Redis Database Analyses

20. 10. 2016

Written by

Florian Sander
florian.sander@student.hs-hannover.de

Nikolai Böker
nikolai.boeker@student.hs-hannover.de

Danar Armin
danaramin5889@gmail.com

Sana Lakhal
sana.lakhal@student.hs-hannover.de

Sascha Becker
s.becker@wertarbyte.com

Contents

1 Data Model

Redis is a NoSQL database with an aggregate orientation and is likewise key-based. This means that keys identify the assigned value, this way a client can extract the value by knowing its key. Further you can put a value for a key or delete a key and likewise its value. Redis keys are binary safe which means any binary sequence can be used as a key. An empty string is also a valid key. But Redis is not a plain key-value store. Because a key is binary safe the value can hold more complex data structures. By complex is meant the possibility to nest values in values or even map objects.

1.1 Data Structures

Redis supports the following data structures:

- Strings: a String-value can have a total length of 512 Mb, assembling Strings
- Lists: simply an ordered list of Strings => a sequence with duplicates, adding new elements on the head (left) or on the tail (right), max length of a list is 232 (around 4 billion of elements per list
- Hashes: maps between String-fields and String-values, mainly represent objects, have a max length of 232
- Sets: an ordered collection of Strings, not allowing duplicates (if u add the same element multiple times will result in having just a single copy of this element), max number of elements is the same as lists, you can apply commands or set operations
- Sorted Sets: similar to Set but associated with a score ordering the sorted set from the smallest to the greatest score, elements are unique but scores may be repeated
- Bitmaps and HyperLogLogs: basing on Strings with their own semantic Message Queues

You can run atomic operations on each of these data types. With SET you set a specific value for a key and with GET you retrieve this value. SET replaces any existing value already stored into a key in the case this key already exists. This way SET can be used to update a value of an existing key. Because the value of the key is overwritten with a new value, the old value is kind of deleted. To delete a key with its associated value you run the command DEL. Redis offers the possibility to set or retrieve the value of multiple keys in a single command using MSET and MGET. MGET returns an array of values.

Redis supports around 50 programing languages. 12 languages are particular recommended. For Java there are 4 different libraries. A few examples for different programming languages in the following are JRedis for Java , Redis-rb for Ruby, Credis for C and Redis-py for Python.

2 Query Support

2.1 Supported query types, i.e. point, range, navigation, and/or arbitrary?

Redis supports different query types. Redis supports point and range queries for: strings, hashes, lists, sets, sorted sets. A string for example can be accesses via GET <key> and multiple Strings can be accesses, to name one possibility, for example with the command: MGET <key> [<key>...]. Geospatial indexes can even be searched by radius queries. That means Redis checks if the geospatial indexes are within a specific radius of a defined position. Arbitrary access of entries is also possible by getting a random key with the command RANDOMKEY. In comparison to plain key-value stores, redis is able to store complex data structures and perform for the data structure specific commands. So for example in redis it is possible in a list to prepend one or multiple values with the command: LPUSH <key> <value> [<value>...]. Another mentionable functionality of redis is that keys can expire. Timeouts can be set on keys and after the timeout has expired, the key will automatically be deleted. The related command for expiring a key is the command: EXPIRE <key> seconds.

2.2 What is the query language of the system? Is it declarative, functional, algebraic and/or imperative?

In comparison to normal SQL Redis doesn‘t have a delarative query language. The query language of Redis is imperative. All queries in Redis are made with commands that can‘t be modified (apart from source code modifications in Ansi-C), except for the arguments, that can be passed into the command.

2.3 Are queries automatically optimized?

There is no automatic command optimization in Redis. One compareable approach in Redis is the Slowlog. Redis loggs queries that exceed a specific, configureable execution time. The execution time is just the time actually needed to execute the command, not included in this case are for example the I/O operations or the communication with the client. With the Slowlog there is a chance to identify slowly executing commands. The commands themselfs aren‘t automatically optimized.

3 Indexes

In Redis different kinds of indexes can be created. The following indexes can be created using sorted sets:

3.1 Simple Numerical Indexes

- Simplest index that can be created in Redis
- Numerical value can be indexed
- Data type of sorted set is used which represents a set of elements
- Elements are ordered by a so-called score
- Scores are in ascending order
- Updating this kind of index must be done manually
  - Adding back again an element with a different score and the same value
  - Two operations (commands) needed:
    1. Updating the data in the hash that representing the element (HSET)
    2. Updating the data in the index (ZADD)

3.2 Lexicographical Indexes

- In Redis when elements with the same score are added into a sorted set, then they are ordered lexicographically
- Strings are indexed
  - Strings are compared as binary data
  - Elements are sorted by the raw values of their bytes (byte after byte)
- Lexicographical indexes are managed manually
  - One possibility to update index values is to take a hash in addition to the sorted set; the hash maps the object ID to the current index value
  - In case of removing old information were indexed the hash value has to be retrieved by object ID and then the information can be removed from the sorted set (command: ZREM)

3.3 Composite Indexes

- Are used when indexes are created by using multiple fields
- Composite indexes can also be used in order to represent graphs by using the so-called Hexastore data structure
  - By using Hexastore relations between objects can be represented
  - An element (relation) must be stored in a lexicographical index

4 Persistence

Redis comes with a different range of options when it comes to persistence.

4.1 RDB

The RDB persistence performs point-in-time snapshots of your dataset at specified intervals. Due to its compact size it is perfect for backups. Store those single files on a seperate server or in the cloud e.g. Amazon S3. In case of a disaster you can easily restory a different version of the data set. Restarting your service with RDB will be faster since it is only a single file that must be loaded.

On the downside you have to live with data being lost when restoring a state. Due to the RDB timed saving interval mechanic data missing a savepoint will be lost once something bad happens. To reduce the amount of data being lost minimizing the time interval will create backups more frequently. Everytime RDB has to persist something on the disk it is using a child process and starts to fork(). This can be time consuming if you are working with a big dataset.

4.1.1 Snapshotting

So called „in-memory“ meaning Redis holds the „Dictionary“ in the RAM and stores it onto the disk at predetermined intervals. The intervals to store data onto the disk can be configured by define the number of writing operations and a time limit.If the systems crushes the past operations can be reloaded to retrieve the primary state of the data.

4.2 AOF

Instead of saving a complete dataset AOF logs your actions and reconstruct your dataset when started. This log is append only which prevents corruption problems if there is a power outage. If the log contains a half written action the provided redis-check-aof tool will be able to fix that. The log will grow for sure. Every action will be appended. To prevent a big logfile redis will rewrite it frequently. AOF will minimize the amount of actions needed to restore your dataset. If the second file is ready refactoring it will switch it with the original and starts appending onto the new one. Further more the log file is human readable. In case of a disaster you can check the log for problems and edit them directly. Restart and AOF will use your edited lines.

On the downside the AOF files are usually bigger than a RDB file for the same dataset. Depending on the used fsync policy AOF can be slower than RDB.

4.2.1 Append-only file

Every writing operation is stored onto the disk immediately.

4.3 Combine AOF and RDB

You can combine those two options for a hybrid solution. In this scenario redis will use the AOF file when restarting to reconstruct the original dataset because it is the most complete.

4.4 No persistence

If you are risky disable persistence completely. Just use redis as is. There will be no backup or restoring features with this option.

4.5 What's the best option now?

It depends on your requierements. If you want a degree of data security comparable to what postgreSQL provides choose the hybrid option. If your data is very delicate but living with a few minutes of data loss is acceptable choose RDB alone. Choosing AOF alone is something you should prevent. Taking a RDB snapshot form time to time is a must have if you seek for backups.

5 Transactions and Concurrency Control

Redis support transactions. A basic transaction is a sequence of multiple commands. One client can execute multiple commands without other clients being able to interrupt them. In Redis the foundation of transactions are following commands: MULTI, EXEC, DISCARD and WATCH, which allow execution of a group of commands in a single step. The basic usage is to enter MULTI, enter multiple queries and enter EXEC for executing the multiple commands or DISCARD for flushing the transaction queue. Every command passed as a part of a basic MULTI/EXEC transaction is executed one after another until they have completed. After completion another client may execute their commands. The WATCH command is used to provide a check-and-set behaviour to transactions. This form of locking is called optimistic locking (which is an Optimistic Concurrency Control, short: OCC approach).

Transaction guarantees:

- all commands in a transaction are serialized no request by other client is served in the middle of the execution of a Redis Transaction guarantees isolation
- all commands are processed or none guarantees atomic transactions

A possible error before a transaction could be that the command maybe failed to queue due to wrong syntax of the command or e.g. memory errors. Another eventuality to produce an error after a transaction is executed could be to execute a command against a key with a wrong data structure (all data structures have specific commands and the actual values are unknown for the system). Even though a command might fail during execution, the execution of the following commands will be processed and not rolled back like in some relative database query languages.

Pessimistic Concurrency Controls (PCC) in distributed systems, with distributed locks are also possible with Redis, but not with the commands in the Redis query language itself. If necessary, the “Redlock” algorithm can be integrated in diverse programming languages.

6 Partitioning

Splitting redis into multiple instances is a process called partitioning. Every instance will only contain a small subset of the keys.

Partitioning servers two main goals:

- The Database can be much larger because using multiple computers merged into one is more powerfull than using a single instance.
- Scaling becomes reality. Giving control over how much cores and isntances used in your cluster is a must for bigger systems.

6.1 Types

There are several ways to split up your keys and route your users to what they need.

6.1.1 Range Partitioning

6.1.2 Hash Partitioning

6.1.3 Consistent Partitioning

6.2 Where and how implemented?

6.3 Disadvantages