System Design Interview Guide (Part I)

Topics Covered: Problem Statement, Requirements clarification, High level Architecture, Data Modeling

- Problem Statement
  - May be very specific:
    - ↳ "Count YouTube video views"
    - ↳ "Count likes on Facebook | Instagram"
  - May be more vague:
    - ↳ "Calculate application performance metrics"
    - ↳ "Analyze data in real time?"

- Begging the question(s):
  - ↳ What does Data Analysis mean?
  - ↳ Who sends us data?
  - ↳ Who uses the results of this analysis?
  - ↳ What is meant by real time?

- Why requirement clarification is important
  - Interviewer:
    "I want to understand how the candidate deals with Ambiguity"

- Requirements Clarification
  - Functional Requirements
    - ↳ System Behavior: APIs (set of operations the system will do)
  - Nonfunctional Requirements
    - ↳ Qualities such as fast, fault tolerant, secure.

- Functional Requirements : API (● example)
  - The system has to count video view events

countViewEvent(videoID)

If we want to generalize our API a bit, so we can count likes & shares too. for example, we may generalize our API a bit & introduce an event type parameter

countEvent(videoID, eventType)

"view"
"like"
"share"

one step further: we can make system calculate not only count function, but other functions as well, such as sum & average.

processEvent(videoID, eventType, function)

"count"
"sum"
"average"

- Sum could allow us to calculate such metric as calculate "total watch time" for a video
- Average function could help us calculate avg. view duration

processEvents(list of Events)
- Process events as a batch in a single object

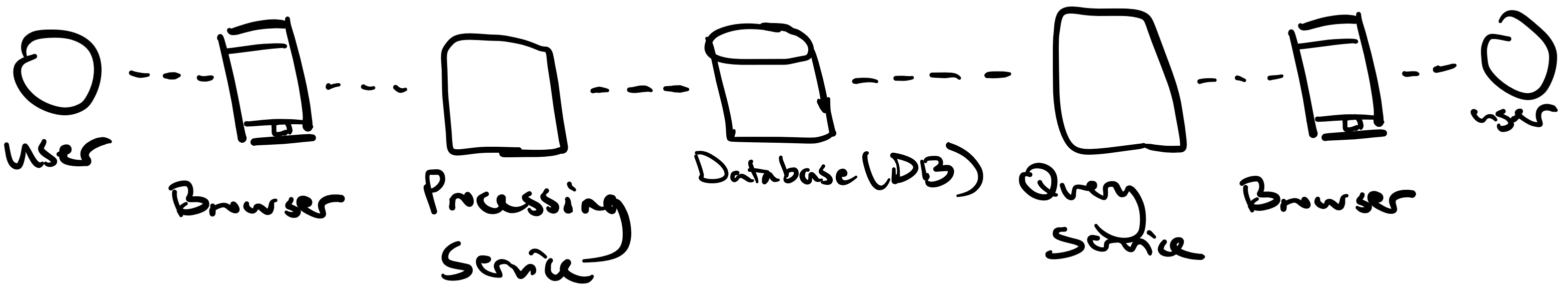
• Non-Functional Requirements
– Interviewer:
"Let's design a system that can handle YouTube scale."

"And let's make it as fast as possible."

System needs to be:
↳ Scalable (tens of thousands of video views per second)
↳ Highly performant (few tens of milliseconds to return total view count of a video)
↳ Highly available (survives hardware/network failures, no single point of failure) data

• High Level Architecture

Start Simple:



What are the pieces to the system design puzzle?

<u>Data.</u>

– We need to define a Data Model:
↳ What we store
– Individual events (every click)
Format: Video ID, Timestamp, user related info such as country, device type, OS, etc.
Pros: fast writes, can slice & dice data however we need (filtering & aggregate when needed),

Can recalculate numbers if needed

Cons: Slow reads, Costly for large scale (many events)

- Aggregate Data (per minute) in real time:
  ↳ video ID, Timestamp, Count

Pros: Fast reads (we don't need to calculate each individual event, we just retrieve total count value), Data is ready for decision making (we may send the total count value to a recommendation service or trending service, for popular videos to be promoted to trends)

Cons: can query only the way data was aggregated (filtering / changing the aggregation is hard). This implies that we need a data aggregation pipeline to pre-aggregate data before storing it, it's hard to fix bugs: say there's a bug w/view counts — how do we fix total counts after bug was fixed?

Should we store raw events, or aggregate data in real time?
We'll need the interviewer's help in making this decision.

↳ We should ask interviewer about expected data delay: time b/t when data is processed & when it happens

- If it's not meant to be more than a few mins, then we have to aggregate data on the fly.

This is known as Stream Data Processing.

- If several hours is okay, we can store raw events & process them in the background.

This is known as batch data processing.

The interviewer will tell us which approach we should focus on.

By the way: combining both approaches makes a lot of sense for many systems out there.

↳ Store Raw events, but b/c there are so many, only store them for a few days/weeks, then purge old data
↳ Calculate & store numbers in real time so that statistics are available to users in real time

We actually get the best of both worlds: fast reads, ability to aggregate data differently & re-calculate stats if there were bugs or failures

But it's expensive to do both.