ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

HAROKOPIO UNIVERSITY

Τμήμα Πληροφορικής και Τηλεματικής

Συστήματα Διαχείρισης Δεδομένων Μεγάλης Κλίμακας

Παρουσίαση Συστήματος

Ιωάννης Παπαδόπουλος

ΑΜ: 20126

Ακαδημαϊκό έτος: 2020-2021

History

REmote DIctionary Server

Originally developed: by **Salvatore Sanfilippo**
Main aim: improving the scalability of his startup, developing a real-time web log analyzer

- Facing problems in scaling some types of workloads with traditional databases
- Development starting with Tcl
- Re-writing in C
- First data type: the list
- Internal success - becoming open source in 2009 under a BSD 3-clause license
- Gaining attraction - GitHub and Instagram being among the first companies adopting it
- Currently sponsored by Redis Labs since 2015
- Introducing Redis Stream in 2018, when Redis 5.0 was released

System Description & Features

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker.

To achieve top performance, Redis works with an in-memory dataset. Depending on your use case, you can persist your data either by periodically dumping the dataset to disk or by appending each command to a disk-based log. You can also disable persistence if you just need a feature-rich, networked, in-memory cache.

Main Features:

- Built-in (asynchronous) replication
- Publish/Subscribe pattern
- LRU eviction of keys
- Transactions
- Keys with a limited time-to-live
- Different levels of on-disk persistence
- High availability via automatic failover
- Automatic partitioning with Redis Cluster

Supported data structures:

- Strings
- Hashes
- Lists
- Sets
- Sorted sets with range queries
- Bitmaps
- HyperLogLogs
- Geospatial indexes
- Streams

Supported atomic operations include:

- Appending to a string
- Incrementing the value in a hash
- Pushing an element to a list
- Computing set intersection, union and difference
- Getting the member with highest ranking in a sorted set.

Implementation architecture

- Redis popularized the idea of a system that can be considered at the same time **a store and a cache**, using a design where data is always modified and read from the main computer memory, but also stored on disk in a format that is unsuitable for random access of data, but only to reconstruct the data back in memory once the system restarts.
- Redis provides a data model that is very unusual compared to a relational database management system (RDBMS). **User commands do not describe a query to be executed by the database engine** but rather specific operations that are performed on given abstract data types.
- The Redis implementation makes heavy **use of the fork system call**, to duplicate the process holding the data, so that the parent process continues to serve clients, while the child process creates a copy of the data on disk
- They can therefore support an order of magnitude more operations and faster response times. The result is blazing **fast performance** with average read or write operations taking **less than a millisecond** and support for millions of operations per second.

Why redis

Popularity

- According to monthly DB-Engines rankings, Redis is often the **most popular key-value database**. Redis has also been ranked the **#4 NoSQL database** in user satisfaction and market presence based on user reviews, the most popular NoSQL database in containers, and the #4 Data store of 2019 by ranking website stackshare.io. It was voted **most loved database** in the **Stack Overflow** Developer Survey in 2017, 2018, 2019, and 2020.

- Large companies such as **Twitter** are using Redis, **Amazon** Web Services is offering Redis in its portfolio, **Microsoft** is offering the Redis Cache in Azure, and **Alibaba** is offering ApsaraDB for Redis in Alibaba Cloud.

- Many programming languages have Redis language bindings on the client side, including but not limited to: ActionScript, Bash, **C**, **C++**, **C#**, Chicken, Clojure, Common Lisp, Crystal, D, Dart, Elixir, Erlang, **Go**, Haskell, Haxe, Io, **Java**, **JavaScript (Node.js)**, Julia, Lua, **Matlab**, Objective-C, OCaml, Pascal, Perl, **PHP**, Prolog, Pure Data, **Python**, R, Racket, Ruby, Rust, Scala, Smalltalk, **Swift**, and Tcl.

  Note: *Several client software programs exist in these languages*

Popular Use Cases

Caching

*Redis can serve frequently requested items at **sub-millisecond response times**, and enables you to easily scale for higher loads without growing the costlier backend.*

- Database query results caching
- Persistent session caching
- Web page caching
- Images, files, metadata

Chat, messaging, and queues

*Redis supports **Pub/Sub** with pattern matching and a variety of data structures such as lists, sorted sets, and hashes.*

- Chat rooms
- Real-time comment streams
- Social media feeds
- Server intercommunication

Session store

*Redis as an in-memory data store with **high availability** and **persistence**, provides the sub-millisecond latency, **scale**, and **resiliency** required to manage several session data.*

- User profiles
- Credentials
- Session state
- User-specific personalization

Geospatial

Redis offers purpose-built in-memory data structures and operators to manage real-time geospatial data at scale and speed. You can use Redis to add location-based features.

- Drive time
- Drive distance
- Points of interest to your applications.

Popular Use Cases

Machine Learning

*Redis gives you a fast in-memory data store to **build**, **train**, and **deploy machine learning models quickly**. The ability to process live data and make decisions within tens of milliseconds is of utmost importance.*

- Fraud detection in gaming and financial services
- Real-time bidding in ad-tech
- Matchmaking in dating and ride sharing

Real-time analysis

*Redis can be used with streaming solutions such as **Apache Kafka** and **Amazon Kinesis** as an in-memory data store to ingest, process, and analyze real-time data with sub-millisecond latency.*

- Social media analytics
- Ad targeting
- Personalization
- IoT

Rich media streaming

*Redis can be used to **store metadata** about users' profiles and viewing histories, authentication information/tokens for millions of users, and manifest files to enable CDNs to stream videos to millions of mobile and desktop users at a time*

Real-time Gaming leaderboards

*You can simply use the Redis **Sorted Set** data structure, which provides **uniqueness** of elements while maintaining the list **sorted** by users' scores. Creating a real-time ranked list is as easy as updating a user's score each time it changes.*

Basic CLI Example

- To install on Debian Linux: **> $ sudo apt-get install redis-server**
- To enter the redis CLI: **> $ redis-cli**
- A Redis database looks like (and functions) as hash maps (or dictionaries).
- It holds *key:value* pairs and supports commands such as **GET**, **SET**, and **DEL**, as well as several hundred additional commands.
- **Redis keys are always strings.**
- Many Redis commands operate in constant *O(1)* time, just like retrieving a value from a ***Python dict***, a ***JavaScript Object*** or any ***generic hash table***.

Redis

```
127.0.0.1:6379> SET Bahamas Nassau
OK
127.0.0.1:6379> SET Croatia Zagreb
OK
127.0.0.1:6379> GET Croatia
"Zagreb"
127.0.0.1:6379> GET Japan
(nil)
```

*This database is a mapping of country:capital city, where we use **SET** to set key-value pairs*

Redis

```
127.0.0.1:6379> MSET Lebanon Beirut Norway Oslo France Paris
OK
127.0.0.1:6379> MGET Lebanon Norway Bahamas
1) "Beirut"
2) "Oslo"
3) "Nassau"
```

*Redis also allows you to set and get multiple key-value pairs in one command, **MSET** and **MGET**, respectively.*

Redis

```
127.0.0.1:6379> EXISTS Norway
(integer) 1
127.0.0.1:6379> EXISTS Sweden
(integer) 0
```

*The **EXISTS** command does what it sounds like, which is to check if a key exists.*

Redis

```
127.0.0.1:6379> FLUSHDB
OK
127.0.0.1:6379> QUIT
```

*You can clear your database with **FLUSHDB** and exit the redis-CLI with **QUIT***

Hashes, Sets, Lists

Two additional value types are lists and sets, which can take the place of a hash or string as a Redis value. They are largely what they sound like. Hashes, lists, and sets each have some commands that are particular to that given data type, which are in some cases denoted by their initial letter:

- **Hashes:** Commands to operate on hashes begin with an **H**, such as **HSET**, **HGET**, or **HMSET**.
- **Sets:** Commands to operate on sets begin with an **S**, such as **SCARD**, which gets the number of elements at the set value corresponding to a given key.
- **Lists:** Commands to operate on lists begin with an **L** or **R**. Examples include **LPOP** and **RPUSH**. The L or R refers to which side of the list is operated on. A few list commands are also prefaced with a **B**, which means *blocking*. A blocking operation doesn't let other operations interrupt it while it's executing. For instance, **BLPOP** executes a blocking left-pop on a list structure.

Note: *One noteworthy feature of Redis' list type is that it is a linked list rather than an array. This means that appending is O(1) while indexing at an arbitrary index number is O(N).*

Python Integration

We will focus on the Python's integration but similar methods exists in any other popular languages like **Java**, **JavaScript *(Node.js),* PHP**, **C++**, **C#**, **Go**, etc.

Redis-py is a well-established Python client library that lets you talk to a Redis server directly through Python calls

Shell

```
$ python -m pip install redis
```

Note: *Make sure that your Redis server is up and running in the background*

Python

```
1 >>> import redis
2 >>> r = redis.Redis()
3 >>> r.mset({"Croatia": "Zagreb", "Bahamas": "Nassau"})
4 True
5 >>> r.get("Bahamas")
6 b'Nassau'
```

- Redis, used in Line 2, is the central class of the package and the workhorse by which you execute (almost) any Redis command. The TCP socket connection and reuse is done for you behind the scenes, and you call Redis commands using methods on the **class instance** r.
- The type of the returned object, b'Nassau' in Line 6, is Python's **bytes type, not str.** It is bytes rather than str that is the most common return type across redis-py, so you may need to call ***r.get("Bahamas").decode("utf-8")*** depending on what you want to actually do with the returned bytestring.
- While the Redis command arguments usually translate into a similar-looking method signature, they take Python objects. For example, the call to ***r.mset()*** in the example above uses a Python dict as its first argument, rather than a sequence of bytestrings.

r.mset()	**MSET**
r.get()	**GET**
r.hgetall()	**HGETALL**
r.ping()	**PING**

The methods in almost all cases match the name of the Redis command that does the same thing

Note: *There are a few exceptions, but the rule holds for the large majority of commands.*

Pipeline example with python

Python

```
1  >>> with r.pipeline() as pipe:
2  ...     for h_id, hat in hats.items():
3  ...         pipe.hmset(h_id, hat)
4  ...     pipe.execute()
5  Pipeline<ConnectionPool<Connection<host=localhost,port=6379,db=1>>>
6  Pipeline<ConnectionPool<Connection<host=localhost,port=6379,db=1>>>
7  Pipeline<ConnectionPool<Connection<host=localhost,port=6379,db=1>>>
8  [True, True, True]
9
10 >>> r.bgsave()
11 True
```

Redis **pipelining** is a way to cut down the number of round-trip transactions that you need to write or read data from your Redis server.

If you would have just called ***r.hmset()*** three times, then this would necessitate a back-and-forth round trip operation for each row written.

With a pipeline, all the commands are buffered on the client side and then sent at once, in one fell swoop, using ***pipe.hmset()*** in Line 3. This is why the three True responses are all returned at once, when you call ***pipe.execute()*** in Line 4.

Key Expiry example with python

Python

```
>>> from datetime import timedelta

>>> # setex: "SET" with expiration
>>> r.setex(
...     "runner",
...     timedelta(minutes=1),
...     value="now you see me, now you don't"
... )
True
```

When you **expire** a key, that key and its corresponding value will be automatically deleted from the database after a certain number of seconds or at a certain timestamp.

In redis-py, one way that you can accomplish this is through ***.setex()***, which lets you set a basic *string:string* key-value pair with an expiration.

Python

```
>>> r.ttl("runner")  # "Time To Live", in seconds
58
>>> r.pttl("runner")  # Like ttl, but milliseconds
54368
```

There are also methods (and corresponding Redis commands, of course) to get the remaining lifetime (**time-to-live**) of a key that you've set to expire.

Python

```
>>> r.get("runner")  # Not expired yet
b"now you see me, now you don't"

>>> r.expire("runner", timedelta(seconds=3))  # Set new expire window
True
>>> # Pause for a few seconds
>>> r.get("runner")
>>> r.exists("runner")  # Key & value are both gone (expired)
0
```

Furthermore, you can accelerate the window until expiration, and then watch the key expire, after which ***r.get()*** will return *None* and ***.exists()*** will return *0*.