Twitter Thread by Sunil Kumar

Sunil Kumar
@sunilc_

How should one approach System Design questions during an interview?

Here's the step by step guide:

■■■

System design interviews are generally open ended discussions. There's no one right answer.

The main focus in this round is to see your thought process and whether you'll be able to design a minimal system keeping future scale in mind and by following the standard principles.

Know that many interviewers deliberately make it hard to see how you approach a given problem.

You cannot possibly design a complex system in 1-2 hours which engineers usually take months to years to design, develop and build.

So this is how you should approach this round:

1. Clarify the requirements: As mentioned earlier the questions are generally very open ended. So you need to ask a lot of question to set the scope clear for the discussion.

Example: "Design an application like Twitter"

Twitter has a lot of features like:
- Post tweet
- Home timeline
- Profile timeline
- Analytics
- Likes
- Retweets
- Following
- Bookmark
- Trending etc

You get the idea!

Ask the interviewer which are features you need to cover in this discussion. They will probably ask you to design 2 to 3 features.

Ask a lot of specific questions in this step and make sure both of you are on the same page.

Also clarify if you need to keep in mind the non functional requirements like:

- consistency
- availability
- durability
- scalability etc

2. Capacity Estimation & Constraints:

In this step you can assume the traffic your service gets and estimate how much data you need to store in your tables so that your system works for the next 5 - 10 years without running into any problems.

When estimating you can assume things like:
- Read / Write ratio (ex: Twitter)
- Each file size (ex: Dropbox)
- Average photo size (ex: Instagram)
- Total active users (Ex: Facebook) etc

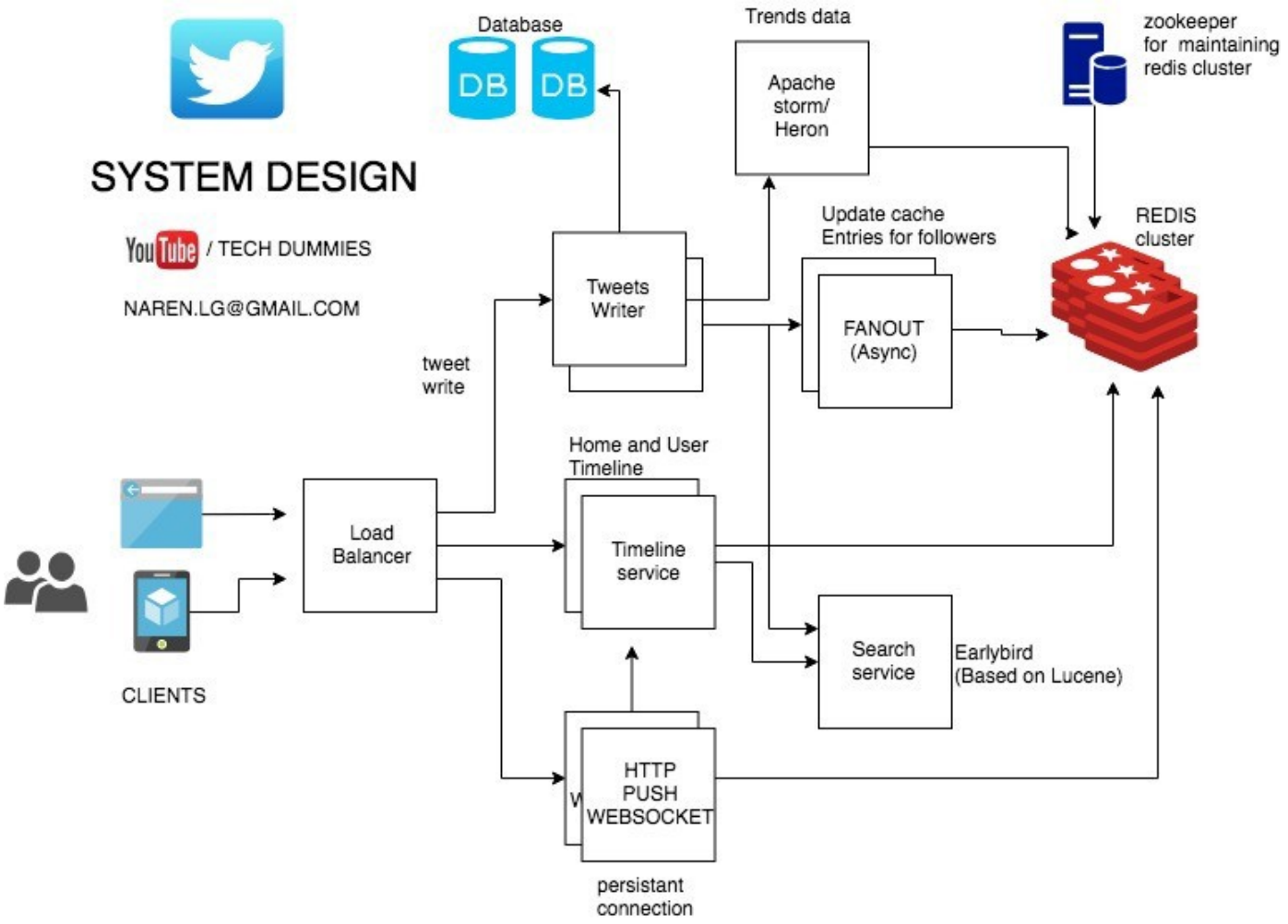
Based on the assumptions, you can calculate how much data you need to store in your tables.

Based on the data storage needs and the non functional requirements you can chose either a SQL or a NoSQL database.

3. High Level Design: This is the component design of your system. You need to list down different components that are involved in the system you're building and the data flow between them.

Ex: In case of Twitter:
- Clients
- Timeline Svc
- Search Svc
- DB Storage
- Cache

etc



4. Database Schema:

Come up with different tables, their schema and relations between them to store the data for your application.

Based on the columns and the data type of each of the column you can compute how much data you may need to store for the next 5-10 years.

Note that you need to choose between a SQL or a NoSQL based on the functional and non functional requirements given.

For example if your data schema constantly changes and scalability is a requirement, you can go with a NoSQL db.

If transactions are important then go with SQL.

5. API design: You can list down the main APIs needed for the features at hand and their corresponding request and response parameters.

6. Core Logic: Some systems have a core algorithm that the interviewer wants you to discuss about.

For example:

- in case of URL Shortener service they may ask you how you would generate the shortened url for every given long url.

- In a system like Twitter you need to discuss how you would go about getting the data needed to display on home timeline or profile timeline.

In this case you may want to precompute the timeline for users because of the huge data volume and the realtime behaviour we need.

- If you take Uber as an example, you may want to discuss the logic of finding the nearby drivers when a ride is requested.

Since this is a spatial search problem, you may want to store the data in a quad tree.

7. Data Sharding & Partitioning: Some systems need you to partition and shard the data when the data volume is really huge. Otherwise your DB queries will slow down and in turn the API calls.

Depending on the requirements, discuss about how you would do partitioning & sharding.

8. Cache: Caching is another important concept in building scalable systems.

Cache allows you to store the frequently used data and access it much faster which improves the overall response time of your APIs.

You may want to follow 80/20 rule here which basically means 80% of the requests are coming for 20% of the data.

In this case you may want to cache that 20% of the data. You can estimate how much cache storage is needed based on each entity data size and your total traffic.

There are many other things involved when designing a complex system. But the above are the important ones which you will be expected to know.

Note that the interviewer may ask you to discuss all of the above step by step at high level as discussed or any one in detail.

At any point, make sure you're communicating clearly about your thought process and mention any assumptions you're making.

I hope this thread has given you some insights into how you should approach a System Design interview round.