DS 4300 HW1: Twitter in an RDB Analysis

Avril Mauro & Katelyn Donn

POST: 1671.31 TWEETS/SEC

RETRIEVE: 1.2 TIMELINES/SEC

OS: MAC

LANGUAGE: PYTHON

SERVER: MySQL

STORAGE ENGINE: MySQL WORKBENCH

PREPARED STATEMENTS/SECONDARY INDEXING: NO

RAM (GB): 16

PROCESSOR: APPLE M1

Our code consists of two main programs: twitter_loader.py and twitter_timelines.py. Twitter_loader.py is a program that reads 1 million tweets from the tweets.csv file, initiating a Tweet object to keep track of each tweet's ID, timeline, text, and user ID using a class we built in our twitter_objects.py file and loading it into our TwitterDB database using the UserAPI class we built in our twitter_mysql.py file. The Python script that we wrote takes roughly 10 minutes to post 1 million tweets, averaging to about 1671.31 tweets posts per second. Our second file, the timelines file, loads in the data from the follows csv, shuffles a list of unique users, and randomly selects a few users. For each user, we grab the user's followers, following, and the tweets of those they follow. After analyzing the frequency of timelines generated in the span of a minute, our retrieval rate averages out to 1.2 timelines per second.

Our program operates slowly because we are using a relational model to design our database. We have two tables, Tweets and Follows, but in order to retrieve timelines we need to join those tables using a relationship between user_id and follows_id each time, and then query the joint table on most recent tweets with a limit of 10. Rather than treating each user as a profile that can post tweets and follow people who can also post tweets, all this data is organized as a series of rows mapped by user_ids in multiple tables. This exposes the limitations of a relational model because our dataset is so vast that these intermediate steps cannot keep up with in real life application. The real twitter app would never run this slow. The relational model promotes excess computing that lowers our processing speeds and becomes our ultimate drawback.

In order to improve this process, we need to release ourselves from the constraints of a relational database and move towards a design that can store all the data for one user in one place. In our next assignment, we will use Redis, a key-value store, to experiment with this. With this design, we don't need to rely on table relationships to retrieve info such as a user's following list. Instead, that information will be stored as a value for the key of user_id. Currently, to retrieve this information we need to look up user_id in the Follows table and select the multiple rows that exist. The difference in computing power and processing speed will be very interesting to observe.