FAQ

Why is Redis different compared to other key-value stores?

There are two main reasons.

- Redis is a different evolution path in the key-value DBs where values can contain more complex data types, with atomic operations defined on those data types. Redis data types are closely related to fundamental data structures and are exposed to the programmer as such, without additional abstraction layers.
- Redis is an in-memory but persistent on disk database, so it represents a different trade off where very high write and read speed is achieved with the limitation of data sets that can't be larger than memory. Another advantage of in memory databases is that the memory representation of complex data structures is much simpler to manipulate compared to the same data structures on disk, so Redis can do a lot, with little internal complexity. At the same time the two on-disk storage formats (RDB and AOF) don't need to be suitable for random access, so they are compact and always generated in an append-only fashion (Even the AOF log rotation is an append-only operation, since the new version is generated from the copy of data in memory). However this design also involves different challenges compared to traditional on-disk stores. Being the main data representation on memory, Redis operations must be carefully handled to make sure there is always an updated version of the data set on disk.

What's the Redis memory footprint?

To give you a few examples (all obtained using 64-bit instances):

- An empty instance uses ~ 3MB of memory.
- 1 Million small Keys -> String Value pairs use ~ 85MB of memory.
- 1 Million Keys -> Hash value, representing an object with 5 fields, use ~ 160 MB of memory.

Testing your use case is trivial. Use the `redis-benchmark` utility to generate random data sets then check the space used with the `INFO memory` command.

64-bit systems will use considerably more memory than 32-bit systems to store the same keys, especially if the keys and values are small. This is because pointers take 8 bytes in 64-bit systems. But of course the advantage is that you can have a lot of memory in 64-bit systems, so in order to run large Redis servers a 64-bit system is more or less required. The alternative is sharding.

I like Redis's high level operations and features, but I don't like that it keeps everything in memory and I can't have a dataset larger than memory. Are there any plans to change this?

In the past the Redis developers experimented with Virtual Memory and other systems in order to allow larger than RAM datasets, but after all we are very happy if we can do one thing well: data served from memory, disk used for storage. So for now there are no plans to create an on disk backend for Redis. Most of what Redis is, after all, is a direct result of its current design.

If your real problem is not the total RAM needed, but the fact that you need to split your data set into multiple Redis instances, please read the Partitioning page in this documentation for more info.

Recently Redis Labs, the company sponsoring Redis developments, developed a "Redis on flash" solution that is able to use a mixed RAM/flash approach for larger data sets with a biased access pattern. You may check their offering for more information, however this feature is not part of the open source Redis code base.

Is using Redis together with an on-disk database a good idea?

Yes, a common design pattern involves taking very write-heavy small data in Redis (and data you need the Redis data structures to model your problem in an efficient way), and big *blobs* of data into an SQL or eventually consistent on-disk database. Similarly sometimes Redis is used in order to take in memory another copy of a subset of the same data stored in the on-disk database. This may look similar to caching, but actually is a more advanced model since normally the Redis dataset is updated together with the on-disk DB dataset, and not refreshed on cache misses.

Is there something I can do to lower the Redis memory usage?

If you can, use Redis 32 bit instances. Also make good use of small hashes, lists, sorted sets, and sets of integers, since Redis is able to represent those data types in the special case of a few elements in a much more compact way. There is more info in the Memory Optimization page.

What happens if Redis runs out of memory?

Redis will either be killed by the Linux kernel OOM killer, crash with an error, or will start to slow down. With modern operating systems malloc() returning NULL is not common, usually the server will start swapping (if some swap space is configured), and Redis performance will start to degrade, so you'll probably notice there is something wrong.

Redis has built-in protections allowing the user to set a max limit to memory usage, using the `maxmemory` option in the configuration file to put a limit to the memory Redis can use. If this limit is reached Redis will start to reply with an error to write commands (but will

continue to accept read-only commands), or you can configure it to evict keys when the max memory limit is reached in the case where you are using Redis for caching.

We have detailed documentation in case you plan to use Redis as an LRU cache.

The INFO command reports the amount of memory Redis is using so you can write scripts that monitor your Redis servers checking for critical conditions before they are reached.

Background saving fails with a fork() error under Linux even if I have a lot of free RAM!

Short answer: `echo 1 > /proc/sys/vm/overcommit_memory` :)

And now the long one:

Redis background saving schema relies on the copy-on-write semantic of fork in modern operating systems: Redis forks (creates a child process) that is an exact copy of the parent. The child process dumps the DB on disk and finally exits. In theory the child should use as much memory as the parent being a copy, but actually thanks to the copy-on-write semantic implemented by most modern operating systems the parent and child process will *share* the common memory pages. A page will be duplicated only when it changes in the child or in the parent. Since in theory all the pages may change while the child process is saving, Linux can't tell in advance how much memory the child will take, so if the `overcommit_memory` setting is set to zero fork will fail unless there is as much free RAM as required to really duplicate all the parent memory pages, with the result that if you have a Redis dataset of 3 GB and just 2 GB of free memory it will fail.

Setting `overcommit_memory` to 1 tells Linux to relax and perform the fork in a more optimistic allocation fashion, and this is indeed what you want for Redis.

A good source to understand how Linux Virtual Memory works and other alternatives for `overcommit_memory` and `overcommit_ratio` is this classic from Red Hat Magazine, "Understanding Virtual Memory". You can also refer to the proc(5) man page for explanations of the available values.

Are Redis on-disk-snapshots atomic?

Yes, Redis background saving process is always forked when the server is outside of the execution of a command, so every command reported to be atomic in RAM is also atomic from the point of view of the disk snapshot.

Redis is single threaded. How can I exploit multiple CPU / cores?

It's not very frequent that CPU becomes your bottleneck with Redis, as usually Redis is either memory or network bound. For instance, using pipelining Redis running on an average Linux system can deliver even 1 million requests per second, so if your application mainly uses O(N) or O(log(N)) commands, it is hardly going to use too much CPU.

However, to maximize CPU usage you can start multiple instances of Redis in the same box and treat them as different servers. At some point a single box may not be enough anyway, so if you want to use multiple CPUs you can start thinking of some way to shard earlier.

You can find more information about using multiple Redis instances in the Partitioning page.

However with Redis 4.0 we started to make Redis more threaded. For now this is limited to deleting objects in the background, and to blocking commands implemented via Redis modules. For future releases, the plan is to make Redis more and more threaded.

What is the maximum number of keys a single Redis instance can hold? and what is the max number of elements in a Hash, List, Set, Sorted Set?

Redis can handle up to 2^{32} keys, and was tested in practice to handle at least 250 million keys per instance.

Every hash, list, set, and sorted set, can hold 2^{32} elements.

In other words your limit is likely the available memory in your system.

My slave claims to have a different number of keys compared to its master, why?

If you use keys with limited time to live (Redis expires) this is normal behavior. This is what happens:

- The master generates an RDB file on the first synchronization with the slave.
- The RDB file will not include keys already expired in the master, but that are still in memory.
- However these keys are still in the memory of the Redis master, even if logically expired. They'll not be considered as existing, but the memory will be reclaimed later, both incrementally and explicitly on access. However while these keys are not logical part of the dataset, they are advertised in INFO output and by the DBSIZE command.
- When the slave reads the RDB file generated by the master, this set of keys will not be loaded.

As a result of this, it is common for users with many keys with an expire set to see less keys in the slaves, because of this artifact, but there is no actual logical difference in the instances content.

What does Redis actually mean?

It means REmote DIctionary Server.

Why did you start the Redis project?

Originally Redis was started in order to scale LLOOGG. But after I got the basic server working I liked the idea to share the work with other people, and Redis was turned into an open source project.

How is Redis pronounced?

It's "red" like the color, then "iss".

This website is open source software. See all credits.

Sponsored by redislabs HOME OF REDIS