Problem Solving Workshop #46 Tech Interviews and Competitive Programming Meetup

June 9, 2019 https://www.meetup.com/tech-interviews-and-competitive-programming/

Instructor: Eugene Yarovoi (can be contacted through the group Meetup page above under Organizers)

More practice questions: leetcode.com, geeksforgeeks.org, hackerrank.com
Books: Cracking the Coding Interview, Elements of Programming Interviews
Have questions you want answered? Contact the instructor on Meetup, or ask on Quora. You can post questions and follow the instructor and other people who write about algorithms.

Try to find optimized solutions, and provide a time and space complexity analysis with every solution.

"Distributed Counter Service" (System & Algorithm Design)

Your goal is to design a distributed counter service. A counter service is a service that can be used by many software components to keep track of real-time counts of events, for example, the number of upvotes on videos or the number of clicks on search results.

The desired API contains the ability to increment a count associated with a key, and then to query how many such increments occurred in some past period of time. The most basic API would look something like this:

incrementKey(string key) -> void
getKeyCount(string key, double timeInSecondsToLookBack) -> long integer

So, for example, getKeyCount("foo", 600) would return how many times the string key "foo" has been incremented in the last 600 seconds. Note that this is a **distributed** system: different machines may be involved in incrementing the keys vs. reading the keys, and the result of an operation like getKeyCount represents the total number of times the counter has been incremented from all sources.

To understand example uses of this service, here are some examples:

1. The number of times a user logs attempts a login is counted by submitting keys like ("password_login_<userid>") to be incremented. Later we query how many times that key has been incremented in the last 15 minutes, to see if the user has exceeded allowable login attempts.
2. When a video is upvoted, we increment a key ("video_upvote_<video_id>"). This is used to later show the video upvote count in the last month.

You can (and perhaps should) enhance the above basic API a little with additional functionality (to give one hint, you should probably think about how you will design the API to allow some information to expire, so it isn't kept forever).

Yet, most of the focus should be on how you will support some API similar to the above at large scale. There could be millions of key increments / key reads per second that this system will have to service. Be sure to address:

- What data will you store and how will it be stored?
- What performance / reliability tradeoffs does your solution have? Are there any levers that can be tweaked in your solution to trade off differently?
- Is there any additional information you would accept in the API to help your solution perform better?
- For many applications, there is no hard requirement for the counts to always be super exact -- approximate counts are good enough. Can you significantly improve performance / storage requirements in that case?

Note: While it's true that in practice, you would probably use a counter service component that is already built, the goal of this question is to design the service yourself from more basic components. Companies test your ability to reason about these kinds of systems because services like this are in fact often built internally, to handle additional use cases that existing solutions don't handle well. Even though you are building the counter service from scratch here, don't assume its subcomponents must be built from scratch -- it's fine to say you will use a database as a subcomponent of this system, for example.

"Uber Eats" (High-Level System Design)

Consider the Uber Eats or Doordash app. In short: users can view menus of restaurants near them from where they can order food to be delivered to them. Upon ordering, the order will be sent to the restaurant and a driver will be dispatched to pick up and deliver the food to the user.

What are the core components of a system like this? What would you need to implement, from a software perspective, to build a basic working version of this kind of app? Try to be comprehensive and discuss all the major components and how they will work together.

You might also discuss how the relevant data will be stored and how some core algorithms, such as matching a driver to the order, will work. While scalability is not the primary focus of this question, you should see if you can design the data storage and algorithms in a way that will scale.

This type of design question is used in interviews to assess one's ability to plan larger software projects and anticipate what components will be necessary to accomplish a task.

Youtube Recommendations (Feature Brainstorming + possibly ML)

Design the feature that shows a user "recommended next videos" on Youtube when they're watching or after they watch a video. How do you select which videos to show the user next?

At a basic level, you don't need a machine learning background here. You can start by discussing what heuristics you would use to select the videos to promote to the user. Consider what might be all the sources from where you can derive recommendations.

If you have some ML background, you can consider how you might train ML models to make the prediction? What features would be involved, what would the learning objective functions and labels be, and how would you evolve the system over time?