YouTube

Can be applied to Netflix, Hulu and other video sharing platform

Step 1: Understand problem scope and establish design scope

qu: important features?
ans: ability to upload video and watch video.

qu: what client to support?
ans: mobile apps, web browser, and smart TV

qu: daily active user?
ans: 5 mn

qu: average daily time spent on product
ans: 30 mins

qu: support international users
ans; yes, large percentage are international user

qu; support idea resolution
ans: accepte more video resolution and format

qu; is encryption required
ans: yes

q:file size requirement
ans: max 1 GB

qu: leverage some cloud infra provided by amazon, google, Microsoft
ans: recommend some existing cloud service

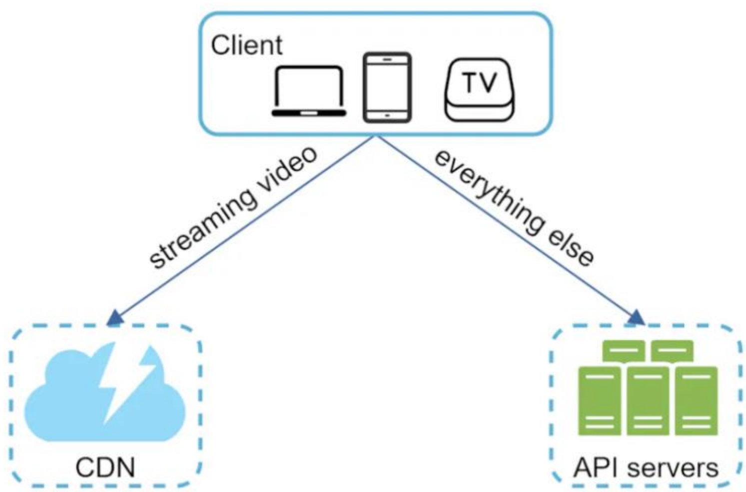
- upload ability videos fast
- smooth video streaming
- ability to change video quality
- low infra cost
- high availability, scalability, and reliability requirements
- client support: mobile apps, web browser and smart TV

Step 2: Back of the envelope estimation

assume product: 5 mn DAU
user watch 5 videos per day

10% user upload 1 video per day
average video size - 300 MB
total daily storage needed - 5 mn * 10% * 300 MB = 150 TB
CDN cost
5 mn * 5 videos * 0.3GB * 0.02 = 150,000 per day

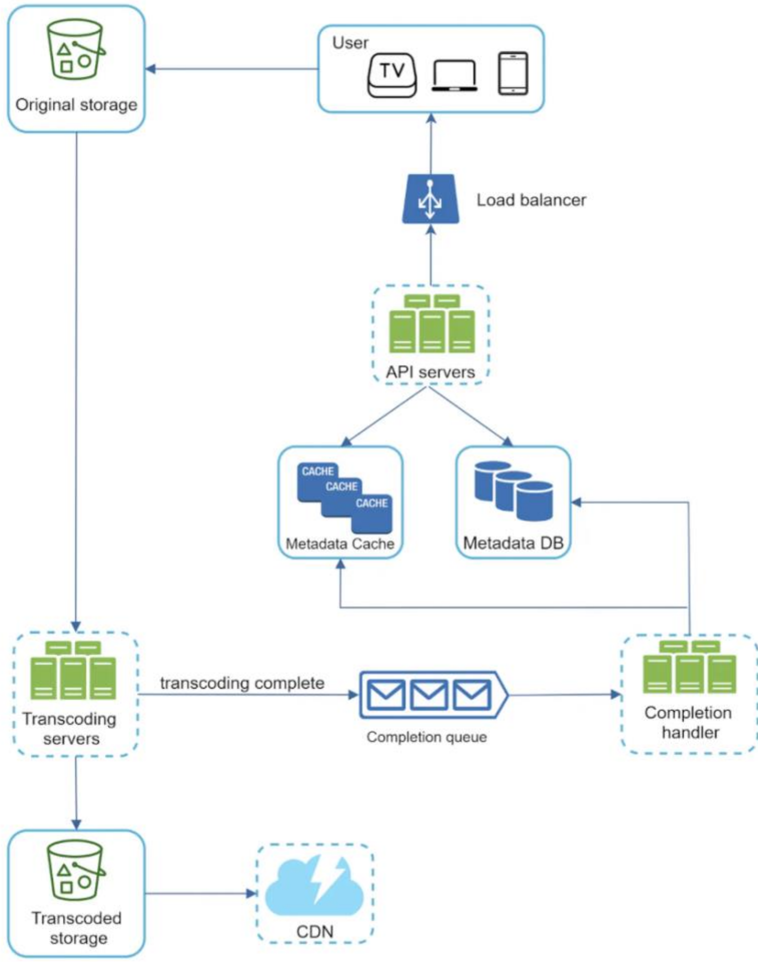
Step 3: propose HLD and get buy in



client - watch YouTube on your computer, mobile phone, smartTV
CDN - video stored in CDN. Play video - streamed from CDN
API server - everything except video streaming through API server. Include feed recommendation, generating video upload URL, update metadata database and cache, user signup etc.

have 2 flow -

video upload flow
video stream flow



following components

User: user watched YouTube on device eg: computer, mobile, tv

load balancer: evenly distribute request evenly among API server

API server: all request go through API server except video streaming

Metadata DB - video metadata stored in metadata DB. It is shared and replicated to meet performance and high availability requirements.

metadata cache - for better performance video metadata and user objects are cached

original storage - blob storage used to store original video: BLOB (binary large object): collection of binary data stored in single entity in db management system

transcoding server: also called video encoding. Convert video format to other formats - provide best video stream possible for different service and bandwidth capabilities.

transcoded storage - blob storage that store transcoded video files.

CDN: video cached in CDN. Play video button get video stream from CDN

completion queue:message queue that store info about video transcoding completion event

completion handler - contains list of worker that pull event data from completion queue and update metadata cache and database.

flow1: upload actual video

1. Video uploaded to original storage
2. Transcoding server fetch video from original storage and start transcoding
3. Once transcoding complete, following step execute in parallel -
    a. Transcoded video sent to transcode storage
    b. Transcoding completion event queued in completion queue.
        - completion handler contain bunch of worker that continuously pull event data from queue
        - once transcoding complete - completion handler update metadata db and cache

flow2: update metadata

while file upload to original storage - client in parallel send request to update video metadata (file name, size, format etc) in cache and db

video streaming flow

- when play video, immediately start streaming instea of wait until fully downloaded
- download means copy to device; stream means continuously receive video stream from remote source videos.
- **streaming protocol -** to control data transfer, different streaming protocol support different video encoding and playback.
    - MPEG-DASH: (moving Picture Experts Group), (Dynamic Adaptive Streaming over HTTP)
    - Apple HLS: HLS = HTTP live streaming
    - microsoft smooth streaming
    - Adobe HTTP dynamic streaming
- video stream from CDN directly, from closest server. So, need little latency.

<u>**Step 4: Design Deep Dive**</u>

entire system break into 2 parts - video uploading and video streaming.

- for video to be smooth, video must encoded into compatible bitrates and formats. Bitrates = rate at which bits are processed over time. **High bitrate -> high video quality.**

- raw video consume high storage.
- many device and browser support only certain video format
- for high quality video - deliver high resolution video to users who have high network bandwidth and low quality for low bandwidth.

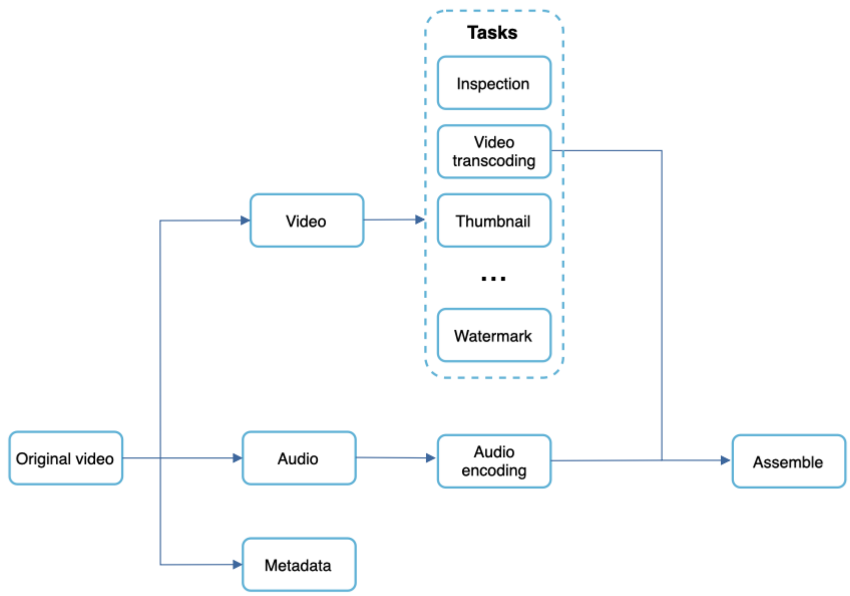
encoding format

- container - basket that contain video file, audio, metadata. Can tell by its file extension eg: .avi, .mov, .mp4

- codecs - compression and decompression algorithm - to reduce video size while preserving quality. Most used are H.264, VP9, and HEVC

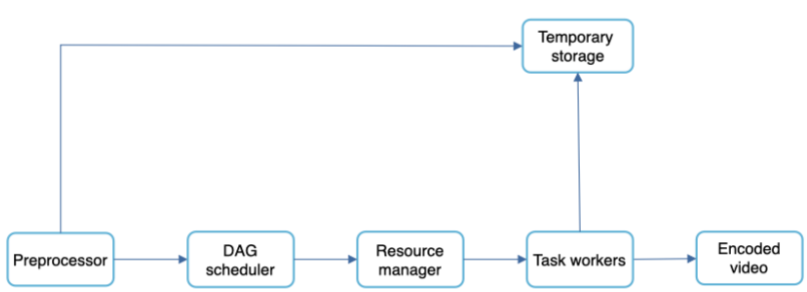
directed acyclic graph

- transcoding video is expensive and Tim-consuming, eg need watermarks or thumbnails etc.
- to maintain processing pipeline and high parallelism, have some abstraction and let client define task.
- eg use directed acyclic graph, with task execution in sequential and parallel.



original video split into - video, audio, and metadata, with task defined.
inspection = make sure video have good quality
video encoding - video convert into different resolution, codec, bitrate
thumbnail - video upload by user or autogenerate by system
watermark - image on video to identify your identity

video transcode architecture



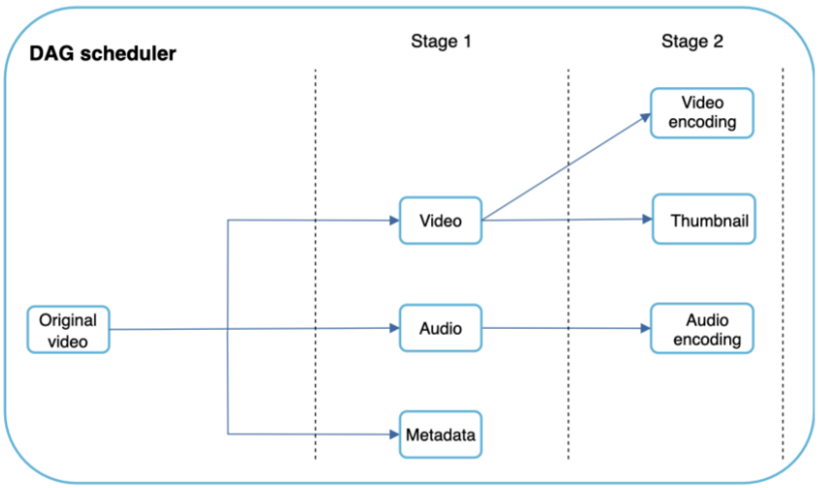
preprocessor -

split video into smaller group of pictures (GOP), arranged in specific order with independent payable unit, with few seconds length

- DAG generation based on configuration files
- cache segmented video in temporary storage for retry operation.

DAG scheduler

- split DAG graph into stage of task and put them in task queue in resource manager.



resource manager

- manage resource allocation
  - task queue - contain task to be executed
  - worker queue - priority queue that contain worker utilization info
  - running queue - contain info about currently running task and worker running the task

  - task scheduler get highest priority task from task queue, and get optimal task worker to run task from worker queue, and instruct to run chosen task.
  - task scheduler bind task/worker info and put it in running queue, remove job from running queue once job is dne.

task workers

- run task identified by DAG

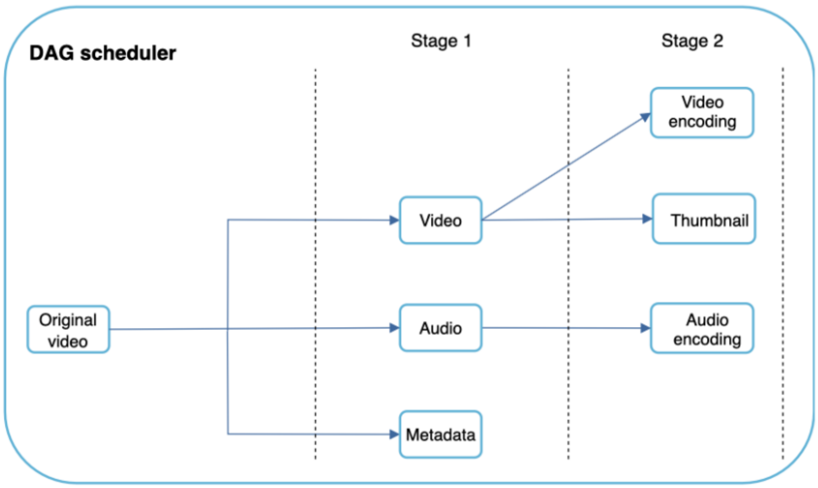
temporary storage

- caching to store metadata
- blob storage - for audio, video data
- data in temporary storage is freed once corresponding video processing complete.

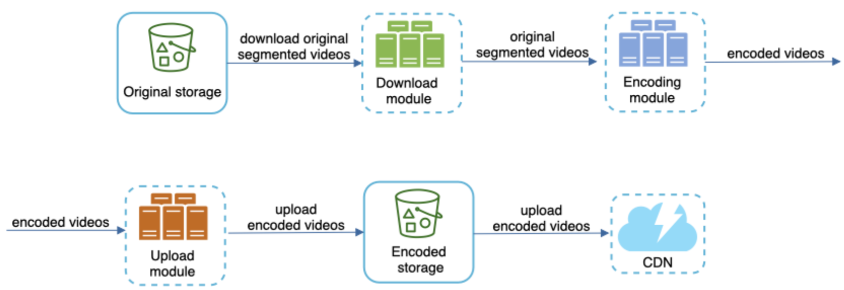
en**coded video -** final pipeline output

system optimization

1. **speed optimization - parallelize video uploading**
    - split video into smaller chunk by GOP

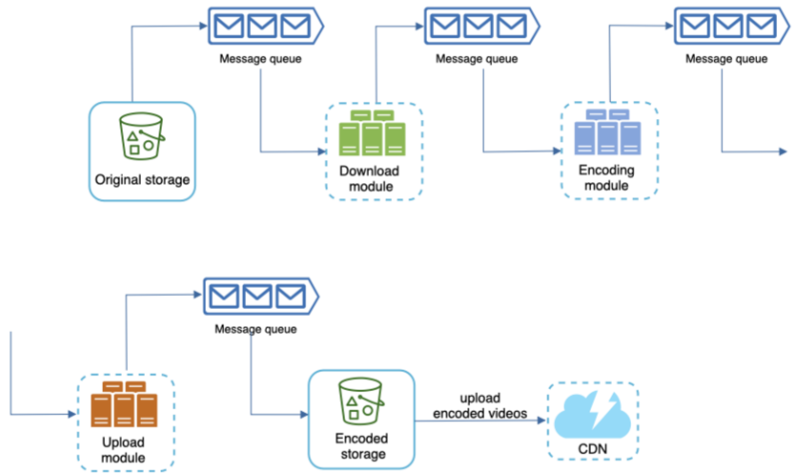


- Place upload center close to user - use CDN
- Parallelism everywhere

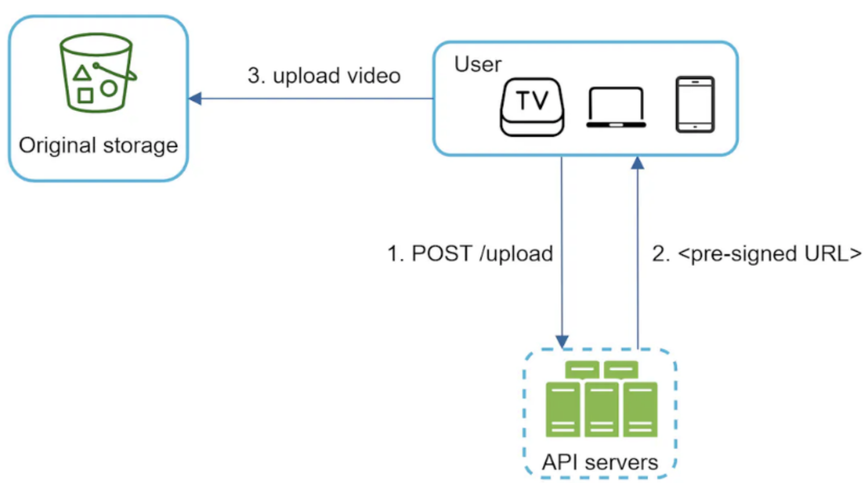


above flow depends on one another so make parallelism difficult. For parallelism, introduce message queue.

- now encoding module doesn't need to depend on download module, if there are events in message queue, encoding module starts executing those jobs in parallel



2. Safety optimization: pre-signed upload URL - ensure only authorized user upload video to right location



- client make HTTP request to API server to fetch resigned URL, which give access permission to object identified in the URL -> S# use this predesigned URL to upload files. API server respond with resigned URL -> client receive response, upload video using pre-signed URL

- protect your videos - protect copyright video using following
    - digital right management system
    - AES encryption: encrypt video and decrypt using playback to ensure only authorized user can watch encrypted video.
    = visual watermarking - company logo/name show on top of video.

3. Cost saving optimization

- since CDN is costly, only serve most popular video from CDN, keep other in high capacity storage video server.

- less popular content, no need to store many encoded version, can be stored on-demand.
- some videos are popular only in certain region, store them only in that region
- by your own CDN by partnering with ISP (eg AT&T, Comcast, Verizon etc)

Error handling

- recoverable error - eg video segment fail to transcode - retry operation, if not recoverable, send error code to client
- non recoverable - eg malformed video format, stop running task associated to it and return proper error
- typical error include
    - upload error - retry few times
    - split error video - if old version can't split video by GOP, video split send to server side, which will than split video
    - transcode error: retry
    - preprocessor error - regenerate DAG diagram
    - DAG scheduler error - reschedule task
    - resource manager queue down - use replica
    - task worker down - retry task on new worker
    - API server down - are stateless, so request directed to different API
    - metadata cache server down - data replicated multiple times so can access other nodes
    - metadata DB server down
    - master down - promote one of the slaves as master
    - slave down - use another salve for read and bring other fb to replace dead one.

<u>Wrap up</u>

- scale API tier
- scale the database
- live streaming
- live streaming has high latency requirement, so need different streaming protocol
- live streaming has low req for parallelism because small chunk are already processed in real time
- live streaming require different set of error handling
- video takedown - eg pornography, copyrights or other illegal rights shall be removed - this can be discover during upload process, or through user flagging