Redis Database Implementation

05. 01. 2017

Written by

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

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

Danar Armin
danaramin5889@gmail.com

Sascha Becker
s.becker@wertarbyte.com

Contents

1 Introduction

The document starts with a basic introduction to Redis and is followed by a description of the scenario which is set. Afterwards a detailed plan is developed. In the end all solutions are presented with a final recommendation.

1.1 The basics

Redis is an aggregated oriented key/value store belonging to the NoSQL databases. Redis stores values with keys, where keys identify the assigned value. For extracting a specific value the key must be known. Redis keys are binary safe, which means any binary sequence can be used as a key. Also an empty string is a valid key. In Redis keys can expire. After a determined time the key is deleted automatically. Redis enables the possibility to store even complex data structures by nesting values into values or mapping objects. Redis supports the following data structures:

- Strings
- Lists
- Hashes
- Sets
- Sorted Sets
- Bitmaps
- Message Queues

Redis provides two different storage mechanisms: Snapshotting and Append Only File Mode (AOF). A “Snapshot” holds all data in the memory. Therefore it is called “in-memory”. The data is stored onto the hard disk at a predetermined interval. These intervals can be configured by defining the number of writing operations and a time limit. Reloading the past operations to retrieve the primary state of the data after a system crash is an advantage of this method. With “AOF” every writing operations is stored onto disk immediately.

When storing strings to a Redis database the command “SET” is used. This way a value is placed to a new key. “SET” overwrites values of existing keys. So “SET” updates the value of existing keys. For retrieving the value “GET” is run. For each data type shown above exists different commands, which operates more or less the same way. Redis does not have a declarative query language. All queries in Redis are based on these commands. They can not be modified, except for the arguments, that can be passed over a command to another. The internal use of these commands is imperative. Having several single commands they can be combined to a single atomic transaction. For this Redis provides pipelining. If a command fails the whole transaction also fails. There are two transaction guarantees: