Redis - Number of Keys affect Redis Latency

Background

> “*Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams.*
> ”

We all know that Redis is fast due to data is stored in-memory of the Server. But would the number of Keys matter in latency of response from Redis ? How does Redis work internally? Does it cycle through memory address table in CPU to find your key?

If it is just merely storing variables in memory address then to retrieve the data it will need cycle through the memory address table, and if so it mean that number of keys does matter and we may gain performance by sharding the key into hashes due to smaller number of keys to search through one million keys vs one thousand keys. And hash will allocate certain size of memory hence it would be faster

I am aware that in Redis documentation it is mentioned that time complexity for:

- `SET` is O(1)
- `HGET` is O(1)

Hey, Steve Jobs says stay foolish and stay hungry right? so let's prove it.

Hypothesis

The number of keys affects Redis response latency.

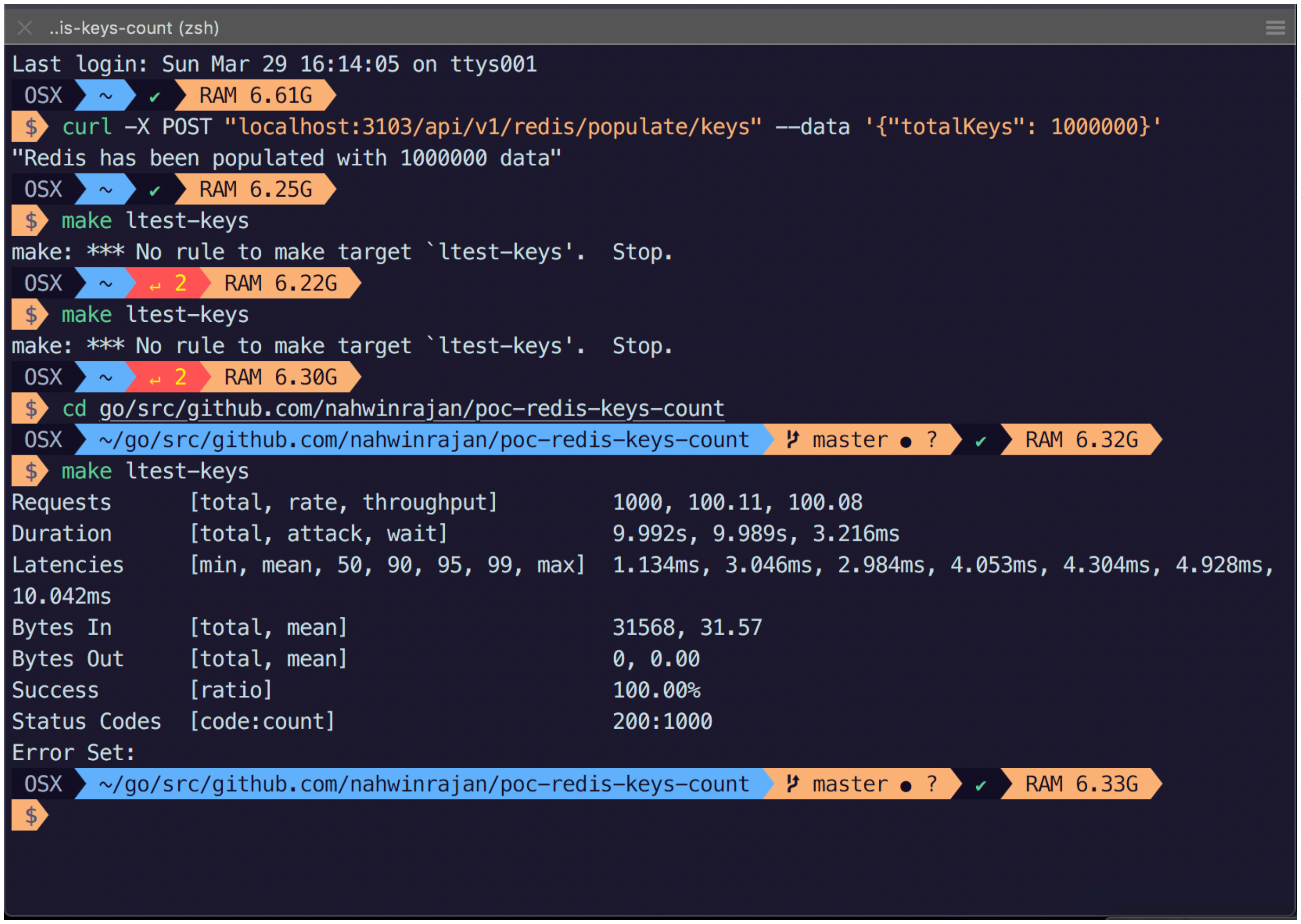
The less we have the faster response from Redis will be.

Specification

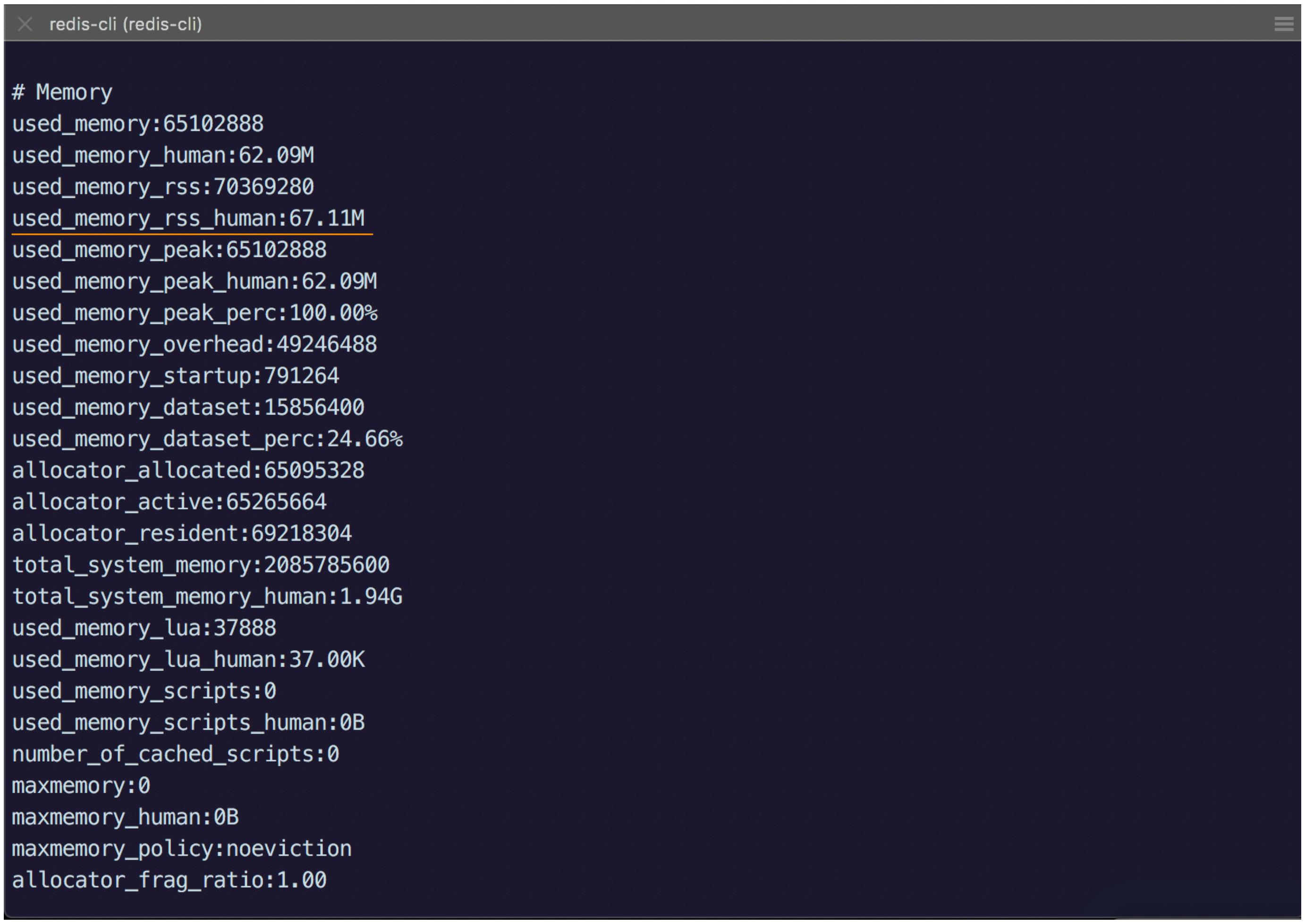
- We'll use `redis:latest` Docker image as our Redis server in our local machine
- We'll only tinker wither following config:
    - `hash-max-zipmap-entries`
        - number of fields in a hash before it is converted into normal hash
        - default to 512
    - `hash-max-ziplist-value`
        - max byte size of field value before the entire hash will be converted to a normal hash
        - default to 64 byte
- We'll use one million of keys as our benchmark and their value will be numerical only
- We'll use Vegeta for load testing and using Golang as our backend server
- Our load test steps will be:
    - Spin up Redis Server and Backend server
    - Populate the Redis with one type of Key
    - Do the load test
    - Flushall data in Redis
    - Spin Down the Redis and Backend server
    - Wait 5 minutes
    - Start over with different Key type
- Backend Repo: https://github.com/nahwinrajan/poc-redis-keys-count

Data

GET

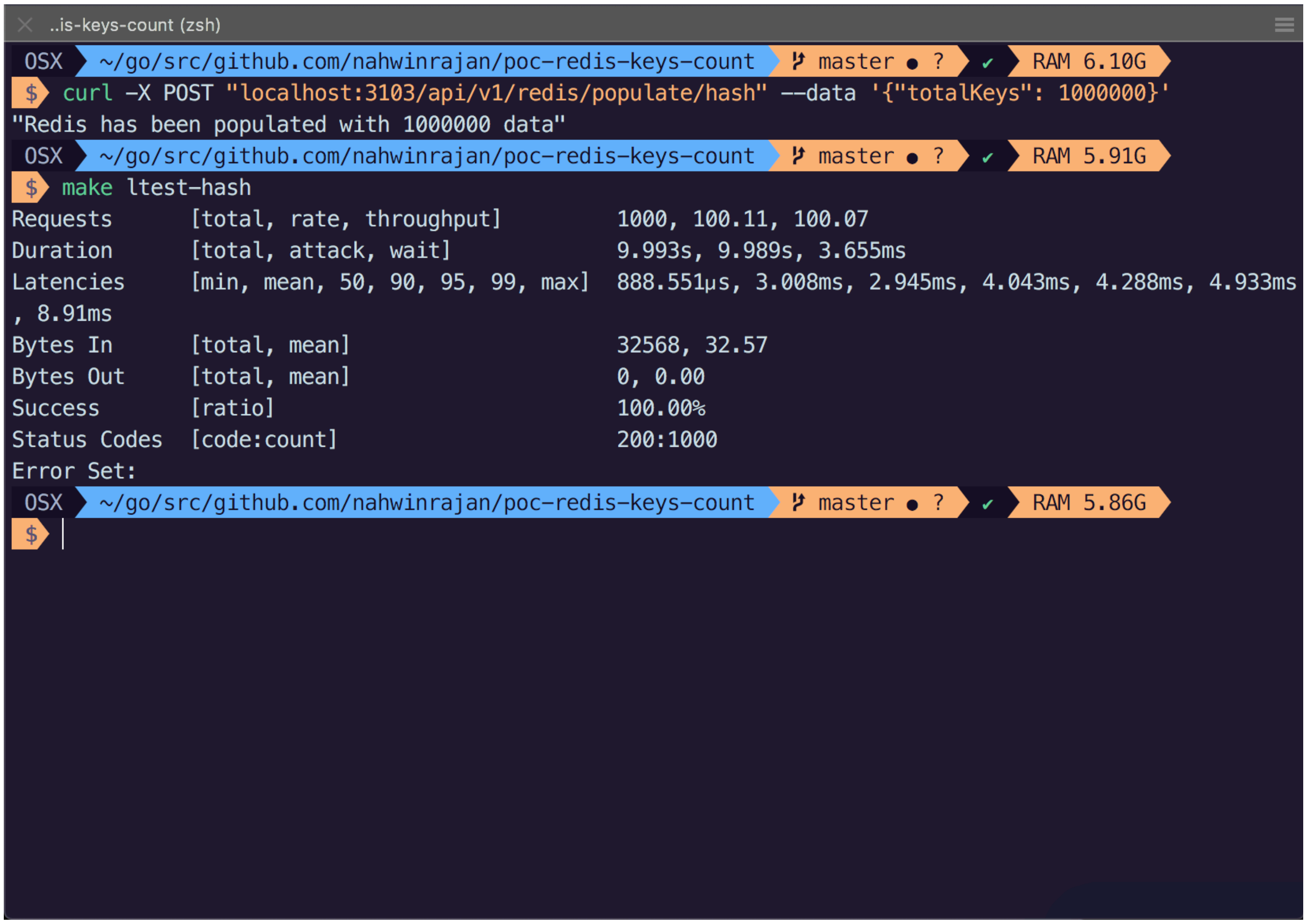


```
..is-keys-count (zsh)
Last login: Sun Mar 29 16:14:05 on ttys001
 OSX  ~  ✔  RAM 6.61G
 $  curl -X POST "localhost:3103/api/v1/redis/populate/keys" --data '{"totalKeys": 1000000}'
"Redis has been populated with 1000000 data"
 OSX  ~  ✔  RAM 6.25G
 $  make ltest-keys
make: *** No rule to make target `ltest-keys'.  Stop.
 OSX  ~  ↵ 2  RAM 6.22G
 $  make ltest-keys
make: *** No rule to make target `ltest-keys'.  Stop.
 OSX  ~  ↵ 2  RAM 6.30G
 $  cd go/src/github.com/nahwinrajan/poc-redis-keys-count
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 6.32G
 $  make ltest-keys
Requests      [total, rate, throughput]         1000, 100.11, 100.08
Duration      [total, attack, wait]             9.992s, 9.989s, 3.216ms
Latencies     [min, mean, 50, 90, 95, 99, max]  1.134ms, 3.046ms, 2.984ms, 4.053ms, 4.304ms, 4.928ms,
10.042ms
Bytes In      [total, mean]                     31568, 31.57
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:1000
Error Set:
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 6.33G
 $
```

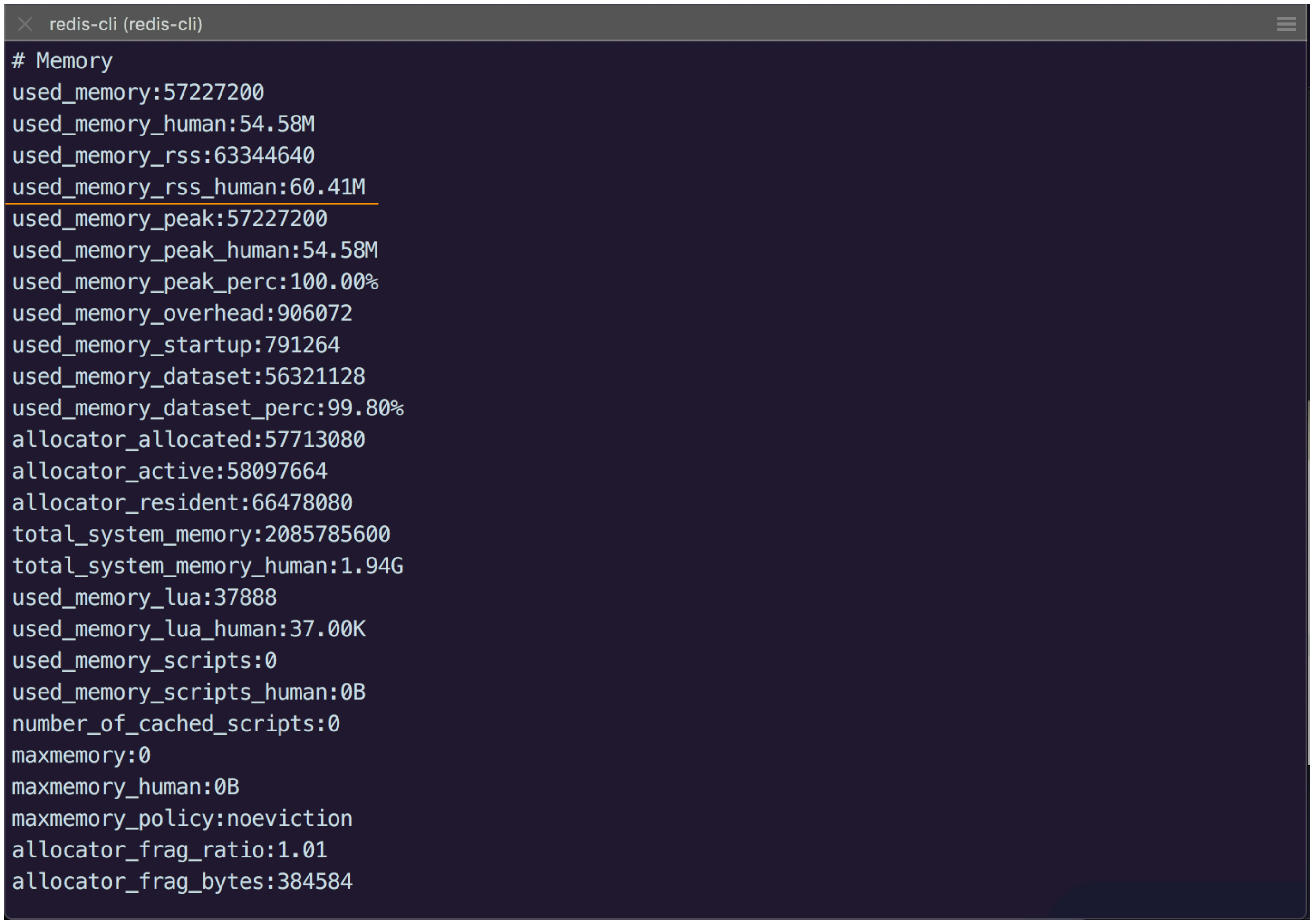


```
# Memory
used_memory:65102888
used_memory_human:62.09M
used_memory_rss:70369280
used_memory_rss_human:67.11M
used_memory_peak:65102888
used_memory_peak_human:62.09M
used_memory_peak_perc:100.00%
used_memory_overhead:49246488
used_memory_startup:791264
used_memory_dataset:15856400
used_memory_dataset_perc:24.66%
allocator_allocated:65095328
allocator_active:65265664
allocator_resident:69218304
total_system_memory:2085785600
total_system_memory_human:1.94G
used_memory_lua:37888
used_memory_lua_human:37.00K
used_memory_scripts:0
used_memory_scripts_human:0B
number_of_cached_scripts:0
maxmemory:0
maxmemory_human:0B
maxmemory_policy:noeviction
allocator_frag_ratio:1.00
```

HGET

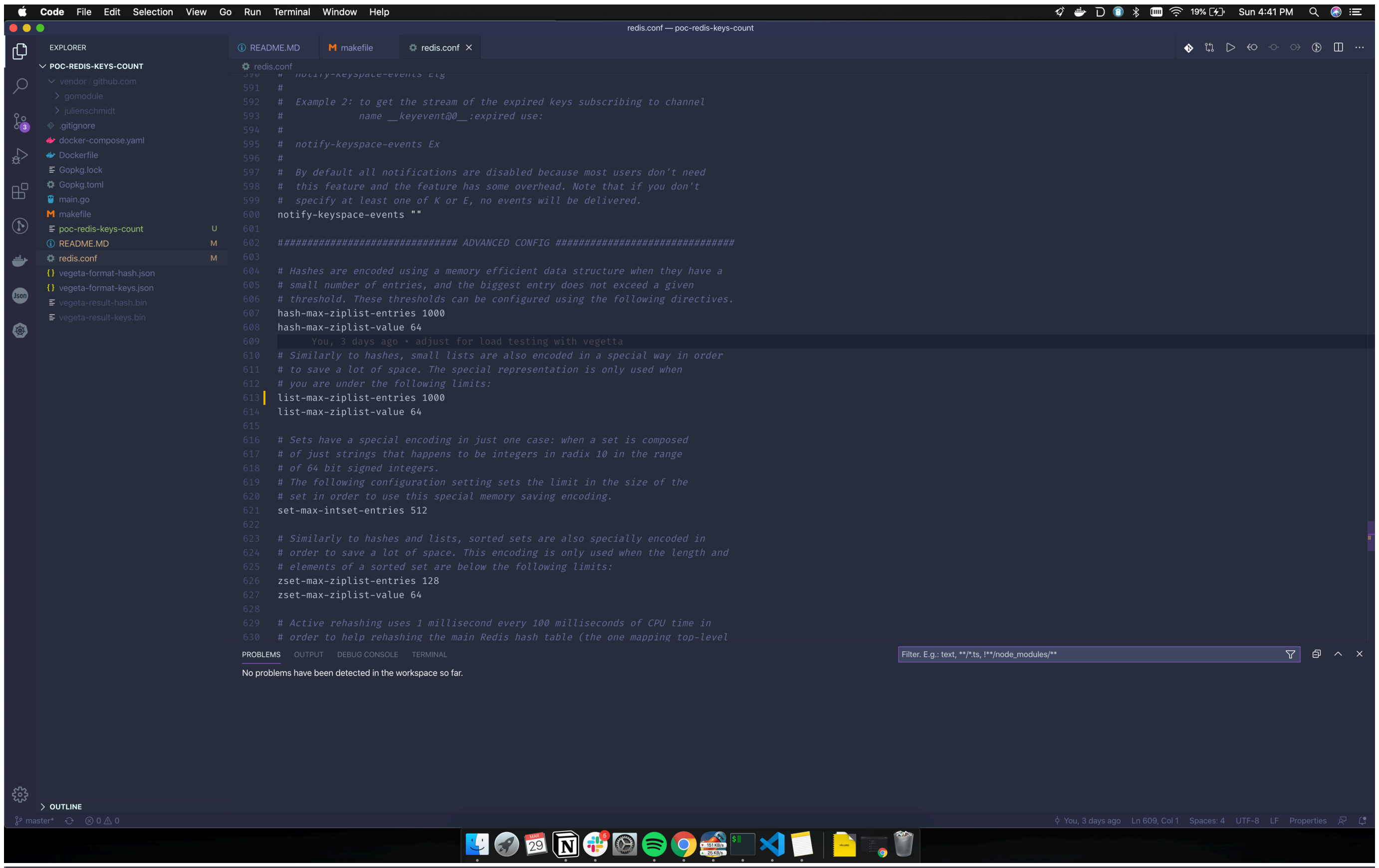


```
..is-keys-count (zsh)
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 6.10G
 $  curl -X POST "localhost:3103/api/v1/redis/populate/hash" --data '{"totalKeys": 1000000}'
"Redis has been populated with 1000000 data"
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 5.91G
 $  make ltest-hash
Requests      [total, rate, throughput]         1000, 100.11, 100.07
Duration      [total, attack, wait]             9.993s, 9.989s, 3.655ms
Latencies     [min, mean, 50, 90, 95, 99, max]  888.551µs, 3.008ms, 2.945ms, 4.043ms, 4.288ms, 4.933ms
, 8.91ms
Bytes In      [total, mean]                     32568, 32.57
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:1000
Error Set:
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 5.86G
 $  |
```



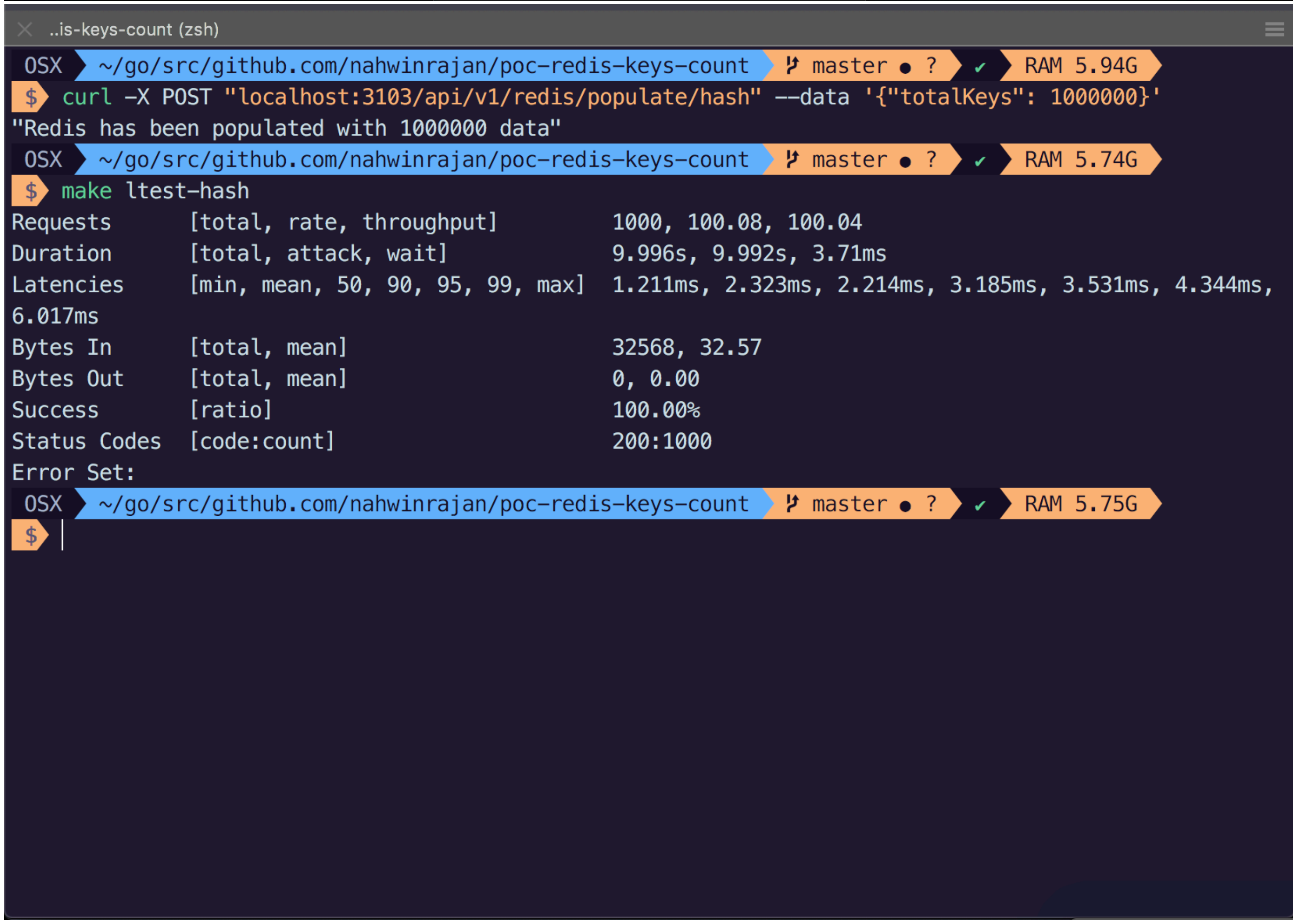
```
# Memory
used_memory:57227200
used_memory_human:54.58M
used_memory_rss:63344640
used_memory_rss_human:60.41M
used_memory_peak:57227200
used_memory_peak_human:54.58M
used_memory_peak_perc:100.00%
used_memory_overhead:906072
used_memory_startup:791264
used_memory_dataset:56321128
used_memory_dataset_perc:99.80%
allocator_allocated:57713080
allocator_active:58097664
allocator_resident:66478080
total_system_memory:2085785600
total_system_memory_human:1.94G
used_memory_lua:37888
used_memory_lua_human:37.00K
used_memory_scripts:0
used_memory_scripts_human:0B
number_of_cached_scripts:0
maxmemory:0
maxmemory_human:0B
maxmemory_policy:noeviction
allocator_frag_ratio:1.01
allocator_frag_bytes:384584
```

HGET with Configured value

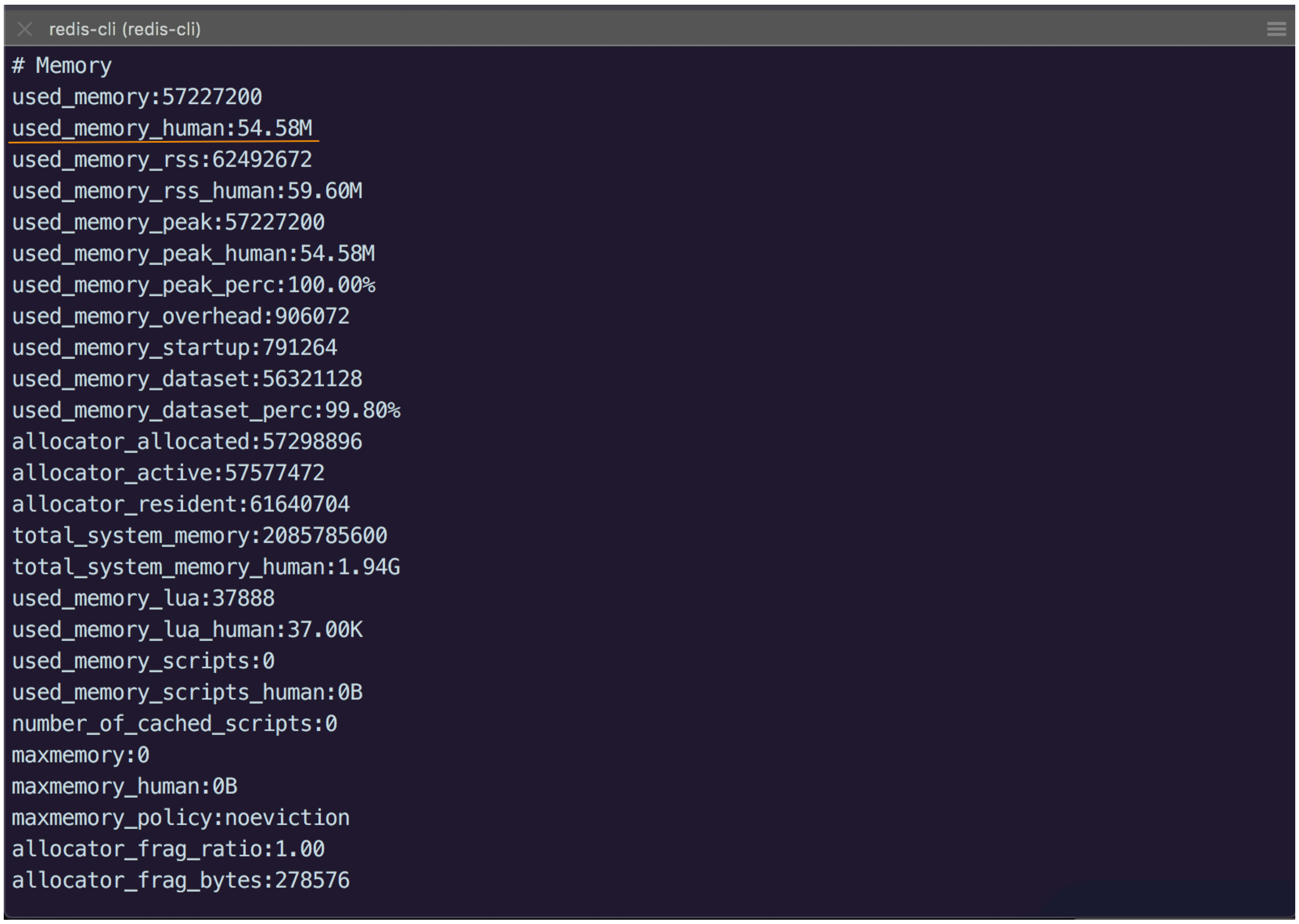


```
#  notify-keyspace-events Elg
#
#  Example 2: to get the stream of the expired keys subscribing to channel
#             name __keyevent@0__:expired use:
#
#  notify-keyspace-events Ex
#
#  By default all notifications are disabled because most users don't need
#  this feature and the feature has some overhead. Note that if you don't
#  specify at least one of K or E, no events will be delivered.
notify-keyspace-events ""

############################### ADVANCED CONFIG ###############################

# Hashes are encoded using a memory efficient data structure when they have a
# small number of entries, and the biggest entry does not exceed a given
# threshold. These thresholds can be configured using the following directives.
hash-max-ziplist-entries 1000
hash-max-ziplist-value 64
      You, 3 days ago • adjust for load testing with vegetta
# Similarly to hashes, small lists are also encoded in a special way in order
# to save a lot of space. The special representation is only used when
# you are under the following limits:
list-max-ziplist-entries 1000
list-max-ziplist-value 64

# Sets have a special encoding in just one case: when a set is composed
# of just strings that happens to be integers in radix 10 in the range
# of 64 bit signed integers.
# The following configuration setting sets the limit in the size of the
# set in order to use this special memory saving encoding.
set-max-intset-entries 512

# Similarly to hashes and lists, sorted sets are also specially encoded in
# order to save a lot of space. This encoding is only used when the length and
# elements of a sorted set are below the following limits:
zset-max-ziplist-entries 128
zset-max-ziplist-value 64

# Active rehashing uses 1 millisecond every 100 milliseconds of CPU time in
# order to help rehashing the main Redis hash table (the one mapping top-level
```

No problems have been detected in the workspace so far.



```
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 5.94G
 $ curl -X POST "localhost:3103/api/v1/redis/populate/hash" --data '{"totalKeys": 1000000}'
"Redis has been populated with 1000000 data"
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 5.74G
 $ make ltest-hash
Requests      [total, rate, throughput]         1000, 100.08, 100.04
Duration      [total, attack, wait]             9.996s, 9.992s, 3.71ms
Latencies     [min, mean, 50, 90, 95, 99, max]  1.211ms, 2.323ms, 2.214ms, 3.185ms, 3.531ms, 4.344ms,
6.017ms
Bytes In      [total, mean]                     32568, 32.57
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:1000
Error Set:
 OSX  ~/go/src/github.com/nahwinrajan/poc-redis-keys-count  master ● ?  ✔  RAM 5.75G
 $
```



```
# Memory
used_memory:57227200
used_memory_human:54.58M
used_memory_rss:62492672
used_memory_rss_human:59.60M
used_memory_peak:57227200
used_memory_peak_human:54.58M
used_memory_peak_perc:100.00%
used_memory_overhead:906072
used_memory_startup:791264
used_memory_dataset:56321128
used_memory_dataset_perc:99.80%
allocator_allocated:57298896
allocator_active:57577472
allocator_resident:61640704
total_system_memory:2085785600
total_system_memory_human:1.94G
used_memory_lua:37888
used_memory_lua_human:37.00K
used_memory_scripts:0
used_memory_scripts_human:0B
number_of_cached_scripts:0
maxmemory:0
maxmemory_human:0B
maxmemory_policy:noeviction
allocator_frag_ratio:1.00
allocator_frag_bytes:278576
```

Conclusion

As we can see from the data that the number of keys does not give a substantial difference with the Redis latency.

Highlights

- If you do `KEYS pattern` in the Redis server you can see that no matter what type of keys it will be placed randomly, and the order will be the same if you do multiple `KEYS pattern` so it is not a matter of `redis-cli` mechanisms.
- Why does memory usage between `HSET` configured and `SET` is bismall ? it was nothing near as described `in-memory optimization` from Redis documentation nor `Redis Usage in Instagram` article?