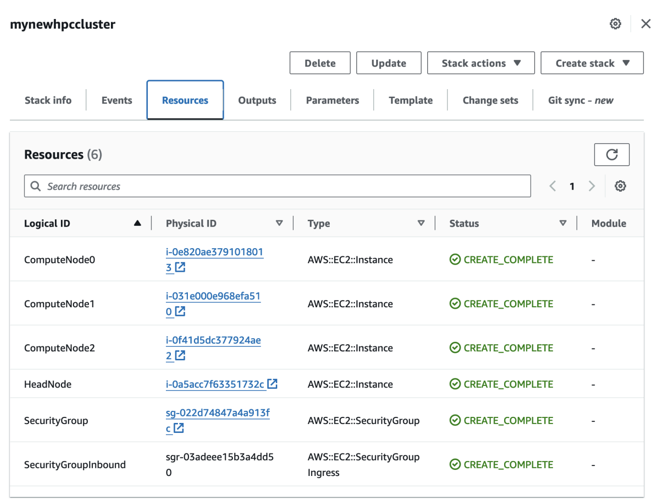
HPC - Cloud - Virtual Cluster

Student: Tiago de Souza Oliveira

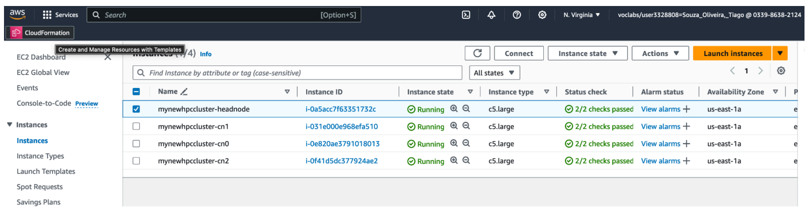
Experimental Configuration (*To create the AWS Virtual Cluster HPC environment*)

The design on the Infrastructure as Code (.yml) for AWS:CloudFormation leveraging the code from https://github.com/micap-hpcn/virtual-clusters/blob/main/template.yaml

HPC c5.large



AWS: Stack resources

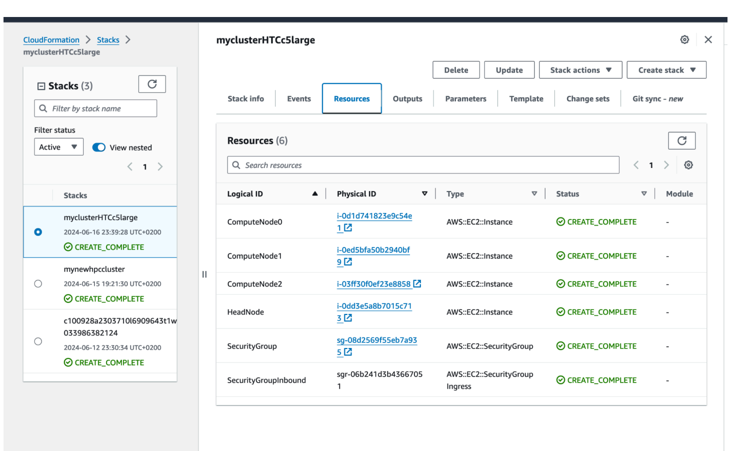


AWS: EC Instances

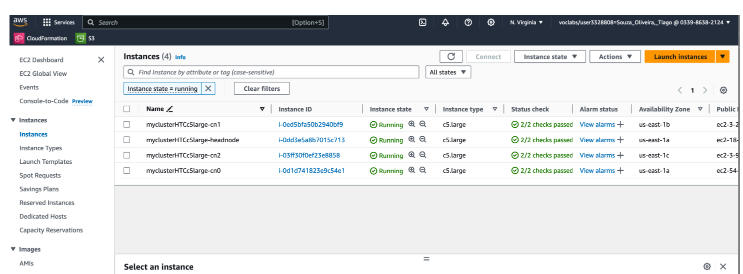
scontrol show nodes

```
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ scontrol show nodes
NodeName=cn1 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.87.110 NodeHostName=ip-172-31-87-110 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3337 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T18:21:00 SlurmdStartTime=2024-06-16T18:21:08
   LastBusyTime=2024-06-16T18:21:05
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

NodeName=cn2 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.86.161 NodeHostName=ip-172-31-86-161 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3339 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T18:21:00 SlurmdStartTime=2024-06-16T18:21:07
   LastBusyTime=2024-06-16T18:21:05
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

NodeName=cn0 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.92.235 NodeHostName=ip-172-31-92-235 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3336 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T18:21:00 SlurmdStartTime=2024-06-16T18:21:07
   LastBusyTime=2024-06-16T18:21:05
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

HTC c5.large



AWS: Stack resources



AWS: EC Instances

scontrol show nodes

aws Services Search [Option+S] CloudFormation S3

```
[ec2-user@ip-172-31-87-84 NPB3.4-MPI]$ scontrol show nodes
NodeName=cn1 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.28.242 NodeHostName=ip-172-31-28-242 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3340 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T23:35:40 SlurmdStartTime=2024-06-16T23:35:46
   LastBusyTime=2024-06-16T23:35:43
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

NodeName=cn0 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.92.164 NodeHostName=ip-172-31-92-164 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3333 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T23:35:39 SlurmdStartTime=2024-06-16T23:35:46
   LastBusyTime=2024-06-16T23:35:43
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

NodeName=cn2 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUEfctv=2 CPUTot=2 CPULoad=0.00
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=172.31.36.245 NodeHostName=ip-172-31-36-245 Version=22.05.11
   OS=Linux 4.14.345-262.561.amzn2.x86_64 #1 SMP Fri May 31 18:15:42 UTC 2024
   RealMemory=3719 AllocMem=0 FreeMem=3332 Sockets=1 Boards=1
   State=IDLE+DYNAMIC_NORM ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=aws
   BootTime=2024-06-16T23:35:39 SlurmdStartTime=2024-06-16T23:35:46
   LastBusyTime=2024-06-16T23:35:43
   CfgTRES=cpu=2,mem=3719M,billing=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

[ec2-user@ip-172-31-87-84 NPB3.4-MPI]$
```

To compile and run

1. Download and unzip the file www.nas.nasa.gov/assets/npb/NPB3.4.2.tar.gz

   [ec2-user@ip-172-31-87-84 mpi]$ tar -xzvf NPB3.4.2.tar.gz

2. Copy /nfs/mpi/NPB3.4.2/NPB3.4-MPI/config/make.def.template to make.def
3. Change parameters in make.def

```
#C based implementation
MPICC = mpicc
CC = $(MPICC)
CFLAGS = -O3 -fomit-frame-pointer -funroll-loops
CLINK = $(CC)
CLINKFLAGS = $(CFLAGS)
CMPI_LIB = -lmpi
#Fortran based implementation
MPIF77 = mpif77
FLINK = $(MPIF77)
FFLAGS = -O3
FLINKFLAGS = $(FFLAGS)
```

[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ module load mpi

[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ make cg CLASS=C
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ make ft CLASS=C
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ make sp CLASS=C
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ make bt CLASS=C

```
make[1]: Leaving directory `/nfs/mpi/NPB3.4.2/NPB3.4-MPI/BT'
[ec2-user@ip-172-31-87-84 NPB3.4-MPI]$ ls -la bin
total 672
drwx------  2 ec2-user ec2-user     62 Jun 16 23:51 .
drwx------ 17 ec2-user ec2-user    235 Jun 18  2021 ..
-rwxrwxr-x  1 ec2-user ec2-user 215280 Jun 16 23:51 bt.C.x
-rwxrwxr-x  1 ec2-user ec2-user  63096 Jun 16 23:51 cg.C.x
-rwxrwxr-x  1 ec2-user ec2-user  76104 Jun 16 23:51 ft.C.x
-rwxrwxr-x  1 ec2-user ec2-user 323840 Jun 16 23:51 sp.C.x
[ec2-user@ip-172-31-87-84 NPB3.4-MPI]$
```

CG - Conjugate Gradient

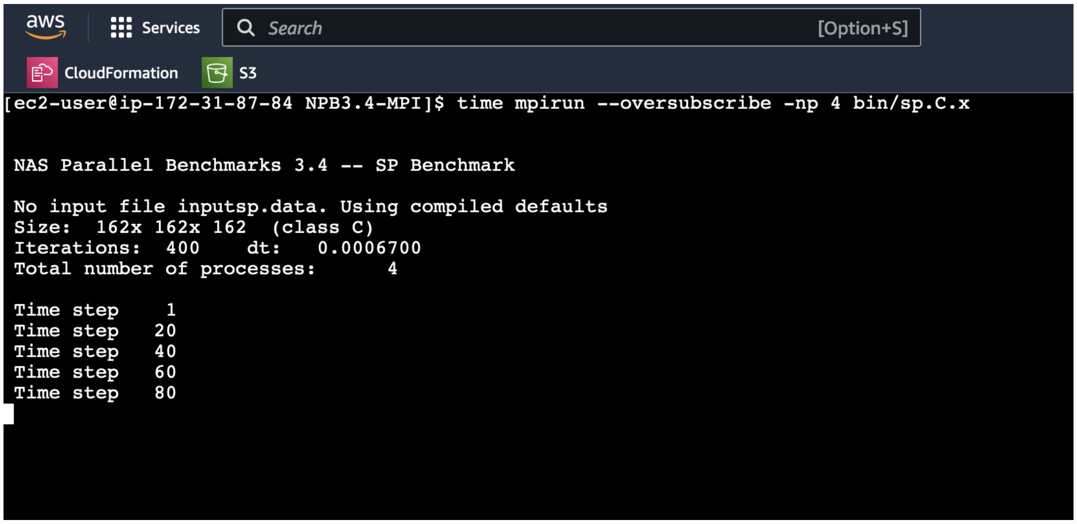
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ mpirun --oversubscribe -np 2 bin/cg.C.x

aws Services Search [Option+S]

CloudFormation

```
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ mpirun -np 2 bin/cg.C.x
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 2
slots that were requested by the application:

  bin/cg.C.x

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.
--------------------------------------------------------------------------
[ec2-user@ip-172-31-94-240 NPB3.4-MPI]$ mpirun --oversubscribe -np 2 bin/cg.C.x


 NAS Parallel Benchmarks 3.4 -- CG Benchmark

 Size:     150000  (class C)
 Iterations:    75
 Number of nonzeroes per row:       15
 Eigenvalue shift:   110.000
 Total number of processes:      2

   iteration           ||r||                 zeta
        1       0.34918028513489E-12   109.9994423237398
        2       0.93261202935200E-15    27.3920437146530
        3       0.94002966917904E-15    28.0339761840276
        4       0.94986672029943E-15    28.4191507551296
        5       0.95373147965057E-15    28.6471670038882
        6       0.95933824815541E-15    28.7812969418424
        7       0.95552953080657E-15    28.8600458537354
```

SP - Scalar Pentadiagonal solver
[ec2-user@ip-172-31-87-84 NPB3.4-MPI]$ time mpirun --oversubscribe -np 4 bin/sp.C.x



Cost of the Experiment in Advance

To estimate the cost from the specified resources in the CloudFormation template script, the following considerations was taken into account regarding charge policy in Aws:

EC2 Instances:

- Head node instance type (default `c5.large`).
- Compute node instance type (default `c5.large`).
- Number of compute nodes (default `2`).

Security Group:

- Although security groups themselves do not incur charges, the instances within them will.

EBS Volumes:

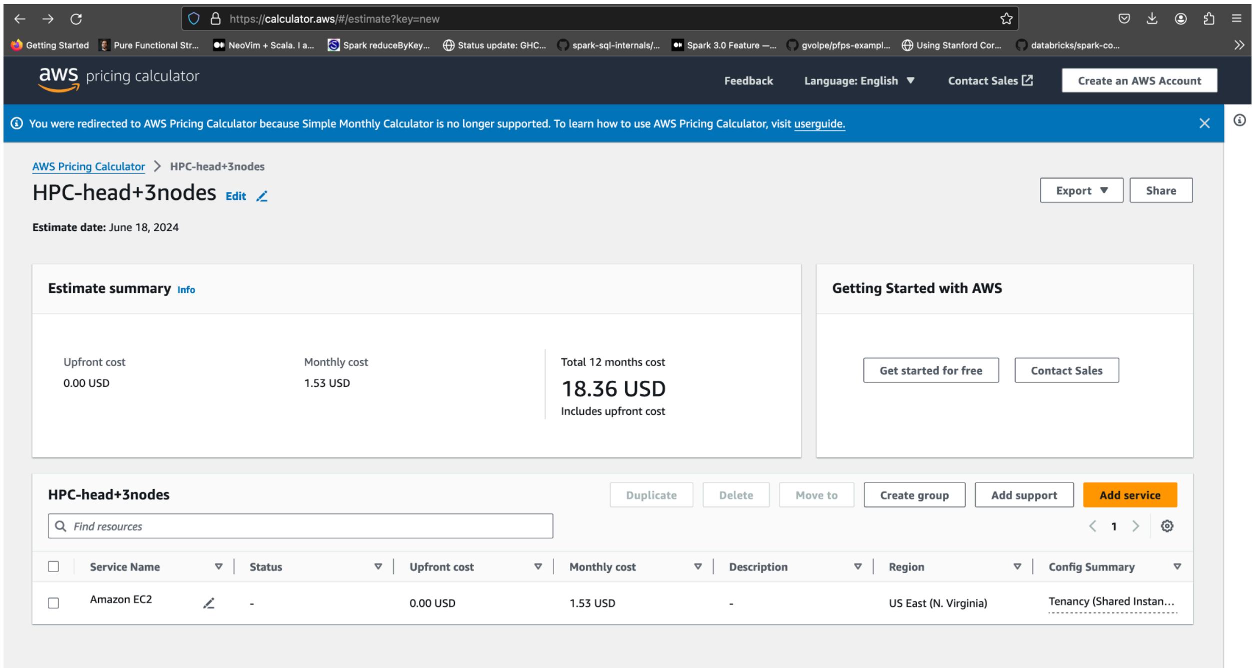
- Implicit with each instance, typically charged based on size and type (gp2/gp3).

Regarding EC2 instances, for 1 head node and 2 compute nodes (default settings):

Total Instances=1(Head Node)+2(Compute Nodes)=3\text{Total Instances} = 1 (\text{Head Node}) + 2 (\text{Compute Nodes}) = 3Total Instances=1(Head Node)+2(Compute Nodes)=3

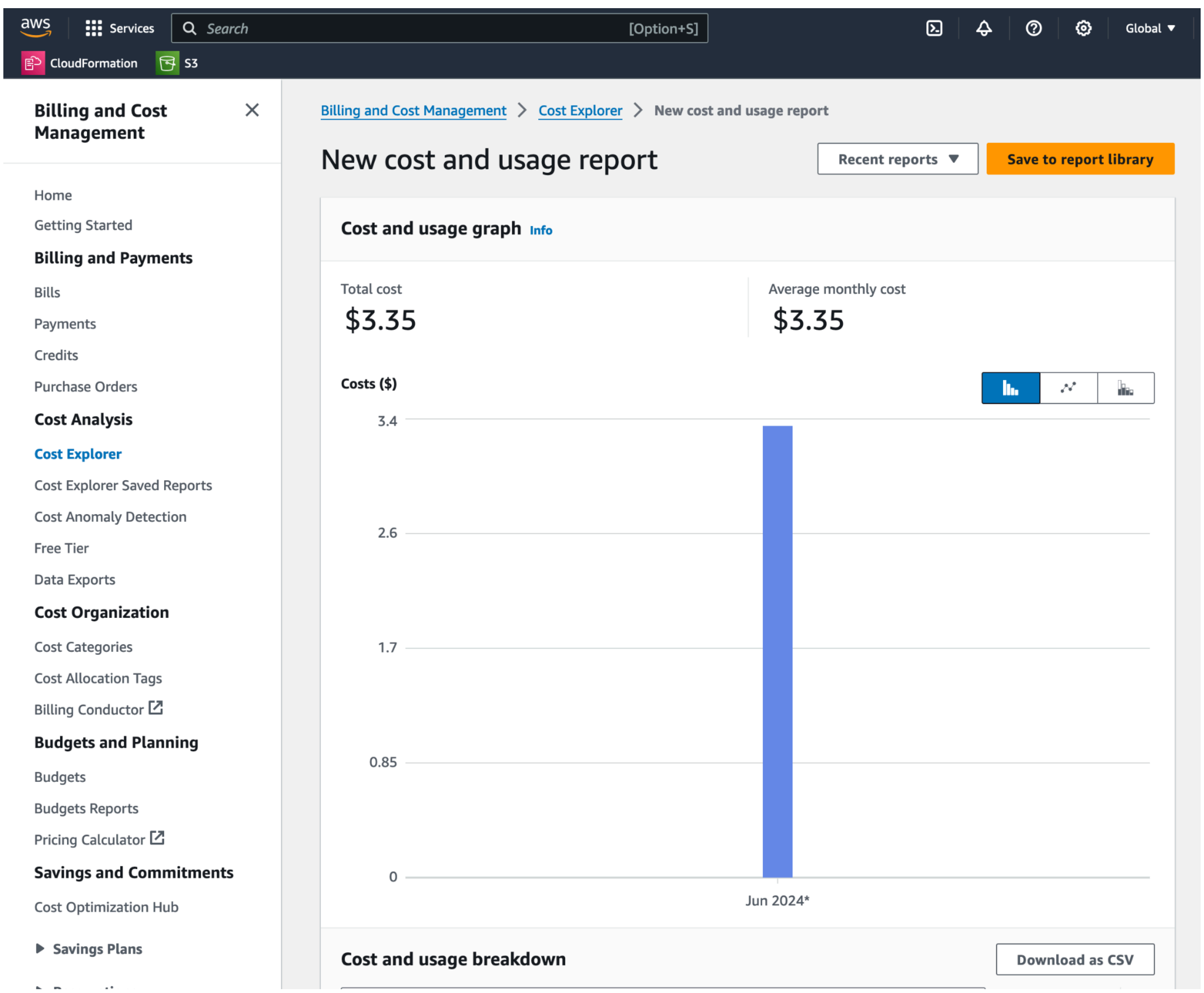
Hourly Cost=3×$0.10=$0.30\text{Hourly Cost} = 3 \times \$0.10 = \$0.30Hourly Cost=3×$0.10=$0.30

The [AWS calculator] can be used to get a more precise estimation.

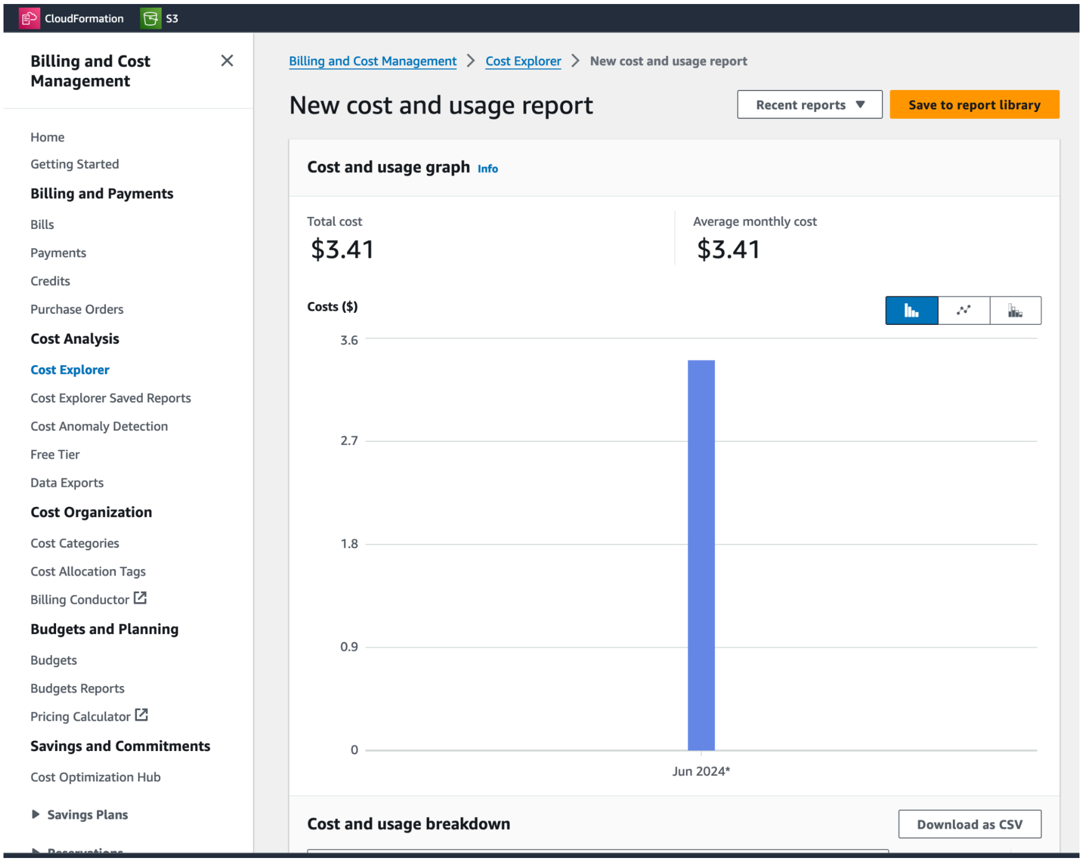


Effective cost of the experiment

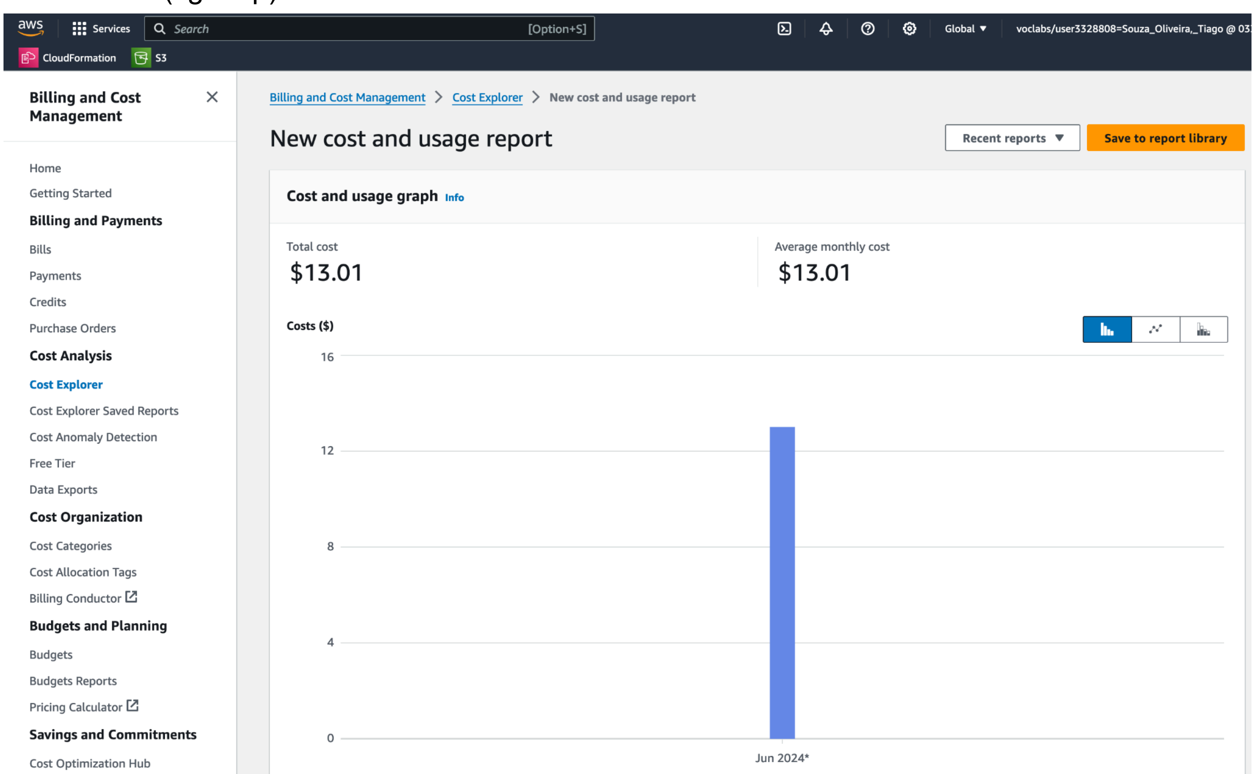
HPC - c5.large (cg, sp)



HTC - c5.large (cg, sp)



HTC + HPC (cg + sp)



Experiment Results

The Conjugate Gradient (cg) kernel was chosen for computationally intensive purposes, while the Scalar Pentadiagonal solver (sp) kernel was chosen for communication intensive purposes.
The submission of the jobs was made in the interactive mode - leveraging mpirun command.

HPC | HTC, c5.large: CG - Conjugate Gradient
$ time mpirun --oversubscribe -np x bin/cg.C.x

HPCOrHTC	c5.large (m) -np 2	c5.large (m) -np 4	c5.large (m) -np 8
HPC (cg)	3.24	3.25	1.34
	3.25	3.24	1.36
	3.34	3.24	1.34
	3.24	3.24	1.35
	3.24	3.24	1.38
avg_hpc	**3.262**	**3.242**	**1.354**
HTC (cg)	3.32	3.22	1.34
	3.35	3.22	1.36
	3.32	3.21	1.34
	3.32	3.22	1.35
	3.24	3.21	1.34
avg_htc	**3.312**	**3.216**	**1.346**

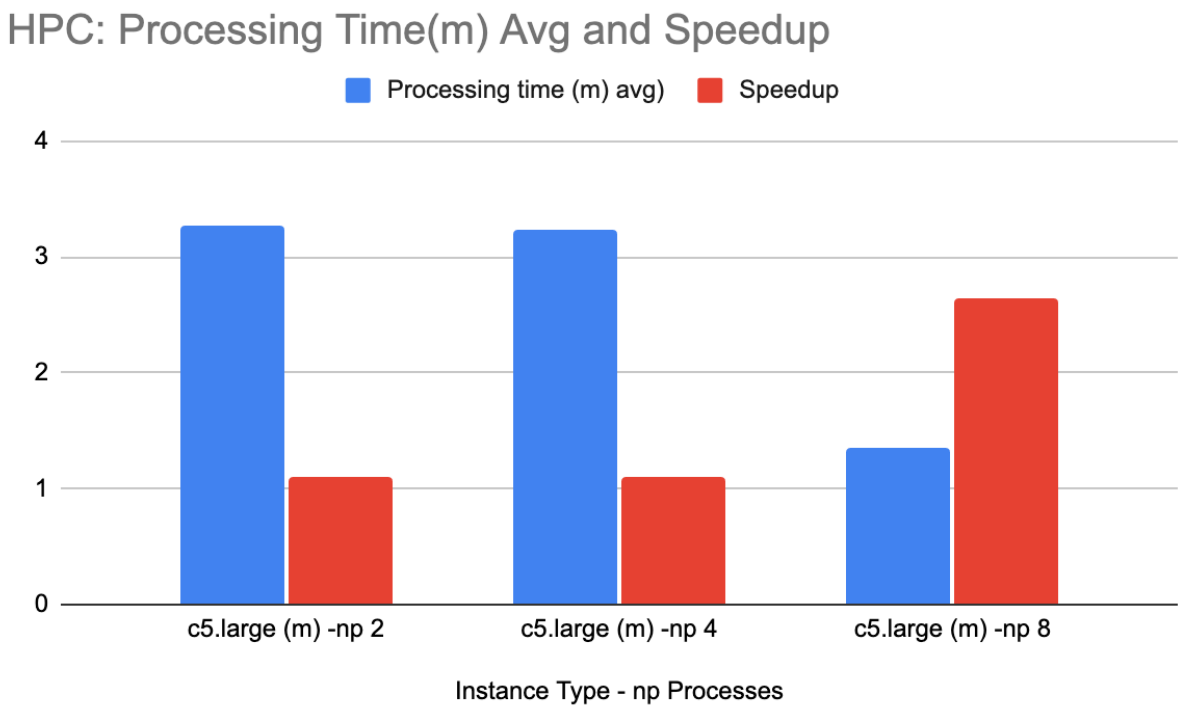
Spedup HPC

The speedup is created by considering the 1 single processor execution average of 3.578 minutes as the following table shows:

	c5.large (m) -np 1
HPC (cg)	3.57
	3.58
	3.58
	3.58
	3.58
avg_hpc	**3.578**

In the following table the speedup counting from 2, 4 to 8 processors:

InstanceType/processes	c5.large (m) -np 2	c5.large (m) -np 4	c5.large (m) -np 8
Processing time (m) avg)	3.262	3.242	1.354
Speedup	1.096817044	1.103639729	2.64254062

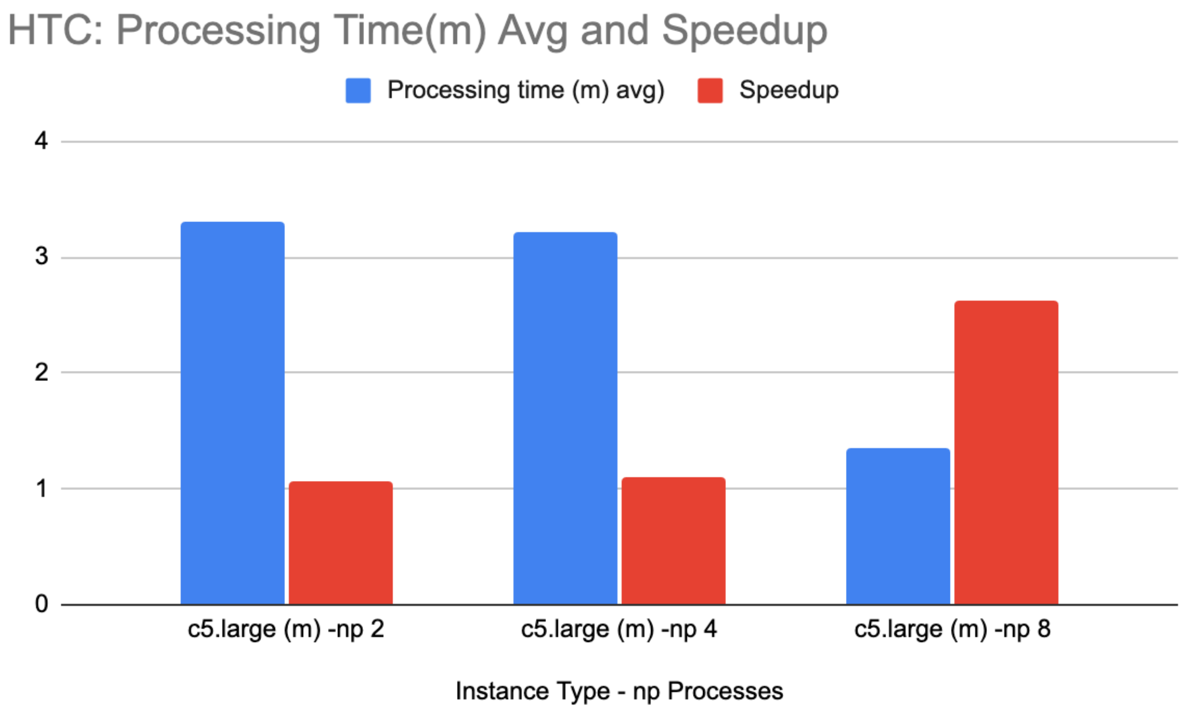


Spedup HTC
The speedup is created by considering the 1 single processor execution average of 3.536 minutes as the following table shows:

	c5.large (m) -np 1
HPC (cg)	3.54
	3.54
	3.54
	3.53
	3.53
avg_hpc	**3.536**

In the following table the speedup counting from 2, 4 to 8 processors:

InstanceType/processes	c5.large (m) -np 2	c5.large (m) -np 4	c5.large (m) -np 8
Processing time (m) avg)	3.312	3.216	1.346
Speedup	1.067761807	1.099502488	2.627043091



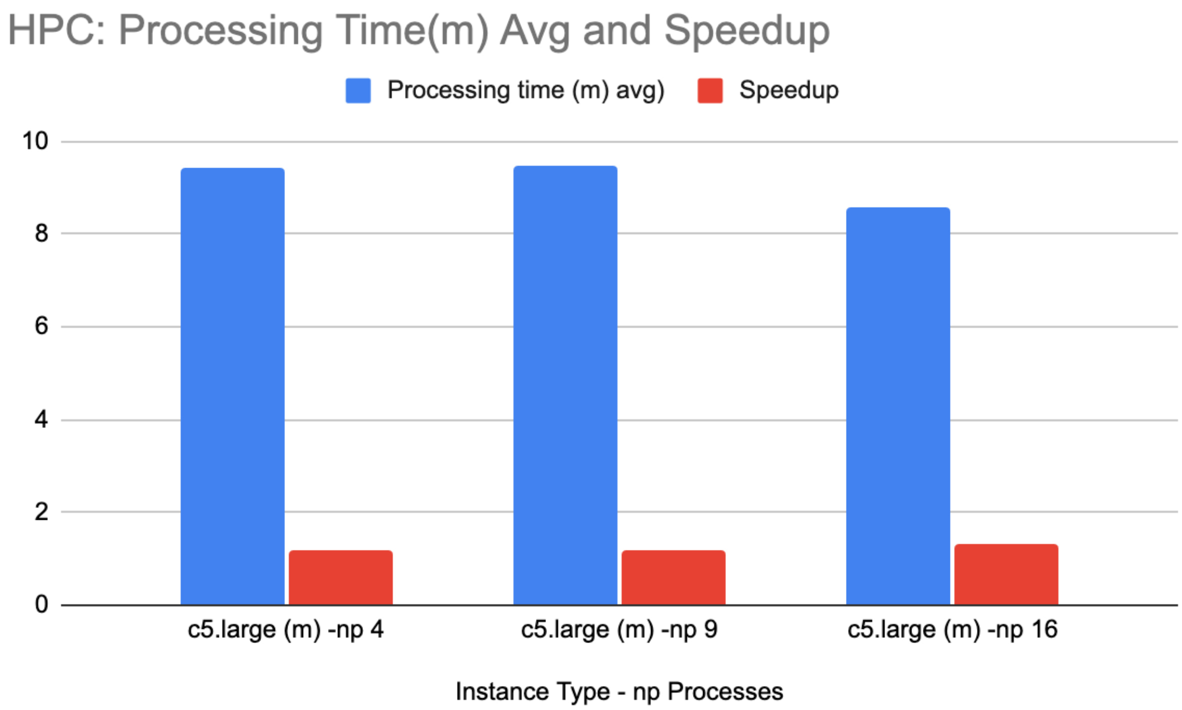
HPC | HTC, c5.large: SP - Scalar Pentadiagonal solver
$ time mpirun --oversubscribe -np x bin/sp.C.x

Spedup HPC
The speedup is created by considering the 1 single processor execution average of 11.206 minutes as the following table shows:

	c5.large (m) -np 1
HPC (cg)	11.2
	11.23
	11.21
	11.17
	11.22
avg_hpc	**11.206**

In the following table the speedup counting from 4, 9 to 16 processors:

InstanceType/processes	c5.large (m) -np 4	c5.large (m) -np 9	c5.large (m) -np 16
Processing time (m) avg)	9.424	9.480	8.576
Speedup	1.189091681	1.182067511	1.306669776

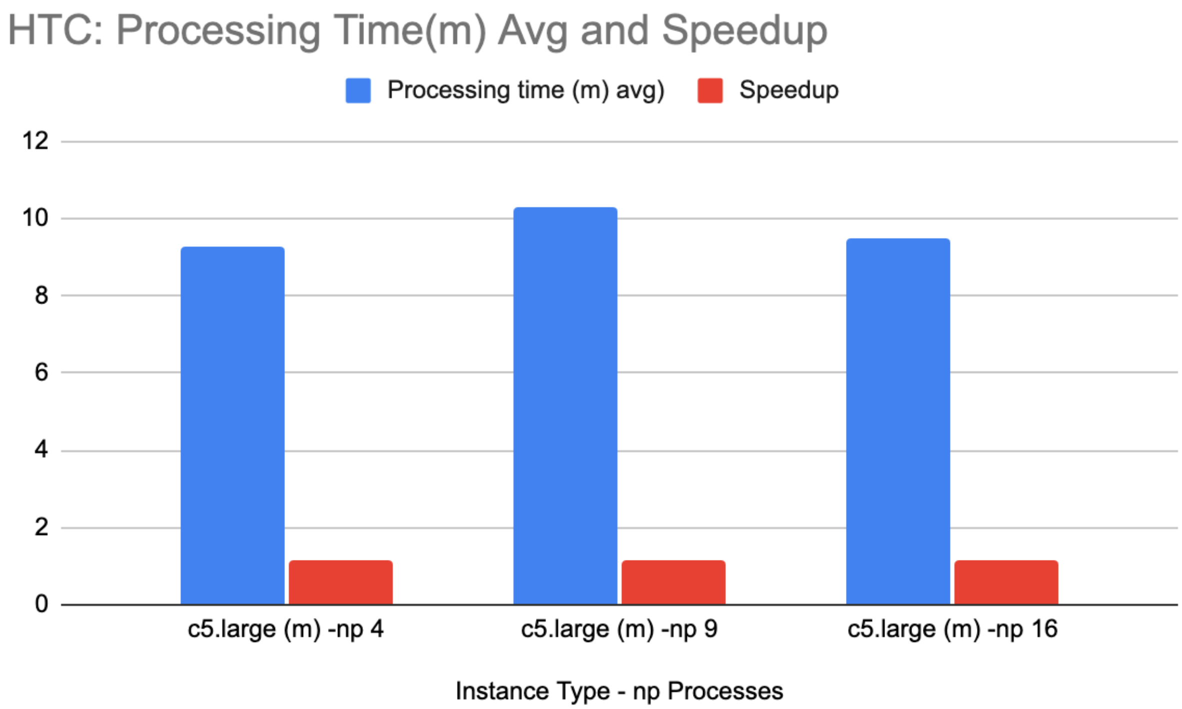


Spedup HTC
The speedup is created by considering the 1 single processor execution average of 10.824 minutes as the following table shows:

avg_hpc	**11.206**
HTC (cg)	10.58
	10.56
	10.58
	11.2
	11.2
avg_htc	**10.824**

In the following table the speedup counting from 4, 9 to 16 processors:

InstanceType/processes	c5.large (m) -np 4	c5.large (m) -np 9	c5.large (m) -np 16
Processing time (m) avg)	9.286	10.320	9.478
Speedup	1.165625673	1.165625673	1.142013083



Conclusion

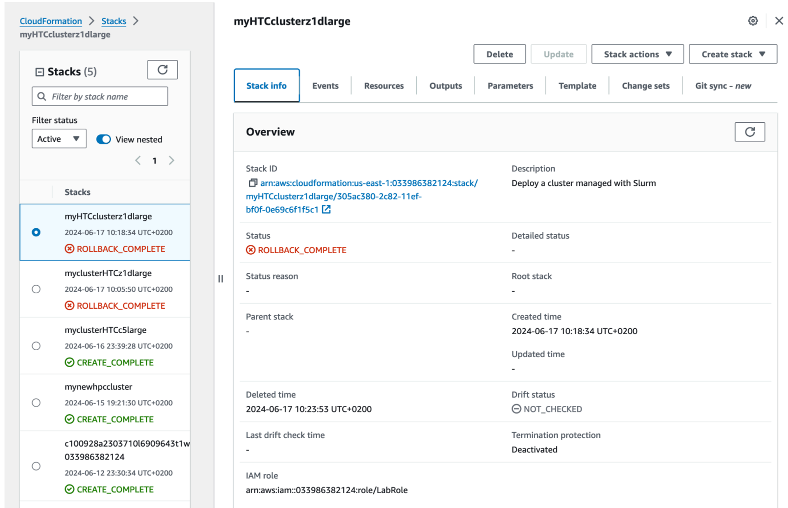
For the intense computing kernel (Conjugate Gradient - cg), HPC and HTC stack setup shows a quite similar scalability behavior. Both having a great result by adding more processors, as can be seen in the speedup report: by scaling from 2 to 8 processors, the performance was shown to double.
For the communication intense kernel (Scalar Pentadiagonal solver - sp), HPC and HTC stack setup shows a slight difference where HPC can bring better scalability, for the sake of comparison with HTC. However, both stacks does not show great scalability, by considering scaling from 4 to 16 processors, the performance, at most, improves 20% - what can be understood from the speedup values.
One behavior that makes sense to study is the worsening of performance when scaling from 4 to 9 processors for the SP kernel.

References

Raw statistics collected can be seen here
https://docs.google.com/spreadsheets/d/1ZmZr5VZmGPUwNsqNyuLKeQTZ3E3qurrC111uUCQKPH0/edit?usp=sharing

Caveats

1. The EC2 instance type z1d.large was used for creating HTC stack template, but resulting in error as can be seen in the following print.



2. Sometimes the connection and session with head nodes via aws console was suddenly interrupted, resulting in the loss of the results in the terminal when firing mpirun commands. To mitigate this issues, the results of commands was added the parameter “>” to sink output to a file. Besides that, sometimes also jobs seemed to be terminated, what could be understood by visualizing output files with incomplete execution logs.
3. For next steps, would be relevant for the study to evaluate other instance types from two categories: one for low latency for intercommunication and for; another for most demanding CPU power. That can be reasoned from the aws page https://aws.amazon.com/es/ec2/instance-types/
4. Also would be interesting to reason about GPU accelerated EC2 instance types. For that, a kernel GPU enabled must be chosen to assess the speedup and cost relationship.