CloudBank 2 HPL Benchmark Report

What is the benchmark

The High Performance Computing Challenge (HPCC) benchmark suite is used to evaluate the performance and cost of variable cloud compute instances in a comprehensive way. Specially HPL performance in the HPCC tests were comparatively analysed with its performance and price on spot instance use in different hardware architectures under various cloud environments: AWS, Google Cloud, Azure.

Main purposes:

- To measure HPL performance and its costs across diverse cloud compute instances in different system architectures (AMD, INTEL, ARM) under various cloud services (AWS, Google Cloud, Azure).
- To provide comparison of their performance and spot instance costs on the instances used.

Considered systems in AWS:

CPU systems	**vCPUs**	**Memory (GB)**	**AWS instances**
4th gen. AMD Epyc 9R14	**4,32,64,192**	**16,64,128,256,768,1536**	**M7a.xlarge, m7a.8xlarge, m7a.16xlarge, c7a.8xlarge,r7a.8xlarge, m8a.8xlarge, m8a.48xlarge,r8a.48xlarge**
ARM (Graviton3,4)	**4,32,64**	**16,64,128,256**	**M7g.xlarge, m7g.8xlarge, m7g.16xlarge, m8g.8xlarge**
4th Gen. Intel Xeon 8488C	**4,32,64**	**16,64,128,256**	**M7i.xlarge, m7i.8xlarge, m7i.16xlarge, m8i.8xlarge**

How is it being run

The OS of Ubuntu (corresponding AMI) is used to build the benchmark environment. Instance launch script appended in this report is used. EC2 launch script needs to be updated to the AMI ids frequently to reflect the current version of the OS.

- ## Compilation

  On each new instance, OpenMPI and HPCC are built with SPACK HPC package manager. Compilation optimization flag “-O2” or “-O3” is used with gcc. Other options can be used like ‘spack install cflags="-O3 -mavx512" cppflags="-O3 -mavx512"’

- ## Problem size

  Approximately 70% of physical memory is used by setting the HPL N parameter in the following way:

  N= sqrt(Memory X 0.7 / 8)

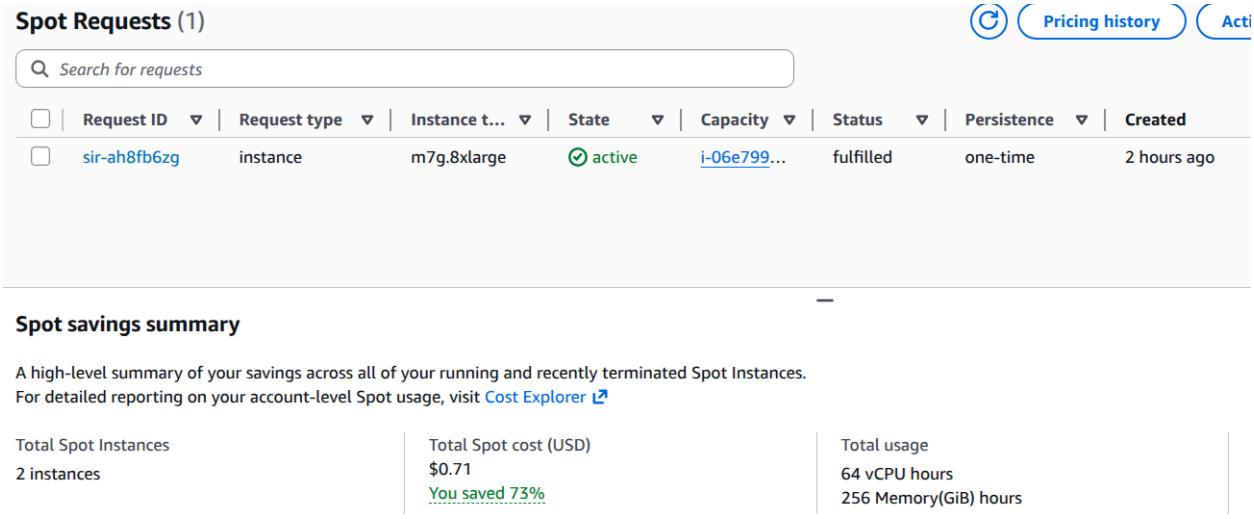
  N size for HPL runs

	MEM Size (GB)							
Mem ratio (%)	16	64	128	256	384	512	768	1536
70	39000	77500	109000	155000	189000	220000	270000	380000

- ## RUN

  Each case is run by a spot EC2 launch script shown in the appendix. Actual start of the instance will be when the spot capacity is available for the request. The following image

shows the spot instance is in the capacity queue with price information at the time of the information status.



Spot instance gives discount on the use of the same resource compared to on-demand price of the instance use. Considering resource capacity availability and eviction risk, planning of the instance type and time, and checkpoint implantation are necessary. More information is available in the documents linked in the reference.

Results and Analysis

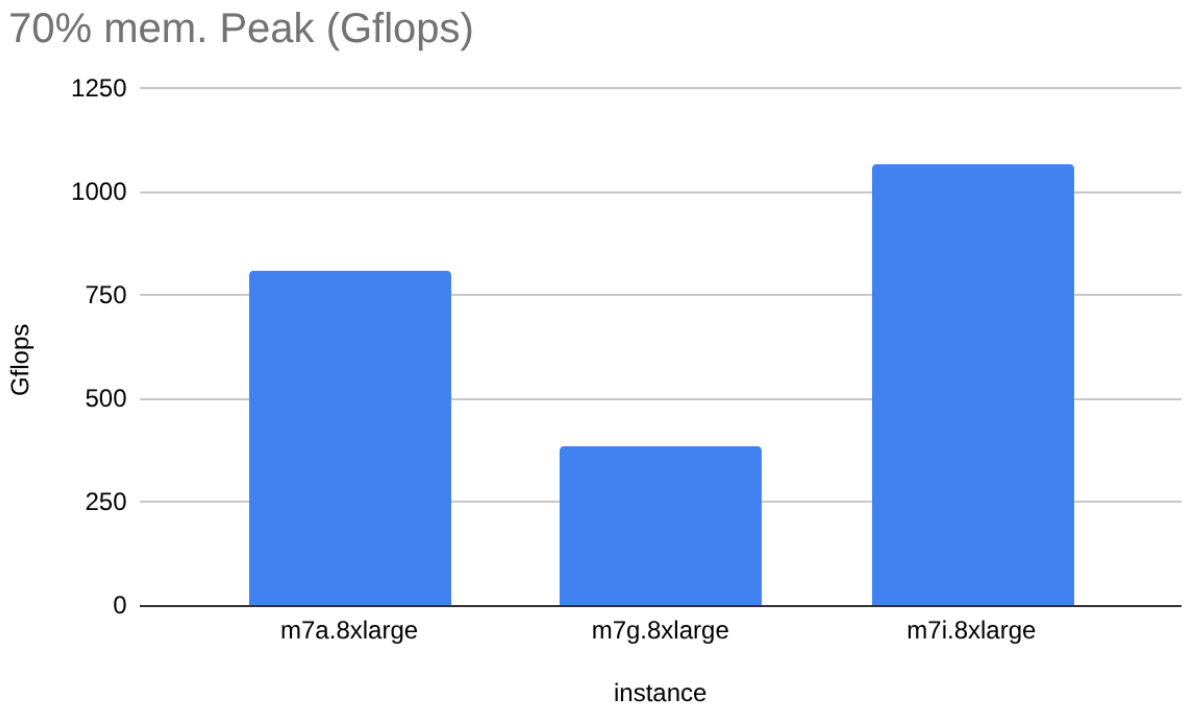
AWS m7 series HPL Performance and cost

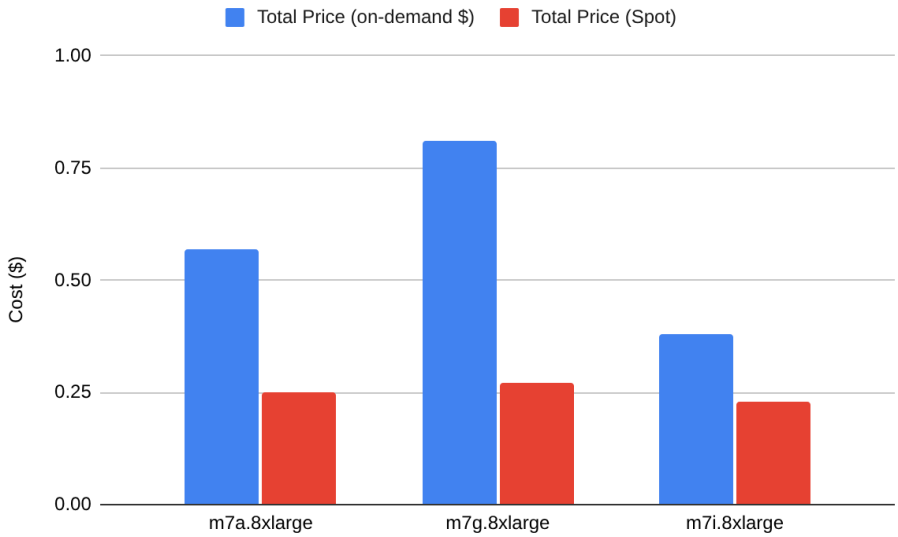
First we are considering the older generation of AWS instances: m7 series. Performance and costs of this AWS instance series is obtained.

instance	CPU	70% mem. Peak (Gflops)	Time took for test (secs)	Total Price (on-demand $)	Total Price (Spot)
m7a.8xlarge	4th gen. AMD Epyc 9R14	808	1068	0.57	0.25
m7g.8xlarge	Graviton3	385	2245	0.81	0.27
m7i.8xlarge	4th Gen. Intel Xeon 8488C	1067	814	0.38	0.23

Results shown in the above table are the average from 3 repetition data shown below.

instance	Run 1			Run 2			Run 3		
	70% mem. Peak (Gflops)	Time took for test (secs)	Hourly. Price (Spot)	70% mem. Peak (Gflops)	Time took for test (secs)	Hourly Price (Spot)	70% mem. Peak (Gflops)	Time took for test (secs)	Hourly Price (Spot)
m7a.8xlarge	812	1062	0.8851	779	1108	0.8557	834.1	1035.13	0.8314
m7g.8xlarge	385	2244	0.3800	384.8	2244	0.4655	384.4	2246	0.4687
m7i.8xlarge	1120	771	0.6696	1128	765	1.6333	952.9	906	0.6983



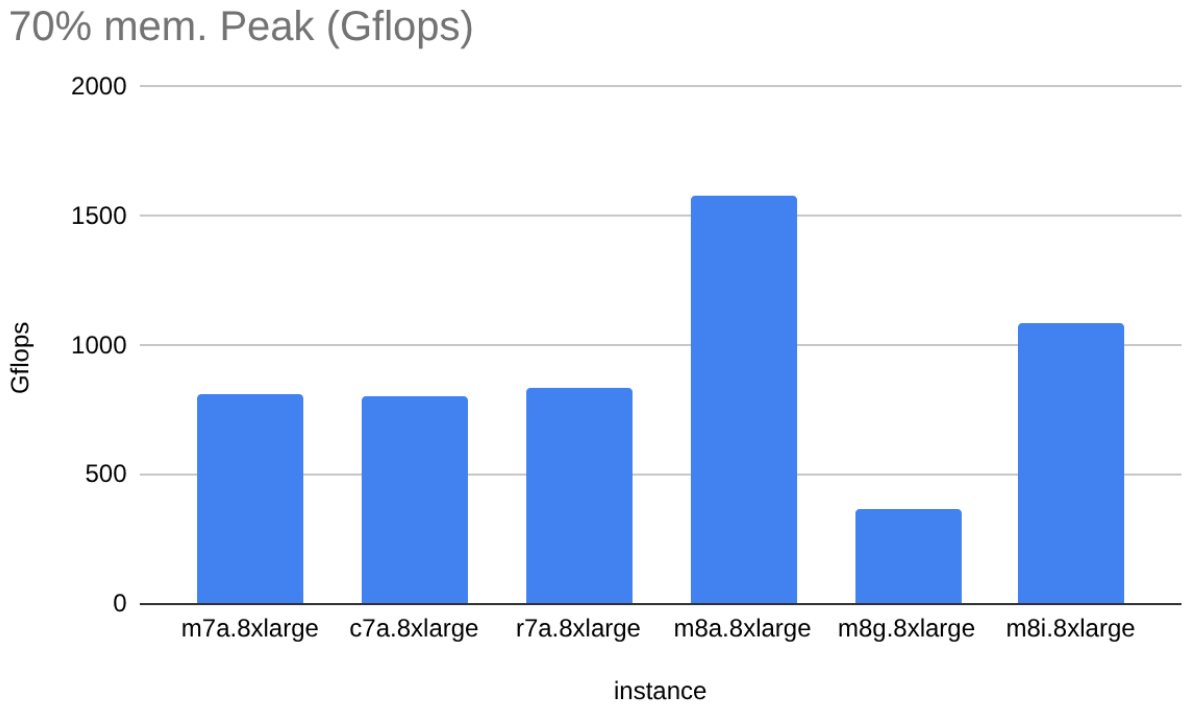


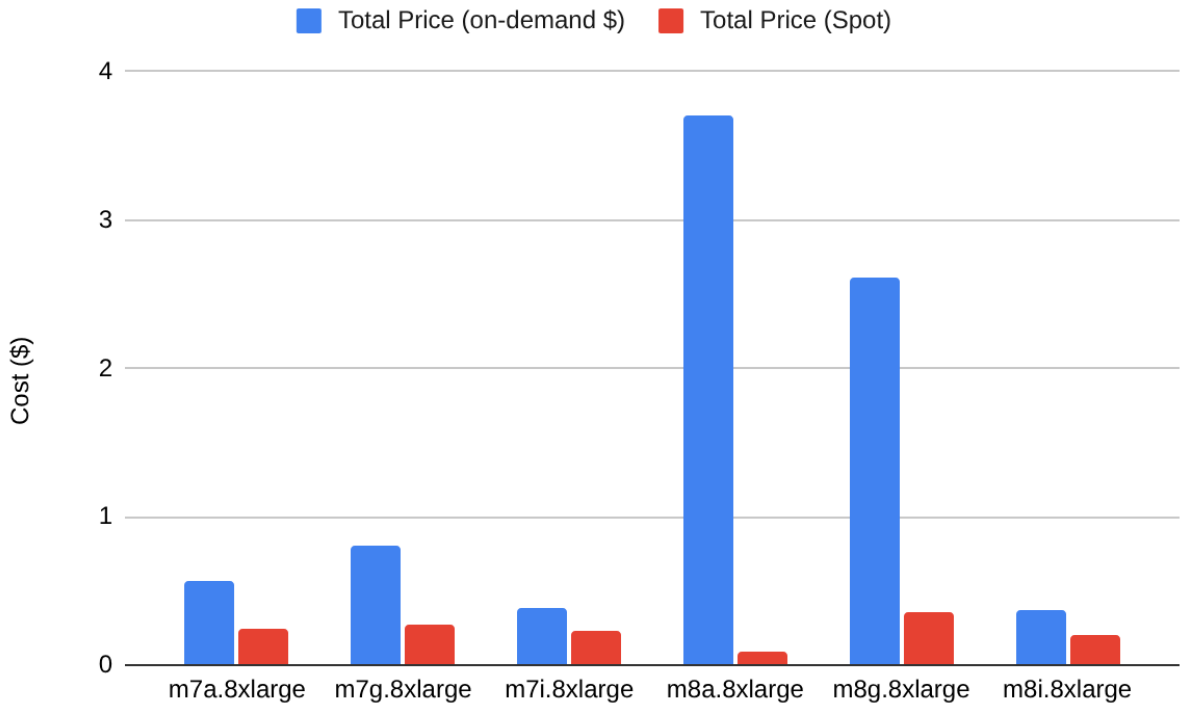
Spot price (red bar) shows almost the same for every considered instance types (AMD Epyc GR14, Graviton 3 and Intel Xeon 8488C). Looking at the 70% memory performance the Intel Xeon 8488C yielded the best performance (shortest run time).

Updated Generation (m8) Instances

Here we are considering updated instance types of m8 series.

instance	CPU	70% mem. Peak (Gflops)	Time took for test (secs)	Total Price (on-demand $)	Total Price (Spot)
m8a.8xlarge	4th gen. AMD Epyc 9R45	1581	546.25	3.71	0.09
m8g.8xlarge	Graviton4	366	2362	2.61	0.35
m8i.8xlarge	Intel Xeon Granite Rapids	1090	792	0.37	0.2





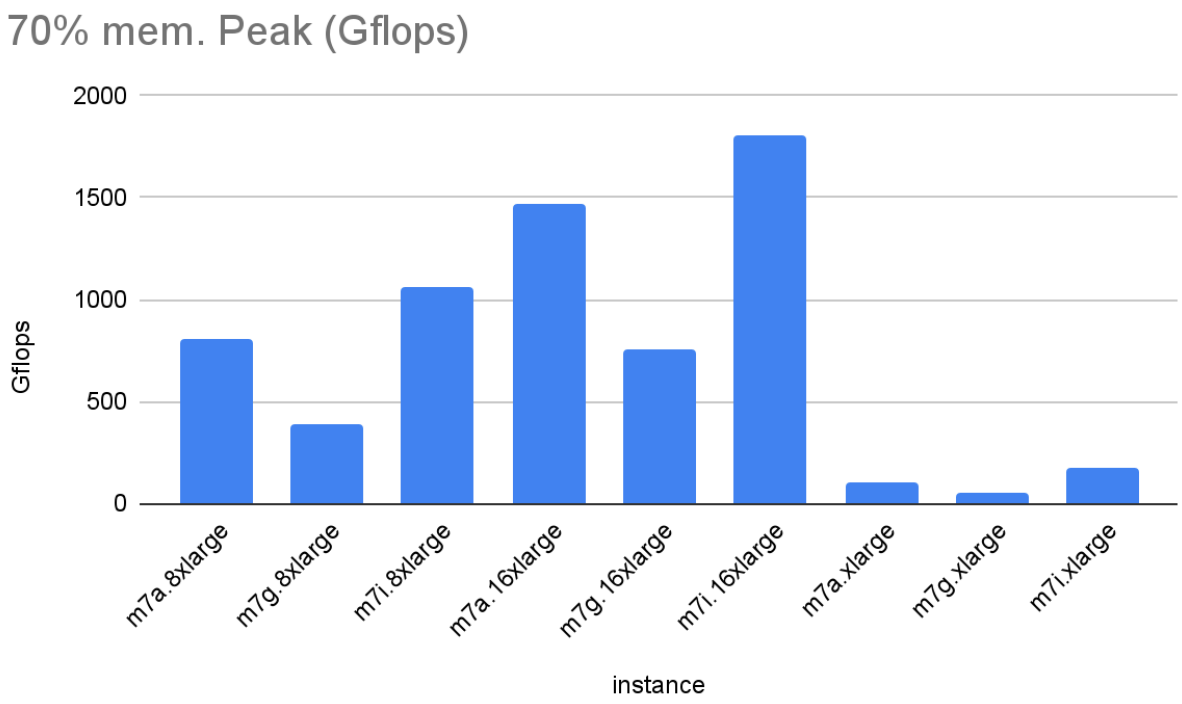
The upgraded m8 instance series yielded great performance and spot price improvement compared to m7 series instance except the Gravition hardware type.

Spot price (red bar) shows the best using m8a instance (AMD Epyc GR45). Ondemand cost of the instance was the highest but the actual spot cost was much lower. Also the performance using this AMD m8 instance yielded the best. Performance by turbo clock speed on each core process must have produced the best on this hardware type.

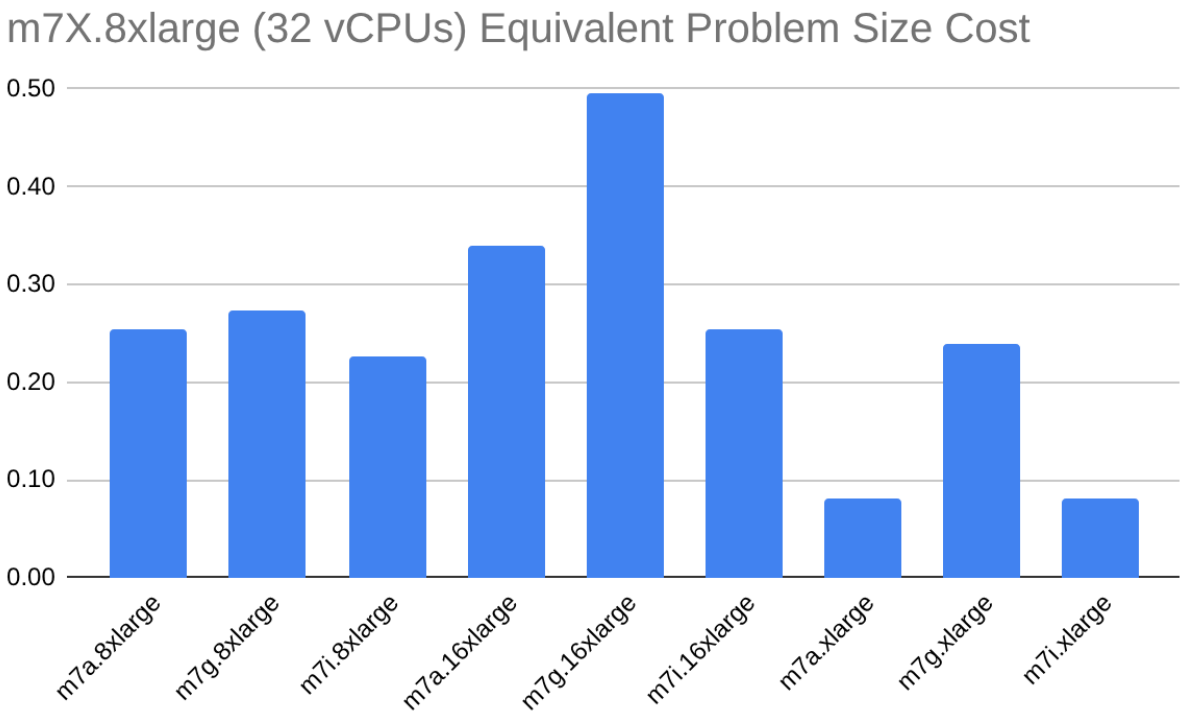
Larger/Smaller Instances

Instances with larger (m7X.16xlarge: 64vCPUs) and (m7X.xlarge: 4vCPUS) were tested to observe its performance and price/cost for the benchmark test.

instance	CPU	70% mem. Peak (Gflops)	Time took for test (secs)	Total Price (on-demand $)	Total Price (Spot)
m7a.16xlarge	4th gen. AMD Epyc 9R14	1472	1686	3.71	0.68
m7g.16xlarge	Graviton3	759.2	3270	2.61	0.99
m7i.16xlarge	4th Gen. Intel Xeon 8488C	1809	1372	1.23	0.51
m7a.xlarge	4th gen. AMD Epyc 9R14	106.6	371	0.02	0.01
m7g.xlarge	Graviton3	48.9	808	0.04	0.03
m7i.xlarge	4th Gen. Intel Xeon 8488C	171.5	231	0.01	0.01



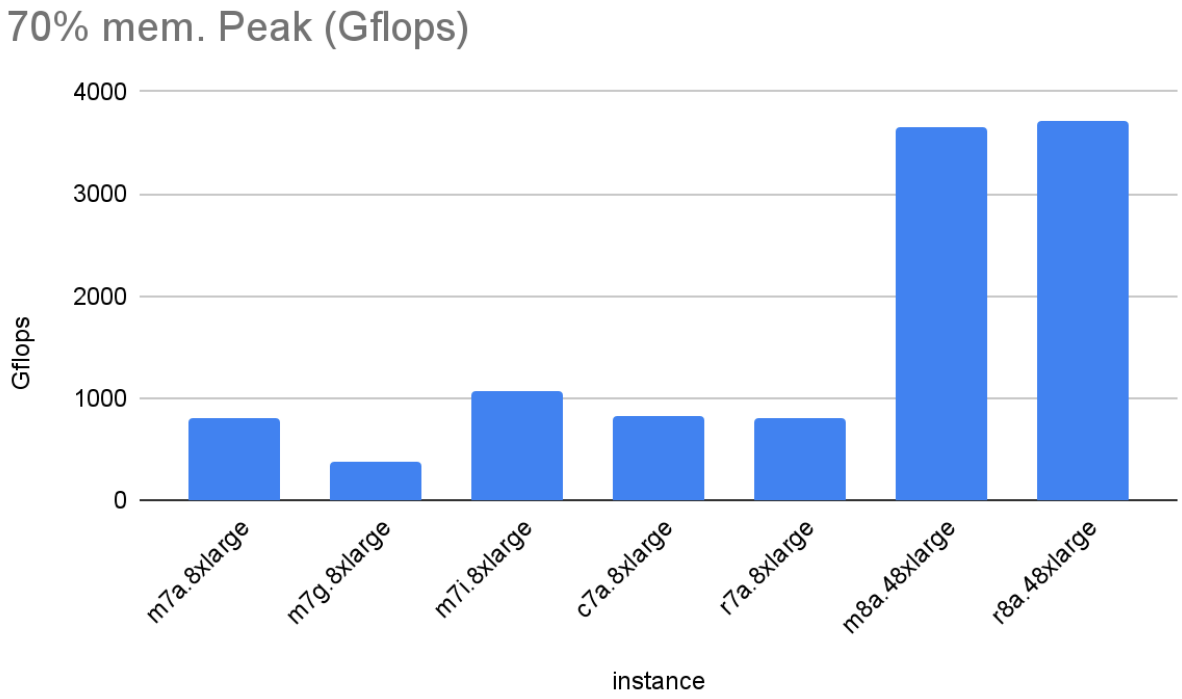
Performance and cost (price) looks scaling to the instance type size (vCPUS).
A naive price/cost estimation on each vCPUs were compared to yield the equivalent problem size for m7X.8xlarge (32 vCPUs) case:



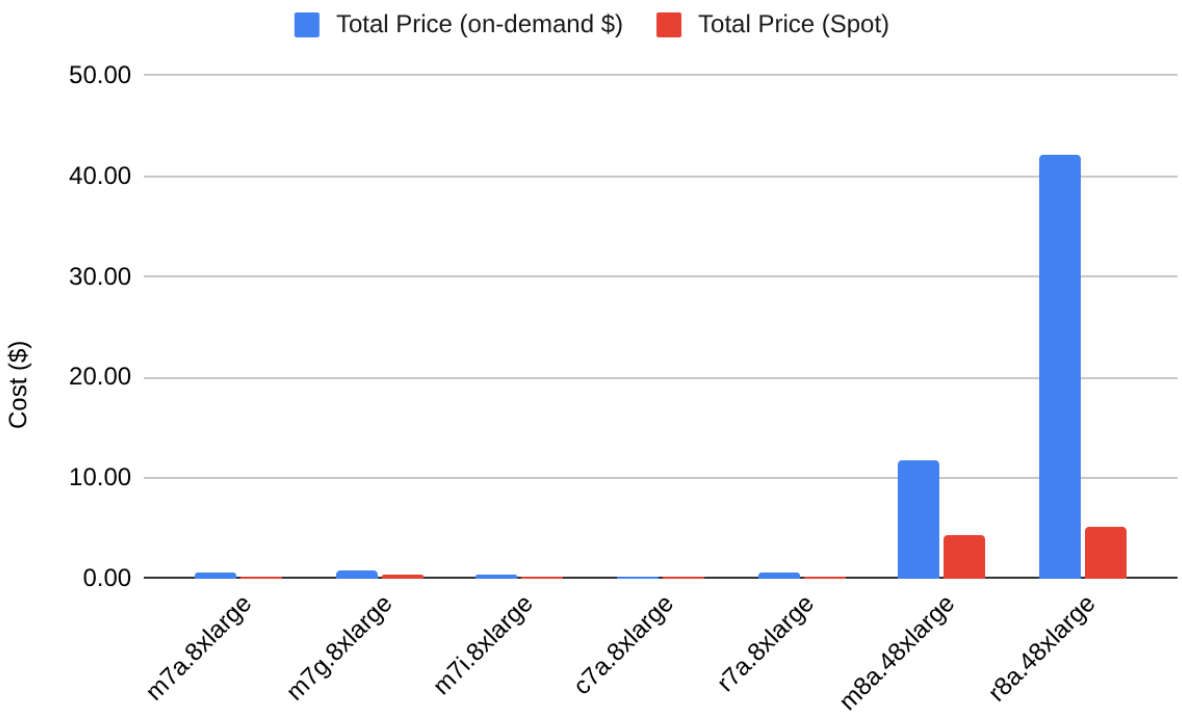
Here m7X.xlarge looks the best cost estimation to run the same problem size. However, considering other actual cost, network first, will go higher.One clear observation is that using large vCPU instances would not help to reduce the cost to run smaller problem sizes.

Memory or CPU/Optimized Instances

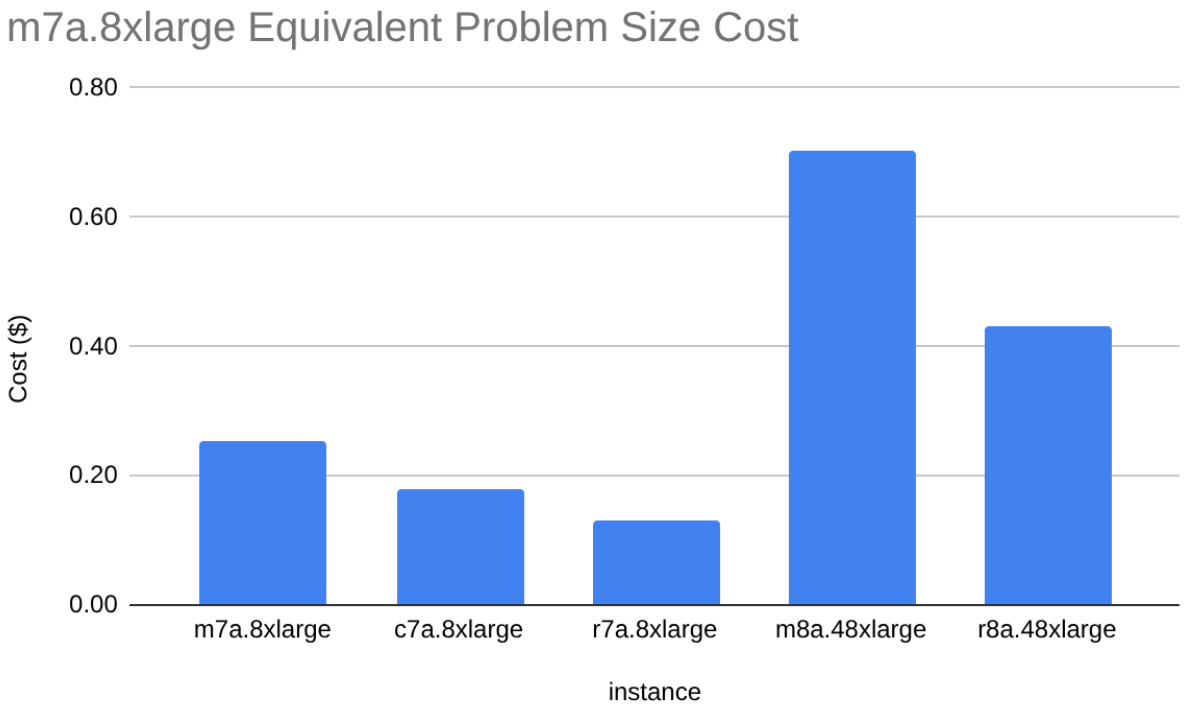
instance	CPU	70% mem. Peak (Gflops)	Time took for test (secs)	Total Price (on-demand $)	Total Price (Spot)
c7a.8xlarge	4th gen. AMD Epyc 9R14	817	380	0.17	0.09
r7a.8xlarge	4th gen. AMD Epyc 9R14	812	1062	0.56	0.26
m8a.48xlarge	4th gen. AMD Epyc 9R14	3658	3587	11.64	4.21
r8a.48xlarge	4th gen. AMD Epyc 9R14	3706	9871	42.05	5.19



Performance and cost (price) also looks scaling to the instance type size (vCPUS) here.



Clearly using larger instances (m8a.48 or r8a.48) yields much higher cost to run the equivalent m7X.8xlarge problem size. Considering the cost, using larger instances looks uneconomical.



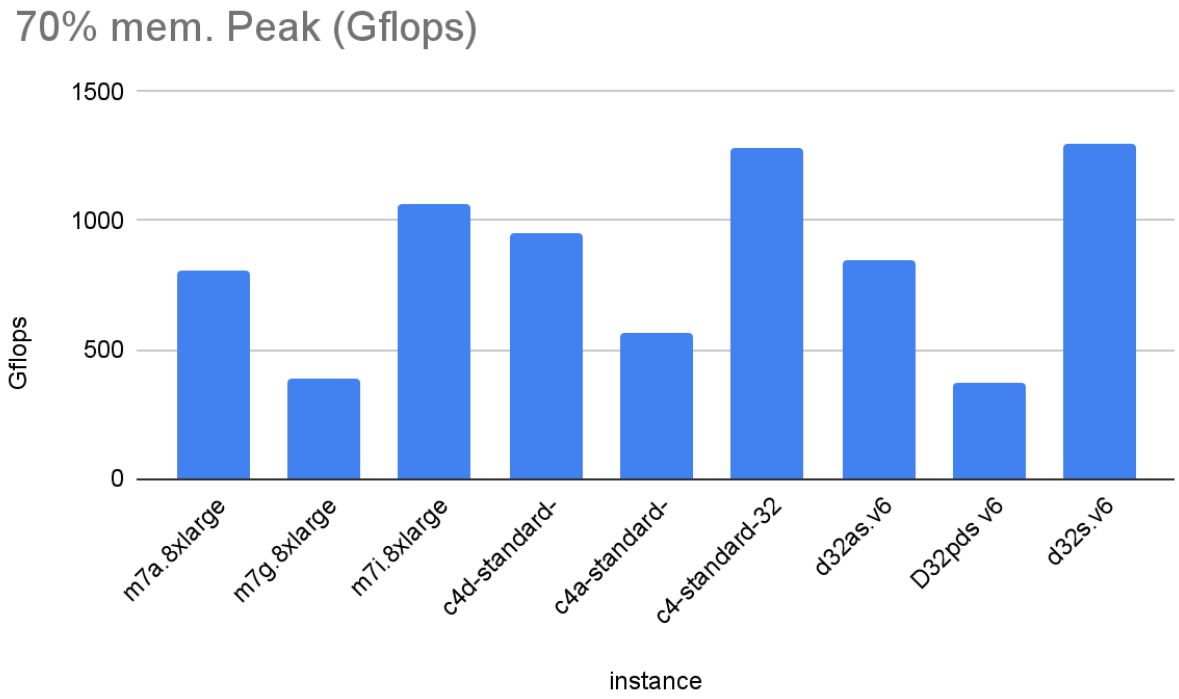
A naive price/cost estimation on each vCPUs were compared to yield the equivalent problem size for m7X.8xlarge (32 vCPUs) case.
Here r7a.xlarge (Memory obtimized instance) looks the best cost estimation to run the same problem size.

Comparable Instances (VMs) on Other Clouds

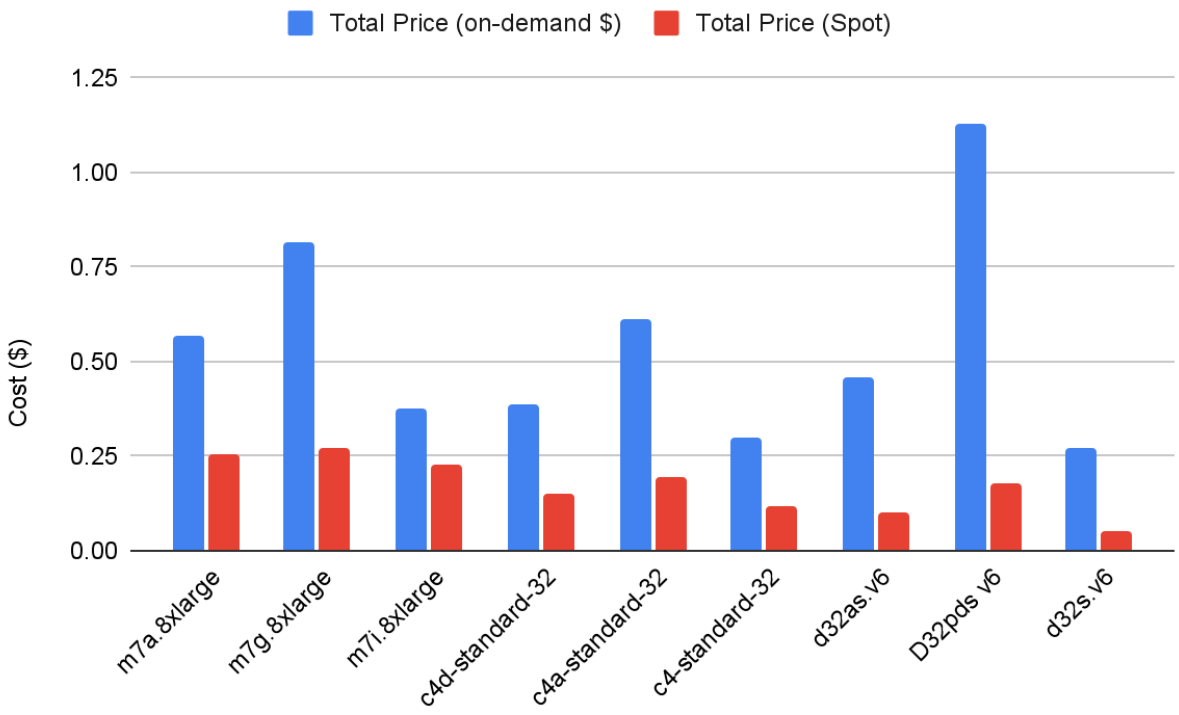
Google and Azure instances equivalent to AWS m7 and m8 are tested to compare with the performance and cost experienced in AWS tests. Google VMs were using the OS of Debian 12 and Azure VMs were using the OS of Ubuntu 24..04.

instance	CPU	70% mem. Peak (Gflops)	Time took for test (secs)	Total Price (on-demand $)	Total Price (Spot)
c4d-standard-32	Amd Epyc Turin	950	909	0.38	0.15
c4a-standard-32	Arm Axion	564	1531	0.61	0.19
c4-standard-3	Intel Granite	1281	674	0.30	0.12

2	Rapids				
d32as.v6	Amd Genoa Epyc 9004	848	1018	0.46	0.10
D32pds v6	Azure Cobalt 100	370	2337	1.13	0.18
d32s.v6	Intel Platinum 8473c	1298	665	0.27	0.05



The performance pattern on these instances on different Cloud services looks similar: Intel instances look superior performance to AMD instances. ARM instance show lower than half of the other systems.



Azure price looks lowest to run the same benchmark problem size. Google VMs also look lower pricing on both ondemand and spot instance use.

CONCLUSION

We observe the followings from the results:

- It is not economical to use larger instances (those with more vCPUs) because the price increases non-linearly relative to the vCPU count on the instance.
- The newer generation of CPU instances offers better price-performance due to significant performance improvements that outweigh the modest price increase.
- For instance types with the same vCPU count, memory-optimized instance can deliver better price-performance when handling larger problem sizes (it shows a smaller price increase relative to the substantial increase in the supported problem size).
- ARM processors appear to offer lower price-performance compared to systems using AMD and Intel x86 architectures consistently. This observation is shown across every cluster service tested: AWS, Google Cloud, and Azure.
- Spot pricing varies significantly among the different cloud service vendors.

APPENDIX

Scripts used in this study

AWS Userdata

```
#!/bin/bash
cd /home/ubuntu
# starting time stamp
date > myproc.txt
chmod a+rw myproc.txt
# access key setup
export AWS_ACCESS_KEY_ID="xxx"
export AWS_SECRET_ACCESS_KEY="xxx"
echo $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY > .access.key
chown ubuntu:ubuntu .access.key
chmod 600  .access.key
pwd >> myproc.txt
cat .access.key >> myproc.txt
# install base software set
sudo apt-get update -y
sudo apt-get install s3fs -y
sudo apt-get install -y gfortran gcc g++ make automake autoconf bzip2 cmake libtool-bin
sudo apt-get install -y patchelf python3-pip
# operation as the user
sudo -u ubuntu -i << 'EOF'
mkdir s3
# mount s3 storage
s3fs cb2bme1 s3 -o passwd_file=.access.key
echo "im in ubuntu" >> myproc.txt
pwd >> myproc.txt
ls s3 >> myproc.txt
# setup for spack
mkdir -p appker akrr_scratch
git clone https://github.com/nsimakov/akrr.git
cp -r akrr/akrr/appker_repo/inputs akrr/akrr/appker_repo/execs ~/appker
cd ~/appker/execs/
git clone -c feature.manyFiles=true https://github.com/spack/spack.git
```

```
source ~/appker/execs/spack/share/spack/setup-env.sh
# install and load openmpi and hpcc
spack install openmpi
spack load openmpi
spack install hpcc
spack load hpcc
cd /home/ubuntu
# set up and run hpcc
mkdir hpcc
cd hpcc
cp ~/s3/hpccinf.txt .
time mpirun -np 16 hpcc >> ../myproc.txt
# exit environment print
cp hpccoutf.txt  ~/s3/.
cd
date >> myproc.txt
cp myproc.txt s3/.
EOF
# shutdown the spot instance
sudo shutdown -h now
```

AWS EC2 Spot Instance Launch Command

```
aws ec2 request-spot-instances \
    --instance-count 1 \
    --launch-specification '{
        "ImageId": "ami-0360c520857e3138f",
        "InstanceType": "m7a.8xlarge",
        "KeyName": "cb2-dl1",
        "SecurityGroupIds": ["sg-07521ca76dfb83920"],
        "Placement": {"AvailabilityZone": "us-east-1a"},
        "UserData":
"IyEvYmluL2Jhc2gKY2QgL2hvbWUvdWJ1bnR1CmRhdGUgPiBteXByb2MudHh0CmNobW9kID
c3NyBteXByb2MudHh0CmV4cG9ydCBBV1NfQUNDRVNTX0tFWV9JRD0iQUtJQVNBMlZTSVN
LRUVPSFRKNUciCmV4cG9ydCBBV1NfU0VDUkVUX0FDQ0VTU19LRVk9IlF6N2V1VmZUOXc
2SFVYY3lQVXVocVJHcll…dHUKc3VkbyBzaHV0ZG93biAtaCBub3cK"
   }'
./price.x m7a.8xlarge > price.txt
```

AWS SPOT price extraction command

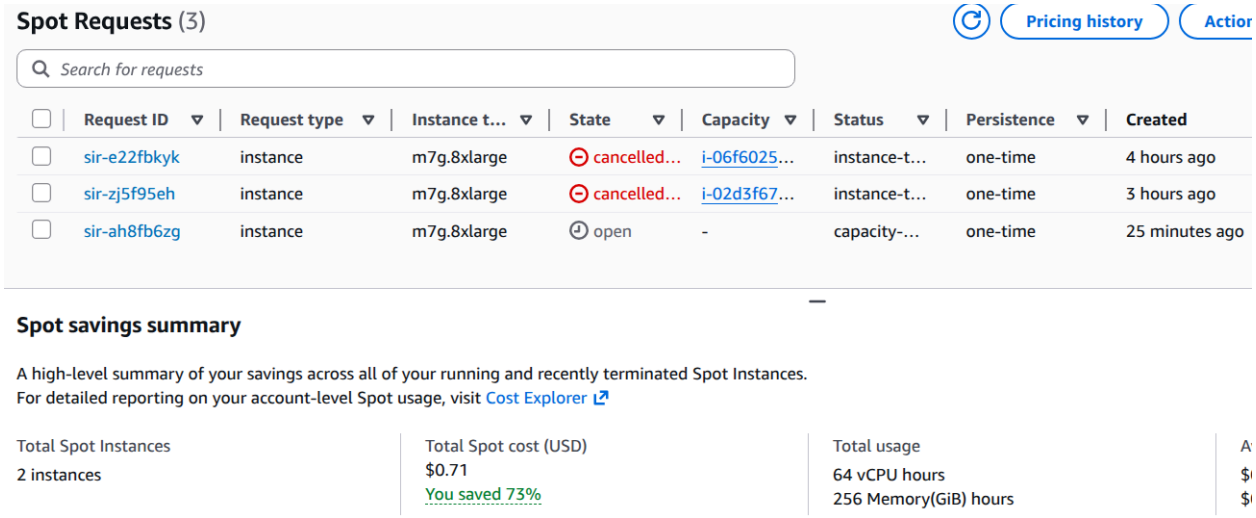
```
cdate=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
aws ec2 describe-spot-price-history --instance-types $1 --availabil
ity-zone us-east-1a --start-time $cdate
```

HPCC input data (hpccinf.txt)

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
4            # of problems sizes (N)
29 30 34 35  Ns
4            # of NBs
1 2 3 4      NBs
0            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
2 1 4        Ps
2 4 1        Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

SPOT Price shown at Instance Creation

AWS:



Spot Requests (3)

Pricing history | Actior

Search for requests

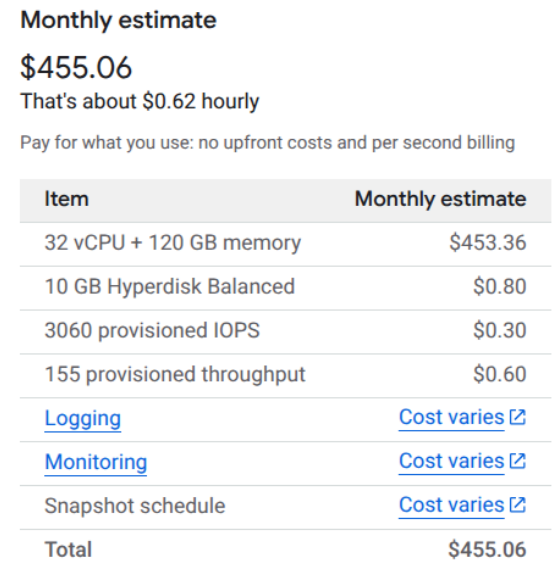
Request ID	Request type	Instance t...	State	Capacity	Status	Persistence	Created
sir-e22fbkyk	instance	m7g.8xlarge	cancelled...	i-06f6025...	instance-t...	one-time	4 hours ago
sir-zj5f95eh	instance	m7g.8xlarge	cancelled...	i-02d3f67...	instance-t...	one-time	3 hours ago
sir-ah8fb6zg	instance	m7g.8xlarge	open	-	capacity-...	one-time	25 minutes ago

Spot savings summary

A high-level summary of your savings across all of your running and recently terminated Spot Instances.
For detailed reporting on your account-level Spot usage, visit Cost Explorer

Total Spot Instances	Total Spot cost (USD)	Total usage	A
2 instances	$0.71 You saved 73%	64 vCPU hours 256 Memory(GiB) hours	$ $

GOOGLE:



Monthly estimate

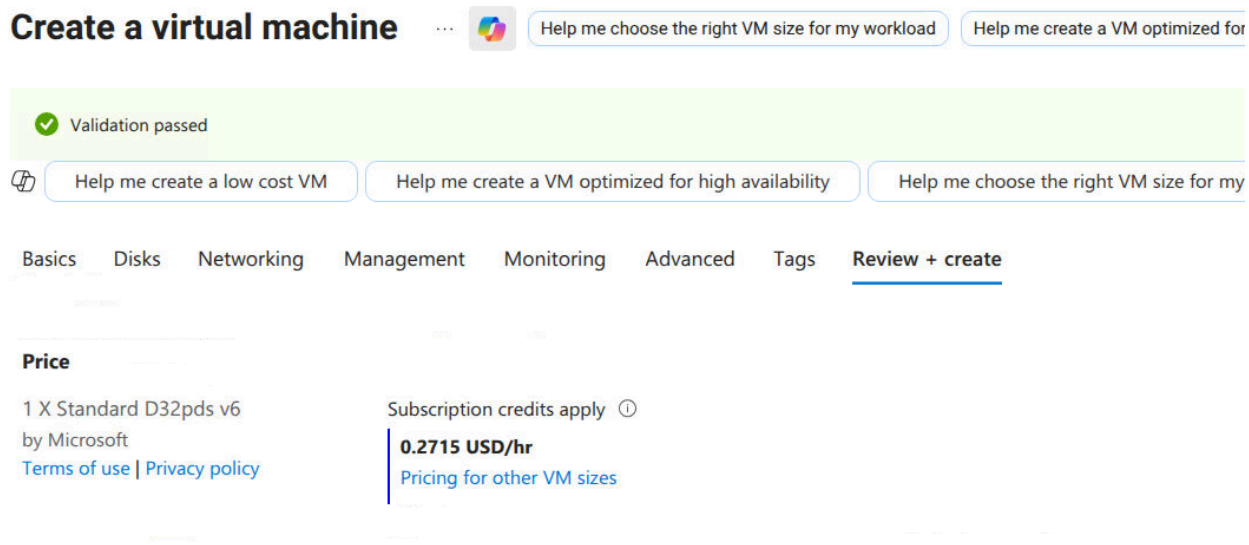
$455.06

That's about $0.62 hourly

Pay for what you use: no upfront costs and per second billing

Item	Monthly estimate
32 vCPU + 120 GB memory	$453.36
10 GB Hyperdisk Balanced	$0.80
3060 provisioned IOPS	$0.30
155 provisioned throughput	$0.60
Logging	Cost varies
Monitoring	Cost varies
Snapshot schedule	Cost varies
Total	$455.06

AZURE:



Reference

AWS Spot price: https://aws.amazon.com/ec2/spot/pricing/

GOOGLE Spot price: https://cloud.google.com/compute/vm-instance-pricing?authuser=1

AZUREspot price: https://azure.microsoft.com/en-us/pricing/spot-advisor/

Spot Instance: https://aws.amazon.com/blogs/compute/
best-practices-to-optimize-your-amazon-ec2-spot-instances-usage/

Cloudbank Checkpoint: https://drive.google.com/drive/u/0/folders/0ACq6oKlccbzQUk9PVA