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).

Results:

	Calls/sec (mysql)	Calls/Sec (Redis)
postTweet	1530.8	641.6
getHomePage	394.5	624.6

Redis Model Details:

In the Redis model, tweets are stored using keys formatted as tweet:{tweet_id} and are implemented as hash structures containing the author ID, timestamp, and tweet text, allowing tweet content to be stored once while timelines reference tweet IDs. Follower relationships are stored as sets using keys formatted as followers:{user_id}, which contain the IDs of users following that account and allow efficient iteration when distributing tweets to follower timelines. User timelines are stored as sorted sets using keys formatted as timeline:{user_id}, where tweet IDs are stored as members and timestamps are used as scores, enabling fast retrieval of the most recent tweets. Additional helper keys include next_tweet_id, a string counter used to generate unique tweet identifiers, and all_followers, a set used to track users for benchmarking operations.

Analysis:

Contrasted to the mysql, the Redis one actually got faster as the postTweet benchmark went along, starting at 489 tweets/sec but ending at 641 tweets/sec. Still, it was much slower overall than the mysql one at the post tweet benchmark, although neither passed the test for what would be needed for twitter overall. The slower result of Redis is because each tweet had to be copied into all of its followers' timelines as soon as it was posted, slowing down the rate that each tweet could be processed. While this benefitted the home page rate (which I'll get to next), it slowed down the post tweet processing rate.

The Redis code outperformed the mysql calls when it came to the home page refreshing, although the difference was greater in the post tweet calls (2.4 times bigger compared to 1.6 times bigger). The Redis call rate was quite consistent for the get home page rate. The aforementioned benefit that Redis had was the tweets being immediately put into the followers

timeline, which drastically sped up the get home page rate. The get home page and posttweet rate for Redis were actually very close, with a difference of 17 calls a second, compared to the 1100+ difference in calls a second that was experienced when using mysql.

Note: The github does not contain a dataset as I just ran the files locally