Brian Reicher

Dr. Rachlin

DS4300

10 February 2023

Redis Twitter Backend Analysis

For the next stage of my implementation of Twitter's backend in a non-relational database format, I extended my initial Java implementation with our Singleton architecture to utilize Redis. All API calls were run on a 16" Macbook Pro with MacOS 13.1 (22C65). It has a 2.3 GHz 8-Core Intel Core i9 processor and 16 GB of RAM with a 16 GB 2667 MHz DDR4 chip. I was able to implement both methods described in the assignments, and in my code the "DPDatabaseRedis" API corresponds to the required implementation, and the "DPDatabaseRedisEC" API corresponds to the optional extra credit. All API call metrics are shown below.

API Method	**API Calls/Sec**
postTweet (required implementation)	4111.9
getHomeTimelines (required implementation)	1883.0
postTweet (optional implementation)	6600.2
getHomeTimelines (optional implementation)	500.7

These rates are all significantly higher than the numbers for the MySQL implementation, showing the advantages of Redis & non-relational databases in our use-case. However, compared to the overall class baseline, my rates are still in the middle-end. I believe that my main limitation here -- other than my hardware -- is because I was using loops to transform lists of strings into lists of integers. Potentially attempting to do this with a map instead may have led to faster performance. As far as the discrepancy

between the two Redis implementations, I think the required Posting method clearly was slower since the API needs to do all of the same work as the EC method, while also getting followers of the tweet's user and pushing the tweet to their timeline bucket with a loop (killer for time complexity). Eliminating this to construct the timelines "on the fly" in the optional implementation sped up posting significantly, however fetching timelines was dramatically slowed down. I believe that this is a result of the fact that the optional getHomeTimelines implementation utilizes multiple loops to stream the tweets directly to timelines without buckets. It needs to build extensively on top of the simple getRecentTweets method used in the required implementation, and ends up performing almost ¼ as well.