

Introduction to Big Data Processing I: Project

Building a Cloud Infrastructure

Krsto Vujovic

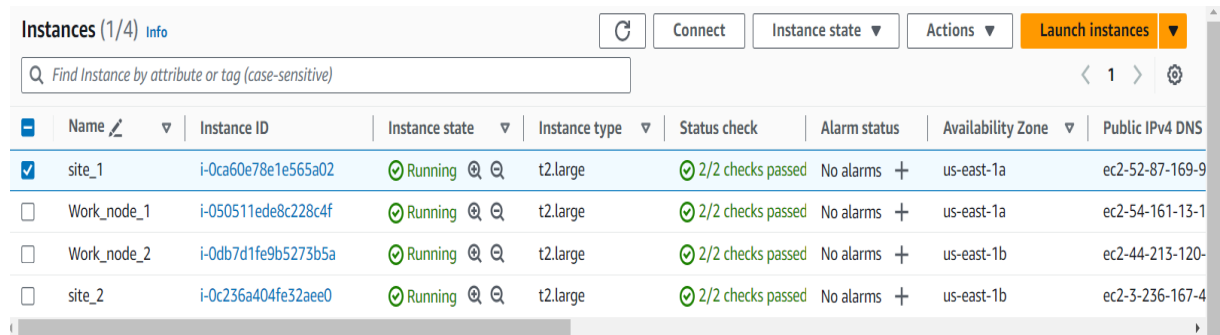
This project is about building an Infrastructure as a Service (IaaS) on the Amazon Web Services Platform with two distinct and geographically distributed “sites”. Each site should have a small CPU farm and storage resources attached (as one or more volumes and they must be shared with all nodes inside site), batch system with at least two nodes (one master (or master/worker node) and one worker node) and data transfer tool. At end of project is discussed an computing model for solving computational challenge of aligning 300 million DNA sequences (reads) against human reference genome hg19.

My infrastructure was created by making two instances in both sites and connecting them with NFS (Network File System), as batch system HTCondor was used where one node is master/worker node and other node is worker node, as data transfer tool, FTP (File Transfer Protocol) was used for sharing data among two sites. At end, there is discussion about computing model in terms of WMS (Work Management System) and Data Management, also some cost/time analysis for computational challenge.

1) Building a computing infrastructure on the Cloud composed of two distinct and geographically distributed “sites” (site1 and site2).

Four instances were created, two for each site. The infrastructure consists of two geographically distinct sites, located in us-east-1a and us-east-1b respectively. Each site is composed by two Elastic Compute Cloud (EC2) instances. The operating system was selected among the Amazon Machine Images (AMIs) provided by the “Community”: the RHEL (Red Hat Enterprise Linux)- version 7.9 has been chosen. For the all the instances, we chose the t3.large (on picture below are shown t2.large because I used all saved images when preparing project for December exam, but in February I recreated just

one site for simulation of computational challenge with t3.large instances. Configuration was same the only difference is they are cheaper and just to make sure I don't run out of credits since its my second account.



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
<input checked="" type="checkbox"/>	site_1	i-0ca60e78e1e565a02	Running	t2.large	2/2 checks passed	No alarms	us-east-1a	ec2-52-87-169-9
<input type="checkbox"/>	Work_node_1	i-050511ede8c228c4f	Running	t2.large	2/2 checks passed	No alarms	us-east-1a	ec2-54-161-13-1
<input type="checkbox"/>	Work_node_2	i-0db7d1fe9b5273b5a	Running	t2.large	2/2 checks passed	No alarms	us-east-1b	ec2-44-213-120-
<input type="checkbox"/>	site_2	i-0c236a404fe32aee0	Running	t2.large	2/2 checks passed	No alarms	us-east-1b	ec2-3-236-167-4

Picture 1. Screenshot of created instances on AWS platform.

So instance type is t3.large with a 100 Gb SSD as root storage. The two instances are set in following availability zones (us-east-1a for one site and us-east-1b for the other site in order to be as much as possible differently geographically distributed but to have a real different geographic distribution I would need not just differently zones but two different regions which is not possible with our AMAZON student AWS configuration. Master node and a Worker node, and is enclosed by the same security group, a set of firewall rules that control inbound and outbound traffic for one or more AWS instances. To all, the security group is set to allow Secure Shell Protocol (SSH) connection, port 22, from everywhere, provided the user has the matching key. We did not open it exclusively to our local public IP, because throughout the project we are accessing the internet from different geographic locations. The IP address of laptop could change if we on different wifi network or if we reset router.

I allowed all inbound traffic within same security group which will be important for the NFS and Condor implementation later, also all TCP traffic within same security group is important for the FTP.

To connect to any instance, the ssh connection is used by using the key-value pair given upon creation of the instances on AWS and the public IP address given once the instance is started. Key was same for all four instances for simpler connection.

In order to simulate a biological computational challenge, a folder with the reference genome and the reads in FASTA format have been granted us by the professor of the IBDPI course. The folder has been downloaded to the local system from the INFN website and suddenly copied to the VM using the following scp command: `scp -i <pem_key> BDP1_2022.tgz ec2-user@<private address>.compute-1.amazonaws.com:.` . Once in the VM, the folder has been unzipped using tar command. There is no need to mount any volume, since the raw folder has been downloaded straightforward: the directory will then be shared with the other nodes of the infrastructure using a Network File System (NFS) Client-Server.

1b) NFS installation

Network File System is a distributed file system protocol that allows clients to access files and directories on remote servers as if they were local.

NFS installation on site_1 was done as shown for part “on the server” in following [link](#). The most important part is editing /etc/exports by adding following <private IP of site_1 instance>(rw,sync,no_wdelay) as shown on picture below.

```
[root@ip-172-31-34-109 ~]# yum install nfs-utils rpcbind
Loaded plugins: amazon-id, product-id, search-disabled-repos, subscription-manager

This system is not registered with an entitlement server. You can use subscription-manager to register.

Package 1:nfs-utils-1.3.0-0.68.el7.2.x86_64 already installed and latest version
Package rpcbind-0.2.0-49.el7.x86_64 already installed and latest version
Nothing to do
[root@ip-172-31-34-109 ~]# systemctl enable nfs-server
[root@ip-172-31-34-109 ~]# systemctl enable rpcbind
[root@ip-172-31-34-109 ~]# systemctl enable nfs-lock
[root@ip-172-31-34-109 ~]# systemctl enable nfs-idmap
[root@ip-172-31-34-109 ~]# systemctl start rpcbind
[root@ip-172-31-34-109 ~]# systemctl start nfs-server
[root@ip-172-31-34-109 ~]# systemctl start nfs-lock
[root@ip-172-31-34-109 ~]# systemctl start nfs-idmap
[root@ip-172-31-34-109 ~]# systemctl status nfs
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
   Drop-In: /run/systemd/generator/nfs-server.service.d
            └─order-with-mounts.conf
   Active: active (exited) since Wed 2023-10-18 19:15:50 UTC; 54min ago
   Main PID: 1253 (code=exited, status=0/SUCCESS)
     Tasks: 0
    Memory: 0B
   CGroup: /system.slice/nfs-server.service

Oct 18 19:15:50 ip-172-31-34-109.ec2.internal systemd[1]: Starting NFS server and services...
Oct 18 19:15:50 ip-172-31-34-109.ec2.internal systemd[1]: Started NFS server and services.
[root@ip-172-31-34-109 ~]# nano /etc/exports
/home/ec2-user/BDP1_2023 172.31.47.193(rw,sync,no_wdelay)
[root@ip-172-31-34-109 ~]# nano /etc/exports
[root@ip-172-31-34-109 ~]# exportfs -r
[root@ip-172-31-34-109 ~]# exportfs
/home/ec2-user/BDP1_2022
172.31.47.193
[root@ip-172-31-34-109 ~]# cat /etc/exports
/home/ec2-user/BDP1_2022 172.31.47.193(rw,sync,no_wdelay)
```

Picture 2. Site_1 (Master node or server node) settings for NFS

NFS installation on Work_node_1 was done by following “on the client” commands shown in [link](#). Similar as for server node, here most important step is editing /etc/fstab by adding following line:

<Site_1 Private IP adress>:/BDPI_2022 /data nfs defaults 0 0

This means that from site_1 BDPI_2022 file will be mounted on data directory on work node (as shown in picture below).

```
[root@ip-172-31-47-193 ec2-user]# ls
[root@ip-172-31-47-193 ec2-user]# cd
[root@ip-172-31-47-193 ~]# nano /etc/fstab
[root@ip-172-31-47-193 ~]# nano /etc/fstab
[root@ip-172-31-47-193 ~]# mount -a
[root@ip-172-31-47-193 ~]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	3.8G	0	3.8G	0%	/dev
tmpfs	3.9G	0	3.9G	0%	/dev/shm
tmpfs	3.9G	17M	3.9G	1%	/run
tmpfs	3.9G	0	3.9G	0%	/sys/fs/cgroup
/dev/xvda2	30G	1.9G	29G	7%	/
tmpfs	782M	0	782M	0%	/run/user/1000
172.31.34.109:/home/ec2-user/BDP1_2022	30G	24G	7.0G	77%	/data

```
[root@ip-172-31-47-193 ~]# ll /d
data/ dev/
[root@ip-172-31-47-193 ~]# ll /data/
total 4
drwxr-xr-x. 3 ec2-user ec2-user 103 Apr 20 2022 condor
drwxr-xr-x. 4 ec2-user ec2-user 4096 Apr 19 2020 hg19
drwxr-xr-x. 2 ec2-user ec2-user 102 Apr 20 2022 trivial
```

Picture 3. Proof of NFS on work node (mounted in data directory).

Just to make sure everything is running smoothly, short exercise was made, creating pippo file inside BDP1_2022 on site_1 and then checking if its shown on worker node and vice versa.

```
[root@ip-172-31-34-109 ~]# cd /home/ec2-user/BDP1_2022
[root@ip-172-31-34-109 BDP1_2022]# ls
condor hg19 pippo trivial
[root@ip-172-31-34-109 BDP1_2022]# █
```

Picture 4. Creating pippo file inside BDP1_2022 on site_1 node

```
[root@ip-172-31-47-193 data]# touch pippo
touch: cannot touch 'pippo': Permission denied
[root@ip-172-31-47-193 data]# logout
[ec2-user@ip-172-31-47-193 ~]$ ls
[ec2-user@ip-172-31-47-193 ~]$ cd /data/
[ec2-user@ip-172-31-47-193 data]$ touch pippo
[ec2-user@ip-172-31-47-193 data]$ ls
condor hg19 pippo trivial
[ec2-user@ip-172-31-47-193 data]$ █
```

Picture 5. Confirmation for NFS on work node by checking display of pippo (ip)

1c) HTCondor installation

As above mentioned for batch system I chose HTCondor, the installation was done according to instructions on the [link](#). Now the installation procedure is same for both master and work node, but configuration of condor_config file is what differentiate master from work node.

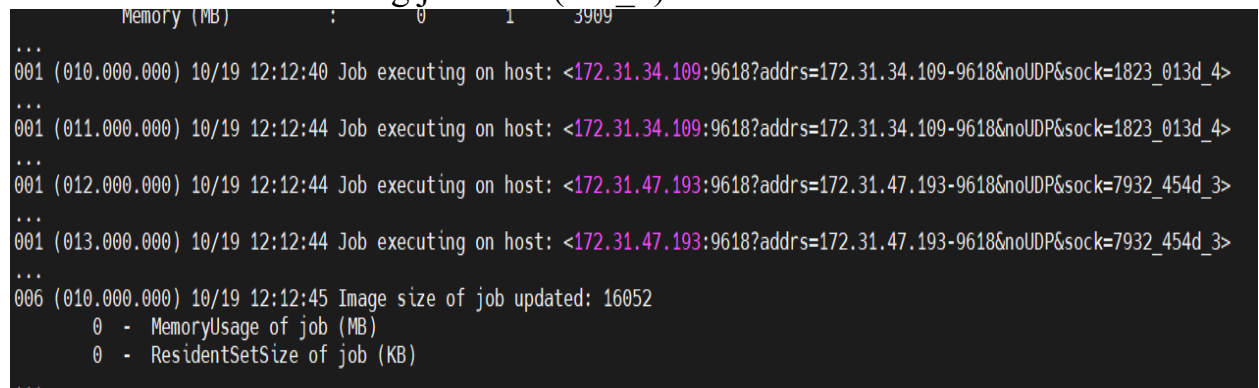
On the master/worker node we want to edit condor_config file by adding:

```
CONDOR_HOST = __put here the master Private IP address(site_1), i.e.:  
172.31.25.191 ____  
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD,  
SCHEDD  
HOSTALLOW_READ = *  
HOSTALLOW_WRITE = *  
HOSTALLOW_ADMINISTRATOR = *
```

On the worker node we want to edit condor_config file by adding:

```
CONDOR_HOST = __put here the master Private IP address(site_1), i.e.:  
172.31.25.191 ____  
DAEMON_LIST = MASTER, STARTD  
HOSTALLOW_READ = *  
HOSTALLOW_WRITE = *  
HOSTALLOW_ADMINISTRATOR = *
```

Now to check does everything works, I run condor_submit first_batch.job 4 times since I was checking just first (site_1) with 4 CPU's.

A screenshot of a terminal window displaying HTCondor logs. At the top, a header shows 'Memory (MB)' with values 0, 1, and 3909. The log contains several entries: three jobs (001) executing on hosts 172.31.34.109 and 172.31.47.193, and one entry (006) showing the image size of a job updated to 16052. The jobs are distributed across four different hosts as indicated by the IP addresses in the log entries.

```
...  
Memory (MB) : 0 1 3909  
...  
001 (010.000.000) 10/19 12:12:40 Job executing on host: <172.31.34.109:9618?addr=172.31.34.109-9618&noUDP&sock=1823_013d_4>  
...  
001 (011.000.000) 10/19 12:12:44 Job executing on host: <172.31.34.109:9618?addr=172.31.34.109-9618&noUDP&sock=1823_013d_4>  
...  
001 (012.000.000) 10/19 12:12:44 Job executing on host: <172.31.47.193:9618?addr=172.31.47.193-9618&noUDP&sock=7932_454d_3>  
...  
001 (013.000.000) 10/19 12:12:44 Job executing on host: <172.31.47.193:9618?addr=172.31.47.193-9618&noUDP&sock=7932_454d_3>  
...  
006 (010.000.000) 10/19 12:12:45 Image size of job updated: 16052  
    0 - MemoryUsage of job (MB)  
    0 - ResidentSetSize of job (KB)  
...
```

Picture 6. Checking execution of four submitted first_batch.job

As shown on picture above jobs are executed on all four cores, notice ip addresses are different which means batch system was installed successfully.

2) DATA sharing using FTP

FTP (File Transfer Protocol) is a standard network protocol used to transfer files to and from a remote network. FTP client connects to the remote server and download or upload files.

Installation of FTP was done by following command on [link](#).

As with others, most important part is setting configuration, adding following lines to the vsftpd.conf on the server node:

```
anonymous_enable=NO
connect_from_port_20=YES
chroot_local_user=NO
userlist_enable=YES
userlist_file=/etc/vsftpd.userlist
userlist_deny=NO
pasv_promiscuous=YES
pasv_enable=YES
```

After that user ravi was created and home populated with shining.txt.gz:

```
su - ravi -c "cp /home/ec2-user/BDP1_2022/trivial/shining.txt.gz /home/ravi/"
```

```
[root@ip-172-31-34-109 ~]# su - root ravi -c "cp /home/ec2-user/BDP1_2022/trivial/shining.txt.gz /home/ravi/"
[root@ip-172-31-34-109 ~]# ls -l /home/ravi/
total 124
-rw-r--r--. 1 root root 125155 Oct 21 15:14 shining.txt.gz
[root@ip-172-31-34-109 ~]# echo "ravi" | tee -a /etc/vsftpd.userlist
ravi
[root@ip-172-31-34-109 ~]# cat /etc/vsftpd.userlist
ravi
[root@ip-172-31-34-109 ~]# systemctl start vsftpd.service
[root@ip-172-31-34-109 ~]# systemctl status vsftpd.service
● vsftpd.service - Vsftpd ftp daemon
   Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2023-10-21 15:20:14 UTC; 9s ago
     Process: 2420 ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf (code=exited, status=0/SUCCESS)
    Main PID: 2421 (vsftpd)
       Tasks: 1
      Memory: 576.0K
     CGroup: /system.slice/vsftpd.service
            └─2421 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

Oct 21 15:20:14 ip-172-31-34-109.ec2.internal systemd[1]: Starting Vsftpd ftp...
Oct 21 15:20:14 ip-172-31-34-109.ec2.internal systemd[1]: Started Vsftpd ftp ...
Hint: Some lines were ellipsized, use -l to show in full.
```

Picture 7.FTP activation on server successful

Installation of FTP on the client in this case (site_2) because we want to connect two sites:

Most important are two commands:

ftp <private ip address of server (site_1)>

get filename(in my case the file is shining.txt.gz)

```
[root@ip-172-31-6-26 ~]# ftp 172.31.34.109
Connected to 172.31.34.109 (172.31.34.109).
220 (vsFTPd 3.0.2)
Name (172.31.34.109:root): ravi
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (172,31,34,109,224,251).
150 Here comes the directory listing.
-rw-r--r--  1 0      0      125155 Oct 21 15:14 shining.txt.gz
226 Directory send OK.
ftp> pwd
257 "/home/ravi"
ftp>
```

Picture 8. Connecting to FTP with client (site_2)

As we can see the connection from client was successful. Now with get command we should receive the file on client.

```
ftp> get shining.txt.gz
local: shining.txt.gz remote: shining.txt.gz
227 Entering Passive Mode (172,31,34,109,141,171).
150 Opening BINARY mode data connection for shining.txt.gz (125155 bytes).
226 Transfer complete.
125155 bytes received in 0.00143 secs (87704.98 Kbytes/sec)
ftp> ls
227 Entering Passive Mode (172,31,34,109,49,171).
150 Here comes the directory listing.
-rw-r--r--  1 0      0      125155 Oct 21 15:14 shining.txt.gz
226 Directory send OK.
ftp>
```

Picture 9. File transfer via FTP

We can see from picture nine that transfer was successful and complete.

3) The computational challenge

Now, I will consider the computational challenge of aligning 300 million sequences to the human genome using the BWA alignment.

The Burrows – Wheeler Aligner (BWA) is a software package to align low divergent sequences against a large reference genome, (e.g. the human reference genome – hg19). The package provides three algorithms better suited for different input sequences: BWA-backtrack, BWA-SW and BWA-MEM. Regardless the algorithm, the first step consists of the construction of a FM-index from the reference genome. The indexing helps the algorithm to find the sequences to be aligned in the genome faster.

To build the FM index of the hg19:

```
bwa index -p hg19bwaidx -a bwtsv entire_hg19.fa
```

The BWA application was installed on the Master Node by copying from the shared volume the bwa-0.7.15 tar file and by running the following commands:

```
tar -xvf bwa-0.7.15.tar
cd bwa-0.7.15/
make
```

All commands for installation can be found on following [link](#). For the simulation of the computational challenge we need two scripts align.py and bwa_batch.job.

To make sure everything is running smoothly, I transferred, bwa, ten fasta sequences from patient 3 and hg19bwaidx (index) in same directory as above mentioned scripts.

The bwa_batch.job is the submit description file taken as argument by the condor_submit command. In this file HTCondor finds everything that is needed to run the application. In a picture below is look inside the script and I will go just briefly over it.


```

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

Executable = align.py

n = $(Process)+1

##### Input Sandbox #####
Input = read_$INT(n).fa

transfer_input_files = bwa, read_$INT(n).fa
Arguments = read_$INT(n).fa
##### Output Sandbox #####
Log = read_$INT(n).log
Output = read_$INT(n).out
Error = read_$INT(n).error
# will contain the standard output
# will contain the standard error
transfer_output_files = read_$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 10

```

Picture 10. Script bwa_batch.job

In this case, in the Input Sandbox there are:
the executable (align.py)
bwa
read to be aligned (read_\$INT(n).fa)

In the Output Sandbox there are all results:
.sai file: it is a binary file returned by the command bwa aln
.sam.gz: the command bwa samse converts the .sai file into the human readable format,
.sam file. The SAM format is a text format for storing sequence data in a series of tab delimited ASCII columns. Once generated, the file is compressed.
md5.txt: the result of the checksum computation by the software md5sum. The md5sum is used to verify that the files have not changed during the file transferring.
.log file: the file that records all events occurring in HTCondor cluster (e.g in which machine a job is running, the allocated memory, the starting time etc)
.error file: the file storing the standard errors. Particularly useful when troubleshooting.
.out file: it contains the 'print' statement of the executable align.py.

Now for the executable align.py the script looks like this:

```

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

# Control variables
#####
start_time= timer()
dbpath = "/home/ec2-user/BDP1_2022/condor/hg/"
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)

```

In the align.py script, the alignment between the transferred 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 in turn gzipped. Moreover the checksums of the .sai and .sam files are computed.


4) Discussion about implementation of computational model

I will consider the computational challenge of aligning 300 million sequences to the human genome using the BWA.


First we need to inspect the computational challenge, we need to find what is main problem in solving this challenge, is it number of the sequences (300 milion) or is it running time of BWA. Now lets start with running time of BWA, if we compare it to the some of other alignment methods.

Comparing the three alignment tools

	In-house Brute Force	BLASTn	BWA
1 line read	409.4s	0.54s	3.2s
1000 lines read	410000s	574.1s	3.6s



To solve your challenge you need a **supercomputer**



To solve your challenge you need your **smartphone**

Picture 11. Comparison of three alignment tools (its taken from presentation slides scientific computational challenge)

Now if we look at picture above we can see that BWA is pretty good alignment tool, so we deduce that main problem of this challenge will be huge number of sequences that we have to align. Which means, that our first decision will be that we need some type of HTC infrastructure, which focus on having highest throughput as possible in a time unit (in other words as many jobs as possible in same time). But for next step we need to inspect what kind of alignments we have?

I have to align 300 milions sequences to the index, but each alignment is independent from one another (its like taking a huge phone book and having to find 300 milion of different contacts, there is no dependency which contact have which number)

This job can be considered an embarrassingly parallel job, since the alignment of one sequence has nothing to do with the alignment of another sequence.

For this reason, I have another proof that sequence alignment is perfect to be run with an HTC infrastructure.

Now, we need to pick a suitable WMS (Workload Management System) strategy and Data Management approach.

The Workload management system (WMS) deals with job scheduling, i.e., distributes jobs across available resources. It should do this according to a scheduling policy which is decided by the owner of the infrastructure.

I would use as submission mode a Push Job Submission mode where the Submission is done through the Workload Management System (HT Condor in our case). I personally like more approach of pull model, but here I wouldn't choose a pull model because for example a pull model could have the disadvantage of easily destroying any system a priori data distribution and data preplacement. Since I am not too skilled in setting up pull model, this above would probably happen to me by Murphy law. That's why I decided to go with combination of push model and for data management would use an a-priori push approach:

before submitting the job, the user distribute the data to the store resources, data that are needed in the input and then submits the job.

We know the resources that will be used, and we distribute the data before launching the jobs (first operation is to distribute data to all the sites we can access to run our job on).

Time/Cost estimation

From patient 3, six fasta files from one till six were taken to simulate challenge and get estimations, I will report just average value out of six. So BWA processed 1000 sequences with each run with average time in about 26 seconds so to align 300 millions sequences to the database, the time needed is:

$300 \text{ million} / 1000 = 300,000$ multiplied by 26 sec = 7 800 000 seconds

to convert the time to hours:

Total Time (hours) = Total Time (seconds) / 3600
Total Time (hours) = 7 800 000 / 3600 \approx 2167 hours (which is equal to 90 days)

Now if we want to scale it up to 4 instances:

$300 \text{ million} / 4000 = 75,000$ multiplied by 26 sec = 1 950 000 seconds

to convert the time to hours:

Total Time (hours) = Total Time (seconds) / 3600
Total Time (hours) = 1 950 000 / 3600 \approx 542 hours (which is equal to 23 days).

In table below are summed results up to 64 instances with 128 CPU's.

Table 1. Time cost depending on number of instances

Instance type	Instances	CPU	Sequences	Total time	Cost
t3.large	1	2	1000	26sec	0.0832\$
t3.large	1	2	300 000	90 days	180\$
t3.large	4	8	75 000	23 days	45\$
t3.large	8	16	37 500	11 days	22\$
t3.large	16	32	18 750	6 days	12\$
t3.large	32	64	Same perform.	6days	12\$
t3.large	64	128	Same perform.	7days	14\$

We can see that it scales linearly at start, after 16 instances and 32 CPU's it does not improve, this is due to Amdahl law, there is a limit to which it can grow and it depends on how high is parallel fraction.

On picture below we can see cost prices of t family on AmazonWebService platform.

As we can see from the picture, t3.large have pretty good price for their performance but it would be improvement to have t4g.large with same characteristics but cheaper cost.

Instance name ▲	On-Demand hourly rate ▼	vCPU ▼	Memory ▼	Storage ▼	Network performance ▼
t4g.nano	\$0.0042	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t4g.micro	\$0.0084	2	1 GiB	EBS Only	Up to 5 Gigabit
t4g.small	\$0.0168	2	2 GiB	EBS Only	Up to 5 Gigabit
t4g.medium	\$0.0336	2	4 GiB	EBS Only	Up to 5 Gigabit
t4g.large	\$0.0672	2	8 GiB	EBS Only	Up to 5 Gigabit
t3.nano	\$0.0052	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t3.micro	\$0.0104	2	1 GiB	EBS Only	Up to 5 Gigabit
t3.small	\$0.0208	2	2 GiB	EBS Only	Up to 5 Gigabit
t3.medium	\$0.0416	2	4 GiB	EBS Only	Up to 5 Gigabit
t3.large	\$0.0832	2	8 GiB	EBS Only	Up to 5 Gigabit
t3a.nano	\$0.0047	2	0.5 GiB	EBS Only	Up to 5 Gigabit
t3a.micro	\$0.0094	2	1 GiB	EBS Only	Up to 5 Gigabit

Picture 12. AWS prices of various t family instances

Below we have some prices from Google Cloud and as we can see there are some instances with same performance that are cheaper then t3large.

Table 2. Prices of some instances on Google Cloud Platform

Machine type	CPUs	Memory	Price(USD)	Cheaper then t3.large
e2-standard-2	2	8GB	\$0.067006	Yes
n2-standard-2	2	8GB	\$0.097118	No
n2d-standard-2	2	8GB	\$0.084492	No

Good improvment could be using instances that are customed for the need, we just analyzed the general purpose ones, but there are lots of types for different needs in terms storage, data transfer and etc...