
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

NFX304

Capacity plan optimally in the cloud

Joseph Lynch

Principal Software Engineer
Netflix - Data Platform

Prateek Sharma

Senior Solutions Architect
AWS

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Outline

Understanding hardware

- Computer shapes and their costs

How to capacity plan

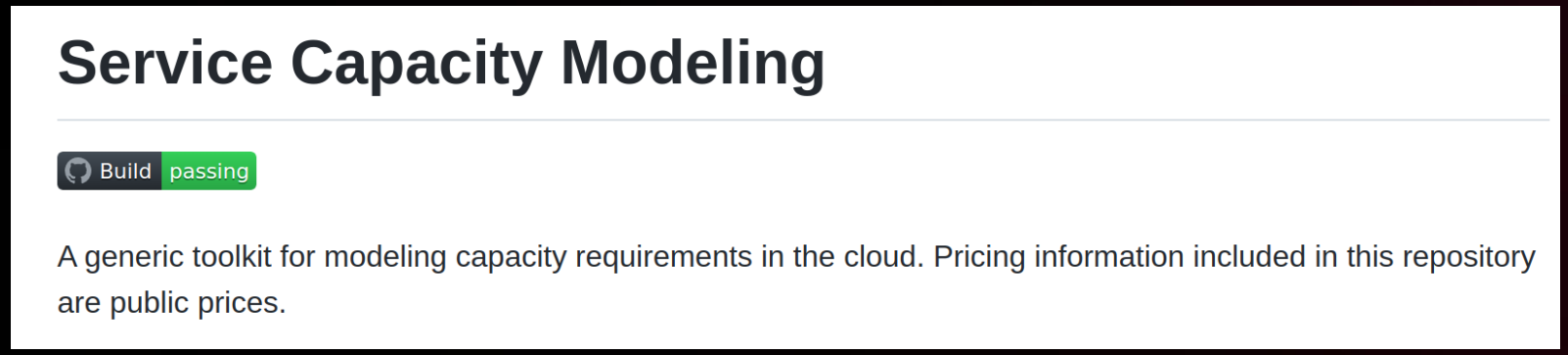
- Requirement domain-specific language (DSL)
- Queuing theory and capacity models

How to monitor your choices

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This system is open source



https://github.com/Netflix-Skunkworks/service-capacity-modeling

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

$$M(D, H, PL) \rightarrow C$$

M = Workload Capacity Model
D = User Desire
H = Hardware Profile
PL = Current Pricing and Lifecycle
C = Candidate Cluster

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Understanding hardware

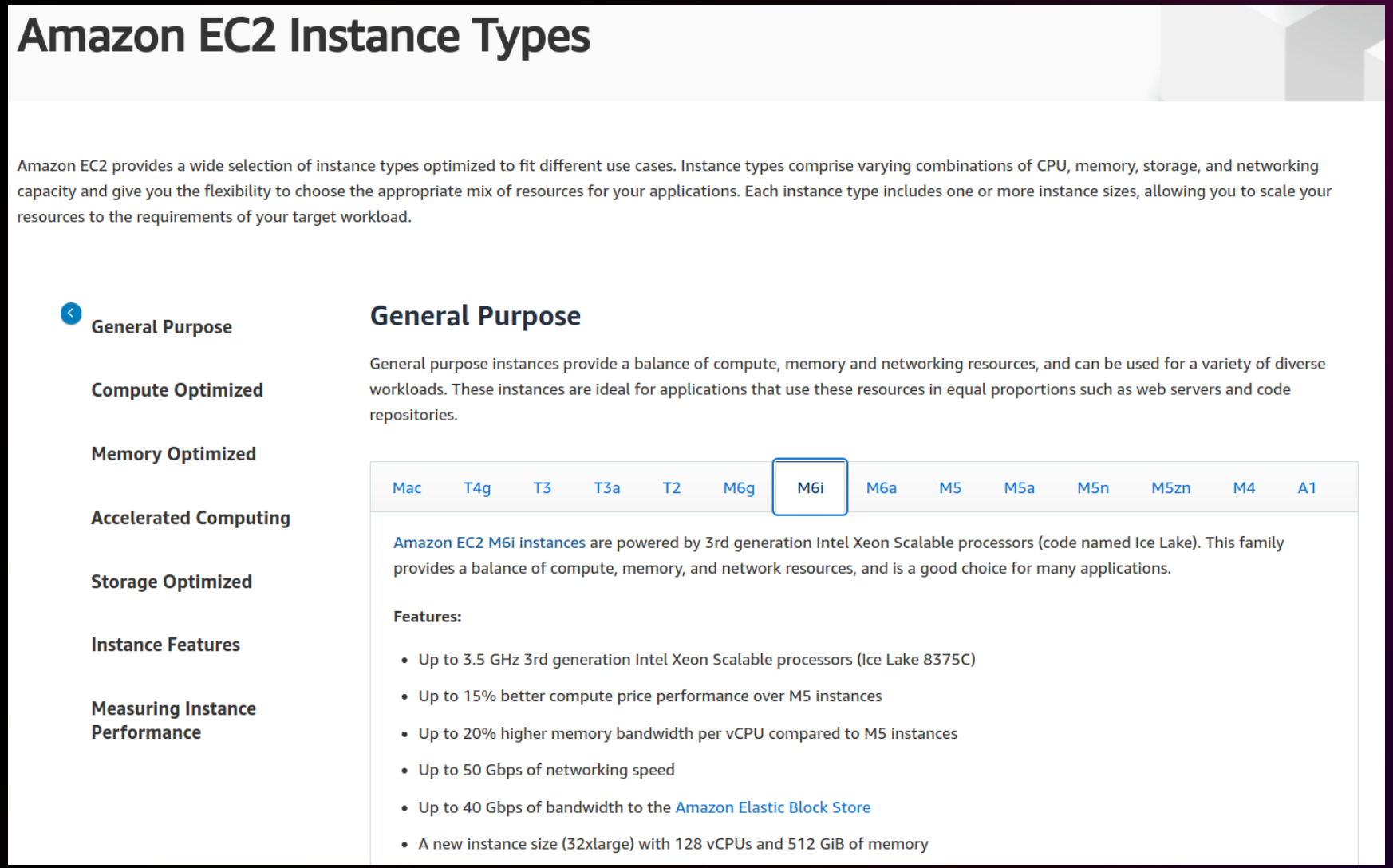
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

$$M(D, H, PL) \rightarrow C$$

M = Workload Capacity Model
D = User Desire
H = Hardware Profile
PL = Current Pricing and Lifecycle
C = Candidate Cluster

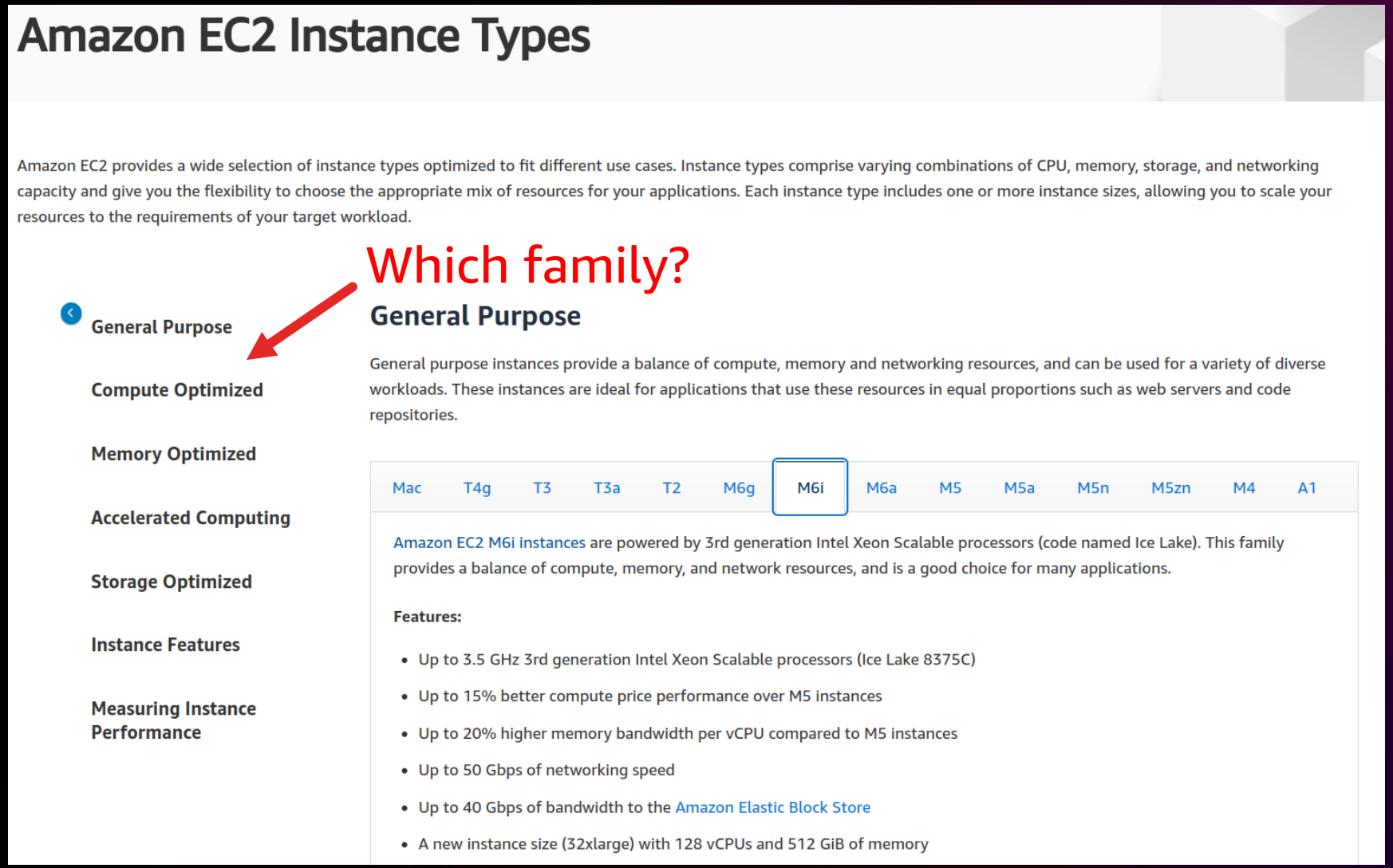
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has hundreds of Amazon EC2 instances



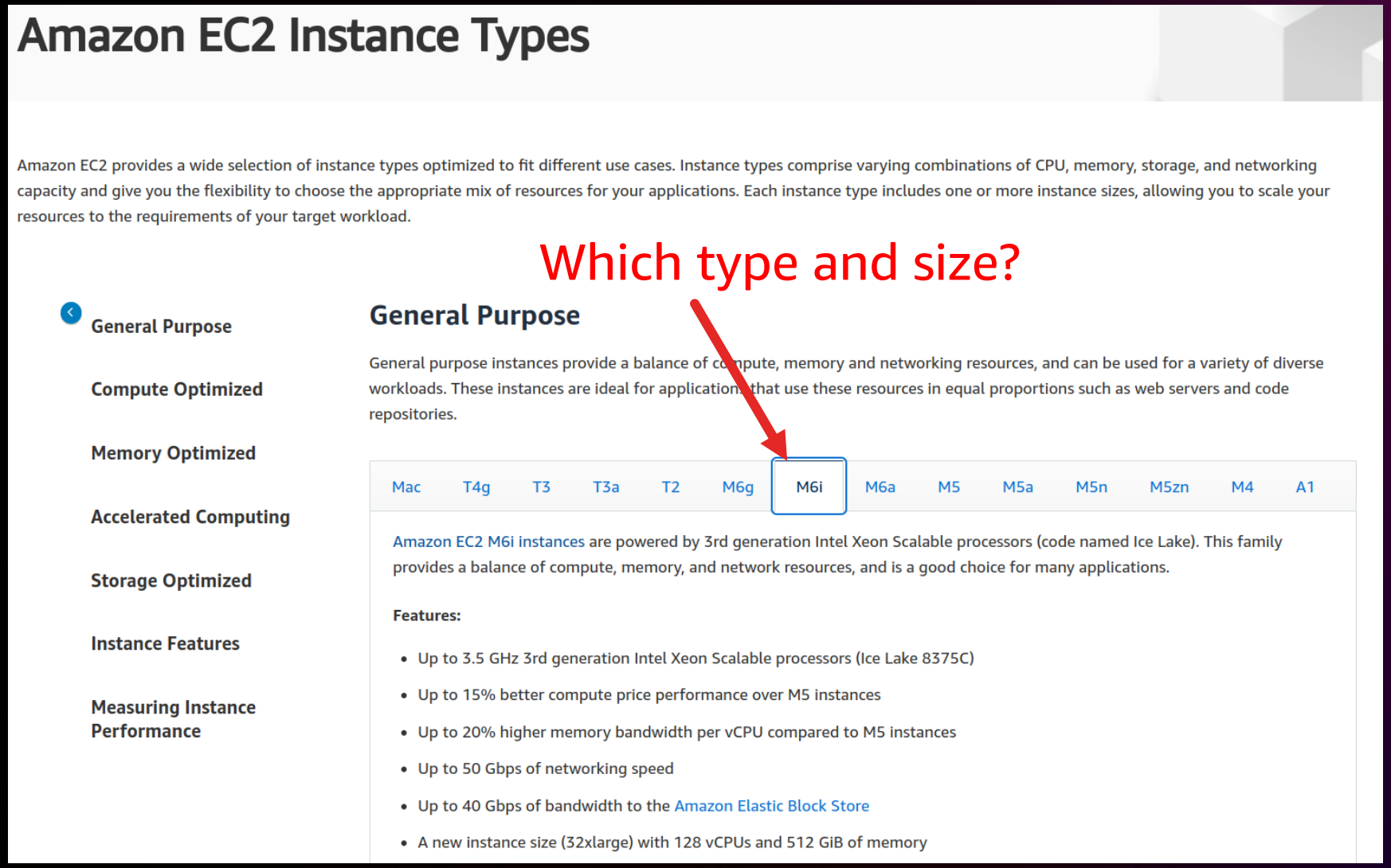
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has hundreds of Amazon EC2 instances



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has hundreds of Amazon EC2 instances



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has hundreds of Amazon EC2 instances

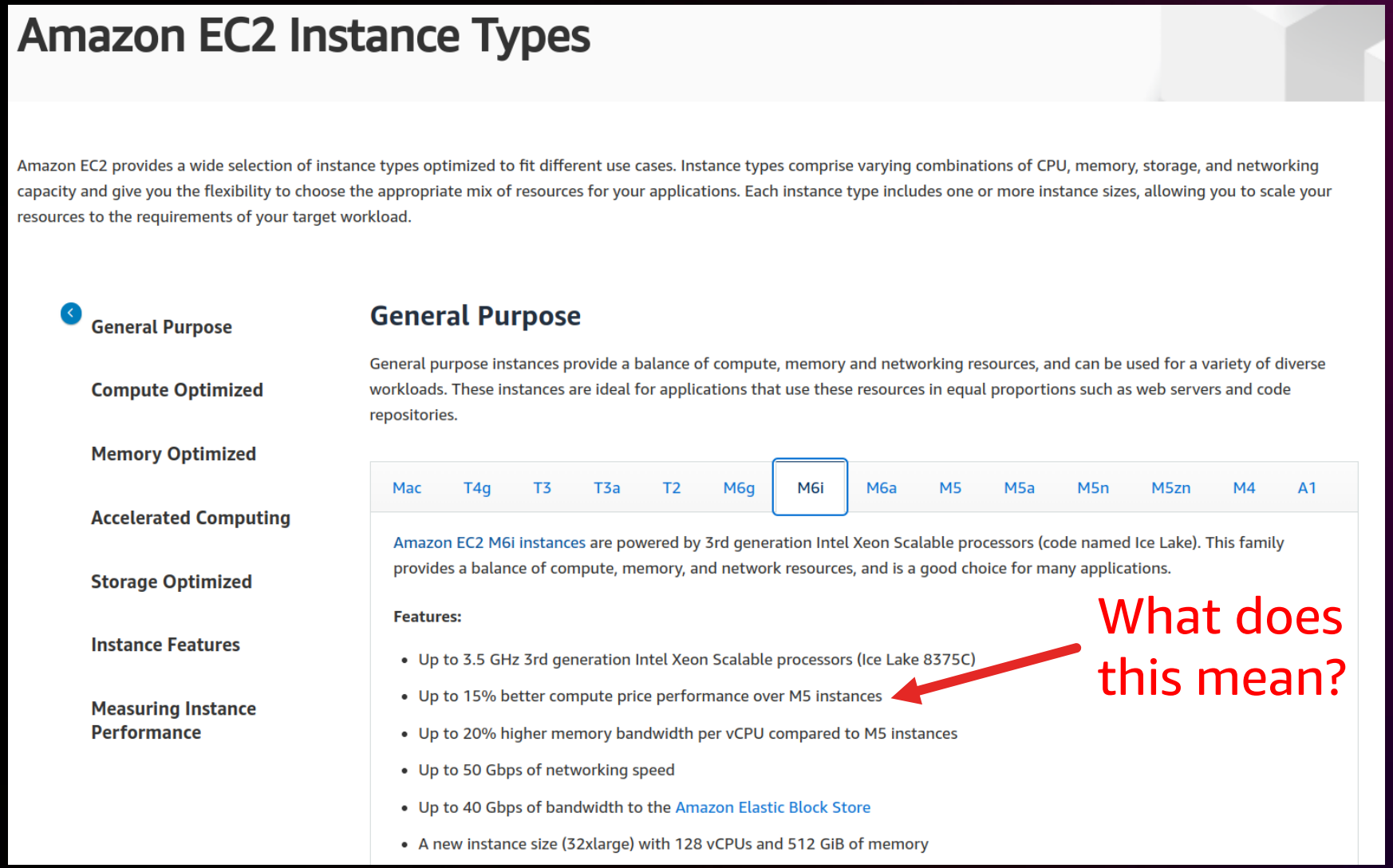
Amazon EC2 | Overview | Features | Pricing | Instance Types | FAQs | Getting Started | Resources

- General Purpose
- Compute Optimized
- Memory Optimized
- Accelerated Computing
- Storage Optimized
- Instance Features
- Measuring Instance Performance

Instance Size	vCPU	Memory (GiB)	Instance Storage (GB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Gbps)
m6i.large	2	8	EBS-Only	Up to 12.5	Up to 10
m6i.xlarge	4	16	EBS-Only	Up to 12.5	Up to 10
m6i.2xlarge	8	32	EBS-Only	Up to 12.5	Up to 10
m6i.4xlarge	16	64	EBS-Only	Up to 12.5	Up to 10
m6i.8xlarge	32	128	EBS-Only	12.5	10
m6i.12xlarge	48	192	EBS-Only	18.75	15
m6i.16xlarge	64	256	EBS-Only	25	20
m6i.24xlarge	96	384	EBS-Only	37.5	30
m6i.32xlarge	128	512	EBS-Only	50	40
m6i.metal	128	512	EBS-Only	50	40
m6id.large	2	8	1x118 NVMe SSD	Up to 12.5	Up to 10
m6id.xlarge	4	16	1x237 NVMe SSD	Up to 12.5	Up to 10
m6id.2xlarge	8	32	1x474 NVMe SSD	Up to 12.5	Up to 10
m6id.4xlarge	16	64	1x950 NVMe SSD	Up to 12.5	Up to 10
m6id.8xlarge	32	128	1x1900 NVMe SSD	12.5	10
m6id.12xlarge	48	192	2x1425 NVMe SSD	18.75	15
m6id.16xlarge	64	256	2x1900 NVMe SSD	25	20
m6id.24xlarge	96	384	4x1425 NVMe SSD	37.5	30
m6id.32xlarge	128	512	4x1900 NVMe SSD	50	40
m6id.metal	128	512	4x1900 NVMe SSD	50	40

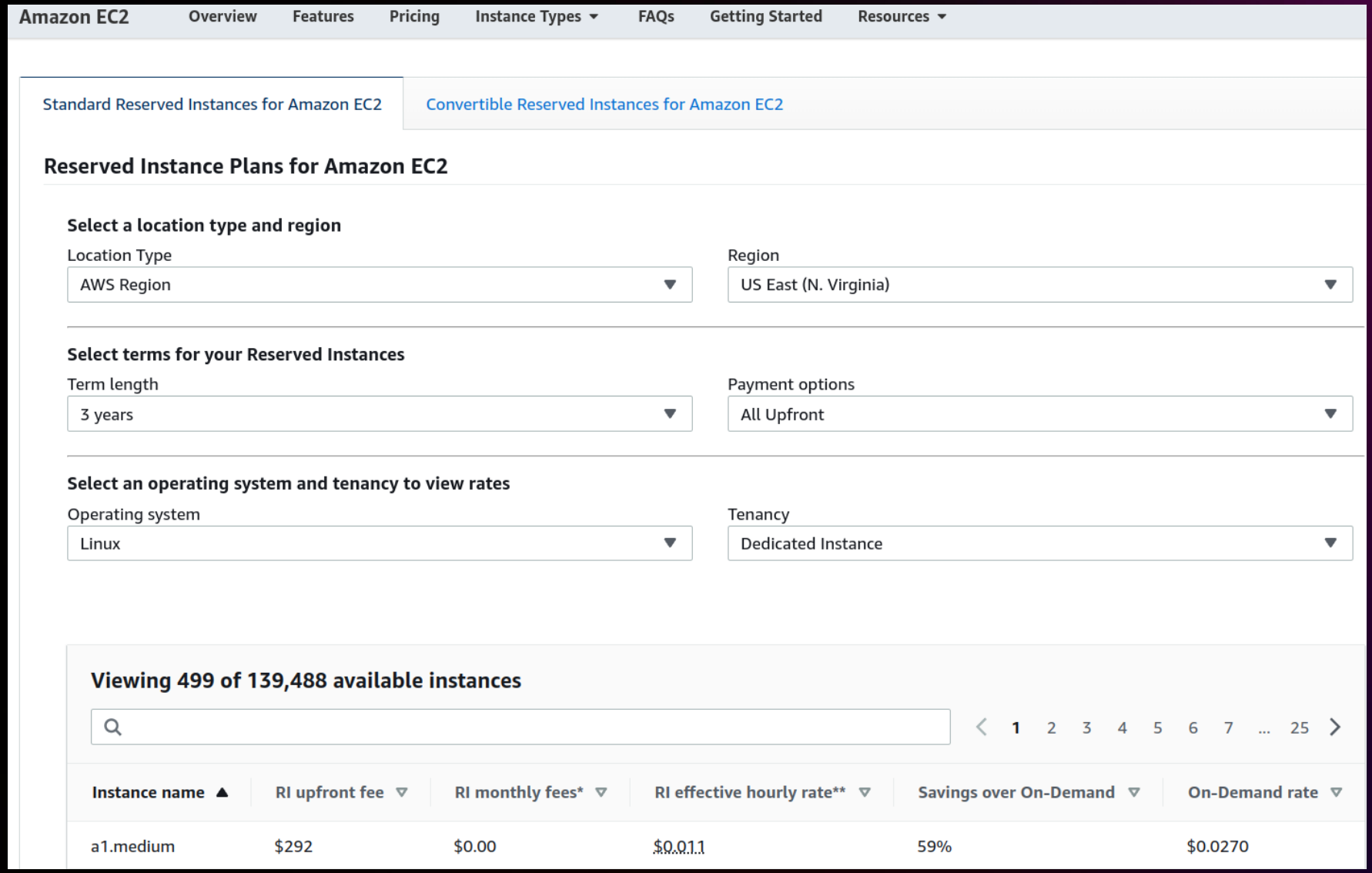
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has hundreds of Amazon EC2 instances



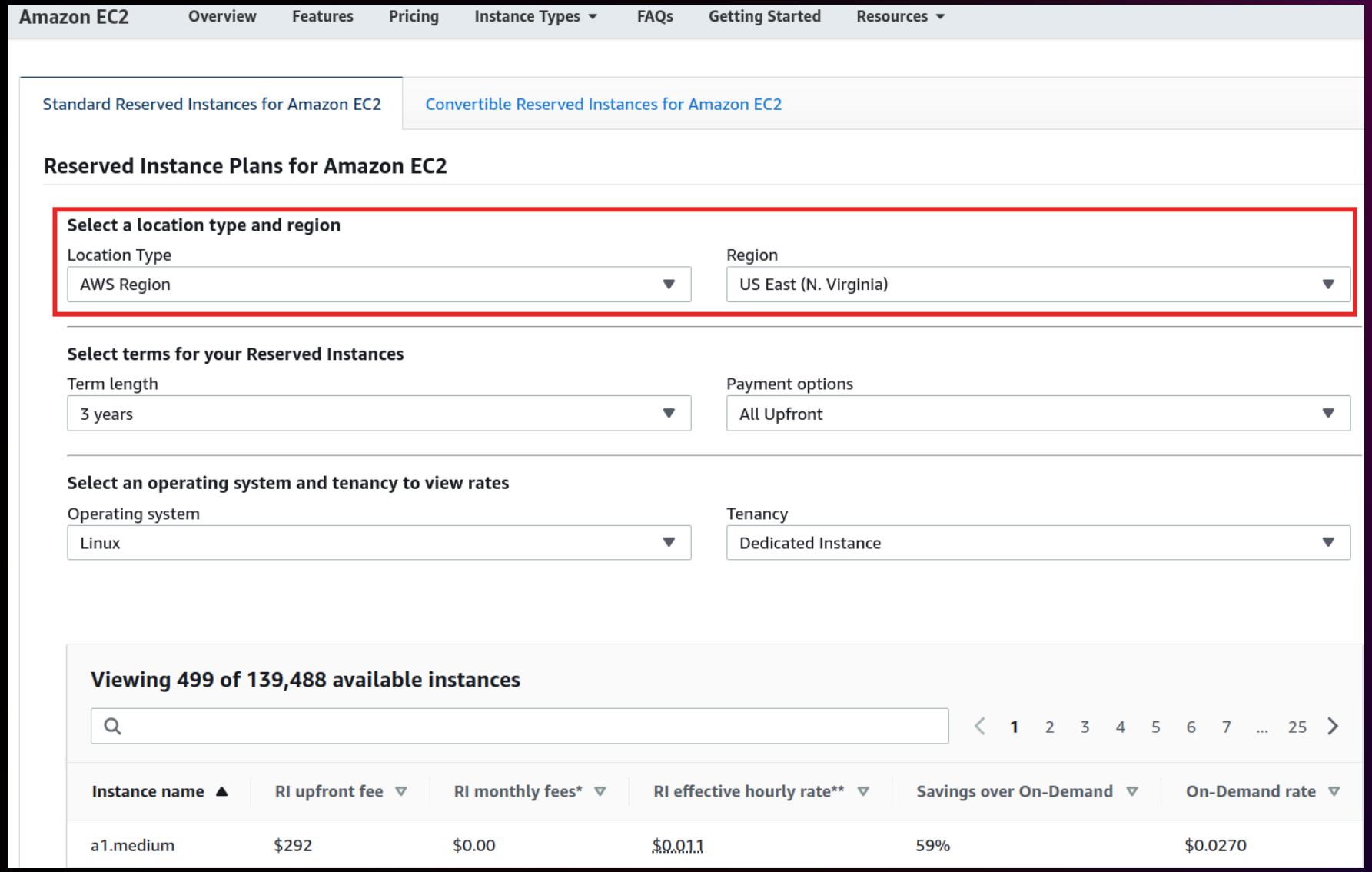
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Their cost is variable



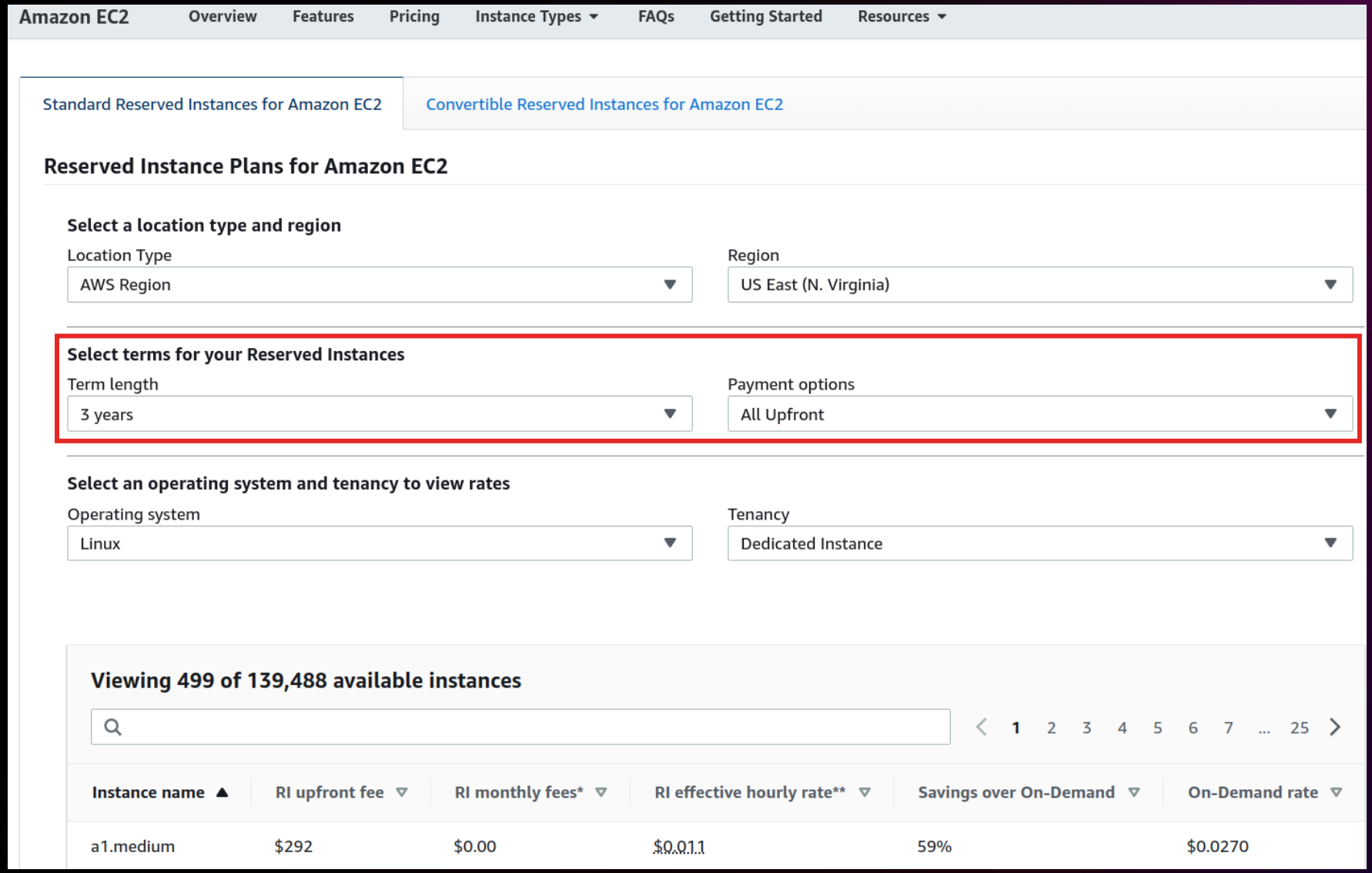
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Their cost is variable



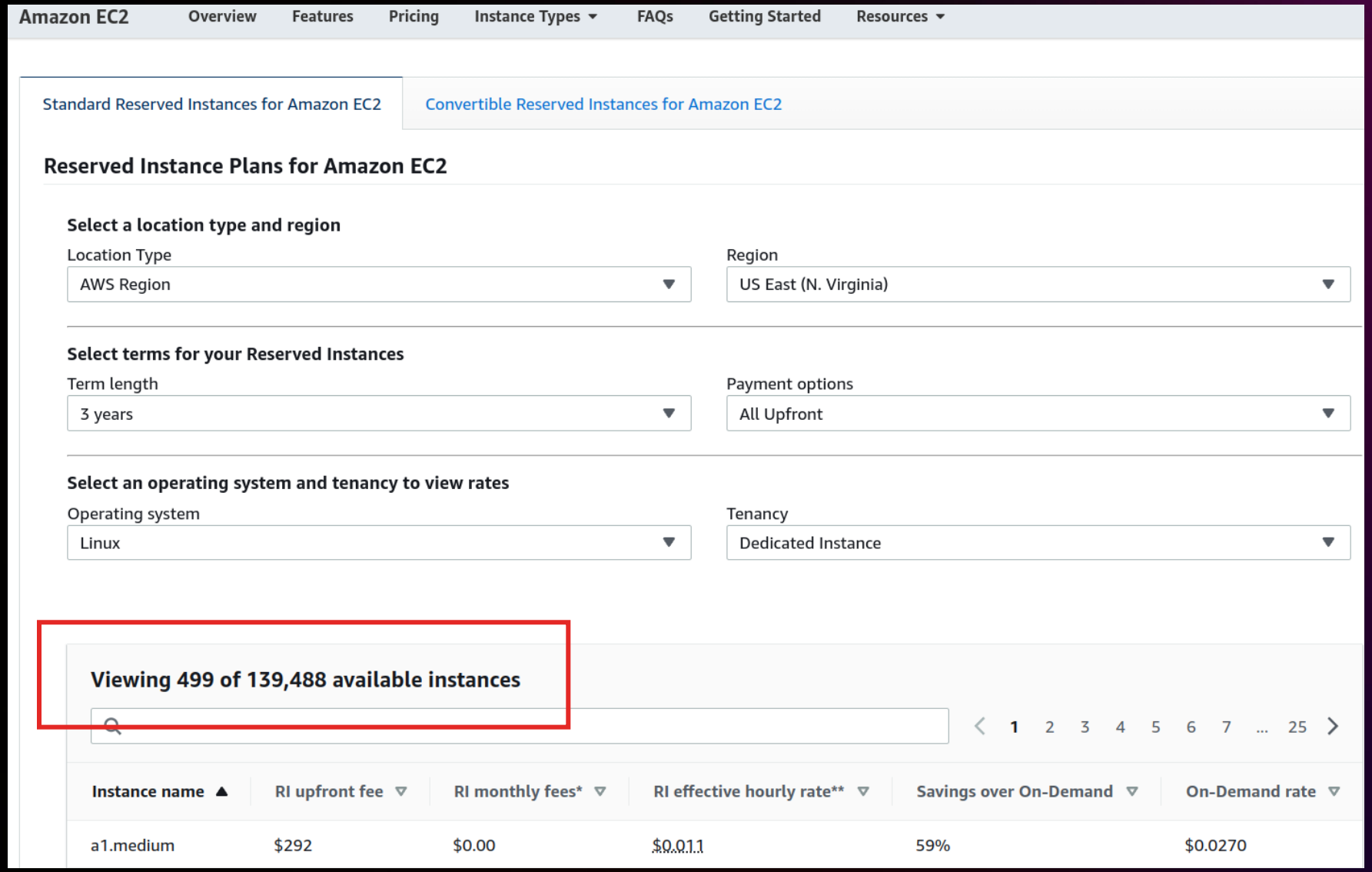
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Their cost is variable



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Their cost is variable



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hardware

- **Capacity**
  - How much CPU, RAM, network, disk space?
- **Latency**
  - How fast are the CPUs, NICs, and drives?
- **Lifecycle**
  - Alpha, beta, stable, deprecated, end of life?
- **Price**
  - How much do **you** pay for this hardware?

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hardware

- **Capacity**
  - How much CPU, RAM, network, disk space?
- **Latency**
  - How fast are the CPUs, NICs, and drives?

Measure these

- **Lifecycle**
  - Alpha, beta, stable, deprecated, end of life?
- **Price**
  - How much do **you** pay for this hardware?

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hardware

- **Capacity**
  - How much CPU, RAM, network, disk space?
- **Latency**
  - How fast are the CPUs, NICs, and drives?

Measure these

- **Lifecycle**
  - Alpha, beta, stable, deprecated, end of life?
- **Price**
  - How much do **you** pay for this hardware?

Depends on you

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Enumerate and measure shapes

https://bit.ly/3NJCipc

```
    "m5d.xlarge": {
      "name": "m5d.xlarge",
      "cpu": 2,
      "cpu_ghz": 3.1,
      "ram_gib": 15.57,
      "net_mbps": 1000,
...
    },
```



How do you measure these?

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Enumerate and measure shapes

https://bit.ly/3NJCipc

```
"m5d.large": {
  ...
  "drive": {
    "name": "ephem",
    "size_gib": 140,
    "read_io_latency_ms": {
       "min":0.07, "low":0.08, "mid":0.12, "high":0.20, "max":2,
       "confidence":0.9, "single_tenant": false},
    "read_io_per_s": 30000,
    "write_io_per_s": 15000,
    "block_size_kib": 4
  }
},
```



How do you measure these?

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Measure shapes

1. Launch instances
2. Gather properties (/proc/meminfo, /proc/cpuinfo, etc.)
3. Generate load (iperf, ndbench, netperf, fio)
4. Measure (bcc, bpftrace, metrics)

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Measure disk latency

https://bit.ly/3FSdidy

We can actually use our interval estimates to approximate all manner of Hardware properties such as:

- On CPU time of operations (e.g. some reads are slower than others)
- Request size distributions (e.g. some reads are bigger than others)
- Disk latency (e.g. some disks are faster or more consistent than others)

For a concrete example let's model a m5d drive which we model with an io latency distribution. Data for comparision comes from using `biosnoop` and `histogram.py` on a Cassandra server (the threads that are servicing reads are from the SharedPool).

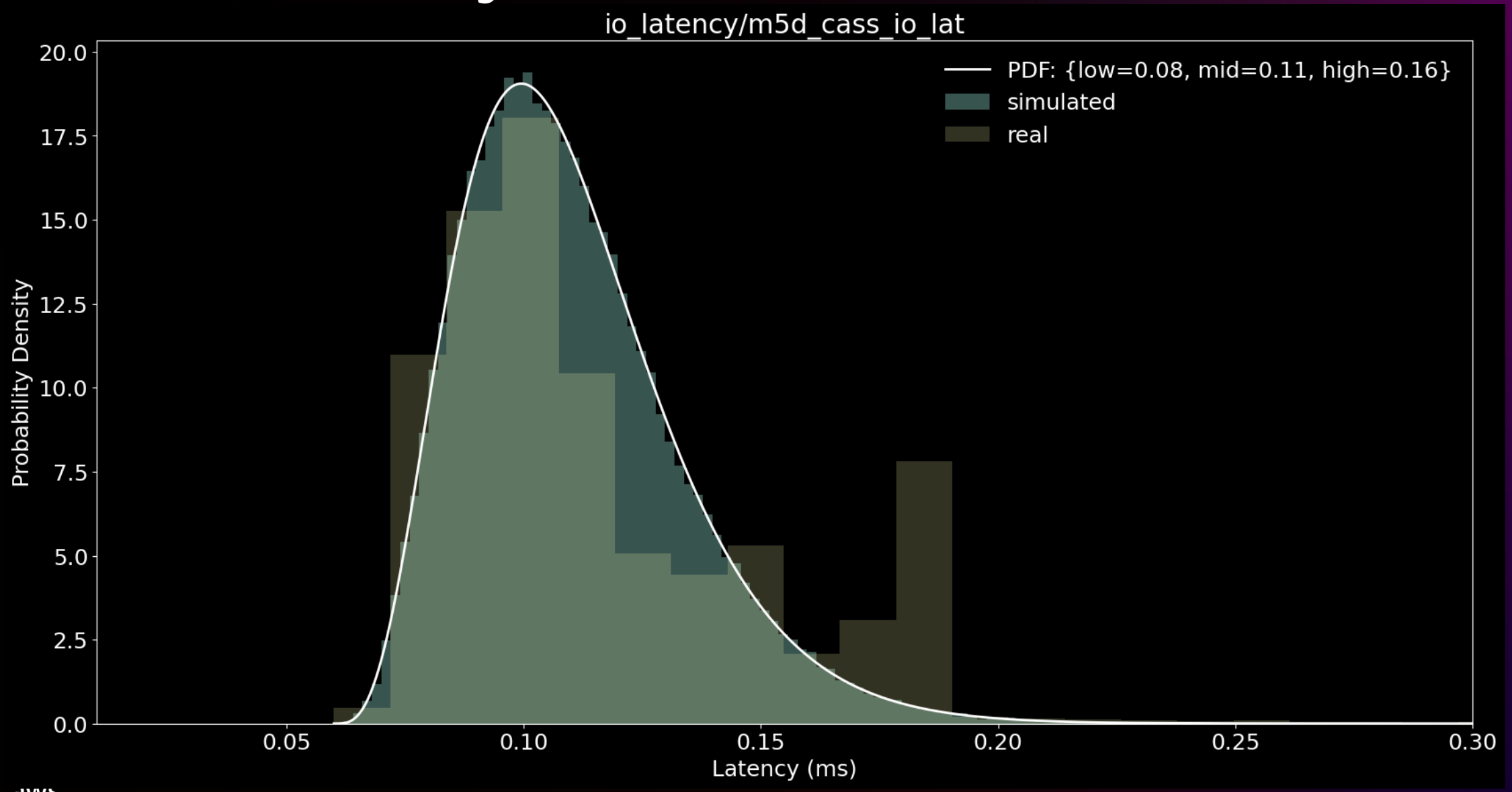
```
$ sudo /usr/share/bcc/tools/biosnoop > ios
$ grep SharedPool ios | tr -s ' ' | cut -f 8  -d ' ' > io_lat
$ cat io_lat | histogram.py -l -p
# NumSamples = 107517; Min = 0.06; Max = 2.43
# Mean = 0.118898; Variance = 0.002304; SD = 0.048005; Median 0.100000
# each ∎ represents a count of 569
    0.0600 -     0.0623 [    94]:  (0.09%)
    0.0623 -     0.0670 [     0]:  (0.00%)
    0.0670 -     0.0762 [   505]:  (0.47%)
    0.0762 -     0.0948 [ 33459]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (31.12%)
    0.0948 -     0.1318 [ 42706]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (39.72%)
    0.1318 -     0.2060 [ 29154]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (27.12%)
    0.2060 -     0.3542 [   994]: ∎ (0.92%)
    0.3542 -     0.6508 [   523]:  (0.49%)
    0.6508 -     1.2438 [    77]:  (0.07%)
    1.2438 -     2.4300 [     5]:  (0.00%)
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Fit disk latency

https://bit.ly/3FSdidy

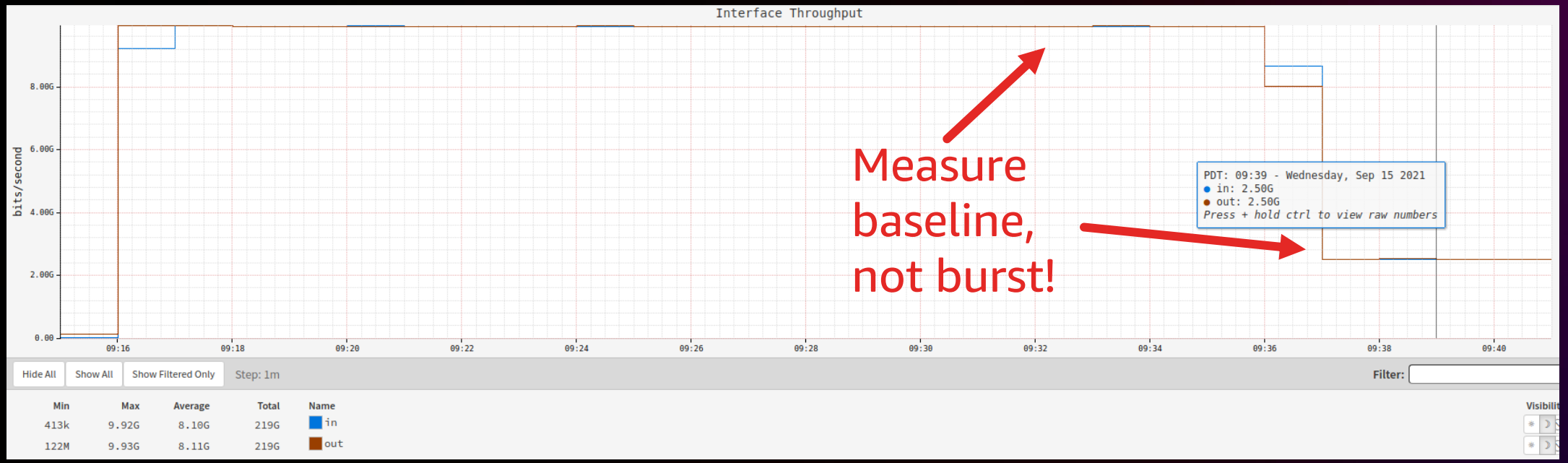


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Measure network

```
$ iperf3 -s -p 8888
-----------------------
Server listening on 8888
-----------------------
```

```
$ iperf3 -c 100.66.192.126 -P $(getconf _NPROCESSORS_ONLN) -p 8888 -t 3600
Connecting to host 100.66.192.126, port 8888
...
```



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hardware lifecycle at Netflix

- **Alpha:** Hardware preview (r6id)
- **Beta:** Production testing (m6i)
- **Stable:** Use in production (m5)
- **Deprecated:** Stop using (i3 → i3en)
- **End of life:** Do not use (m3, i2, . . .)



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Central lifecycle and pricing

```
curl https://<url>/pricing?regions=us-east-1,instance_types=m5.xlarge
```

```
{
 "m5.xlarge": [
   {
     "lifecycle": "stable",
     "ondemand_rate": X,
     "os": "linux",
     "rate": Y,
     "region": "us-east-1",
     "reserved_rate": Z
   }
 ]
}
```

`service-capacity-modeling` supports layering your pricing on top of shapes – https://bit.ly/3DOyfTU

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User’s input language

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

$$M(D, H, PL) \rightarrow C$$

M = Workload Capacity Model
D = User Desire
H = Hardware Profile
PL = Current Pricing and Lifecycle
C = Candidate Cluster

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User input

The user probably knows how much CPU, RAM, network, disk space, and disk IO they need

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User input

The user probably knows how much CPU, RAM, network, disk space, and disk IO they need

They probably don’t

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User input

The user probably knows how much CPU, RAM, network, disk space, and disk IO they need

They probably don’t

Okay, well, they must know how much traffic they will send and how big their requests or data will be

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User input

The user probably knows how much CPU, RAM, network, disk space, and disk IO they need

They probably don’t

Okay, well, they must know how much traffic they will send and how big their requests or data will be

They probably don’t

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

User input

The user probably knows how much CPU, RAM, network, disk space, and disk IO they need

They probably don't

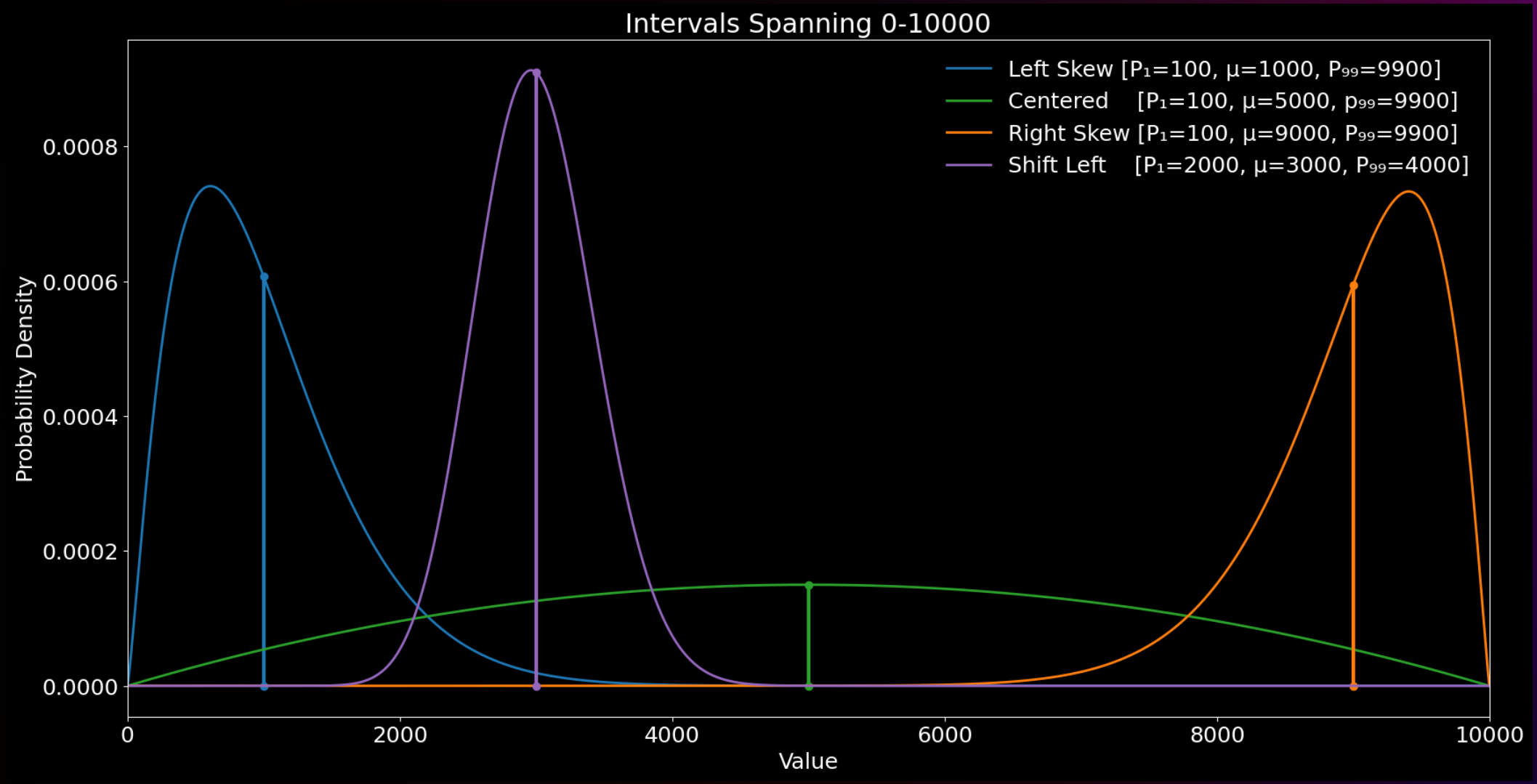
Okay, well, they must know how much traffic they will send and how big their requests or data will be

They probably don't . . . know *exactly*

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Use intervals



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity desires

https://bit.ly/3TbOuQB

```
class CapacityDesires(ExcludeUnsetModel):
    # How critical is this cluster, impacts how much "extra" we provision
    # 0 = Critical to the product           (Product does not function)
    # 1 = Important to product with fallback (User experience degraded)
    # 2 = Care about it but don't wake up    (Internal apps)
    # 3 = Do not care                        (Testing)
    service_tier: int = 1

    # How will the service be queried
    query_pattern: QueryPattern = QueryPattern()

    # What will the state look like
    data_shape: DataShape = DataShape()
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

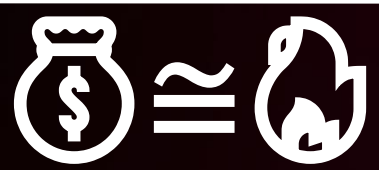
What is Service Tier?

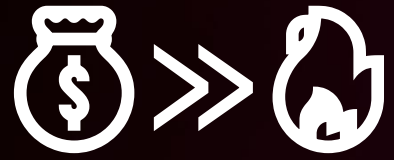
HOW SAD ARE YOU IF THIS CLUSTER FAILS!

Tier 0 = 

Tier 1 = 

Generally "Prod"

Tier 2 = 

Tier 3 = 

Generally "Test"

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Query pattern

https://bit.ly/3t9e6Db

```
class QueryPattern(ExcludeUnsetModel):
    # Will the service primarily be accessed in a latency sensitive mode
    # (aka we care about P99) or throughput (we care about averages)
    access_pattern: AccessPattern = AccessPattern.latency
    access_consistency: GlobalConsistency = GlobalConsistency()

    # A main input, how many requests per second will we handle
    # We assume this is the mean of a range of possible outcomes
    estimated_read_per_second: Interval = certain_int(0)
    estimated_write_per_second: Interval = certain_int(0)

    # A main input, how much on cpu time per operation do you take.
    # This depends heavily on workload, but this is a generally ok default
    # For a Java workload (C or C++ will be ~10x better, python ~2-4x slower, etc...)
    estimated_mean_read_latency_ms: Interval = certain_float(1)
    estimated_mean_write_latency_ms: Interval = certain_float(1)
    ...
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Stateful data shape

https://bit.ly/3fQMvDH

```
class DataShape(ExcludeUnsetModel):
    # How much state do we expect this service to have. Even stateless services
    # have logs or other temporary files that require space on volumes
    estimated_state_size_gib: Interval = certain_int(0)
    estimated_state_item_count: Optional[Interval] = None

    ...
    # How compressible is this dataset. Note that databases might offer
    # better or worse compression strategies that will impact this
    #   Note that the ratio here is the forward ratio, e.g.
    #   A ratio of 2 means 2:1 compression (0.5 on disk size)
    estimated_compression_ratio: Interval = certain_float(1)
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Stateful data shape

https://bit.ly/3fQMvDH

```
class DataShape(ExcludeUnsetModel):

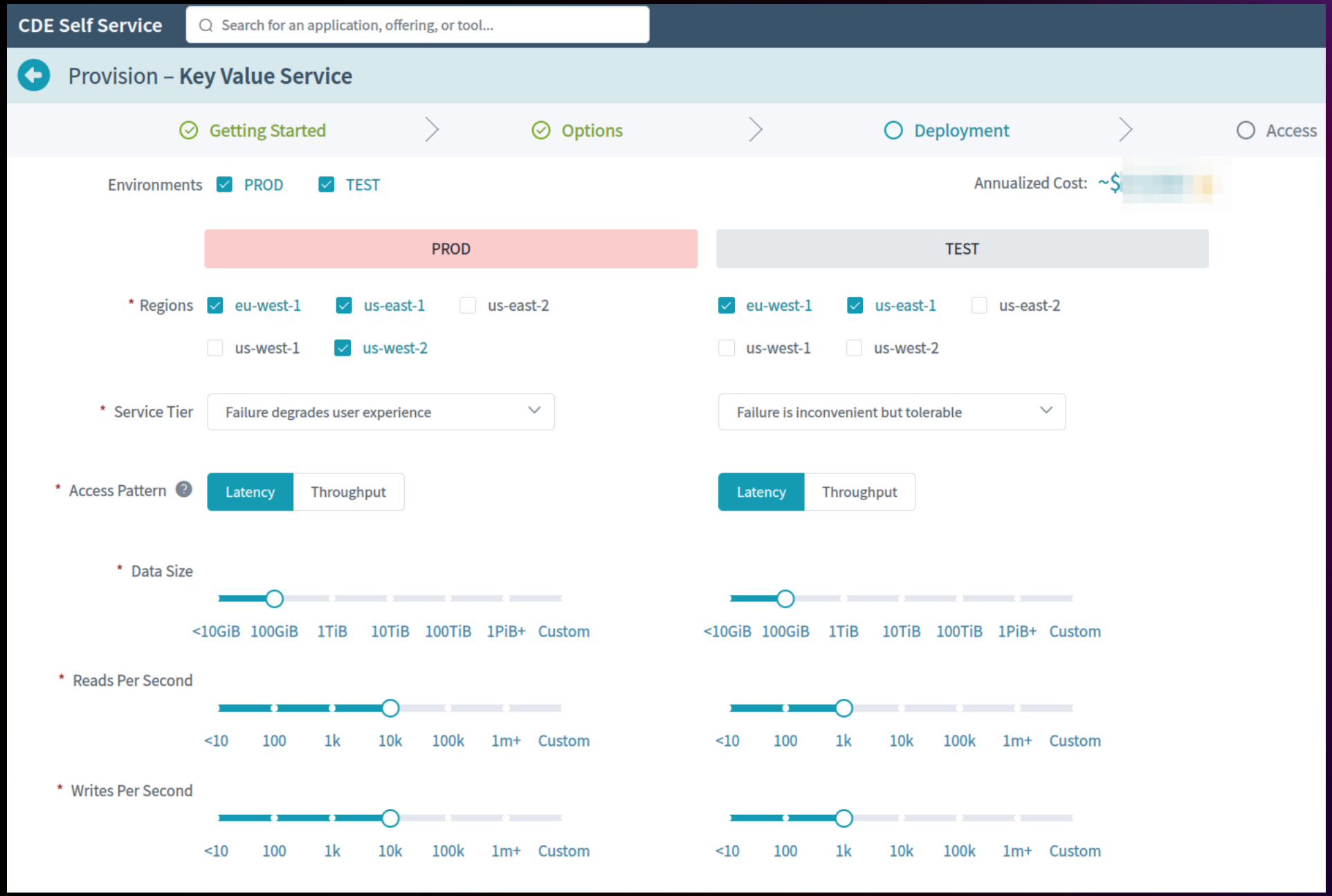
    ...
    # How much fixed memory must be provisioned per instance for the
    # application (e.g. for process heap memory)
    reserved_instance_app_mem_gib: int = 2

    # How much fixed memory must be provisioned per instance for the
    # system (e.g. for kernel and other system processes)
    reserved_instance_system_mem_gib: int = 1

    ...
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

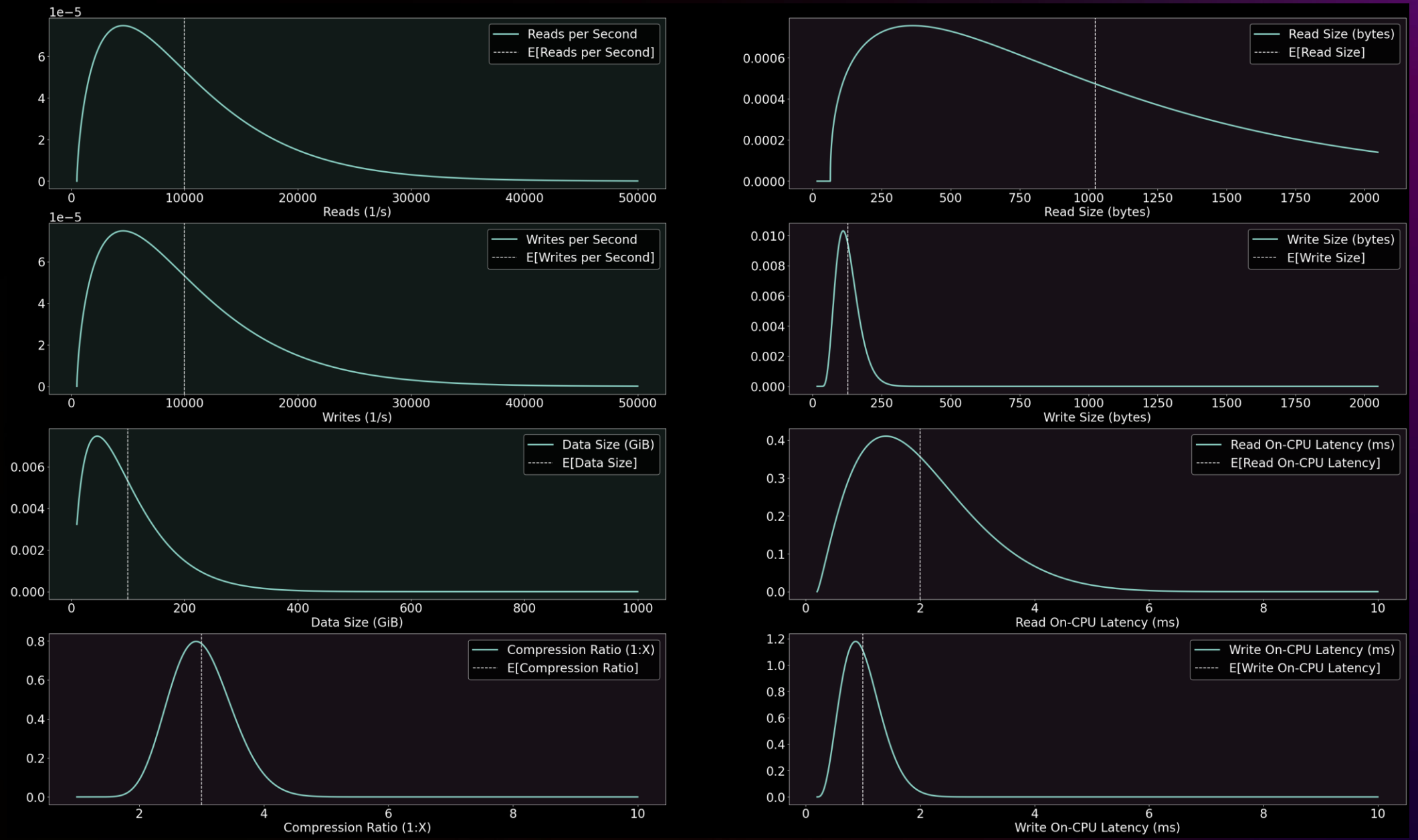
Example user input

https://bit.ly/3EboIYq

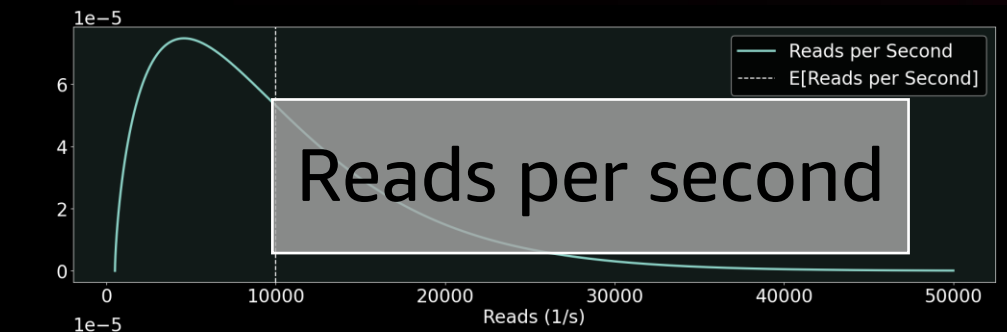
```
desires = CapacityDesires(
    service_tier=1,
    query_pattern=QueryPattern(
        # Not sure exactly how much QPS we will do, but we think ~10k / s
        estimated_read_per_second=Interval(
            low=1_000, mid=10_000, high=100_000, confidence=0.98
        ),
        estimated_write_per_second=Interval(
            low=1_000, mid=10_000, high=100_000, confidence=0.98
        ),
    ),
    # Not sure how much data, but we think around 100 GiB
    data_shape=DataShape(
        estimated_state_size_gib=Interval(
            low=10,    mid=100,    high=1_000,   confidence=0.98
        ),
    )...
```

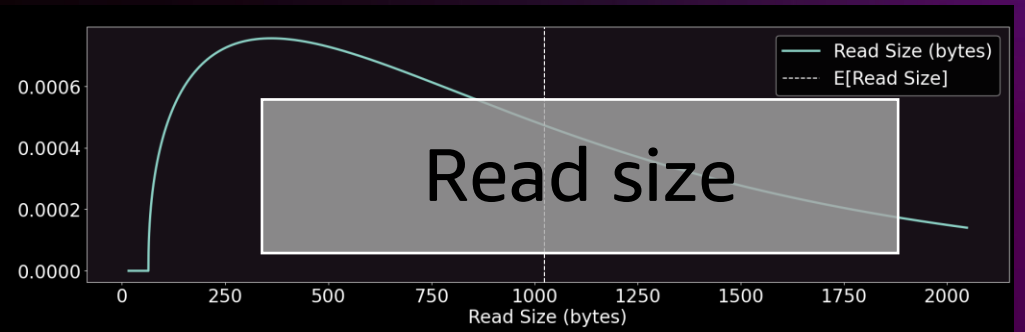
aws

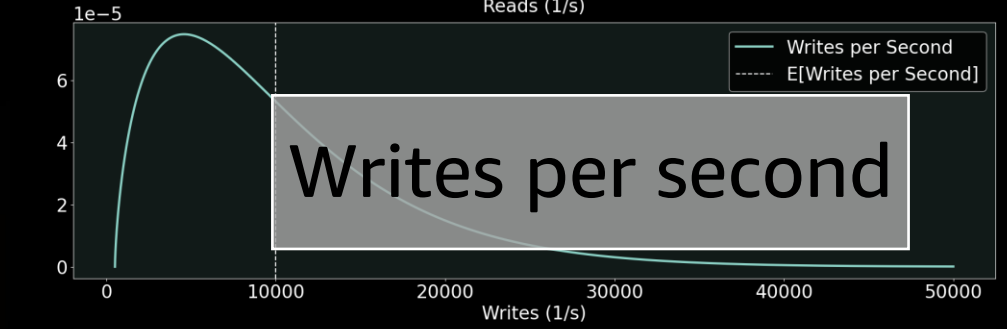
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

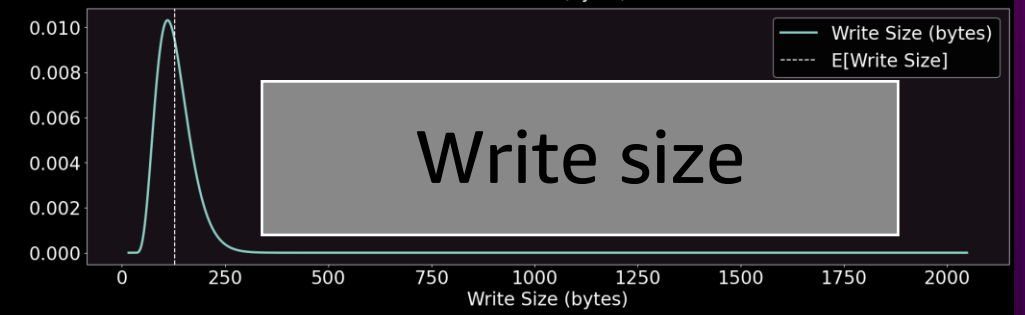


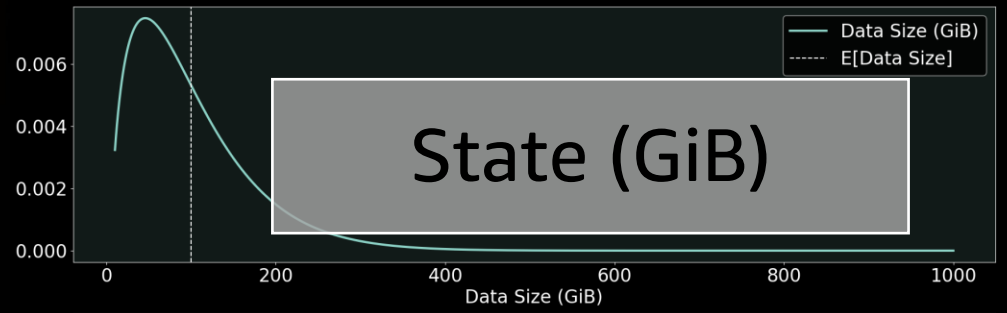
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

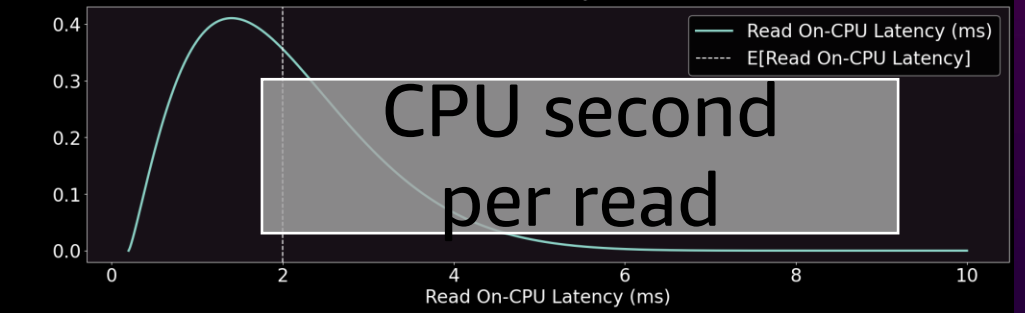


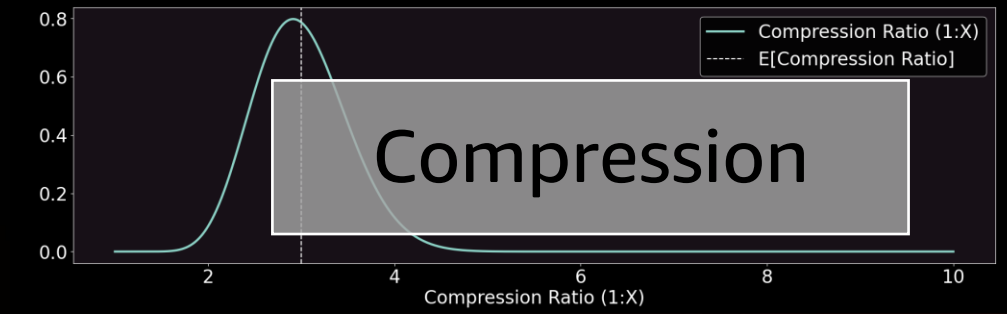


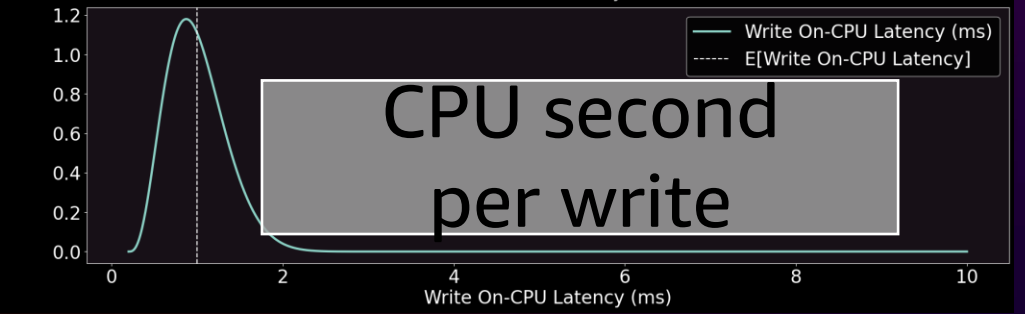




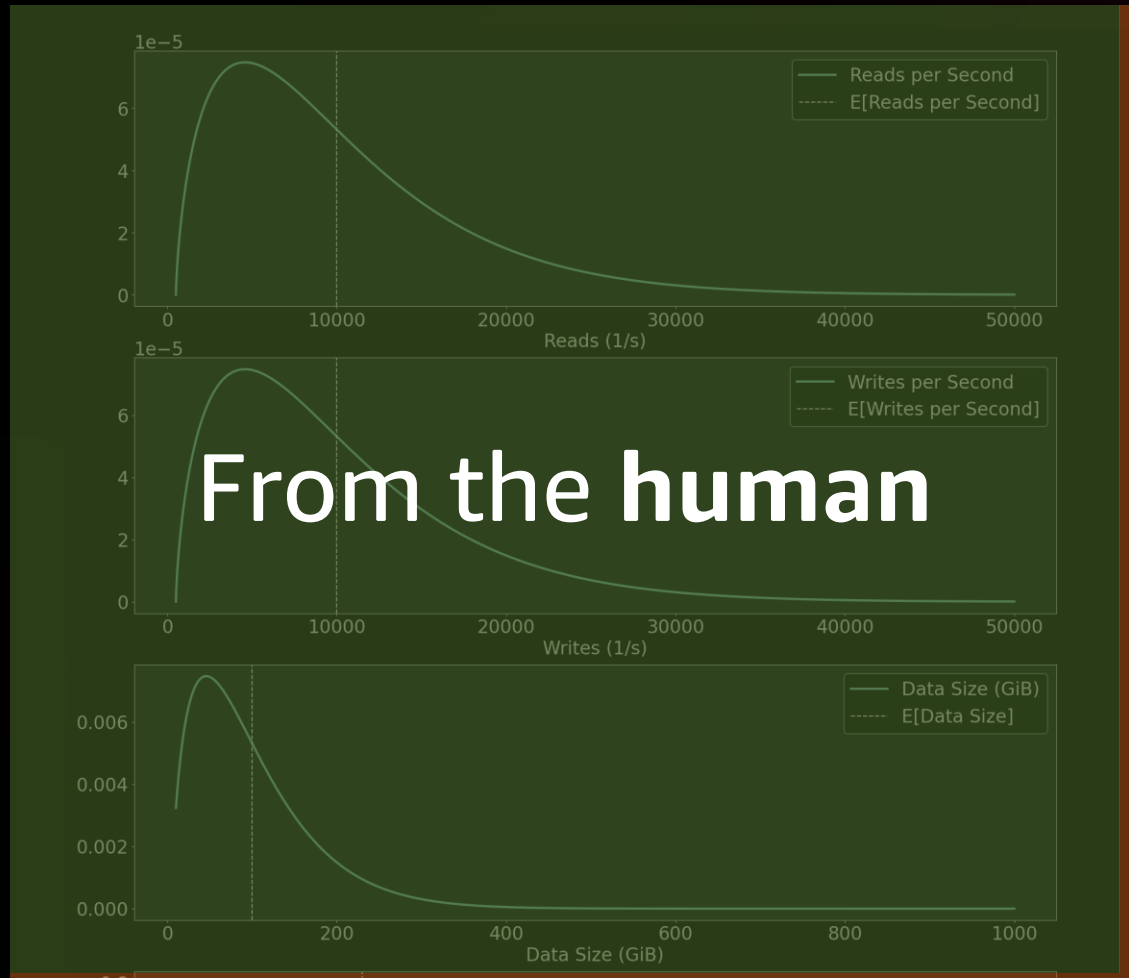




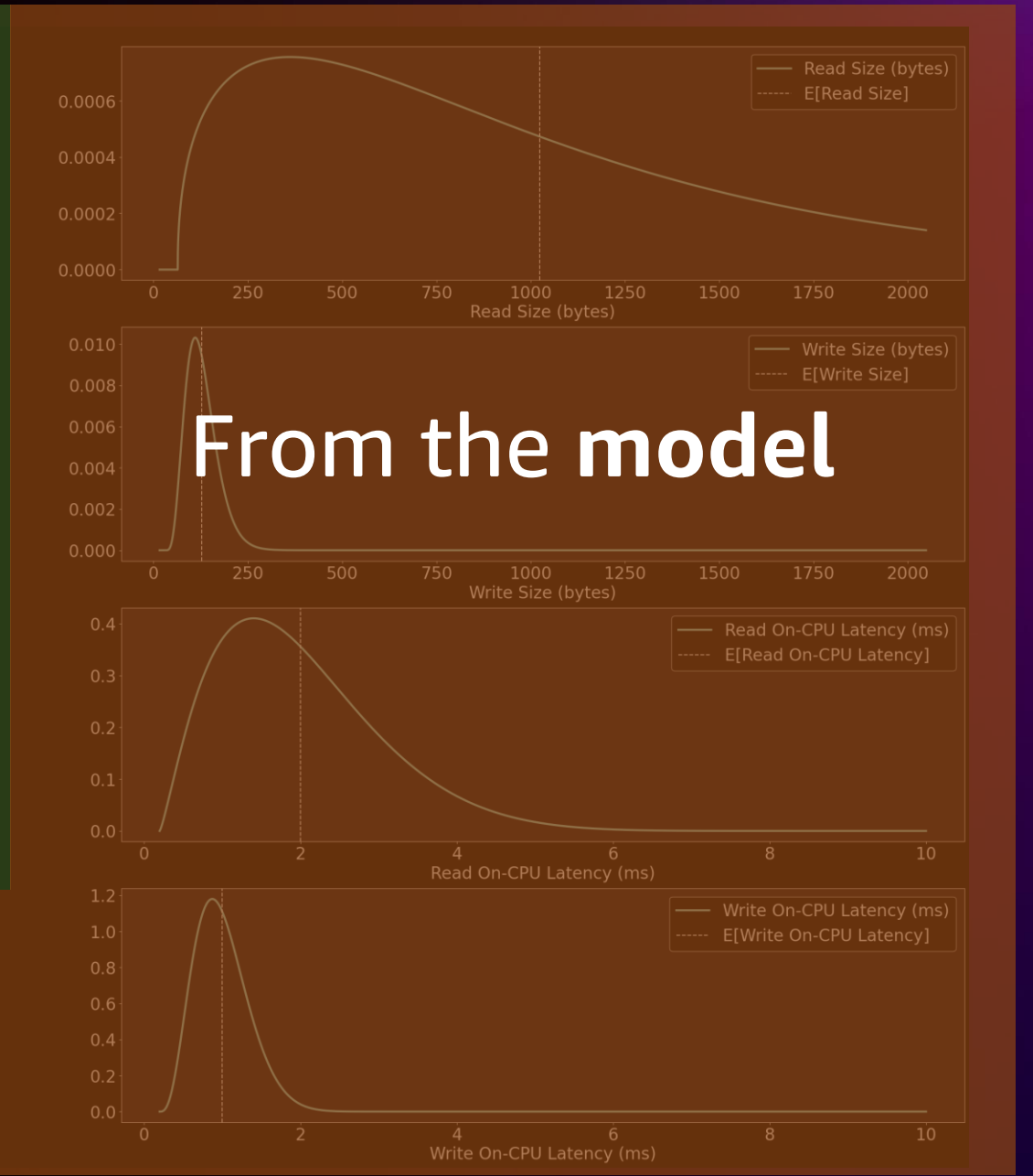




© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



From the **human**



From the **model**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning with certainty

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

$$M(D, H, PL) \rightarrow C$$

M = Workload Capacity Model
D = User Desire
H = Hardware Profile
PL = Current Pricing and Lifecycle
C = Candidate Cluster

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning 301

We have the **right inputs**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning 301

We have the **right inputs**

Now we need the **right math**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning 301

We have the **right inputs**

Now we need the **right math**

… but don’t worry, it is easy math!

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning 301

We have the **right inputs**

Now we need the **right math**

… but don’t worry, it is easy math!

… and we already did it for you!

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Build a stateful model

```
class NflxCassandraCapacityModel(CapacityModel):
    @staticmethod
    def capacity_plan(
        instance: Instance,
        drive: Drive,
        context: RegionContext,
        desires: CapacityDesires,
        extra_model_arguments: Dict[str, Any],
    ) -> Optional[CapacityPlan]:
```

https://bit.ly/3FSjfHh

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Build a stateless model

```
class NflxJavaAppCapacityModel(CapacityModel):
    @staticmethod
    def capacity_plan(
        instance: Instance,
        drive: Drive,
        context: RegionContext,
        desires: CapacityDesires,
        extra_model_arguments: Dict[str, Any],
    ) -> Optional[CapacityPlan]:
```

https://bit.ly/3UyTZKf

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Let’s implement Cassandra

Cassandra is our most complicated model at Netflix

Databases are the most complex software you have to capacity plan for; stateless apps are easy

Let’s see how to build the complex case!

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CPU

https://bit.ly/3fLC9Fi

$$\text{let } \mu = \text{average } \frac{\text{CPU time}}{\text{request}}$$

$$\text{let } \lambda = \text{average } \frac{\text{request}}{\text{second}}$$

$$R = \lambda \times \mu$$

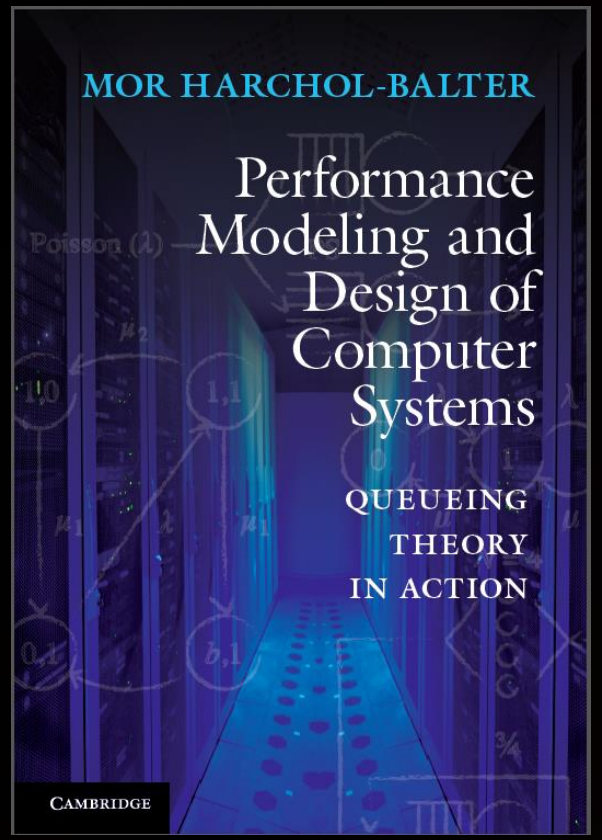
$$\text{CPUs} = R + Q * \sqrt{R}$$

Service Tier	P(Queue)	Q
0	1%	2.375
1	5%	1.761
2	20%	1.16
3	30%	1

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CPU – Determine QoS



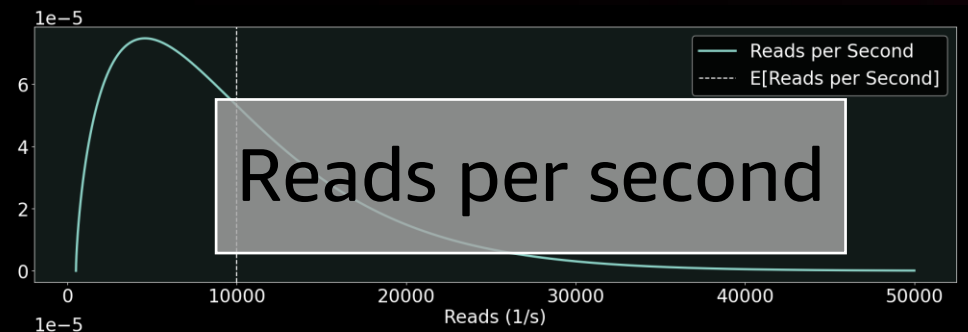
Assume a M/M/k system

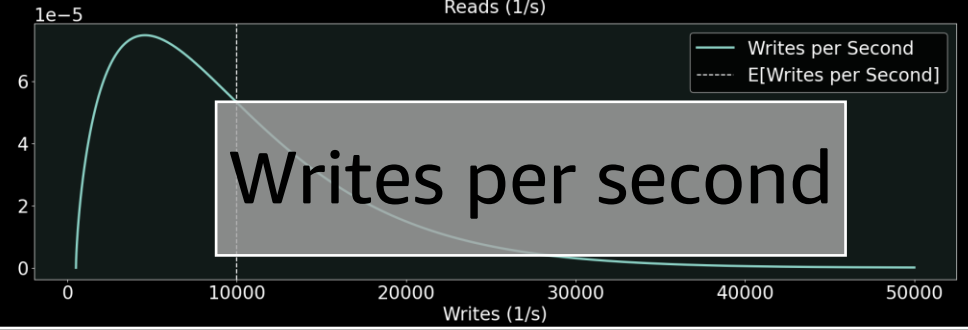
Bound the probability of queueing by setting a bound on the mean response time

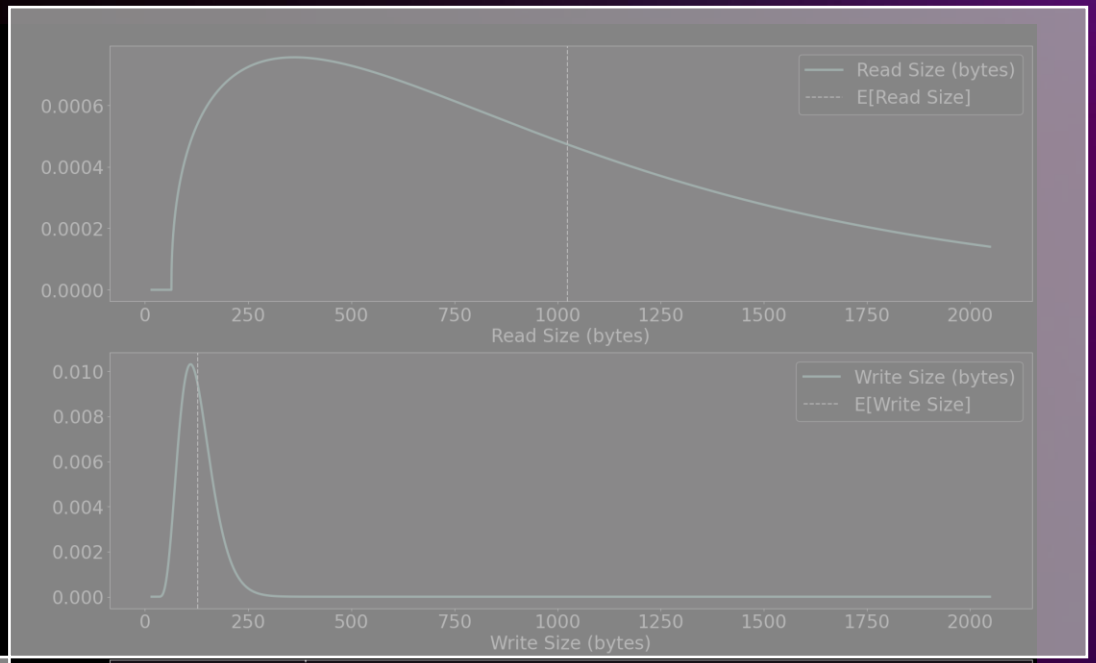
Solve the value Q under the normal assumption

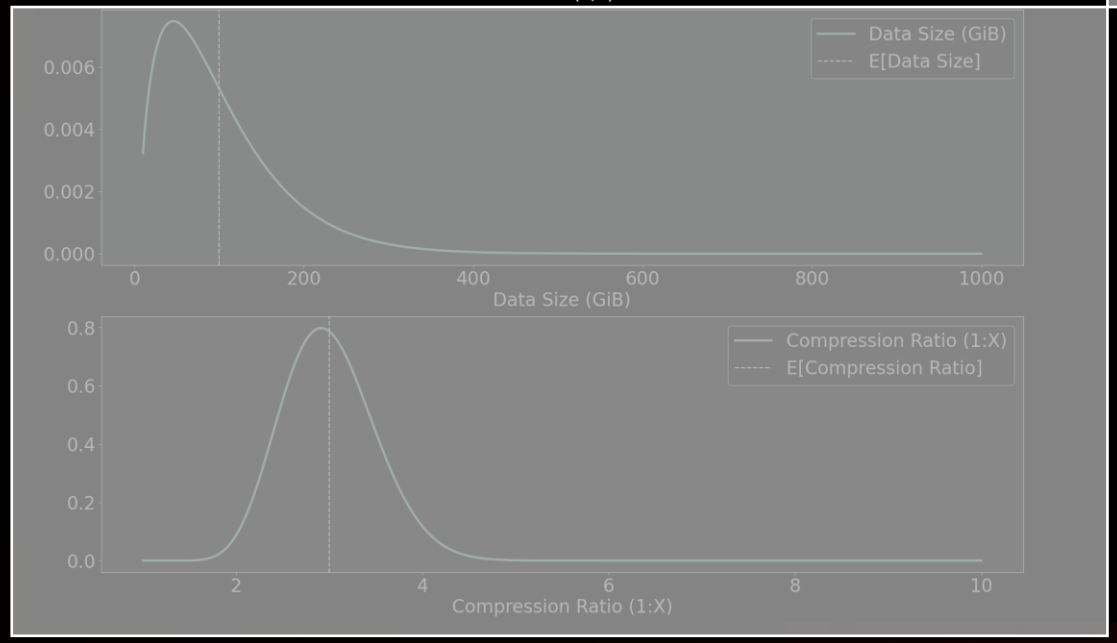
https://www.cs.cmu.edu/~harchol/PerformanceModeling/book.html

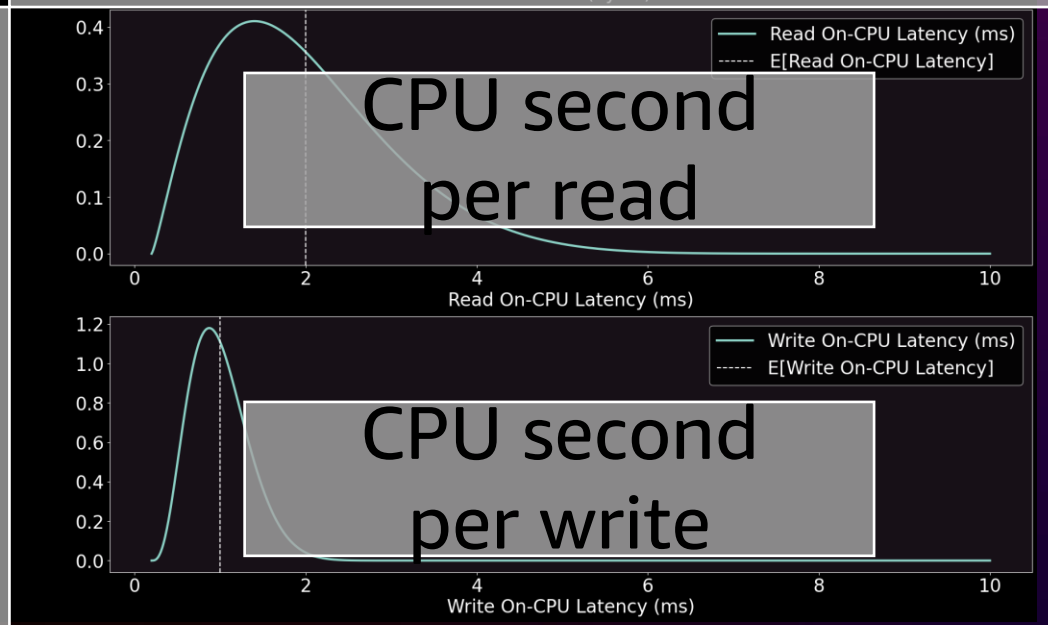
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.











© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Network

https://bit.ly/3DMTXYH

For the simple case, it's easy

$$\text{let } \mu_r = \frac{\text{bytes}}{\text{read}} \qquad \text{let } \mu_w = \frac{\text{bytes}}{\text{write}}$$

$$\text{let } \lambda_r = \frac{\text{read}}{\text{second}} \qquad \text{let } \lambda_w = \frac{\text{write}}{\text{second}}$$

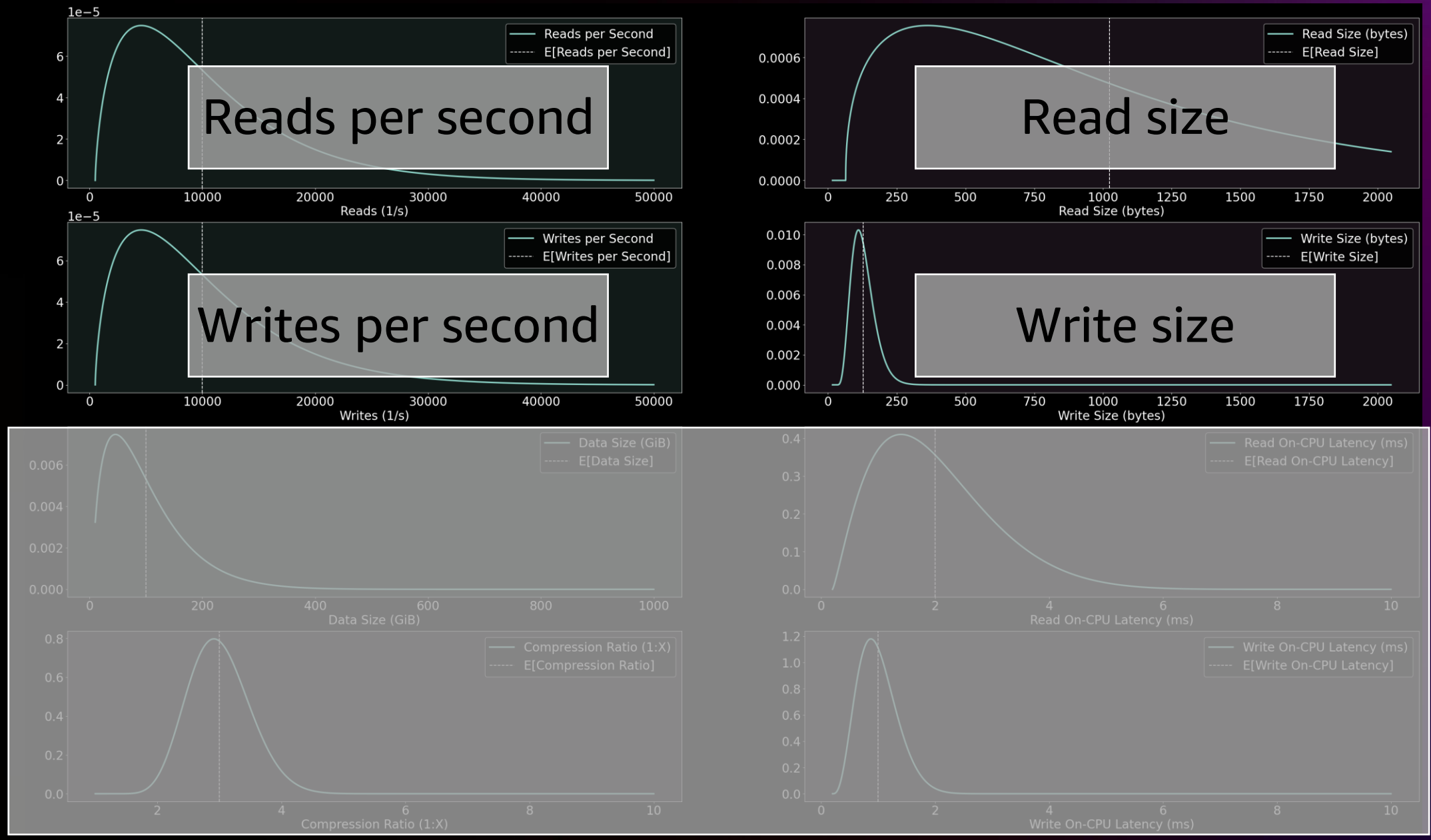
Tricky: In the complex case, we have to know the consistency level of reads and the replication factor of writes

$$\mathrm{BW}_{\text{simple}} = \mathrm{K} \times (\mu_r \times \lambda_r + \mu_w \times \lambda_w)$$

$$\mathrm{BW}_{\text{complex}} = \mathrm{K} \times (\mathrm{CL} \times (\mu_r \times \lambda_r) + \mathrm{RF} \times (\mu_w \times \lambda_w))$$

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Disk

https://bit.ly/3DPXN32

How the stateful service compacts data (merges, over-writes, etc.) matters

$$\text{size}_{\text{zone}} = \frac{\text{RF} \times \text{data size}}{\#\text{zones} \times \text{compression}}$$

$$\text{size}_{\text{node}} = \frac{\text{size}_{\text{zone}}}{\#\text{nodes}_{\text{zone}}} \times f(\text{compaction})$$

Tricky: Remember that network drives may have to be sized for IO (e.g., Amazon EBS gp2)

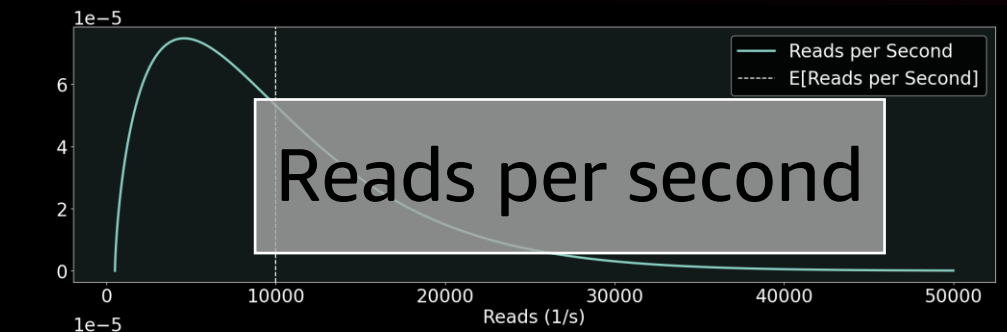
$$\text{Epem}_{\text{node}} = \text{size}_{\text{node}}$$

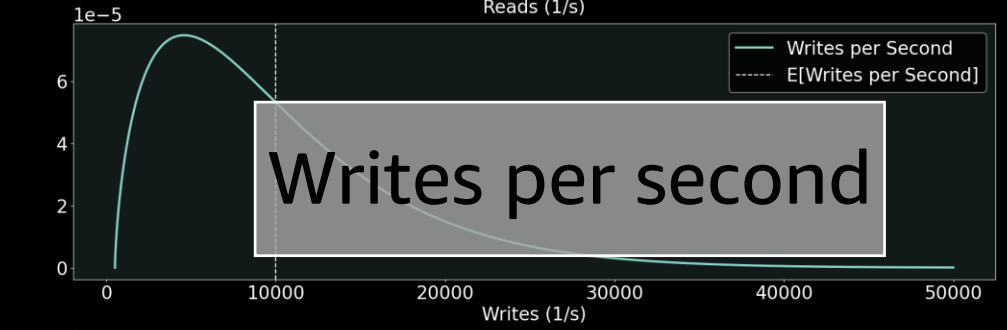
$$OR$$

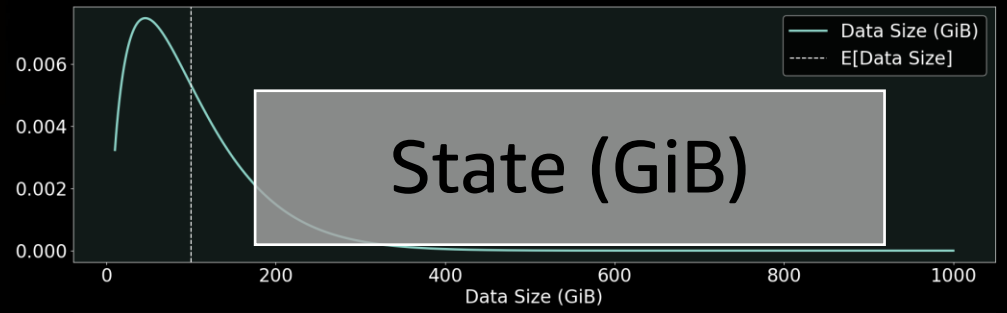
$$\text{EBS}_{\text{node}} = \max(\text{size}_{\text{node}}, f(\text{read BW}))$$

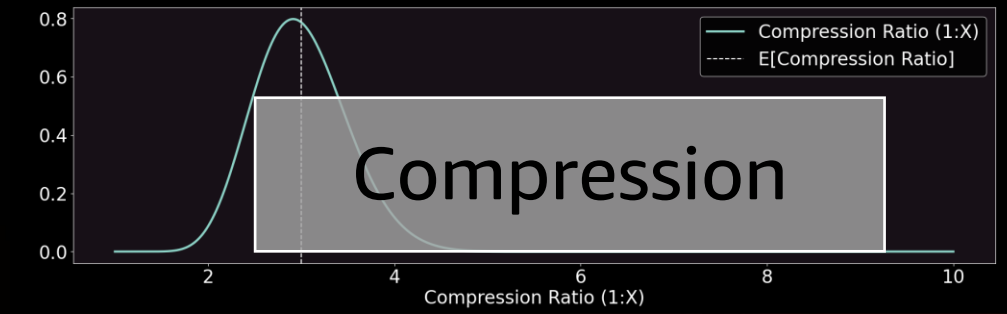
aws

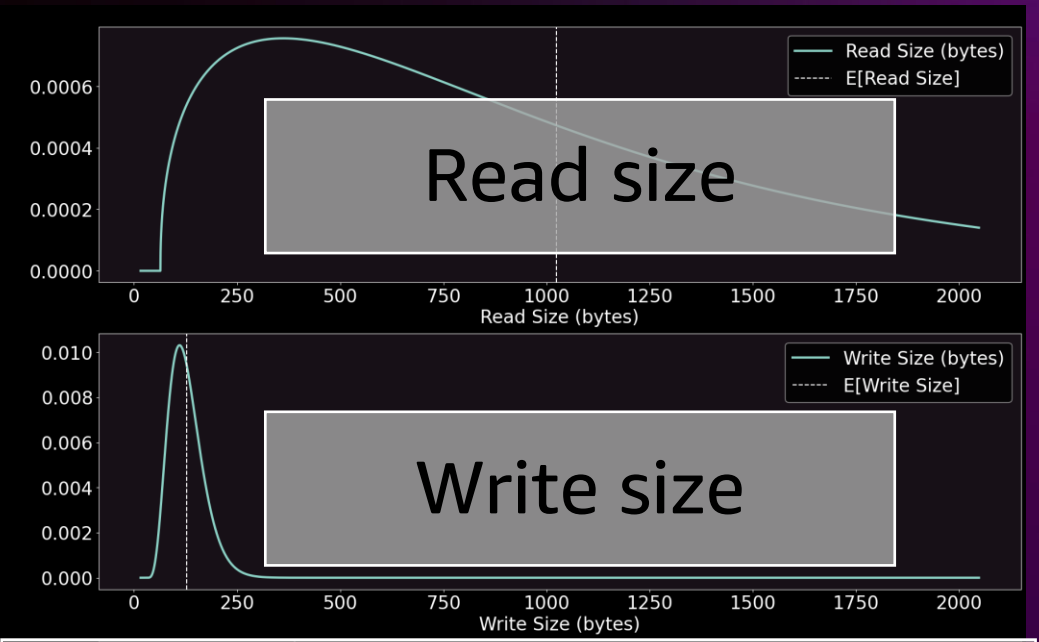
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

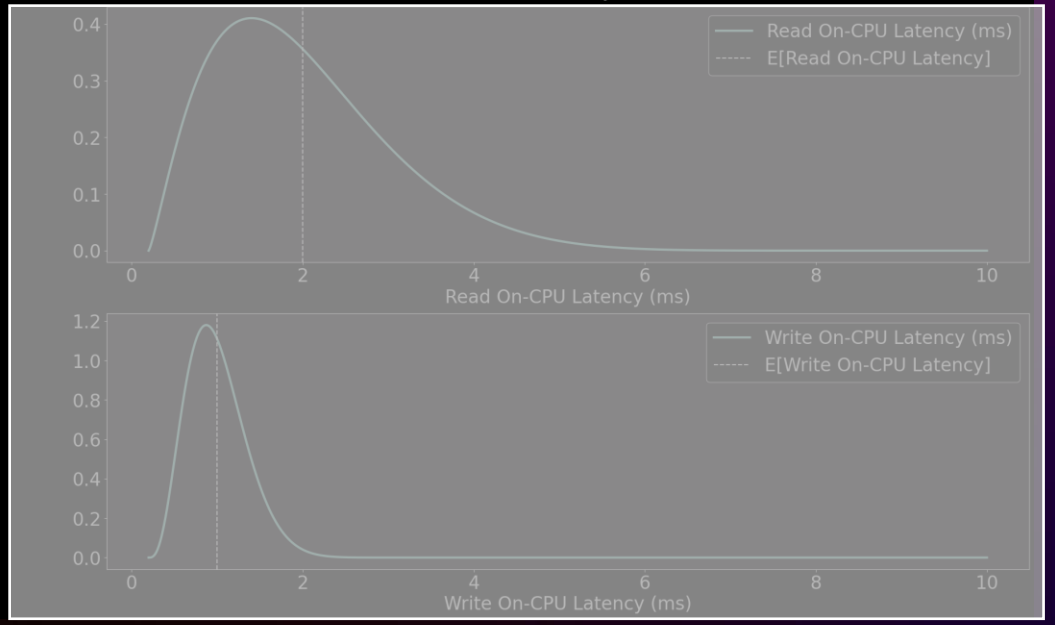












© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Memory

https://bit.ly/3zTq3ka

Fundamental trade-off: Spend memory on reads (page cache/read buffer) or writes (heap/write buffer)

Tricky: This depends on the number of nodes

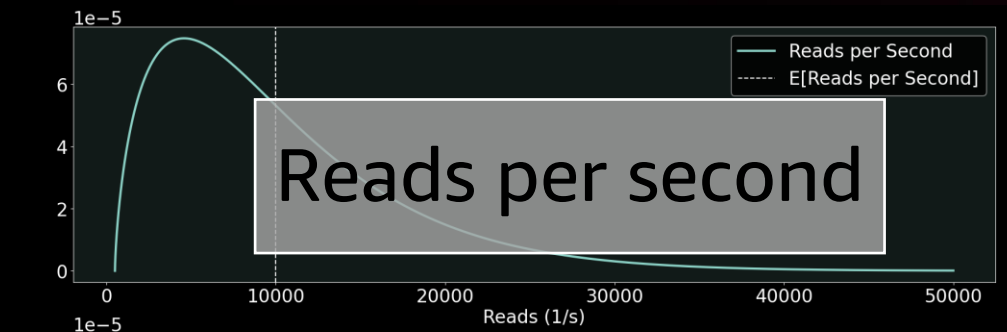
$$\mathrm{RAM}_{\mathrm{read}} = f(\text{data size, working set})$$
$$\mathrm{RAM}_{\mathrm{write}} = f(\text{write BW, compaction})$$

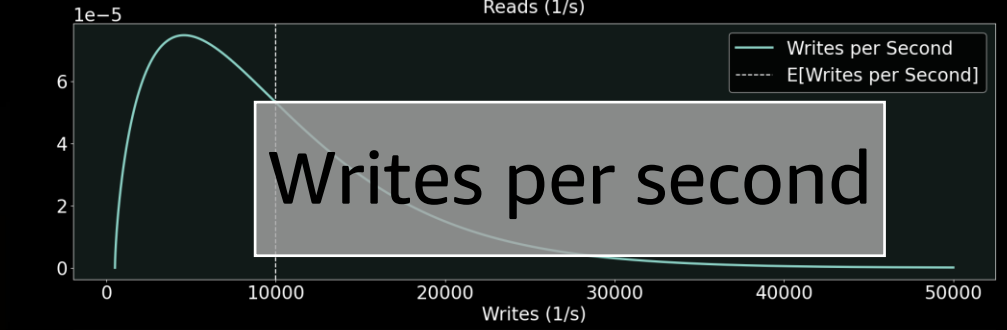
$$\mathrm{RAM}_{\mathrm{JVM}} = f(\text{write BW, read BW})$$
$$\mathrm{RAM}_{\mathrm{system}} = f(\text{sidecars, kernel})$$

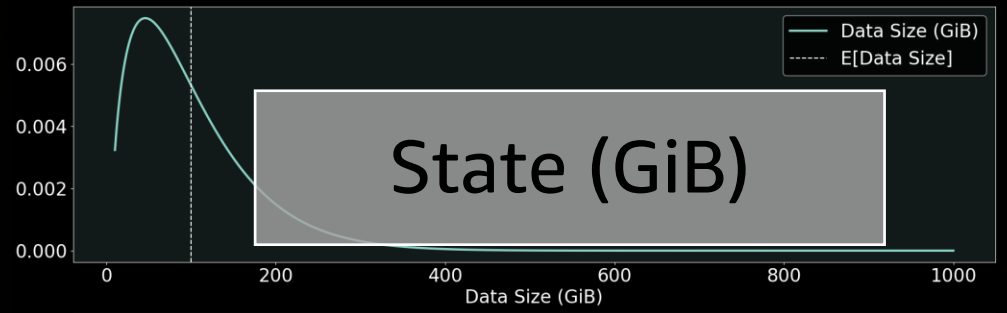
$$RAM = \sum RAM_{component}$$

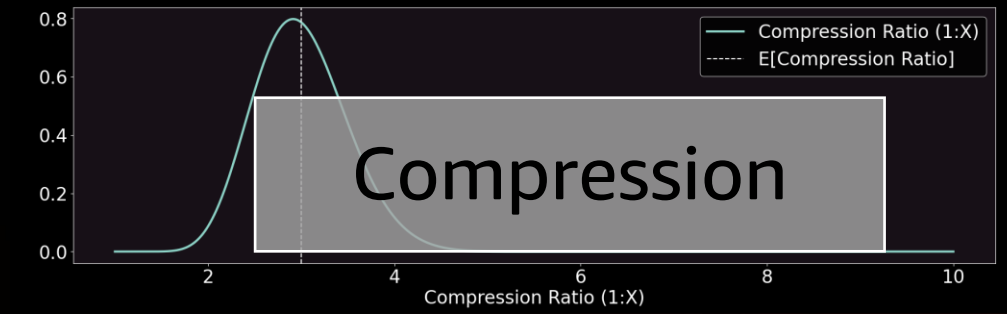
aws

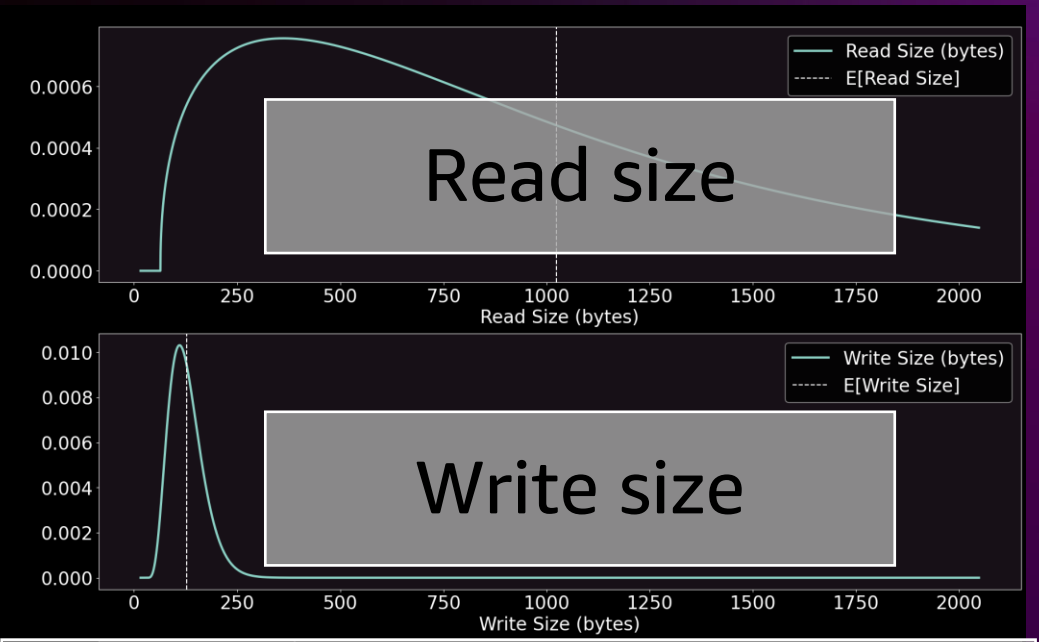
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

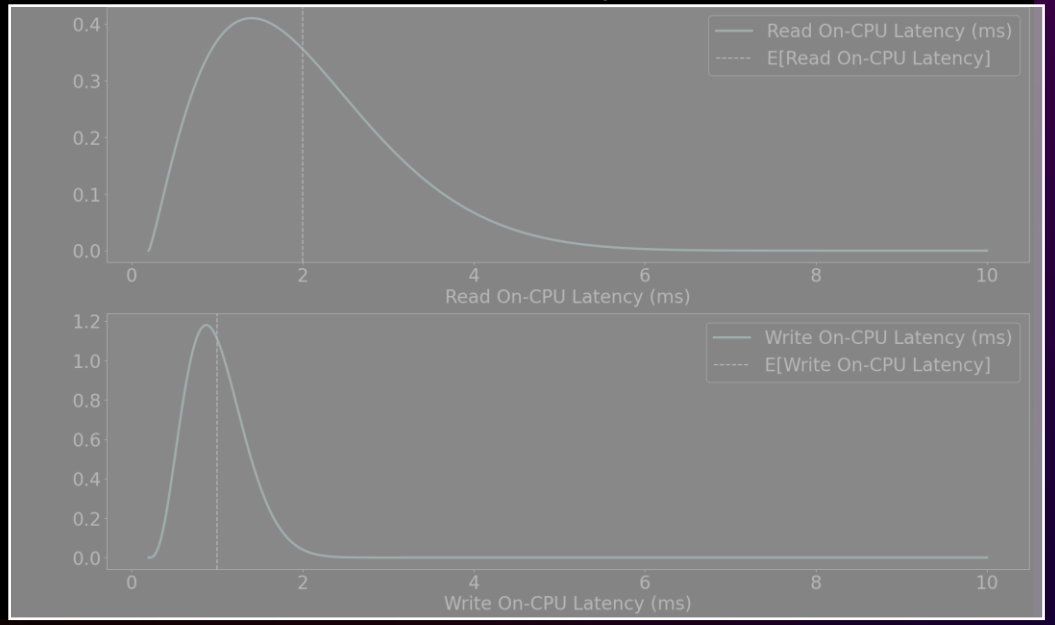






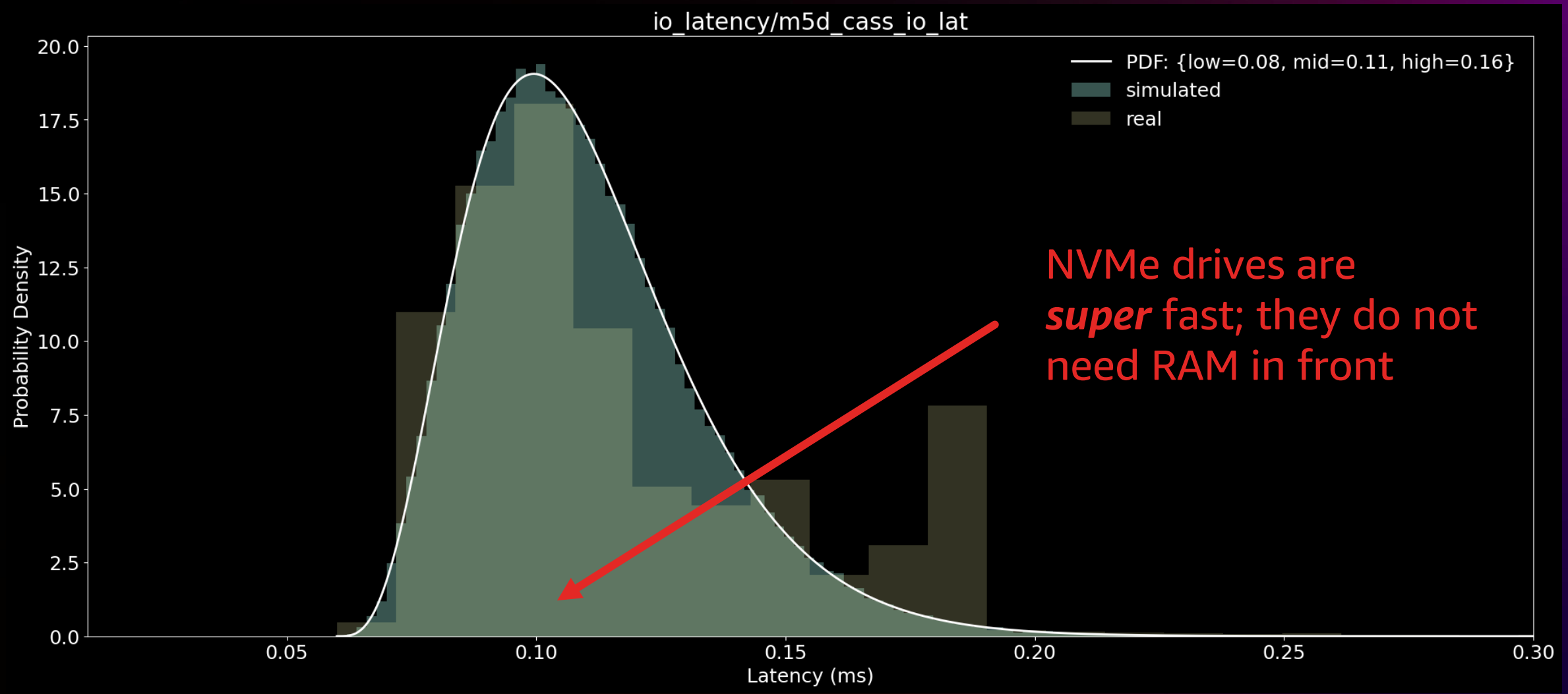






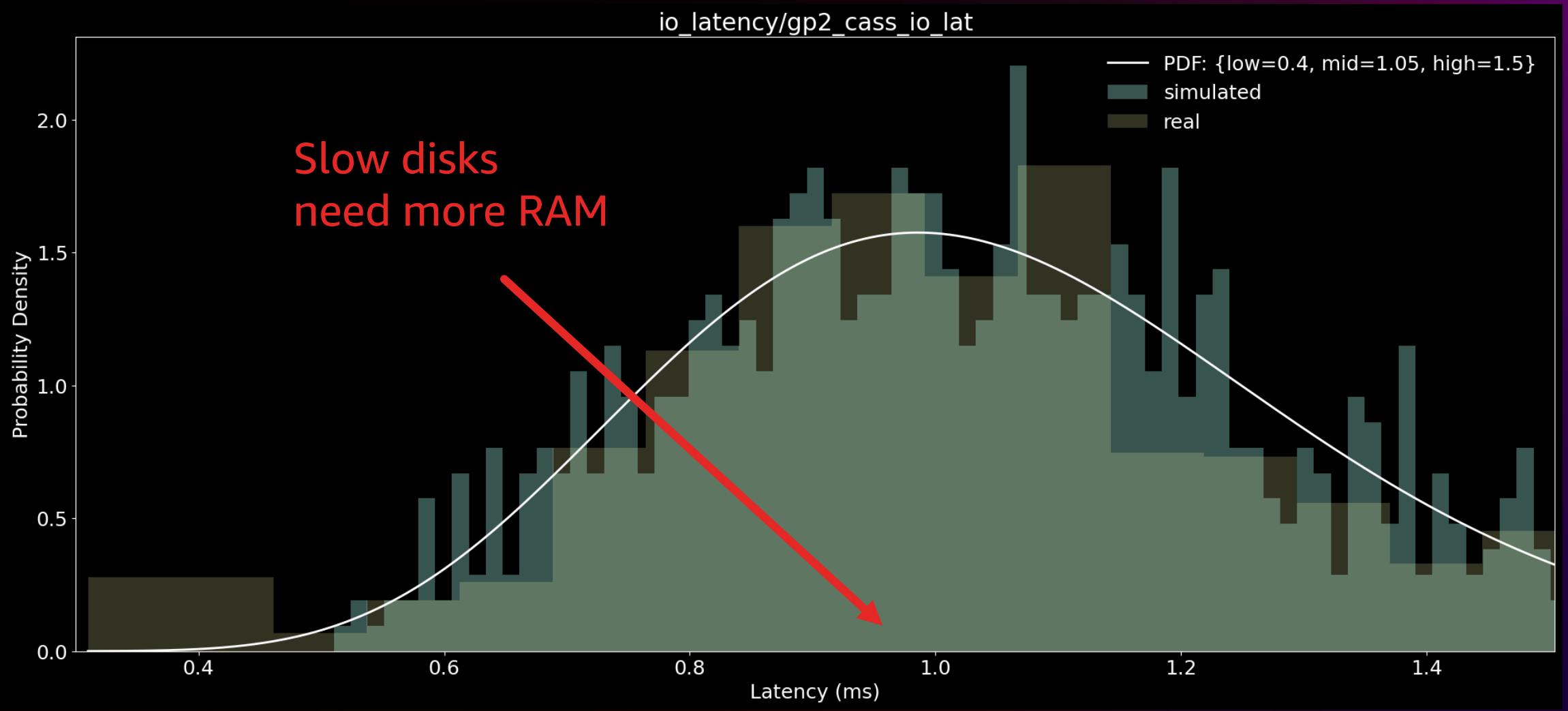
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working set?



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Working set?



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Calculate from SLO

https://bit.ly/3t9n9nD

Take latency service level objective and compare to the drive latency distribution

Ask, “How much memory do I have to buy to guarantee my service meets SLO given drives that don’t meet SLO?”

Example: P95 latency of Amazon EBS gp2 is ~1.35 ms; for a database with a <5 ms SLO, we need ~45% of our data in RAM; for a cache with a <1ms SLO, we need ~99% in RAM

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Success!

Now we can capacity plan for a **specific instance type** and a **concrete user desire**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Success?

Now we can capacity plan for a **specific instance type** and a **concrete user desire**

But we have dozens of shapes and cloud drives to consider …

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Solution: For loop!

$$\forall H(M_{cassandra}(D,\ H,\ P)) \rightarrow C_H$$

choose

$$C = \text{argmin}_H(cost(C_H))$$

$M(D, \text{m5.2xlarge})) \rightarrow$ 12 m5.2xlarge + 200GiB gp2

$M(D, \text{m5.4xlarge})) \rightarrow$ 6 m5.4xlarge + 400GiB gp2

$M(D, \text{r5d.2xlarge})) \rightarrow$ 6 r5d.2xlarge

... now pick the cheapest one

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Success!

Now we can capacity plan over all **instance and cloud drive types** and a **concrete user desire**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Success?

Now we can capacity plan over all **instance and cloud drive types** and a **concrete user desire**

But we don’t have a concrete user desire, we have like 20 **distributions** of inputs that could lead us to millions of possibilities

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Capacity planning with uncertainty

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Monte Carlo to the rescue

Simulate possible worlds

Do enough simulations to get **tail** events

Pick the cluster that is the **least regretful** over all worlds



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Regret

We regret money

- Bought too little
- Bought too much

We regret operator time (money)

- Running out of disk space
- Operational overhead

$$\text{regret}(X, Y)_{\$} = K_{\$}(X_{\$} - Y_{\$})^{r_{\$}}$$

$$\text{regret}(X, Y)_{\text{disk}} = K_{\text{disk}}(X_{\text{disk}} - Y_{\text{disk}})^{r_{\text{disk}}}$$

$$\text{regret}(X, Y) = \sum_{i} \text{regret}(X, Y)_i$$

$$\text{regret}(X_i) = \sum_{j}^{X} \text{regret}(X_i, X_j)$$

$$\boxed{\text{regret}_{\text{least}} = \text{argmin}_X (\text{regret})}$$

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Regret

https://bit.ly/3fNcwnt

```
@staticmethod
def regret(
    regret_params: CapacityRegretParameters,
    optimal_plan: CapacityPlan,
    proposed_plan: CapacityPlan,
) -> Dict[str, float]:
    """Cost model for how much we regret a choice

    After the capacity planner has simulated a bunch of possible outcomes
    We need to evaluate each cluster against the optimal choice for a
    given requirement. Our default model of cost is just related
    to money and disk footprint:

    Under an assumption that cost is a reasonable single dimension
    to compare to clusters on, we penalize under-provisioned (cheap)
    clusters more than expensive ones.
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example world comparison

World 1

We buy 48 i3en.xlarge,
costing $73,652

We require 6.6 TiB of space

World 2

We buy 96 r5.8xlarge,
costing $646,309

We require 17.9 TiB of space

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Buy world 1, live in world 2

World 1

We **bought** 48 i3en.xlarge, costing $73,652

We **have** 6.6 TiB of space

World 2

We **needed to buy** 96 r5.8xlarge, costing $646,309

We **needed** 17.9 TiB of space

$$\text{regret}(W_1 \text{ in } W_2)_{\$} = 1.25 \times |73,652.57 - 646,309.93|^{1.2} \approx 10M$$

$$\text{regret}(W_1 \text{ in } W_2)_{\text{disk}} = 1.10 \times |6,634.0 - 17,941|^{1.05} \approx 20K$$

$$\text{regret}(W_1 \text{ in } W_2) \approx 10 \text{ million dollars (underprovisioned)}$$

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Buy world 2, live in world 1

World 1

We **needed to buy** 48 i3en.xlarge, costing $73,652

We **needed** 6.6 TiB of space

World 2

We **bought** 96 r5.8xlarge, costing $646,309

We **have** 17.9 TiB of space

$$\text{regret}(W_2 \text{ in } W_1)_{\$} = 1.0 \times |73,652.57 - 646,309.93|^{1.2} \approx 8M$$

$$\text{regret}(W_2 \text{ in } W_1)_{\text{disk}} = 0.0 \times |6,634.0 - 17,941|^{1.05} = 0K$$

$$\text{regret}(W_2 \text{ in } W_1) \approx 8 \text{ million dollars (overprovisioned)}$$

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Regret is not symmetric!

- Choice of constants determines relative cost for **your organization**
- **Underprovisioning** (buying too little)
- **Overprovisioning** (buying too much)

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worked example

https://bit.ly/3DN5aIz

```
desires = CapacityDesires(
    service_tier=1,
    query_pattern=QueryPattern(
        # Not sure exactly how much QPS we will do, but we think ~10k / s
        estimated_read_per_second=Interval(
            low=1_000, mid=10_000, high=100_000, confidence=0.98
        ),
        estimated_write_per_second=Interval(
            low=1_000, mid=10_000, high=100_000, confidence=0.98
        ),
    ),
    # Not sure how much data, but we think around 100 GiB
    data_shape=DataShape(
        estimated_state_size_gib=Interval(
            low=10,    mid=100,    high=1_000,   confidence=0.98
        ),
    )...
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Least regret

https://bit.ly/3DN5aIz

```
from service_capacity_modeling.capacity_planner import planner
from service_capacity_modeling.models.org import netflix

# Load up the Netflix capacity models
planner.register_group(netflix.models)

# Plan a cluster
plan = planner.plan(
    model_name=model_name,
    region="us-east-1",
    desires=desires,
    simulations=1024,
    explain=True
)
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Least regret

https://bit.ly/3DN5aIz

```
Least Regret Choice:
--------------------
 12 m5d.xlarge costing 8966.38 -> 43580727.7051395

All Choices
-----------
{'  6  r5d.large': 20,
 '  6 m5d.xlarge': 221,
 ' 12 m5d.xlarge': 426,
 ' 24 m5d.xlarge': 299,
 ' 48 m5d.xlarge': 57,
 ' 96 m5d.xlarge': 1}
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Worked example

https://bit.ly/3DN5aIz

```
desires_footprint = CapacityDesires(
    service_tier=1,
    query_pattern=QueryPattern(
        # Not sure exactly how much QPS we will do, but we think ~10k / s
        estimated_read_per_second=Interval(
            low=1_000, mid=10_000, high=100_000, confidence=0.98
        ),
        estimated_write_per_second=Interval(
            low=1_000, mid=100_000, high=1_000_000, confidence=0.98
        ),
    ),
    # Not sure how much data, but we think around 100 GiB
    data_shape=DataShape(
        estimated_state_size_gib=Interval(
            low=10,    mid=10_000,    high=100_000,   confidence=0.98
        ),
    )...
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Least regret

https://bit.ly/3DN5aIz

```
from service_capacity_modeling.capacity_planner import planner
from service_capacity_modeling.models.org import netflix

# Load up the Netflix capacity models
planner.register_group(netflix.models)

# Plan a cluster
plan = planner.plan(
    model_name=model_name,
    region="us-east-1",
    desires=desires_footprint,          # Change the desire
    simulations=1024,
    explain=True
)
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Least regret

https://bit.ly/3DN5aIz

```
Least Regret Choice:
--------------------
 48 i3en.xlarge costing 73876.09 -> 328005165.5220482
```

```
# A LOT more variability in this one.

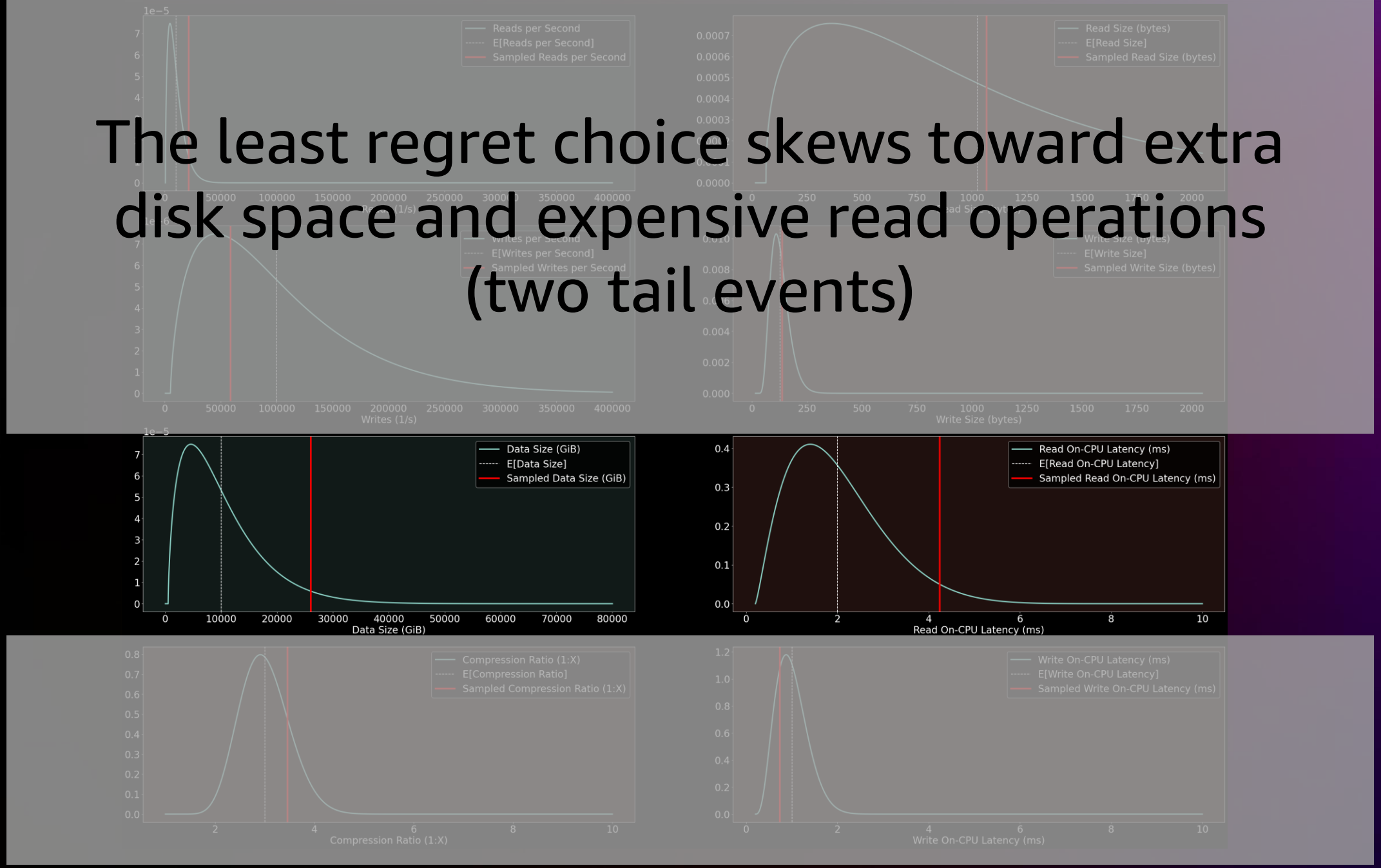
# But we still picked 48 i3en.xlarge
# 118/1024 times
```

```
All Choices
-----------
{
'  6 i3en.2xlarge': 7,
'  6 i3en.3xlarge': 10,
'  6 m5.8xlarge': 7,
'  6 r5.8xlarge': 5,
' 12 i3en.2xlarge': 69,
' 12 i3en.3xlarge': 75,
' 12 i3en.xlarge': 14,
' 12 m5.8xlarge': 4,
' 12 r5.2xlarge': 2,
' 12 r5.4xlarge': 4,
' 12 r5.8xlarge': 7,
' 24  i3.xlarge': 4,
' 24 i3.2xlarge': 3,
' 24 i3en.2xlarge': 159,
' 24 i3en.3xlarge': 89,
' 24 i3en.xlarge': 118,
' 24 m5.8xlarge': 8,
' 24 r5.2xlarge': 2,
' 24 r5.4xlarge': 3,
' 24 r5.8xlarge': 1,
' 48  i3.xlarge': 7,
' 48  m5.xlarge': 1,
' 48 i3.2xlarge': 2,
' 48 i3en.2xlarge': 55,
' 48 i3en.3xlarge': 11,
' 48 i3en.xlarge': 204,
' 48 m5.2xlarge': 1,
' 48 m5.8xlarge': 1,
' 48 m5d.xlarge': 17,
' 48 r5.2xlarge': 2,
' 96  i3.xlarge': 9,
' 96  m5.xlarge': 1,
' 96  r5.xlarge': 1,
' 96 i3.2xlarge': 1,
' 96 i3en.2xlarge': 1,
' 96 i3en.3xlarge': 1,
' 96 i3en.xlarge': 40,
' 96 m5d.2xlarge': 22,
' 96 m5d.xlarge': 53,
' 96 r5.2xlarge': 1,
' 96 r5.8xlarge': 2}
```

aws

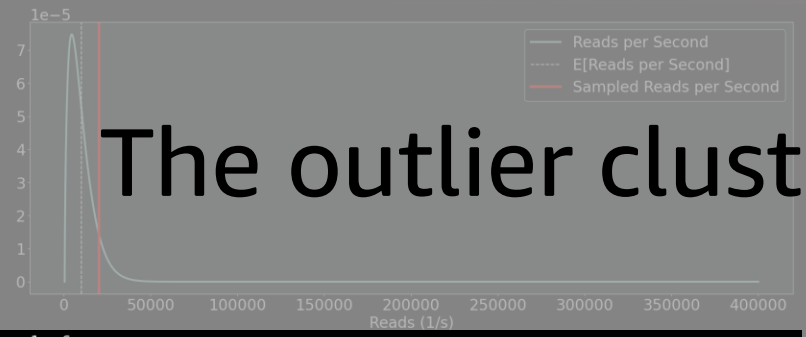
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

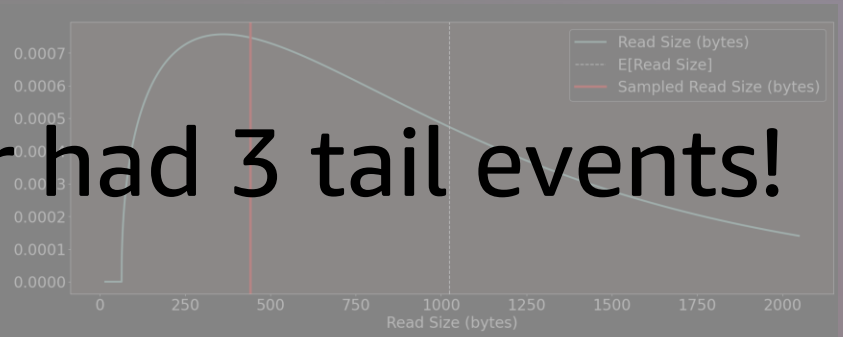
The least regret choice skews toward extra disk space and expensive read operations (two tail events)

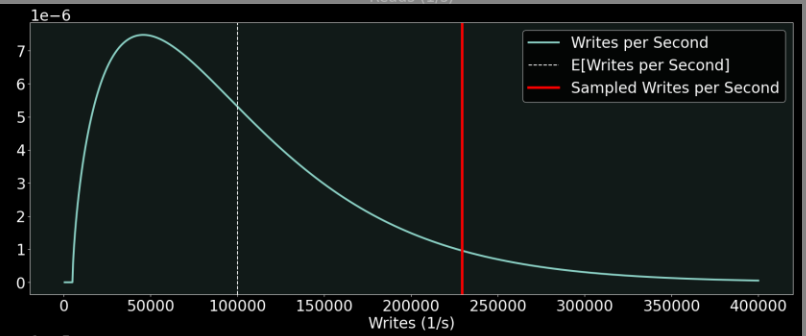


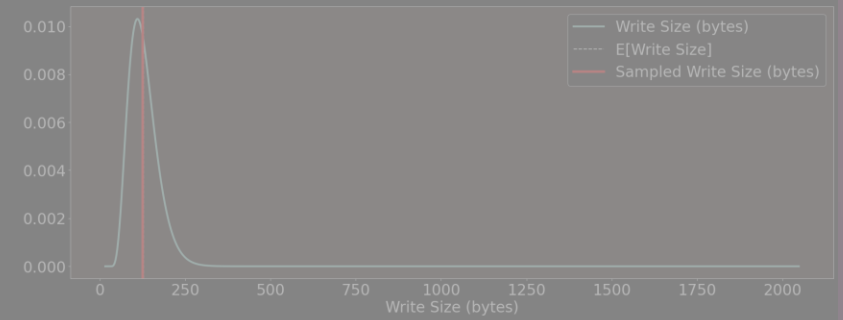
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

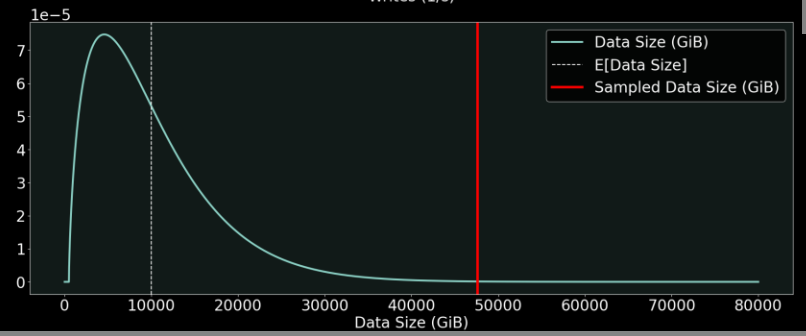
The outlier cluster had 3 tail events!

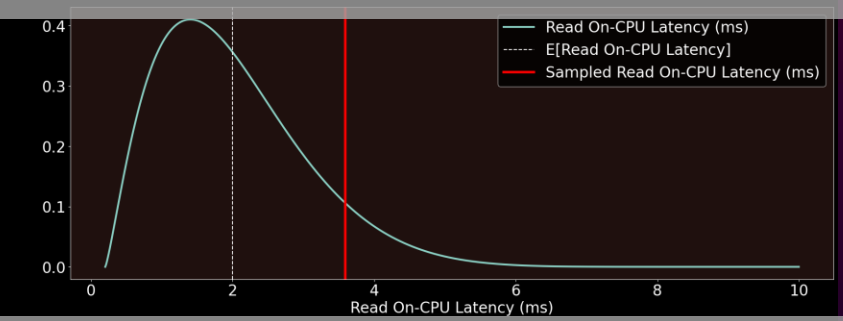


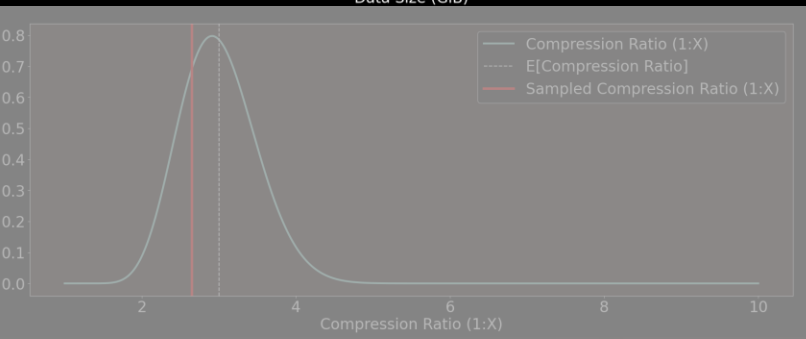


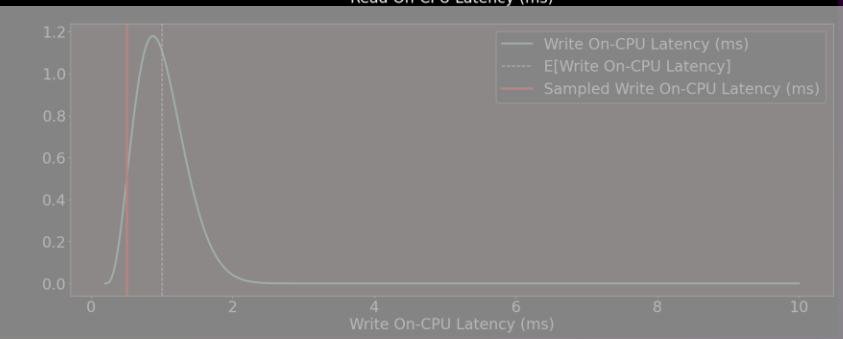












© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Monitoring choices

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Measure CPU worker queuing – /proc/schedstat

```
def gather_metric():
    # scale time spent in the scheduler by this factor
    schedstat_lines = open('/proc/schedstat').readlines()
    delays = [
      int(i.split(' ')[8]) for i
      in schedstat_lines if i.startswith('cpu')
    ]

    delays = delays or [0]
    return sum(delays) / float(len(delays))
```

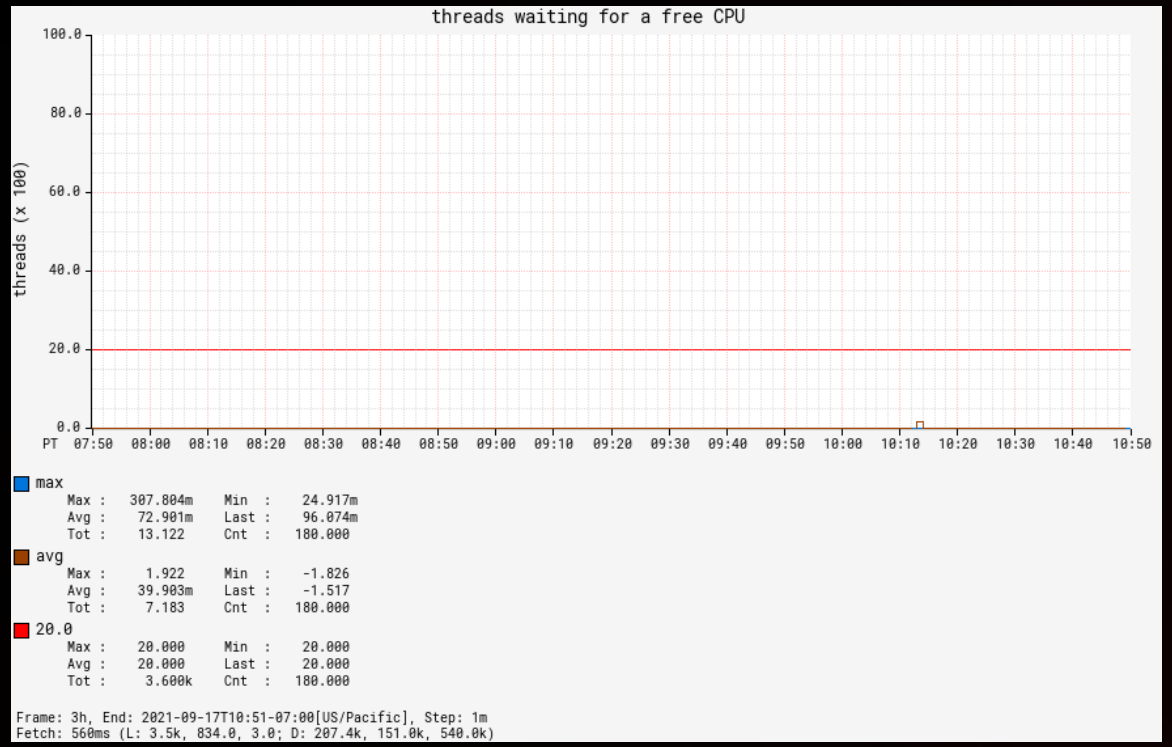
https://www.kernel.org/doc/html/latest/scheduler/sched-stats.html

aws

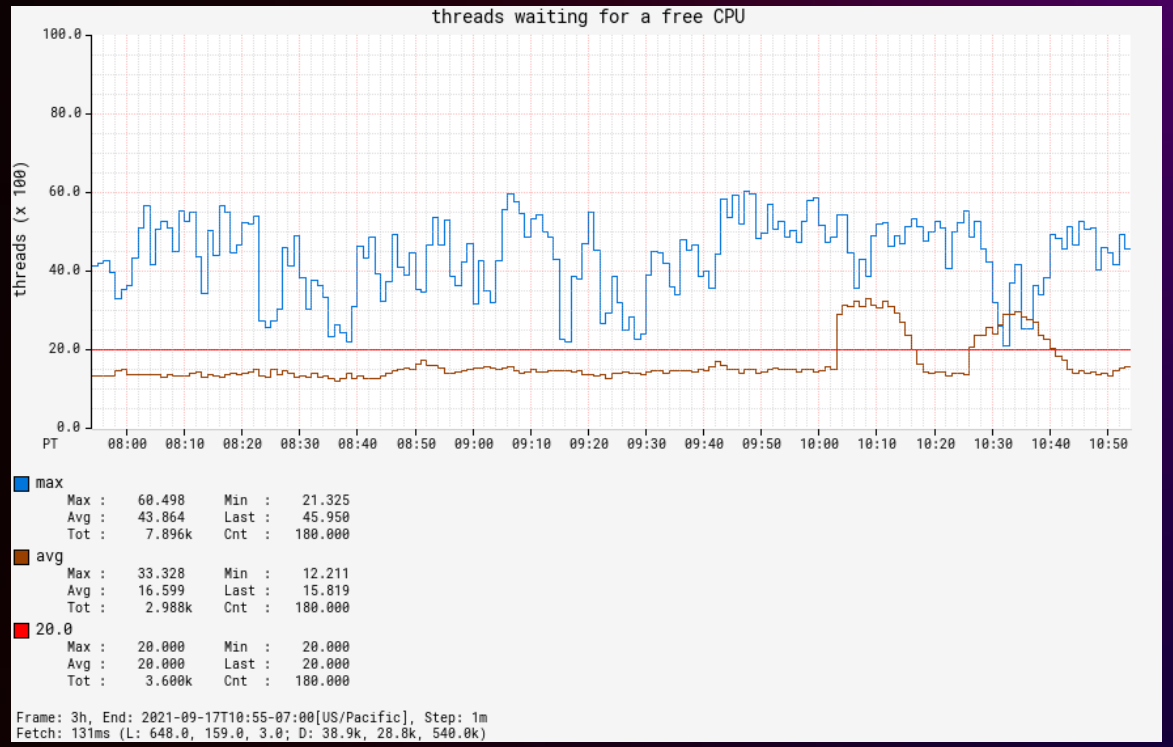
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CPU – % time waiting for free CPU

Tier 0 – latency sensitive



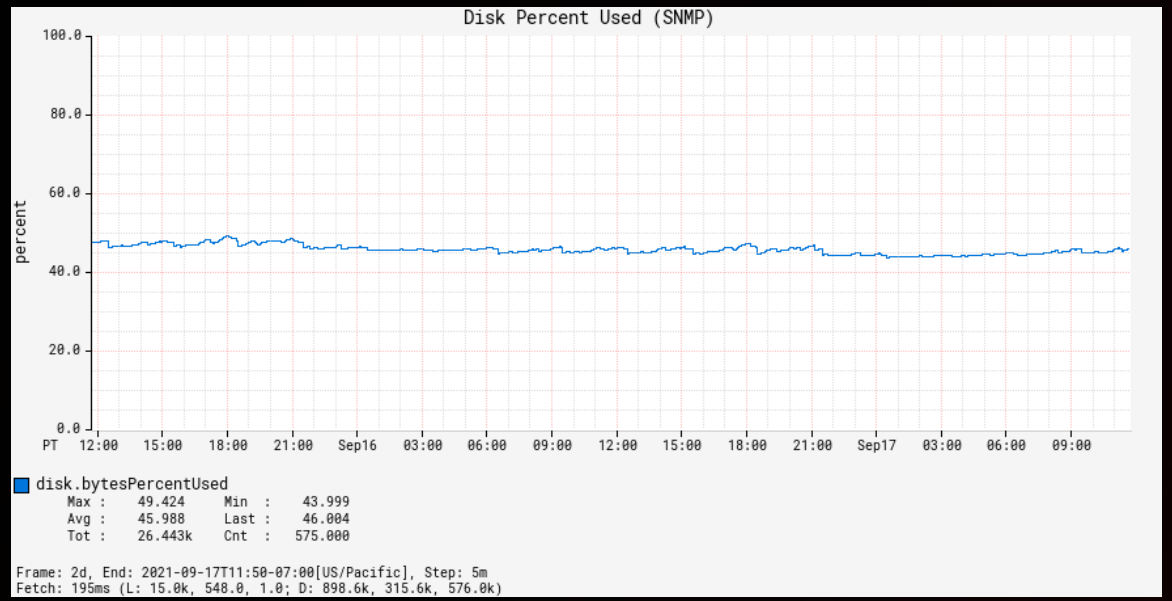
Tier 3 – throughput oriented



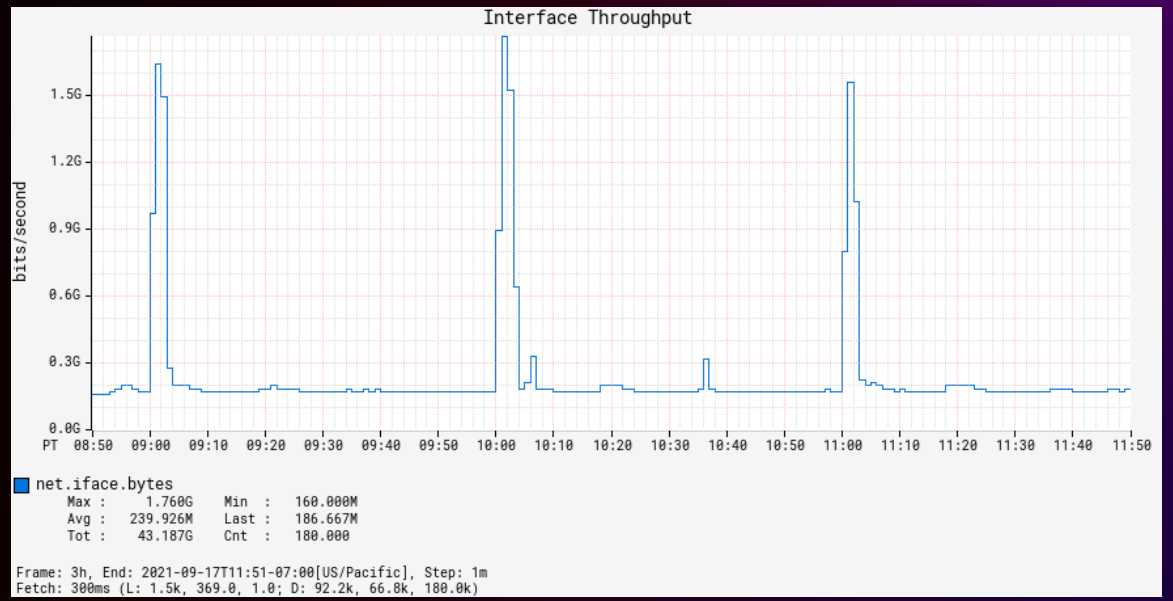
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Disk and network – Utilization

Disk space – utilization



Network bandwidth – utilization



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Memory

Memory for reads

- Monitor /proc/diskstats column 4 (read/s), should be << your drive's read IO capacity
- Use bcc cachestat to monitor page cache misses

Memory for writes

- Monitor /proc/diskstats column 10 (writes) and 8 (512 byte sectors written); write throughput should be << your drive's write IO capacity
- Monitor flush frequency; should be <1/10 minutes

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What to do now?

Ran out of?	Action?
Network	M5 →m5**n**
CPU	r6id →**m**6id | **c**6id
Disk	m6id →**i**4i | **i**3e**n**
Everything?	**Scale up**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What to do now?

Ran out of?	Action?
Network	M5 →m5**n**
CPU	r6id →**m**6id | **c**6id
Disk	m6id →**i**4i | **i**3e**n**
Everything?	**Scale up**

Buy what you need!

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Conclusions

Understanding hardware

We measured, priced, and imposed lifecycle on our hardware

Capacity planning

Apply queuing theory with anger; simulate worlds, pick the least regretful

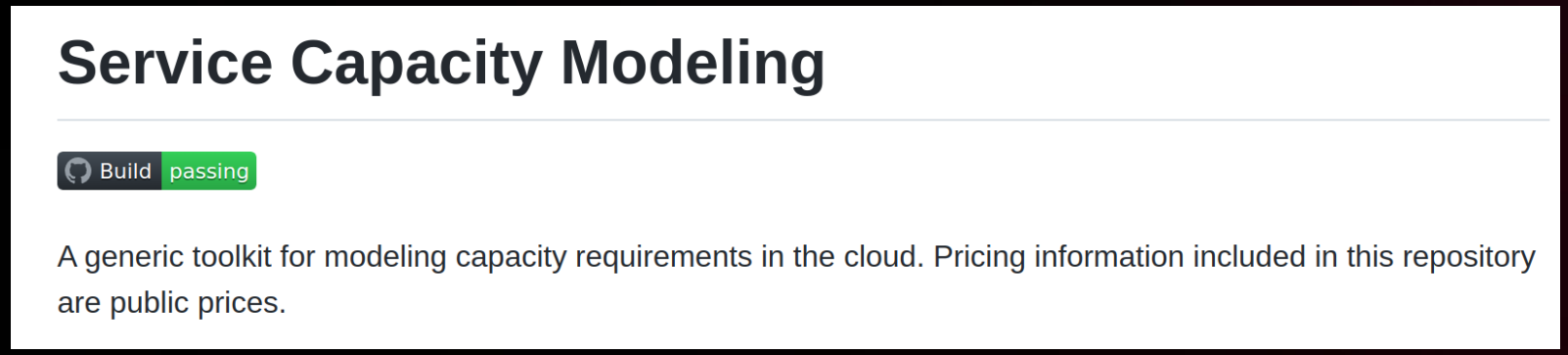
Monitoring your choices

Buy more of what you need

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This system is open source



https://github.com/Netflix-Skunkworks/service-capacity-modeling

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Joseph Lynch
josephl@netflix.com
https://jolynch.github.io/

Prateek Sharma
prshaz@amazon.com



Please complete the session survey in the **mobile app**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.