#zenantirez

Design tradeoffs in the Redis project

Salvatore Sanfilippo
@antirez — Pivotal

Before: some history…

- Software is about humans, not just engineering
- The history of an OSS project changes its design

Too much time in my hands

- Creating a startup as a side project is hard.
- OSS is no exception, it is like a startup.
- You need a lot of time and focus.
- Good moments? After you graduate, or when your previous thing provided a *money buffer*.

To be alone…

- It’s hard to find a co-founder in an “pure” OSS project.
- To be alone is hard: accountability, self-motivation.
- To be alone is fun: single-minded design.
- To be alone narrows your POV: use the community.

Design to scale yourself

- Complex software is rarely a good idea…
- Far worse if you want to maintain, often alone, a non trivial project.
- Sometimes complexity is intrinsically required: too simple is also not optimal.
- It is critical (and hard) to find a good balance.

#zenantirez

Important contribs? Mostly from payed people

- Peter Noordhuis (VMware)
- Matt Stancliff (Pivotal)
- Important fixes / debugging / contribs from RedisLabs, Weibo, Pinterest, Citrusbye.
- There are exceptions, of course (example: IPv6 support, high quality support in maling list).

Stay motivated

- Rule 0: Do things you want to see existing.
- Rule 1: Evolve by sub-projects that are *interesting to you*.
- Rule 2: Create an economically sustainable model.
- Rule 3: Build a successful OSS, or abandon and try again.

Design tradeoffs

To design means to sacrifice something

Copy on write based persistence

Sacrifice memory

for predictability and design freedom

Append-only disk access

Sacrifice incremental updates
to avoid disk seeks and corruptions

Single thread model

Sacrifice single process scalability
for big wins in simplicity and reliability

(spoiler: you need to scale cross-process, anyway…)

Asynchronous replication

Sacrifice safety

for two order of magnitude more performances

(but it’s not a black-or-white tradeoff: we now have async ACKs from slaves)

No nested data structures

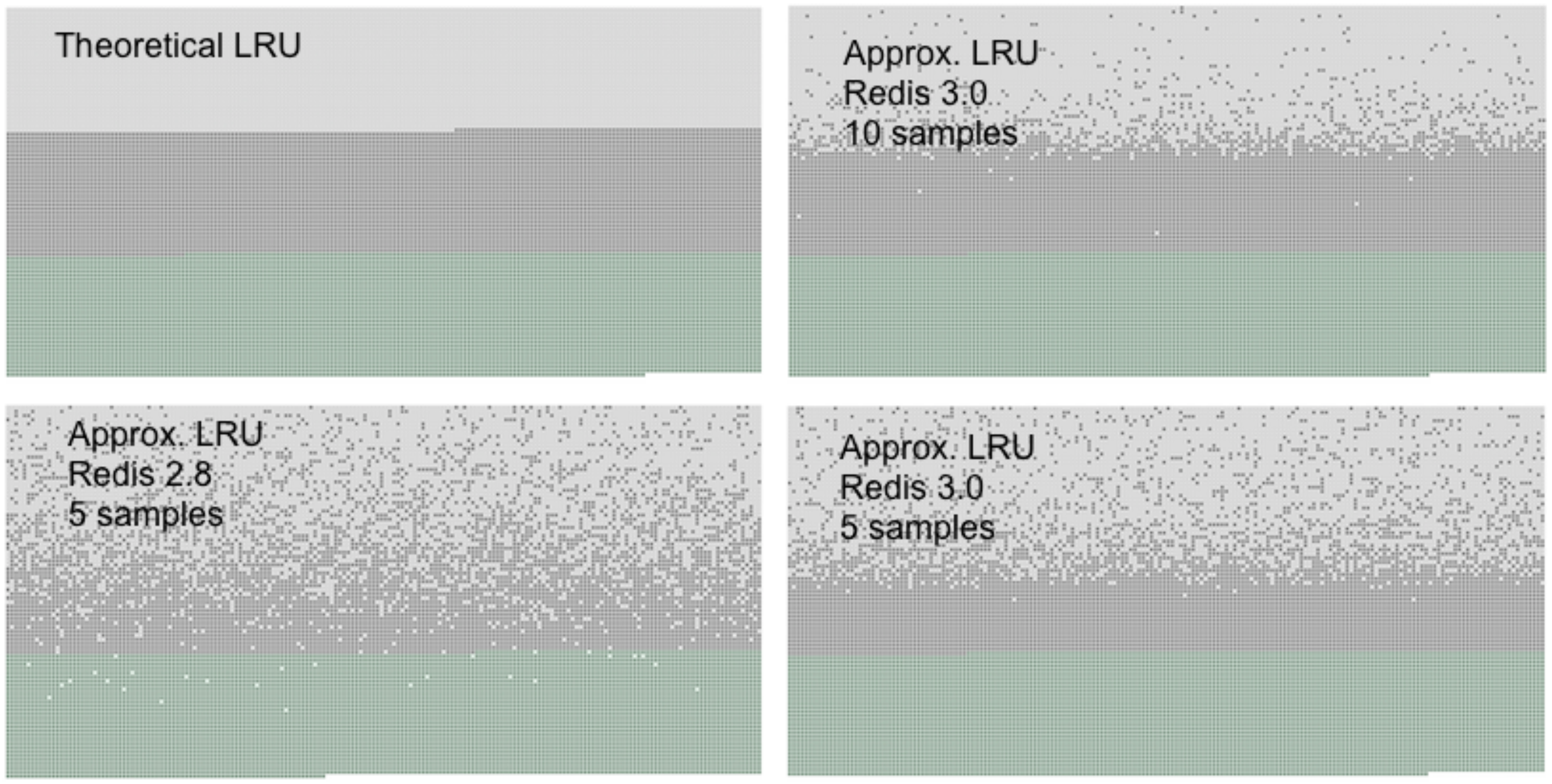
Sacrifice expressiveness

for data model simplicity and simpler sharding

Approximated LRU

Sacrifice theoretical correctness
to save memory

We managed to improve it lately…



No stored procedures

Sacrifice least surprise principle
for a simpler to manage system

Redis Cluster non trivial client role

Sacrifice separation of concerns for better latency and simplicity

Redis Cluster consistency model

Sacrifice safety
for data model freedom

(but limiting the danger when shit happens)

Ask me anything!

Now or later…
I’m @antirez on Twitter.