OTT → Over the top

Eg → Netflix, Hotstar, Amazon Prime, etc.

High level Design

Steps to design any system

1) Define MVP → Minimum Viable Product

Whatsapp → sending message, ✓
receiving message, ✓
get a notification of message. ✓

calling a friend ✗
story on whatsapp ✗

2) Estimation of Scale →

1) Store data → small data can fit in 1 machine.
→ large data cannot fit in 1 machine.

Sharding
↳ Divide data s.t we can store in multiple machines and most frequent usecase can be solved by going to only one machine.

Uber (Driver's data) → Book a cab is most frequent query.
↓
location ✓

2) Read Heavy → Quora, Instagram, etc.
Write Heavy → WhatsApp, Email, etc.

3) QPS → Queries per second.

Instagram → High QPS
Car Purchase → Not very high QPS

3) Design Goals →

Bank Application → Highly Consistent. (100% correct data)
Instagram → Highly Available → CAP Theorem

Google Search Typeahead → Extremely fast (low latency)

4) Define API → How external world is going to use the system.
Design challenges → Actual design ✓

System Design of Netflix (OTT Platform)

1) Define MVP →
- Playing a movie ✓
- Search for a movie ✓
- Jump to a time in a movie ✓

- Recommend a movie ✗
- Pause/Play ✓
- Paid platform →
  - User details ✓
  - Storing card details ✗

2) Estimation of Scale →

4) Store Data →
- User Data
- Movie Data (metadata) → title, duration, cost, rating etc.

Movie file, Trailer file, Thumbnail → Blob Store

Google Sheet → we store link of google drive & not complete movie file.

↓ Actual DB

↓ Blob Store

S3 by AWS

Haystack by FB

Azure Blobs by Microsoft etc.

movie file → Blob Store

Blob Store → file path → use it to play movie.

Data

- User Data
  - Movie Data
    - name, email, contact number, password, userid, plan, etc. } 1KB for 1 user
- Large Video files, Images → S3

200 M users → 1KB → Total data = 200 M * 1 KB

$\simeq 200 * 10^6 * 10^3 B$

$= 200 * 10^9 B \simeq 200 GB$ ✓

Movie Data (metadata) → title, duration, cost, rating etc. } 1KB per movie

10K movies → 1KB → $10K * 1KB = 10 * 10^3 * 10^3 B$

$= 10 * 10^6 B = 10 MB$

Total storage → $200 GB + 10 MB \simeq 200 GB$

Store in 1 Machine

(User data + Movie metadata)

SQL DB

2) Read Heavy ✓
3) QPS ✓

---

3) Design Goals → consistency vs Availability ✓

→ latency → No buffering is important (very fast loading) ✓

4) APIs → 1) user signup / sign in.
2) payment gateway.
✓ 3) play a video (movie id)
✓ 4) search for a movie (search text)
5) pause & play (movieid, time stamp)
6) list of movies on homepage → simple → show latest movies

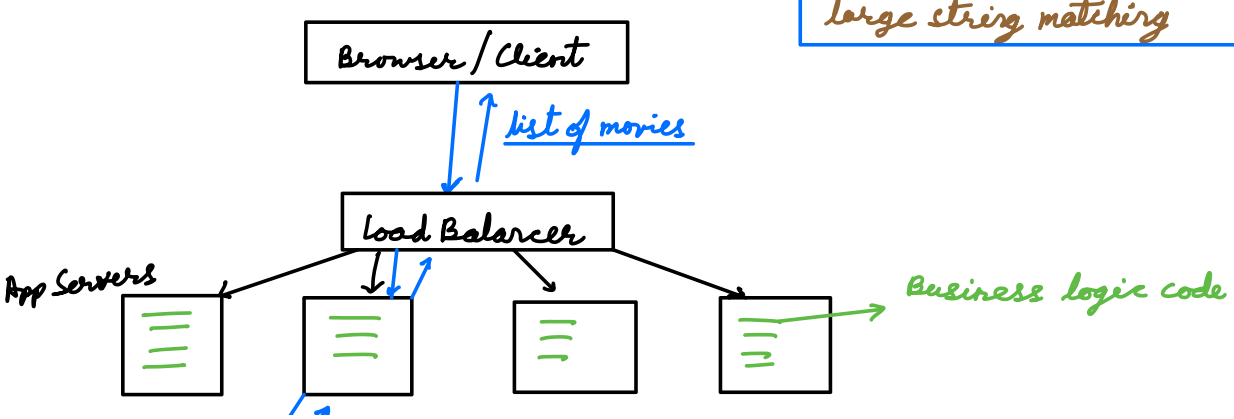
---

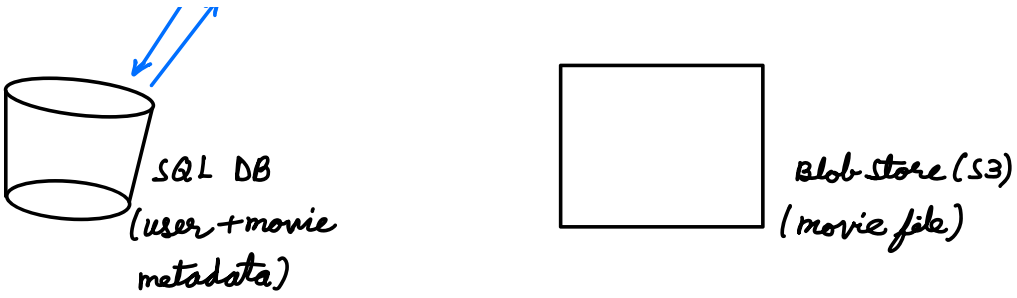
1) Search for a movie

comic
- just a string
- list of movies
- length < 100 char ∴ use any string matching Algo.

```
findMovieResult (text) {
  r1 = findMatchingMovieTitle (text)
  r2 = findMatchingGenre (text)
  r3 = findMatchingCast (text)
  return union (r1, r2, r3)
}
```

Elastic Search → Email App, Chat App etc.
large string matching



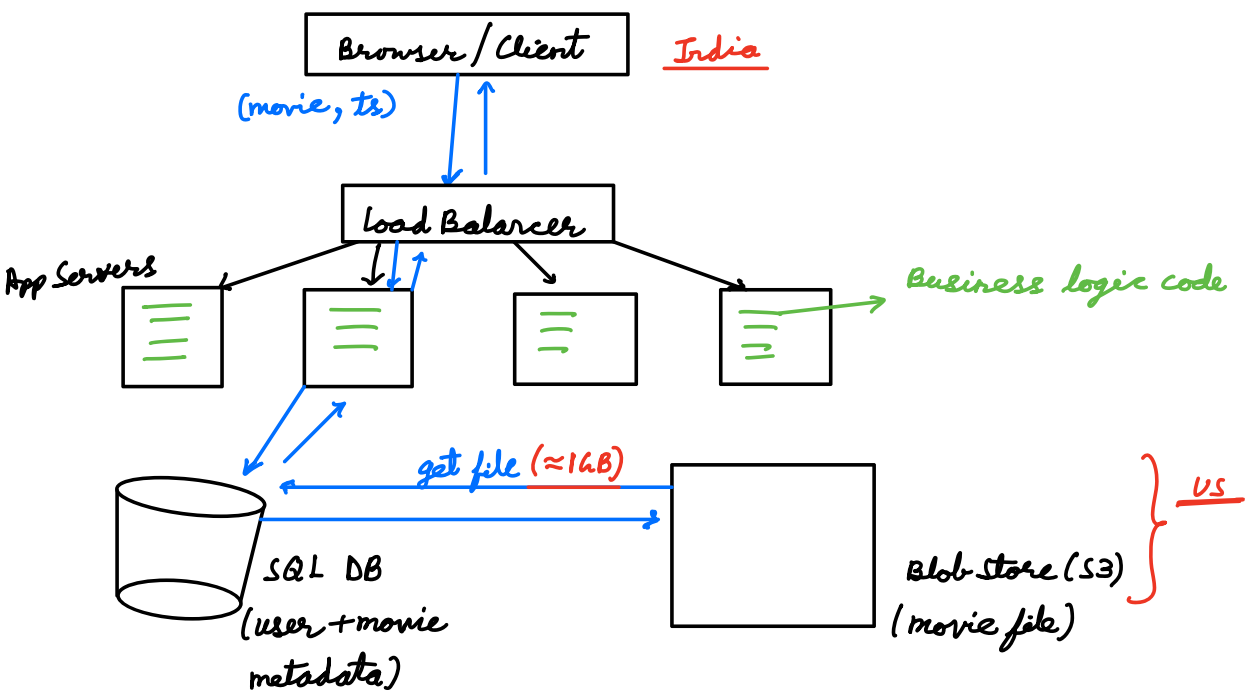


Tables

1) userTable → name, id, contact no., password etc.
2) movieTable → id, title, duration, rating, etc.
3) movieCost → movieId, costId.
4) cost → id, name, age, awards etc.
5) movieGenre → movieId, genre.
6) movie_filepath → movieId, filepath.
   - 720p
   - 360p
   - 480p

---

3) Play a video file → (No buffering)

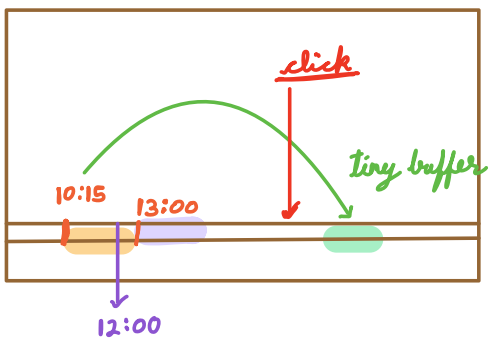


CDN → Content Delivery Network Eg → Akamai, Cloudflare, CloudFront (by Amazon).

file → CDN → distribute it throughout the world.
file url ✓

India ——✗—— US
Mumbai ✓

Still getting 1GB file is heavy task. ✓



Divide movies in multiple chunks & load movie chunk by chunk.

movie, file path → resolution check ✓

↳ trigger to load next chunk.
(a min before current chunk completes)

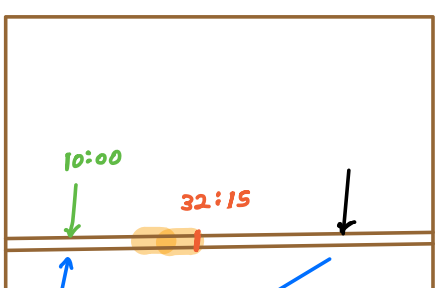
movieFileChunk → movieId ✓, resolution ✓, from_ts, to_ts, file_path ✓.

1GB → 3 Hours
3-4 min → 20 MB ✓

from < requested ts < to

11:00 — 14:00 ✓
12:20

if current chunk is about to end, load next chunk.



Jump to random ts → loading (15-20 MB chunk)
✓ Normally watching → no loading.

manually handle for some cases.