Intro to Redis

CS-554 – WEB PROGRAMMING

Terms

Redis: An in-memory data structure store

Cache: A collection of data stored in a quickly-performing lookup for future use

Fast: Like “racing stripes on a Lamborghini that is being chased by a Ferrari driven by fire-breathing demons” sort of fast.

Redis Concepts

What is Redis?

Redis is an in memory data structure store.

- http://redis.io/topics/introduction

Redis is often used as a high-performing lookup. Reads and write operations are **extremely** fast.

You can store a number of data types in Redis:

- Strings
- Sets
- Lists
- Sorted sets
- Hashes
- Bit arrays

How it works

Redis keeps the entire data-set in memory in a highly efficient structure

- The more data you have, the more memory you'll need
- Can vertically scale this with huge RAM machines
- Can horizontally scale by sharding the data across many machines

Because this data is in memory, Redis is extremely fast.

- As a tradeoff, it has **no** querying (ie, find all user with an age greater than 24)

All operations are atomic

What it’s good at

Amazing at write and retrieval

- Does not sync to disk every operation

Better than Memcache at caching

- Redis was able to learn from failings in Memcache
- Can store more than just strings; can store lists, sets, etc
- Allows for more complex uses

Amazing scale out capabilities

Constraints and issues

Does not sync constantly to disk; some data loss expected

No querying

Can be costly to keep everything in RAM

Common Uses

Session Storage

Full-page caching

Real time statistics

Recent posts

Scoreboards

Pub / Sub

Redis Syntax

Using Redis CLI

After installing Redis and the Redis CLI, you will be able to interact with Redis by:

- Making sure your Redis server is running
- Running the *redis-cli* command

You can test your install with the CLI

- Run: *redis-cli PING*
- You should get a response back of: *PONG*

Syntax

You can interact with Redis through a series of simple commands.

- http://redis.io/commands

The lecture code demonstrates many of these.

Storing Complex Data-Types

As usual, the internet has found a better way to do things and solved an issue for us before we even had the issue in the first place:

- https://medium.com/@stockholmux/store-javascript-objects-in-redis-with-node-js-the-right-way-1e2e89dbbf64#.fn548p1f9

This approach has ups and downs. YMMV.

Redis Use Cases

Session Storage

What is a session?

- A session is a period of communication between a server and a client of some sort, where data is temporarily stored on the server and persists between requests

How Redis helps

- Since you can store arbitrary data as either a hash or a JSON string, you can store all sorts of data about users and (using a middleware) read that data on each request.
- Redis will allow you to store entries out of the server's memory, and can easily handle expiration

Full-page caching

What is full-page caching?

- Full page caching is when you cache the contents of a page for quick output
- Often, your pages will change far fewer times across loads.

How Redis helps

- In general, acting as a cache allows Redis to greatly increase load time; it is often far faster to pull the contents of a page from Redis, rather than regenerate the view each time.
- Redis can set your page to expire after a certain amount of time so that it is not cached forever
- Since Redis can store a great deal of content, you can store many pages; you can also store user specific pages so that if there are any differences between users you can cache the same page for each user.

“Recent posts”

What are “recent posts”?

- Recent post lists are any list that are sorted by some factor; in terms of many websites, this is a listing of recent content that is created.

How Redis helps

- Redis supports a list data structure, and allows you to quickly store a list of IDs of recent posts (or whatnot) and query what entries are “new”

“Scoreboards”

What are “scoreboards”?

- “Scoreboards” are metrics that are sorted based on their data; in general, it’s the equivalent of saying: “give me the top 100 users who [DID X]”

How Redis helps

- Redis allows you to sort based on custom formulas.

Pub / Sub

What is pub/sub?

- Pub / sub is an event driven way of communicating between processes.

How Redis helps

- Redis supports pub / sub out of the box; there is no real configuration to do.

Implementing Session Storage

Recap: sessions

One of the easiest ways to keep track of users is to give them a cookie with a UUID and have that UUID associated with relevant data

- Often achieved with static memory on the server
- Also achieved with many database queries

Sessions are particularly expensive.

- If you keep the data on the web server in memory, you risk slowing down when many sessions are going on.
- If you keep the data on your database, you have slow operations to read and update it constantly

Why?

Sessions are perfect for Redis.

- Transient in nature; they expire
- Memory based for fast retrieval / updating

You can have more than one web server, so you can't trust static memory

Database servers care about long-term data and are great at storing data, but not so great at constantly reading / updating it.

Sessions **tend** not to be important in the long-term, so if the session dies it's not a huge issue.

How?

The session algorithm does not particularly change at all:

- Decide on a key to share with the user.
- Store that data in Redis
- Set an expiration time
- **Each time while updating the session object, you will update the expiration of the entry**

Details

We have two options that we can implement:

- Store the user as a hash
- Store the user as a JSON string

The process of injecting session storage into Express is fairly simple:

- Add a middleware that runs before the views are generated
- Check if the session exists in memory
  - If so, use that
  - If not, store a session
- Pull in the associated data

Benefits analysis over traditional session storage

There are many benefits to caching your session via Redis:

- You can have multiple Redis servers in a cluster to store a great deal of info
- It is extremely fast and flexible
- Can access from many locations atomically

Implementing Full Page Caching

Why?

Many applications have forms of content that do not particularly change much.

- Static home pages
- Text-heavy posts, where only portions of the page change (blogs with comments)
- Listings that only update daily, hourly, weekly, etc

Sometimes, this content takes quite a great deal to generate, despite not changing often!

- Many queries to get a blog post list and its author, related posts, etc
- If your data hasn't changed, why should anyone have to re-render it?

It is often useful to store the entire content of a page in memory, and make constantly updating parts sit out of memory:

- For a recipe page, store all the content in a cache and have the comments load over an API call with a much smaller cache time

How

We can drop in the *express-redis-cache* module to easily cache content

- https://www.npmjs.com/package/express-redis-cache

With this module, we would configure routes to have their results cached automatically

- Cached results would pull the response from the cache, not by recomputing.

Details on how

There are many nuances to caching full pages

- You may want to store an individual entry per user (IE: user home pages)
- You may want to not-cache certain pages
- You will need to decide how long your data is valid for; some pages it could be 5 minutes, some it could be 5 days.
- You have to decide if it is always cached, or selectively cached (ie: page is not cached for logged in users)

You will have to split content up between *cacheable content* and *non-cacheable content*.

- Things that update often, like comments, could be created dynamically with the use of some frontend-code to query a route over AJAX for comments and the rest can be cached
- The comments itself can be cached; you can have the page be on a long expiration, and the comments on a small one.