Data extraction for Reddit

Alexandru Pavel

August 2, 2021

Contents

1 Introduction

Build a strategy to download and store all reddit posts and comments (including upvotes and downvotes) for a given subreddit (eg reddit.com/r/sanfrancisco). Write down an executable script in any language to run your strategy. Storage of your choice among Redis, MongoDB, or Mysql. Up to you choose which one you think fits best and/or you are more familiar with.

The subreddit choice will be /r/sanfrancisco, but for testing I will use a smaller subreddit like /r/haxball/.

2 Technologies and choices

Scripts will be written in **Python**, using *requests* to access Reddit's API.

Docker, along with docker-compose, has been used to run a MongoDB instance. Mongosh was used to connect to it to perform data cleanup and queries to verify the data was saved correctly.

2.1 Storage

The nature of Reddit posts suggests choosing **MongoDB** as the storage layer. A Reddit post is quite a structured object, and it might be useful to be able to access its internal data with ease (upvotes, content, comments, etc..). Also a NoSQL DB allows to scale well with increasing data requirements.

Data in MongoDB will be stored as follows:

- db: a single database containing all the data
- posts: a collection containing all the posts in the subreddit. Data is going to be fairly homogeneous here, apart from the post text and optional media attachments (not considered). It might be useful to keep track of when the data was fetched, to keep the ups/downs updated, and perhaps to know when new comments are posted (how?).
- $< post_id >$_comments: a new collection for each post, containing all of its comments. Comments are going to be dynamic, and are required to be periodically fetched again. Having a separate collection allows to logically separate each comment tree for each post, making it easier to keep the data fresh. This might not be an optimal choice in the end, because of the high number of posts. It could be better to save the comment tree inside each post, and then access the data programmatically.

Another possible choice is to have different collections for each subreddit to fetch. Then, each document inside will be a post with all the comments as an attribute. For the purpose of this homework such strategies have not been implemented.

The other storage DBs might not be a good fit for this assignment:

- **Redis**: Key-Value, would not allow to access the internal structure of a single post easily (need to parse manually). On a positive side, Redis might be greatly useful in storing post/comments ids and allowing to have quick lookups between them.
- **MySQL**: an RDBMS, even harder to store heterogeneous data and access it. Also when data scales up it cannot make use of clustering to keep performance up.

2.2 Reddit API

Pure Python has been used to obtain the necessary api keys needed to make requests. The usage of the package **Praw** could have been investigated (seems easier?).

The API calls that have been used are:

- POST /api/v1/access_token
- GET /r/sanfrancisco/new

- GET /r/sanfrancisco/comments/article
- GET /api/morechildren: specifying important parameters as a dictionary, like the post_id

The JSON format returned by post/comment api calls is not obvious to understand. Here it's documented in a readable way: `https://github.com/reddit-archive/reddit/wiki/JSON`

A Reddit post structure is defined by many fields, the most relevant are:

- title: post title
- text: post content
- ups: number of upvotes
- downs: number of downvotes
- id: base36 unique post identifier in a subreddit
- kind: t3 for regular posts (link)
- score: net-score of the post (ups - downs)
- upvote_ratio: ups/downs
- created_utc: time of creation in UTC time
- author: name of the post author (username, might be [deleted])
- author_fullname: id36 of the post author (absent if user is deleted)

A Reddit comment structure is defined by many fields, the most relevant are:

- title: post title
- author: the name of the post author (username, might be [deleted])
- author_fullname: the id of the post author (absent if user is deleted)
- body: raw text of the comment
- ups: number of upvotes
- downs: number of downvotes
- id: base36 unique comment identifier
- parent_id: the parent comment in the comment tree, or the post if it's a top-level comment
- kind: t1 for a comment
- replies: a list of comments that are replies to the current one
- link_id: full id of the post where the comment is
- created_utc: time of creation in UTC time

3 Strategy

3.1 Post extraction

A strategy to fetch all the posts in a subreddit could be to obtain all the posts beginning from the latest one currently available (sorting by
new, up to the oldest one (using the *after* parameter in a request. Such posts might be numerous, but will be static in the end (except modified posts or deleted ones). This approach has been pursued fetching 100 posts at a time, and then obtaining all of the comments for each post.

For the new posts arriving a streaming approach is the solution: using the latest saved post_id, and specifying the *before* parameter, it's possible to fetch the posts arriving in real time (and also fine-tune the parameters for crawling frequency and number of posts obtained).

It could also be interesting to fetch the top voted/hot posts or comments, to have more valuable data in less time, effectively catching up to recent trends and discussions. This approach could also allow to examine the most popular data instead of just the newest.

3.2 Comments extraction

To add the comment tree to MongoDB the *Parent References* pattern has been used `https://docs.mongodb.com/manual/tutorial/model-tree-structures-with-parent-references/`. For each comment its parent id is memorized along with the comment _id and other attributes.

When a post's comment number exceeds the *limit* value in the parameters a **more** kind of comment will be returned. This requires to perform a series of API calls until there are no more remaining comments to fetch for that post. Here a choice could be made to only retrieve comments up to a certain depth in the tree, making the process faster at the cost of losing some data.

4 Conclusions

The scripts to extract the data work, even though they weren't tested for a longer period of time to expire the api-key. Also when data scales up attention must be placed on not overloading the servers with too many requests, or the current credential could be invalidated requiring a new one. Some perplexities on how the data should be structured for persistence still remain, and mainly depend on the application requirements. For the database/collections structure chosen here the usage of a key-value db might be suitable, to map posts to comments faster for easier access.