Liam Miller

Computer Specs:

CPU: Intel® Core™ i9-14900HX processor with 24 physical cores (32 logical processors) and a maximum clock speed of 2.2 GHz.
Memory: 32 GB of RAM.
Disk: 1 TB NVMe solid-state drive (SSD).

Software Specs:

Programming Language: Python 3.14.2
Database Driver: `mysql-connector-python`
Relational Database: MySQL 8.0.45 (MySQL Community Server), running inside a Docker container
Containerization Platform: Docker Desktop for Windows (Docker version 29.1.3)

Results:

API Benchmark Test	Calls/sec
postTweet	1530.8
getHomePage	394.5

Analysis:

My program can't keep up with the post tweet quota, though does come somewhat close, however comes nowhere near to the get home page quota. For the postTweet test, my initial tests were only calling between 60-80 calls a second, which was not good. The problem I found with my initial code was that the system opened and closed a database connection for every request. In addition, each tweet insertion was committed as its own transaction, forcing MySQL to perform expensive disk synchronization operations on every request. Once I made the system use one database connection and defer disk synchronization, the calls sped up drastically. I did notice in my testing that while I was using my computer outside of the program running, the speed dropped from its initial peak of 1556 to a low of 1512, and when I stopped using it the average came out to what it ended up as.

While you would only need 4 to 7 of my computers running my post tweet api to keep up, you would need 500-770 to keep up with the home page. There is clearly something I can do to improve this speed, but I couldn't figure it out. It does make sense that the calls are less than the posttweet test however, given what both tests are trying to do. For the test, the code selects a random user and queries the database for the ten most recent tweets from users they follow. While some small query optimizations were applied, such as using index lookups, each request

still requires multiple reads and sorting, which limits how much the performance can be improved using a relational database.

Note: The github contains only 1 dataset as I realized that the tweet dataset was too large to upload, and I just ended up doing the tests locally.