Improving Cassandra Client Load Balancing

Ammar Khaku
Joey Lynch

Speaker

Ammar Khaku

Senior Software Engineer
Cloud Data Engineering at Netflix

Database clients, Java libraries

https://akhaku.com/

Speaker

Joey Lynch

Senior Software Engineer
Cloud Data Engineering at Netflix
Cassandra Committer

Database shepherd and data wrangler

https://jolynch.github.io/

Outline

Load Balancing Background

Why Stateful Load Balancing is Special

Proposed Solution - Weighted Least Loaded

Experiments and Real World Results

Goal: Upgrade to Datastax 4

Had some performance issues at scale with LoadBalancer and Throttler.

(Un)Balance The Load

A quick crash course on queueing theory and load balancing

Best in class implementations

HAProxy, Nginx, Envoy

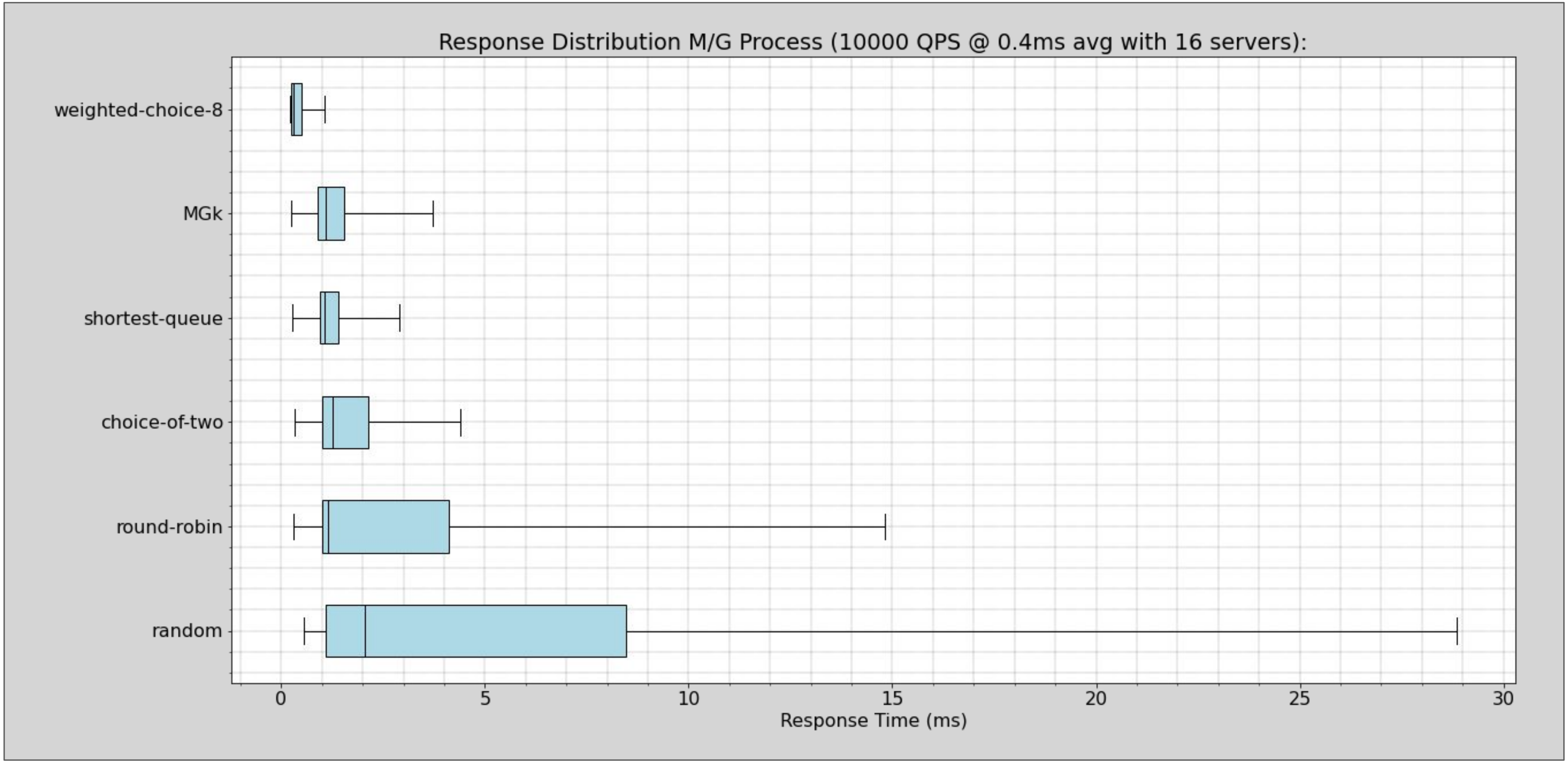
- Weighted **Round Robin**
- Weighted **Least Connection/Load**
- Weighted **Choice of N** (random/hash)

Netflix gRPC: **Random Choice of 2**

Google uses **Random Subsetting** with weighted **Round Robin**

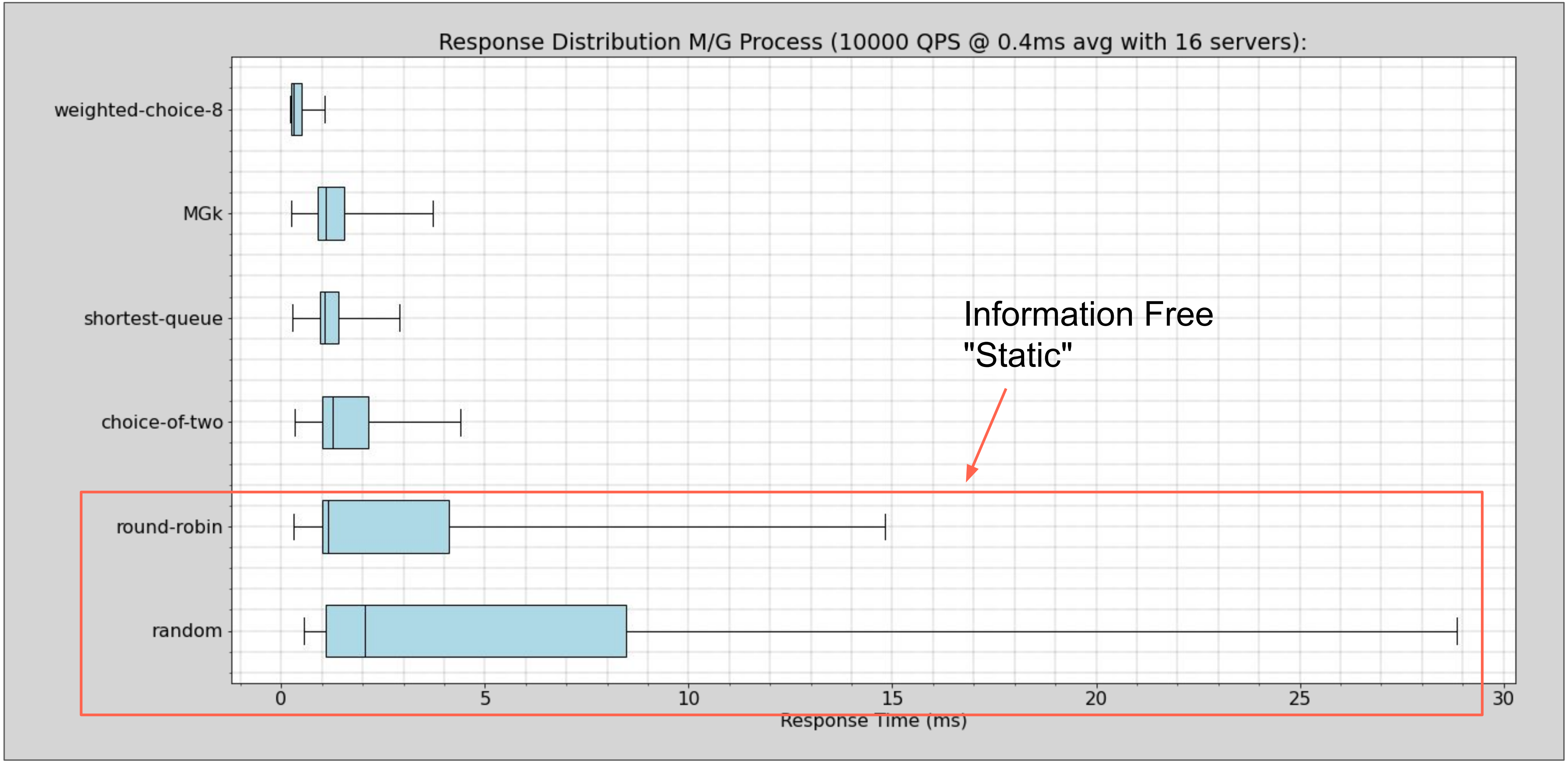
Many DB clients choose **Random**

What to choose?



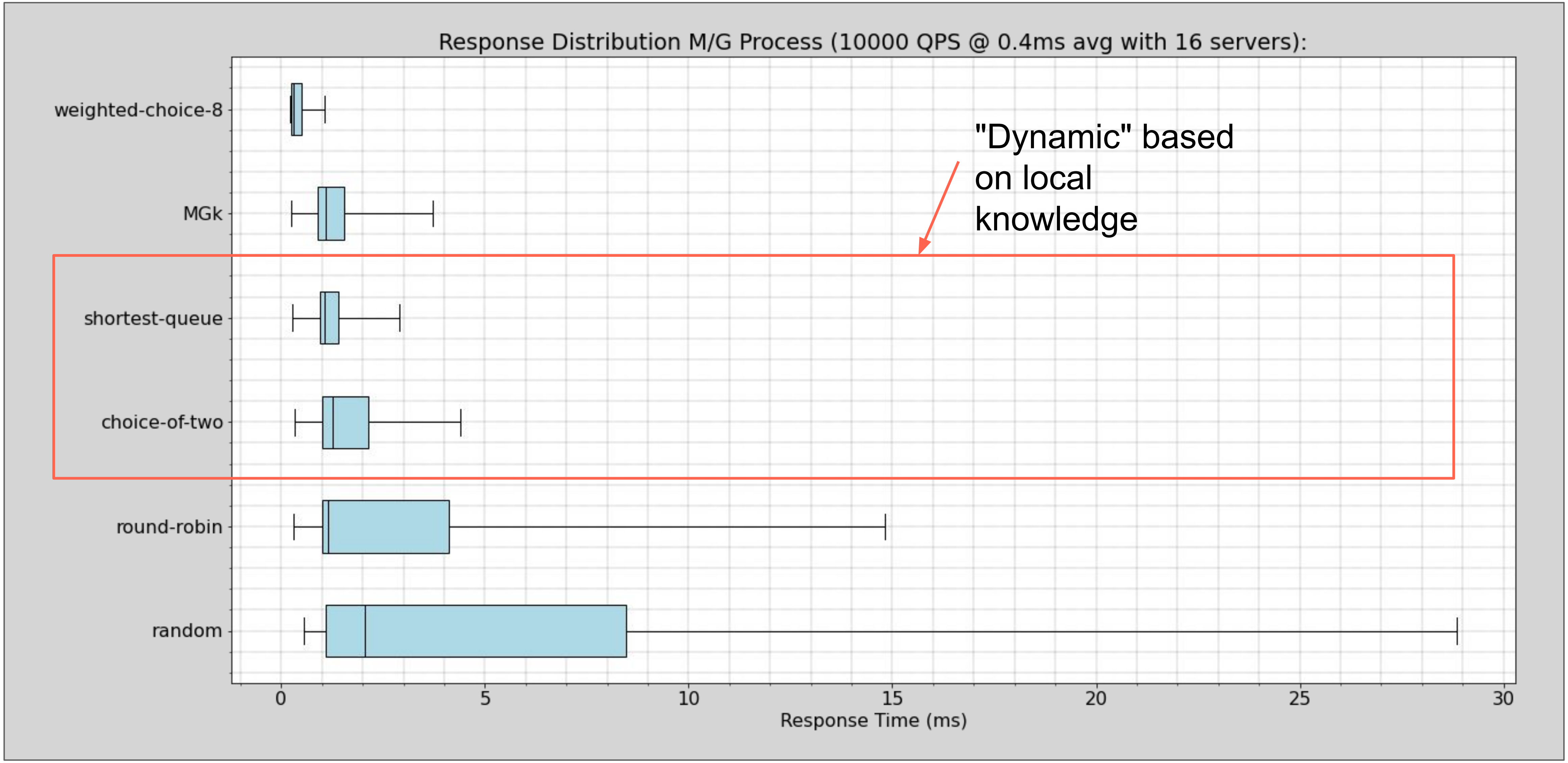
https://github.com/jolynch/performance-analysis/blob/master/notebooks/queueing_theory/load_balancing_analysis.ipynb

What to choose?



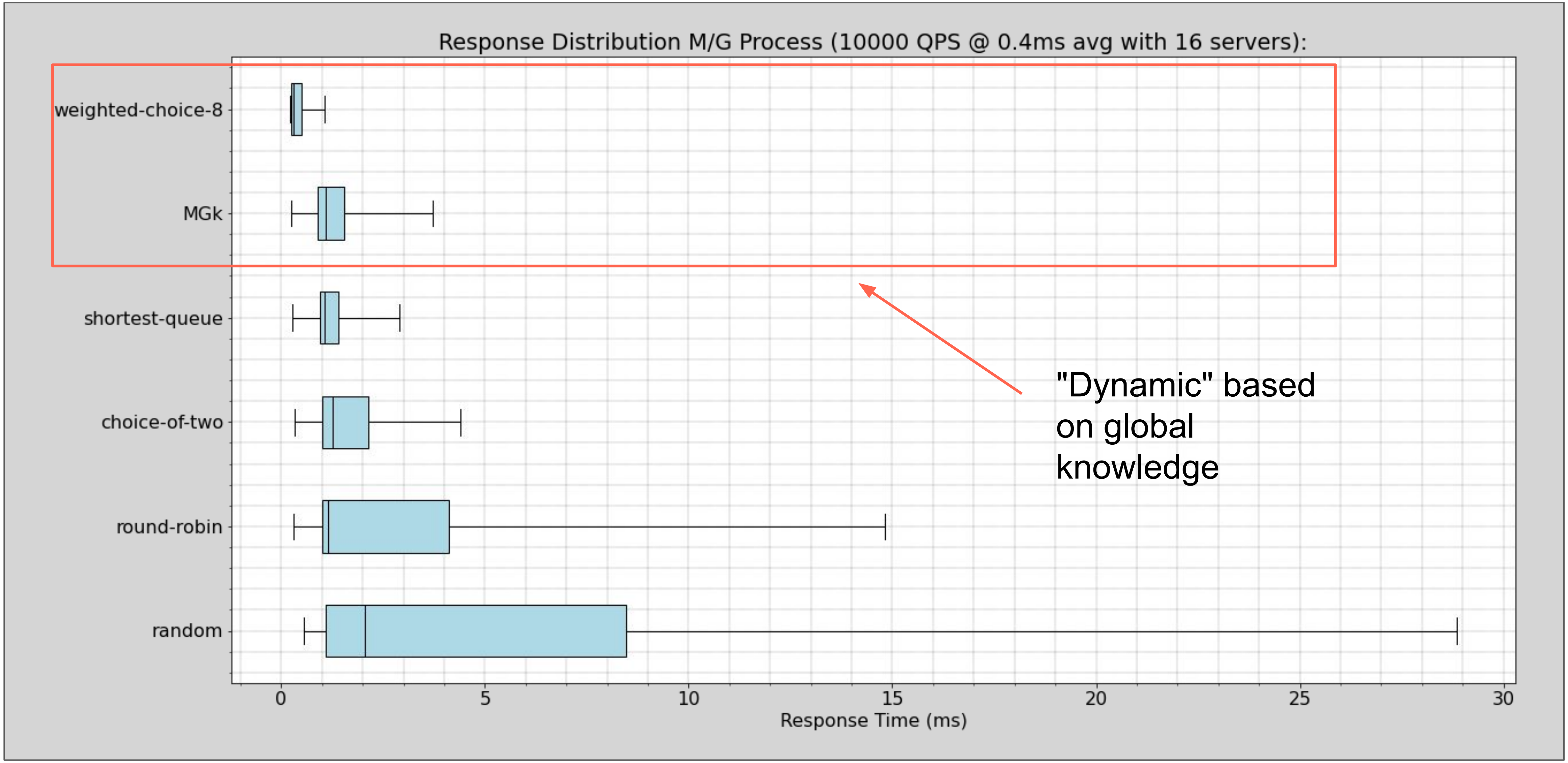
https://github.com/jolynch/performance-analysis/blob/master/notebooks/queueing_theory/load_balancing_analysis.ipynb

What to choose?



https://github.com/jolynch/performance-analysis/blob/master/notebooks/queueing_theory/load_balancing_analysis.ipynb

What to choose?



https://github.com/jolynch/performance-analysis/blob/master/notebooks/queueing_theory/load_balancing_analysis.ipynb

What to choose?

HAProxy recommends least connections as being strictly dominate to choice of 2 with an efficient impl

This matches the math and literature absent information.

Google allows servers to communicate back with clients to adjust weights in RR. Very clever.

Stateful Load Balancing

State makes the problem different

What makes datastores special?

The node you hit matters!

- Postgres: master, replica
- ZooKeeper: leader, followers
- CockroachDB: lease holder

What makes Cassandra special?

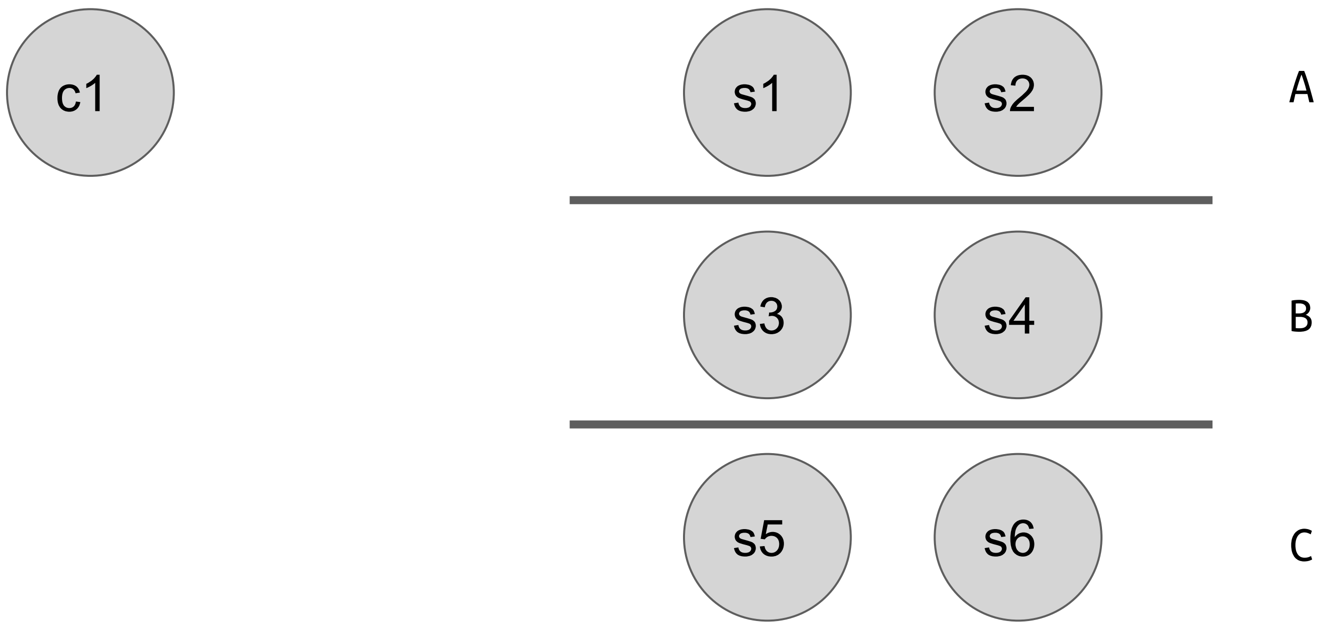
1. For any piece of data we typically have one **replica** per availability zone

2. Depending on the **consistency** we may need to hop to more hosts

3. Datastores have hiccups frequently (drives mostly)

4. Our network latency is **asymmetric**

Stateful load balancing with real networks



	A	B	C
A	**150us**	800us	250us
B	800us	**220us**	850us
C	380us	700us	**160us**

DataStax Java Driver for Apache Cassandra®

DataStax Java Driver 3.x for Apache Cassandra®

No Token? Round Robin

Token Aware? Hash key, shuffle replicas*, return first. (**random subsetting**)

DataStax Java Driver 3.x for Apache Cassandra®

Slow to react to slow coordinators, erroring coordinators, paused coordinators, etc …

Traffic often goes **cross-zone**

DataStax Java Driver 4.x for Apache Cassandra®

No Token?

Round Robin

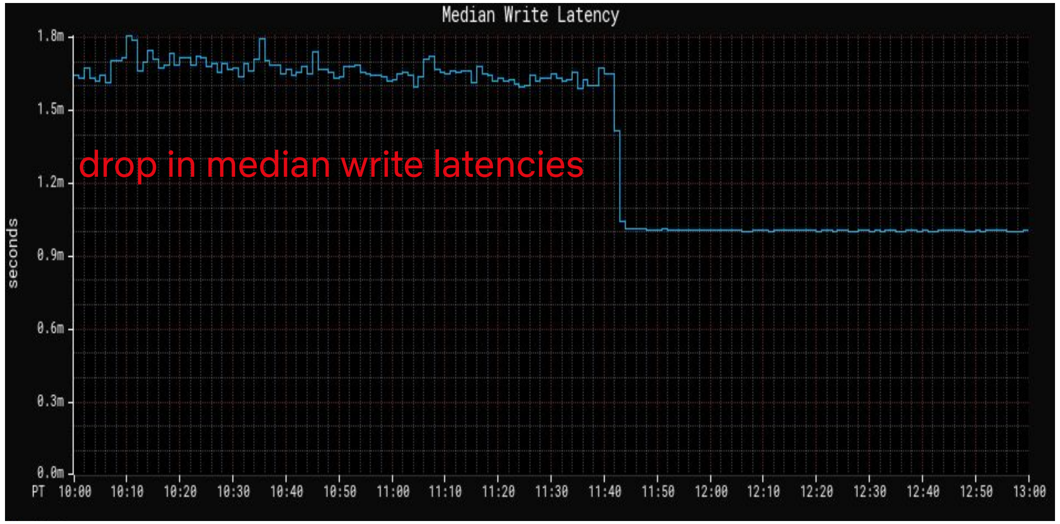
Token Aware?

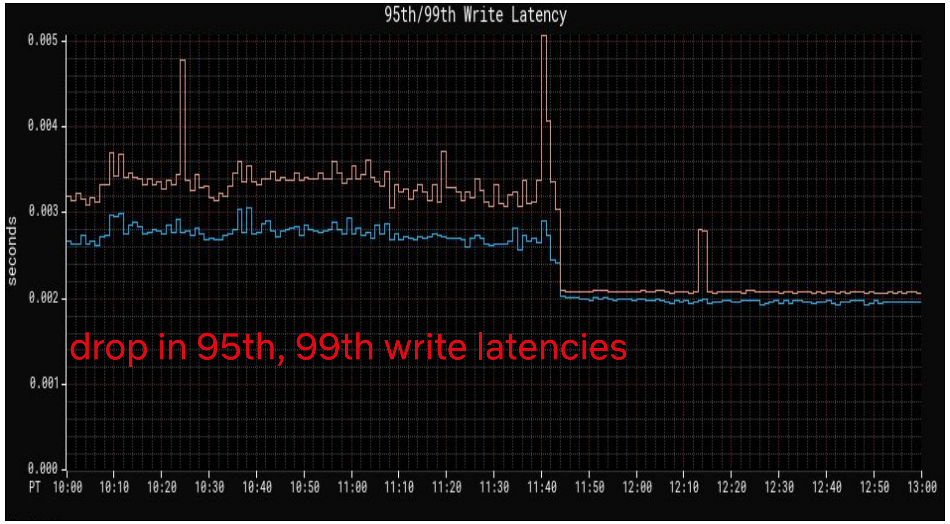
Hash key, shuffle replicas, return least loaded between first and second.

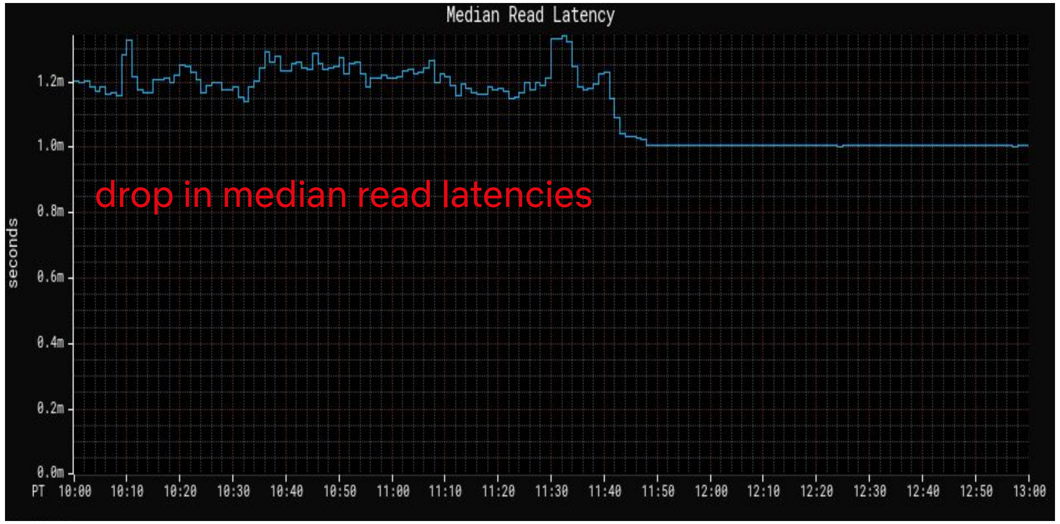
Avoids very slow replicas!

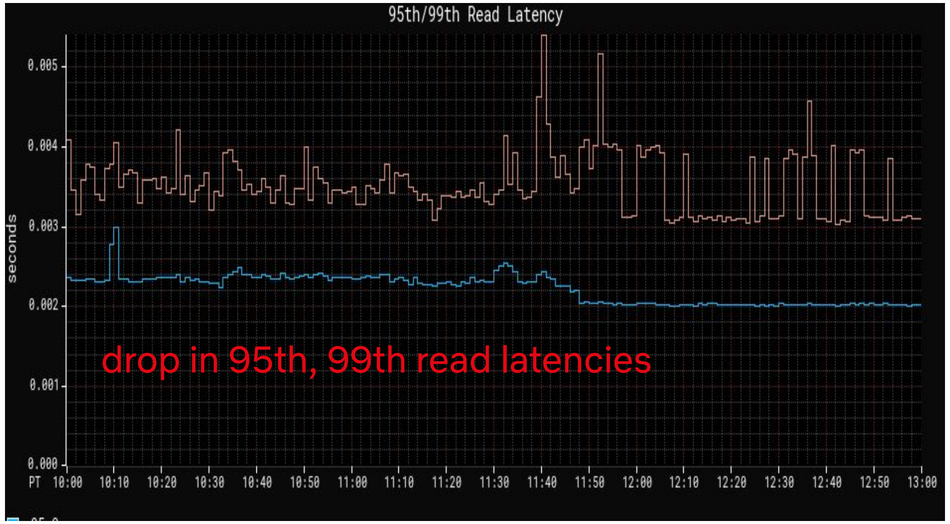
Basically choice of 2 over random subsets! Nice!

DataStax Java Driver 4.x for Apache Cassandra®









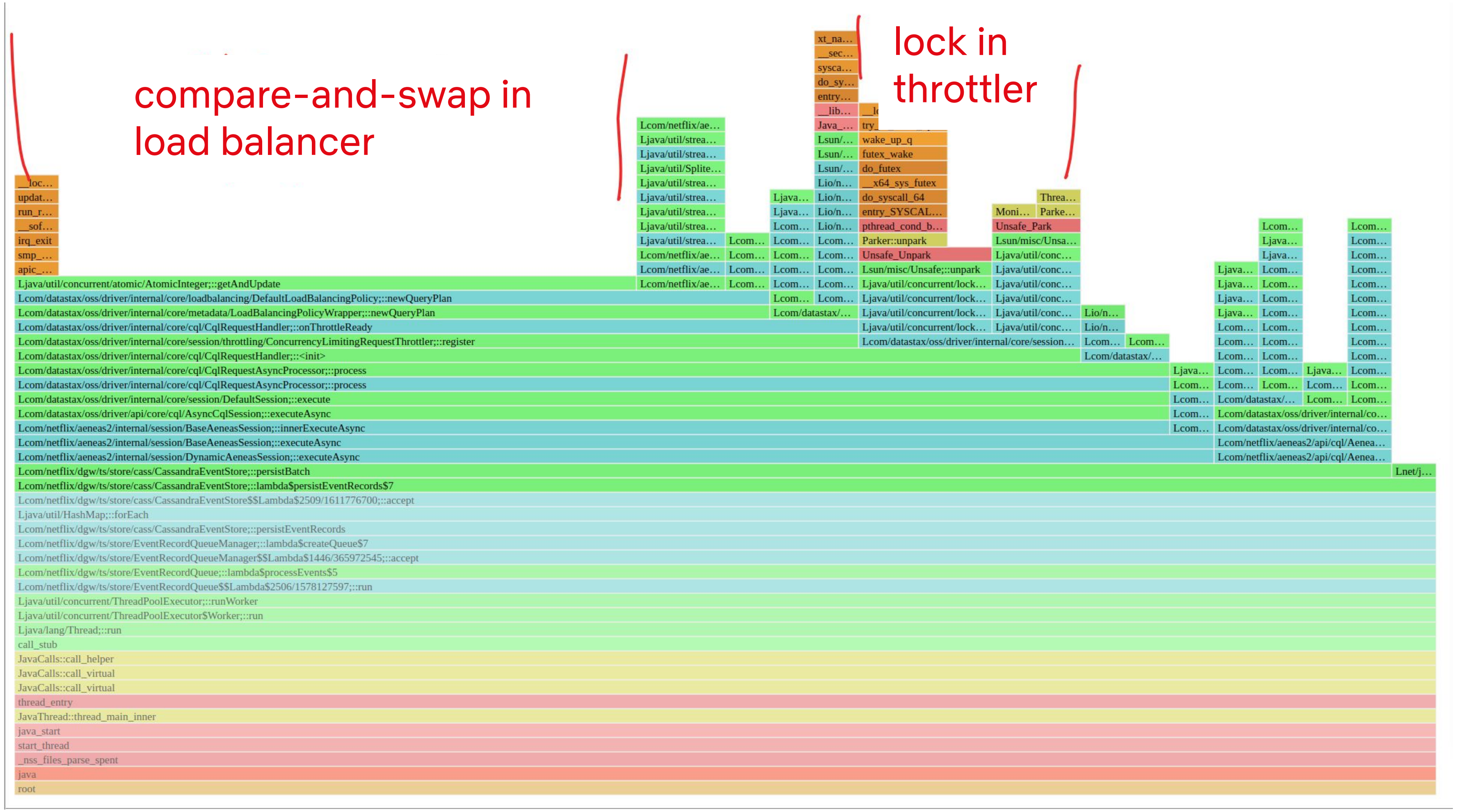
DataStax Java Driver 4.x for Apache Cassandra®

Perf regression with **high-throughput** cases

We needed to do 20k QPS per client to Cassandra and Datastax 4.x could barely do 8k.

DataStax Java Driver 4.x for Apache Cassandra®

Pays expensive **compare and update** and a **lock acquire-release**



DataStax Java Driver 4.x for Apache Cassandra®

Pays expensive **compare and update** and a **lock acquire-release**

```
LOG.trace("[{}] Prioritizing {} local replicas", logPrefix, replicaCount);

// Round-robin the remaining nodes
ArrayUtils.rotate(
    currentNodes,
    replicaCount,
    length: currentNodes.length - replicaCount,
    roundRobinAmount.getAndUpdate(INCREMENT));

QueryPlan plan = currentNodes.length == 0 ? QueryPlan.EMPTY : new SimpleQueryPlan(currentNodes);
return maybeAddDcFailover(request, plan);
}
```

DefaultLoadBalancingPolicy#newQueryPlan

Weighted Least Loaded

Started with fixing compare-and-swap, ended up rewriting the algorithm

No Token?

Chose 8 random nodes

Token Aware?

Choose all RF replicas and 8-RF random

Weight concurrency by:

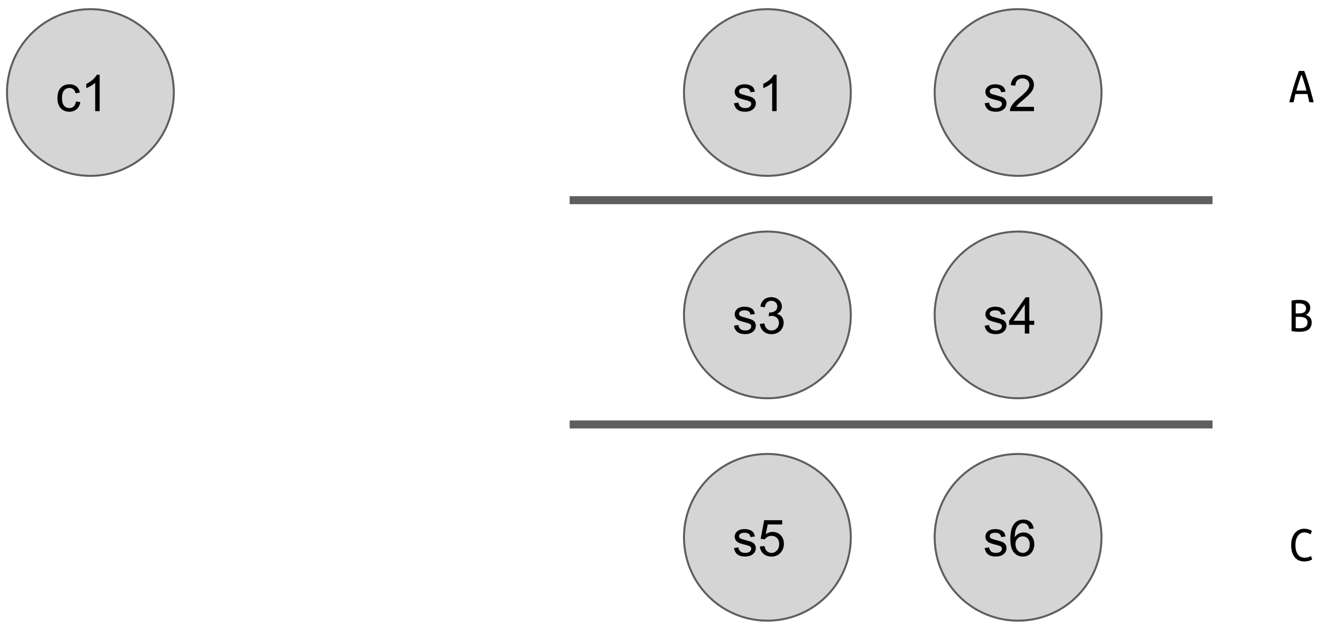
!Rack = 4

!Replica = 12

Unhealthy = 64

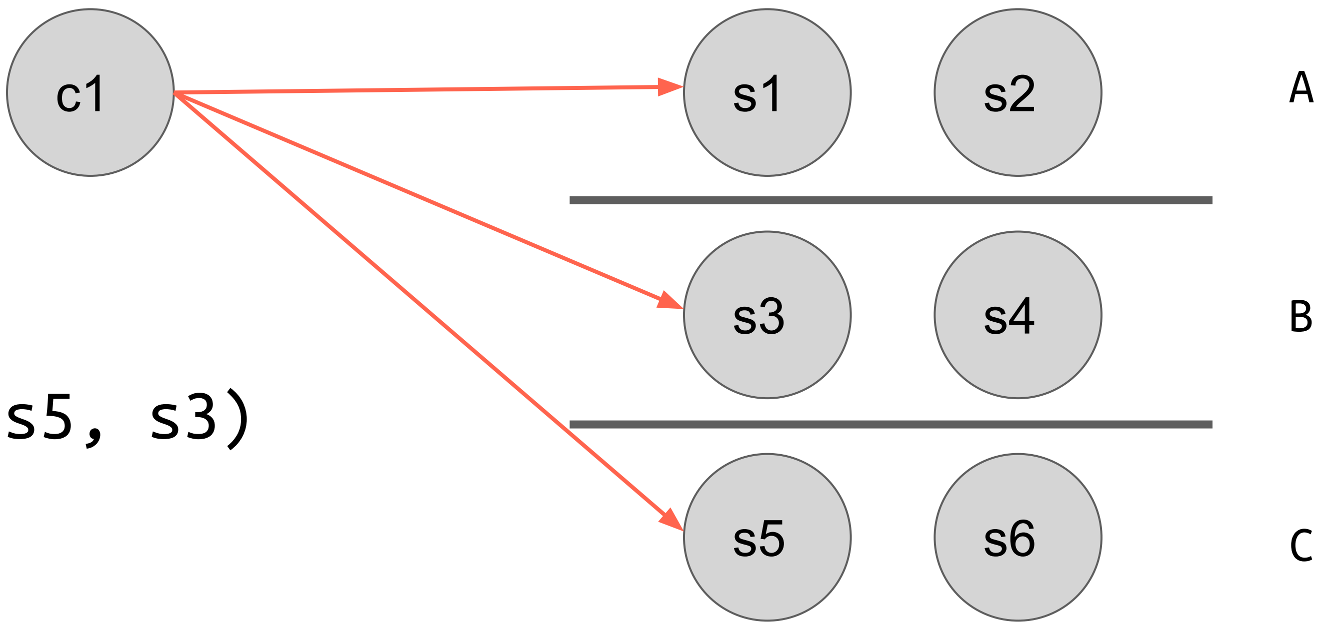
Sort the sublist. Done!

Stateful load balancing with real networks



	A	B	C
A	**150us**	800us	250us
B	800us	**220us**	850us
C	380us	700us	**160us**

LOCAL_ONE (Control)

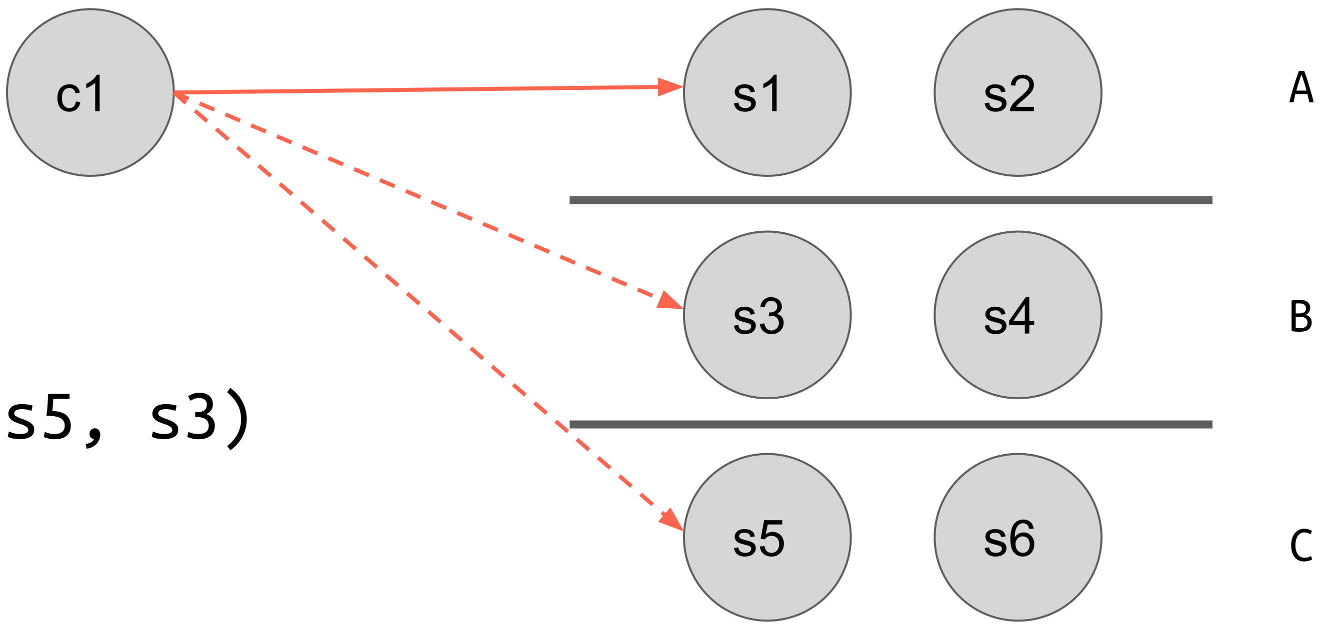


set(x=0)
replicas(x) = (s1, s5, s3)

End to End Latency = Latency (L) + Processing (R)

E_LO = ⅓ (L(A, A) + R) + ⅓ (L(A, B) + R) + ⅓ (L(A, C) + R)
Let R = 100us
E_LO = ⅓ (150 + 100) + ⅓ (800 + 100) + ⅓ (250 + 100) = **500us**

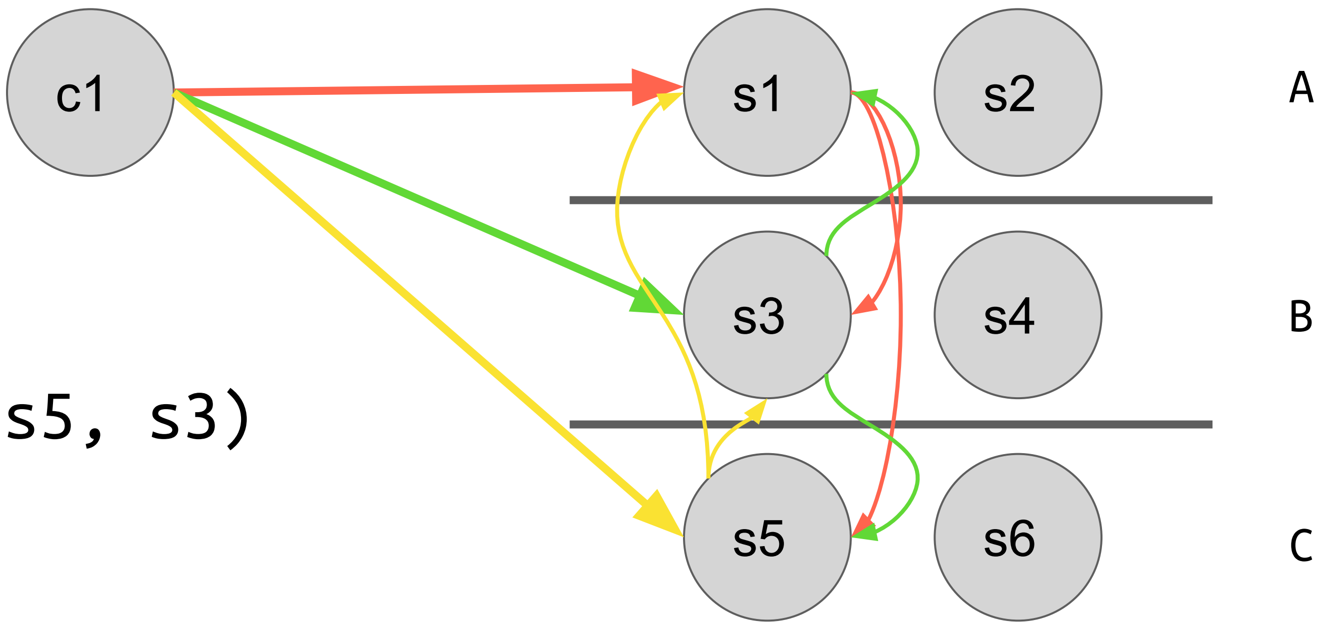
LOCAL_ONE (WLLLB)




```
set(x=0)
replicas(x) = (s1, s5, s3)
```

```
End to End Latency = Latency (L) + Processing (R)

E_LO = L(A, A) + R
Let R = 100us
E_LO = 150 + 100 = 250us (50% reduction)
```

LOCAL_QUORUM (Control)

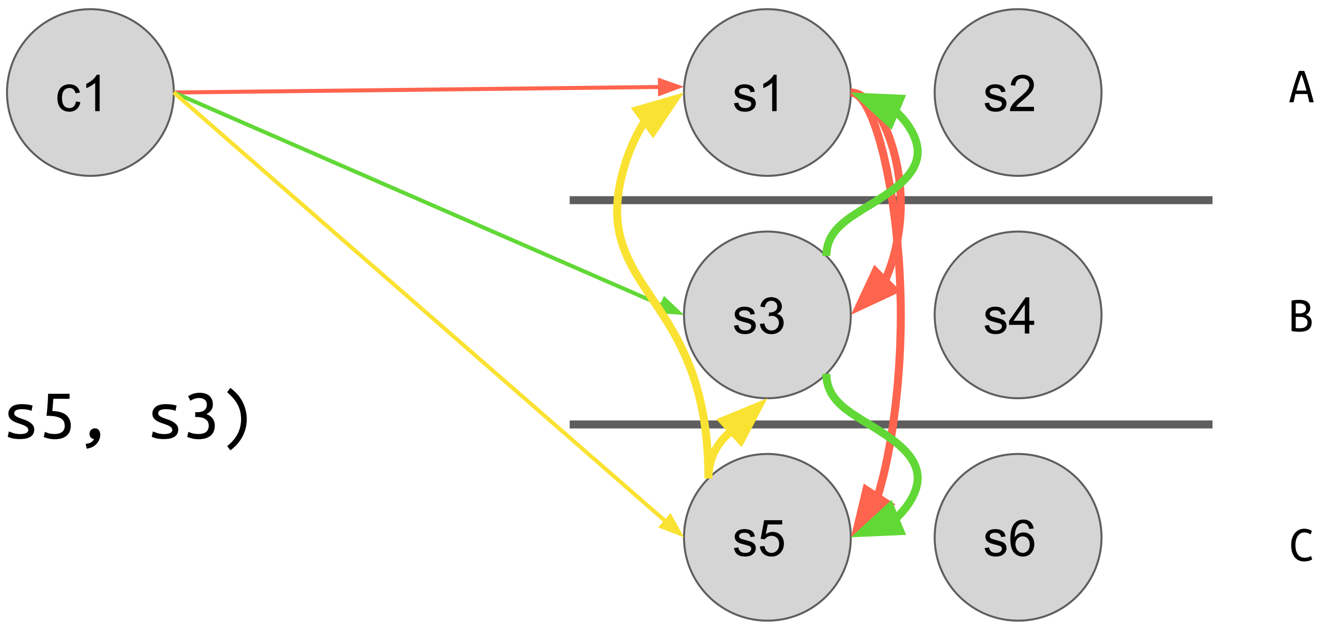



```
set(x=0)
replicas(x) = (s1, s5, s3)
```

```
E_LQ = ⅓ (L(A, A) + min(R, L(A, C) + R))
       ⅓ (L(A, B) + min(R, L(B, A) + R))
       ⅓ (L(A, C) + min(R, L(C, A) + R))
```

```
Let R = 100us
E_LQ = ⅓ (150 + 350) + ⅓ (800 + 900) + ⅓ (250 + 480) = 980us
```

LOCAL_QUORUM (Control)

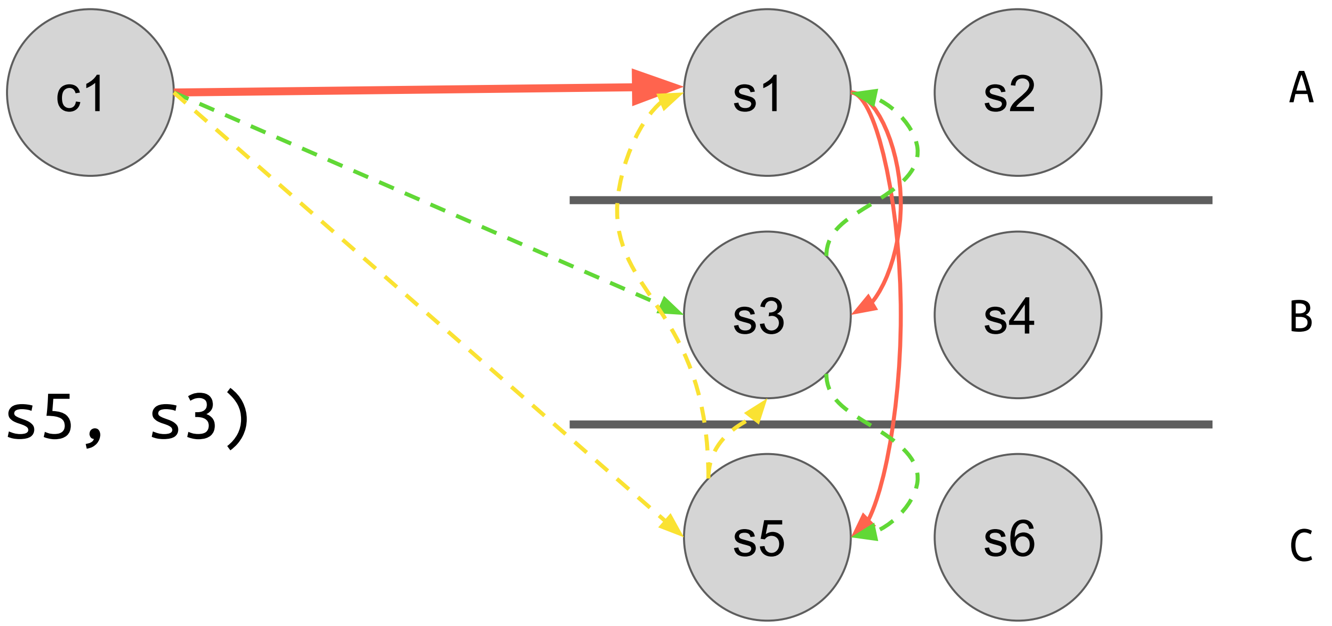



```
set(x=0)
replicas(x) = (s1, s5, s3)
```

```
E_LQ = ⅓ (L(A, A) + min(R, L(A, C) + R))
       ⅓ (L(A, B) + min(R, L(B, A) + R))
       ⅓ (L(A, C) + min(R, L(C, A) + R))
```

```
Let R = 100us
E_LQ = ⅓ (150 + 350) + ⅓ (800 + 900) + ⅓ (250 + 480) = 980us
```

LOCAL_QUORUM (WLLLB)



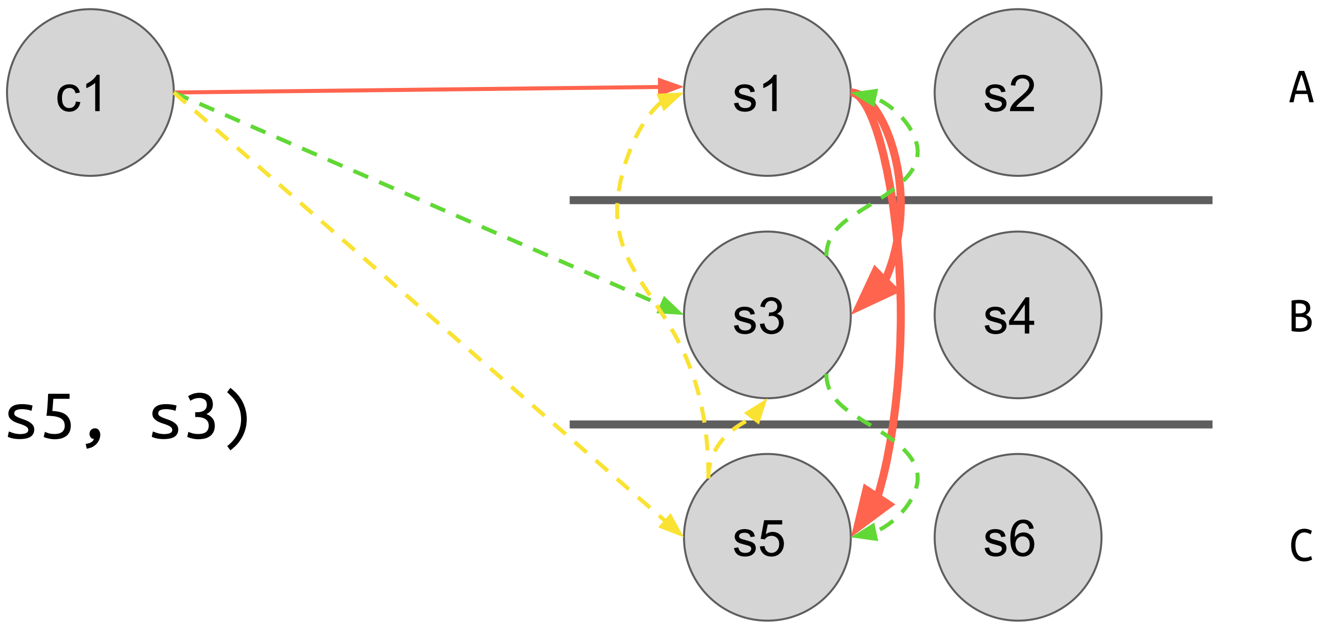

```
set(x=0)
replicas(x) = (s1, s5, s3)
```

E_LQ = **L(A, A)** + min(R, L(A, C) + R)

Let R = 100us

E_LQ = 150 + 100 + 250 = 500us **(50% reduction)**

LOCAL_QUORUM (WLLLB)



set(x=0)
replicas(x) = (s1, s5, s3)

E_LQ = L(A, A) + **min(R, L(A, C) + R)**

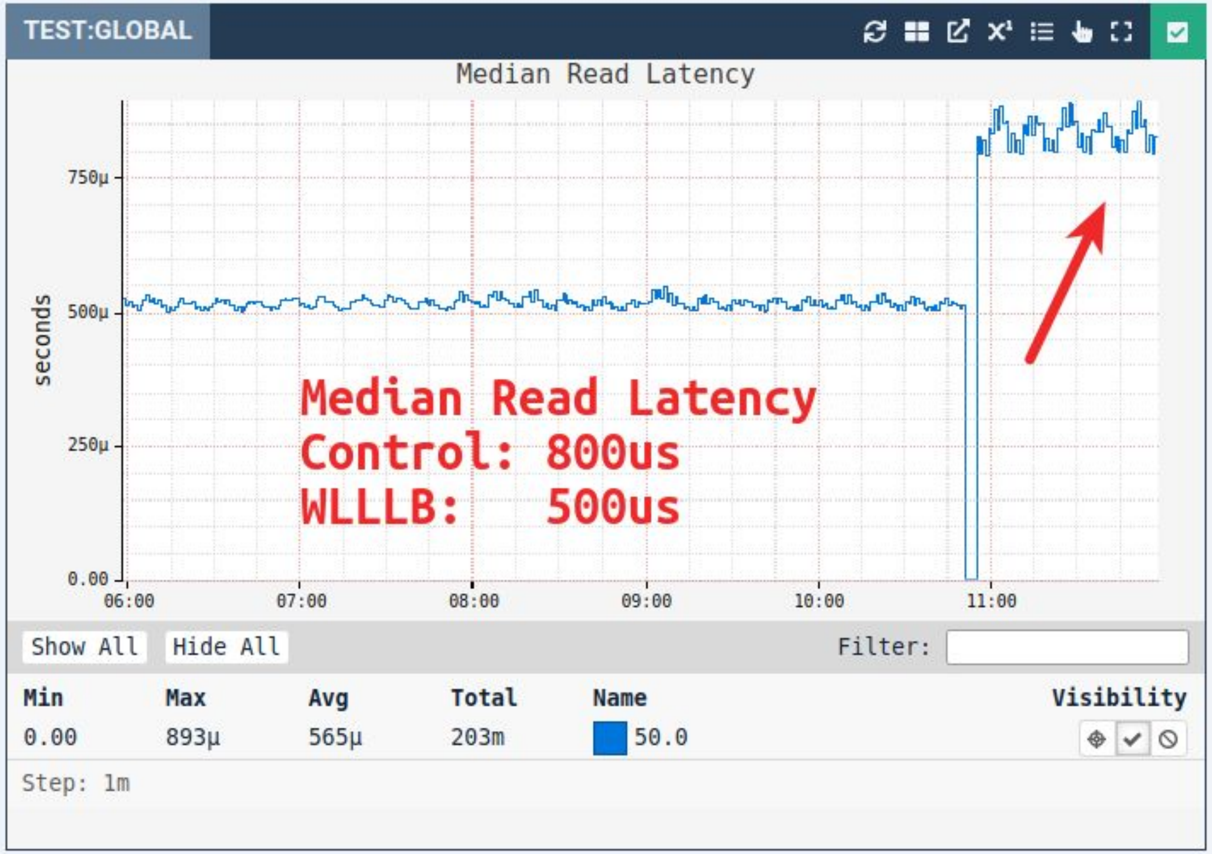
Let R = 100us
E_LQ = 150 + 100 + 250 = 500us **(50% reduction)**

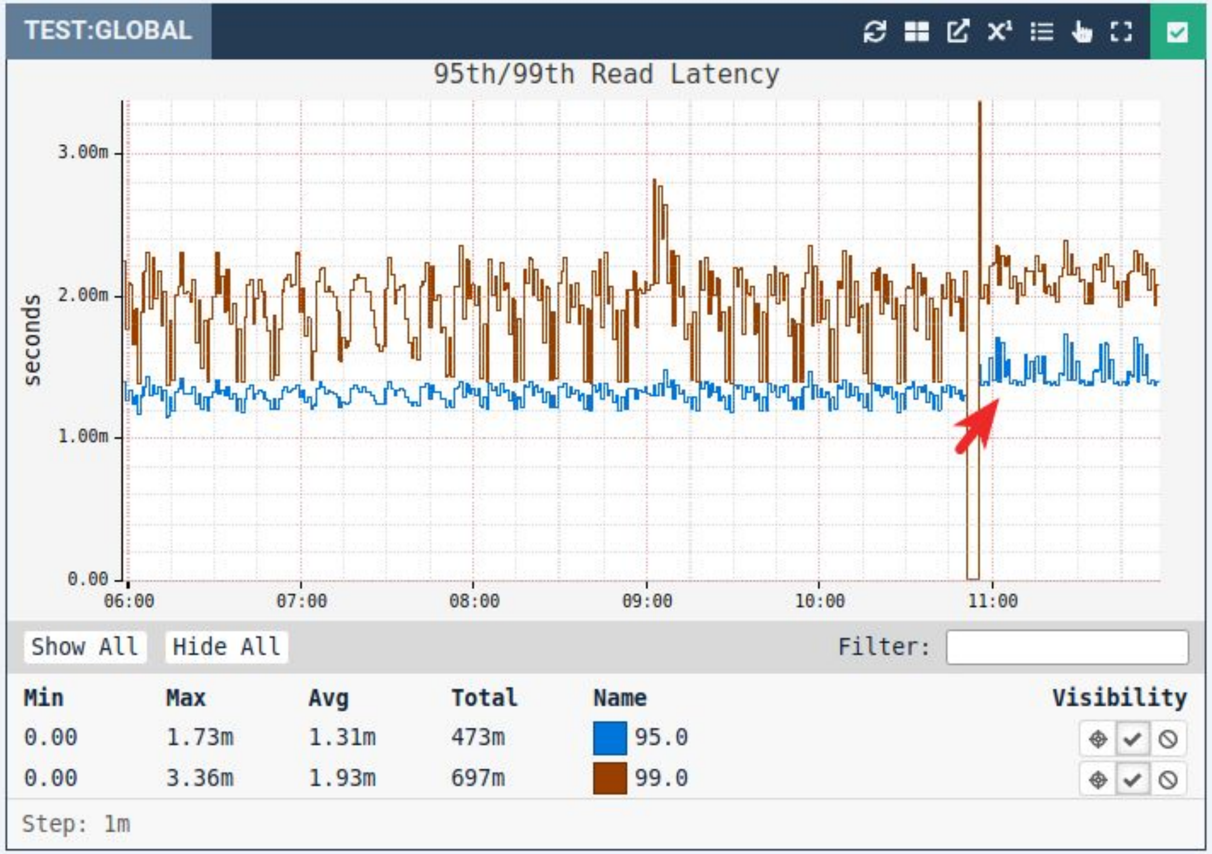
Experiments

Synthetic Traffic

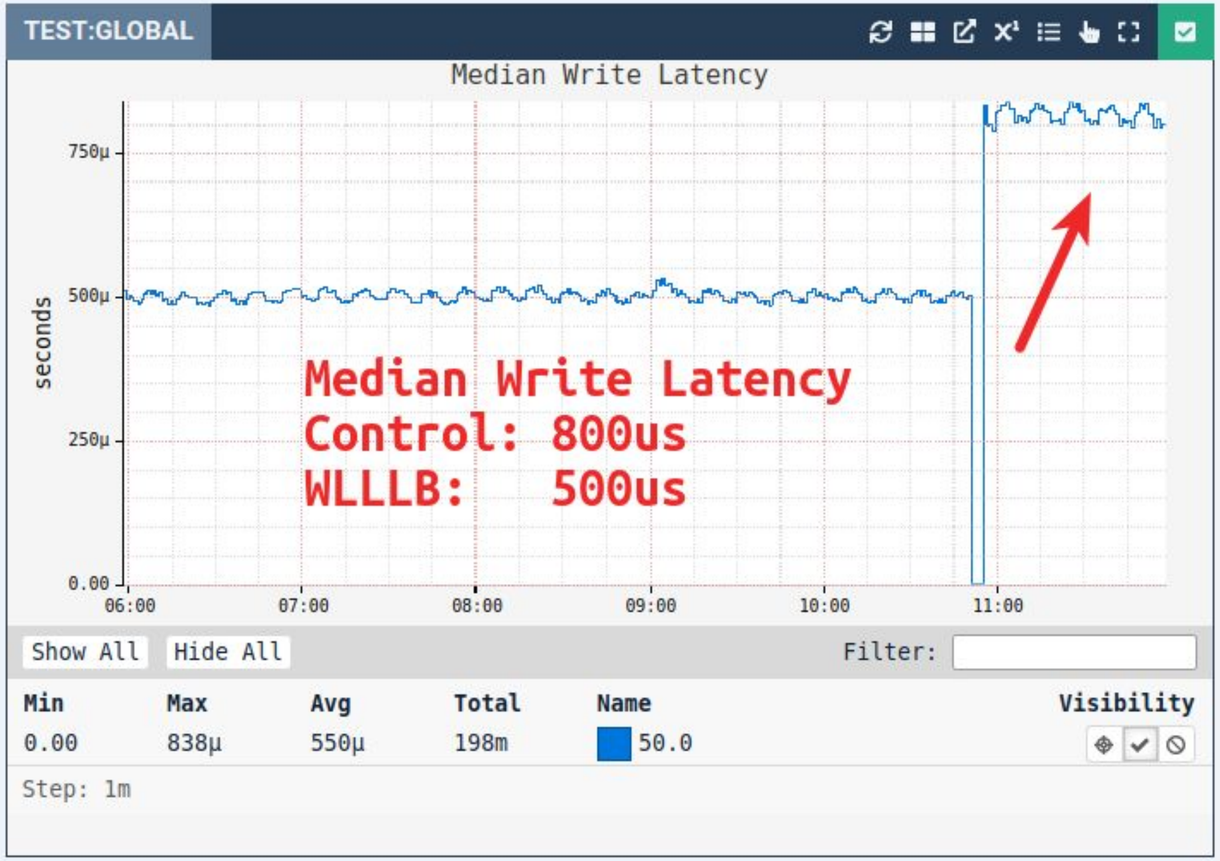
Apply Load
Measure Results

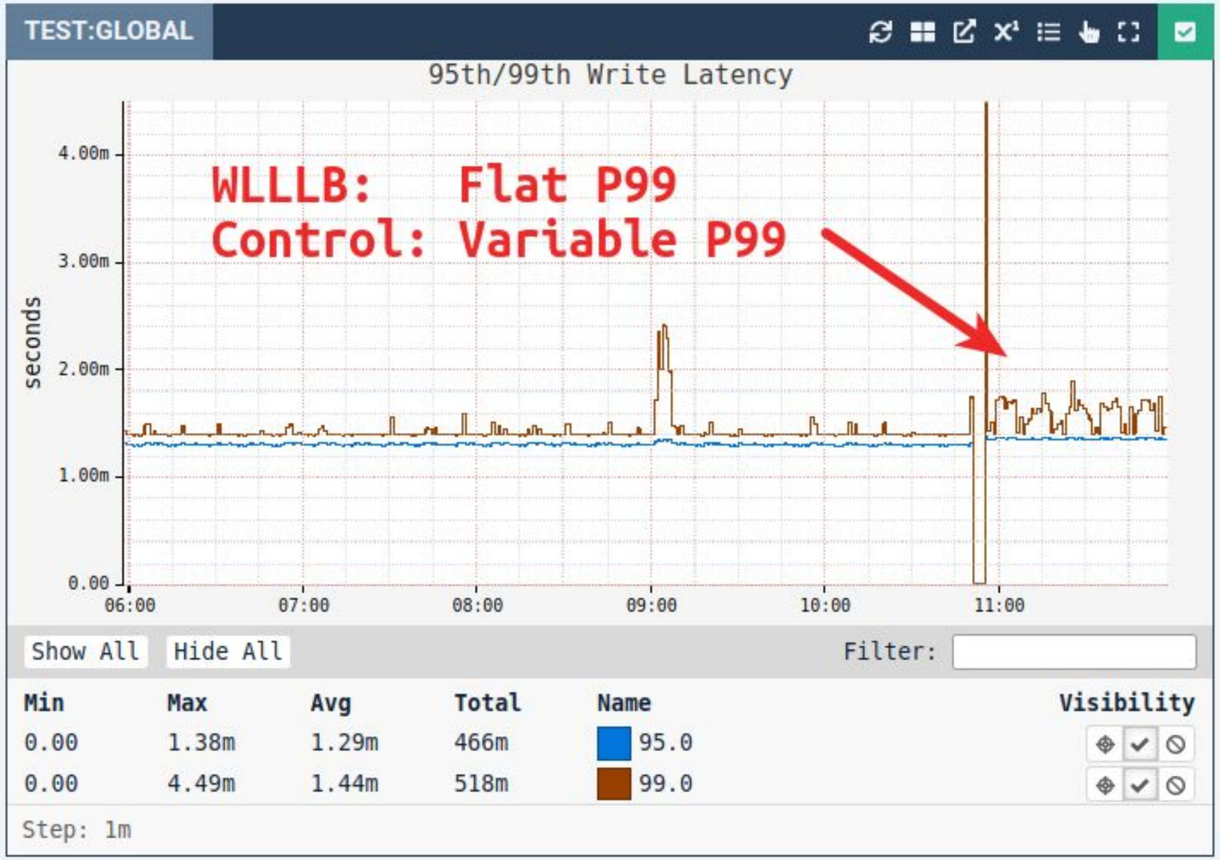
Latency results
LOCAL_ONE





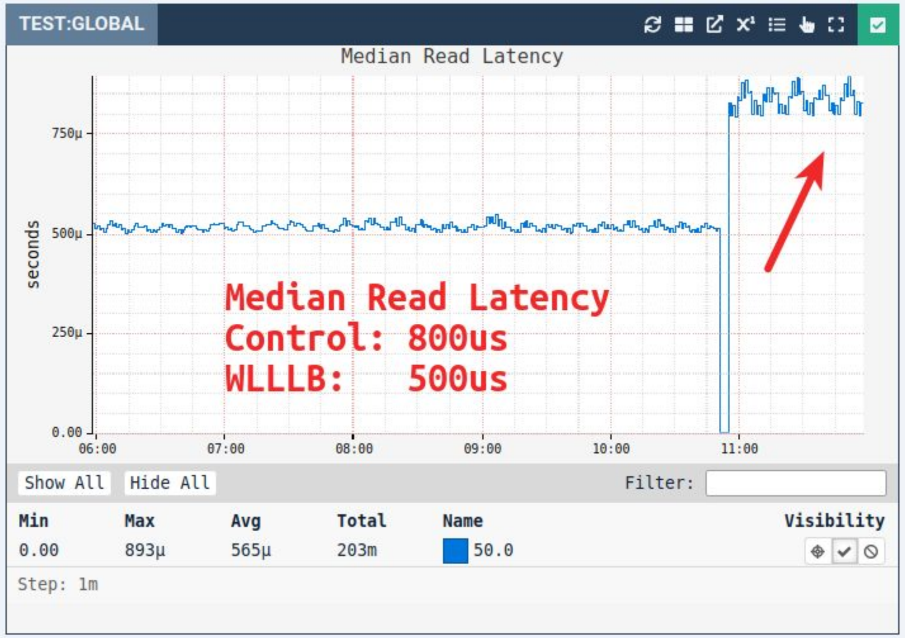
Latency results
LOCAL_ONE

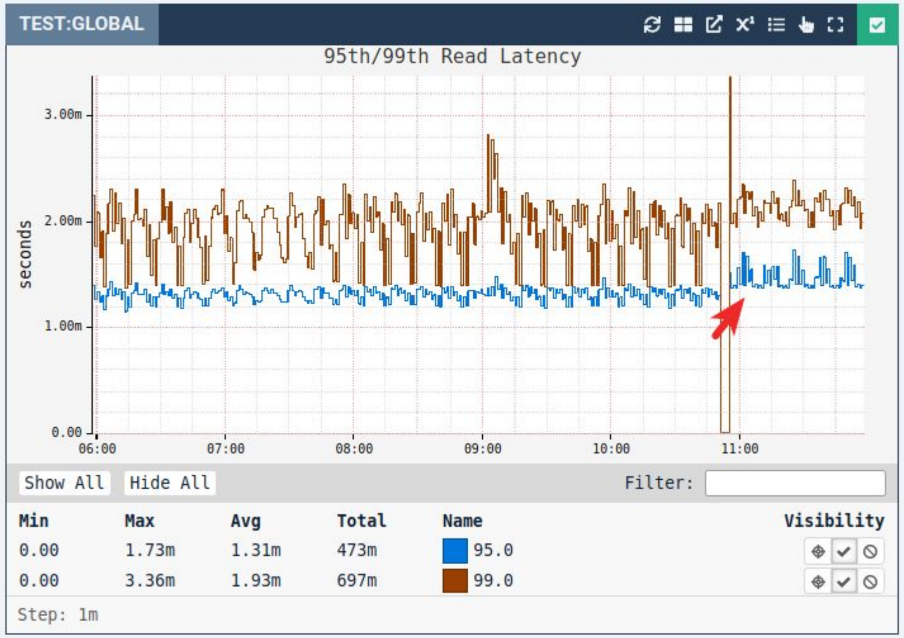


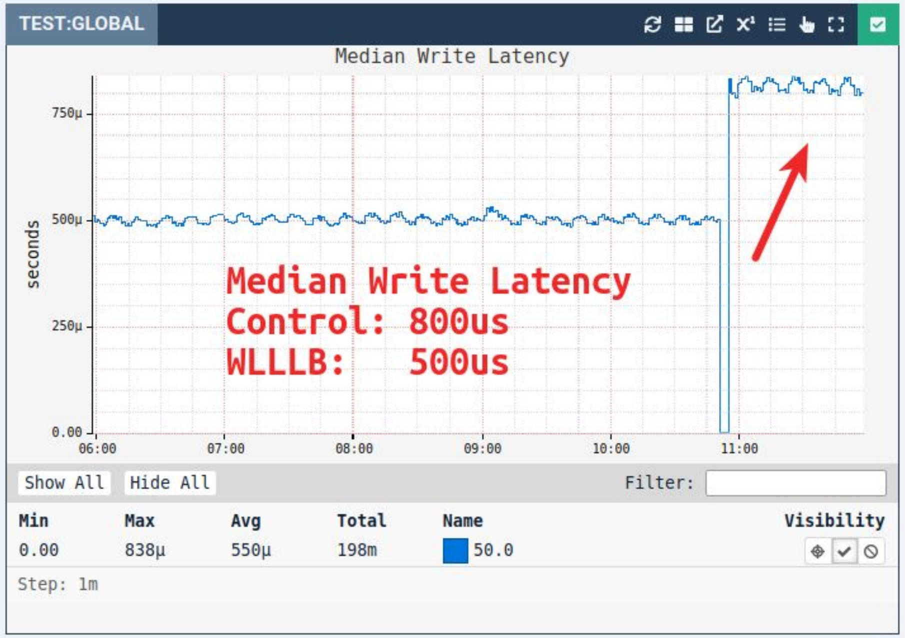


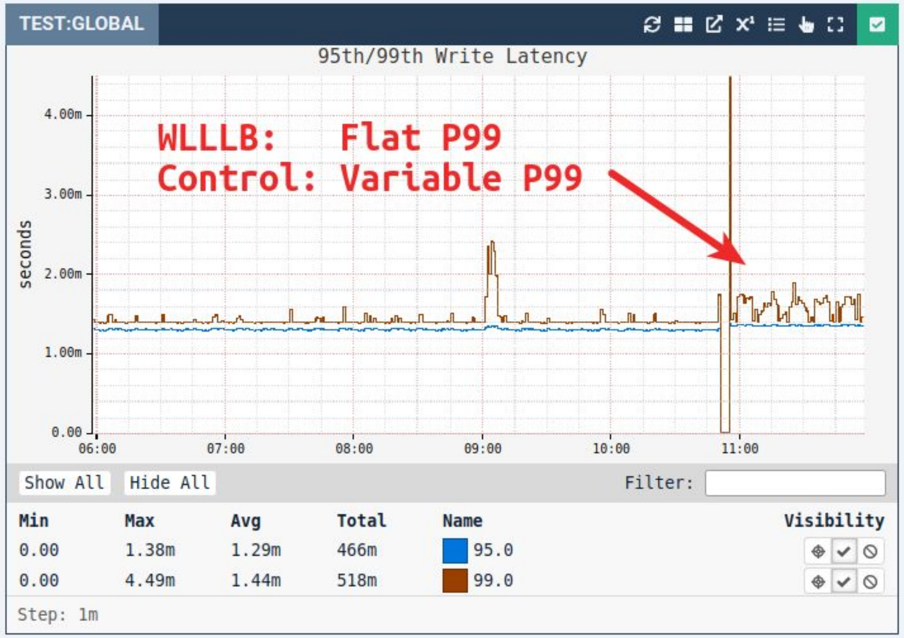
Latency results LOCAL_ONE

About a 40% improvement

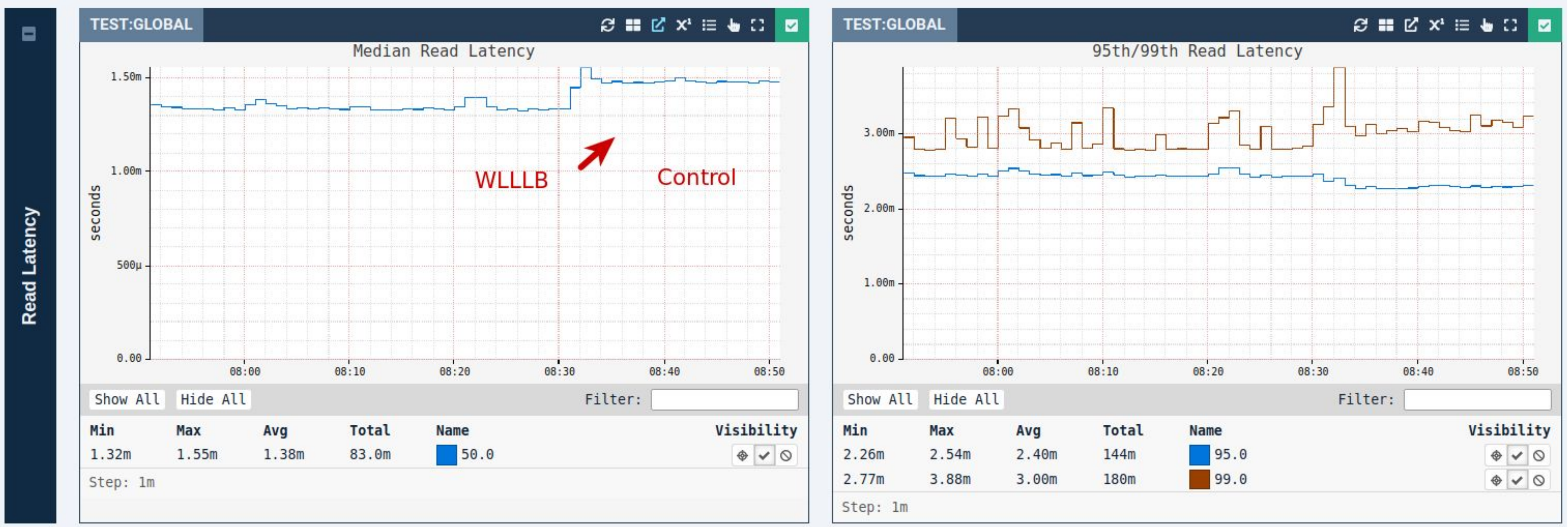




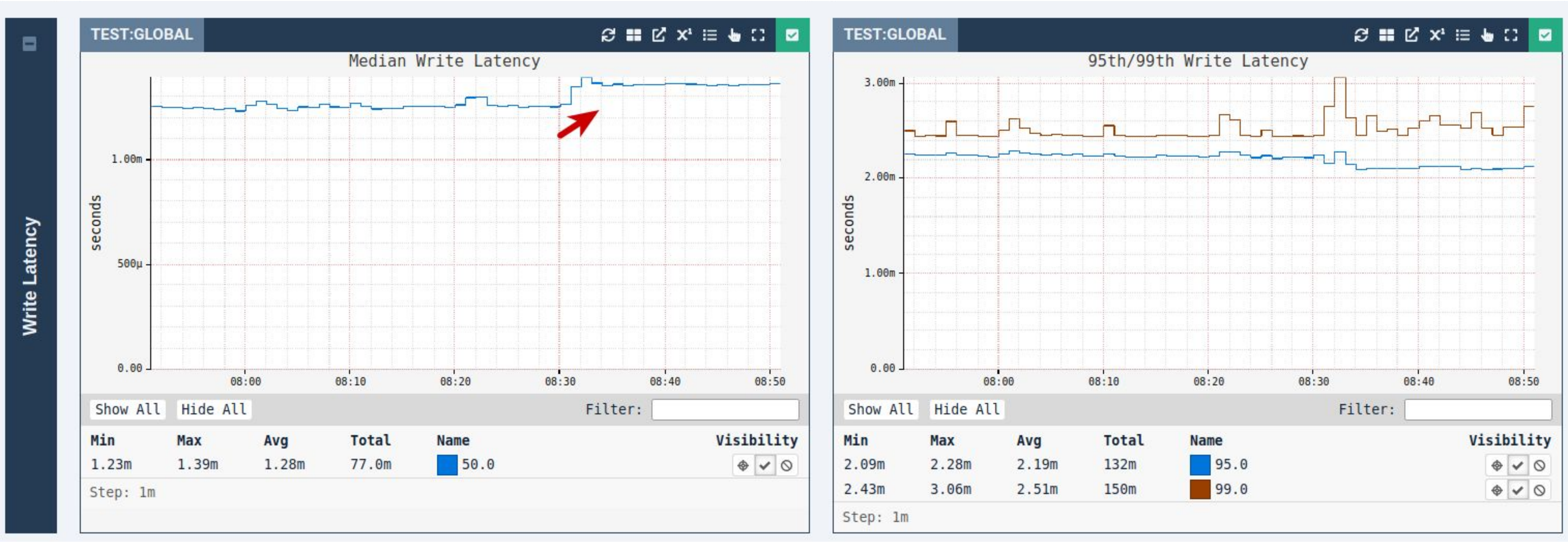




Latency results
LOCAL_QUORUM

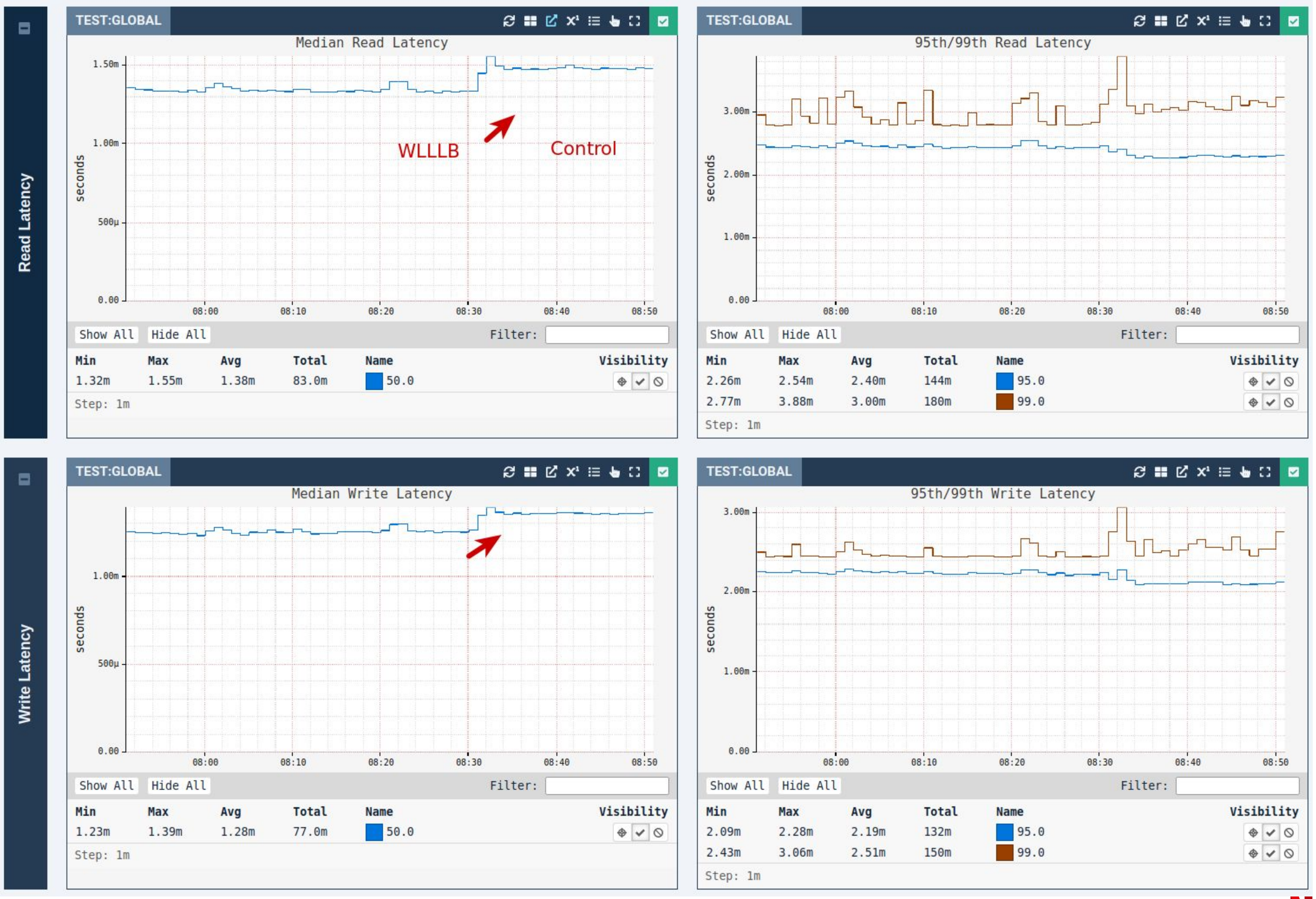


Latency results
LOCAL_QUORUM



Latency results LOCAL_QUORUM

About a 10% improvement



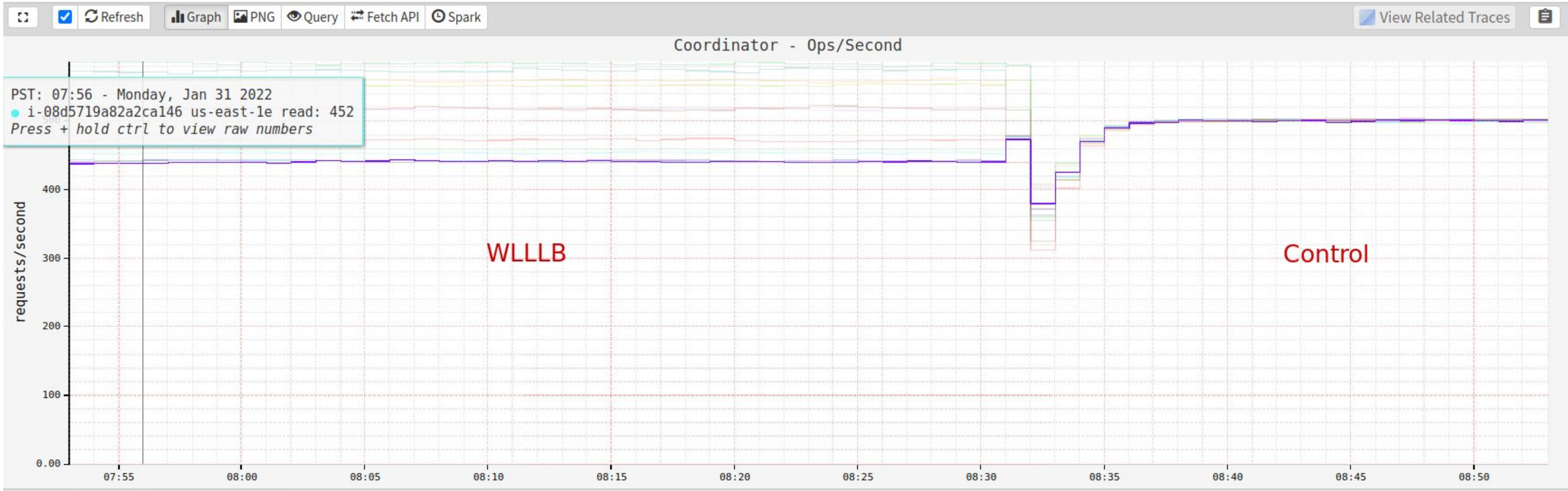
Latency results

	WLLLB P50/P95/P99 Read (ms)	WLLLB P50/P95/P99 Write (ms)	Control P50/P95/P99 Read (ms)	Control P50/P95/P99 Write (ms)	Read Latency Difference	Write Latency Difference
LO-1	0.52/1.30/1.92	0.50/1.30/1.41	0.84/1.45/2.14	0.82/1.35/1.59	**38%**/10%/10%	**39%**/4%/11%
LQ-1	1.33/2.42/2.90	1.21/2.15/2.45	1.52/2.25/3.07	1.36/2.06/2.48	**12.5/**-7.5%/5.6%	**11%/**-4.3%/1.2%
LQ-2	1.40/2.56/4.45	1.27/2.08/2.46	1.55/2.32/3.93	1.32/2.03/2.47	**10%**/-10%/-13%	**4%**/-5%/-1%

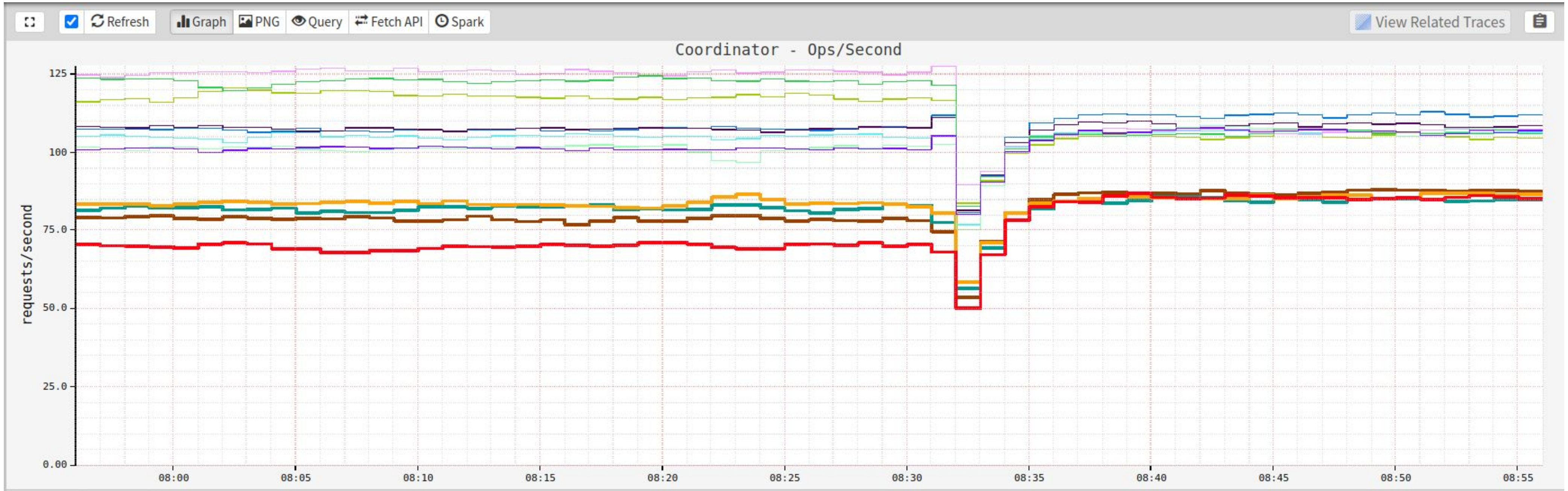
Why the slight P95 regression in LQ? Theories:

1. Load Imbalance due to asymmetric latency
2. Dynamic Endpoint Snitch

Load imbalance
Reads



Load imbalance
Writes



Hide All | Show All | Show Filtered Only | Step: 1m | Filter:

Min	Max	Average ▼	Total	Name	Visibility
89.33	127.365	117.154	7.146k	i-09eb1027f00c8d4cb us-east-1c write	
82.488	124.24	115.029	7.017k	i-0dc167503aeb65b01 us-east-1e write	
83.454	120.442	111.713	6.815k	i-058edfd49acf20acf us-east-1e write	
80.464	112.734	108.054	6.591k	i-0020298335a9fa171 us-east-1c write	
81.224	110.829	107.071	6.531k	i-05b3b14eaba6cf90b us-east-1c write	
76.579	106.785	104.528	6.376k	i-08d5719a82a2ca146 us-east-1e write	
80.00	107.598	102.404	6.247k	i-0df3aa8a13c5ebc59 us-east-1c write	
75.05	108.346	102.354	6.244k	i-0c331c8d41436b4f6 us-east-1e write	
58.215	86.822	83.451	5.091k	i-066a4368fa876dcc8 us-east-1d write	
56.153	85.604	81.991	5.001k	i-05ef379148c188e77 us-east-1d write	
53.365	87.827	80.92	4.936k	i-003ee66a243bfe478 us-east-1d write	
49.812	86.641	74.946	4.572k	i-0dae4a6623071ef95 us-east-1d write	

Network Delay

Force packet delay

Measure results

Linux Traffic Control (tc)!

```
$ sudo tc qdisc show dev eth0
qdisc mq 8005: root
qdisc fq 0: parent 8005:4 limit 10000p flow_limit 100p buckets 1024 orphan_mask 1023 quantum 18030
initial_quantum 90150 low_rate_threshold 550Kbit refill_delay 40.0ms
qdisc fq 0: parent 8005:3 limit 10000p flow_limit 100p buckets 1024 orphan_mask 1023 quantum 18030
initial_quantum 90150 low_rate_threshold 550Kbit refill_delay 40.0ms
qdisc fq 0: parent 8005:2 limit 10000p flow_limit 100p buckets 1024 orphan_mask 1023 quantum 18030
initial_quantum 90150 low_rate_threshold 550Kbit refill_delay 40.0ms
qdisc fq 0: parent 8005:1 limit 10000p flow_limit 100p buckets 1024 orphan_mask 1023 quantum 18030
initial_quantum 90150 low_rate_threshold 550Kbit refill_delay 40.0ms
```

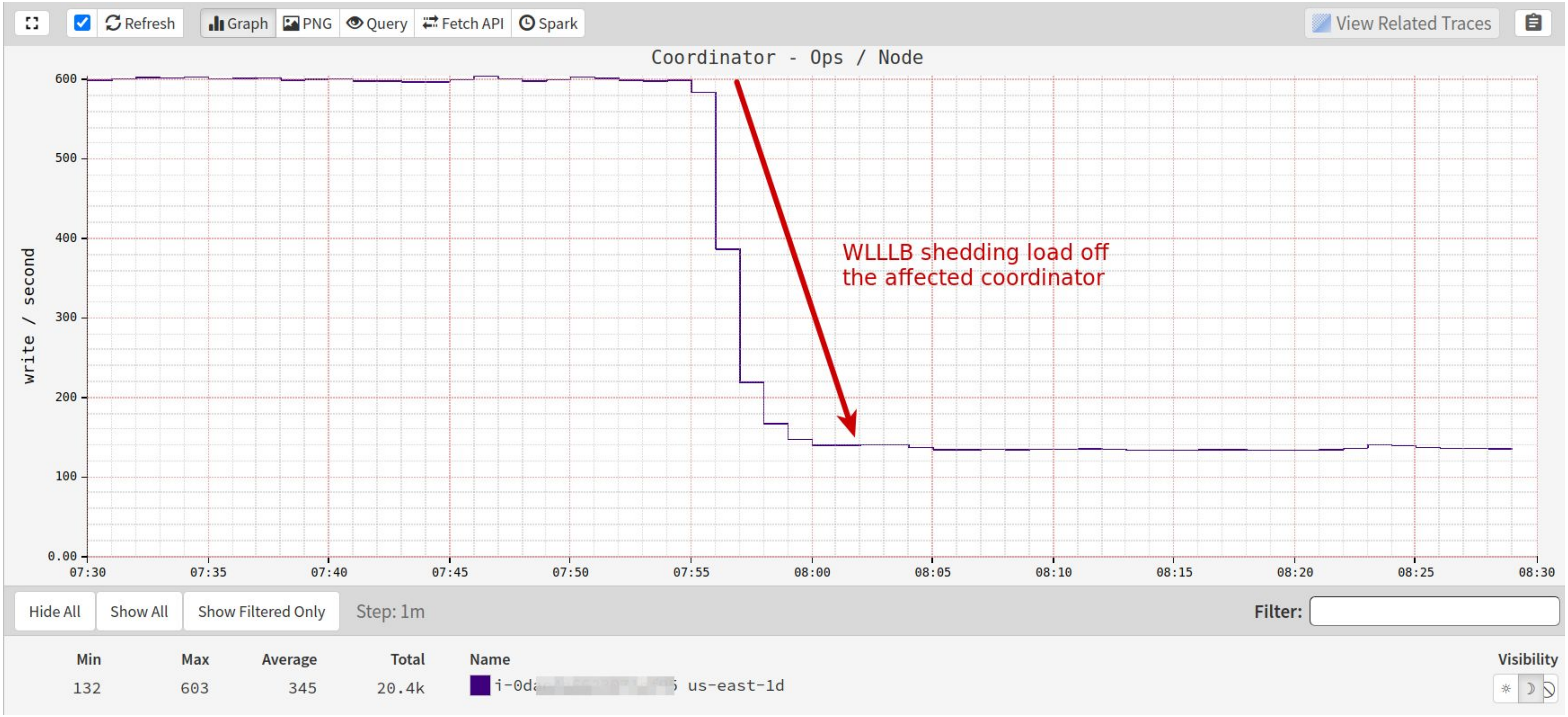
Netem to the rescue (tc-netem)

```
# Server adds 10ms delay
server$ sudo tc qdisc replace dev eth0 root netem delay 10ms

# Client now observes 10ms additional latency on all requests
client$ ping 100...
…
64 bytes from 100...: icmp_seq=525 ttl=64 time=0.215 ms
64 bytes from 100...: icmp_seq=526 ttl=64 time=0.212 ms
# When netem was enabled
64 bytes from 100...: icmp_seq=527 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=528 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=529 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=530 ttl=64 time=10.2 ms

# Now Revert on server
server$ sudo tc qdisc replace dev eth0 root mq
```

Netem to the rescue (tc-netem)

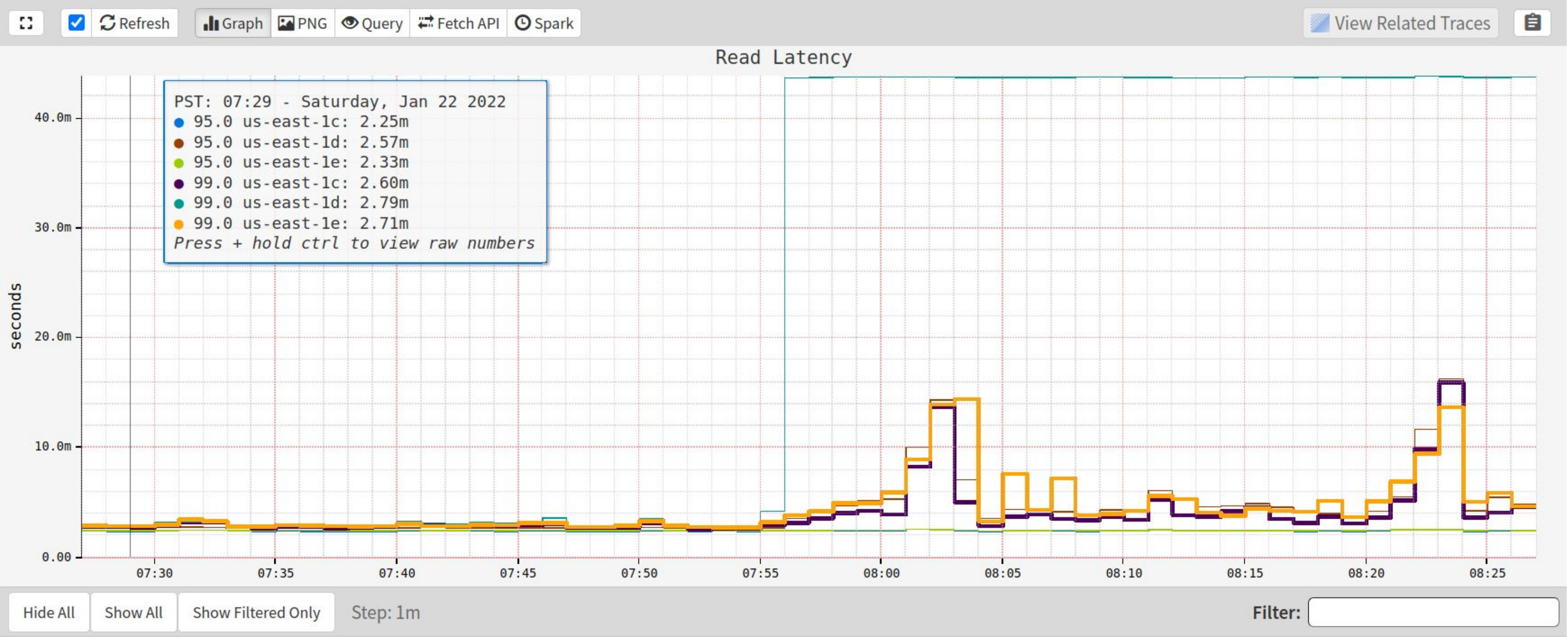
```
# You can also use netem to simulate packet loss, corruption,
duplication, reordering and other TCP issues.
# For example you could add a distribution of delay with

$ tc qdisc change dev eth0 root netem delay 10ms 4ms distribution normal
```

Slow coordinators



Slow coordinators

Limited latency impact in ⅔ zones

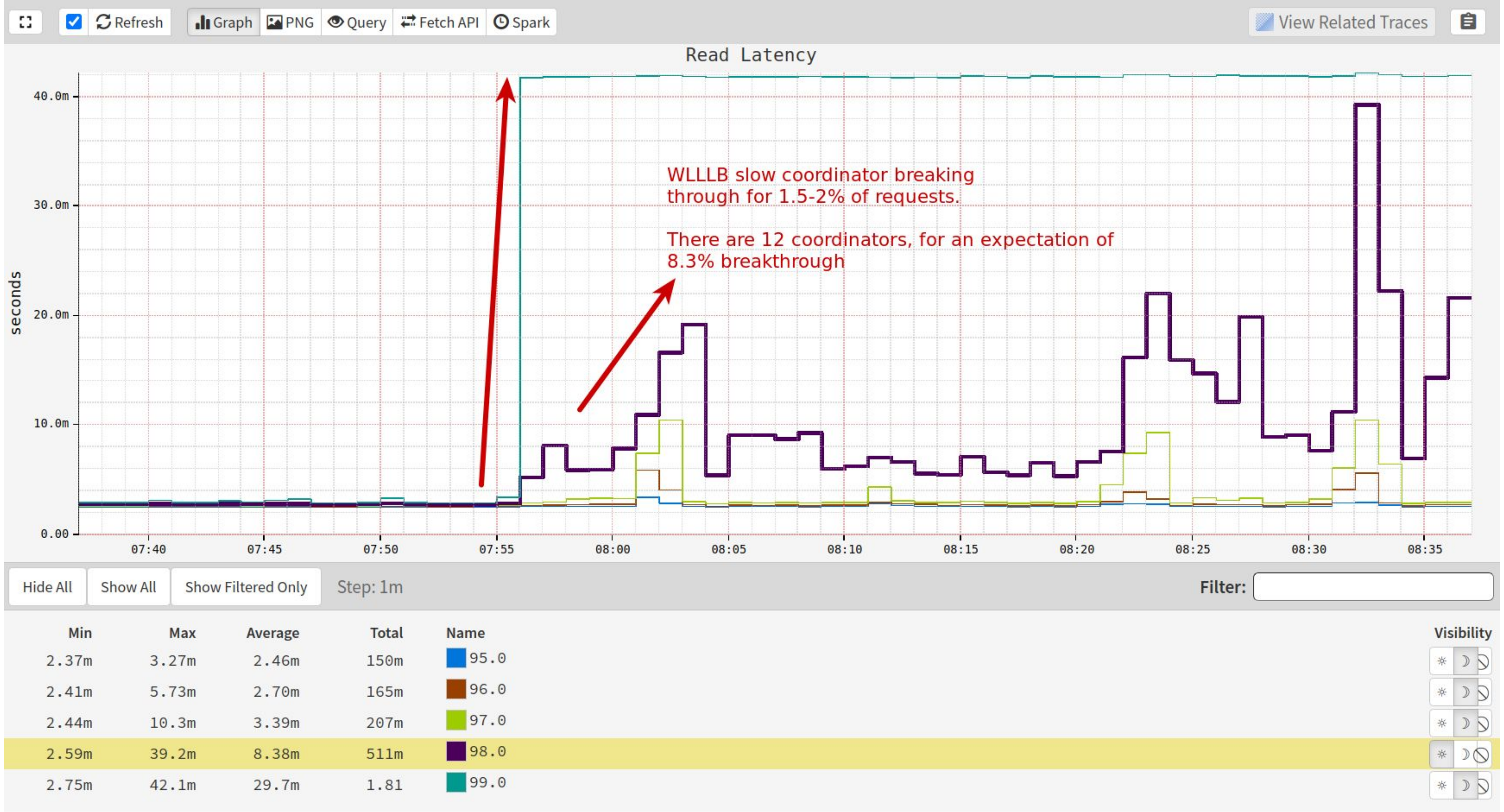


Min	Max ▼	Average	Total	Name	Visibility
2.78m	43.7m	24.3m	1.48	99.0 us-east-1d	
2.53m	16.1m	4.14m	253m	95.0 us-east-1d	
2.45m	15.8m	3.80m	232m	99.0 us-east-1c	
2.60m	14.3m	4.42m	269m	99.0 us-east-1e	
2.30m	2.44m	2.34m	143m	95.0 us-east-1e	
2.24m	2.43m	2.28m	139m	95.0 us-east-1c	

Slow coordinators

1/12 = 8.3% should have been affected

But only 1.5% were



Garbage Collection

Simulate pauses

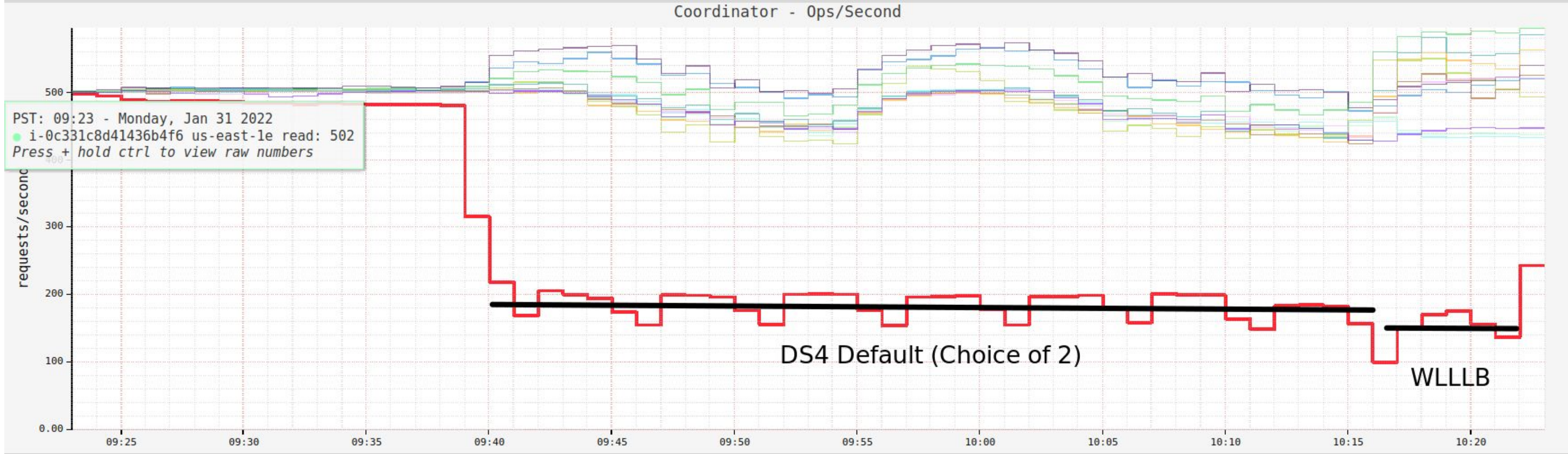
Measure results

STOP + CONT

```
# pause.sh
while [ 1 ]
do
sudo -u www-data kill -STOP $(pgrep -f CassandraDaemon)
# Duration of pause
sleep 20
sudo -u www-data kill -CONT $(pgrep -f CassandraDaemon)
# Interval between pauses
sleep 30
done
```

Slow coordinators

Simulate "GC" pause via stopping the Java process.



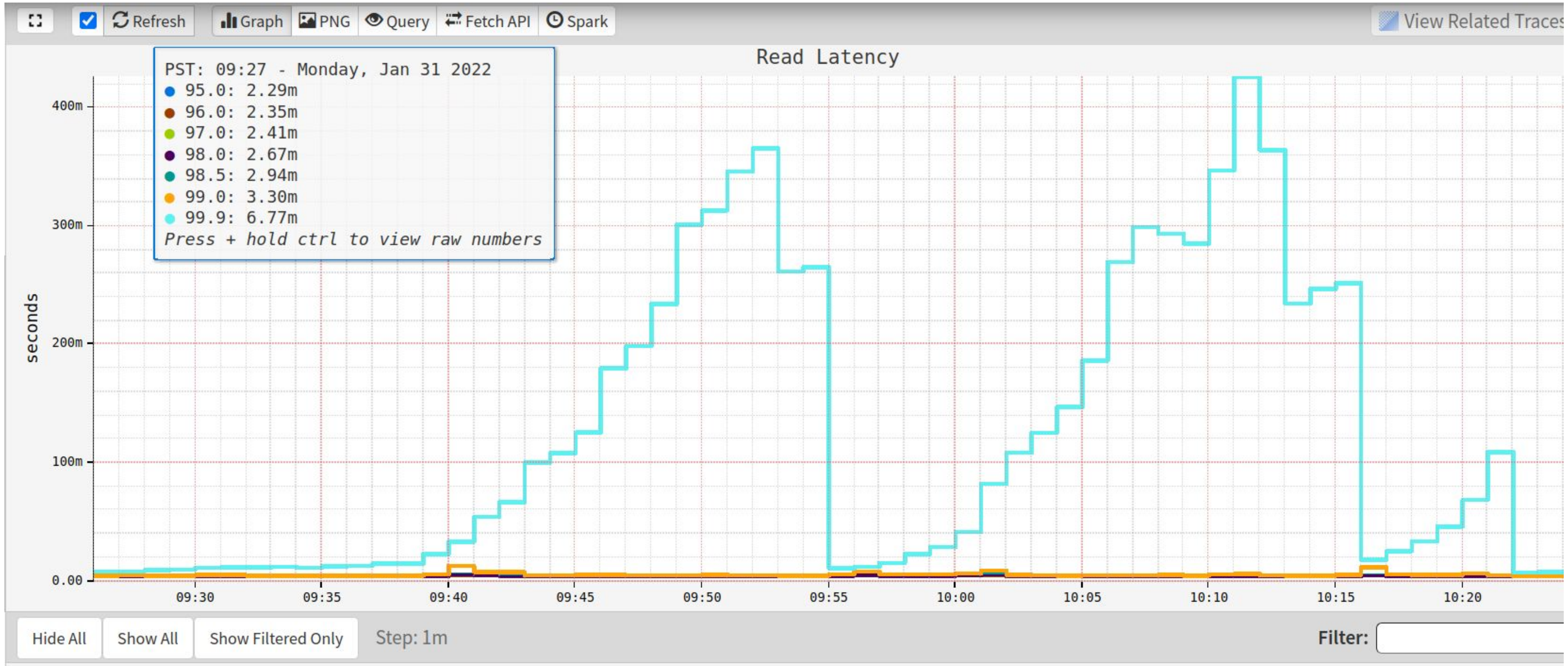
Hide All | Show All | Show Filtered Only | Step: 1m | Filter:

Min	Max	Average	Total	Name	Visibility
471.552	565.731	517.053	31.54k	i-0020298335a9fa171 us-east-1c read	
422.962	526.476	483.861	29.516k	i-003ee66a243bfe478 us-east-1d read	
422.703	549.304	483.589	29.499k	i-058edfd49acf20acf us-east-1e read	
475.699	572.951	523.743	31.948k	i-05b3b14eaba6cf90b us-east-1c read	
427.894	584.444	494.878	30.188k	i-05ef379148c188e77 us-east-1d read	
425.205	561.475	485.735	29.63k	i-066a4368fa876dcc8 us-east-1d read	
431.587	503.508	477.151	29.106k	i-08d5719a82a2ca146 us-east-1e read	
432.514	521.587	484.692	29.566k	i-09eb1027f00c8d4cb us-east-1c read	
434.94	502.931	475.15	28.984k	i-0c331c8d41436b4f6 us-east-1e read	
98.057	497.085	262.733	16.027k	i-0dae4a6623071ef95 us-east-1d read	
464.346	594.606	514.229	31.368k	i-0dc167503aeb65b01 us-east-1e read	
426.248	501.955	475.496	29.005k	i-0df3aa8a13c5ebc59 us-east-1c read	

Slow coordinators

1/12 = 8.3% should have been affected

But only .1% were



Min	Max	Average	Total	Name
2.272m	2.52m	2.346m	143.134m	95.0
2.33m	2.795m	2.411m	147.057m	96.0
2.389m	3.299m	2.52m	153.714m	97.0
2.446m	4.284m	2.842m	173.371m	98.0
2.71m	5.581m	3.186m	194.331m	98.5
2.767m	11.53m	3.951m	240.997m	99.0
5.865m	425.082m	117.75m	7.183	99.9

Real World Results

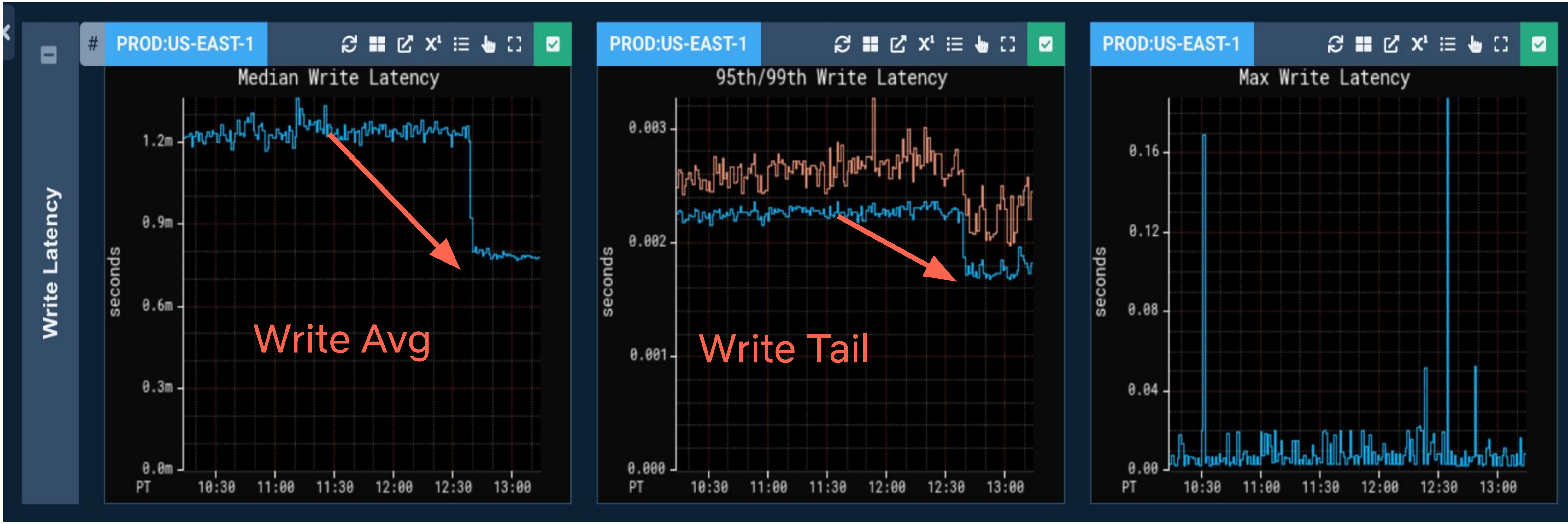
Apply Real Load
Measure Results

Watch Graphs Drop

Service #1 - LOCAL_ONE

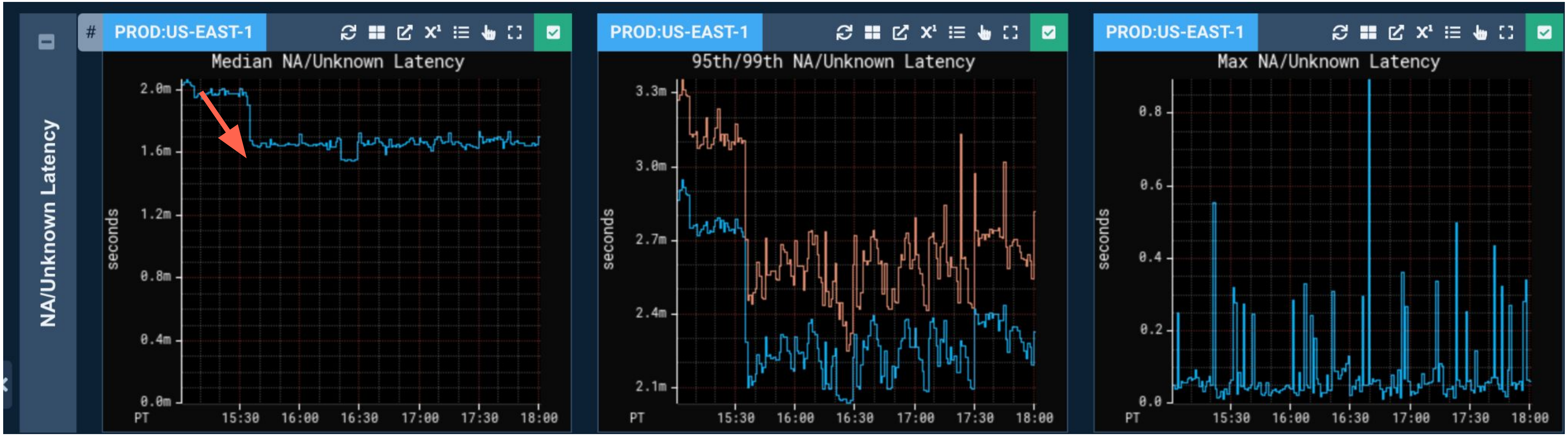
P50 1.1ms -> 0.7ms = **36%** improvement
P95 1.9ms -> 1.4ms = **26%** improvement
Local One workload

Service #1 - LOCAL_ONE



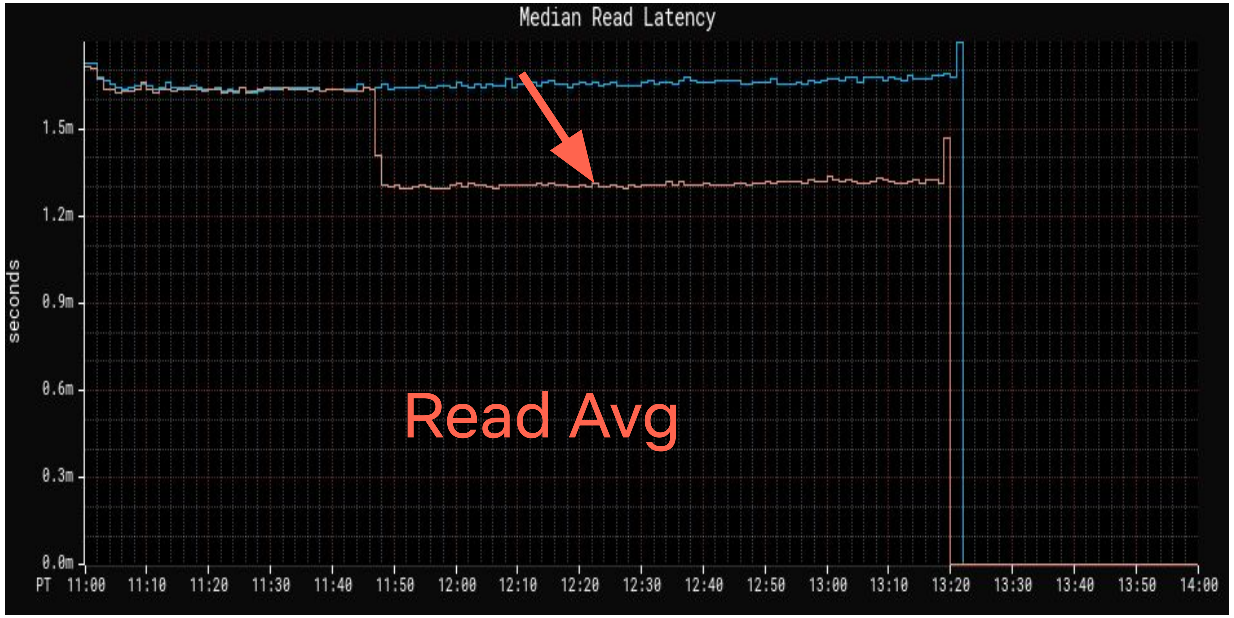
P50 1.2ms -> 0.7ms = **41%** improvement
P95 2.2ms -> 1.7ms = **22%** improvement
Local One workload

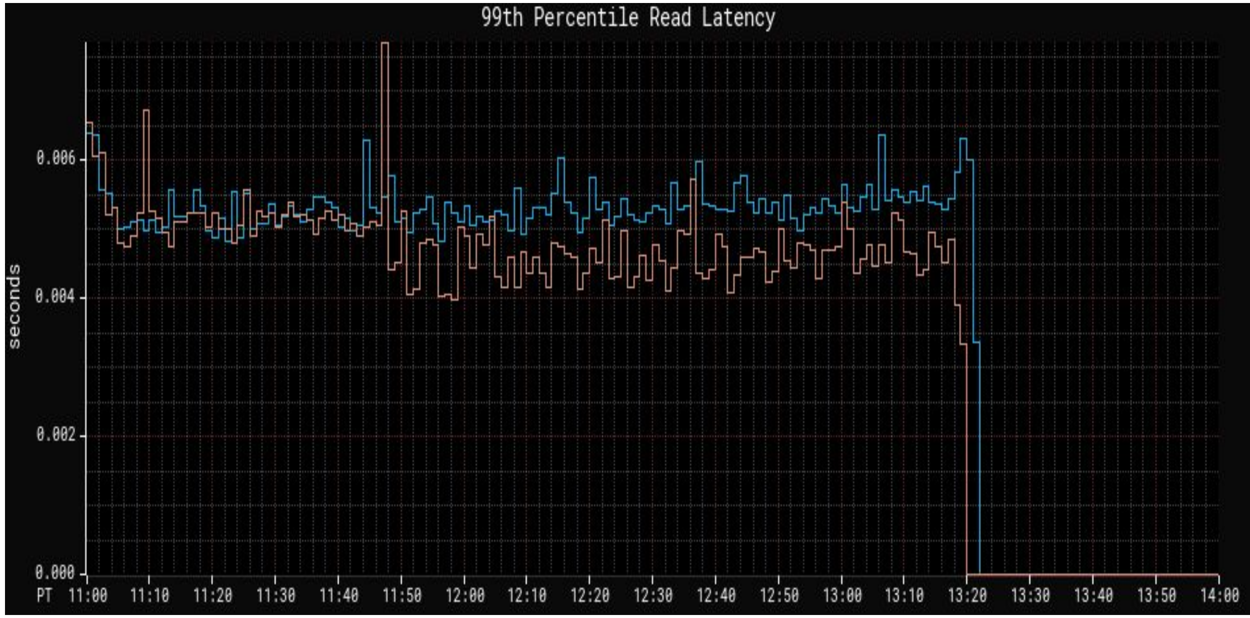
Service #2 - LOCAL_QUORUM



P50 2.0ms -> 1.6ms = **20%** improvement
P95 2.8ms -> 2.2ms = **22%** improvement
LWT (Local Serial) workload

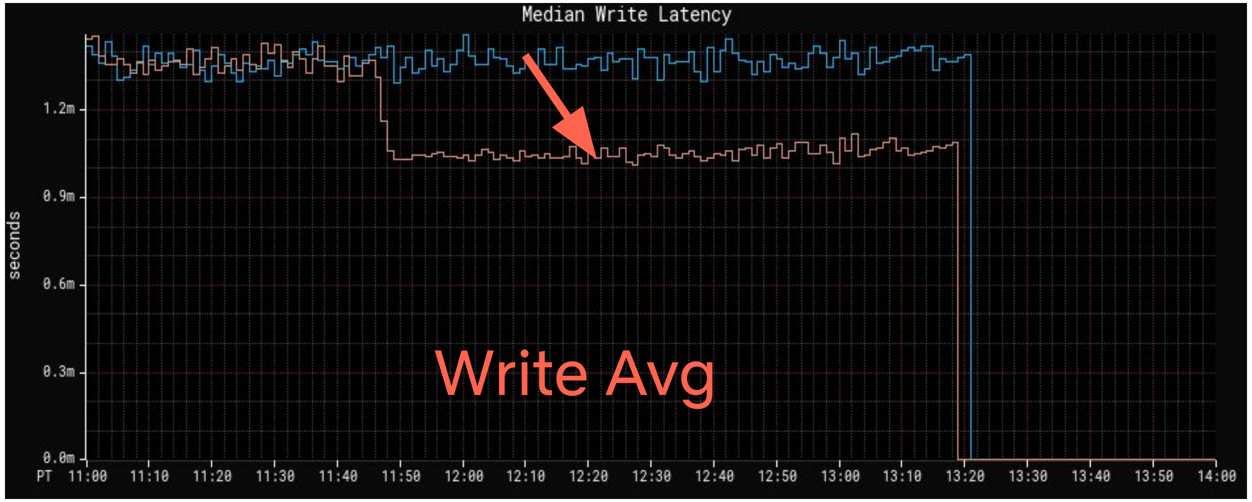
Service #3 - LOCAL_ONE

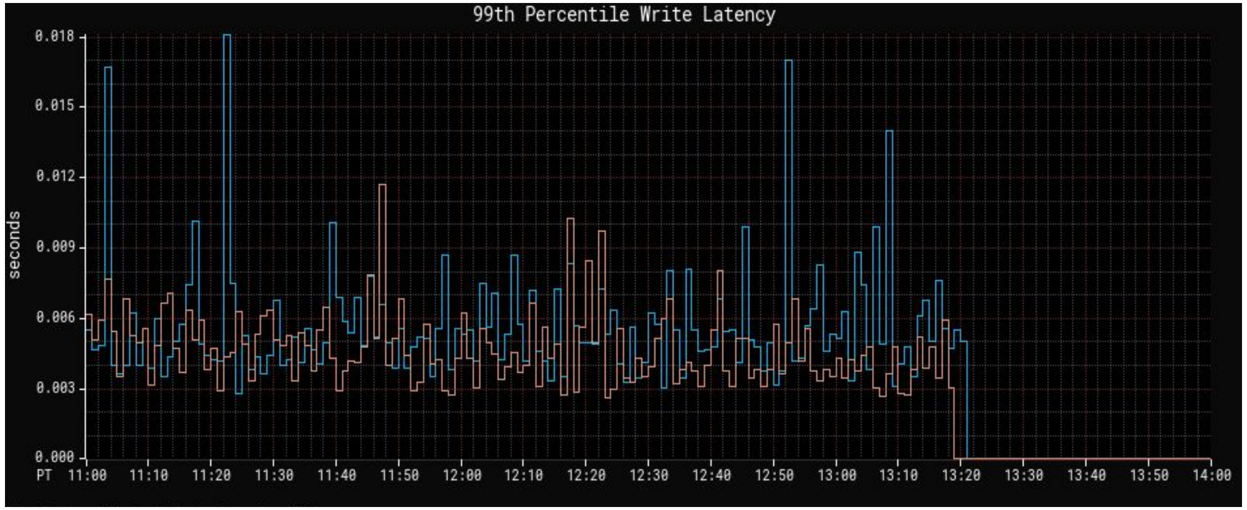




P50 1.6ms -> 1.2ms = **25%** improvement
P99 5.0ms -> 4.2ms = **16%** improvement
Local one workload

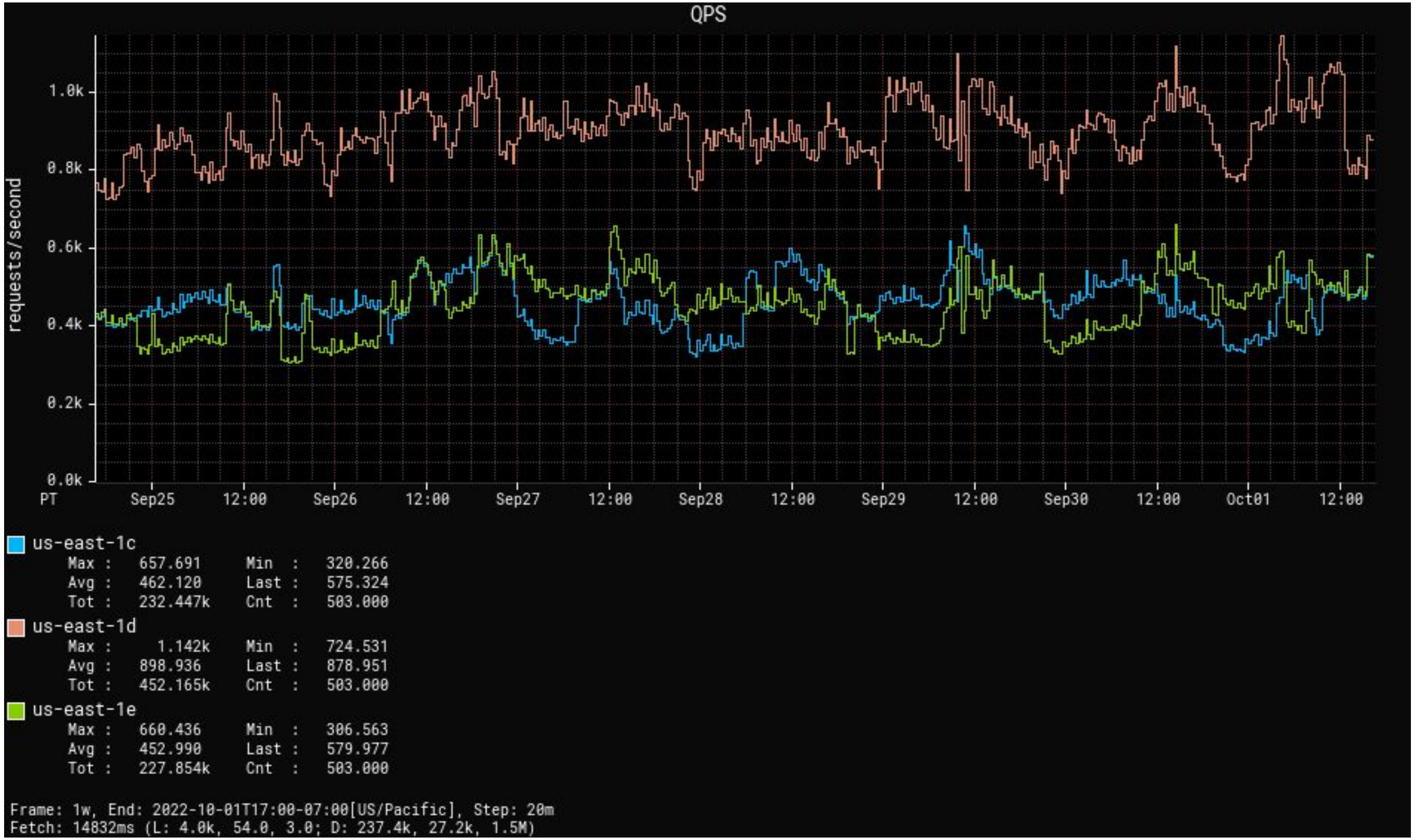
Service #3 - LOCAL_ONE



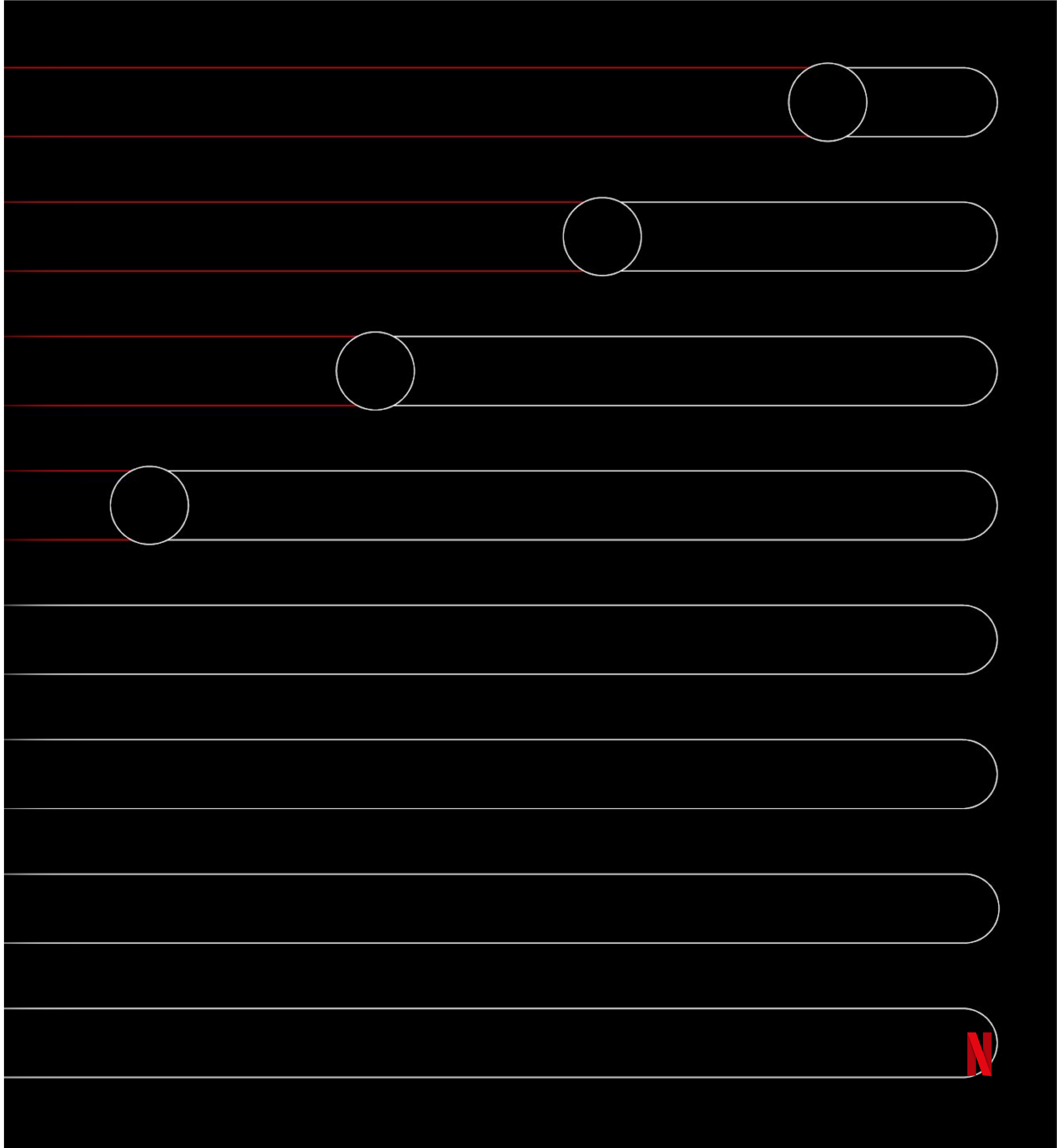


P50 1.3ms -> 0.9ms = **31%** improvement
P99 6.0ms -> 6.0ms = **~0%** improvement
Local one workload

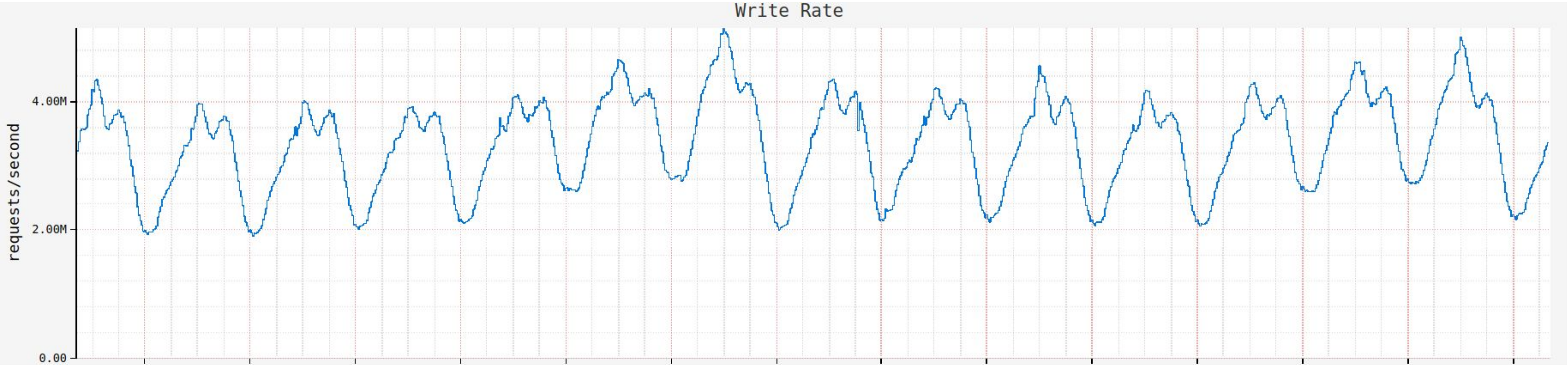
Uneven distribution of requests across zones



At Scale?



Scale?

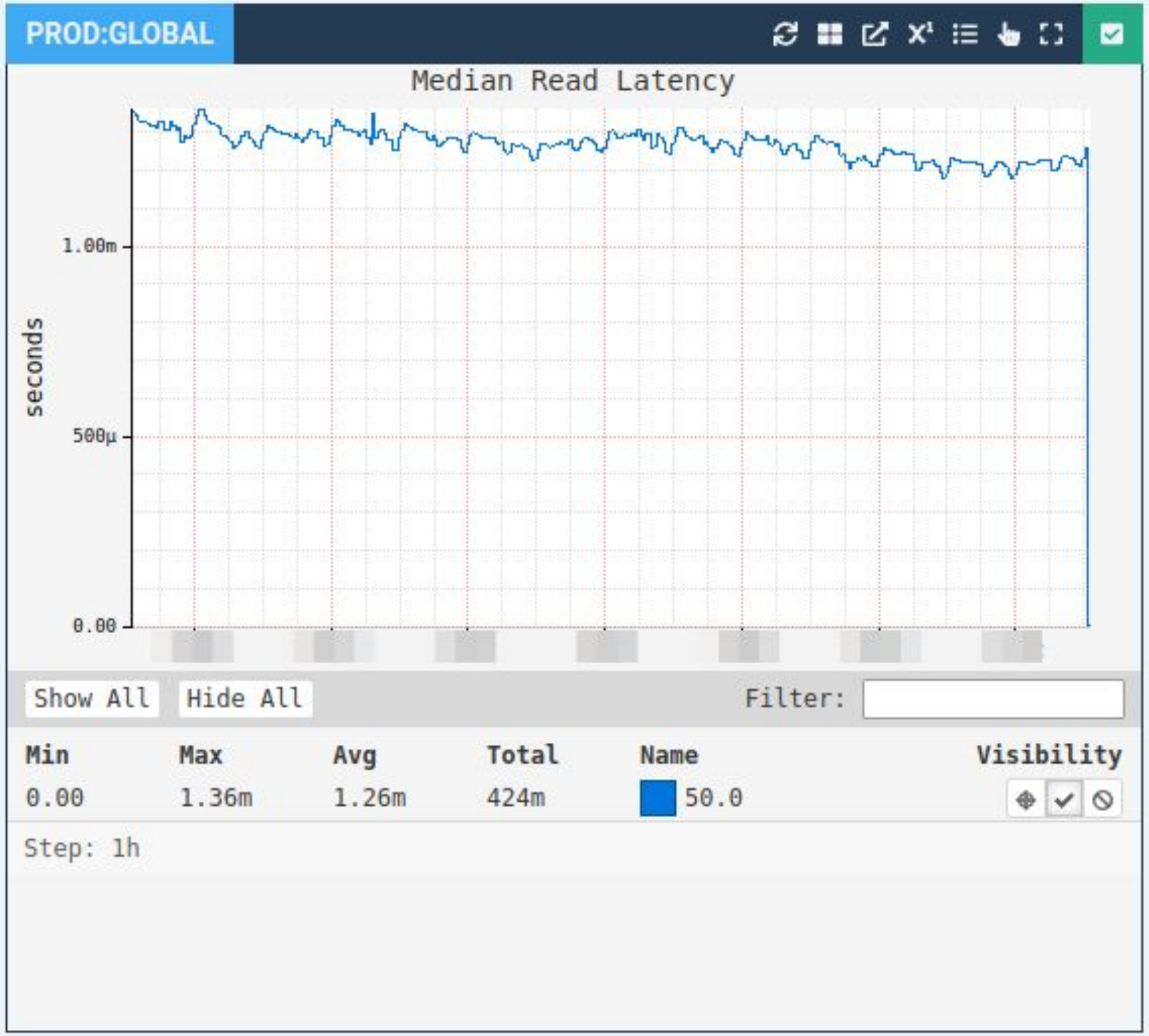


Peak Traffic is 5 Million Writes per Second

Scale?

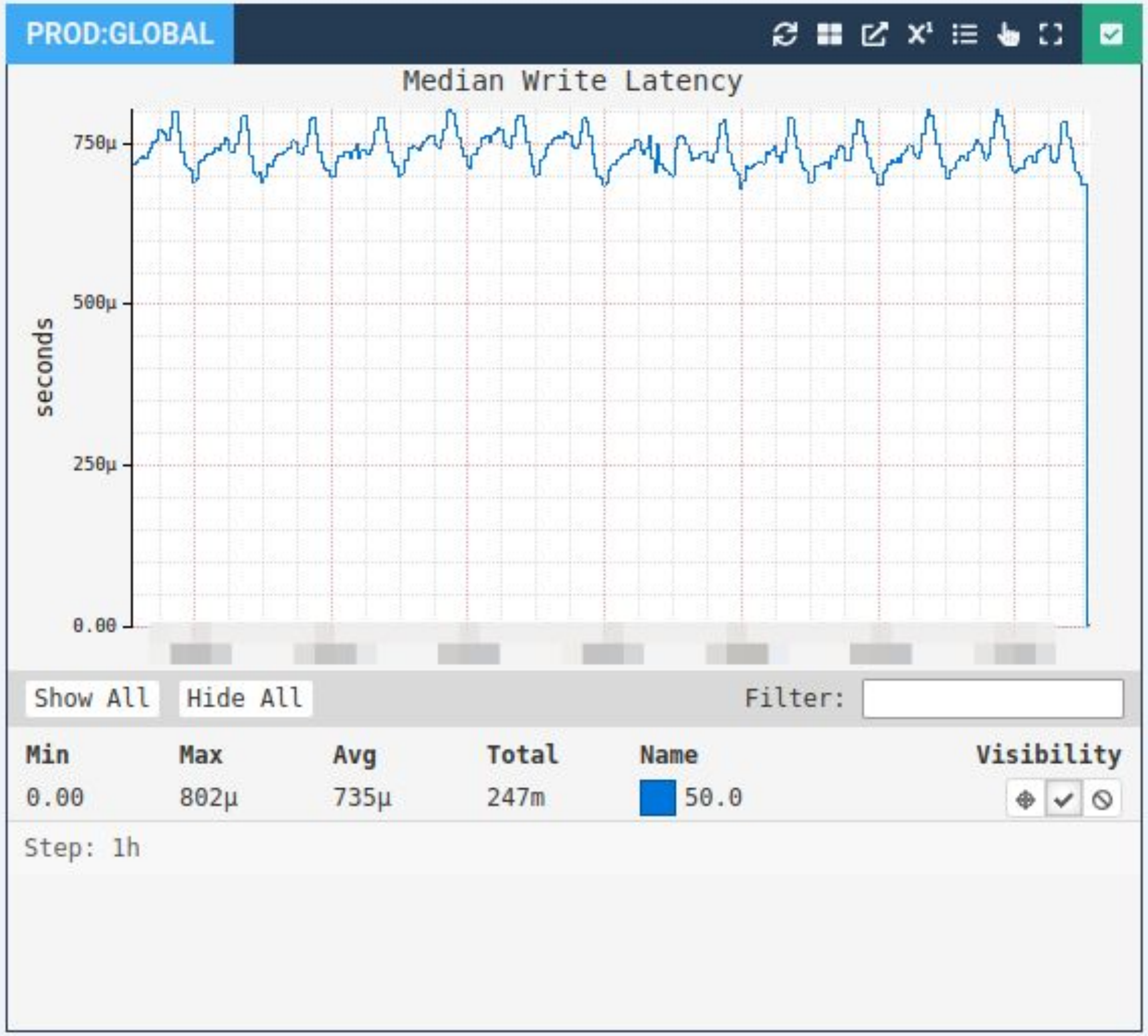
Peak Traffic is 6 Million Reads per Second

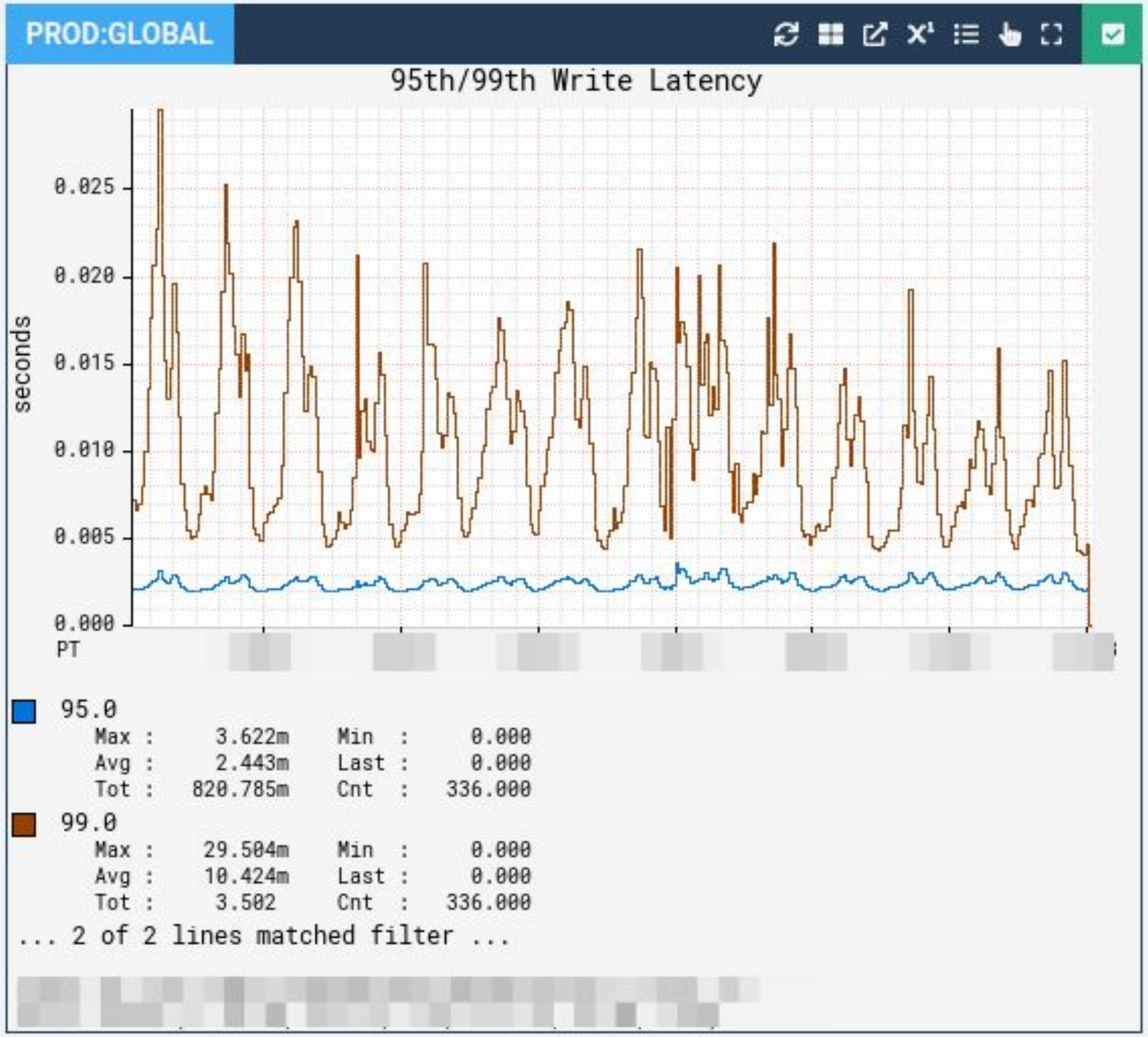
Scale?


PROD:GLOBAL
Median Read Latency
seconds
1.00m
500µ
0.00
Show All
Hide All
Filter:
Min
Max
Avg
Total
Name
Visibility
0.00
1.36m
1.26m
424m
50.0
Step: 1h


PROD:GLOBAL
95th/99th Read Latency
seconds
0.3
0.2
0.1
0.0
PT
95.0
Max : 6.067m Min : 0.000
Avg : 4.180m Last : 0.000
Tot : 1.404 Cnt : 336.000
99.0
Max : 319.619m Min : 0.000
Avg : 23.810m Last : 0.000
Tot : 8.000 Cnt : 336.000
... 2 of 2 lines matched filter ...

Scale?


PROD:GLOBAL
Median Write Latency
seconds
750µ
500µ
250µ
0.00
Show All
Hide All
Filter:
Min
Max
Avg
Total
Name
Visibility
0.00
802µ
735µ
247m
50.0
Step: 1h


PROD:GLOBAL
95th/99th Write Latency
seconds
0.025
0.020
0.015
0.010
0.005
0.000
PT
95.0
Max : 3.622m Min : 0.000
Avg : 2.443m Last : 0.000
Tot : 820.785m Cnt : 336.000
99.0
Max : 29.504m Min : 0.000
Avg : 10.424m Last : 0.000
Tot : 3.502 Cnt : 336.000
... 2 of 2 lines matched filter ...

Conclusions

1. Stay in Zone, failover when loaded

2. LO is easier to load balance for than LQ because we control the entire flow (snitch impacts LQ)

3. We can simulate slow coordinators, and protect against them.

WLLLB is widely deployed at Netflix handling over 10M QPS

Q/A