What is Redis?

Redis (for **RE**mote **DI**ctionary **S**erver) (link resides outside IBM) is an open source, in-memory, NoSQL key/value store that is used primarily as an application cache or quick-response database. Because it stores data in memory, rather than on a disk or solid-state drive (SSD), Redis delivers unparalleled speed, reliability, and performance.

When an application relies on external data sources, the latency and throughput of those sources can create a performance bottleneck, especially as traffic increases or the application scales. One way to improve performance in these cases is to store and manipulate data in-memory, physically closer to the application. Redis is built to this task: It stores all data in-memory—delivering the fastest possible performance when reading or writing data—and offers built-in replication capabilities that let you place data physically closer to the user for the lowest latency.

Other Redis characteristics worth noting include support for multiple data structures, built-in Lua scripting, multiple levels of on-disk persistence, and high availability.

Quick Overview

Redis (REmote DIctionary Server) is an in-memory, key-value database, commonly referred to as a data structure server. One of the key differences between Redis and other key-value databases is Redis's ability to store and manipulate high-level data types. These data types are fundamental data structures (lists, maps, sets, and sorted sets) that most developers are familiar with. Redis's exceptional performance, simplicity, and atomic manipulation of data structures lends itself to solving problems that are difficult or perform poorly when implemented with traditional relational databases.

Common Use Cases

- **Caching** – Due to its, developers have turned to Redis when the volume of read and write operations exceed the capabilities of traditional databases. With Redis's capability to easily persist the data to disk, it is a superior alternative to the traditional memcached solution for caching.

- **Publish and Subscribe** – Since version 2.0, Redis provides the capability to distribute data utilizing the Publish/Subscribe messaging paradigm. Some organizations have moved to Redis and away from other message queuing systems (i.e., zeromq, RabbitMQ) due to Redis's simplicity and performance.

- **Queues** – Projects such as Resque use Redis as the backend for queueing background jobs.

- **Counters** – Atomic commands such as HINCRBY, allow for a simple and thread-save implementation of counters. Creating a counter is as simple as determining a name for a key and issuing the HINCRBY command. There is no need to read the data before incrementing, and there are no database schemas to update. Since these are atomic operations, the counters will maintain consistency when accessed from multiple application servers.

Key Features

- **High-Level Data Structures**

– Provides five possible data types for values: strings, lists, sets, hashes, and sorted sets. Operations that are unique to those data types are provided and come with well documented time-complexity (Big O notation).

- **High Performance** – Due to its in-memory nature, the project maintainer's commitment to keeping complexity at a minimum, and an event-based programming model, Redis boasts exceptional performance for read and write operations.

- **Lightweight With No Dependencies** – Written in ANSI C, and has no external dependencies. Works well in all POSIX environments. Windows is not officially supported, but an experimental build is provided by Microsoft.

- **High Availability** – Built-in support for asynchronous, non-blocking, master/slave replication to ensure high availability of data. There is currently a high-availability solution called Redis Sentinel that is currently usable, but is still considered a work in progress.

Differentiating capabilities

Redis stands apart from 'traditional' NoSQL data stores as an auxiliary component designed specifically to improve application performance. Here are a few differentiating capabilities of Redis:

Redis cache sessions

Again, unlike NoSQL databases such as MongoDB and PostreSQL, Redis stores data in the server's main memory rather than on hard disks and solid-state drives. This leads to significantly faster response times when performing read and write operations. It also helps ensure high availability (together with Redis Sentinel—see below) and scalability of services and application workloads.

Redis queues

Redis can queue tasks that may take web clients longer to process than usual. Multiprocess task queuing is commonplace in many of today's web-based applications, and Redis makes it easy to implement automated Python-written processes that run in the background of request/response cycles.

Redis data types

While technically a key/value store, Redis is an actual data structure server that supports multiple data types and structures, including the following:

- Unique and unsorted string elements
- Binary-safe data
- HyperLogLogs
- Bit arrays
- Hashes
- Lists

Redis client handling

Redis features native client integration capabilities to help developers manipulate and interact with their data. There are currently well over 100 different open source clients available in the Redis client library, and developers can easily add new integrations to support additional features and programming languages.

Features

Some of the most important or noteworthy features of Redis include the following:

Redis Sentinel

Redis Sentinel (link resides outside IBM) is a stand-alone distributed system that helps developers calibrate their instances to be highly available for clients. Sentinel uses a series of monitoring processes, notifications, and automatic failovers to inform users when there is something wrong with master and slave instances, while automatically reconfiguring new connections for applications when necessary.

Redis Cluster

Redis Cluster (link resides outside IBM) is a distributed implementation of Redis that automatically splits datasets among multiple nodes. This supports higher performance and scalability of database deployments, while ensuring continuous operations in the event that node subsets are unable to communicate with the rest of the cluster.

Redis Pub/Sub

Because Redis supports the use of publish and subscribe (Pub/Sub) commands (link resides outside IBM), users can design high-performance chat and messaging services across all their applications and services. This includes the ability to use list data structures to run atomic operations and blocking capabilities.

Redis persistence

Redis uses persistent disk storage (link resides outside IBM) designed to survive process outages and network bottlenecks. Redis can persist datasets by taking regular snapshots of data and appending them with changes as they become available. Redis can then be configured to generate these database backups on demand or at automatic intervals to ensure database durability and integrity.

Redis vs. Memcached

Both Redis and Memcached are open source, in-memory data stores, but they differ when it comes to their benefits and features. Memcached is often the preferred choice for simple applications requiring fewer memory resources, but it is limited when storing data in its serialized form. Redis' use of data structures provides much more power when working with large datasets and more ability to fine-tune cache contents and maintain greater efficiency in specific application scenarios.

Redis vs. MongoDB

While Redis is an in-memory database store, MongoDB is known as an on-disk document store. Although both solutions are built for different purposes, they are often used together to maximize the speed and efficiency of

a NoSQL database. Because of its caching ability, Redis can locate required data extremely quickly, serving as an ingestion buffer that makes MongoDB more efficient and able to manage larger frequencies of document updates in near real-time. With MongoDB's ability to store significant amounts of data and Redis' ability to process it faster, the pairing offers a powerful database management solution for a variety of use cases.

Use cases

Here are some common use cases that enterprises benefit from when working with Redis:

- **Real-time analytics:** Because Redis can process data with sub-millisecond latency, it is ideal for real-time analytics, online advertising campaigns, and AI-driven machine learning processes.
- **Location-based applications:** Redis simplifies the development of location-based applications and services by providing geospatial indexing, sets, and operations. Using sorted sets, Redis is able to offload time-consuming searching and sorting of location data while also using an intelligent geo-hashing implementation.
- **Caching for databases:** Redis is able to handle large amounts of real-time data, making use of its in-memory data storage capabilities to help support highly responsive database constructs. Caching with Redis allows for fewer database accesses, which helps to reduce the amount of traffic and instances required. By using Redis for caching, development teams can dramatically improve their application throughputs by achieving sub-millisecond latency. And since Redis' caching layer can scale quickly and economically, organizations are able to develop these highly responsive applications while reducing their overall expenditures.

Installing Redis

Getting started with Redis is a fairly seamless process, especially with the use of the Redis Desktop Manager (RDM) (link resides outside IBM). And since Redis and RDM are open source, active development communities are always working to improve their efficiency of operation and continuously evolve supported tools and integrations.

For more information on installing and setting up Redis, follow the setup instructions in the community (link resides outside IBM).

https://www.credera.com/insights/redis-explained-5-minutes-less Ref