Analysis

1. Hardware Configuration

CPU: 3.1 GHz Dual-Core Intel Core i5
RAM: 8GB

2. Data Modeling Assumptions:

Number of users: 10,000 (user_id from 1000001 to 1010000)
Number of tweets per user: Randomly distributed
Distribution of the number of followers per user:

(a) For user_id between 1000001 and 1001000 (1,000 users): follows 100
(b) For user_id between 1001001 and 1002000 (1,000 users): follows 300
(c) For user_id between 1002001 and 1003000 (1,000 users): follows 500
(d) For user_id between 1003001 and 1001000 (7,000 users): follows 10

Total follows relation: 970,000 lines

Time Stamp: Between 2021-01-01 and 2021-01-31
Tweet_text: Randomly selected from upper and lower case letters and all punctuations with random length

3. Strategy Description:

Strategy 1:

(1) For each follow relation, set user_id as key, and add user_id of the users he/she follows to a set
(2) For each tweet, set tweet_id as key and the tweet_text as value
(3) For each tweet, set user_id as key, and add the tweet_id of the tweets the user posted to a list
(4) When retrieve the timeline, first get the set of users that this user follows. Then, for each of the user in the set, get all tweet_id of the tweet they post and add them to a python list (l_tweet_id). Next, sort the l_tweet_id by the id in reverse order and get the top ten ids. Those ten ids are the ten most recent tweet_id in this user's timeline (tweet posted latter has larger tweet_id). Finally, transform the tweet_id to tweet_text.

Strategy 2:

(1) For each follow relation (A follows B), set the user_id of user (B) that's being followed as key, add the user (A) to the set.
(2) For each tweet, set tweet_id as key, and the tweet_text as value
(3) When inserting a tweet, get all the followers of this user as a list (l_followers), then for each of the follower in l_followers, use its user_id + "timeline" as key, add the tweet_id to its corresponding list.

(4) When retrieve the timeline, just get the first ten tweet_id in this user's timeline, and then transform them to twee_text. (Latter added tweet_id is posted latter, already in reverse order)

4. Results

API Calls / Sec	Relational	Redis without Broadcasting	Redis with Broadcasting
Post Tweet	2513 tweet / sec	2451 tweet / sec	92 tweet / sec
Get Home Timeline	0.33 timeline / sec	5.25 timeline / sec	895 timeline / sec

As you can see, post tweet with broadcasting is much slower than post tweet without broadcasting. There're 970,000 follow relations, and each user is followed by 97 users on average. Therefore, when inserting a tweet with broadcasting, we need to update nearly 97 user's homepage. That will make the inserting speed really slow.

The speed for retrieving by using strategy one is much slower than the speed of retrieving by using strategy two. Some of the users follow 500 other users. When retrieving the timeline for them by using strategy one, we have to iterate all of the users in its following set and add all tweets post by those users to a list. Finally, sort the list and get the first ten tweets. On average, one user follows 97 other users and post 100 tweets. When retrieve the timeline for one user, on average, it has to iterate 97 * 100 times. That's time consuming. However, as for retrieval for strategy two, we only need to get first ten tweets in the user's timeline. That's simple.

The insertion speed of tweet without broadcasting is similar to the insertion with RDB. However, I have tried to use Java to do the insertion with Redis. The speed is about three times of the code written by python. One way may increase the speed is that just retrieve tweet_id instead of the tweet_text for timeline. Therefore, we don't need to connect tweet_id and tweet_text when inserting the tweet. The insertion speed may increase to some degree.

There's a huge difference of retrieval speed between Redis and RDB. Retrieval speed of using strategy two is about 2700 times of using RDB.