DS 4300: Relational vs. NoSQL

The Twitter Case Study

DESCRIPTION

When Twitter first started out, its engineers used MySQL as a backend relational database. As the service grew in popularity, they were forced to abandon MySQL in favor of Redis, a NoSQL key-value store. 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 operations of twitter that our code needs to support.

a) Users post tweets. We are ignoring hashtags.

b) Users retrieve all of the tweets posted by all users followed by that user. This set of tweets – which the user sees when he or she opens up the twitter app on a smart phone – is known as the user's *home timeline*.

In this assignment you'll push your relational database to the limit by seeing how fast you can post tweets and retrieve home timelines. In the next assignment, we'll swap out our relational database implementation in favor of Redis.

DATABASE INITIALIZATION

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

TWEET – The tweets posted by users

tweet_id	INT (PK)
user_id	INT (FK)
tweet_ts	DATETIME
tweet_text	VARCHAR(140)

FOLLOWS – Who follows whom. The user "user_id" follows the user "follows_id"

user_id	INT (FK)
follows_id	INT (FK)

You don't need to worry about a user table or a hashtag table. Technically, the listed foreign keys are foreign keys to the user table which is not being implemented for this exercise. You may add additional secondary indexes to your table design as you see fit.

2. Load the data in *follows.csv* into the FOLLOWS table. You can do this programmatically or by using data importing utilities such as *mysqlimport* (for those of you using MySQL).

PERFORMANCE TESTING

1. Write one program that reads pre-generated tweets from the file *tweets.csv*. Note that the file contains just the user_id and the text of the tweet. Your code (or the database) should auto-assign tweet_ids and timestamps as the tweet is loaded into the database. Keep track of how long it takes to load all of the tweets. This program simulates users posting tweets. How many tweets can be posted per second? (Twitter receives 6-10 thousand new tweets per second. Can MySQL keep up?) Insert tweets as you read them from the file. Batch no more than 5 tweets at a time into the insert.

2. Write a second program that repeatedly picks a random user and returns that user's home timeline: The 10 most recent tweets posted by any user that that user follows. For example, if user A follows X, Y, and Z, then A's home timeline consists of the 10 most recent tweets posted by X, Y, or Z. 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?

REQUIREMENTS

Your programs should be properly engineered with a clear separation between the driver program that carries out the performance testing, and the API which implements the twitter-related function. For example, at a bare minimum, your twitter API should include operations such as:

```
void postTweet(Tweet t)
List<Tweet> getTimeline(Integer user_id)
```

You might also want:

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

Implement your code so that the driver code which is calling these API methods does not need to know anything about the underlying database implementation. It is simply making API method calls. In this assignment, the *implementation* of that API will use a relational database. In the next assignment, you will re-implement the API using Redis. Your driver program should be *unchanged.*

ANALYSIS AND REPORTING

Document your hardware configuration (CPU speed, number of cores, RAM, Disk etc.) and software stack (RDB and version, Redis and version, programming language, libraries used, etc.)

Report TWO numbers in units of API calls per second:

API Method	**API Calls/Sec**
postTweet	?
getHomeTimeline	?

Describe any factors you think might have impacted your results.

GRADING

You will be graded on your database implementation, the quality of your code, your profiling code, and the documentation of your methodology. *We will compare results in class and I may ask several of you to present your analysis to the rest of the class. 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 might be!*

WHAT TO SUBMIT

Code files and a PDF of your analysis.