A short-term plan for Redis

@antirez - Pivotal
Redis is made of pieces

Transactions

Replication

API

Scripting

Sentinel

Networking

Pub/Sub

CLI

Storage

Cluster

Persistence
Evolution

- Redis can be analyzed as separated components. Most of them are modular.
- Evolution: adding or removing components.
- Evolution: altering existing components.
Mem storage

What it is.

- It organizes data into memory.
- Files: `dict.c`, `ziplist.c`, `zipmap.c`, `adlist.c`, `intset.c`, `skiplist` implementation.
- **Effects**: memory usage, cache locality, API
- **Last changes**: Redis object embedded string.
Mem storage

Possible evolution.

• Unrolled linked lists.

• Compressed Redis objects (Hi HTML!).

• Key space iterator (Hi Pieter!).
Persistence

What it is.

- Dumps and loads RDB / AOF data on disk.
- Files: `rdb.c`, `aof.c`.
- **Effects**: durability, replication, startup speed, on-disk space efficiency.
- **Last changes**: COW memory reporting, CRC64, verbatim zipped values...
Persistence
Possible evolution.

- AOF and RDB format (not scope!) unification.
- **Gain**: Faster AOF rewrites and reloads, One format is better than two.
- dump.rdb, aof.rdb
Replication

What it is.

- Asynchronous replication, finally able to incrementally resynchronize.

- Files: `replication.c`.

- **Effects**: durability, memory usage, availability, consistency.

- **Last changes**: PSYNC, Slave ACKs, deny writes when sensing less than N slaves.
Replication

Possible evolution.

- Synchronous replication.
- **Gain**: Consistency, a cluster nearest to a CP system.
Replication

Discarded evolution

- SYNC via AOF.

- In theory, you could avoid to create the RDB, and feed the AOF file if enabled.

- **Discard reason**: AOF rewrite requires to dump anyway. Systems that can’t cope with slaves? This is the symptom not the illness.

- Also: PSYNC makes full resyncs less likely.
Transactions

What it is.

• Isolated execution of a group of commands.

• Files: multi.c.

• Effects: API, persistence, replication, scripting.

• Last changes: Refactoring only.
Transactions

Discarded evolution.

- **Remove MULTI/EXEC** since there is scripting.

- **Discard reason**: transactions are composable, don’t need to be fast (only linearizability, no serializability), good API building block for new features.

- There are a total of 19 commits on multi.c.
Pub/Sub

What it is.
• Fire and forget style Publish / Subscribe.
• Used for notification of internal events.
• Files: pubsub.c, notify.c.
• Effects: Events API, external tools bus, messages reliability.
• Last changes: Pub/Sub in Redis Cluster, Notification API.
Pub/Sub

Possible evolution.

- HPUBLISH chan msg history_len
- Snowflake-alike unique IDs for every message.
- API to subscribe & get history.
- **Gain**: Reliable Pub/Sub.

API IS JUST AN EXAMPLE :-(
Cluster

What it is.

- Automatic partitioning and failover.
- Files: `cluster.c`.
- **Effects**: Consistency, Resharding speed, Usefulness.
- **Last changes**: Complete implementation. Use of proper algorithms.
Cluster

Possible evolution #1.

- Non blocking MIGRATE.
- Semi-automatic resharding (currently it is assisted by redis-trib for every key moved).
  
  - **Gain**: Faster rehashings with less impact on latency / availability.
Cluster

Possible evolution #2.

- Redis Cluster as an highly available AP store? (as an optional mode).
- The design is compatible with this idea.
- Type-based merge semantics.
- **Gain**: Ability to serve different use cases where availability is the first concern but values are small.
API

What it is.

• The set of exported commands.

• Short term plan: avoid bloating it, no new data types or command if not very general.

• Except for the iterator.

• Scripting is helping a lot (big adoption!).
Scripting

What it is.

- Server side execution of Lua scripts.
- Files: `cluster.c`.
- **Effects**: Speed, Applicability.
- **Last changes**: Replication of EVALSHA when possible.
Scripting

Possible evolution.

• **Speed!**

• Currently we dispatch Redis calls from Lua via the normal command execution path.

• What we can do: write direct implementations of notable commands.

• **Gain**: Reduce the execution time of scripts.
CLI

What it is.

- redis-cli command, basically.
- Files: redis-cli.c.
- **Effects**: User experience, observability, debugging.
- **Last changes**: --stat, --bigkeys, --pipe, --latency-history.
CLI

Possible evolution.

• Better way to test scripts: multi line editing, call scripts by name, ...

• Commands expansion. Example:
  TYPE `RANDOMKEY`

• Better Redis Cluster support.

• Simplify working with many instances.
Stats and reporting

What it is.

- **INFO, Slow log, Watchdog, MONITOR.**
- **Files:** `replication.c`, `slowlog.c`, `debug.c`, `redis.c`.
- **Effects:** Observability, debugging, monitoring.
- **Last changes:** None important recently.
INFO

Possible evolution

• INFO is pretty bad: requires parsing, is slow. We need backward compatible changes :-(

• Proposal: tree alike properties.

INFO memory.used # get single field
INFO replication.slave.0.lag
INFO memory # today output
Redis Doctor?

Possible evolution

• Check latency of many operations.

• Store metrics as time series.

• Be able to tell the user if there are problems.

• redis> DOCTOR

Probably disk is too slow:
45 recently delayed fsync()
RDB saving time 2 mb/sec
Sentinel

What it is.

- Automatic failover and monitoring.
- Files: sentinel.c.
- Effects: Availability, durability.
- Last changes: Beta implementation.
Sentinel

Possible evolutions.

• Sentinel is here to stay, but needs changes.
• Use Redis Cluster algorithms (versioned changes).
• Use persistent state like Redis Cluster.
• Or... just use Redis Cluster itself? Only enabling monitoring and failover.
Thanks!
for your attention

• Ask any question, there are no stupid ones.
• What changes you like most, what do you think is a bad idea?
• What changes do you propose?