



REPORT

Building Cloud IaaS infrastructures and Computing Models

Giacomo Orsini, Laia Torres Masdeu

Abstract

The aim of this work was the construction of a Cloud computing infrastructure using Amazon Web Services (AWS), simulating a geographical separation of sites, in order to solve a computational challenge for a biological model. During the creation of such infrastructure, we installed and used the Network File System (NFS), HTCondor batch system and the WebDav data transfer tool, to manage data intra and inter sites. A volume of existing data previously created for the IBDPI course of the Master in Bioinformatics was attached to the master nodes, and shared among the respective worker nodes using NFS. During the building process, we implemented the infrastructure with two other sites, each composed by a single Master node, created with the Google Cloud Platform (GCP). While solving the computational challenge, we also created a Docker custom image to run jobs with the HTCondor system. In the end, we retrieved a time/cost table to evaluate the performance of our infrastructure, discussing different possibilities to enhance it.

Contents

| | | |
|----------|---|-----------|
| 1 | Build a computing infrastructure on the Cloud | 2 |
| 1.1 | Create the instances on AWS | 3 |
| 1.2 | Attach an existing volume | 4 |
| 1.3 | Install a NFS Client-Server | 5 |
| 1.4 | Install a batch system | 6 |
| 1.5 | Install a data transfer tool | 7 |
| 2 | Simulation of the geographic distribution of sites | 8 |
| 3 | Extension of the Infrastructure | 9 |
| 4 | The biological challenge | 10 |
| 4.1 | BWA algorithm, hg19 database and project data | 10 |
| 4.2 | Preparing and submitting jobs to HTCondor | 11 |
| 5 | Work with containers: Docker | 14 |
| 6 | Results: time-costs estimation | 15 |
| 7 | Discussion | 16 |
| 7.1 | Managing the workload | 17 |
| 8 | Supplementary materials | 18 |

1. Build a computing infrastructure on the Cloud

A small Iaas (Infrastructure as a Service) infrastructure was built with Amazon Web Services (AWS) Elastic Compute Cloud (EC2) as a provider. This service provides fundamental hardware resources, servers, storage, and networking components, allowing us to manage operating systems, applications, and data. We wanted to simulate two geographically distributed computing sites, composed by at least 2 nodes each (Fig 1):

- Site 1, with 3 nodes: 1 master (S1_M) and 2 workers (S1_W1 and S1_W2)
- Site 2, with 2 nodes: 1 master (S2_M) and 1 worker (S2_W2)

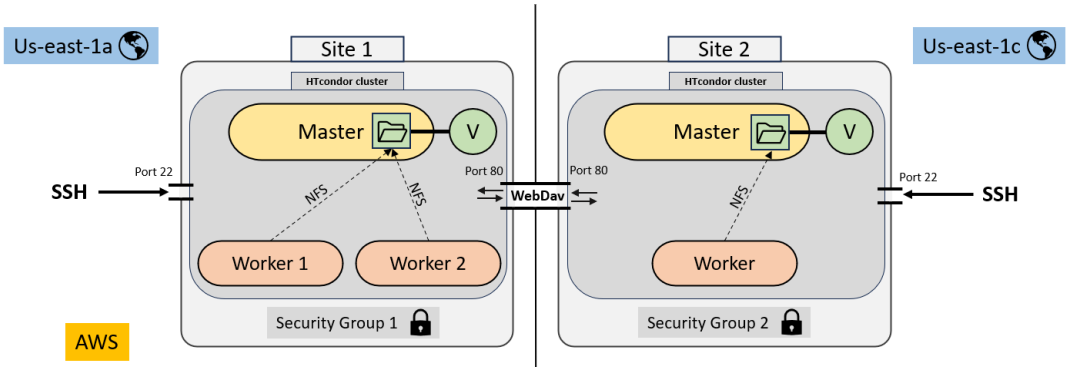


Figure 1. Scheme of the Cloud infrastructure.

A data volume created from an existing snapshot was attached to both Master nodes. In both sites, the Network File System (NFS) was installed to share data from the master nodes to the worker nodes of the same site. Then, a batch system for handling computations and CPUs (HTCondor) was installed, configuring the machines as Master/Workers. We then simulated the sharing of some data from the volume attached to the Master to the workers in each site. Finally, the WebDav tool was installed in both sites to simulate data transfer between the two. The HTCondor system has been used to test the computational challenge in section 4.

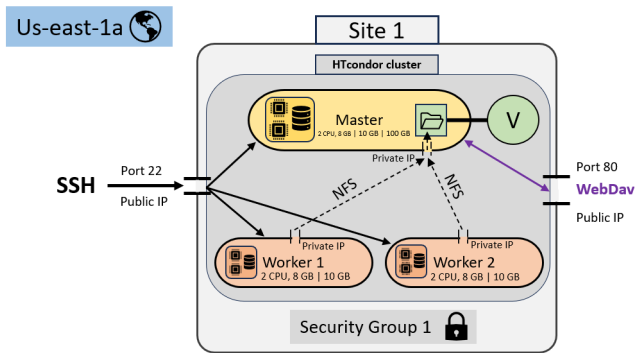


Figure 2. Site 1 diagram, showing nodes, nodes features, ports, connections, security group and zone.

1.1 Create the instances on AWS

All the instances (Virtual Machines) were created with the Amazon Elastic Compute Cloud (Amazon EC2) service, each instance represents a node, and each of the 2 sites were created using 2 different AWS account.

To create each node, we used the following procedure:

1. Rename the instances (`site1_master`, `site1_worker1`, `site1_worker2`, and `site2_master`, `site2_worker1`)
2. Select an open source Linux distribution: we decided to use the `RHEL-7.9_HVM-20220512-x86_64-1-Hourly2-GP2` version, published on the 2022-05-12.
3. Select a type of instance for each node: we selected `t2-large` (2 CPU, 8 GB RAM) for all.
4. Create a new key pair. This is done only once, when creating the first instance of each site. Save the key (`.pem`) document to directory (from which you will need to initialise all the instances from this directory, or change the key file directory from the initialisation command). In the other instances, select the previously created key pair.
5. To simulate the geographical separation, select two different Availability Zones (sub nets) in the network settings. We selected `us-east 1a` for site 1 and `us-east 1c` for site 2.
6. Create a security group for each site (create it for master, select existing for workers). We created `site1-security` for site 1 and `site2-security` for site 2.
7. Modify inbound security groups rules so that only you can access the sites: we removed the default (open to all, 0.0.0.0/0) and restricted to the current IP address: Security group rule 1 (TCP, 22, <your_IP>/32, SSH for my IP address)
8. (Optional) Select different size storage, default being 10 gb. We selected 30 gp2 for site2.

Once each site was created, we connected to each node using `ssh` (`ssh -i "<key_name>.pem" ec2-user@ec2-<AWS_node_public_IP>.compute-1.amazonaws.com`). To help with the prompt visualisation, we decided to change the prompt both in the `ec2-user` and `root` users to the name of each site and node we were working on. To do this, we modified the `.bashrc` file in both users. The specific lines of code used can be found in our [GitHub repository](#).

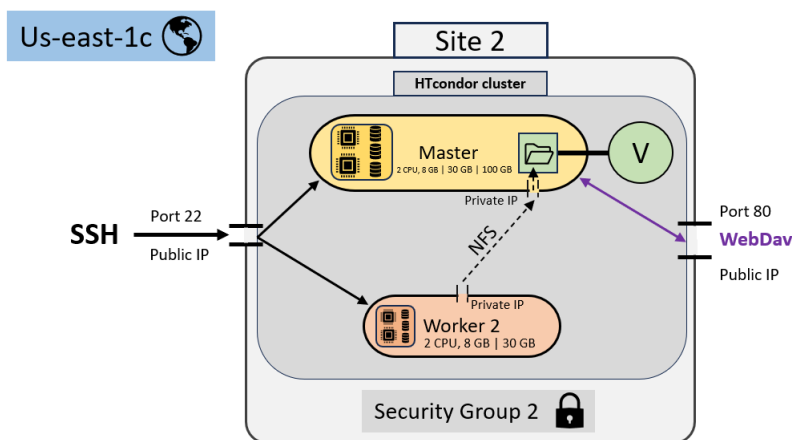


Figure 3. Site 2 diagram, showing nodes, nodes features, ports, connections, security group and zone.

1.2 Attach an existing volume

To extend the capacity of our nodes and to have data on which to operate, we attached a volume to each of our master nodes. These volumes were created in AWS with default settings from a snapshot made by the professor of the BDPI course of the Master in Bioinformatics (University of Bologna), hence they were retrieved from the "Snapshot ID" option by using the code `snap-BDPI_2023`. We selected the correct Availability Zone for the volume of each site. Once the 2 volumes were created, they had to be attached to a node. To attach the volumes to the master nodes, we connected them to the Master instances through the EC2 control panel, in the volume section; then, on the virtual machines, we used the commands found in our [GitHub repository](#).

It is important to remark that the first mount command, `mount -a`, already mounted the volume into the `\project_data` directory of our worker node (Figure 4).

```
root@S1_M:~$ fdisk -l
WARNING: fdisk GPT support is currently new, and therefore in an experimental phase. Use at your own discretion.
Disk /dev/xvda: 10.7 GB, 10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: gpt
Disk identifier: 6E264842-3CE1-44EF-8459-092B8A48B7F0

#               Start          End              Size      Type           Name
#-----
1             2048             4095             1M        BIOS boot
2             4096          20971485          10G        Linux filesystem

Disk /dev/xvdf: 107.4 GB, 107374182400 bytes, 209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0xa3daa87c

   Device Boot      Start         End      Blocks    Id  System
/dev/xvdf1          2048      209715199      104856576   83  Linux
root@S1_M:~$ mkdir /project_data
root@S1_M:~$ mount -t ext4 /dev/xvdf1 /project_data/
root@S1_M:~$ ll /project_data/
total 20
drwxr-xr-x. 5 root root 4096 Apr 19 2020 BDPI_2023
drwx-----. 2 root root 16384 Apr 26 22:33 lost+found
```

Figure 4. Screenshot of the terminal, Site 1 Master: commands used in the attaching the nodes procedure. These commands do not perform a permanent attachment.

However, this was not permanent. To make it permanent, the paths for the volume and the destination source in the VM were added to the `/etc/fstab` file (Figure 5).

```
root@S1_M:~$ vi /etc/fstab
root@S1_M:~$ cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Thu May 12 23:48:51 2022
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=95070429-de61-4430-8ad0-2c0f109d8d50 / xfs defaults 0 0

/dev/xvdf1 /project_data ext4 defaults 0 0
root@S1_M:~$ mount -a
root@S1_M:~$ ll /project_data/
total 20
drwxr-xr-x. 5 root root 4096 Apr 19 2020 BDPI_2023
drwx-----. 2 root root 16384 Apr 26 22:33 lost+found
```

Figure 5. Screenshot of the terminal, Site 1 Master. Modifying the `fstab` file makes the volume permanently attached.

After these steps, each volume was attached to the respective master node through a path leading to a folder, the `project_data` folder. The volume can be found locally in this directory. However, the directory is not accessible to any of the worker nodes. To enable the data transfer, we installed NFS in each master and worker.

1.3 Install a NFS Client-Server

The Network File System (NFS) is a network file sharing protocol that allows for machines **under the same network** to share directories and files. The client can access files and directories exposed by the host as if they were locally stored. This tool allowed us to share the attached data volume from the Master node (host) to the Worker nodes (clients) in each site. More precisely, it allowed the worker nodes to access the directory of the mounted volume in the master nodes.

In our case, we just wanted the worker nodes to access the volume, not to modify it. Therefore, we granted *read*, *write*, and *execute* permissions to the owner (master node) of the directory and to the groups associated to it, but only *read* and *execute* permissions (4+1) to other users (worker nodes). This was done using the `chmod 775 \project_data` command. The specific lines of code used to install and connect the NFS to the master and worker nodes can be found in our [GitHub repository](#).

Note: To grant *write* access to the worker nodes (so that files/directories can be added to the volume and be accessed by all nodes connected to NFS) run `chmod 777`.

It is very important to add the worker(s) private IP address(es) to the security group inbound rules for the NFS port, so that the master node can accept requests from the worker node(s).

To enable the share of the volume (in the `\project_data` dir), the file `/etc/exports` of the server (master) node was modified so that the directory was accessible from the client (worker) node(s). After exporting, we checked that the directory and client (worker) IP address(es) are correct (Figure 6).

```
root@S1_M:~# cat /etc/exports
/project_data 172.31.3.87(rw,sync,no_wdelay)
/project_data 172.31.0.156(rw,sync,no_wdelay)
root@S1_M:~# exportfs -r
root@S1_M:~# exportfs
/project_data 172.31.3.87
/project_data 172.31.0.156
```

Figure 6. Screenshot of the terminal, Site 1 Master Node: the exports file was correctly modified with the workers IP addresses. The export procedure was then performed.

On the worker node(s), we created a directory, also called `\project_data`, on which we mounted the NFS. It was set to be mounted automatically at boot time by modifying the `/etc/fstab` file. In Figure 7, we can see that the `mount -a` command worked correctly.

```
root@S1_W1:~# mkdir /project_data
root@S1_W1:~# cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Thu May 12 23:48:51 2022
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=95070429-de61-4430-8ad0-2c0f109d8d50 / xfs defaults 0 0
172.31.2.239:/project_data /project_data nfs defaults 0 0
root@S1_W1:~# ll /project_data/
total 0
root@S1_W1:~# mount -a
root@S1_W1:~# ll /project_data/
total 20
drwxr-xr-x. 5 root root 4096 Apr 19 2020 BDP1_2023
drwx----- 2 root root 16384 Apr 26 22:33 lost+found
```

Figure 7. Screenshot of the terminal, Site 1 Worker Node 1: the procedure worked fine, as we can see the data from the worker node's directory.

With this, the Storage resources were correctly shared among all the nodes in each site.

1.4 Install a batch system

A batch system is a computer application for controlling unattended background program execution of jobs (execution tasks). Our sites can be considered as CPU farms; the batch processing is the automatic execution of jobs by the CPUs of the farm. One of the machines acts as a manager, a single point of control of the submitted jobs. The user has just to submit the jobs, then the submission gets organised through queues, all thanks to the batch system. We selected HTCondor as batch system (Condor High-Throughput Computing System). The specific lines of code used to install and connect the HTCondor batch system in the master and worker nodes can be found in our [GitHub repository](#).

It is very important to modify some settings in the security group inbound rules of each site, to allow HTCondor to work correctly:

- tcp ports (0 - 65535) for the same security group, for all of the nodes connected to said security group (i.e. A11 TCP TCP 0 - 65535 sg-008742ba0467986fe)
- ping port for the same security group (i.e. A11 ICMP-IPv4 A11 N/A sg-008742ba0467986fe)

In Figure 8, we can see how the final HTCondor structure was defined for the master (Figure 8a) and worker (Figure 8b) nodes.

```
root@S2_M:~# systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2023-06-18 13:59:40 UTC; 12s ago
   Main PID: 2703 (condor_master)
   Status: "All daemons are responding"
   CGroup: /system.slice/condor.service
           └─2703 /usr/sbin/condor_master -f
             └─2746 condor_proc -A /var/run/condor/procdd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
               └─2747 condor_shared_port -f
                 └─2748 condor_collector -f
                   └─2749 condor_negotiator -f
                     └─2750 condor_startd -f
                       └─2751 condor_schedd -f
                         └─2771 kflaps

Jun 18 13:59:40 ip-172-31-47-82.ec2.internal systemd[1]: Started Condor Distributed High-Throughput-Computing.
root@S2_M:~# ps -aux | grep condor
condor 2703 0.1 0.0 70228 6616 ? Ss 13:59 0:00 /usr/sbin/condor_master -f
root 2746 0.2 0.0 25952 3116 ? S 13:59 0:00 condor_proc -A /var/run/condor/procdd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
condor 2747 0.1 0.0 45736 5500 ? Ss 13:59 0:00 condor_shared_port -f
condor 2748 0.2 0.0 46460 6180 ? Ss 13:59 0:00 condor_collector -f
condor 2749 0.1 0.0 46204 5964 ? Ss 13:59 0:00 condor_negotiator -f
condor 2750 0.2 0.0 46608 6824 ? Ss 13:59 0:00 condor_startd -f
condor 2751 0.1 0.0 47516 7804 ? Ss 13:59 0:00 condor_schedd -f
condor 2805 91.0 0.0 21268 1128 ? R 13:59 0:00 mips
root 2807 0.0 0.0 112808 976 pts/0 S+ 14:00 0:00 grep --color=auto condor
root@S2_M:~#
```

(a) Screenshot of the terminal, Site 2 Master Node: structure of the HTCondor system on the master node; all the procedures were done correctly and the system was active

```
root@S2_M:~# systemctl enable condor
Created symlink from /etc/systemd/system/multi-user.target.wants/condor.service to /usr/lib/systemd/system/condor.service.
root@S2_M:~# systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2023-06-18 14:00:23 UTC; 14s ago
   Main PID: 2683 (condor_master)
   Status: "All daemons are responding"
   CGroup: /system.slice/condor.service
           └─2683 /usr/sbin/condor_master -f
             └─2714 condor_proc -A /var/run/condor/procdd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
               └─2715 condor_shared_port -f
                 └─2716 condor_startd -f
                   └─2727 kflaps

Jun 18 14:00:23 ip-172-31-47-189.ec2.internal systemd[1]: Started Condor Distributed High-Throughput-Computing.
root@S2_M:~# ps -aux | grep condor
condor 2683 0.1 0.0 70160 6080 ? Ss 14:00 0:00 /usr/sbin/condor_master -f
root 2714 0.2 0.0 25952 3032 ? S 14:00 0:00 condor_proc -A /var/run/condor/procdd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
condor 2715 0.1 0.0 45736 5496 ? Ss 14:00 0:00 condor_shared_port -f
condor 2716 0.2 0.0 46604 6824 ? Ss 14:00 0:00 condor_startd -f
condor 2761 102 0.0 21268 1124 ? R 14:00 0:02 mips
root 2763 0.0 0.0 112808 972 pts/0 R+ 14:00 0:00 grep --color=auto condor
root@S2_M:~#
```

(b) Screenshot of the terminal, Site 2 Worker Node: structure of the HTCondor system on the master node; all the procedures were done correctly and the system was active

Figure 8. Commands to assess structure and activation of HTCondor

With these settings, the master node (Manager Machine) worked both as scheduler and as worker, whereas the worker node(s) could only execute jobs (Figure 9). Hence, the master is the only node which was able to manage and share the submitted jobs (schedd) according to the available resources (negotiator), and it was also the node that received and collected the outputs once the jobs were executed (collector).

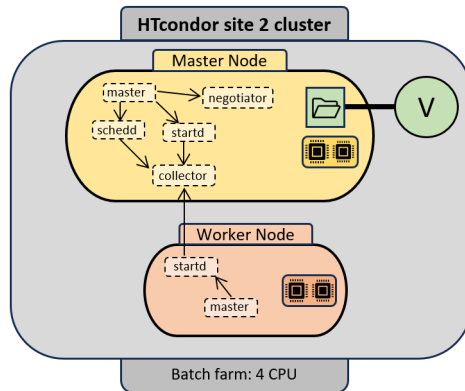


Figure 9. Diagram of Site 2 HTCondor cluster

1.5 Install a data transfer tool

At this point, both sites were set up for sharing and managing files between their respective nodes. We supposed, though, that we would also want to exchange files between the two sites. To do that, we select a Data Transfer Tool, the WebDav application (Web Distributed Authoring and Versioning). It is an extension of the Hypertext Transfer Protocol (HTTP) that allows clients to perform remote Web content authoring operations. With this, we can create, change and move documents on a server. We will show a brief data transfer with the application. Site 1 was set as server and site 2 as client. The specific lines of code used to install and connect WebDav in the master and worker nodes can be found in our [GitHub repository](#).

As in the previous steps, it is very important to add the **public** client IP address (in our case, site 2 master) to the security group inbound rules of site 1 for port 80, so that the server (site 1 master) can connect to the client (site 2 master). To access the tool, there is the need to have a username and a password: the account accessed from each site has to be the same.

In Figure 10, we can see how the data transfers among site 1 (Figure 13a) and 2 (Figure 13b) are working. Note that the IP address to connect to each site is different: for site 1 master, the server, we used the **private** S1_M IP, whereas for site 2 master, the client, we used the **public** S1_M IP.

```

root@S1_M:/project_data$ ls
BDP1_2023 hi_site2.txt lost+found
root@S1_M:/project_data$ cadaver http://172.31.2.239/webdav/
Authentication required for webdav on server '172.31.2.239':
Username: bdp1_project
Password:
dav:/webdav/> ls
Listing collection '/webdav/': succeeded.
Coll: dir1 0 Jun 18 16:08
Coll: dir2 0 Jun 18 16:09
dav:/webdav/> cd dir1
dav:/webdav/dir1/> ls
Listing collection '/webdav/dir1/': collection is empty.
dav:/webdav/dir1/> put /project_data/hi_site2.txt
Uploading /project_data/hi_site2.txt to '/webdav/dir1/hi_site2.txt':
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir1/> cd ../dir2
dav:/webdav/dir2/> ls
Listing collection '/webdav/dir2/': succeeded.
Coll: hi_site1.txt 12 Jun 18 16:13
dav:/webdav/dir2/> get hi_site1.txt
Downloading '/webdav/dir2/hi_site1.txt' to hi_site1.txt:
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir2/> exit
Connection to '172.31.2.239' closed.
root@S1_M:/project_data$ ls
BDP1_2023 hi_site1.txt hi_site2.txt lost+found

```

(a) Screenshot of the terminal, Site 1 Master Node: from Site 1, the user was able to retrieve data uploaded from Site 2 (dir2/hi_site1.txt). The file was downloaded in the working directory.

```

root@S2_M:/project_data$ ls
BDP1_2023 hi_site1.txt lost+found
root@S2_M:/project_data$ cadaver http://52.54.54.184/webdav/
Authentication required for webdav on server '52.54.54.184':
Username: bdp1_project
Password:
dav:/webdav/> ls
Listing collection '/webdav/': succeeded.
Coll: dir1 0 Jun 18 16:08
Coll: dir2 0 Jun 18 16:09
dav:/webdav/> cd dir2
dav:/webdav/dir2/> ls
Listing collection '/webdav/dir2/': collection is empty.
dav:/webdav/dir2/> put /project_data/hi_site1.txt
Uploading /project_data/hi_site1.txt to '/webdav/dir2/hi_site1.txt':
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir2/> cd ../dir1
dav:/webdav/dir1/> ls
Listing collection '/webdav/dir1/': succeeded.
Coll: hi_site2.txt 12 Jun 18 16:13
dav:/webdav/dir1/> get hi_site2.txt
Downloading '/webdav/dir1/hi_site2.txt' to hi_site2.txt:
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir1/> exit
Connection to '52.54.54.184' closed.
root@S2_M:/project_data$ ls
BDP1_2023 hi_site1.txt hi_site2.txt lost+found

```

(b) Screenshot of the terminal, Site 2 Master Node: from Site 2, the user was able to retrieve data uploaded from Site 1 (dir1/hi_site2.txt). The file was downloaded in the working directory.

Figure 10. Example of data transfer on WebDav.

2. Simulation of the geographic distribution of sites

To simulate the geographic distribution of our two sites we used two approaches:

1. Security groups: we created a different security group for each site (site1-security and site 2-security). Security groups act as a firewall that control the traffic that is allowed to enter and exit the VMs associated to that group. Inbound and outbound rules, respectively, allow to choose what ports or protocols can access the security group. Each site was made in a different AWS account, so there was no way that the two sites could share the security group, and therefore traffic had to be controlled separately for each.
2. AWS availability zones: we selected a different availability zone for each site (us-east 1a and us-east 1c). This clearly situated the two sites in different zones: both in us-east, but each in a different subzone.

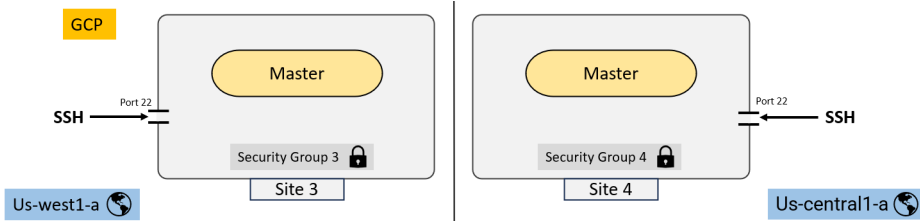


Figure 11. Diagram of Site 3 and Site 4

3. Extension of the Infrastructure

While building the infrastructure, we decided to extend it by adding another 2 sites, built with the Google Cloud Platform (GCP). The Google Cloud Platform provides the same services as AWS, allowing us to create and manage various types of instances. We created two sites, each composed by a Master node, separated geographically from each other and the AWS sites, with a different subnet and a different security group (Figure 11):

- Site 3, with 1 node: 1 master (S3_M)
- Site 4, with 1 node: 1 master (S4_M)

The steps to build the instances are very similar to the one already performed, translated on the Google system:

1. On the VM instances click on the create instance menu, it will open a page similar to the AWS one with the settings to create the instances.
2. Rename the instances (site3_master, site4_master)
3. To simulate the geographical separation, select two different Availability Zones (regions). We selected us-central1-a for site 3 and Us-west1-a for site 4.
4. Select a type of instance for each node: we selected e2-medium (2 CPU) for all.
5. Select as boot disk CentOS 7, balanced permanent disk as disk type.
6. Default network settings.
7. Create the instances. You can access them with Google itself, if you want to connect to them as ssh client you have to generate and download an ssh key (the commands to do that can be found in the Google manuals [Add SSH keys to VMs](#) and [Connecting using third party tools](#)).

The instance can be accessed using the command: `ssh -i <path_to_the_key> username@ <VM_external_ip>`.

After creating the two sites, in each we installed Cadaver (WebDav) for transferring data between sites. We were able to connect these two new sites with site 1 and site 2 trough WebDav and perform some basic data transfer between them, moving data from the AWS instances to the GCP instances and viceversa (Figure 12 and 13).

```
root@S1_M:/project_data/file_transfer$ ls
hi_site1.txt hi_site2.txt hi_site3_from_site1.txt
root@S1_M:/project_data/file_transfer$ cadaver http://172.31.2.239/webdav/
Authentication required for webdav on server '172.31.2.239':
Username: bdp1_project
Password:
dav:/webdav/> ls
Listing collection '/webdav/': succeeded.
Coll: dir1                0 Jul 9 22:30
Coll: dir2                0 Jul 9 22:29
Coll: dir3                0 Jul 9 22:30
Coll: dir4                0 Jul 9 22:31
dav:/webdav/> cd dir1
dav:/webdav/dir1/> ls
Listing collection '/webdav/dir1/': succeeded.
Coll: hi_site2.txt       12 Jun 18 16:13
dav:/webdav/dir1/> put hi_site3_from_site1.txt
Uploading hi_site3_from_site1.txt to '/webdav/dir1/hi_site3_from_site1.txt':
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir1/> ls
Listing collection '/webdav/dir1/': succeeded.
Coll: hi_site2.txt       12 Jun 18 16:13
Coll: hi_site3_from_site1.txt 12 Jul 9 22:33
dav:/webdav/dir1/> cd ../dir4
dav:/webdav/dir4/> ls
Listing collection '/webdav/dir4/': succeeded.
Coll: hi_site1_from_site4.txt 18 Jul 9 22:33
dav:/webdav/dir4/> get hi_site1_from_site4.txt
Downloading '/webdav/dir4/hi_site1_from_site4.txt' to hi_site1_from_site4.txt:
Progress: [=====] 100.0% of 18 bytes succeeded.
dav:/webdav/dir4/> exit
Connection to '172.31.2.239' closed.
root@S1_M:/project_data/file_transfer$ ls
hi_site1_from_site4.txt hi_site1.txt hi_site2.txt hi_site3_from_site1.txt
```

(a) Screenshot of the terminal, Site 1 Master Node: from Site 1, the user was able to upload data on its directory (dir1/hi_site3_from_site1.txt); also it was able to retrieve a file made for Site 1 (dir4/hi_site1_from_site4.txt). The file was downloaded in the working directory.

```
root@S2_M:/project_data/data_transfer$ ls
hi_site1.txt hi_site2.txt hi_site4_from_site2.txt
root@S2_M:/project_data/data_transfer$ cadaver http://34.205.90.164/webdav/
Authentication required for webdav on server '34.205.90.164':
Username: bdp1_project
Password:
dav:/webdav/> ls
Listing collection '/webdav/': succeeded.
Coll: dir1                0 Jul 9 22:30
Coll: dir2                0 Jul 9 22:29
Coll: dir3                0 Jul 9 22:30
Coll: dir4                0 Jul 9 22:31
dav:/webdav/> cd dir2
dav:/webdav/dir2/> ls
Listing collection '/webdav/dir2/': succeeded.
Coll: hi_site1.txt       12 Jun 18 16:13
dav:/webdav/dir2/> put hi_site4_from_site2.txt
Uploading hi_site4_from_site2.txt to '/webdav/dir2/hi_site4_from_site2.txt':
Progress: [=====] 100.0% of 13 bytes succeeded.
dav:/webdav/dir2/> ls
Listing collection '/webdav/dir2/': succeeded.
Coll: hi_site1.txt       12 Jun 18 16:13
Coll: hi_site4_from_site2.txt 13 Jul 9 22:33
dav:/webdav/dir2/> cd ../dir3
dav:/webdav/dir3/> ls
Listing collection '/webdav/dir3/': succeeded.
Coll: hi_site2_from_site3.txt 0 Jul 9 22:33
dav:/webdav/dir3/> get hi_site2_from_site3.txt
Downloading '/webdav/dir3/hi_site2_from_site3.txt' to hi_site2_from_site3.txt: [.] succeeded.
dav:/webdav/dir3/> exit
Connection to '34.205.90.164' closed.
root@S2_M:/project_data/data_transfer$ ls
hi_site1.txt hi_site2_from_site3.txt hi_site2.txt hi_site4_from_site2.txt
```

(b) Screenshot of the terminal, Site 2 Master Node: from Site 2, the user was able to upload data on its directory (dir2/hi_site4_from_site2.txt); also it was able to retrieve a file made for Site 2 (dir3/hi_site2_from_site3.txt). The file was downloaded in the working directory.

Figure 12. Data transfer on WebDav from different instances made with different Cloud computing services (AWS, GCP).

```

[root@site3-master ~]# ls
hi_site2_from_site3.txt
[root@site3-master ~]# cadaver http://34.205.90.164/webdav
Authentication required for webdav on server '34.205.90.164':
Username: bdp1_project
Password:
dav:/webdav/> ls
Listing collection '/webdav/': succeeded.
Coll: dir1      0 Jul 9 22:30
Coll: dir2      0 Jul 9 22:29
Coll: dir3      0 Jul 9 22:30
Coll: dir4      0 Jul 9 22:31
dav:/webdav/> cd dir3
dav:/webdav/dir3/> ls
Listing collection '/webdav/dir3/': succeeded.
dav:/webdav/dir3/> put hi_site2_from_site3.txt
Uploading hi_site2_from_site3.txt to '/webdav/dir3/hi_site2_from_site3.txt': succeeded.
dav:/webdav/dir3/> ls
Listing collection '/webdav/dir3/': succeeded.
          hi_site2_from_site3.txt      12 Jun 18 16:13
dav:/webdav/dir3/> cd ../dir1
dav:/webdav/dir1/> ls
Listing collection '/webdav/dir1/': succeeded.
          *hi_site2.txt                12 Jul 9 22:33
          hi_site2_from_site1.txt      12 Jul 9 22:33
dav:/webdav/dir1/> get hi_site3_from_site1.txt
Downloading '/webdav/dir1/hi_site3_from_site1.txt' to hi_site3_from_site1.txt:
Progress: [=====] 100.0% of 12 bytes succeeded.
dav:/webdav/dir1/> exit
Connection to '34.205.90.164' closed.
[root@site3-master ~]# ls
hi_site2_from_site3.txt  hi_site3_from_site1.txt

```

(a) Screenshot of the terminal, Site 3 Master Node: from Site 3, the user was able to upload data on its directory (dir3/hi_site2_from_site3.txt); also it was able to retrieve a file made for Site 3 (dir1/hi_site3_from_site1.txt). The file was downloaded in the working directory.

```

[giacomorsini2001@site4-master ~]$ ls
hi_site1_from_site4.txt
[giacomorsini2001@site4-master ~]$ cadaver http://34.205.90.164/webdav/
Authentication required for webdav on server '34.205.90.164':
Username: bdp1_project
Password:
dav:/webdav/> cd dir4
dav:/webdav/dir4/> ls
Listing collection '/webdav/dir4/': collection is empty.
dav:/webdav/dir4/> put hi_site1_from_site4.txt
Uploading hi_site1_from_site4.txt to '/webdav/dir4/hi_site1_from_site4.txt':
Progress: [=====] 100.0% of 18 bytes succeeded.
dav:/webdav/dir4/> ls
Listing collection '/webdav/dir4/': succeeded.
          hi_site1_from_site4.txt      18 Jul 9 22:33
dav:/webdav/dir4/> cd ../dir2
dav:/webdav/dir2/> ls
Listing collection '/webdav/dir2/': succeeded.
          *hi_site1.txt                12 Jun 18 16:13
          hi_site4_from_site2.txt      13 Jul 9 22:33
dav:/webdav/dir2/> get hi_site4_from_site2.txt
Downloading '/webdav/dir2/hi_site4_from_site2.txt' to hi_site4_from_site2.txt:
Progress: [=====] 100.0% of 13 bytes succeeded.
dav:/webdav/dir2/> exit
Connection to '34.205.90.164' closed.
[giacomorsini2001@site4-master ~]$ ls
hi_site1_from_site4.txt  hi_site4_from_site2.txt

```

(b) Screenshot of the terminal, Site 3 Master Node: from Site 3, the user was able to upload data on its directory (dir4/hi_site1_from_site4.txt); also it was able to retrieve a file made for Site 4 (dir2/hi_site4_from_site2.txt). The file was downloaded in the working directory.

Figure 13. Data transfer on WebDav from different instances made with different Cloud computing services (AWS, GCP).

4. The biological challenge

To test our infrastructure, we tried to solve a biological challenge, a computational task that requires an amount of resources that a Cloud platform can potentially provide. The challenge we chose was the following: aligning 300 million sequences returned by a Next-Generation Sequencing (NGS) experiment against the hg19 database. The volumes we attached to our sites provided us some test data, as well as the hg19 database and the alignment algorithm: the BWA algorithm. The test data were reads obtained from 3 medical patients; in particular, patient 1 and patient 2 with circa 550000 sequences, and patient 3 with circa 350000, all stored in `read.fa` files containing 1000 sequences each. The idea to solve the challenge was to run the alignments as jobs in the HTCondor batch system, which allows the master and nodes of each site to work together as a CPU farm, hence assigning jobs to each CPU iteratively. We ran the jobs both in site 1 and site 2, making an average of the performances, to then calculate the time-cost ratio of the entire processes. We then discussed how to enhance the process through data transfer between sites, as well as how to implement this challenge in different types of Cloud Infrastructures.

4.1 BWA algorithm, hg19 database and project data

BWA (Burrows-Wheeler Alignment Tool) is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. For all the algorithms, BWA first needs to construct the FM-index from the reference genome to facilitate fast searching. Once the index is constructed, BWA proceeds with the alignment step.

The hg19 database, also known as the GRCh37 (Genome Reference Consortium human genome build 37), is a widely used reference genome assembly for the human species. It represents a comprehensive and extensively studied version of the human genome.

The data we used in this section, as well as the BWA algorithm and the hg19 database, were already available in the volume created from the snapshot of the BDPI course, mounted locally in the Master nodes of site 1 and site 2 in the `\project_data` directory. A sub-directory from this directory contains almost 1.500.000 sequences coming from 3 different patients (patient 1, 2 and 3). We considered just the data from patient 1 and patient 2; in each patient directory, there are around 550 `read.fasta` files, and in each of these files there are 1000 `.fasta` sequences, which are the

reads coming from the NGS experiment. The single `read.fasta` files were considered as a jobs. We used the first 100 `read.fasta` files from patient 1 and 2 in site 1 and site 2, respectively, to align with the database through the BWA algorithm. From these sequences, we calculated the average time required to run the 100 jobs, and scaled it to the amount of time that would be required to align 300 million sequences of the same kind (that ideally would be stored in 300,000 `read.fasta` files of the same kind).

```
ec2-user@52.H: $ condor_status
```

| Name | OpSys | Arch | State | Activity | LoadAv | Mem | ActvtyTime |
|-------------------------------------|-------|--------|----------------|----------|--------|------------|------------|
| slot1@ip-172-31-47-82.ec2.internal | LINUX | X86_64 | Unclaimed Idle | 0.000 | 3909 | 0+00:19:36 | |
| slot2@ip-172-31-47-82.ec2.internal | LINUX | X86_64 | Unclaimed Idle | 0.000 | 3909 | 0+00:20:03 | |
| slot1@ip-172-31-47-189.ec2.internal | LINUX | X86_64 | Unclaimed Idle | 0.000 | 3909 | 0+00:19:38 | |
| slot2@ip-172-31-47-189.ec2.internal | LINUX | X86_64 | Unclaimed Idle | 0.000 | 3909 | 0+00:20:03 | |

| Machines | Owner | Claimed | Unclaimed | Matched | Preempting | Drain |
|--------------|-------|---------|-----------|---------|------------|-------|
| X86_64/LINUX | 4 | 0 | 0 | 4 | 0 | 0 |
| Total | 4 | 0 | 0 | 4 | 0 | 0 |

```
ec2-user@52.H: $
```

Figure 14. Screenshot of the terminal, Site 2 Master Node: HTCondor status. The 4 CPUs (2 Master node, 2 Worker Node) were listed correctly, and their status was unclaimed, as we hadn't submitted any jobs yet.

4.2 Preparing and submitting jobs to HTCondor

To run the test with HTCondor, we first had to create some files:

- Index
- A file to align the sequences: `align.py`
- A file to give instructions to HTCondor: `bwa_batch.job`

HTCondor works as job scheduler, it enables the management and execution of computing jobs in background on distributed computing resources; the scheduler assigns the submitted jobs based on available resources (in our case, the CPUs). To do that, HTCondor monitors the availability and status of computing resources within the distributed environment. It also track the jobs progress. To be able to run our test and submit our 100 alignments as jobs in HTCondor, we had to modify the two already existing files: the alignment file, `align.py`, a python script that enables the alignment trough the BWA tool, and the `bwa_batch.job` file, which is a job description file written in the HTCondor job submission language; this file specifies the details of our job, such as the executable, input files, output locations, job universe and paths. The `bwa_batch.job` was taken as argument by the `condor_submit` command.

- `bwa_batch.job`: the file contains 5 sections (Figure 15):
 - Section 1 (executable): the `align.py` script, and a variable so to make the process analyse each of the 100 sequences, depending on the `$Process` variable, which indicates the process ID of an individual job within an HTCondor job cluster (starting at 0).
 - Section 2 (input sandbox): a sandbox refers to a restricted and isolated environment where untrusted or potentially harmful code can be executed safely. In here, we specified as input the BWA program and as input and argument, our specific sequence file (i.e., in the first individual job, `$Process=0`, and `argument=read_1.fa`).
 - Section 3 (output sandbox): where the output files were specified. They were generated in 6 different formats:
 - * `.sai` file is a binary file returned by the command `bwa aln`.
 - * `.sam.gz` file is a human readable compressed version of the `.sai` file made by the command `bwa samse` (SAM format is a text format for storing sequence data in a series of tab delimited ASCII columns).

```

##### The program that will be executed #####
Executable = align.py

n = $(Process)+1
##### Input Sandbox #####

#Input      = read_${INT}(n).fa
#Can contain standard input

transfer_input_files = bwa, read_${INT}(n).fa

## Arguments that will be passed to the executable ##

Arguments = read_${INT}(n).fa

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

Log        = read_${INT}(n).log
# will contain condor log

Output     = read_${INT}(n).out
# will contain the standard output

Error      = read_${INT}(n).error
# will contain the standard error

transfer_output_files = read_${INT}(n).sam.gz, read_${INT}(n).sai, read_${INT}(n).md5

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

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Universe   = vanilla

#####

Queue 100

```

Figure 15. Site 2 Master Node, align.py file

- * `md5.txt` is the result of the checksum computation by the `md5sum` software, used to verify that the files have not changed (data integrity) during the file transferring.
 - * `.log` file is the file that records all events occurring in HTCondor cluster (i.e. in which machine a job is running, the allocated memory, the starting time etc).
 - * `.error` file is the file storing the standard errors.
 - * `.out` file contains the 'print' statement of the executable `align.py`.
- Section 4 (control variable): allowed us to define which files were transferred from the Scheduler to the Worker Nodes and viceversa and on which circumstances, by exploiting the Condor's File Transfer Mechanism. HTCondor transfers the executable and the input file from the Scheduler to the Worker Node where the job is executed. Then, the output files were transferred back to the machine where the job was submitted only when the job exited on its own. A universe in HTCondor defines an execution environment for a job: the Vanilla universe is usually used as default because it has the fewest restrictions on the jobs.
 - Section 5 (queue): the number of jobs to be queued.
- `align.py` script (Figure 16): the alignment between the input sequence (`queryname`) and the reference genome (`hg19bwaidx`) is calculated with `bwa-aln`. The `.sai` output file is converted into the human-readable version, `.sam` which is then gunzipped. The checksums of the `.sai` and `.sam` files are computed. Finally, the time module is imported to calculate the running time and printed in the `.out` file.

Lastly, the test was run with the command `condor_submit bwa_batch.job`. The tasks were dispatched by the Master Node and by using the `condor_q` command it was possible to check the status of the enqueued jobs. It is possible to inspect the workflow of the application by using `condor_q -analyse` (Figure 17).

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

# Control variables
#####

#bwa aln -t 1 /project_data/BDP1_2023/hg19/hg19bwaidx myread.fa > myread.sai
#bwa samse -n 10 /project_data/BDP1_2023/hg19/hg19bwaidx myread.sai myread.fa > myread.sam

start_time= timer()
dbpath = "/project_data/BDP1_2023/hg19/"
dbname = "hg19bwaidx"

queryname = sys.argv[1]

out_name = queryname[:3]

md5file = out_name+".md5"

command = "./bwa aln -t 1 " + dbpath + dbname + " " + queryname + " > " + out_name + ".sai"
print "launching command: " , command
os.system(command)

command = "./bwa samse -n 10 " + dbpath + dbname + " " + out_name + ".sai " + queryname + " > " + out_name + ".sam"
print "launching command: " , command
os.system(command)

print "Creating md5sums"
os.system("md5sum " + out_name + ".sai " + " > " + md5file)
os.system("md5sum " + out_name + ".sam " + " >> " + md5file)

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

total_time= timer() - start_time
print("Running time =" + str(total_time))
print "exiting"

exit(0)
```

Figure 16. Site 2 Master Node, align.py file

```
ec2-user@52_M: $ condor_q

-- Schedd: ip-172-31-47-82.ec2.internal : <172.31.47.82:9618?... @ 06/27/23 13:34:20
OWNER   BATCH_NAME   SUBMITTED   DONE   RUN    IDLE   TOTAL   JOB_IDS
ec2-user ID: 7       6/27 13:34   -       4      96     100   7.0-99

Total for query: 100 jobs; 0 completed, 0 removed, 96 idle, 4 running, 0 held, 0 suspended
Total for ec2-user: 100 jobs; 0 completed, 0 removed, 96 idle, 4 running, 0 held, 0 suspended
Total for all users: 100 jobs; 0 completed, 0 removed, 96 idle, 4 running, 0 held, 0 suspended

ec2-user@52_M: $ condor_status
Name                                           OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
slot1@ip-172-31-47-82.ec2.internal            LINUX      X86_64    Claimed    Busy      0.050 3909  0+00:00:03
slot2@ip-172-31-47-82.ec2.internal            LINUX      X86_64    Claimed    Busy      0.060 3909  0+00:00:03
slot1@ip-172-31-47-189.ec2.internal            LINUX      X86_64    Claimed    Busy      0.040 3909  0+00:00:03
slot2@ip-172-31-47-189.ec2.internal            LINUX      X86_64    Claimed    Busy      0.040 3909  0+00:00:03

Machines Owner Claimed Unclaimed Matched Preempting Drain
X86_64/LINUX      4      0      4      0      0      0      0
Total             4      0      4      0      0      0      0
ec2-user@52_M: $
```

Figure 17. Screenshot of the terminal, Site 2 Master Node: the 100 jobs were correctly submitted. 96 of them were queued, while 4 were being executed. The 4 CPUs were claimed.

5. Work with containers: Docker

We decided to try the aforementioned biological challenge using a container image. To do so, we installed docker, created the [Dockerfile](#) to build our image and built the container from said image. The specific lines of code used can be found in our [GitHub repository](#).

Once this was done, we tried to use udocker, which was unsuccessful. The Dockerfile used to build the image looked like this:

```
FROM ubuntu:latest                                #create from existing image
RUN apt-get update                                #update system
RUN apt-get install -y python3                    #install python3
COPY align.py /align.py                          #copy the applications
COPY bwa /bwa
WORKDIR /                                          #set workdir to app dir
ENTRYPOINT ["/bin/python3", "align.py"]          #where and what to run
```

The image was built and pushed to DockerHub. Then, it was pulled to udocker, we created a container from said image and we ran it, using as parameters the volumes needed to access the working directory and the index database and the name of the fasta file of interest. However, it returned an error, stating that the index was not found (Figure 18).

```
ec2-user@SL_H:/project_data/docker/udocker$ ./udocker run bwa_udocker -v /project_data/BDP1_2023/hg19/:/project_data/BDP1_2023/hg19/ -v /project_data/docker/udocker:/workdir read_1.fa
Warning: non-existing user will be created

*****
*                               *
*   STARTING ff81d324-2ba3-3ed6-8069-337d99ca2fee   *
*                               *
*****
executing: python3
launching command: ./bwa aln -t 1 /project_data/BDP1_2023/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] fail to locate the index
launching command: ./bwa samse -n 10 /project_data/BDP1_2023/hg19/hg19bwaidx read_1.sai read_1.fa > read_1.sam
[bwa_sai2sam_sel] fail to locate the index
Creating md5sums
gzipping out text file
launching command: gzip read_1.sam
gzip: read_1.sam.gz already exists; do you wish to overwrite (y or n)? y
The program took 10.954543 seconds to run
exiting
```

Figure 18. Screenshot of the udocker run command (with its parameters) and its output.

After trying manually with the prompt, we tried to run it using bwa_udocker . job, but it still raised the same error, that the index was not found (Figure 19a). However, looking at the .out file, the path to the index appeared to be correct (Figure 19b).

```
[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] fail to locate the index
read_10.error (END)
```

(a) Screenshot of an example error file, after running the udocker container with HTCondor.

```
launching command: ./bwa aln -t 1 /project_data/BDP1_2023/hg19/hg19bwaidx read_10.fa > read_10.sai
launching command: ./bwa samse -n 10 /project_data/BDP1_2023/hg19/hg19bwaidx read_10.sai read_10.fa > read_10.sam
Creating md5sums
gzipping out text file
launching command: gzip read_10.sam
The program took 0.034293 seconds to run
exiting
read_10.out (END)
```

(b) Screenshot of an example out file, after running the udocker container with HTCondor.

Figure 19. Output files after running udocker with HTCondor.

6. Results: time-costs estimation

The results of our test indicate that the average running time of the 100 jobs was of 63,41 seconds; this number was obtained by doing the average of the average job running time for both tests (both sites). Considering the two sites, the running time for each job ranged between 33 and 167 seconds. Site 1 completed the task in roughly 20 minutes while site 2 in 23. The t2.large instances we used cost 0.055\$ each considering the yearly plan; the price varies according to the plan (Table 1). All the prices used as reference were taken from the [AWS webpage](#) in July 2023. We always considered the yearly price as it is more convenient than the on demand price.

Table 1. Prices list for 4 different instance types

| Instance type | Price on demand | Yearly plan price | 3Y plan price | vCPUs |
|---------------|-----------------|-------------------|---------------|-------|
| t2.large | 0,0928 | 0,055 | 0,037 | 2 |
| t2.2xlarge | 0,3712 | 0,219 | 0,148 | 8 |
| t3.2xlarge | 0,3341 | 0.199 | 0.133 | 8 |
| c5d.12xlarge | 2,88 | 0,0326 | 0,0256 | 48 |

Price table showing different renting plans for the instance types used and considered in this work.

¹ All prices are in USD.

In tables 2 and 3, we calculated the time and price that our system would need to align 300 million sequences, our biological challenge. Moreover, we also considered the time and cost of completing this task with more t2.large instances (Table 2) and with different types of instances (Table 3):

Table 2. Time and costs for the biological challenge using t2.large instance type

| Instance price | Instances | CPUs | Aligned seq | Avg run time | Total time | Total cost |
|----------------|-----------|------|-------------|--------------|------------|------------|
| 0,055 | 2 | 4 | 100 | 63,41 | 26 min | 0,05 |
| 0,055 | 3 | 6 | 100 | 63,41 | 18 min | 0,05 |
| 0,055 | 3 | 6 | 30000000 | 63,41 | 37 days | 145,31 |
| 0,055 | 10 | 20 | 300000000 | 63,41 | 11 days | 145,31 |
| 0,055 | 50 | 100 | 300000000 | 63,41 | 53 hrs | 145,31 |
| 0,055 | 100 | 200 | 300000000 | 63,41 | 26 hrs | 145,31 |
| 0,055 | 500 | 1000 | 300000000 | 63,41 | 5 hrs | 145,31 |

The first two rows represent our site 1 and 2; then row by row the number of sequences to align and of CPUs increases, so to show the time and the costs of different infrastructure sizes.

¹ All prices are in USD and the instance type is always t2.large. Average running time is in seconds.

IMPORTANT: the aforementioned costs do not include the costs of internet connection and electric energy; even if the infrastructure is in the Cloud and hosted by Amazon/Google services, for the user the costs is still not 0, in particular considering a yearly activity plan and the usage of the infrastructure for multiple works of the same kind. Moreover, we are not considering the amount of space that would be required to host all of the data, therefore the costs only depend on that.

Table 3. Time and costs for the biological challenge using 3 different instance types

| Instance type | Instance price | Instances | CPUs | Aligned seq | Avg run time | Total time | Total cost |
|---------------|----------------|-----------|------|-------------|--------------|------------|------------|
| t2.2xlarge | 0,219 | 3 | 24 | 300000000 | 63,41 | 9 days | 144,65 |
| t2.2xlarge | 0,219 | 10 | 80 | 300000000 | 63,41 | 3 days | 144,65 |
| t2.2xlarge | 0,219 | 50 | 400 | 300000000 | 63,41 | 13 hrs | 144,65 |
| t2.2xlarge | 0,219 | 100 | 800 | 300000000 | 63,41 | 7 hrs | 144,65 |
| t3.2xlarge | 0,199 | 3 | 24 | 300000000 | 63,41 | 9 days | 131,44 |
| t3.2xlarge | 0,199 | 10 | 80 | 300000000 | 63,41 | 3 days | 131,44 |
| t3.2xlarge | 0,199 | 50 | 400 | 300000000 | 63,41 | 13 hrs | 131,44 |
| t3.2xlarge | 0,199 | 100 | 800 | 300000000 | 63,41 | 7 hrs | 131,44 |
| c5d.12xlarge | 0,0326 | 3 | 144 | 300000000 | 63,41 | 2 days | 3,59 |
| c5d.12xlarge | 0,0326 | 10 | 480 | 300000000 | 63,41 | 11 hrs | 3,59 |
| c5d.12xlarge | 0,0326 | 30 | 1440 | 300000000 | 63,41 | 4 hrs | 3,59 |
| c5d.12xlarge | 0,0326 | 50 | 2400 | 300000000 | 63,41 | 2 hrs | 3,59 |

Time and costs table considering different instances type to solve our challenge.

¹ All prices are in USD. Average running time is in seconds.

7. Discussion

The aim of our job was to create a Cloud infrastructure using Amazon Web service, Elastic Computing console, and perform file management with various tools. The ultimate goal was to solve a computational challenge, in particular the alignment of 300 million genomic sequences. Our infrastructure was used to run a smaller, more manageable test to calculate how much time and money would indeed cost to execute our task, with a bigger infrastructure of the same kind. The data we worked with were taken from a volume given by the professor of the course.

It is to be mentioned again that during the tests no other costs (internet, energy, storage space) than the instances prices were calculated. We are also did not take into consideration any other instance settings other than the CPU amount (such as the bursting) and made the assumptions of having no problems related to sharing the data trough network.

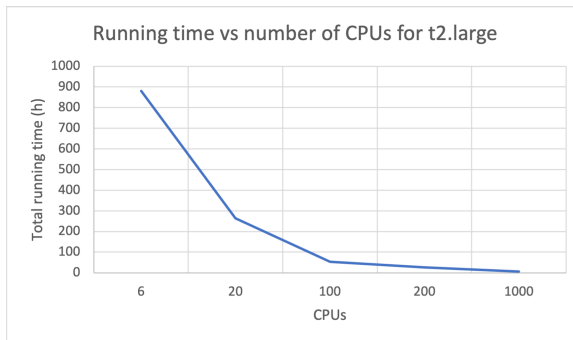


Figure 20. Plot of the running time vs the number of CPUs for the t2 large instance. We can see how the running time decreases as the number of CPUs increases. The correlation is exponential.

From the tests we have run, it is clear that the infrastructure we created is unable to complete the task in a reasonable amount of time, and this was expected: aligning 300 million sequences against a database is a high challenging computational task and our infrastructure, considering all the sites, is composed just by 5 nodes for a total of 10 CPUs, as the instances are all of the t2 large type (not considering the Google instances). Our sites should work in parallel, dividing the data to analyze

between the two. Considering the enlargement of the infrastructure by having more instances of the same type, we have seen that the time needed to complete the challenge decreases while the costs increase reaching a plateau: if we increase the number of instances, the price increases, but the total run time decreases, decreasing the price; so overall, the price remains the same. The performances of infrastructure that considers more CPUs from the t2large instance type are shown in Figure 20.

A better performance would have been obtained by using the t2.xlarge instance type, as it provides more CPUs, although it costs more (Table 1). The amount of time needed to complete the challenge decreases as well as the price, even if this decrease is not very perceptible (Table 2 and 3).

If we change instance type to consider the t3 family, in particular t3.xlarge, which provides instances with 8 CPUs like the t2.xlarge, we have the same total run time, while the price goes through a significant reduction. Here, as well, it is worth mentioning that the number of instances – time needed – costs relationship reaches a plateau, because the less time we use the more instances we require and the relationship is linear.

The t2 and t3 type instances have different specifications: in particular, while t2s are generally less expensive, t3s have better baseline performances and bursting capabilities (CPU credits are earned over time when the instance operates below its baseline performance; these credits can be used to burst above the baseline when the workload demands it). We can affirm that using a t3 type instances would have been more favorable. Both the instances types are, however, not suitable for this challenging tasks, and this report demonstrates that other, more suited and optimized for high-performance instances, should be used. We investigated, for example, the usage of the c5 instances

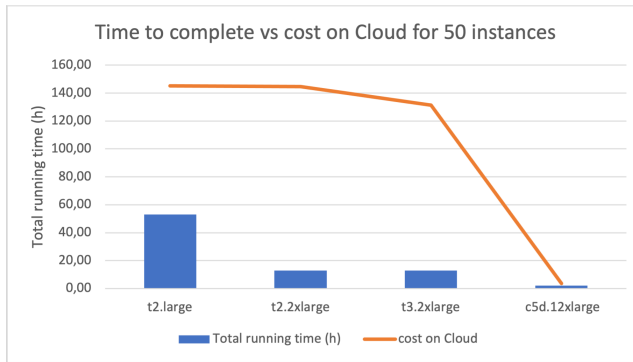


Figure 21. Plot of the time to complete the challenge for each instance type vs the cost of the instances type on the cloud for the t2 large instance. We made this plot considering data having 50 instances of each type. It is clear that the c5d.12xlarge instance has better performances overall.

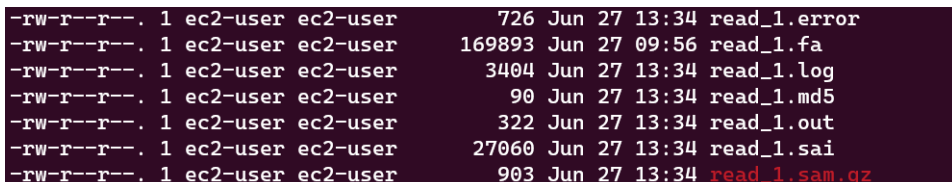
type, in particular the c5d.12xlarge. These instances are part of the AWS Compute Optimized family and offer a high ratio of CPU resources to memory. They provide excellent compute performances and are suitable for applications that require substantial computational power. By implementing them in our infrastructure, the time and the costs would decrease significantly. Hence, this type of instances, as well as others designed for high challenging computational tasks and high performance (e.g m5, r5, p3), would be way more effective. The performances of the different instance types are shown in Figure 21.

7.1 Managing the workload

Although not adequate for its dimensions, the tools that we implemented in the infrastructure we built are suitable for managing works such as our computational challenge of aligning 300 million sequences: NFS (Network File System), HTCondor and WebDav. The principles that these tools use and that we have demonstrated are valid also for managing data this big.

Lets consider the c5d.12xlarge instances we have mentioned before: to run the challenge in roughly 2 hours we need about 50 instances (Table 3). We should therefore divide these instances in more sites, considering multiple nodes. Depending on the situation we are in:

- if we are working, for example, in a lab and we have no other supporting infrastructures except a single site cloud infrastructure, we may have all the nodes under the same security group and subnet.
- if we have the chance to use multiple sites as, for example, a group of labs, we should have different sites under different security groups.



```

-rw-r--r--. 1 ec2-user ec2-user      726 Jun 27 13:34 read_1.error
-rw-r--r--. 1 ec2-user ec2-user  169893 Jun 27 09:56 read_1.fa
-rw-r--r--. 1 ec2-user ec2-user    3404 Jun 27 13:34 read_1.log
-rw-r--r--. 1 ec2-user ec2-user     90 Jun 27 13:34 read_1.md5
-rw-r--r--. 1 ec2-user ec2-user    322 Jun 27 13:34 read_1.out
-rw-r--r--. 1 ec2-user ec2-user   27060 Jun 27 13:34 read_1.sai
-rw-r--r--. 1 ec2-user ec2-user    903 Jun 27 13:34 read_1.sam.gz

```

Figure 22. Screenshot of the terminal, Site 1 Master: example of input and output files with respective weights in bytes, shown with the `ls -l` command

In each case, some nodes should be set as storage system(s), so to host the input data (the 300 million sequences, proportionally divided between sites): these nodes should have high storage capacity, maybe trough the implementation of some additional empty volumes. The same should be done for nodes that will be used to store the outputs. We should consider again that the sequences are stored in 300 thousand `read.fasta` files, each `read.fasta` weights approximately 170 KB, so then we have roughly 51 GB of data, plus a couple of GB more for the index database, `align.py` and `bwa_batch.job`. The output files (`.sai .sam.gz .out .md5 .error .log`) weight about 32,7 KB, hence the total output would be roughly 10 GB (considering that the `.sam` files are gunzipped).

With HTCondor we can set then 3 types of nodes: a manager node which works as a negotiator and a scheduler, able to partition the submitted job between the worker nodes, which will act as executors; the output should be then collected in the output nodes, acting as collectors. Trough the NFS, we can share the data from the input nodes containing the input sequences to the manager. In the case we are working with multiple sites in multiple places, we would share data with the WebDav system, so that, for example, the lab doing the NGS experiment can share the reads with the other labs; the WebDav tool would be necessary also if we have computing infrastructures under different security groups in the same facility, for example the Biotech lab infrastructure and the Bioinformatics one. In any case, considering these type of instances, this number of instances and this challenge, the cost of such computations would be of around 3,60\$.

8. Supplementary materials

All files that we created, as well as all the commands used, can be found in our GitHub repository: <https://github.com/torresmasdeu/bdp1-final-project>.