How to Benchmark Redis

Or Shwartz
Product Manager @Redis Labs

Filipe Oliveira
Performance Engineer @Redis Labs

redisconf 2021

RedisConf 2021 How to benchmark Redis demo performance report

Your home, RedisConf 2021

0. Requirements

As part of a re-architecture of our product XXX, we were tasked with kicking off the analysis of Redis performance given our Product, Infra, and Engineering requirements.

- Our product workload is majorly read-based (95% READS, 5% WRITES).
- The common case data size is 1KB.
- We are expected to have 1M application entities present in the database.
- SLA: 99% requests to be served below 10ms.
- At max we expect 50 concurrent APP connections to the database.
- The engineering is targeting HASHES as the suitable data type for our data.
- Our DevOps team recommended using m5d.8xlarge AWS cloud instances, OS Ubuntu 20.04

0.1 Goal

- Maximum achievable throughput for our data/use-case constraints
- Provide detailed overall latency and assess if we would be able to match the SLA

1. Summary report

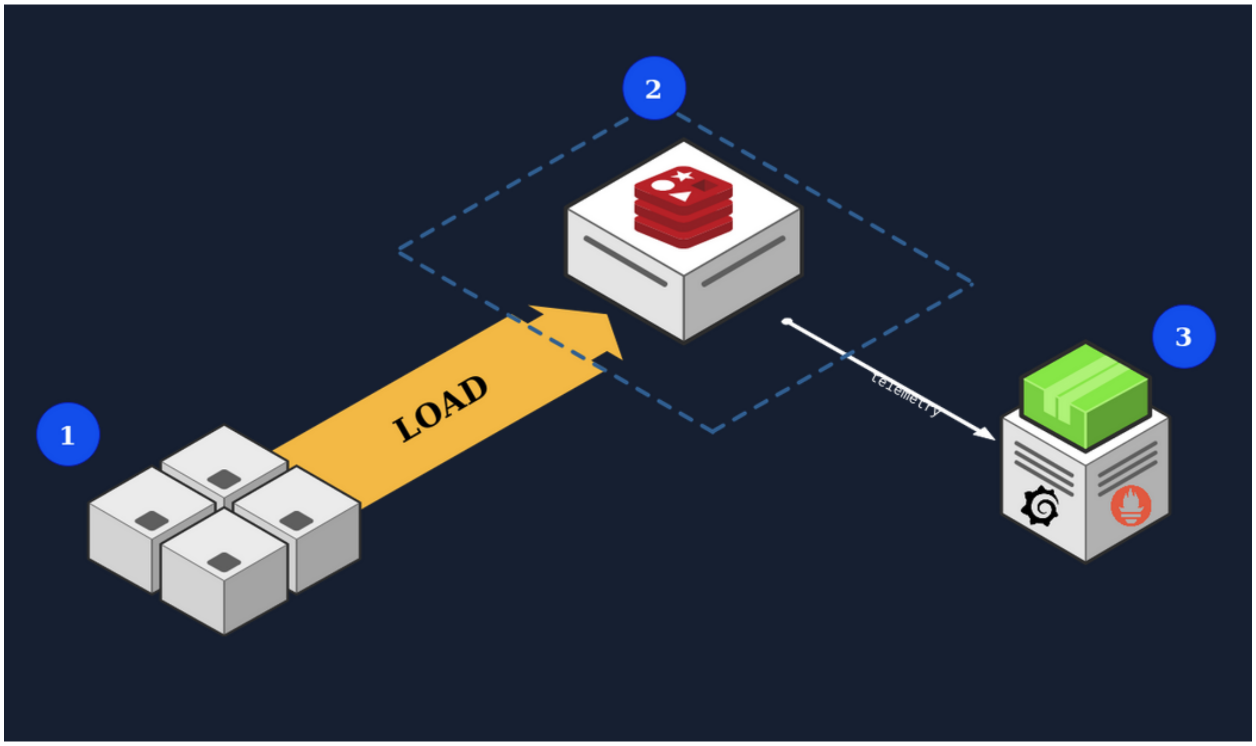
- Maximum achievable throughput for our data/use-case constraints
  - <FILL IN>
- SLA: 99% requests to be served below 10ms assessment
  - <FILL IN>

2.0 Test case description

Given the above requirements:

- We will use memtier_benchmark given it provides better reporting and mixed reads/writes workloads.
- Write-to-read ratios of 5:95 (5% write, 95% read).
- We will use object sizes of 1000 bytes to simulate common use cases.
- We will use 50 concurrent clients to match our maximum moment in time APP client connections.
- Database Persistence is kept to default.
- Replication is disabled.
- We will use one m5.8xlarge DB instance, composed of 32 VCPUs, 128GB of RAM, with the full specs on section 4.
- We will use one i3.8xlarge Benchmark instance, composed of 32 VCPUs, 128GB of RAM, with the full specs on section 4.

3. Benchmark setup



4. Methodology and Infrastructure

For each tested version:

- We've performed 3 distinct full repetitions, and report as a result the full latency spectrum, and overall achievable throughput.

4.1 Platform Description

Both Benchmark and DB instances were placed in an optimal networking scenario. All DB and Benchmark machines presented the same HW specifications:

AWS VM type	m5d.8xlarge
Operating System:	GNU/Linux
Kernel Release:	5.4.0-1041-aws
Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	32

Thread(s) per core:	2
Core(s) per socket:	16
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
Model name:	Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz
Hypervisor vendor:	KVM
Virtualization type:	full
L1d cache:	512 KiB
L1i cache:	512 KiB
L2 cache:	16 MiB
L3 cache:	35.8 MiB
NUMA node0 CPU(s):	0-31
Total online memory:	126.6G

4.2 - Underlying network conditions

All performance benchmarks are run as instances in AWS through our benchmark testing infrastructure. In addition to the memtier performance scenarios described above, we also run baseline netperf TCP_RR, a trading process of multiple TCP request and response occurring in the same TCP connection (frequent pattern in DB workloads), in order to understand the underlying network characteristics.

Per Packet's latency

```
netperf -t TCP_RR -H <your host> -n 8 -l 180 -- -r 1000,1000 -o
```

As visible on the retrieved values you can expect:

- 50% of the requests to have a RTT below <FILL IN> us.
- 99% of the requests to have a RTT below <FILL IN> us.

Network BW

```
netperf  -H <your host> -S -l 180 -- -m 1000 -M 1000
```

<FILL IN> Mbits/sec

5. Detailed results section

5.1 - prepopulation of data

```
memtier_benchmark -t 4 -c 1 --pipeline 10  -s 10.3.0.91 -p 6379 -d 1000  --key-maximum=1000000
--command="hset __key__ field1 __data__"  --command-key-pattern=P -n allkeys
```

5.2 - Used benchmark command

```
memtier_benchmark -t 5 -c 10 -s 10.3.0.91 -p 6379 -d 1000 --test-time 300 --run-count 3
--key-maximum=1000000 --distinct-client-seed  --command="hset __key__ field1 __data__"
--command-ratio=5 --command="hgetall __key__" --command-ratio=95
--hdr-file-prefix="mixed.latency.data" --json-out-file "mixed.workload.json"
```

5.2.1 - benchmark output

```
<FILL IN>
```

5.3 - Underlying DB conditions during the benchmark

5.3.1 - Overall CPU of DB

<FILL IN WITH NODE EXPORTER DATA>

5.3.2 - Overall network of DB over time

<FILL IN WITH NODE EXPORTER DATA>

5.3.3 - …

5.4 - Plotting the full latency spectrum

https://hdrhistogram.github.io/HdrHistogram/plotFiles.html

<FILL IN WITH PLOT IMAGE>