Functional Reqs:

- Chrome Extension that sends an additional API request to backend whenever "like" or "dislike" is clicked
- Should be toggled clicks with some way to "track" that a user has clicked like or dislike on a video already
- Don't want users to have to explicitly log in and authenticate with our service
- We want the like/dislike count to be very close in its estimation, but it doesn't need to be real time or accurate to the order of seconds/milliseconds
- We should always be available or have really good fault tolerance/recovery since estimating the dislike count if we are down will make things very difficult to maintain correctness
- Users should not be able to spam or click "like"/"dislike" more than once
- Latency of displaying the "like"/"dislike" count can be a bit slow since the component can render after the page/video loads

Functional Reqs Cont:

- Cannot store sensitive user info like their email / phone number

Non-Objectives:

- Don't need to deal with storing the information that a user clicked "like" or "dislike" on youtube video
- Don't need to deal with user authentication

Parameters

Total Active Youtube Users Per Day = 122 million

$$= \frac{122 \text{ million}}{24 \cdot 60} = 84{,}722.22 \text{ users per min}$$

$$= \frac{84{,}722.22}{60} \approx 1{,}412.04 \text{ users per second}$$

Assuming each user clicks either "like" or "dislike" this means there is approximately 1412.04 reqs per second ← means pretty something in on order of mag = 4 greater is very unlikely

This can be better modeled with a Poisson Dist. with parameter $\lambda = 1412$ meaning that in the span of one second as an interval on average we will see 1412 requests a second

It is a pretty big assumption to say that each user is going to like/dislike every video they watch, but I think it is a fairly accurate estimate and overestimating it will not lead to too many negative consequences

Parameters Cont.

Consistency – Does not need to be absolute consistency but it should be fairly strong

Availability – We want high availability since we are not the owners of youtube any amount of downtime will lead to inaccurate "like" / "dislike" counts

Partitioning – Dependent on number of users/and size of the data/approximate total storage

Latency – No hard requirements so I will set mu to 5 seconds

Data Model

At the very least we need these models

Schema:

Video id (Varchar, 50)
Like Count (BigInt) — Highest like count for a video is 50 million so this will be more than enough
Dislike Count (BigInt)

If using a relational database:

Video id = 50 bytes
like count = 8 bytes
Dislike count = 8 bytes
) 66 bytes per video entry

Assuming a higher upper bound of 1 billion videos

66 bytes · 1 billion ≈ 66 GB of storage (ignoring metadata/indexes)

A star said there are 800 million YouTube videos.

Data Model Cont

If using a non-relational DB like Redis

Key	Value
video_id + like	# of likes
video_id + dislike	# of dislikes

Each of the key/values are variable size but the keys a good goal is a like % of 4 to be vers on a video. Can assume the dislike % is 1. Let's assume any youtube video gets 500,000 views.

$0.04 \cdot 500{,}000 = 20000 = 5$ digits

$0.01 \cdot 500{,}000 = 5000 = 4$ digits

video id + like = 56 bytes
video id + dislike = 59 bytes
like count $\approx$ 5 bytes
dislike count $\approx$ 4 bytes

124 bytes $\times$ 1 billion = 124 GB of storage

Because you are able to save so much more bytes with relational, we will use that

Throughput

Earlier we estimated that on average we would see 1412 req/s. Assuming each req/res uses 8 bytes → 1412 · 8 = 11296 Bytes/sec
= 11.296 KB/sec

The max theoretical throughput is 2376 Mbit/s
= 297.886 KB/s

This means that theoretically one application server can handle the traffic

Let's assume there is 50% reads / 50% writes on each request ⇒ 706 reads/s
706 writes/s

Online it says a single mysql node handles about 200 writes/s and the theoretical ceiling is 1000 writes/s with SSDs. To make this easier on the load we can add a replica/batch up the write requests and add a queue to store the messages

Throughput Cont

To increase the read throughput, we can add a replication layer to the database for reads only especially since consistency is not a huge issue

Fault Tolerance

Since we are aiming for high availability we always want a hot standby ready in the case that the current one goes down, maybe even 2

Here we want a policy on DNS to swap the record entries in the event something goes down

Every 5 minutes, create a snapshot back up of the data in the currently used DB node in case something fails

Worst case lose $5 \cdot 60 \cdot 706 = 211800$ writes, but we use queing to get over that

Security

Since it is a public facing API and we don't have authentication, we need some form to prevent spamming

1. To prevent DDoS attacks, should put a WAL in front of the application server that times people out if they send more than 5 req/s
2. Have a cache that stores 10 GB worth of user cookies to their state
   Only happens on writes so this has 766 req/s which a single redis node can easily handle

   If a user recently wrote "like" a video, the state is stored in cache that the App servers check. If the user is in the "like" state, it prevents "liking" again. Same for "dislike"

Overall Design

Based on the previous pages, our final design looks like this: