server
#

condor_manager_private_ip="173.31.3.111"
local_ipv4_address=`curl http://169.254.169.254/latest/meta-data/local-ipv4
2>/dev/null`
local_hostname=`curl http://169.254.169.254/latest/meta-data/hostname
2>/dev/null`
public_ipv4_address=`curl http://169.254.169.254/latest/meta-data/public-
ipv4 2>/dev/null`
public_hostname=`curl http://169.254.169.254/latest/meta-data/public-
hostname 2>/dev/null`


#
# get rid of all aliases - don't want things like cp -i causing prompts
#

unalias -a

#
# add the default condor user
#

groupadd condor
useradd -g condor condor

#
# set up Condor installation repo
#

cd /etc/yum.repos.d/
yum -y install wget
wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-
rhel7.repo
wget http://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor
rpm --import RPM-GPG-KEY-HTCondor

#
# install latest version of Condor
#

yum -y install condor.x86_64

#
# set up the Condor config files
#
yum -y install condor.x86_64

#
```

```
# set up the Condor config files
#

cp -f ~ec2-user/condor_config /etc/condor/condor_config
cat <<END >/etc/condor/condor_config.local
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = $public_ipv4_address
CENTRAL_MANAGER = $condor_manager_private_ip
PUBLIC_HOSTNAME = $public_hostname
END

#
# set up the NAT using iptables, clear out existing rules (chains first)
#

yum -y install iptables-services

iptables -F
iptables -X
iptables -F -t nat
iptables -X -t nat

yum -y install iptables-services
iptables -t nat -A OUTPUT -d $public_ipv4_address -j DNAT --to-destination
$local_ipv4_address
service iptables save

#
# start and enable iptables and Condor services
#

systemctl start iptables
systemctl enable iptables
systemctl start condor
systemctl enable condor

#
# Copy the startup into /etc/init.d to ensure it runs on each reboot
#

cp ~ec2-user/startup_configure /etc/init.d/condor_configure
ln -s ../init.d/condor_configure /etc/rc2.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc3.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc4.d/S11condor_configure
ln -s ../init.d/condor_configure /etc/rc5.d/S11condor_configure

# Add (small) applications

yum -y install gcc-gfortran

#
# set up the usage monitor crontab
#

cp ~ec2-user/monitor_usage.sh /usr/local/bin
crontab -l > /tmp/cronfile
echo '0,5,10,15,20,25,30,35,40,45,50,55 * * * *
/usr/local/bin/monitor_usage.sh 2>&1 > /tmp/usage_monitor.log' >>
/tmp/cronfile
crontab /tmp/cronfile
```

Startup configuration script

```
#!/bin/bash

condor_manager_private_ip="173.31.3.111"
local_ipv4_address=`curl http://169.254.169.254/latest/meta-data/local-ipv4
2>/dev/null`
public_ipv4_address=`curl http://169.254.169.254/latest/meta-data/public-
ipv4 2>/dev/null`
public_hostname=`curl http://169.254.169.254/latest/meta-data/public-
hostname 2>/dev/null`
#
# Adverstise this host using its public IP/hostname via Condor
#
cat <<END >/etc/condor/condor_config.local
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = $public_ipv4_address
CENTRAL_MANAGER = $condor_manager_private_ip
PUBLIC_HOSTNAME = $public_hostname
END
#
# Set up the NAT using iptables to give "loopback" on public IP for Condor.
# Flush existing rules first with -X, -F.
#
iptables -X -t nat
iptables -F -t nat
iptables -t nat -A OUTPUT -d $public_ipv4_address -j DNAT --to-destination
$local_ipv4_address
service iptables save
#
# Restart Condor so that none of the old settings are still there
service iptables save
systemctl stop condor
systemctl start condor
```

condor_config (excerpt)

```
CONDOR_HOST = $(CENTRAL_MANAGER)
DAEMON_LIST = MASTER, STARTD
PRIVATE_NETWORK_NAME = aws.amazon.com
TCP_FORWARDING_HOST = $(MY_PUBLIC_IP)
NETWORK_HOSTNAME = $(PUBLIC_HOSTNAME)
```

condor_config.local (example)

```
ARC_CONDOR_HOST = condor.liv.ac.uk
MY_PUBLIC_IP = 34.240.240.58
CENTRAL_MANAGER = 173.31.3.111
PUBLIC_HOSTNAME = ec2-34-240-240-58.eu-west-1.compute.amazonaws.com
```

Auto poweroff cron script

```
#!/bin/bash

max_unclaimed_time=3600

public_hostname=`condor_config_val PUBLIC_HOSTNAME`
```

```
usage_info=`condor_status -direct $public_hostname -af State
EnteredCurrentState MyCurrentTime`

state=`echo $usage_info | cut -d' ' -f 1`
entered_state=`echo $usage_info | cut -d' ' -f 2`
time_now=`echo $usage_info | cut -d' ' -f 3`

activity_time=`expr $time_now - $entered_state`
load_average=`cat /proc/loadavg | cut -d' ' -f 1`
no_of_users=`users | wc -l`


if [ $state == "Unclaimed" ]  && [[ $activity_time -gt
$max_unclaimed_time ]] && \
   [[ $load_average < 0.1 ]] && [[ $no_of_users -eq  0 ]]
then
  date
  echo "the load average is $load_average"
  echo "there are $no_of_users logged in users"
  echo "We are in Unclaimed for $activity_time seconds"
  echo "shutting down because of inactivity..."
  /sbin/poweroff
fi
```

AWS User Data

```
#!/bin/bash
echo "******************** THIS IS THE STARTUP INSTALLER ************ "
yum -y install wget
cd ~ec2-user
wget http://condor.liv.ac.uk/aws/execute_host/full_installation
wget http://condor.liv.ac.uk/aws/execute_host/startup_configure
wget http://condor.liv.ac.uk/aws/execute_host/condor_config
wget http://condor.liv.ac.uk/aws/execute_host/monitor_usage.sh
chmod a+rx full_installation startup_configure
chown ec2-user full_installation startup_configure condor_config
~ec2-user/full_installation
echo "******************** STARTUP INSTALLER FINISHED  ************ "
```

Submit Host

condor_config (excerpt)

```
CONDOR_HOST = 34.240.27.31
DAEMON_LIST = MASTER SCHEDD ROOSTER
ROOSTER_INTERVAL = 300
USE_SHARED_PORT = FALSE
SCHEDD_NAME = AWS@$(FULL_HOSTNAME)
CCB_ADDRESS = $(CONDOR_HOST)
PRIVATE_NETWORK_NAME = condor.liv.ac.uk
ALLOW_WRITE =  ulgp5.liv.ac.uk, $(CONDOR_HOST)
```