Introduction

This report evaluates the performance of Graviton 2 and Graviton 4 platforms under a modeled AdTech workload, designed to test the platform's ability to manage high-throughput, latency-sensitive operations. The workload is based on a previously published AdTech workload (https://aerospike.com/resources/white-papers/running-operational-workloads/) at scale, emulating real-world demands with billions of records and concurrent database operations. This workload measures maximum sustainable TPS while adhering to strict SLA latency thresholds for both reads and writes. These results offer a detailed assessment of the platforms' performance and suitability for demanding operational environments.

Workload Summary

Data Models

The workload is focused on emulating real-world demands of AdTech and similar industries, emphasizing ultra-low latency, high availability, scalability, and cost efficiency.

User Profile Database:

- Generates simulated data akin to that which tracks market segments, geographic data, etc.
- Maintains 50 billion unique records (~100 TB uncompressed, before replication).
- Read-heavy workload to support real-time ad bidding decisions.

Campaign Database:

- Stores transactional information for ad campaigns, such as budgets and bidding data.
- Maintains 600 million unique records (~2 TB uncompressed).
- Balanced 50/50 read/write workload.

Workload:

- Concurrent workloads for both databases.
- 80/20 read/write for `user-profile` data.
- 50/50 read/write for `campaign` data.

Replication and Compression:

- **Replication Factor:** 2 (ensures data availability).
- **Compression:** Lz4 with a 4:1 ratio to reduce storage requirements.

SLA

Many workloads have tight latency requirments, and this is especially true for our customers in the AdTech space. For this workload, we are repoting the maximum TPS that the cluster can sustain while the p99.9 for reads was under 1.0 ms, and writes were under 2.0 ms, for both concurrent workloads.

Infrastructure Configuration

- Deployed on a 9-node Aerospike cluster

- Deployed 16 load generation servers (lgs)
- User profile database stored entirely on Flash (SSDs) for cost efficiency
- Campaign database used hybrid storage (indexes in DRAM, data on Flash) to optimize read/write balance

More details are available in the Appendix

Methodology

Cluster Hydration

We used **asbench** to generate the data. For each of the load generation servers, 8 copies of asbench were run. Hydration took between 6 and 12 hours.

Details on the exact parameters are available in the appendix.

Cluster Conditioning

Aerospike does not mutate the data in place, and thus has to defrag disk blocks in order to reclaim space. Defrag puts pressure on the machines' IO subsystem, and so in order to measure what a system can do in production, we must condition the cluster to be in a state where it is doing both the workload, and the ongoing background maintaince tasks. The clusters were loaded with data, and then an update workload was applied to put the cluster into defrag equilibrium. The clusters were under this workload for approximately one week before the workloads were run.

TPS Sweeps

We pushed each system to the maximum throughput it could support for the selected SLA. The run that achieved the maximum throughput for the chosen SLA is presented in this report.

We aim to demonstrate the upper limits of system performance under controlled conditions. Showing the maximum sustainable TPS highlights the hardware and software's capabilities in handling demanding real-world scenarios.

Workload

For this workload, we used **asbench** to generate the requests. For each of the load generation servers, N copies of asbench were run. Details on the exact parameters are available in the appendix.

TPS Measurement

From above, we know that our clusters were in equilibrium, and we knew what TPS the cluster can sustain without breaking the SLA. For expediancy we are showing the workload over a 10 min run. However, since the cluster is in defrag equilibrium we know that this throughput is sustainable untils either there is HW failures, or a significant change in the network latency.

Results

Below are the results for the Graviton 2 platform:

Namespace	Config	Read TPS	Write TPS	Total TPS
`user-profile`	ALL FLASH	150,860	37,715	188,575
`campaign`	HMA	2,395	2,395	4,790
			Total:	193,365

and the Graviton 4 platform:

Namespace	Config	Read TPS	Write TPS	Total TPS
`user-profile`	ALL FLASH	905,160	226,290	1,131,450
`campaign`	HMA	14,370	14,370	28,740
			Total:	1,160,190

We can further distill this down,

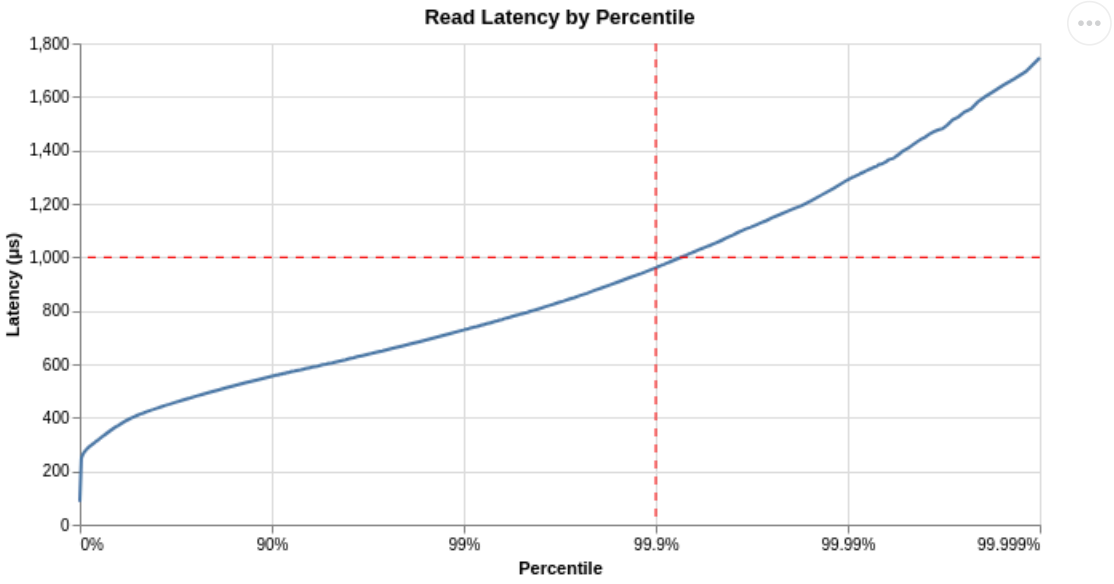
$$TPS_{mult} = \frac{\text{Graviton 4 TPS}}{\text{Graviton 2 TPS}} = \frac{1,160,190}{193,365} = 6.0$$

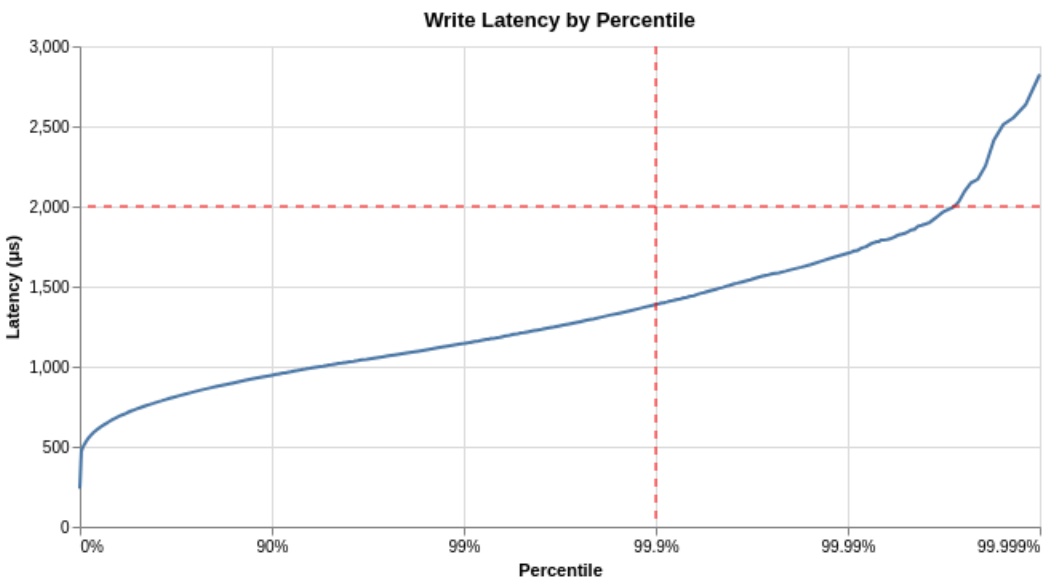
The load generation tool, `asbench`, emits `HdrHistogram` files, which allows us to aggregate the latency across all of the `asbench` instances. Below, you can see the full percentile spectrum plots for latency. The red annotations denote the target SLA.

For further information refer to Gil Tene's HdrHistogram (https://hdrhistogram.github.io/HdrHistogram/) project.

Full Percentile Spectrum Plot - `theseus-g2-asd`

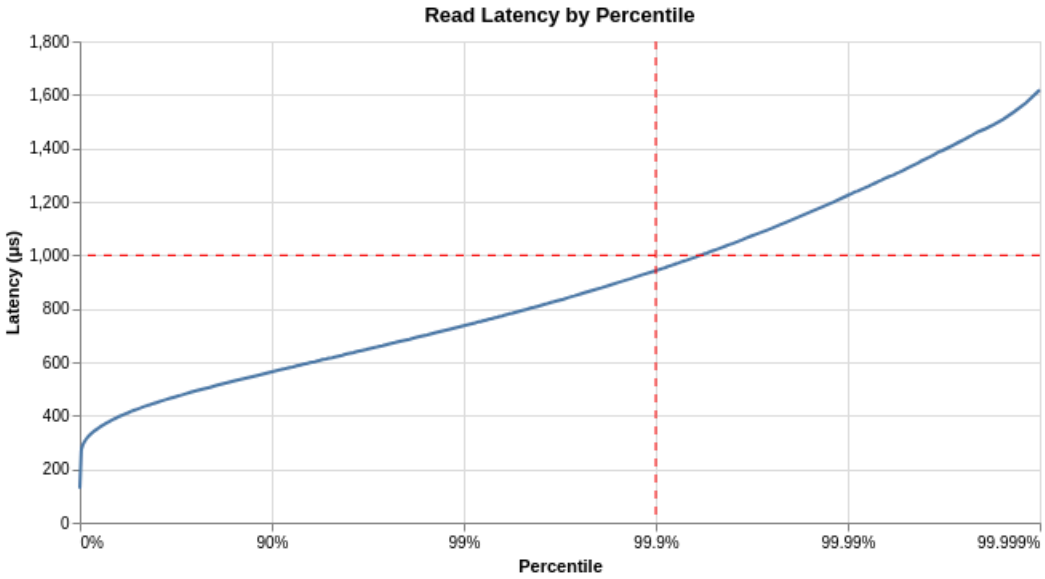
The p99.9 for reads were 961µs <= SLA of 1000µs, and the p99.9 for writes were 1387µs <= SLA of 2000µs.

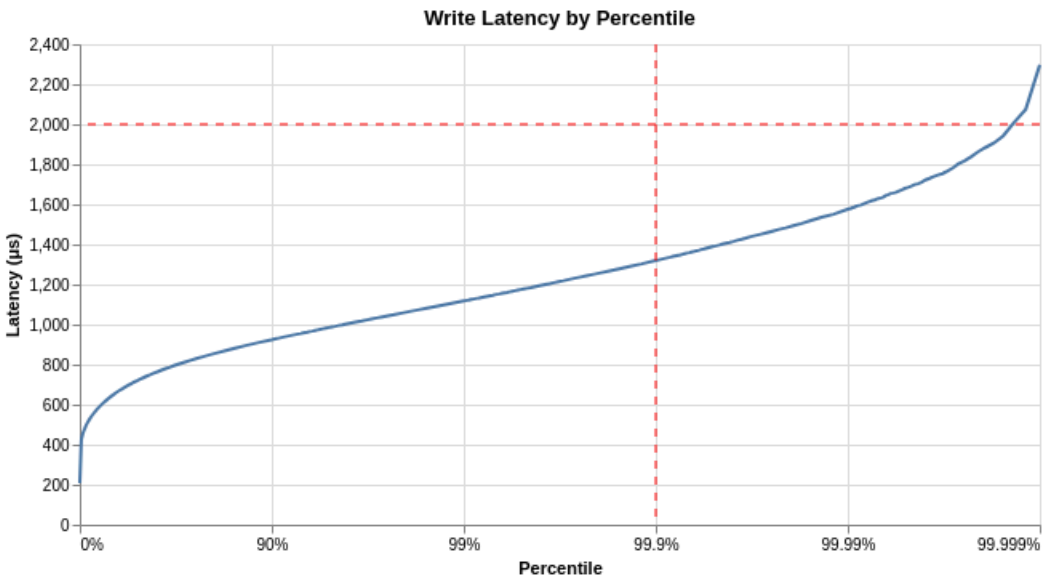




Full Percentile Spectrum Plot - `theseus-g4-asd`

The p99.9 for reads were 942µs <= SLA of 1000µs, and the p99.9 for writes were 1320µs <= SLA of 2000µs.





Appendix

Deployment Details

Each system is equipped with four local nvme drives. Each has six partitions.

The first partition is for storage for the **campaign** namespace. The remaining partitions are for storage for the **user-profile** namespace and its index.

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
nvme2n1     259:0    0    12G  0 disk
├─nvme2n1p1 259:1    0   200M  0 part /boot/efi
├─nvme2n1p2 259:2    0   512M  0 part /boot
└─nvme2n1p3 259:3    0  11.3G  0 part /
nvme0n1     259:4    0   3.4T  0 disk
├─nvme0n1p1 259:8    0  69.8G  0 part
├─nvme0n1p2 259:9    0 768.3G  0 part
├─nvme0n1p3 259:10   0 768.3G  0 part
├─nvme0n1p4 259:11   0 768.3G  0 part
├─nvme0n1p5 259:12   0 768.3G  0 part
└─nvme0n1p6 259:13   0 349.2G  0 part /mnt/nvme0n1p6
nvme1n1     259:5    0   3.4T  0 disk
├─nvme1n1p1 259:14   0  69.8G  0 part
├─nvme1n1p2 259:15   0 768.3G  0 part
├─nvme1n1p3 259:16   0 768.3G  0 part
├─nvme1n1p4 259:17   0 768.3G  0 part
├─nvme1n1p5 259:18   0 768.3G  0 part
└─nvme1n1p6 259:19   0 349.2G  0 part /mnt/nvme1n1p6
nvme3n1     259:6    0   3.4T  0 disk
├─nvme3n1p1 259:20   0  69.8G  0 part
├─nvme3n1p2 259:21   0 768.3G  0 part
├─nvme3n1p3 259:22   0 768.3G  0 part
├─nvme3n1p4 259:23   0 768.3G  0 part
├─nvme3n1p5 259:24   0 768.3G  0 part
└─nvme3n1p6 259:25   0 349.2G  0 part /mnt/nvme3n1p6
nvme4n1     259:7    0   3.4T  0 disk
├─nvme4n1p1 259:26   0  69.8G  0 part
├─nvme4n1p2 259:27   0 768.3G  0 part
├─nvme4n1p3 259:28   0 768.3G  0 part
├─nvme4n1p4 259:29   0 768.3G  0 part
├─nvme4n1p5 259:30   0 768.3G  0 part
└─nvme4n1p6 259:31   0 349.2G  0 part /mnt/nvme4n1p6
```

The **user-profile** namespace was configured with the primary index on disk for cost efficiency. The **campaign** namespace uses primary index in memory. Both namespaces are configured to compress the data using **zstd** compression level 4. The cluster configuration can be seen below.

```
service {
    cluster-name theseus-g2-asd
    migrate-max-num-incoming 8
    migrate-threads 4
    proto-fd-max 90000
}
logging {
    console {
        context any info
    }
}
network {
    service {
        address any
        port 3000
    }
    heartbeat {
        interval 150
        mesh-seed-address-port 172.31.17.97 3002
        mesh-seed-address-port 172.31.21.105 3002
        mesh-seed-address-port 172.31.29.74 3002
        mesh-seed-address-port 172.31.27.229 3002
        mesh-seed-address-port 172.31.21.189 3002
        mesh-seed-address-port 172.31.18.12 3002
        mesh-seed-address-port 172.31.31.162 3002
        mesh-seed-address-port 172.31.26.220 3002
        mesh-seed-address-port 172.31.19.185 3002
        mode mesh
        port 3002
        timeout 10
    }
```

```
    fabric {
        port 3001
    }
    info {
        port 3003
    }
}
namespace user-profile {
    default-ttl 0
    enable-benchmarks-write true
    index-type flash {
        evict-mounts-pct 90
        mount /mnt/nvme0n1p6
        mount /mnt/nvme1n1p6
        mount /mnt/nvme3n1p6
        mount /mnt/nvme4n1p6
        mounts-budget 1256277934080
    }
    partition-tree-sprigs 262144
    replication-factor 2
    storage-engine device {
        compression zstd
        compression-level 4
        device /dev/nvme0n1p2
        device /dev/nvme0n1p3
        device /dev/nvme0n1p4
        device /dev/nvme0n1p5
        device /dev/nvme1n1p2
        device /dev/nvme1n1p3
        device /dev/nvme1n1p4
        device /dev/nvme1n1p5
        device /dev/nvme3n1p2
        device /dev/nvme3n1p3
        device /dev/nvme3n1p4
        device /dev/nvme3n1p5
        device /dev/nvme4n1p2
        device /dev/nvme4n1p3
        device /dev/nvme4n1p4
        device /dev/nvme4n1p5
        flush-size 128K
        stop-writes-used-pct 80
    }
}
namespace campaign {
    default-ttl 0
    enable-benchmarks-write true
    replication-factor 2
    storage-engine device {
        compression zstd
        compression-level 4
        device /dev/nvme0n1p1
        device /dev/nvme1n1p1
        device /dev/nvme3n1p1
        device /dev/nvme4n1p1
        flush-size 128K
    }
}
```

Graviton 2 Cluster

For the Graviton 2 cluster we deployed:

```
  cluster name: theseus-g2-asd
  instance type: i4g.16xlarge

  run start: 2024-11-22 21:13:37.247000
  run end:   2024-11-22 21:27:06.904000
```

Aerospike Cluster Servers

Attribute	Value
Cluster Type	aerospike

Load Generation Servers

Attribute	Value
Cluster Type	tools

Attribute	Value
Aerospike Version	7.1.0.9
Architecture	arm64
Distribution	centos
OS Version	9
Zone	us-east-1
Instance Type	i4g.16xlarge
Node Count	9

Attribute	Value
Aerospike Version	client
Architecture	x86_64
Distribution	centos
OS Version	9
Zone	us-east-1
Instance Type	c5a.4xlarge
Node Count	16

Graviton 4 Cluster

For the Graviton 4 cluster we deployed:

```
cluster name: theseus-g4-asd
instance type: i8g.16xlarge

run start: 2024-11-20 22:03:16.806000
run end:   2024-11-20 22:16:36.345000
```

Aerospike Cluster Servers

Attribute	Value
Cluster Type	aerospike
Aerospike Version	7.1.0.9
Architecture	arm64
Distribution	centos
OS Version	9
Zone	us-east-1
Instance Type	i8g.16xlarge
Node Count	9

Load Generation Servers

Attribute	Value
Cluster Type	tools
Aerospike Version	client
Architecture	x86_64
Distribution	centos
OS Version	9
Zone	us-east-1
Instance Type	c5a.4xlarge
Node Count	16

Hydration Details

User profile namespace:

- All Flash (index and data on SSDs)
- 62.5 billion unique objects of each of these sizes (in bytes):
  - 256 B, 512 B, 768 B, 1024 B, 2048 B, 3072 B, 4096 B, and 5120 B
- Replication factor 2
- Lz4 compression; compression ratio of 4
- Consistency mode: AP (available/partition tolerant)

Campaign namespace:

- Hybrid memory (index in DRAM and data on SSDs)

- 600 million unique keys
- Object size: 256 B (bytes)
- Replication factor 2
- Lz4 compression; compression ratio of 4
- Consistency mode: AP (available/partition tolerant)

```
...
38: export USER_PROFILE_ENTRIES_PER_DATUM_SIZE=6250000000
39: export USER_PROFILE_ENTRIES=$(( USER_PROFILE_ENTRIES_PER_DATUM_SIZE * 7 ))
40: export CAMPAIGN_ENTRIES=600000000
...
```

The relevant calls to **asbench** for the hydrating the cluster are

```
...
namespace="user-profile"
index=0
for bin_size in 5720 4096 2028 1024 768 512 256; do

        par-exec🔬 --scriptname "${petname}-$(basename $0)-${namespace}-B${bin_size}" --skip-collect -- \
          ${CURRENT_PROJECT_ROOT}/project-scripts/asbench_workload.sh \
                --cluster-name "${TARGET_CLUSTER}" \
                --client-cluster-name "${TARGET_CLIENT_CLUSTER}" \
                --start-key $(( USER_PROFILE_ENTRIES_PER_DATUM_SIZE * index )) \
                --total-keys ${USER_PROFILE_ENTRIES_PER_DATUM_SIZE} \
                --bin-size "${bin_size}" \
                --compression-ratio 0.25 \
                --namespace "${namespace}" \
                --total-tps 0 \
                --client-count 8

        index=$(( index + 1 ))
done
...
namespace="campaign"
index=0
for bin_size in 256; do
        par-exec🔬 --scriptname "${petname}-$(basename $0)-${namespace}-B${bin_size}" -- \
          ${CURRENT_PROJECT_ROOT}/project-scripts/asbench_workload.sh \
                --cluster-name "${TARGET_CLUSTER}" \
                --client-cluster-name "${TARGET_CLIENT_CLUSTER}" \
                --start-key 0 \
                --total-keys "${CAMPAIGN_ENTRIES}" \
                --bin-size "${bin_size}" \
                --compression-ratio 0.25 \
                --namespace "${namespace}" \
                --total-tps 0 \
...
```

and the full listing is available in the **10-hydrate** script in the repository that occupanies this report

And the after hydration, the namespaces looked like

Namespace	Objects	Tombstones	Master Objects	Prole Objects	Data Used Bytes	Evicted Objects
user-profile	87.5B	0	43.8B	43.8B	48.6Ti	0
campaign	1.2B	0	600.0M	600.0M	178.8Gi	0

Workload Details

The relevant calls to `asbench` for the workload are

```
...
namespace="user-profile"
        index=0
        for bin_size in 5720 4096 2028 1024 768 512 256; do
                par-exec🔬 --scriptname "${petname}-$(basename $0)-${namespace}-B${bin_size}" --skip-collect -- \
                  "${CURRENT_PROJECT_ROOT}/project-scripts/asbench_workload.sh" \
                        --cluster-name "${TARGET_CLUSTER}" \
                        --client-cluster-name "${TARGET_CLIENT_CLUSTER}" \
                        --start-key $(( USER_PROFILE_ENTRIES_PER_DATUM_SIZE * index )) \
                        --total-keys "${USER_PROFILE_ENTRIES_PER_DATUM_SIZE}" \
                        --bin-size ${bin_size} \
                         --compression-ratio 0.25 \
                        --namespace ${namespace} \
                        --workload-type gaussian \
                        --total-tps ${tps_per_bin} \
                        --read-percentage ${read_percentage} \
                        --client-count 2 \
                        --standard-deviation 7 \
                        --timeout "${timeout}" &

                index=$(( index + 1 ))
...
for bin_size in 256; do
                par-exec🔬 --scriptname "${petname}-$(basename $0)-${namespace}-B${bin_size}" --skip-collect -- \
                  "${CURRENT_PROJECT_ROOT}/project-scripts/asbench_workload.sh" \
                        --cluster-name "${TARGET_CLUSTER}" \
                        --client-cluster-name "${TARGET_CLIENT_CLUSTER}" \
                        --start-key 0 \
                        --total-keys "${CAMPAIGN_ENTRIES}" \
                        --bin-size ${bin_size} \
                        --compression-ratio 0.25 \
                        --namespace ${namespace} \
                        --workload-type gaussian \
                        --total-tps $(perl -le "print int(9580 * $tps_multiplier)") \
                        --read-percentage 50 \
                        --client-count 2 \
                        --standard-deviation 7 \
                        --timeout "${timeout}" &

                index=$(( index + 1 ))
...
```

and the full listing is available in the `11-run-workload` script in the repository that occupanies this report

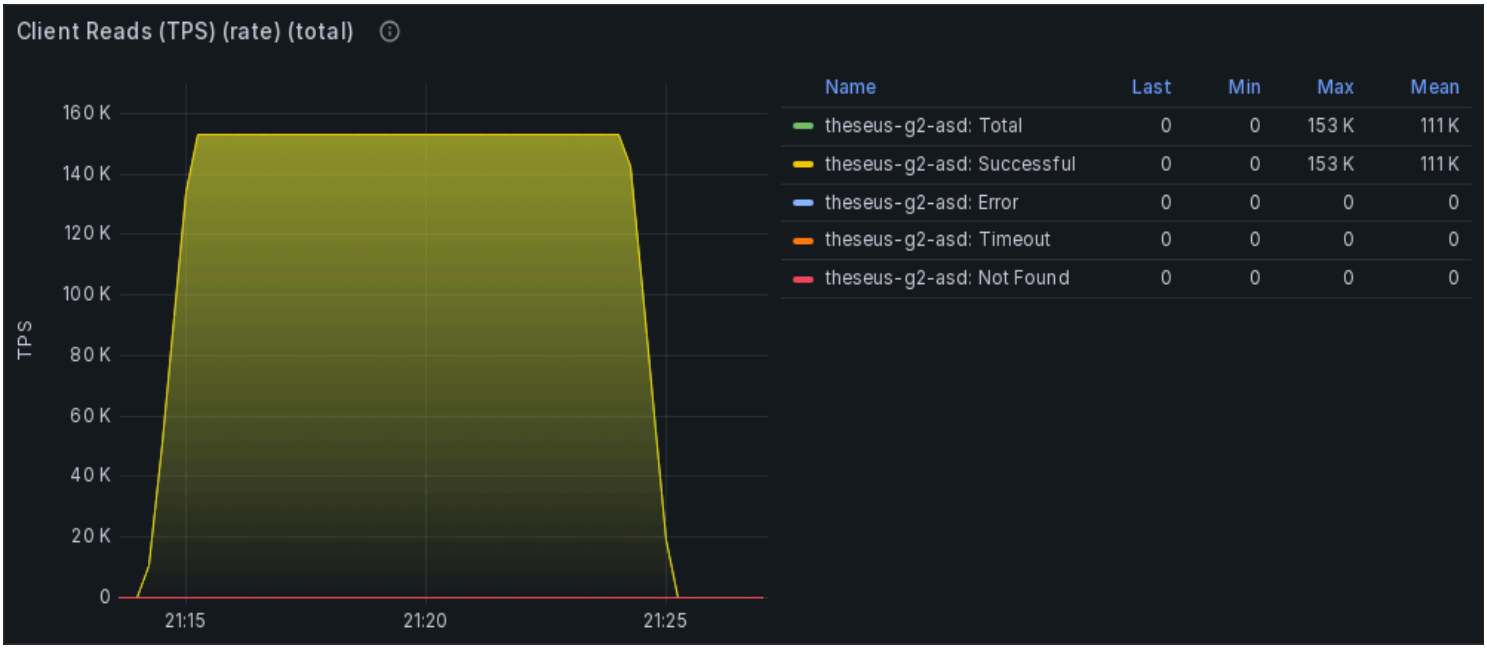
Run Details

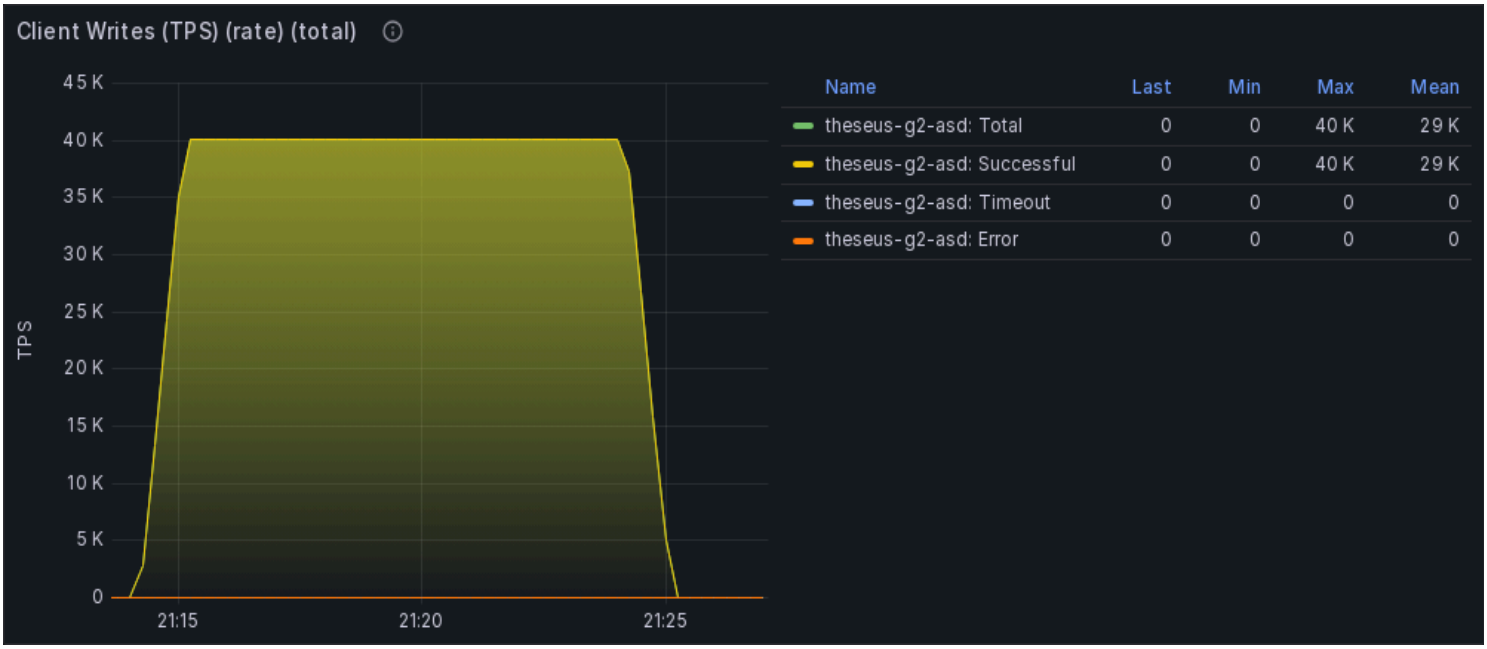
Below you can find the `asbench` parameters used during this test.

Graviton 2 Latency and throughput

Sustained TPS

Namespace	Config	Read TPS	Write TPS	Total TPS
`user-profile`	ALL FLASH	150,860	37,715	188,575
`campaign`	HMA	2,395	2,395	4,790
			Total:	193,365





Graviton 4 Latency and throughput

Sustained TPS

Namespace	Config	Read TPS	Write TPS	Total TPS
`user-profile`	ALL FLASH	905,160	226,290	1,131,450
`campaign`	HMA	14,370	14,370	28,740
			Total:	1,160,190