Building IaaS infrastructures on the AWS Cloud

Saul Pierotti

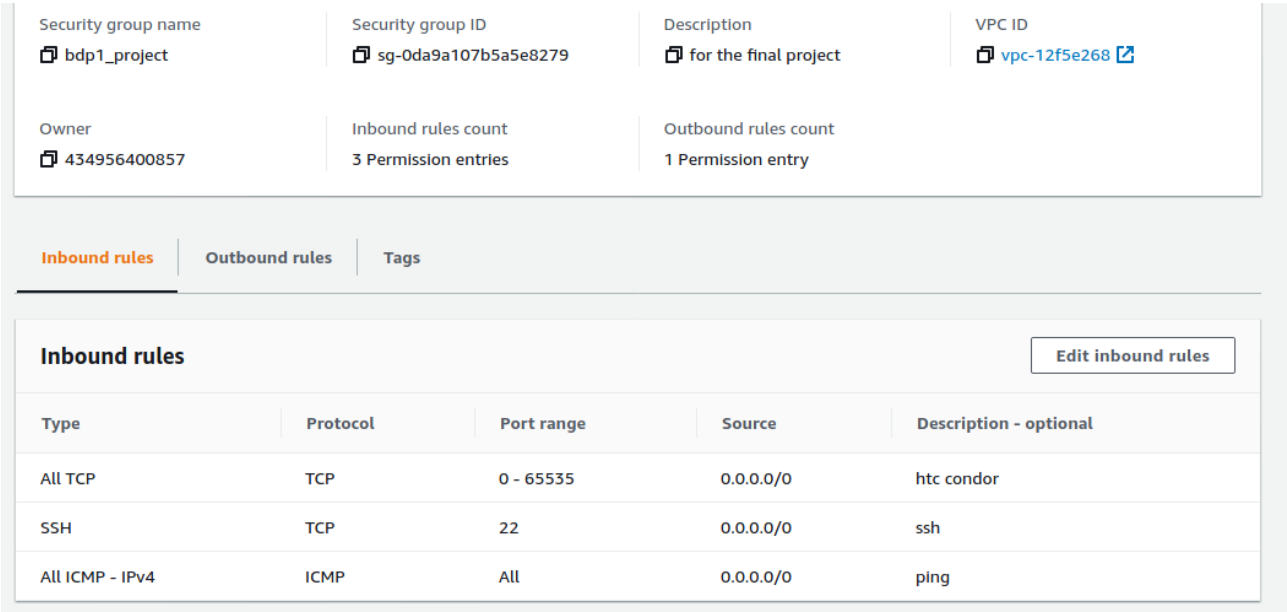
July 25, 2020

1 Creation of an HTCondor Cluster for the Alignment of NGS Reads

The demonstrative infrastructure described in this project consists of an HTCondor cluster of three nodes. One node is configured as Master Node, while two nodes are configured as Worker Nodes. The infrastructure can be easily expanded by replicating the Worker Node instances. The Master Node was not used also as a Worker Node since the performance benefits and cost savings would be marginal. This was done also to avoid overloading the Master Node, on which the entire cluster depends. A shared storage space directly attached to the Master Node but available to all the Worker Nodes was also implemented using the distributed file system NFS.

1.1 Initialization of the Instances on the AWS Cloud

Worker Nodes and the Master Node were both built on similar machines. For the Master Node, the `t2.medium` instance type was used with a 50 Gb SSD as root storage. For the Worker Nodes, the `t2.large` instance type was used with a 50 Gb SSD as root storage. The operating system chosen for both machine types is Ubuntu Server 18.04.4 LTS. The Master Node and the Worker Nodes were all instantiated in the same availability zone (`us-east-1a`) so that they would be able to communicate through private IPv4 addresses. The security group for the instances was configured as follows:



All the TCP ports were opened to the other members of the same security group since HTCondor daemons use a dynamically assigned port. The TCP ports were also needed for the setup of a shared NFS volume. ICMP ports were opened for accepting incoming `ping` requests for testing purposes. TCP port 22 was opened for allowing remote control of the machines via `ssh`.

1.2 Configuration of the Master Node

The PS1 prompt of the Master Node was changed so to make the node easily identifiable from the command line.

HTCondor was then installed with the following commands:

```
ubuntu@bdp1-master-node:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\
]\u@bdp1-master-node\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

```
sudo su
wget -qO - https://research.cs.wisc.edu/htcondor/ubuntu/HTCondor-Release.gpg.key | apt-key add - #
    import the gpg key of HTCondor
echo "deb http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" >> /etc/apt/sources.
    list # add the repository
echo "deb-src http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" >> /etc/apt/
    sources.list
apt update
apt install htcondor
systemctl start condor # start and enable the condor service
systemctl enable condor
```

The correct proceeding of the installation and the start of the `condor` service where checked with the following commands:

```
ubuntu@bdp1-master-node:~$ sudo systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-06-16 10:31:25 UTC; 1min 16s ago
 Main PID: 15225 (condor_master)
   Status: "All daemons are responding"
    Tasks: 3 (limit: 32767)
   CGroup: /system.slice/condor.service
           ├─15225 /usr/sbin/condor_master -f
           ├─15266 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 111
           └─15268 condor_shared_port -f

Jun 16 10:31:25 ip-172-31-8-109 systemd[1]: Started Condor Distributed High-Throughput-Computing.
ubuntu@bdp1-master-node:~$ ps ax | grep condor
15225 ?        Ss     0:00 /usr/sbin/condor_master -f
15266 ?        S      0:00 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 111
15268 ?        Ss     0:00 condor_shared_port -f
15286 pts/0    S+     0:00 grep --color=auto condor
ubuntu@bdp1-master-node:~$
```

The following lines where appended at the end of the main HTCondor configuration file, located at `/etc/condor/condor_config`:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>

# Master Node config
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD
```

Finally, the `condor` service was restarted with the following command:

```
sudo systemctl restart condor
```

The NFS server was then implemented in the Master Node. A new 100 Gb standard magnetic volume was created from the AWS interface and attached to the Master Node. Fom the server, a primary partiton was initialized on the volume using `fdisk` and an `Ext4` file system was created onto it using `mkfs.ext4`. The file `/etc/fstab` of the Master Node was modified so that the machine would mount the volume automatically at boot under the newly created directory `/data`. The following line was appended to `/etc/fstab`:

```
</new_volume/partiton>          /data    ext4   defaults             0 0
```

The following commands were then issued, so to install the appropriate packages:

```
sudo apt install nfs-kernel-server
```

The following line was appended to the NFS configuration file `/etc/exports`:

```
/data 172.31.0.0/16(rw,sync,no_wdelay)
```

Finally, the owner and group of the shared folder were changed to `nobody:nogroup` and the permissions of the folder were edited so to grant unlimited access to it:

```
sudo chown nobody:nogroup /data
sudo chmod 777 /data
```

The `/data` folder was so made available to all the Worker Nodes on the address range `172.31.0.0/16`. This configuration does not pose a security risk since the Master Node and Worker Nodes machines belong to a Virtual Private Cloud (VPC), and so only machines instantiated on it will be able to access the volume. Moreover, this configuration grants immediate access to the volume to newly instantiated Worker Nodes. A mock file was created on the `/data` folder so to be able to recognize it when mounted.

```
touch /data/this_is_a_shared_NFS_volume
```

1.3 Configuration of the Worker Nodes

A virtual machine identical to the one used for the Master Node was instantiated. The PS1 prompt of this first Worker Node was changed so to make the node easily identifiable from the command line.

```
ubuntu@bpd1-worker-node:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@bpd1-worker-node\[\033[00m\]:\[\
033[01;34m\]\w\[\033[00m\]\$
```

HTCondor was installed in this system with the same procedure used for the Master Node. Only the `/etc/condor/condor_config` file was configured differently, by appending the following lines to it:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>

# Worker Node config
DAEMON_LIST = MASTER, STARTD

HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *
```

On the Worker Node access to the shared NFS volume was then set up. The following command was issued to install the required packages and enable the respective services:

```
sudo apt install nfs-common
sudo systemctl start nfs-server
sudo systemctl enable nfs-server
```

A new directory was then created at `/data` using the `mkdir` command. The `/etc/fstab` file was edited by appending the following line, so that the shared volume would be automatically mounted at boot under the directory `/data`:

```
<Master_Node_private_IP>:/data   /data   nfs   defaults            0 0
```

It was then verified that the shared volume was accessible from the Worker Node.

```
ubuntu@bpd1-worker-node:~$ ll /data
total 24
drwxrwxrwx  3 nobody nogroup  4096 Jun 17 07:16 ./
drwxr-xr-x 24 root   root     4096 Jun 17 06:46 ../
drwx------  2 root   root    16384 Jun 16 17:38 lost+found/
-rw-rw-r--  1 ubuntu ubuntu      0 Jun 17 07:16 this_is_a_shared_NFS_volume
```

On the Worker Nodes the application BWA was installed with the command:

```
sudo apt install -y bwa
```

An image of the virtual machine was taken through the AWS interface. In this way, the Worker Nodes would be easily replicable when more computational power is needed by instantiating new virtual machines from the image, without the need for manual configuration.

1.4 Submission of a Test Job to the HTCondor Cluster

For demonstrating the use of the newly created cluster, a new Worker Node was instantiated from the Worker Node image, so that the cluster would contain the Master Node and 2 Worker Nodes.

Subsequently, a new volume was created in the AWS interface from a snapshot containing test data used for the BDP1 course (`snap-09ee52d8038fb8094`, BDP1_2020). This snapshot contains NGS reads from 3 different patients. Each patient has a folder with around 500 fasta files, with 1000 reads each. Part of this data will be used for testing the infrastructure. The new volume was mounted on the Master Node under `/data`, substituting the empty volume used before. The `/etc/fstab` file was update accordingly and the `nfs-server` service restarted. It was then tested that the new volume was accessible from the Worker Nodes.

```
ubuntu@bpd1-worker-node:~$ ll /data
total 28
drwxrwxrwx  4 root root  4096 Apr 26 13:38 ./
drwxr-xr-x 24 root root  4096 Jun 17 06:46 ../
drwxr-xr-x  5 root root  4096 Apr 19 14:39 BDP1_2020/
drwx------  2 root root 16384 Apr 26 09:37 lost+found/
```

The test job consisted in aligning ten fasta files (`read_1.fa` to `read_10.fa` from patient 1) to the human genome build hg19, also stored on the shared volume. The BWA alignment tool was used for the scope. This tool takes advantage of indexing the genome for speeding up the alignment of low-divergent reads. The index for the hg19 build was already present in the volume snapshot, so there was no need to create it from scratch. The test fasta files were copied from the shared volume to the home folder of the Master Node. The test job file `alignment_test.job` was created, with the following content:

```
#####################################################
################### Alignment Test ##################
#####################################################


############ The program that will be executed #######

Executable = alignment_test.py
readnum = $(Process)+1
arguments = read_$INT(readnum).fa

############ Input Sandbox ##########################

Input     = read_$INT(readnum).fa
transfer_input_files = read_$INT(readnum).fa

###### Output Sandbox ################################

Log       = read_$INT(readnum).log
# will contain condor log

Output    = read_$INT(readnum).out
# will contain the standard output

Error     = read_$INT(readnum).error
# will contain the standard error

############## condor control variables #############

should_transfer_files = YES
```

```
when_to_transfer_output = ON_EXIT

Universe = vanilla

##################################################

Queue 10
```

The script `alignment_test.py` called as executable in `alignment_test.job` was the following:

```
#!/usr/bin/python
import sys,os
from timeit import default_timer as timer

start = timer()
dbpath = "/data/BDP1_2020/hg19/"
dbname = "hg19bwaidx"
queryname = sys.argv[1]
out_name = queryname[:-3]
fafile = queryname
samfile = out_name + ".sam"
gzipfile = out_name + ".sam.gz"
saifile = out_name + ".sai"
md5file = out_name + ".md5"

print "Input: ", queryname

command = "bwa aln -t 1 " + dbpath + dbname + " " + fafile + " > " + saifile
print "launching command: " , command
os.system(command)

command = "bwa samse -n 10 " + dbpath + dbname + " " + saifile + " " + fafile + " > " + samfile
print "launching command: " , command
os.system(command)

# Checksums
print "Creating md5sums"
os.system("md5sum " + samfile + " > " + md5file)

print "gzipping out text file"
command = "gzip " + samfile
print "launching command: " , command
os.system(command)

# Transfer files to shared volume and clean the Output Sandbox
print "Moving files and clearing the Output Sandbox"
os.system("mv "+ gzipfile + " /data/outputs/"+ gzipfile)
os.system("mv "+ md5file + " /data/outputs/"+ md5file)
os.system("rm "+ saifile)

execution_time = timer() - start

print "Total execution time: " + str(execution_time)
print "exiting"

exit(0)
```

Ten instances of this test job were run on the cluster. The following outputs of `condor_q` and `condor_status` were recorded after submission:

```
ubuntu@bdp1-master-node:~$ condor_q


-- Schedd: ip-172-31-8-109.ec2.internal : <172.31.8.109:9618?... @ 06/18/20 08:03:52
OWNER  BATCH_NAME    SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
ubuntu ID: 24       6/18 08:03      _      _     10     10 24.0-9

Total for query: 10 jobs; 0 completed, 0 removed, 10 idle, 0 running, 0 held, 0 suspended
Total for ubuntu: 10 jobs; 0 completed, 0 removed, 10 idle, 0 running, 0 held, 0 suspended
Total for all users: 10 jobs; 0 completed, 0 removed, 10 idle, 0 running, 0 held, 0 suspended
```

```
ubuntu@bdp1-master-node:~$ condor_status
Name                               OpSys      Arch   State     Activity LoadAv Mem

slot1@ip-172-31-8-22.ec2.internal  LINUX      X86_64 Claimed   Busy      0.000 3979
slot2@ip-172-31-8-22.ec2.internal  LINUX      X86_64 Claimed   Busy      0.000 3979
slot1@ip-172-31-14-66.ec2.internal LINUX      X86_64 Claimed   Busy      0.000 3979
slot2@ip-172-31-14-66.ec2.internal LINUX      X86_64 Claimed   Busy      0.000 3979

               Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

  X86_64/LINUX     4     0       4         0       0          0        0      0

         Total     4     0       4         0       0          0        0      0
```

The cluster was able to successfully complete the task and the following output files were produced:

```
ubuntu@bdp1-master-node:/data/outputs$ ll
total 408
drwxrwxrwx 2 nobody nogroup  4096 Jun 18 09:12 ./
drwxrwxrwx 5 root   root     4096 Jun 17 15:11 ../
-rw-r--r-- 1 nobody nogroup    45 Jun 18 08:57 read_1.md5
-rw-r--r-- 1 nobody nogroup    31 Jun 18 08:56 read_1.sam.gz
-rw-r--r-- 1 nobody nogroup    46 Jun 18 09:07 read_10.md5
-rw-r--r-- 1 nobody nogroup 55074 Jun 18 09:07 read_10.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 08:57 read_2.md5
-rw-r--r-- 1 nobody nogroup 41129 Jun 18 08:57 read_2.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 08:57 read_3.md5
-rw-r--r-- 1 nobody nogroup    31 Jun 18 08:56 read_3.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 08:57 read_4.md5
-rw-r--r-- 1 nobody nogroup 54914 Jun 18 08:57 read_4.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 09:03 read_5.md5
-rw-r--r-- 1 nobody nogroup 77299 Jun 18 09:03 read_5.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 09:03 read_6.md5
-rw-r--r-- 1 nobody nogroup 55883 Jun 18 09:03 read_6.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 09:00 read_7.md5
-rw-r--r-- 1 nobody nogroup    31 Jun 18 08:58 read_7.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 09:00 read_8.md5
-rw-r--r-- 1 nobody nogroup    31 Jun 18 08:58 read_8.sam.gz
-rw-r--r-- 1 nobody nogroup    45 Jun 18 09:07 read_9.md5
-rw-r--r-- 1 nobody nogroup 53760 Jun 18 09:07 read_9.sam.gz
```

The time required to complete the task on a single fasta file ranged from 62.77 to 422.54 seconds, with an average required time of 227.7 seconds.

```
ubuntu@bdp1-master-node:~$ cat read_*.out |grep "Total execution ti
me:"
Total execution time: 62.765912056
Total execution time: 417.533436775
Total execution time: 116.913583994
Total execution time: 66.0304908752
Total execution time: 115.42266202
Total execution time: 383.357795
Total execution time: 377.316680908
Total execution time: 158.487968922
Total execution time: 161.080944061
Total execution time: 422.54254508
```

1.5 Data Management Model

The general data management model followed would be that of transferring the executables and the input fasta files using the HTCondor Input Sandbox. This is because those files are generally small and their transfer will not overload the Master Node. Moreover, input data are likely to be uploaded dynamically when new sequencing experiments are performed. This system will allow sending easily new inputs to the cluster. On the contrary, the large hg19 genome and index files will be made available to the Worker Nodes through the shared NFS volume. The Output Sandbox will contain the condor log, the standard output, and the standard error. These files are really small and need to be immediately available to the submitter for inspecting the proceeding of the job. The aligned reads will instead be compressed and put in the shared volume, to avoid moving large data on the Output Sandbox.

2 Creation of a Container Image for the Alignment of Next-Generation Sequencing reads

A containerized version of the BWA application was built using `docker`. A container image for the BWA application is already available on DockerHub (`biocontainers/bwa`), but for educational purposes, a new image was built from scratch. The Ubuntu docker image was used as a base. The local docker image `bwa` was built from the following `Dockerfile`:

```
FROM ubuntu
RUN apt update
RUN apt install -y bwa
RUN apt install -y python
COPY ./alignment_test_docker.py alignment_test_docker.py
```

The container was then run using a bind-mount for the `/data` volume and starting an interactive shell.

```
docker run -v /data:/data -ti bwa /bin/bash
```

The script `alignment_test.py` was slightly modified so to omit the final data movement to the shared volume, and renamed `alignment_test_docker.py`. Inside the container, the following command was run so to align a sample fasta file which had been copied from the shared volume:

```
./alignment_test_docker.py read_1.fa
```

The application ran successfully, producing the following output in 213.33 seconds (the container was run on a Ubuntu Server 18.04.4 LTS t2.large AWS virtual machine):

```
root@871e3f6d9c26:/# cp /data/BDP1_2020/hg19/Patients/patient1/read_1.fa read_1.fa
root@871e3f6d9c26:/# ./alignment_test_docker.py read_1.fa
Input:  read_1.fa
launching command:  bwa aln -t 1 /data/BDP1_2020/hg19/hg19bwaidx read_1.fa > read_1.sai
[bwa_aln] 17bp reads: max_diff = 2
[bwa_aln] 38bp reads: max_diff = 3
[bwa_aln] 64bp reads: max_diff = 4
[bwa_aln] 93bp reads: max_diff = 5
[bwa_aln] 124bp reads: max_diff = 6
[bwa_aln] 157bp reads: max_diff = 7
[bwa_aln] 190bp reads: max_diff = 8
[bwa_aln] 225bp reads: max_diff = 9
[bwa_aln_core] calculate SA coordinate... 0.21 sec
[bwa_aln_core] write to the disk... 0.00 sec
[bwa_aln_core] 1000 sequences have been processed.
[main] Version: 0.7.17-r1188
[main] CMD: bwa aln -t 1 /data/BDP1_2020/hg19/hg19bwaidx read_1.fa
[main] Real time: 121.035 sec; CPU: 4.582 sec
launching command:  bwa samse -n 10 /data/BDP1_2020/hg19/hg19bwaidx read_1.sai read_1.fa > read_1.sam
[bwa_aln_core] convert to sequence coordinate... 4.28 sec
[bwa_aln_core] refine gapped alignments... 1.13 sec
[bwa_aln_core] print alignments... 0.00 sec
[bwa_aln_core] 1000 sequences have been processed.
[main] Version: 0.7.17-r1188
[main] CMD: bwa samse -n 10 /data/BDP1_2020/hg19/hg19bwaidx read_1.sai read_1.fa
[main] Real time: 92.240 sec; CPU: 5.416 sec
Creating md5sums
gzipping out text file
launching command:  gzip read_1.sam
Total execution time: 213.335064888
exiting
```

3 Evaluation of the Resources Needed for Solving a Real Use-Case

A plausible non-trivial use-case that can be addressed using the HTCondor cluster developped in Section 1 is the allignment of a big number of fasta files deriving from a Next-Generation Sequencing (NGS) project to the human reference genome assembly hg19. It is typical to aim for a coverage of at least 30x in a human Whole-Genome Sequencing (WGS) experiment. Given that the human genome has a size of about 3.3 billions base pairs, this will result in a raw output from the sequencer of at least 99 billions base pairs. Considering also the possible occurrence of artifacts and low-quality reads, planning for a raw output of at least 100 billion base pairs is advisable. A single NGS read of an Illumina MySeq sequencer can have a length of up to 300 base pairs (https://www.illumina.com/systems/sequencing-platforms/miseq.html).Here I will adopt a read lenght of 150 base pairs, which is recommended for a WGS experiment (https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/read-length.html). This will result in a total of almost 667 million reads for a single patient. This amounts to 667000 fasta files of 1000 reads each. From the tests performed, I roughly determined that the time needed to align one fasta file with 1000 reads of 150 base pairs to the hg19 assembly on a single core of the AWS `t2.large` machine is 227.3 seconds. Thus, completing the alignment of the output of a single WGS experiment would require 151609100 seconds of CPU time on a single core. Of course this can vary based on the lenght and composition of the reads themselves. The cluster described in this project consists of one Master Node and two Worker Nodes. The Worker Nodes have two cores each, so the cluster can employ a total of four cores. In this scenario, completing the task would require 37902275 seconds, 1 year 2 months and 11 days. Of course this running time is not acceptable, and because of that it would be advisable to replicate the Worker Nodes, so to have more compuattional power. For instance, using a cluster of 1 Master Node and 100 Worker Nodes the task would be completed in less than 9 days. This will result in a cluster with a capacity of 3 patients per month.

I will now describe a breakdown of the expected costs for aligning the the output of a WGS experiment from a single patient using the AWS EC2 system with On-Demand pricing on the US East N. Virginia availability zone, on a Linux system. The `t2.large` machine costs $ 0.0928 per hour, and the cluster will require 100 of them for the Worker Nodes. The `t2.medium` machine costs $ 0.0464 per hour, and the cluster will require just 1 of them for the Master Node. With an expected time needed for the task of 9 days, the costs for the computation for one patient will amount to $ 1963. Storage on general purpose SSD EBS devices costs $ 0.1 per Gb per month. A 1 Tb disk can be used for storing the hg19 genome and index, as well as the reads for the patients. Therefore, storage will have a cost of $ 100 per month. Note that costs for data movements in and out of the AWS system

have been omitted.