Engineering

A PRELUDE TO ANALYSIS OF REDIS MEMORY-STORE

NOVEMBER 28, 2021 BY ROMAN GERSHMAN

During the last 13 years, Redis has become a truly ubiquitous memory store that has won the hearts of numerous dev-ops and software engineers. Indeed, according to StackOverflow survey in 2021, Redis is the most loved database for the 5th time in a row and is at the top of db-engines ranking, way before the next contestant. But how well does Redis utilizes modern hardware systems? Will it stay competitive in a few years without reinventing itself?

To understand choices behind Redis design, I have been reading the old posts of Salvatore @antirez - the creator of Redis. Before I begin, I want to add that I have tremendous respect for Salvatore and for how much he achieved by being a talented and authentic software programmer (even though he does not want to be remembered as such).

Based on his notes and GitHub discussions, I identified the following architectural principles in Redis:

1. **Simple is beautiful (and code is a poem)**
   Probably the strongest motive around Redis is simplicity. Salvatore's preference is towards simple solutions and he expressed his attitude to coding in his Redis manifesto. As a consequence, Redis resides in a single self-contained codebase without much reliance on third-party projects and its functionality is implemented in plain C using posix API.

2. **Single-threaded architecture**
   Redis development started a few years after Memcached. By that time, Memcached, the predecessor of Redis, has already been a mature system that has been used and supported by large, highly technological companies like Facebook and Twitter. It used multi-threaded architecture to scale its I/O performance vertically within a single node. Remarkably, antirez decided against this architecture and adopted the single-threaded design instead. Specifically, Redis utilizes a single thread that manipulates its main in-memory dictionary. antirez has defended this approach many times with the following arguments:

   - Redis cares very much about latency and adopts share-nothing architecture to control its tail and average latencies.
   - Most of the CPU spent on system-cpu handling I/O and not on userland cpu handling Redis data-structures. Therefore, the upside of parallelization is limited anyway.
   - Pipelining of requests can increase throughput by order of magnitude.
   - On the other hand, multi-threading adds complexity. Quoting antirez: "… Slower development speed to achieve the same features. Multi-thread programming is hard… In a future of cloud computing, I want to consider every single core as a computer itself…"
   - Vertical scale has physical limit anyway, therefore horizontal scaling (aka Redis cluster) is the way to scale.

In addition to the principles above, Redis maintains unique design goals that differentiate it from, say, a disk-based database. I believe that if we list Redis design goals by priority, it will be:

1. Low latency
2. High throughput
3. Memory efficiency
4. High availability
5. Strong consistency guarantees
6. Durability

Obviously, if one would want to implement an alternative memory store he would need to prioritize design goals similarly to Redis. In other words, the new store design should not sacrifice low latency for durability, or for strong consistency.

Retrospective

I think that *simplicity* was the main guideline for Redis which heavily affected architectural decisions like its serialization algorithm, efficiency of its data structures, reliability and more. Even the choice of its threading model has been, in part, done because of simplicity reasons. And if Redis is the experiment on how far one can go nowdays by implementing relatively simple solutions, it without a question succeeded tremendously.

Redis in 2021 is a mature product with relatively stable feature set, and the questions I am asking myself today are: a) how more efficienty Redis could be today if it would adopt state-of-the-art algorithms and datastructures. b) how much simpler it would be for a user if it adopt product simplicity over simplicity of implementation.

In other words, if one would want to implement a drop-in replacement for Redis **today** by redesigning it from scratch, how it would compare to Redis. I do not have a definite answer to this question today but hope to have one in few months.

Again, I am not arguing with the tremendous popularity of the Redis memory store. It seems that Salvatore's decision to go for simplicity and deliver features quickly in Redis early days - paid off: today Memcached is a niche system, and the vast majority of software stacks use Redis. However, I do claim (currently without proof) that it is possible to **vastly** improve reliability, performance and cost-efficiency metrics of Redis-like memory store that follows similar design goals but with different architectural principles.

In my next posts, I am going to detail Redis specific design choices based on the principles state above and show how a different architecture, if aligned better with modern hardware systems, could provide, what I think a disruptive change to in-memory datastores.