Advanced Data Management

Redis Intro

Peyer Lars

Berger Adrian

Version 1.0

10. November 2019

1 Key Value Store

1.1 Definition

Ein Key Value Store basiert auf einer Tabelle mit genau zwei Spalten:
In der einen befindet sich ein eindeutiges Identifikationsmerkmal, der Schlüssel (Key), in der anderen der Wert (Value). Somit wird jeder Wert eindeutig einem Schlüssel zugewiesen. Welche Datentypen im Wert gespeichert werden können, ist abhängig vom verwendeten Key-Value Store. Es handelt sich somit um eine NoSQL Datenbank.

1.2 Vorteile und Nachteile

1.2.1 Vorteile

- Geschwindigkeit
- Skalierbarkeit
- Simples Model, einfach verständlich

1.2.2 Nachteile

- Für Daten mit vielen relationalen Abhängikeiten sehr komplex
- Keine Lookup Optimierungen für das Suchen

1.3 Einsatzgebiete

Überall wo schnelle Zugriffszeiten bei großen Datenmengen benötigt werden, eignen sich Key Value Stores. Typische Einsatzgebiete sind deshalb Warenkörbe in Onlineshops oder das Speichern von Session-Daten.

1.4 In-Memory

Bei vielen Key Value Stores handelt es sich um In-Memory Datenbanken. Im Gegensatz zu herkömmlichen On-Disk Datenbanken verwenden diese anstelle des ROM, den Arbeitspeicher eines Computers. Da die Schreib- und Lesezugriffe auf HDDs (durch die magnateische Scheiben) und SSDs (als Flash-Speicher) beschränkt sind, können in herkömmlichen On-Disk Datenbanken längere Ladezeiten auftretten. Durch die Verwendung von In-Memory, kann dieses Limit stark erhöht werden. Nachteil davon ist, dass die Daten nicht direkt persistent gespeichert sind. Dadurch sind In-Memory Datenbanken gut geeignet, um grosse Datenmenegen auszuwerten.

1.5 Top 5 Key Value Stores

Gemäss db-engines.com (Stand 27.10.2019) sind die folgenden Datenbanksysteme führend:

1. Redis
2. Amazon DynamoDB
3. Microsoft Azure Cosmos DB
4. Memcached
5. Hazelcast

2 Redis

2.1 Charakterisierung

Redis ist ein Open Source, In-Memory Datenbanksystem, das als vorallem als Datenbank, Cache und Message Broker verwendet werden kann.
Redis ist BSD lizenziert und stellt somit keine Anforderungen an die Weiterverteilung der Software.
Der Name Redis entstand als Abkürzung für remote dictionary server.
Entwickelt wurde Redis 2009 von Salvatore Sanfilippo in C und gehört mitlerweile zu VMWware. Die aktuelle Version (Stand 5. November 2019) ist 5.0.5, wobei die aktuelle LTS Version 3.2.11 ist. Redis ist single-threaded und lässt sich einfach replizieren.

2.1.1 Platformen, welche Redis verwenden

Redis erfreut sich grosser Beliebtheit und ist laut DB-Engines.com die verbreitetste Key-Value Datenbank. Einige der grossen Kunden sind etwa Twitter (Real-Time Delivery), Github (Speicherung von Routing Information) und StackOverflow (2-Level Cache).
Mehr Infos unter: https://redis.io/topics/whos-using-redis

2.2 Installation

Redis lässt sich leicht als tar.gz gerunterladen und installieren:

```
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make
make install
```

Mehr Infos unter: https://redis.io/topics/quickstart

2.3 Datentypen

Redis unterstützt die folgenden Datentypen:

Typ	Beschreibung
String	Eine beliebige Zeichenfolge
List	Entspricht einem Array, die Elemente werden über einen positiven, ganzzahligen, null-basierten Index angesprochen
Hash	Entspricht einem assoziativem Array, die Elemente bestehen hier wiederum aus Schlüssel-Werten Paaren.
Set	Wie List, jedoch kann jedes Element nur einmal vorkommen darf.
Sorted Set	Entspricht einem Set, nur dass zusätzlich der Index des Elements explizit mit angegeben wird.
Bitmaps	Speicherung von Bits
HyperLogLogs	Probabilistische Datenstruktur, mit der sich die Kardinalität einer Datenmenge sehr speichereffizient bestimmen lässt

Mehr Infos unter: https://redis.io/topics/data-types

2.4 Abfragesprache

Für String lässt sich ein Speichern und Abrufen mit get und set erreichen (alle Beispiele mit Python Redis):

```
import redis
r = redis.Redis()
r.set("msg:hello", "Hello Redis!!!")
msg = r.get("msg:hello")
# prints b'Hello Redis!!!'
```

Für jeden Datentyp gibt es solche Schreib- und Lesebefehle. Zudem gibt es diverse weitere Befehle (z.B. zum Zählen der Elemente). Nachfolgend ein Beispiel für ein Set:

```
mountainSet = "Berge"
# Add elements to the Redis set
r.sadd(mountainSet, "Eiger")
r.sadd(mountainSet, "Moench")
r.sadd(mountainSet, "Jungfrau")

print("Anzahl Elmente: ")
print(r.scard(mountainSet))
print(r.smembers(mountainSet))
# prints Anzahl Elmente: 3 {b'Eiger', b'Moench', b'Jungfrau'}
```

Eine Liste aller Abfragen findet sich hier: https://www.cheatography.com/tasjaevan/cheatsheets/redis/

2.4.1 Transaktionen

Eine Transaktion in Redis besteht aus einem Block von Befehlen, die zwischen MULTI und EXEC (oder DISCARD für Rollback) platziert werden. Sobald ein MULTI gefunden wurde, werden die Befehle auf dieser Verbindung nicht ausgeführt - sie stehen in der Warteschlange (und der Anrufer erhält die Antwort QUEUED). Wenn eine EXEC angetroffen wird, werden sie alle in einer einzigen Transaktion angewendet (d.h. ohne dass andere Verbindungen Zeit zwischen den Operationen erhalten). Wenn anstelle von EXEC eine DISCARD gesehen wird, wird alles weggeworfen. Da die Befehle innerhalb der Transaktion in der Warteschlange stehen, können Sie innerhalb der Transaktion keine Entscheidungen treffen.

Sämtliche Informationen dazu finden sich hier: https://redis.io/topics/transactions

2.5 Massenload

Manchmal müssen Redis-Instanzen in kurzer Zeit mit einer großen Menge an bereits vorhandenen oder benutzergenerierten Daten geladen werden, so dass Millionen von Schlüsseln so schnell wie möglich erstellt werden.

Ab Redis 2.6 unterstützt das Dienstprogramm redis-cli einen neuen Modus namens Pipemode, der entwickelt wurde, um eine Masseneinfügung durchzuführen.

Die genaue Spezifikation dazu findet sich hier: https://redis.io/topics/mass-insert

2.6 Benchmarks

Redis bietet das Dienstprogramm redis-benchmark, das laufend Befehle von Clients simuliert und gleichzeitig Gesamtabfragen sendet. Damit kann die Performance von Redis getestet und visualisiert werden.
Anleitungen und Beispiele dazu finden sich hier: https://redis.io/topics/benchmarks

2.7 API

Redis-Module sind dynamische Bibliotheken, die beim Start oder mit dem Befehl MODULE LOAD in Redis geladen werden können.

Redis-Module ermöglichen es, die Redis-Funktionalität durch externe Module zu erweitern, indem sie neue Redis-Befehle mit einer Geschwindigkeit und mit ähnlichen Funktionen implementieren, wie sie im Kern selbst möglich sind.

Weiter Informationen dazu finden sich hier: https://redis.io/topics/modules-intro

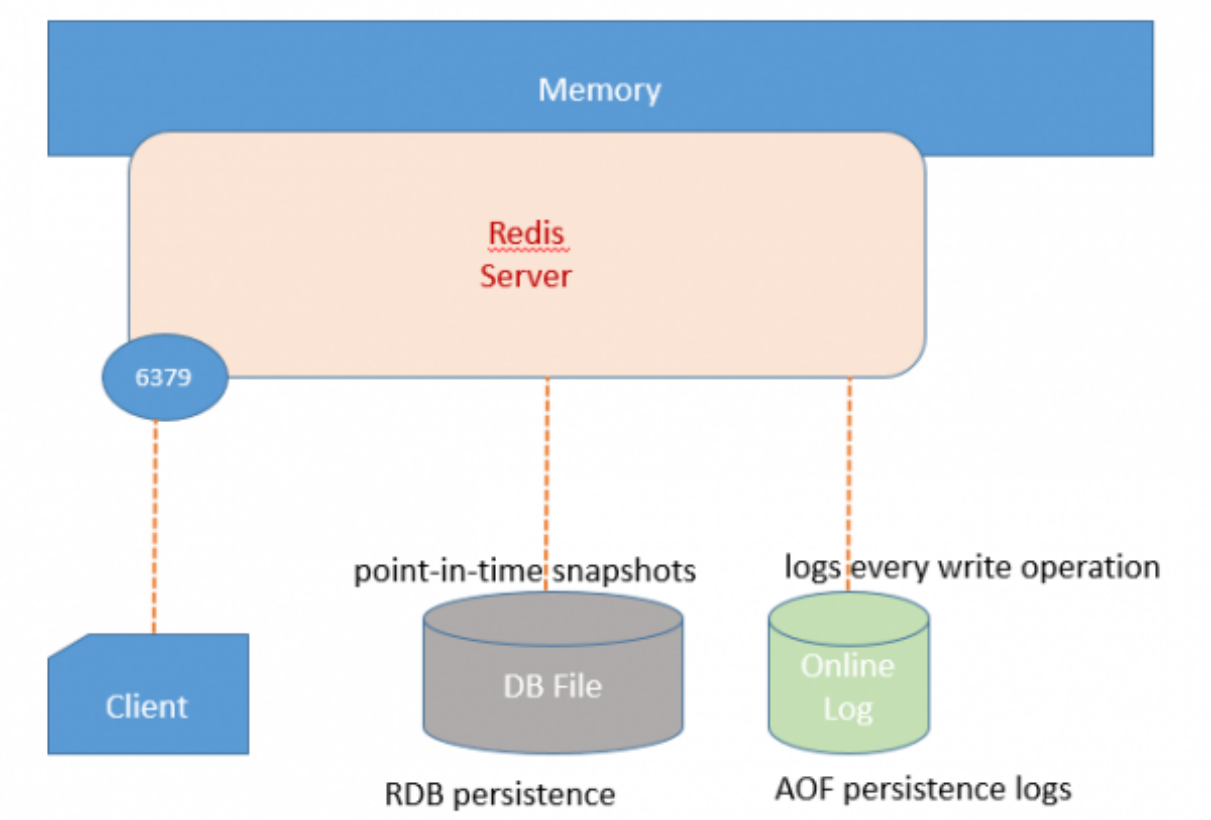
2.8 Persistenz

Redis bietet generell zwei Varianten der sicheren Datenspeicherung:

RDB (Snapshots): Zu bestimmten Zeitpunkten (z.B. jede volle Stunde) wird eine vollständige Kopie der im Speicher befindlichen Daten erstellt. Wenn z.B. zwischen zwei Snapshots die Stromversorgung verlieren geht, gehen die Daten aus der Zeit zwischen dem letzten Snapshot und dem Crash verloren.

AOF: Logt jede vom Server empfangene Schreiboperation und führt diese beim Serverstart erneut ab zur Rekonstruktion des ursprünglichen Datensatzes.

Grundsätzlich lassen sich RDB und AOF einzel aktiviren. Somit wäre auch keine Persistenz möglich, in dem beides deaktiviert wird. Die beste Persistenz wird erreicht, in dem beides aktiviert wird. Grundsätzlich handelt es sich hierbei um eine Frage zwischen Datensicherheit und Leistung.



Weitere Vor- und Nachteile finden sich hier: https://redis.io/topics/persistence

2.9 Replikation

Auf der Basis der Redis-Replikation gibt es eine sehr einfach zu bedienende und zu konfigurierende Master-Slave-Replikation. Sie ermöglicht es, dass Redis-Instanzen als Replikate exakt von Master-Instanzen kopiert werden.

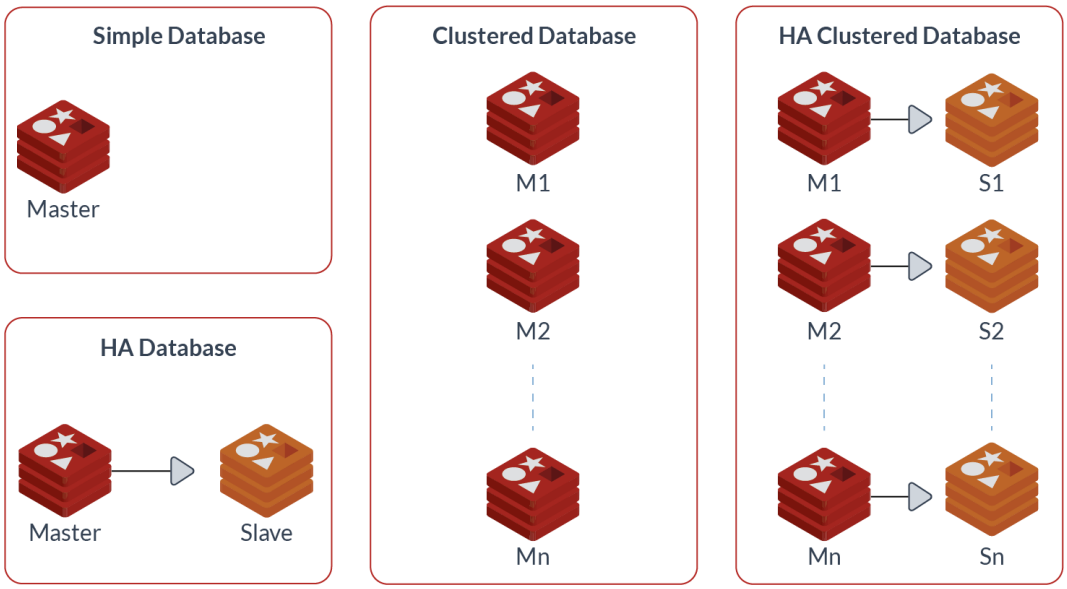
Dieses System arbeitet mit drei Hauptmechanismen:

1. Wenn Master und Slave verbunden sind, hält der Master den Slave auf dem neuesten Stand, indem er einen Befehlsstrom sendet, um die Auswirkungen auf den Slave Datensatz zu replizieren.
2. Wenn die Verbindung zwischen Master und Slave abbricht, verbindet sich der Slave erneut und versucht, mit einer teilweisen Resynchronisierung fortzufahren: Er erhäötn nur den Teil des Befehlsstroms, den er während der Trennung verpasst hat.
3. Wenn eine teilweise Resynchronisation nicht möglich ist, fordert der Slave eine vollständige Resynchronisation an. Dies erfordert einen komplexeren Prozess, bei dem der Master einen Snapshot aller seiner Daten erstellen, an den Slave sendet.

Redis verwendet standardmässig asynchrone Replikation, die eine geringe Latenz und hohe Leistung aufweist. Redis-Slaves bestätigen jedoch asynchron die Menge der Daten, die sie periodisch vom Master empfangen haben. Der Master wartet somit nicht jedes Mal darauf, dass ein Befehl von den Replikaten verarbeitet wird, weiß aber bei Bedarf, welcher Slave bereits welchen Befehl verarbeitet hat.

Redis lässt sich zudem mittels Redis Cluster auf mehrere Knoten verteilen.

Redis Cluster bietet die Möglichkeit den Betrieb fortzusetzen, wenn einige Knoten ausfallen oder nicht kommunizieren können. Bei größeren Ausfällen (z.B. wenn die Mehrheit der Master nicht verfügbar ist) stoppt der Cluster jedoch den Betrieb.



Die Dokumentation dazu fndet sich hier: https://redis.io/topics/replication

2.10 Optimierungsmöglichkeiten

2.10.1 Performance

Mit Piplines kann die Performance verbessert werden, da nicht bei jeder Anfrage auf das Resultat vom Server gewartet wird, bis die nächste Anfrage gestartet wird. Somit laufen mehrere Abfragen gleichzeitig. Mehr dazu findet sich hier: https://redis.io/topics/pipelining

Zudem kann die Performance optimiert werden, in dem RBD und AOF ausgeschalten werden. Dadurch geht jedoch auch das persistieren der Daten verloren.

2.10.2 Memory

Daten sollten immer, wenn es möglich ist, mit Hashes dargestellt werden, damit Memory gespart werden kann. So kann anstelle verschiedener Schlüssel ein einzigr Hash mit allen erforderlichen Feldern verwendet werden.

Zudem kann Redis auf 32 Bit umgestellt werden. Redis kompiliert mit 32-Bit verbraucht viel weniger Speicher pro Schlüssel, da die Zeiger klein sind, ist aber auf 4 GB maximale Speichernutzung beschränkt. RDB- und AOF-Dateien sind zwischen 32-Bit- und 64-Bit-Instanzen kompatibel.

Weitere Optimierungen finden sich hier: https://redis.io/topics/memory-optimization

2.11 GUI

Die bekanntesten und verbreitetsten GUIs für Redis sind: Redsmin, Redis Commander, Redis Desktop Manager und redis-browser.
Eine gute Übersicht findet sich hier: https://redislabs.com/blog/so-youre-looking-for-the-redis-gui/

3 Ressourcen

https://en.wikipedia.org/wiki/Key-value_database
https://db-engines.com/de/ranking/key-value+store
https://www.aerospike.com/what-is-a-key-value-store/
https://www.ionos.de/digitalguide/hosting/hosting-technik/key-value-store/
https://redis.io/
https://redislabs.com
https://www.pipperr.de/dokuwiki/doku.php?id=nosql:redis_overview
https://wikipedia.org/wiki/Redis
https://stackoverflow.com/questions/51314487/redis-memory-optimization-suggestions