# Design and Evaluation of a Cloud-Based IaaS Infrastructure for Large-Scale Sequence Alignment

ALESSIO MUOLO

*Master's Degree in Bioinformatics, Alma Mater Studiorum*
*Bologna, Italy*
*E-mail: alessio.muolo@studio.unibo.it*

**The scope of this project is to design, implement, and evaluated the time-cost performance of a cloud-based Infrastructure as a Service (IaaS) using Amazon Web Services (AWS) to tackle a hypothetical computational challenge: aligning sequencing data from 1000 individuals using the Burrows-Wheeler Aligner (BWA) MEM algorithm. The infrastructure was deployed across two geographically distinct sites within the AWS us-east-1 region to facilitate testing across different instance types. For this project, a shared environment using Network File System (NFS) was implemented for data exchange, as well as HTCondor for workload scheduling, and WebDAV for cross-site data transfer. Two key computational parameters (input batch size and threading mode) were taken into consideration for this evaluation to assess their impact on performance.**

## 1. INFRASTRUCTURE

### A. Overview

The infrastructure consists of two geographically distributed sites (Site 1 and Site 2) located in the us-east-1a and us-east-1d AWS availability zones, respectively, within the Northern Virginia region (Figure 1).

Site 1, in us-east-1a, includes one master node and two worker nodes, all based on the r6a.large Amazon EC2 instance type. These are memory-optimized instances with a memory-to-vCPU ratio of 8:1. Each instance is provisioned with 2 vCPUs, 16 GB of RAM, and a 10 GB gp3 SSD. The master node also has an additional empty 100 GB gp2 volume attached, which was later populated with part of the data needed for the computational challenge.

Site 2, in us-east-1d, mirrors the same topology with one master node and two worker nodes, all using the r7iz.large instance type. Like the r6a.large, these are also memory-optimized with an 8:1 memory-to-vCPU ratio. However, they offer higher CPU performance and use newer DDR5 memory. Each instance is equipped with 2 vCPUs, 16 GB RAM, and a 10 GB gp3 SSD. A

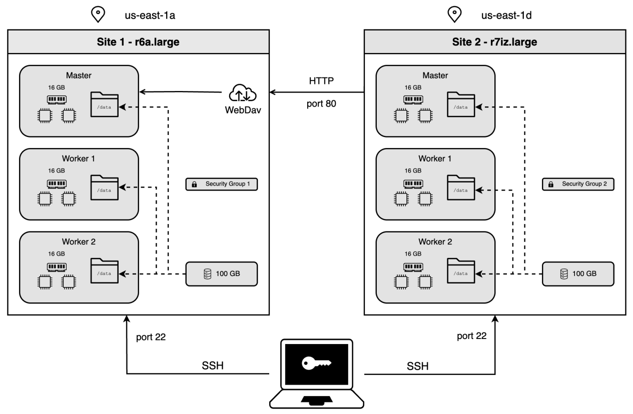


**Fig. 1.** Overview scheme of the cloud IaaS.

separate 100 GB volume is attached to the master node, which is populated with data transferred from Site 1 using WebDAV.

Within each site, data is shared from the master node to its corresponding worker nodes via the Network File System (NFS). Job scheduling and resource management are handled by HTCondor, with all nodes configured as execute nodes. Additionally, the master node in each site functions as both the Central Manager and Access Point. Data transfers between the two sites is ensured through WebDAV.

Each site is associated with its own security group (Security Group 1 for Site 1 and Security Group 2 for Site 2) containing their respective instances. SSH access (port 22) is restricted to a specific personal IP address to enhance security. Intra-site communication is enabled by allowing all traffic within each security group. To facilitate inter-site communication, inbound HTTP traffic on port 80 is permitted on Site 1, but only from the public IP address of Site 2's master node. This configuration ensures secure SSH access, unrestricted internal communication within each site, and controlled connectivity between the two sites.

### B. AWS instances (setup and Specs)

The infrastructure is hosted on Amazon Elastic Compute Cloud (Amazon EC2), where each instance represents a virtual machine running in the AWS Cloud. Instances are created from Amazon

Machine Images (AMIs), which define the operating system, application server, and pre-installed software used during deployment. All the instances used in this project were launched using the RHEL-7.9_HVM-20240930-x86_64-0-Hourly2-GP3 as AMI from the Community AMIs. When creating an instance, a key pair must be created as well to connect to the instance. The private key consists of a `.pem` file that needs to be stored in the directory from where the SSH connection is established. For each site, two separate key pairs were created to access the master and the worker nodes, respectively. The preferred subnet is chosen upon the first launch. All instances within the same site utilize the same security group, which control the incoming and outgoing traffic. All instances could be accessed using the ssh command and the private key from my laptop.

The selection of instance types was primarily influenced by the limitations imposed by the AWS Learner Lab plan, which restricted us to large-size instances. Additionally, memory capacity played a crucial role in this decision. Instances equipped with 16 GB of RAM were favored over those with only 8 GB to prevent running out of memory during the benchmarking process Two instance types were evaluated: r6a.largein Site1 and r7iz.largein Site2. The r6a.large instances utilize DDR4 memory and AMD processors, whereas the r7iz.large instances are equipped with DDR5 memory and Intel processors, offering marginally higher performance at a slightly increased cost. While the infrastructure comprised heterogeneous instance types due to the constraints mentioned above, the optimal configuration for addressing the computational challenge and conducting a time-versus-cost analysis would ideally involve a homogeneous setup using the most efficient instance type.

## C. Volumes

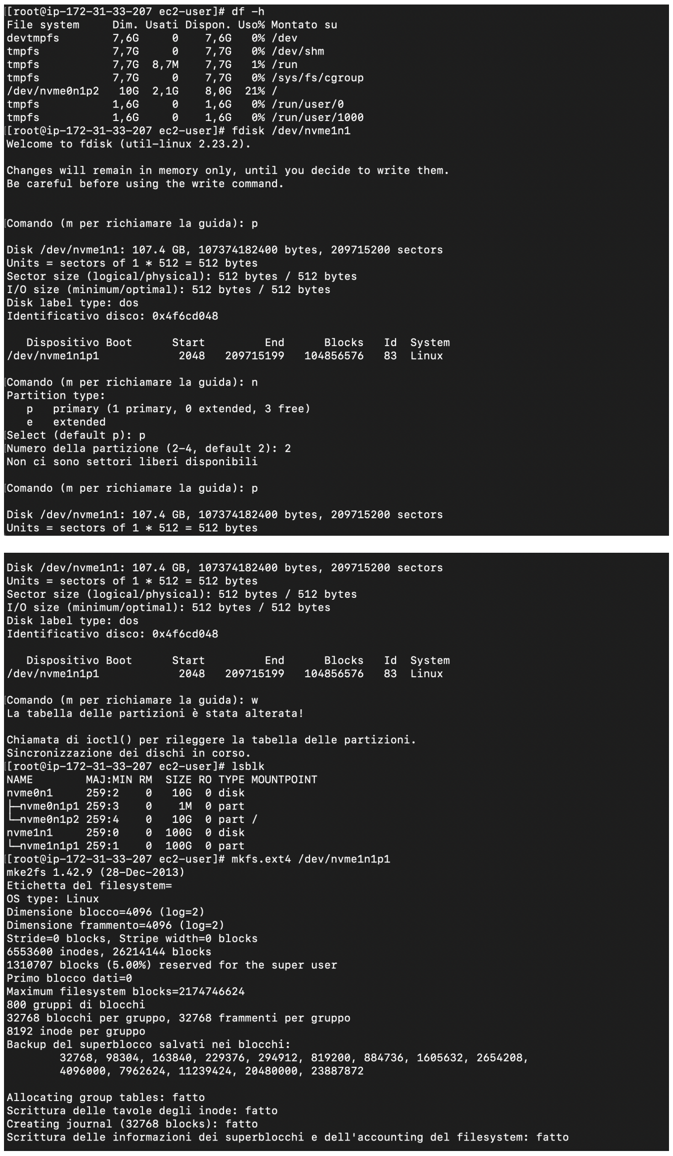
To expand the storage capacities of our sites, an empty 100GB gp2 SSD volume was attached to each master node. These volumes were configurated and directly attached when launching the master nodes and were initiated in the same availability zone as the respective instance. It was necessary to partition the volumes by connecting to the instances and using the interactive command `fdisk`. A filesystem was also created using the `ext4` command (Figure 2).

Following disk formatting, the mount point `/data` was created to serve as the target directory. The volumes were then permanently mounted to this location by editing the `fstab` file, which defines the source path of the volume and its corresponding mount destination. The `fstab` (file systems table) is a system configuration file that enumerates available disk partitions and other file systems, enabling automatic mounting at boot time (Figure 3).

## D. Installing the NFS Client-Server

To enable data sharing between the master and worker nodes within each site, the Network File System (NFS) was installed on all instances. NFS is a protocol that facilitates the sharing of directories and files over a network, where systems can function either as servers or clients. A server hosts a file system and makes it accessible to other machines, while clients mount and interact with the shared file system remotely.

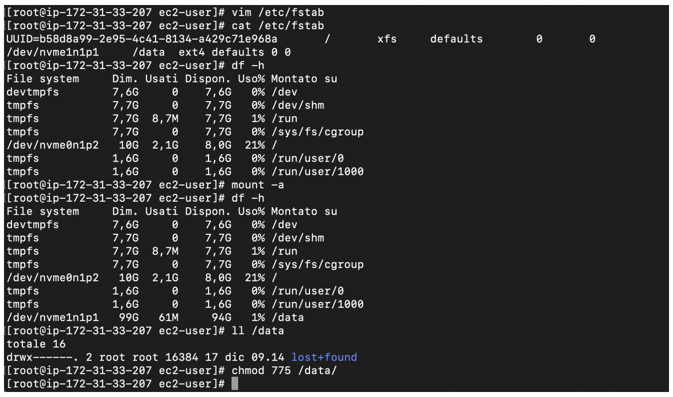
In this setup, the master node operated as the NFS server, and the worker nodes functioned as clients. After confirming that the NFS service was active, the master node's export file (`/etc/exports`) was configured to share the "`/data`" directory with the worker nodes, identified by their private IP addresses.



```
[root@ip-172-31-33-207 ec2-user]# df -h
File system     Dim. Usati Dispon. Uso% Montato su
devtmpfs         7,6G     0    7,6G   0% /dev
tmpfs            7,7G     0    7,7G   0% /dev/shm
tmpfs            7,7G  8,7M    7,7G   1% /run
tmpfs            7,7G     0    7,7G   0% /sys/fs/cgroup
/dev/nvme0n1p2   10G  2,1G    8,0G  21% /
tmpfs            1,6G     0    1,6G   0% /run/user/0
tmpfs            1,6G     0    1,6G   0% /run/user/1000
[root@ip-172-31-33-207 ec2-user]# fdisk /dev/nvme1n1
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Comando (m per richiamare la guida): p

Disk /dev/nvme1n1: 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
Identificativo disco: 0x4f6cd048

   Dispositivo Boot      Start         End      Blocks   Id  System
/dev/nvme1n1p1            2048   209715199   104856576   83  Linux

Comando (m per richiamare la guida): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Numero della partizione (2-4, default 2): 2
Non ci sono settori liberi disponibili

Comando (m per richiamare la guida): p

Disk /dev/nvme1n1: 107.4 GB, 107374182400 bytes, 209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
```

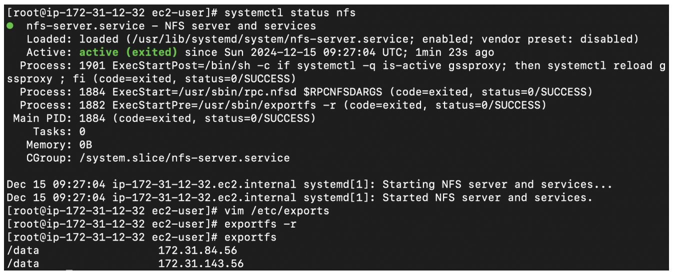
```
Disk /dev/nvme1n1: 107.4 GB, 107374182400 bytes, 209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Identificativo disco: 0x4f6cd048

   Dispositivo Boot      Start         End      Blocks   Id  System
/dev/nvme1n1p1            2048   209715199   104856576   83  Linux

Comando (m per richiamare la guida): w
La tabella delle partizioni è stata alterata!

Chiamata di ioctl() per rileggere la tabella delle partizioni.
Sincronizzazione dei dischi in corso.
[root@ip-172-31-33-207 ec2-user]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
nvme0n1     259:2    0   10G  0 disk
├─nvme0n1p1 259:3    0    1M  0 part
└─nvme0n1p2 259:4    0   10G  0 part /
nvme1n1     259:0    0  100G  0 disk
└─nvme1n1p1 259:1    0  100G  0 part
[root@ip-172-31-33-207 ec2-user]# mkfs.ext4 /dev/nvme1n1p1
mke2fs 1.42.9 (28-Dec-2013)
Etichetta del filesystem=
OS type: Linux
Dimensione blocco=4096 (log=2)
Dimensione frammento=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
6553600 inodes, 26214144 blocks
1310707 blocks (5.00%) reserved for the super user
Primo blocco dati=0
Maximum filesystem blocks=2174746624
800 gruppi di blocchi
32768 blocchi per gruppo, 32768 frammenti per gruppo
8192 inode per gruppo
Backup del superblocco salvati nei blocchi:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: fatto
Scrittura delle tavole degli inode: fatto
Creating journal (32768 blocks): fatto
Scrittura delle informazioni dei superblocchi e dell'accounting del filesystem: fatto
```

**Fig. 2.** Terminal connected through SSH to the Site 2 Master node. Here is represented the process of formatting the empty volume.



```
[root@ip-172-31-33-207 ec2-user]# vim /etc/fstab
[root@ip-172-31-33-207 ec2-user]# cat /etc/fstab
UUID=b58d8a99-2e95-4c41-8134-a429c71e968a       /       xfs     defaults        0       0
/dev/nvme1n1p1     /data  ext4 defaults 0 0
[root@ip-172-31-33-207 ec2-user]# df -h
File system     Dim. Usati Dispon. Uso% Montato su
devtmpfs         7,6G     0    7,6G   0% /dev
tmpfs            7,7G     0    7,7G   0% /dev/shm
tmpfs            7,7G  8,7M    7,7G   1% /run
tmpfs            7,7G     0    7,7G   0% /sys/fs/cgroup
/dev/nvme0n1p2   10G  2,1G    8,0G  21% /
tmpfs            1,6G     0    1,6G   0% /run/user/0
tmpfs            1,6G     0    1,6G   0% /run/user/1000
[root@ip-172-31-33-207 ec2-user]# mount -a
[root@ip-172-31-33-207 ec2-user]# df -h
File system     Dim. Usati Dispon. Uso% Montato su
devtmpfs         7,6G     0    7,6G   0% /dev
tmpfs            7,7G     0    7,7G   0% /dev/shm
tmpfs            7,7G  8,7M    7,7G   1% /run
tmpfs            7,7G     0    7,7G   0% /sys/fs/cgroup
/dev/nvme0n1p2   10G  2,1G    8,0G  21% /
tmpfs            1,6G     0    1,6G   0% /run/user/0
tmpfs            1,6G     0    1,6G   0% /run/user/1000
/dev/nvme1n1p1   99G   61M     94G   1% /data
[root@ip-172-31-33-207 ec2-user]# ll /data
totale 16
drwx------. 2 root root 16384 17 dic 09.14 lost+found
[root@ip-172-31-33-207 ec2-user]# chmod 775 /data/
[root@ip-172-31-33-207 ec2-user]#
```

**Fig. 3.** Terminal connected through SSH to the Site 2 Master node. Here is represented the edit of the `fstab` file for permanently mounting the volume.



```
[root@ip-172-31-12-32 ec2-user]# systemctl status nfs
●  nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
   Active: active (exited) since Sun 2024-12-15 09:27:04 UTC; 1min 23s ago
  Process: 1901 ExecStartPost=/bin/sh -c if systemctl -q is-active gssproxy; then systemctl reload g
ssproxy ; fi (code=exited, status=0/SUCCESS)
  Process: 1884 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited, status=0/SUCCESS)
  Process: 1882 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
 Main PID: 1884 (code=exited, status=0/SUCCESS)
    Tasks: 0
   Memory: 0B
   CGroup: /system.slice/nfs-server.service

Dec 15 09:27:04 ip-172-31-12-32.ec2.internal systemd[1]: Starting NFS server and services...
Dec 15 09:27:04 ip-172-31-12-32.ec2.internal systemd[1]: Started NFS server and services.
[root@ip-172-31-12-32 ec2-user]# vim /etc/exports
[root@ip-172-31-12-32 ec2-user]# exportfs -r
[root@ip-172-31-12-32 ec2-user]# exportfs
/data                   172.31.84.56
/data                   172.31.143.56
```

**Fig. 4.** Terminal at the Site 1 Master node. Notice how the IP address of both worker nodes is listed when calling `exportfs`.

The changes were applied using `exportfs -r`, and the shared directories were verified with `exportfs` (Figure 4).

To ensure proper communication, the security group settings were adjusted to allow unrestricted internal traffic, enabling seamless data sharing among nodes within the same site.

### E. Installing HTCondor

HTCondor is a high-performance computing (HPC) workload management system that facilitates the execution of computational tasks across a distributed set of machines. It is specifically designed for managing batch jobs on clusters and grids, handling job scheduling, resource allocation, and job execution. HTCondor operates in a pool of machines, where each machine can take on different roles:

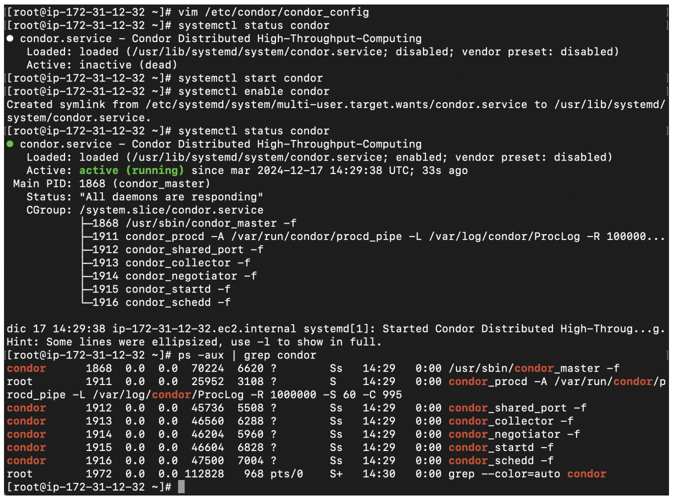
- Central Manager: manages the overall pool, schedules jobs, negotiates between resources and resource requests and monitors the status of the machines and jobs.
- Submitter: submits the jobs to the pool for execution and manages the queue.
- Worker (Execution Point): these machines run the submitted jobs and can either be dedicated to the pool or dynamically added, depending on the system's configuration.

HTCondor ensures efficient resource usage by assigning jobs based on available resources, prioritizing tasks, and handling failures through job retries. This makes it particularly useful for environments requiring flexible, scalable, and efficient management of large numbers of jobs.

For this infrastructure, in both sites, the master node takes the role of Central Manager, Submitter and Worker, while the other two instances only act as execution machines. To implement this configuration, HTCondor must be installed on both the master and worker nodes, and the `condor_config` file on each node must be properly edited. The `CONDOR_HOST` parameter should be set to the private IP address of the master node, enabling all nodes in the pool to locate and communicate with it. Additionally, the appropriate daemons must be specified: on the master node, this includes `MASTER`, `COLLECTOR`, `NEGOTIATOR`, `STARTD`, and `SCHEDD`; on the worker nodes, only the `MASTER` and `STARTD` daemons are required (Figure 5).

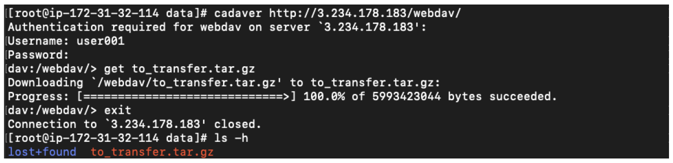
### F. Installing WebDav

WebDAV (Web Distributed Authoring and Versioning) is a protocol that enhances HTTP by enabling remote file management on web servers. Within the described infrastructure, WebDAV was utilized to establish data exchange between two previously isolated sites. Site 1's master node was configured as the WebDAV server, while Site 2's master acted as the client. To facilitate



```
[root@ip-172-31-12-32 ~]# vim /etc/condor/condor_config
[root@ip-172-31-12-32 ~]# systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
[root@ip-172-31-12-32 ~]# systemctl start condor
[root@ip-172-31-12-32 ~]# systemctl enable condor
Created symlink from /etc/systemd/system/multi-user.target.wants/condor.service to /usr/lib/systemd/
system/condor.service.
[root@ip-172-31-12-32 ~]# 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 mar 2024-12-17 14:29:38 UTC; 33s ago
 Main PID: 1868 (condor_master)
   Status: "All daemons are responding"
   CGroup: /system.slice/condor.service
           ├─1868 /usr/sbin/condor_master -f
           ├─1911 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 100000...
           ├─1912 condor_shared_port -f
           ├─1913 condor_collector -f
           ├─1914 condor_negotiator -f
           ├─1915 condor_startd -f
           └─1916 condor_schedd -f

dic 17 14:29:38 ip-172-31-12-32.ec2.internal systemd[1]: Started Condor Distributed High-Throug...g.
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-12-32 ~]# ps -aux | grep condor
condor      1868  0.0  0.0  70224  6620 ?        Ss   14:29   0:00 /usr/sbin/condor_master -f
root        1911  0.0  0.0  25952  3108 ?        S    14:29   0:00 condor_procd -A /var/run/condor/p
rocd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
condor      1912  0.0  0.0  45736  5508 ?        Ss   14:29   0:00 condor_shared_port -f
condor      1913  0.0  0.0  46560  6288 ?        Ss   14:29   0:00 condor_collector -f
condor      1914  0.0  0.0  46204  5960 ?        Ss   14:29   0:00 condor_negotiator -f
condor      1915  0.0  0.0  46604  6828 ?        Ss   14:29   0:00 condor_startd -f
condor      1916  0.0  0.0  47500  7004 ?        Ss   14:29   0:00 condor_schedd -f
root        1972  0.0  0.0 112828   968 pts/0    S+   14:30   0:00 grep --color=auto condor
[root@ip-172-31-12-32 ~]#
```

**Fig. 5.** Terminal at the Site 1 Master node showing the status of the HTCondor management system.



```
[root@ip-172-31-32-114 data]# cadaver http://3.234.178.183/webdav/
Authentication required for webdav on server `3.234.178.183':
Username: user001
Password:
dav:/webdav/> get to_transfer.tar.gz
Downloading `/webdav/to_transfer.tar.gz' to to_transfer.tar.gz:
Progress: [=============================>] 100.0% of 5993423044 bytes succeeded.
dav:/webdav/> exit
Connection to `3.234.178.183' closed.
[root@ip-172-31-32-114 data]# ls -h
lost+found  to_transfer.tar.gz
```

**Fig. 6.** Terminal at the Site 2 Master node showing the download of the files necessary for the challenge that were previously uploaded from Site 1 Master node.

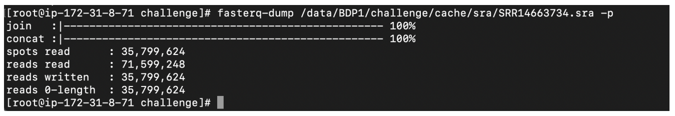
this connection, a new security group rule was added to permit inbound HTTP traffic (port 80) from Site 2's public IP address to Site 1.

To access the server, user authentication is required. For this reason, a dedicated account with a username and password is created. Local access from Site 1 was performed using the master node's private IP address, whereas remote access from Site 2 required the master's public IP. For the data transfer, only the essential files for the computational challenge were packaged into a `.tar` archive and uploaded to the WebDAV server on Site 1. The cadaver command-line client was then used on Site 2 to retrieve the archive, effectively enabling controlled and secure cross-site data transfer (Figure 6).

## 2. THE COMPUTATIONAL CHALLENGE

### A. Aim of the challenge

The infrastructure developed in this project is designed to address a hypothetical large-scale computational task: the alignment of sequencing reads obtained from whole-exome sequencing (WES) experiments for 1,000 individuals against a reference genome using the BWA-MEM algorithm. To approximate the time and financial resources required to complete this scenario, a prototype version of the infrastructure was evaluated using a subset of reads sourced from a publicly available WES dataset. Based on this dataset, the average FASTQ file from a single individual contains approximately 35 million reads, which served as the basis for extrapolating the total data volume in the full-scale scenario. The hypothetical use case is based on the condition that all sequencing data are initially stored and staged at Site 1 and needs to be processed at both sites. Both sites must have



```
[root@ip-172-31-8-71 challenge]# fasterq-dump /data/BDP1/challenge/cache/sra/SRR14663734.sra -p
join   :|-------------------------------------------------- 100%
concat :|-------------------------------------------------- 100%
spots read      : 35,799,624
reads read      : 71,599,248
reads written   : 35,799,624
reads 0-length  : 35,799,624
[root@ip-172-31-8-71 challenge]#
```

**Fig. 7.** Terminal at the Site 1 Master node showing the download of the genomic sample from the SRA using the `fasterq_dump` command.

the same number of instances and respective roles. Under this frameset, half of the input dataset is transferred from Site 1 to Site 2 for processing, after which the resulting output files are sent back to Site 1.

### B. Data

The genomic data needed for the computational challenge was retrieved from the Sequence Read Archive (SRA) [1], which is a public repository maintained by the National Center for Biotechnology Information (NCBI) that stores raw high-throughput sequencing data from various experiments in standardized formats. For this project, a publicly available adult male blood sample from a single-end whole exome sequencing experiment conducted using the Illumina platform for a multi-ethnic semen quality analysis was chosen [2]. The genomic data was downloaded in the form of a FASTQ file using the fasterq-dump tool of the SRA toolkit using the correspondent SRA accession number (SRR14663734) (Figure 7). The downloaded FASTQ file had a size of 12.8 GB (gz: 2.17 GB) and contained 35,799,624 reads. The library was single-end, and the average read length was 136 bp. As reference genome we adopted GRCh37 (Genome Reference Consortium Human Build 37) assembly, commonly known as hg19. The reference genome and its BWA index files were kindly provided by prof. Cesini in the form of a downloadable `tar` archive.

### C. Burrows-Wheeler Aligner algorithm

BWA (Burrows-Wheeler Aligner) is a widely used software package for aligning low-divergent sequences to a reference genome, and it is designed to handle large-scale datasets generated by high-throughput sequencing technologies [3]. It consists of several algorithms: BWA-backtrack for short reads, BWA-SW for longer or gapped reads, and BWA-MEM, the most recent and commonly used version, optimized for reads over 70 bp and up to a few megabases. BWA-MEM is the default and recommended option for most modern sequencing datasets due to its accuracy, speed, and robustness in handling longer reads with varying levels of divergence. Since our sample (and generally most whole-exome sequencing samples) presents an average read length of 150bp, the BWA-MEM algorithm was adopted.

### D. Computing parameters

One of the main scopes of this project is to understand how computing parameters can impact the performance of the challenge [4]. When comparing difference environments, achieving good performance and scalability can be challenging. For this project, two computing parameters were taken into consideration: batch size and ingle-threading (ST) versus multithreading (MT) modality. The batch size refers to the number of sequencing reads contained in a single FASTQ file provided as input to BWA-MEM. Smaller batch sizes enable a greater degree of parallelism, as more individual jobs can be distributed across available processors. However, this can lead to increased overhead due to repeated initialization of the BWA-MEM process. In contrast, larger batch sizes reduce initialization overhead but limit parallelization, as fewer jobs are available for distribution. Thus, determining an optimal batch size is essential to achieve efficient resource utilization.

Two execution modes were considered: single-threaded (ST) and multi-threaded (MT). In the ST mode, each job is allocated a single vCPU, enabling concurrent execution of multiple jobs per instance. While this increases parallelism and CPU utilization, it may also introduce significant overhead from redundant BWA-MEM initializations and lead to contention for shared resources. In the MT mode, a single job is executed using multiple vCPUs (via the `-t` option in BWA-MEM), potentially reducing overhead and resource contention. However, this comes at the cost of lower parallelization, as the number of simultaneous jobs per instance is reduced in proportion to the number of threads used.

## 3. METHODS

### A. Job submission and benchmarking

To run the jobs on our infrastructure and test its performance, considering the parameters above-mentioned, we conducted a series of benchmarking experiments on both sites. We tested different discrete batch sizes of 32k, 128k, 256k and 512k reads per input file. To account for variability in running time, the execution time measurement was repeated 5 times for each batch size value in both ST (1 CPU per job) and MT (2 CPUs per job) modalities. The execution time is measured inside the python script and stored in a log file as the total time necessary to align the entire batch. Lastly, the log files were parsed to calculate average and standard deviation. The values were then normalized by total number of reads to estimate the time to align 1k reads.

To automate the benchmarking process, a bash script was created (`benchmark.sh`). When executed on the master node, the script takes a single-end FASTQ file as input, splits it into three smaller FASTQ files containing roughly equal numbers of reads, and prepares a corresponding HTCondor submit file to distribute the workload across the three nodes. Each split file is processed independently by a Python script (`bwa_alt.py`) using the reference genome. The script ensures consistent naming of intermediate files, generates appropriate log, output, and error files for each job, and manages the submission to HTCondor. The Python script roughly consists of three main parts:

1. Batching Input FASTQ files were partitioned into smaller subsets of reads using a batching function (`split_fastq`). Each batch consisted of 32,000, 128,000, 256,000, or 512,000 reads. Reads were written into separate files in a 4-line FASTQ format, and each generated batch file was tracked for subsequent alignment and clean-up.
2. Alignment Each batch was aligned independently to the reference genome using BWA-MEM. Two execution modes were compared:
   (a) Parallel ST modality: Multiple batches were processed concurrently using `ProcessPoolExecutor`, with each BWA-MEM invocation restricted to a single thread.
   (b) MT modality: Batches were processed sequentially, with each BWA-MEM run utilizing two threads.
3. Time Profiling and Cleanup Wall-clock execution time for each run was recorded using Python's time module. All generated FASTQ and SAM files were removed after each benchmarking run to ensure a clean environment. Execution times were logged to a log file for downstream analysis.

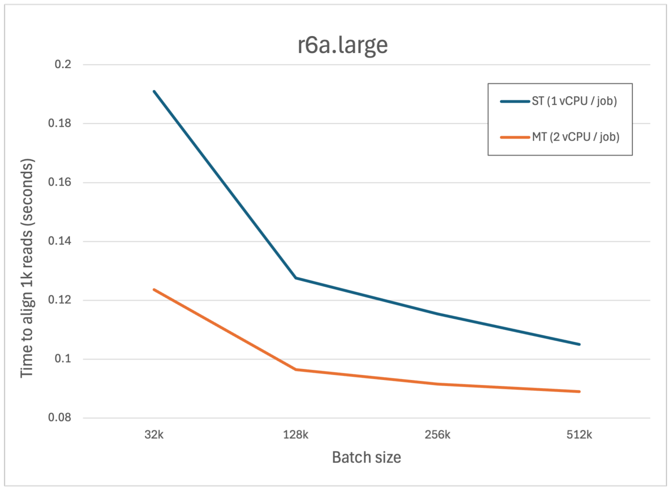


**Fig. 8.** On the x-axis batch sizes and on the y-axis the average time to align a thousand reads. Standard deviation is not reported because negligible.

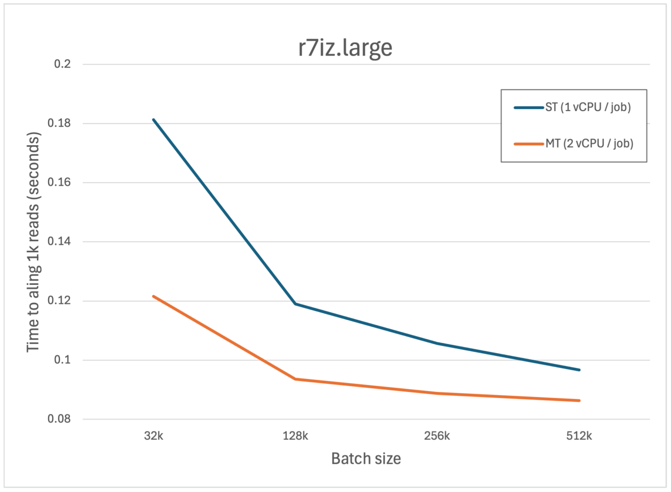


**Fig. 9.** On the x-axis batch sizes and on the y-axis the average time to align a thousand reads. Standard deviation is not reported because negligible.

## 4. RESULTS

### A. Site 1: r6a.large

To evaluate the impact of instance type and computing parameters, a performance assessment was conducted on Site 1. In this site, 3 instances of the type r6a.large are present.

Overall, it is possible to observe a reduction in job execution time by increasing the batch size in both ST and MT modalities (Figure 8). It is quite evident how, for every batch size, performance drastically increases when using the MT modality. However, we can observe how the MT modality performs much better at lower batch sizes, and how the improvement reduces as batch size increases. This observation suggests that further batch size increments will not lead to significant time improvement. The best average running time occurs at batch size of 512k for both ST and MT, with average times of 0.105 ± 0.0019 and 0.089 ± 0.0003 seconds per 1k reads, respectively. It is also interesting to point out how consistent time measurements resulted in very low standard deviation values.

### B. Site 2: r7iz.large

The same performance assessment was conducted in Site 2, which is composed of three instances of the r7iz.large type.

We can observe a comparable trend as in Site 1, with a reduction in execution times by increasing the batch size (Figure 9). As expected, the recorded times for all modalities and batch sizes were systematically faster than the respective times in Site 1. The best average running time also occurs at batch size of 512k for both ST and MT, with average times of 0.0967 ± 0.001 and 0.0863 ± 0.0003, respectively.

### C. Time and cost estimation

Based on the analysis conducted across both sites, the optimal computing configuration was determined to be a batch size of 512k combined with multi-threaded (MT) execution. Using the time required per vCPU to align 1,000 reads, we estimated the total time and cost needed to align 35 billion reads on a three-node CPU farm (Table 1). The r7iz.large instances, featuring DDR5 memory and Intel processors, showed higher alignment speed compared to the r6a.large instances, which use DDR4 memory and AMD processors. However, this performance advantage came at a higher financial cost. Specifically, aligning 35 billion reads on r6a.large instances required approximately 288.8 hours, at a total cost of $123.18. In contrast, the r7iz.large instances completed the same task in about 280 hours, but at a cost of $180.43. This represents a modest 3% improvement in speed and a 32% increase in cost.

| Site | Instance | Nodes | TTA 1k (s) | TTA 35B (h) | Cost ($) |
|---|---|---|---|---|---|
| Site 1 | r6a.large | 3 | 0.0297 | 288.75 | 123.18 |
| Site 2 | r7iz.large | 3 | 0.0288 | 280 | 180.43 |

**Table 1.** Time and cost estimation to complete the computational challenge. The total cost estimation is calculated by assuming that the final layout would be comprised of two CPU farms of the same instance type.

It is important to note that this cost assessment has some limitations, most importantly the inability to assess the impact of more nodes per site. Due to the inherent limitations of a star topology, performance is unlikely to scale linearly with the number of nodes.

For this assessment, we assume that both sites will be provisioned with an identical infrastructure. Given the cost-effectiveness of the r6a.large instances, they were selected as the optimal choice. Our final configuration will therefore consist of two CPU farms, each comprising five r6a.large instances. Assuming similar performance across both sites, the total time to align 35 billion reads using this infrastructure is expected to be 144.4 hours, at the same estimated cost of $123.18. It is also essential to account for the storage costs associated with both the input and output files generated by the BWA-MEM alignment process. For this estimation, only the `FASTQ.gz` and `SAM.gz` files were considered, as script files, logs, and `MD5` checksum files are negligible in size. Based on the observed sizes of individual `FASTQ.gz` and `SAM.gz` files corresponding to 512k reads, the total storage requirement for processing 35 billion reads was extrapolated. The resulting estimate amounts to approximately 4.9 TB. To account for temporary uncompressed files and potential variability in `gzip` compression ratios, a 15% overhead was incorporated into the estimation (Table 2).

| Data | Obs. # Reads | Obs. Size | Exp. # Reads | Exp. Size |
|---|---|---|---|---|
| FASTQ.gz | 512k | 32MB | 35G | 2.1TB |
| SAM.gz | 512k | 40MB | 35G | 2.6TB |
| | | | | **5.4TB (4.7TB + 15%)** |

**Table 2.** Observed and expected data sizes for FASTQ and SAM files, with total storage estimation.

Initially, all input files (`FASTQ.gz`) are expected to be located at a single site, with half of them subsequently transferred to a second site. Based on this distribution, the storage requirement is estimated at 5.4 TB for the first site and 2.7 TB for the second site. Additionally, each compute instance is equipped with a 10 GB EBS volume, resulting in 60 GB of total instance storage across six instances. In total, the storage required for the computing infrastructure amounts to approximately 8.2 TB (5.4 TB + 2.7 TB + 60 GB). It is important to note that the storage volumes do not share identical characteristics. The EBS volumes attached to the compute instances are General Purpose SSD (gp3), offering 3,000 IOPS and 125 MB/s throughput by default. In contrast, the mounted storage volumes used for input and output files are General Purpose SSD (gp2), which provide 3 IOPS per GB of provisioned volume size (up to a maximum of 16,000 IOPS at 5,334 GB) and a maximum throughput of 250 MB/s. To estimate storage duration, it should be considered that although the computational challenge itself is expected to last approximately 145 hours, additional time is necessary for setup, file transfers, and temporary file retention before final data migration (e.g., to external hard drives). An additional buffer of around five business days (approximately 120 hours) is added, bringing the total estimated storage duration to 265 hours.

AWS EBS volumes are billed based on the amount of provisioned storage per month, regardless of actual usage time. Pricing details are as follows:

- gp3 volumes: \$0.08 per GB-month, including 3,000 provisioned IOPS and 125 MB/s throughput (AWS EBS Pricing).
- gp2 volumes: \$0.10 per GB-month, with performance scaling proportionally with volume size, up to 16,000 IOPS and 250 MB/s throughput.

Below the costs calculated by AWS Pricing Calculator service:

- **Instance EBS Volumes**
  - 3,000 IOPS / 10 GB = 300.00 IOPS to GB ratio (gp3)
  - 125 MBps / 3,000 IOPS = 0.04 IOPS to Throughput ratio
  - 6 volumes × 265 instance hours = 1,590.00 total instance hours
  - 1,590.00 instance hours / 730 hours in a month = 2.18 instance months
  - 10 GB × 2.18 instance months × 0.08 USD = 1.74 USD (EBS Storage Cost)
  - EBS Storage Cost: 1.74 USD
  - 3,000 IOPS - 3,000 GP3 IOPS free = 0.00 billable GP3 IOPS
  - EBS IOPS Cost: 0.00 USD
  - 125 MBps - 125 GP3 MBps free = 0.00 billable MBps
  - EBS Snapshot Cost: 0 USD
  - Amazon Elastic Block Storage (EBS) total cost (monthly): 1.74 USD
- **Site 1 Mounted Volume**
  - Storage amount per volume: 5.4 TB × 1024 GB per TB = 5,529.6 GB
  - Pricing calculations:
    - 1 volume × 265 instance hours = 265.00 total instance hours
    - 265.00 instance hours / 730 hours in a month = 0.36 instance months
    - 5,529.60 GB × 0.36 instance months × 0.10 USD = 199.07 USD (EBS Storage Cost)
  - EBS Storage Cost: 199.07 USD
  - EBS Snapshot Cost: 0 USD
  - Amazon Elastic Block Storage (EBS) total cost (monthly): 199.07 USD
- **Site 2 Mounted Volume**
  - Storage amount per volume: 2.7 TB × 1024 GB per TB = 2,764.8 GB
  - Pricing calculations:
    - 1 volume × 265 instance hours = 265.00 total instance hours
    - 265.00 instance hours / 730 hours in a month = 0.36 instance months
    - 2,764.80 GB × 0.36 instance months × 0.10 USD = 99.53 USD (EBS Storage Cost)
  - EBS Storage Cost: 99.53 USD
  - EBS Snapshot Cost: 0 USD
  - Amazon Elastic Block Storage (EBS) total cost (monthly): 99.53 USD

The total cost for this challenge, assuming the network bandwidth free of charge, would be:

$$\$1.74 + \$199.07 + \$99.53 + \$123.18 = \$423.52$$

## 5. DISCUSSION

This study aimed to design, deploy, and evaluate a cloud-based Infrastructure as a Service (IaaS) platform on AWS to address a large-scale bioinformatics task: aligning genomic data from 1000 individuals using the BWA-MEM algorithm. The infrastructure spanned two geographically distinct sites within the us-east-1 region, with data shared via a Network File System (NFS) and job management coordinated through HTCondor. Cross-site data transfer was facilitated using WebDAV. Two AWS instance types, r6a.large and r7iz.large, were evaluated for their suitability in terms of computational performance and cost-efficiency. Throughout the design phase, emphasis was placed on achieving a balance between scalability, performance, and economic viability.

A key component of the project involved optimizing computational parameters to maximize efficiency. Both infrastructures showed that increasing the input batch size consistently reduced execution times, mainly by minimizing scheduling overheads. Multi-threaded (MT) processing significantly outperformed single-threaded (ST) at smaller batch sizes, but the advantage decreased at larger batch sizes due to saturation effects from I/O limits, memory bandwidth, or BWA-MEM's own scaling limits. An optimal setup with batch size of 512k reads and MT modality was identified, balancing efficiency and resource use.

The comparison between r6a.large and r7iz.large instances showed that the r7iz.large, with DDR5 memory and newer Intel CPUs, outperformed the r6a.large by about 3% in execution time. However, this small performance gain came at a 32% higher cost, which made them less favourable for this application. Therefore, r6a.large was selected for the final deployment, consisting of two CPU farms with three nodes each. Storage requirements, estimated at approximately 8.2 TB, emerged as a significant secondary driver of overall costs. A detailed analysis of EBS storage options revealed that gp3 volumes offer notable advantages over gp2 in terms of both cost-efficiency and performance flexibility. While gp2 volumes require larger storage allocations to achieve higher IOPS, gp3 decouples performance from volume size, allowing independent provisioning of IOPS and throughput. This separation enables more accurate resource allocation and cost control. Moreover, gp3 volumes provide higher baseline performance and are priced more competitively on a per-gigabyte basis, making them a more effective solution for workloads requiring consistent and high-performance storage. The overall projected cost of the infrastructure, including both compute and storage, was $423.52.

While the study successfully evaluates the performance of an AWS-based cloud IaaS for sequence alignment, several limitations must be considered. Larger instance types have been shown to improve cost-efficiency in BWA-MEM, suggesting that further testing with such instances is necessary [5]. The infrastructure uses a star topology, where the master node handles all communication and job management. However, as the number of nodes increases, the master node can become overwhelmed, leading to potential bottlenecks and limiting scalability. To address this, a modified topology could be used to alleviate the master node's load. Additionally, the time and cost analysis are based on a hypothetical scenario, which may not represent the most efficient real-world setup.

## 6. PROGRAM VERSION

| **Program** | **Version** |
|---|---|
| sra-toolkit | 3.2.0 |
| bwa | 0.7.15 |
| HTCondor | 8.8.17 |
| NFS | 1.3.0 |
| Apache (WebDav) | 2.4.6 |
| cadaver | 0.23.3 |

## 7. SUPPLEMENTARY MATERIAL

All supplementary material (executables and results) can be found in this GitHub repository.

## REFERENCES

1. R. Leinonen, H. Sugawara, M. Shumway, and on behalf of the International Nucleotide Sequence Database Collaboration, Nucleic Acids Res. **39**, D19 (2011).
2. S. Kolmykov, G. Vasiliev, L. Osadchuk, *et al.*, Front. Genet. **12** (2021). Publisher: Frontiers.
3. H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," (2013). ArXiv:1303.3997 [q-bio].
4. S. Chen and M. A. Senar, Parallel Comput. **87**, 11 (2019).
5. "A generalized approach to benchmarking genomics workloads in the cloud: Running the BWA read aligner on Graviton2 | AWS Public Sector Blog," (2021). Section: Academic medical centers.