DS 4300: Homework 2
Twitter performance with Redis

DESCRIPTION

In this assignment we'll use Redis, a Key-Value store, to implement a Twitter API and we'll compare our performance to the read / write performance you measured in Homework #1. You'll have some design choices to make in terms of what distinct keys you manage, and whether to put raw data into each value, or simply references to other keys in order to reduce storage overhead. As before, we are interested in the performance of two basic operations:

A) Post a tweet by a random user
B) Obtain the home timeline of a random user (i.e., for a random user, get the tweets of all the users that your randomly chosen user follows.)

IMPLEMENTATION

As I alluded to in class, we want our application to be well-structured such that it makes use of a clearly-defined twitter API. The twitter API should define an interface that include operations for reading and writing to the backend database, whatever that might be. If you already did this in Homework 1, your task for Homework 2 is to simply re-implement the API to use Redis rather than MySQL!

At a bare minimum, your twitter API should include operations such as:

```
void postTweet(Tweet t)
void addFollower(long follower_id, long followee_id)
List<Tweet> getTimeline(long user_id)
```

You might also want:

```
List<Long> getFollowers(long user_id)  // who is following user_id
List<Long> getFollowees(long user_id) // who is user_id following?
List<Tweet> getTweets(long user_id) // tweets posted by user_id
```

Having these might make your implementation of the required operations simpler!

ANALYSIS

There are two strategies for implementing the postTweet and getTimeline operations.

__Strategy 1__: When you post a tweet, it is a simple addition of a key and a value. But the getTimeline operation would now require that you look up the tweets of each of your followers, constructing the home timeline on the fly on demand.

__Strategy 2__: As you post each tweet, you copy the tweet (or a reference to the tweet) to the user's home timeline automatically. This is actually what twitter does! Your write performance should be lower, but since the timeline is now ready and waiting, getTimeline should be a much faster operation. Let's find out how much faster.

Report the postTweet and getTimeline performance for both strategies. (Altogether, I am looking for four numbers.)

How do the two strategies compare?
How does Redis compare with MySQL?

GRADING

Submit your code, including a documented Twitter API and its Redis implementation.

You will be graded on the quality and completeness of your code and the design and implementation of your Twitter API.