BLOG

Introduction to Redis (What it is, what are the use cases etc)

Paul Namuag

Published: July 28, 2021
Last Updated: June 28, 2023

Redis Overview

Redis (which means Remote Dictionary Serve) is an open-source NoSQL database known to be a fast, in-memory key-value data store, cache, message broker, and queue. The project started by Salvatore 'antirez' Sanfilippo, the original developer of Redis. He was trying to improve the scalability of his Italian startup developing a real-time web log analyzer. After encountering significant problems in scaling some types of workloads using traditional database systems, he began to prototype a first

Redis has been written in the ANSI C language and it works in most of the POSIX systems such as BSD, Linux, OS X without having any external dependencies. OS X and Linux are considered being the two operating systems where Redis has been developed and tested the most whereas Linux has been used for deploying the same. Redis might operate in the Solaris-derived systems such as SmartOS, but the support is the best effort. Unfortunately, there is no official support that is provided for the Windows builds, but Microsoft has developed and maintained a Win-64 port for Redis. In 2019, Redis celebrated its 10th anniversary.

Redis now delivers sub-millisecond response times enabling millions of requests per second for real-time applications in Gaming, IoT, Social Networking, Financial Services, Healthcare Industry, and Ad Tech. Redis is a popular choice for caching, session management, gaming, leaderboards, real-time analytics, geospatial, ride-hailing, chat/messaging, media streaming, and pub/sub apps.

It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. It has LRU eviction, Lua scripting, built-in replication, transactions as well as different stages of on-disk persistence. It has built-in support for high-availability solutions through Redis Sentinel or by leveraging Redis Cluster with automatic partitioning.

Common use case by experience users of Redis can be applied in such tasks such as running a number of operations like incrementing a hash value, appending to a string, computing set intersection, union and difference, pushing an element to a list, or collecting the member with the greatest ranking in a sorted set. Redis manages these types of tasks to an optimal level which can result in outstanding performance. This is because Redis operates with a built in-memory dataset.

Based on the use case, one can persist the same either by dumping a dataset to the disk once in a while or by attaching each of the commands to a log. Persistence can be easily disabled if one needs a networked, feature-rich, in-memory cache.

Just like with other mainstream databases, Redis also supports the master-slave asynchronous replication, alongside a very fast non-blocking synchronization, auto-reconnection with a partial resynchronization when the link between the master and the of commands it missed during the disconnection.

Redis Features

Features of Redis highlight the significance of its capabilities that it can provide and power up your existing environment with its distinctive features. Every technology has many individual features and attributes. Redis also has some very important and useful features to be used. We'll take these as we go along in this blog.

Rich Data Structures

Redis offers five possible data options for the values. These are hashes, lists, sets, strings, and sorted sets. The operations which are unique to these data types are given and come along with the well-documented time-complexity (The Big O notation).

Redis is more than just a usual key-value datastore technology. It offers a vast variety of data structures to meet our application needs. It enables users to implement applications according to the client's requirements not associated with the technology limitations. Available data structures are listed below.

Strings: Text data up to 512MB in size.

Lists: A collection of Strings. Adding order will be preserved.

Sets: An unordered collection of strings with the ability to do set operations

Sorted Sets: Sets ordered by a value associate with a key. Can be used for leader board and scoreboard implementations.

Hashes: Adata structure for storing a list of fields and values similar to Hashmaps.

Bitmaps: A data type that offers bit-level operations to be done.

HyperLogLogs: A probabilistic data structure to estimate the unique items in a data set. Can be used to do probabilistic calculations.

Data Persistence means that the data survives after the producer process of the particular data has ended. In other terms, Saved data must last even if the server fails. For a data store to be considered persistent, It must write on permanent storage (i.e. non-volatile storage such as hdd or ssd).

Redis typically holds the whole dataset in memory. For persistence, Redis supports point-in-time backups (copying the Redis data set to disk). Redis supports RDB, AOF persistence mechanism to persist the data to the non-volatile memory.

Persistence in Redis can be achieved through two different methods. First by snapshotting, where the dataset is asynchronously transferred from memory to disk at regular intervals as a binary dump, using the Redis RDB Dump File Format. Alternatively by journaling, where a record of each operation that modifies the dataset is added to an append-only file (AOF) in a background process. Redis can rewrite the append-only file in the background to avoid an indefinite growth of the journal. Journaling was introduced in version 1.1 and is generally considered the safer approach.

By default, Redis writes data to a file system at least every 2 seconds, with more or less robust options available if needed. In the case of a complete system failure on default settings, only a few seconds of data would be lost.

Performance

Performance in Redis is extremely efficient. Because of its in-memory nature, a project manager's commitment to ensuring that complexity stays at the bare minimum, as well as an event-based programming model, the application boasts of having an exceptional performance for the read and write operations.

When the durability of data is not needed, the in-memory nature of Redis allows it to perform well compared to database systems that write every change to disk before considering a transaction committed. Redis operates as a single process and is single-threaded or double-threaded when it rewrites the AOF (append-only file). Thus, a single Redis instance cannot use parallel execution of tasks such as stored procedures.

Redis has Built-in support for non-blocking, asynchronous, primary/replica replication, in order to ensure high-level data availability. We have discussed this already in our previous blogs. You can start reading these blog topics Redis High Availability Architecture with Sentinel or Hash Slot vs. Consistent Hashing in Redis which also covers a brief overview of Redis Cluster.

Basically, Redis offers a primary-replica architecture in a single node primary or a clustered topology. This allows you to build highly available solutions providing consistent performance and reliability. Redis is, mostly, a single-threaded server but modern versions of Redis use threads for different things. It is not designed to benefit from multiple CPU cores. People are supposed to launch several Redis instances to scale out on several cores if needed. Redis server uses master-slave (primary/replica) architecture to increase the load that can be handled by the server. Due to Redis primary-replica architecture, read performance can be improved when splitting read requests among the servers and faster recovery when the primary server experiences an outage. You may use this to get more ideas about the Redis availability.

If you are using Redis Cluster, scaling vertically is super efficient and very easy to manage. There are a bunch of commands for you to utilize such as resharding and rebalancing hash slots. If you're going to use traditional primary-replica setup with Redis, using Sentinel has to be integrated and using the right Redis clients but with consistent hashing is your choice if you want to shard and scale your primary nodes for distributing write requests.

Simply Efficient and Lightweight

Redis is written in the ANSI C language, and it has no external dependencies. The program works perfectly well in all the POSIX environments. The Windows platform is not officially supported for Redis, but an experimental build has been provided by Microsoft for the same.

Redis also supports languages including C, C++, C#, Java, JavaScript, Node.js, Python, PHP, Ruby, R and many others. The Redis comes with native data structures as mentioned above. Redis has very large and dynamic community support. It simple.

Redis Use Cases

When speaking of use cases of using Redis, the most generic use case that comes to many is for caching data and for session store (i.e. web session). Redis has a lot of use cases that can be applied and useful for any situations especially that involves speed and scalability wise, it's easy to manage either scaling up or scaling down.

For most experienced users or companies that have been adopting Redis for a long time, typical use cases are session caching, full page cache, message queue applications, leaderboards and counting among others. Redis is also prevalent among large cloud companies that offer fully-managed databases or DBaaS. Large companies such as Twitter are using Redis, Amazon Web Services offers a managed Redis service called Elasticache for Redis, Microsoft offers Azure Cache for Redis in Azure, and Alibaba is offering ApsaraDB for Redis in Alibaba Cloud.

Redis use cases implies which type of scenario you can apply, implement, and integrate Redis to your environment and business applications. As the Redis delivers super-fast performance. It is commonly used for real-time applications. We'll take a look at each of these where it is ideally applicable.

Caching

The cache is temporary storage where data is stored so that in the future data can be accessed faster. So, caching is the process of storing data in Cache. Redis is a good choice for implementing a highly available in-memory cache to decrease data access latency with disk or SSD, high throughput and ease the load of the database and application. Web page caching, Database query results caching, persistent session caching and caching of frequently used objects such as images, files, and metadata are popular examples of caching solutions with Redis.

Session store

session for each connected user with the system, for as long as the user is logged in with the system. Session state is how applications remember user identity, login credentials, personalization information, recent actions, shopping cart details, and more.

Reading and writing session data at each user interaction must be done without disturbing any of the user experience. Therefore, while the user session is live, no round-trip to the main database should be needed. The last step in the session state life cycle occurs when the user is disconnected. Some data will be persisted in the database for future use, but transient information can be discarded after the session ends.

Chat and Messaging Applications

Redis supports Pub/Sub with pattern matching and many different varieties of data structures such as lists, sorted sets, and hashes. This allows Redis to support high-performance chat rooms, real-time comment streams, social media feeds and server intercommunication.

Chatting application's use cases and respective data structures usage

Save active conversations in a SET: Set only contains unique items. So active users are put into a set.

Save the last 10 messages of a conversation using a Redis LIST: A Redis list is somewhat of a persistent array of strings. We push a new message into a list and pop the oldest when the list size reaches your threshold.

Save some basic user details in a HASH: Hash can be used to store the user details, online status of the user.

Maintain a "recent conversations" list in a SORTED LIST: Simply using the timestamp as score whenever there is an interaction between two users sorted list will change the order according to the score.

Gaming leaderboard applications

leaderboards or scoreboards. Redis Sorted Set data structure can be simply used to implement this use case, which provides uniqueness of elements while keeping the list sorted by users' scores(points) associated with the key. Need to update the user's score whenever it changes. We can also use Sorted Sets to handle time-series data by using timestamps as the score for ranking based on timestamps.

Redis is one of the famous technologies for Session store implementation. As it has a very high throughput, It is widely used in session store implementations.

Other than the above-mentioned use cases, There are many more Redis based use cases such as Machine Learning, Real-time Analysis, media streaming etc use cases use Redis in many different ways.

Conclusion

Once you know the areas to apply Redis for your business applications, you can leverage the power and speed that this technology can provide from scalability, high availability, and optimal performance that can benefit your needs. Redis might not be that old compared to other existing databases but its wide adoption and large support by the community proves how powerful and efficient this tool can be.

For more best practices for managing your Redis database, be sure to subscribe to our newsletter and follow us on LinkedIn and Twitter. See you soon!

CATEGORY

DATABASE - GENERAL REDIS