Felix Yang and Ayushi Shirke

DS4300 HW2 Report

Strategy 1	postTweet	16793.0
	getHomeTimeline	46.1
Strategy 2	postTweet	4833.1
	getHomeTimeline	2143.3

Our code is implemented in a driver.py, which runs both strategy implementations, and a corresponding RedisTwitterAPI_1 and RedisTwitterAPI_2 for each of the strategies. In our implementation, we found that it would be more efficient to implement each of the redis database connection commands within the API itself because if the commands were extracted, then it would result in unnecessary delegations of single-line function calls.

Strategy 1:

We first used a list called “followers:{user_id}” to represent the “follows” relationship, but this list represents the users as the keys and their values are the users that follow the user as the key. In other words, the “follows” relationship is represented more accurately as a “followers” relationship, so that the tweets would be posted to all the user’s followers. We also have the “follows:{user_id}” key to have a list of users that this user follows

We used a sorted set that would have a unique identifier for each element as an auto-incremented index. We set the tweet with its index as a hash of the tweet properties. We also “lpushed” to “users:{id}” the tweet key value. In getting the timelines, we look through the followers of randomly generated user with Redis’s “srandommember” and then through the tweets of the user and adding that to the above sorted set before getting the most recent 10 tweets via “zrange”.

Strategy 2:

We used the same list to represent the followers from Strategy 1, and from here, we posted a tweet using its index as a hash of the tweet properties as well, but this time we first obtained a list of all the users the current tweet’s poster follows. Then, we loop through each of the followers that the tweet-posting-user follows, which results in the tweet object being pushed onto lists for each of the user’s home timelines called “get_user_timeline{user_id}”. Since there is a for-loop for each tweet that is posted, this process is considerably slower than it would be to read a user’s home timeline which is already generated from this process.

Next, we generated a random user using Redis’s “srandommember” method, and we found that the home timeline can simply be looked up and the first 10 posts can be returned since they are the most recent, which is guaranteed by only using the “lpush” function and avoiding any multi-processing or concurrency. We then recreate the Tweet object using the hashes of the tweet ids and return the timeline.