DS 4300: Large-Scale Storage and Retrieval
Prof. Rachlin

Assignment 1: Twitter - The Early Years

DESCRIPTION

When Twitter first started out, its engineers used MySQL as a backend database. As the service grew in popularity, they were forced to abandon MySQL in favor of a NoSQL key-value store (Redis). In the assignment and the next, you will play the role of a twitter engineer trying to understand the performance limitations of MySQL and the potential benefits of a key-value store.

There are two key functions of twitter. First, it allows users to post tweets. Second, it allows a user to see all the tweets posted by people that the user is following. This set of tweets – which the user sees when he/she opens up the twitter app on a smart phone – is known as the user's *home timeline*.

In this assignment you'll push MySQL to its limit by seeing how fast you can post tweets and retrieve home timelines.

DATABASE AND DATA SETUP

1. Implement a simple relational database to manage users and their tweets. Your database should have two tables:

TWEETS – The tweets posted by users

tweet_id	long
user_id	long
tweet_ts	datetime
tweet_text	varchar(140)

FOLLOWERS – Who follows whom

user_id	long
follows_id	long

You don't need a user table or a hashtag table and you can use any relational database you want but document your choice.

2. Write a tweet generator that creates 1 million random tweets and saves these tweets to a file - we will need them again later. The tweets can be random words from some corpus of your own choosing, or they can be complete gibberish.

3. For the followers table, generate a dataset consisting of pairs of user IDs. Again, save this data to a file – we'll need this later as well. Then pre-populate your FOLLOWERS table with this data. Decide for yourself how many different users should be responsible for this collection of 1 million tweets and how many followers or followees each user should have. As in real life, perhaps the number of followers should follow a distribution, with some users having many followers, and many users having very few. Whatever you ultimately decide, *document your modeling assumptions.*

PERFORMANCE TESTING

1. Write one program that reads pre-generated tweets from the file, *one record at a time*, while then inserting the tweet into the database. Do not use batch insert utilities such as **mysqlimport**, but it is ok to use prepared statements. Keep track of how long it takes to load all of the tweets. This program is intended to simulate an incoming stream of tweets. How many tweets can be posted per second? Twitter receives 6-10 thousand new tweets per second. Can your database, running on a laptop, keep up with this load?

2. Write a second program that repeatedly picks a random user and returns that user's home timeline. For example, if Joe follows Ann, Beth, and Charlie, then Joe's home timeline consists of the 10 most recent tweets posted by Ann, Beth, or Charlie. This process simulates users opening the twitter app on their smartphone and refreshing the home timeline to see new posts. How many home timelines can be retrieved per second? (Twitter users worldwide collectively refresh their home timeline 200-300 thousand times per second. Can your program keep up with this load?)

ANALYSIS AND REPORTING (Be Creative – but here are some general guidelines)

1. Clearly document your hardware configuration (CPU speed, number of cores, RAM, Disk etc.) and data modeling assumptions: number of users, number of tweets per user, distribution of the number of followers per user, or any other parameters or database settings that might impact your results.

2. Report two performance results:
    a. The number of tweets inserted *per second*
    b. The number of random home timelines retrieved *per second*

GRADING

You will be graded on your implementation, the quality and architecture of your code, your experimental design, and the thoroughness of your analysis. **You should have a well-defined API (Interface in Java) and clearly defined API implementation so that it is relatively easy to swap out a relational database for a NoSQL database such as Redis**. Don't worry if your performance is really slow – or slower than your fellow students. This isn't a competition and there are many factors that can influence the results. Let's figure out what those factors are!