techaheadcorp.com

Enhance database performance with Redis | TechAhead

Ketan Varshneya October 4, 2019 | 3555 Views

9-11 minutes

Before we jump right into Redis, let's zoom out a bit and understand the basics of database caching. *A relational database system designed without a cache* works in the following manner:

- A *user* sends a request to the *server*
- The *server* queries the *database* to fetch the relevant data.
- The database fetches and sends the data to the server.

To optimize this system, you can write better queries which would improve the performance. However, this does not help when it comes to scaling the application for thousands of users.

What is a database cache and how caching works?

Let's see how a *database system with cache* works.

A database cache helps your server fetch data as quickly as possible without querying the database. If the data is not present in the cache, only then it will hit the database. Once the data is served to the user, the next step is to update the cache for future requests. A cache is useful in two scenarios:

- **While querying for commonly used data**, such as user profiles, geographical coordinates, restaurant menus, and so on. This is the type of data very frequently requested by the users. It will improve performance if the latest copy of this data is available on the server.
- **While frequently querying the same data from the database which involves complex computations**

Redis: In-memory datastore

Redis stands for Remote Dictionary Server and is developed in C programming language. This is the primary reason for Redis being so fast. As per their official documentation,

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

Redis: Language support

Redis supports a huge number of programming languages with a huge list of Redis clients (mentioned in brackets below) A few of the popular ones can be listed as:

- C (hiredis, hiredis-vip)
- C# (Redis, StackExchange.Redis)
- C++ (acl-redis)

- Java (Jedis, lettuce)
- Python (redis-py)
- js (ioredis, node_redis)
- PL/SQL (oredis)
- PHP (phpredis, Predis)
- Perl
- Swift (PSSRedisClient)
- Scala (scala-redis)
- Go (Radix)

The latest version of Redis can be downloaded from here. What makes Redis so popular as a cache is its support for a plethora of data types such as:

- Strings
- Hashes
- Lists
- Sets
- Bitmaps
- Hyperloglogs
- Geospatial indexes

Redis Architecture: An Overview

A single instance Redis architecture comprises of:

- Redis Client
- Redis Server

These two components can be on the same computer. The **Redis**

server takes care of storing and managing the data. The Redis client is the Redis console or a programming language's Redis API. As Redis stores everything on RAM, it's volatile and therefore you need to ensure that the data is PERSISTENT.

Redis Persistence

Redis can be made persistent in the following ways:

- RDB Mechanism
- AOF
- SAVE

How does Redis work?

Redis cache stores data in the form of key-value pairs in an in-memory format. This means that Redis stores data in the primary memory (RAM) which enable for very fast read and write speeds as compared to relational databases that store data on SSDs or HDDs. By storing data in-memory, Redis can do away with seek time delays. The key has to be a string, but the value can be any of the data types that it supports such as string, list and so on.
Example of Redis' key-value pairs:

```
name="john"

occupation=["developer", "designer"]
```

Here *name* and *occupation* are keys, with their corresponding string and list values.

Benefits of Redis

- **Reduce network calls to the database**

  The example of a ***Twitter user profile*** is a prime example, which is accessed frequently. This can be cached and every time a user queries the server, it saves a network call to the database since the

data is already cached.

- **Reduce re-computations and database load from multiple servers**

  You can store results of complex database computations on your cache for faster turnaround times and reducing database rework. However, the most critical factor for this would be deciding what computations are most frequently requested by the users and storing only those on the cache.

- **High Availability and Scalability**

  A cache-based system design reduces or distributes the number of queries that your database needs to handle. This way your server can handle more concurrent requests and reduce the load on your database.

- **Performance**

  Your server performance is analyzed based on how fast it can handle a request. With a cache sitting in between your server and database, the response times are significantly reduced for data that is already available in the cache.

- **Open-source Community**

  Redis is open-source with a huge community to support it. Hence there is no vendor lock-in or technological barrier faced. It has huge support for data formats and languages.

Use cases of Redis

Redis can be used for a multitude of things:

- **Caching**

  Redis is excellent for caching due to its high-speed read and write speeds which result from being stored on the primary memory (in-memory) itself. Redis has sub-millisecond response times and

hugely reduces latency and load on your traditional RDMS/NoSQL database.

- **Forums, live chat rooms**

  Live chat rooms and message forums rely on a robust, high-performing message broker, which is where Redis comes in. Redis' support for List data structures and hashes makes it ideal for this use case.

- **Live game leaderboards**

  Game developers opt for a live in-game leaderboard and Redis as their go-to choice with the Sorted Set data structure.

- **Media streaming**

  Online media streaming companies like Netflix need extremely fast cache for metadata, viewing histories, timelines for extremely smooth viewer experience. Redis makes it possible for millions of users to stream content seamlessly without any lag.

- **Machine learning**

  Apps belonging to FinTech, Gaming, Advertisement industries that implement ML techniques need to process large volumes of data very quickly. to deliver. Redis' in-memory data storage helps deliver meaningful real-time insights.

- **Real-time analytics**

  Redis' sub-milliseconds latency speed helps with real-time data analytics for social media, ad-tech and so on.

Cache Policy

So, do we put everything on the cache? No. That's a bad idea for two reasons:

- Hardware used for caching is expensive and not cheap commodity

hardware.

- If your cache has a lot of data, then the search times increase. Instead, why not hit the database. Cache becomes counterproductive.

  The cache should have the most relevant data basis the prediction of what might be needed soon. How you decide what to put in the cache and what to evict from it is called **Cache Policy**.

Database caching strategies for designing your Cache Policy

There are multiple caching strategies, predominantly basis different use cases. Some of these are:

- **Cache-aside**

  Here the server queries the cache first to fetch the data. In case of a ‘cache hit’, the data is retrieved. If it’s a ‘cache miss’ (data isn’t available in the cache), it queries the database. There’s an additional step to use the same data to update the cache for future use.

- **Read-through**

  Here the cache sits in-line between the server and the database. All requests go through the cache only. In case of a ‘cache-miss’, the cache updates itself first, then sends the data to the server.

- **Write-through**

  Similar to the read-through strategy, all writes go through the cache. This way the cache is always consistent with the database.

- **Write-back/Write-behind**

  This is similar to write-through, the only difference being that the cache doesn’t update the database for every write operation. Instead, it updates the database after a set period of delay to

reduce network calls.

How to use Redis as cache

Redis can be implemented in several ways.

Step 1: Install Redis.

For Ubuntu: apt-get install redis-server

For Mac: brew install redis

Step 2: Configure Node.js and Redis : npm i redis

```
const express = require('express');

const app = express () // create express app\

const redis = require('redis');

const client = redis.createClient(6379); //connect redis client with local instance.

// echo redis errors to the console

client.on('error', (err) => {

console.log ("Error " + err)

});

const companyName = "ABCD";

// Save variable value in Redis store, data expire time in 3600 seconds, it means one hour

client.setex(singleVar, 3600, companyName);

const jsonData = {

    "name": "Steve",

    "email": "steve@mail.com",

    "department": "MEAN"

}
```

```
// Save JSON in Redis store, data expire time in 3600 seconds, it means one hour
client.setex(jsonVar, 3600, jsonData);
```

To clear all the keys in Redis use the below command:

```
redis-cli FLUSHALL
```

Summarize

Distributed database caching hugely improves user experience and application performance. Redis has a competitor in Memcached, however certain features offered by Redis' outshine Memcached such as Replication, Snapshots and Geospatial support. Redis is trusted by some of the world's leading companies, namely, Twitter, GitHub, Pinterest, Snapchat, Stack Overflow and Flickr.