RedisBloom quick start

For this quick start tutorial, you need:

- Either:
  - A Redis Cloud database with the RedisBloom module
    You can set up a free Redis Cloud database to see the module in action.
  - A Redis Enterprise Software database with the RedisBloom module
  - redis-cli with connectivity to a redis database

Trying it out

You can play with it a bit using redis-cli:

Connect to redis.

```
$ redis-cli -p 12543
127.0.0.1:12543>
```

Run these commands:

```
127.0.0.1:12543> BF.ADD bloom kirk
1) (integer) 1
127.0.0.1:12543> BF.ADD bloom redis
1) (integer) 1
127.0.0.1:12543> BF.EXISTS bloom kirk
(integer) 1
127.0.0.1:12543> BF.EXISTS bloom redis
(integer) 1
127.0.0.1:12543> BF.EXISTS bloom nonexist
(integer) 0
127.0.0.1:12543> BF.EXISTS bloom que?
(integer) 0
127.0.0.1:12543> BF.MADD bloom elem1 elem2 elem3
1) (integer) 1
2) (integer) 1
3) (integer) 1
127.0.0.1:12543> BF.MEXISTS bloom elem1 elem2 elem3
1) (integer) 1
```

You can also create a custom Bloom filter. The BF.ADD command creates a new Bloom filter suitable for a small-ish number of items. This consumes less memory but may not be ideal for large filters. In that case:
Using Cuckoo filters in Redis Enterprise Software

Cuckoo filters can also be used as part of the RedisBloom module. You can play with it using redis-cli:

```
127.0.0.1:12543> CF.ADD cuckoo redis
(integer) 1
127.0.0.1:12543> CF.EXISTS newcuckoo redis
(integer) 1
127.0.0.1:12543> CF.EXISTS newcuckoo notpresent
(integer) 0
127.0.0.1:12543> CF.DEL newcuckoo redis
(integer) 1
```

Debugging Bloom filters

Finally, I added a BF.DEBUG command, to see exactly how the filter is being utilized:

```
127.0.0.1:6379> BF.DEBUG test
1) "size:987949"
2) "bytes:239627 bits:1917011 hashes:14 capacity:100000 size:100000 ratio:0.0001"
3) "bytes:551388 bits:4411101 hashes:16 capacity:200000 size:200000 ratio:2.5e-05"
4) "bytes:1319180 bits:10553436 hashes:19 capacity:400000 size:400000 ratio:3.125e-06"
5) "bytes:3215438 bits:25723497 hashes:23 capacity:800000 size:287949 ratio:1.95313e-07"
```

This outputs the total number of elements as the first result, and then a list of details for each filter in the chain. As you can see, whenever a new filter is added, its capacity grows exponentially and the strictness for errors increases.

Note that this filter chain also uses a total of 5MB. This is still much more space efficient than alternative solutions, since we’re still at about 5 bytes per element, and the uppermost filter is only at about 12% utilization. Had the initial capacity been greater, more space would have been saved and lookups would have been quicker.

You can find more information in:

- Developing with Bloom Filters
- Original Cuckoo Filter paper

Updated: May 13, 2022