

# Building Cloud IaaS infrastructures and Computing Models

## Introduction to Big Data Processing Infrastructures – Report







### 1. Aim

The aim of this work is to implement a computing model able to align 300M sequences against the hg19 database, using the Burrows-Wheeler Aligner (BWA) software. Both implementation and evaluation in terms of time and cost are discussed. In order to address the computational challenge discussed here the following considerations were considered:

- An alignment of 1000 sequences creates an ascii file of 2.7GB.
- Inside the patient direction there are 1444 read files of 170KB each (more or less), i.e. a total of 245MB.
- Each patient file has 1000 sequences.

### 2. Building the cloud infrastructure

The aim of the project will be achieved building a cloud computing infrastructure. The infrastructure consists in an HTC site, a HPC site, and storage site. For each site, a different Google Cloud project will be created in order to simulate geographically distributed sites.

RECENT	STARRED	ALL
Name		ID
✓ ☆  <a href="#">site1</a> 	site1-355609	
☆  <a href="#">site2</a> 	site2-355609	
☆  <a href="#">site3</a> 	seraphic-rarity-355609	

#### 2.1. High Throughput Computing (HTC) site

The HTC site provides CPU (batch farm) and storage (one shared volume). Therefore, three instances were created: one master node (main), followed by two slaves (worker1, worker2). All these instances have 2 vCPUs and 8GB of memory (**e2-standard-2**).

<input type="checkbox"/>	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	✓	<a href="#">master</a>	us-central1-a			10.128.0.2 ( <a href="#">nic0</a> )	34.27.207.242 ( <a href="#">nic0</a> )	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">worker1</a>	us-central1-a			10.128.0.3 ( <a href="#">nic0</a> )	34.28.142.154 ( <a href="#">nic0</a> )	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">worker2</a>	us-central1-a			10.128.0.4 ( <a href="#">nic0</a> )	35.194.3.24 ( <a href="#">nic0</a> )	SSH ▾ ⋮

An additional volume (disk of 50GB) was created and attached to the master node.

<input type="checkbox"/>	Status	Name ↑	Type	Size	Architecture	Zone(s)	In use by	Snapshot schedule	Actions
<input type="checkbox"/>	✓	<a href="#">htcdisk</a>	Standard persistent disk	50 GB	—	us-central1-a	<a href="#">master</a>	<a href="#">default-schedule-1</a>	⋮
<input type="checkbox"/>	✓	<a href="#">master</a>	Standard persistent disk	20 GB	x86/64	us-central1-a	<a href="#">master</a>	None	⋮
<input type="checkbox"/>	✓	<a href="#">worker1</a>	Standard persistent disk	20 GB	x86/64	us-central1-a	<a href="#">worker1</a>	None	⋮
<input type="checkbox"/>	✓	<a href="#">worker2</a>	Standard persistent disk	20 GB	x86/64	us-central1-a	<a href="#">worker2</a>	None	⋮

The disk was also partitioned; a new file system was created on top of the disk, followed by the creation of a new mount point for the same. Thus, the **fstab** file was modified in order to change permission and make it available to every instance in the site.

```

fdisk /dev/sdb
p
n
all default settings
# To check if the disk was created with that defaulting settings
p
# To create a partition table to the disk
w
# To check that everything went correctly (check also for the partition table)
fdisk -l
# To create a file system (on top of the partition)
mkfs.ext4 /dev/sdb1
# To mount the file system and access it, we need to create a mount point for it
# Creating a dir
mkdir /data
# Editing the following file
vim /etc/fstab #it contains all files system that are mounted and are available to the user
/dev/sdb1 /data ext4 defaults 0 0 #adding the following line at the end of the file
#equivalent command: mount -t ext4 /dev/sdb1 /data (however you should run it every time you reboot the VM)
# Checking if the editing went correctly
cat /etc/fstab
# To mount the file system
mount -a

```

Access to the disk of both slaves was performed by installing the NFS server. The file system will be therefore exposed to the NFS client installed in these instances. In order to achieved that, the server configuration `etc/exports` file was edited. The mount point of the file system, the private ID of the slaves were both added in the file.

```

[root@main ~]# vim /etc/exports
[root@main ~]# cat /etc/exports
/data 10.128.0.3(rw,no_root_squash,sync,no_wdelay)
[root@main ~]# exportfs -r
[root@main ~]# exportfs
/data      10.128.0.3

```

The directory `/data` was also created in the workers. To mount the remote file system, the private IP address of the main node was added to the `fstab` file.

The batch system was managed by HTCondor (installed in all instances):

```

# HOW TO INSTALL AN HT-CONDOR MACHINE #
# Going in the master-server shell
# Installing dependencies
yum install wget #not needed, already installed
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
yum localinstall epel-release-latest-7.noarch.rpm #apparently this one too (?)
yum clean all

# Installing condor repos and packages
wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo
cp htcondor-stable-rhel7.repo /etc/yum.repos.d/
yum install condor-all

```

HTCondor was configured editing its configuration file `condor-config`. The following lines were added:

```

## Install the minicondor package to run HTCondor on a single node
CONDOR_HOST = 10.128.0.2

# on the master
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD, SCHEDD

# on the nodes
# #DAEMON_LIST = MASTER, STARTD
#
#
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *

```

Condor was enabled and started:

```
[root@main ~]# condor_status
```

Name	OpSys	Arch	State	Active
slot1@mn-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle
slot2@mn-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle
slot1@wn1-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle
slot2@wn1-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle
slot1@wn2-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle
slot2@wn2-htc.us-central1-a.c.smart-sandbox-347909.internal	LINUX	X86_64	Unclaimed	Idle

	Machines	Owner	Claimed	Unclaimed	Matched	Preempting	Drain
X86_64/LINUX	6	0	0	6	0	0	0
Total	6	0	0	6	0	0	0

The **condor-config** file of both slaves was also edited. The private IP of the master node was added:

```
# CHANGE THE FOLLOWING IP TO YOUR MASTER IP
CONDOR_HOST = 10.128.0.2 #this is your master private IP
# on the master
# DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD, SCHEDD
#on the nodes
DAEMON_LIST = MASTER, STARTD
#
#
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *
```

6 jobs were run (changing the **QUEUE** line in the **bwa\\_batch.job** to 6). At first, jobs were run in all instances:

```
[root@main ~]$ condor_history 11
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	COMPLETED	CMD
11.5	vvuong	2/14 05:34	0+00:07:03	C	2/14 05:41	/home/vvuong/condor/hg/align.py several.fa
11.4	vvuong	2/14 05:34	0+00:07:03	C	2/14 05:41	/home/vvuong/condor/hg/align.py several.fa
11.3	vvuong	2/14 05:34	0+00:06:41	C	2/14 05:41	/home/vvuong/condor/hg/align.py several.fa
11.2	vvuong	2/14 05:34	0+00:06:41	C	2/14 05:41	/home/vvuong/condor/hg/align.py several.fa
11.0	vvuong	2/14 05:34	0+00:05:49	C	2/14 05:40	/home/vvuong/condor/hg/align.py several.fa
11.1	vvuong	2/14 05:34	0+00:05:47	C	2/14 05:40	/home/vvuong/condor/hg/align.py several.fa

then on two nodes:

```
[root@main ~]$ condor_history 12
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	COMPLETED	CMD
12.4	vvuong	2/14 05:51	0+00:05:09	C	2/14 06:01	/home/vvuong/condor/hg/align.py several.fa
12.5	vvuong	2/14 05:51	0+00:05:09	C	2/14 06:01	/home/vvuong/condor/hg/align.py several.fa
12.3	vvuong	2/14 05:51	0+00:06:33	C	2/14 05:57	/home/vvuong/condor/hg/align.py several.fa
12.2	vvuong	2/14 05:51	0+00:06:33	C	2/14 05:57	/home/vvuong/condor/hg/align.py several.fa
12.0	vvuong	2/14 05:51	0+00:05:27	C	2/14 05:56	/home/vvuong/condor/hg/align.py several.fa
12.1	vvuong	2/14 05:51	0+00:05:27	C	2/14 05:56	/home/vvuong/condor/hg/align.py several.fa

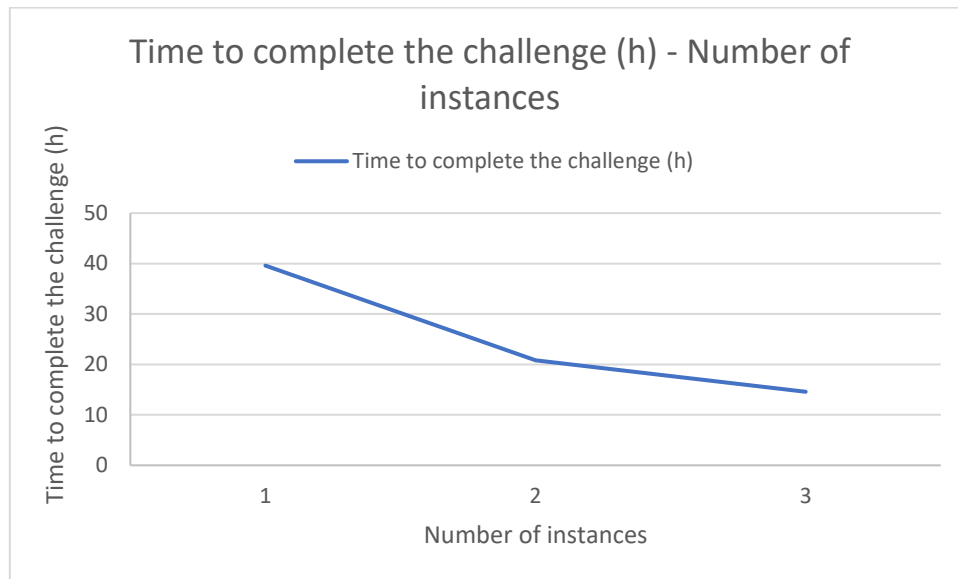
then in a single node:

```
[root@main ~]$ condor_history 13
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	COMPLETED	CMD
13.5	vvuong	2/14 06:45	0+00:06:07	C	2/14 07:04	/home/vvuong/condor/hg/align.py several.fa
13.4	vvuong	2/14 06:45	0+00:06:07	C	2/14 07:04	/home/vvuong/condor/hg/align.py several.fa
13.3	vvuong	2/14 06:45	0+00:06:23	C	2/14 06:58	/home/vvuong/condor/hg/align.py several.fa
13.2	vvuong	2/14 06:45	0+00:06:23	C	2/14 06:58	/home/vvuong/condor/hg/align.py several.fa
13.1	vvuong	2/14 06:45	0+00:06:47	C	2/14 06:51	/home/vvuong/condor/hg/align.py several.fa
13.0	vvuong	2/14 06:45	0+00:06:47	C	2/14 06:51	/home/vvuong/condor/hg/align.py several.fa

By running BWA with 6 jobs of 1M sequences each, a reduction in terms of running times was observed at the increasing number of cores. Assuming a linear increase, the time to complete the entire challenge varies from 39,6 hours with a single instance to 14,58 hours with three. As the hourly cost for an **e2-standard-2** machine is around 0,07\$, the total cost to run the whole computational challenge is 2,77\$, 2,92\$ and 3,06\$, respectively with 1, 2, 3, instances.

Numer of instances	Number of cores	Time for 6 jobs for 1M alignments (s)	Time to complete the challenge (h)	Expected cost (\$)	Total (\$)
1	2	1140	39,6	1*39.6h*0,07\$	2,772
2	4	600	20,83	2*20,83h*0,07\$	2,9162
3	6	420	14,58	3*14,58h*0,07\$	3,0618



## 2.2. High Performance Computing (HPC) site

The HPC site provides CPU only for parallel jobs. In order to achieve this goal, a single multi-core instance was created (acting as both master and slave). This instance has 4 vCPUs and 4GB in memory (**e2-highcpu-4**). As the node already had 50GB in memory available, no additional disk was attached to it.

<input type="checkbox"/>	Status	Name <span>↑</span>	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	✓	<a href="#">main</a>	us-central1-a			10.128.0.2 ( <a href="#">nic0</a> )	34.121.157.250 ( <a href="#">nic0</a> )	SSH <span>▼</span> <span>⋮</span>

As HTCondor is not necessary in this case, BWA is directly installed on the master (BWA runs only on the main node):

```
# First, we need to download the file containing all script for the exercise
wget https://pandora.infn.it/public/bdp12022tgz/d1/BDP1_2022.tgz

# Remember, when moving files, always check the checksum
md5sum BDP1_2022.tgz

# To uncompress and execute the tarball
tar -xvzf BDP1_2022.tgz # -x: execute
                        # -v: verbose mod
                        # -z: unzip

# Copying the tar file into the root directory
cp /data/BDP1_2022/hg19/bwa-0.7.15.tar .

# Installing the following applications
yum install gcc gcc-c++
yum install zlib
yum install zlib-devel

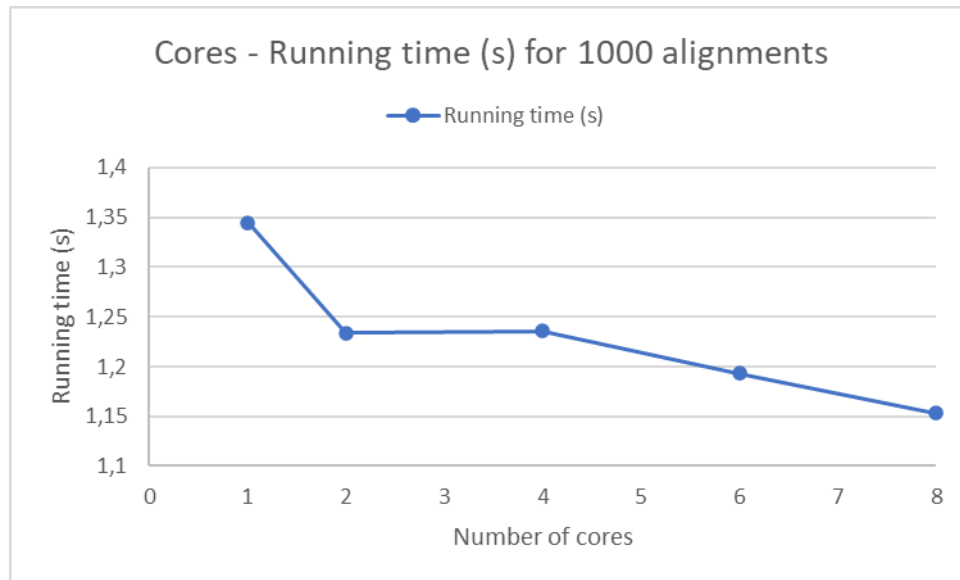
# Dealing with the tarball
tar -xvf bwa-0.7.15.tar

# Moving dir
cd bwa-0.7.15/

# Making the application executable
make #install it first with yum install make
```

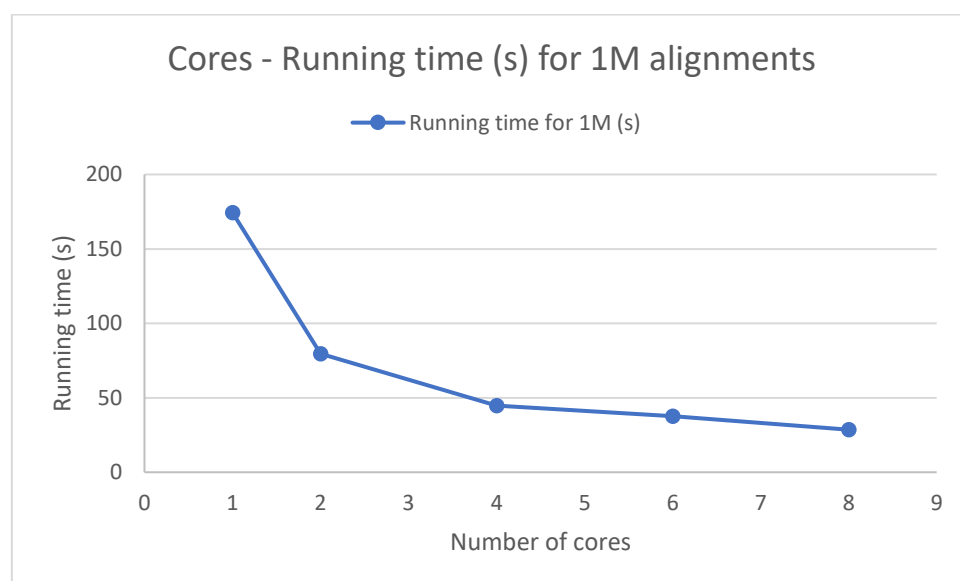
BWA was tested running 1000 alignments:

Number of cores	Running time (s)	Efficiency	Speedup
1	1,345	/	/
2	1,234	0,5045	1,009
4	1,236	0,252	1,009
6	1,193	0,187	1,123
8	1,153	0,145	1,162



By running BWA on 1000 sequences, the running time equals to 1,345 seconds (time to upload the database, read it for the first time: from now on, it will be stored in the cache and, thus, this pre-processing is not required anymore). Looking at the plotted results, the speedup is modest due to the dimension of the job. Indeed, according to the Amdahl's law the pre-processing phase (setting up the environment, loading the data, transforming the data) is scalar. This phase affects only one processor. From this moment on, the computational procedure can be parallelized, which means that the running time can be reduced by incrementing the number of cores. Therefore, in order to exploit the parallelization feature of BWA, the alignment tool works best on bigger jobs. 1M alignments were run.

The following plot indeed shows that the running time decrease rapidly by incrementing the number of cores. Notice also the speedup and efficiency.



Number of cores	Running time for 1M (s)	Efficiency	Speedup
1	174,373	/	/
2	79,514	1,096	2,192
4	44,812	0,972	3,89
6	37,605	0,773	4,636
8	28,597	0,762	6,097

### 2.3. Storage site

This site only provides storage. A single **e2-micro** (2 vCPUs, 1GB in memory) instance was therefore created.

<input type="checkbox"/>	Status	Name <span>↑</span>	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	✓	<a href="#">main</a>	us-central1-a			10.128.0.2 ( <a href="#">nic0</a> )	34.123.22.204 ( <a href="#">nic0</a> )	SSH <span>▼</span> <span>⋮</span>

As the storage stores all output files of the alignment jobs run, a volume of 50GB was created, attached, partitioned and mounted as seen in section 2.1.

<input type="checkbox"/>	Status	Name <span>↑</span>	Type	Size	Architecture	Zone(s)	In use by	Snapshot schedule	Actions
<input type="checkbox"/>	✓	<a href="#">main</a>	Standard persistent disk	20 GB	x86/64	us-central1-a	<a href="#">main</a>	None	⋮
<input type="checkbox"/>	✓	<a href="#">storedisk</a>	Standard persistent disk	50 GB	—	us-central1-a	<a href="#">main</a>	<a href="#">default-schedule-1</a>	⋮

As the output file for 1000 alignments is 2.7GB, completing the entire challenge (300 000 alignments) will require at least 810 000GB of available storage.

To transfer data from all three sites, the data transfer tool WebDAV was installed here:

```
# HOW TO INSTALL WEBDAV #
# Starting on the server
cat /etc/yum.repos.d/epel.repo
yum install httpd #installing Apache
sed -i 's/^#&/g' /etc/httpd/conf.d/welcome.conf #disabling Apache's default welcome page:
sed -i "s/Options Indexes FollowSymLinks/Options FollowSymLinks/" /etc/httpd/conf/httpd.conf #preventing the Apache web server from displaying files within the web directory
# To start the service
systemctl start httpd.service
httpd -M | grep dav
mkdir /var/www/html/webdav
chown -R apache:apache /var/www/html
chmod -R 755 /var/www/html
# To create a user account, which will be used to log into your WebDAV server
htpasswd -c /etc/httpd/.htpasswd user001
chown root:apache /etc/httpd/.htpasswd
chmod 640 /etc/httpd/.htpasswd
```

The `webdav.conf` file was edited:

```
DavLockDB /var/www/html/DavLock
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/webdav/
    ErrorLog /var/log/httpd/error.log
    CustomLog /var/log/httpd/access.log combined
    Alias /webdav /var/www/html/webdav
    <Directory /var/www/html/webdav>
        DAV On
        AuthType Basic
        AuthName "webdav"
        AuthUserFile /etc/httpd/.htpasswd
        Require valid-user
    </Directory>
</VirtualHost>
```

`cadaver` was installed to access the WebDAV server using the internal ID of the instance:

```
[root@main ~]# cadaver http://10.128.0.2/webdav/  
Authentication required for webdav on server `10.128.0.2':  
Username: user001  
Password:  
dav:/webdav/>
```

A firewall rule was installed in order allow the access of the HTC and HPC instances to WebDAV. Both sites can access to the server following the commands:

```
yum install cadaver  
cadaver http://34.123.22.204/webdav/
```

```
[root@main ~]# cadaver http://34.123.22.204/webdav/  
Authentication required for webdav on server `34.123.22.204':  
Username: user001  
Password:  
dav:/webdav/>
```

### **3. Implementation of the computing model**

Here, the computational challenge is to align 300M sequences. These 300M sequences can be considered as 300 sets of 1 000 000 alignments, each having dimension of 170MB.

To exploit fully the computational resources of the HTC site and of the HPC site, the workload management system (WMS) pull model would be the best strategy. The WMS is a specific service able to manage the job distribution according to the given policy. It compares the user requirement with the available resources, selects which resources are going to be used, assigns a rank to the matches resources, and distributes jobs. In particular, the pull strategy will hold the job in queue until a resources become available. The pulling operation is performed by pilots, which make sure that resources are indeed available and pull the jobs out of the queue. This strategy would be the best one as jobs will be submitted according to the available resources present in that moment. Indeed, the model makes it easier to optimize the usage of the site resources, optimal for those challenges in which job's time and resources are not known apriori. The WMS could be installed in the third site, and pilots are distributed in the other two sites. As jobs are too big to be distributed through the WMS, WebDAV is exploited to transfer them to these sites. Since each job could be submitted in any working node, and it's not possible to known apriori in which site the input data will be necessary, the optimal data distribution strategy for this case seems to be the apriori push strategy. Following this strategy, all input data are distributed to all the nodes. Input data in the HTC site can be shared to all nodes using the shared volume, containing all necessary files (human genome's BWA indices and BWA alignment tool). This solution, though, does not deal with the problems of scalability and failover: if more than three nodes are present, the best option would be using a Storage Area Network to connect all the nodes and share the input data. Output files are stored on the site, zipped (to save space) through WebDAV to the storage site. The proposed computing model is therefore CPU-driven: jobs are sent where the computing power is available and data is transferred according to an apriori distribution.

### **4. Model evaluation to solve the entire challenge (in terms of time and cost)**

To evaluate time and costs needed to complete the challenge of aligning 300M sequences to the human genome, 6 jobs of 1M sequences were run on the HTC site with a different number of instances in the infrastructure. The time required with one node (each one with 2 vCPUs and 8GB of memory) to compute 6 jobs with 1 million alignments is 1140 seconds, meaning that the time to complete 300M alignments is 39,6 hours. Since the hourly cost of an **e2-standard-2** is around 0,07\$, the cost to complete the challenge would be of 2,772\$. Increasing the number of instances, a slight increase of cost along with a reduction of the running hours is observed.

The total budget estimated for complete the challenge is the sum of the computational cost and the storage cost. Output files should be stored on the HTC and HPC sites, then sent long-term on the storage site. The output files of one job of 1M alignments is estimated to be 2700GB. Therefore, for 300M alignments, storage on HTC and HPC is 810 000GB. The standard storage cost is 0,026\$/month, for a total of  $0,026\$/\text{month} \times 810\,000\text{GB} = 21\,060\$/\text{month}$ . The overall cost to store all processed data on the storage is 0,007\$/GB/month, for a total of  $0,007\$/\text{GB/month} \times 810\,000\text{GB} = 5670\%$  each month. In addition, the cost for storing jobs (51GB) on the storage is 1,326\$/month.

In conclusion, the total budget to run the entire challenge and store the data for one month is approximately 28 058,772\$.