Design a live video broadcasting system (like ESPN)

Requirements

Services

- Users management (IAM): registration, login, preferences …
- Subscription management : subscribe, renew, billing systems
- Choice of channel to subscribe to
- Selection of a channel to watch
- Available on computers, smartphones, smartTVs, tablets,...
- Current program available from the beginning

Additional services:

- different subscription packages (nb of channels, quality of video, nb of concurrent devices)
- Replay service

Latency

Minimum latency depends on what event is broadcasted. Let's assume a **target latency of 1 minute**

- For news events like public speach, 1min delay is acceptable
- For sport events (match), 1min delay may already be too long.

Estimations

US market is 300 millions people -> 100 million households.
Target 20% of subscriptions -> 20 million subscriptions.

On peak hours / events, all customers use service -> 20 millions concurrent users on peak times.
Rest of the day, assuming 10% of usage -> 2 millions concurrent users.
-> High scalability, high elasticity.

Bandwidth:
1 min of video == 50 MB (incl. multiple formats and qualities) -> 50MB/min/user -> 800KB/sec/user
- Normal times: 800KB * 2M = 1.6TB/sec total bandwidth
- Peak times: 800KB * 20M = 16TB/sec total bandwitdh

Storage (on disk), per channel:
for 1 day of replays, we need 50MB * 60 * 24 = 70 GB.
With 70% capacity model -> 100GB storage / day * 3 (for replication) = **300GB/day**
+ programs thumbnails. 10KB * 24 programs/day = 240 KB -> negligeable.

Cache (in memory):
Target is 2h of cache for instant availability of the current program.
2h of video = 50MB * 120 = **6GB**
+ cache for most watched videos of last 24 hours (rule of 80-20): 100GB * 0.2 = **20 GB**
Total cache: 26 GB

Storage (databases):

Users DB:
20 million subscribers:
- Email: 32 bytes, Name: 20 bytes, Password 20 bytes, subscription_level: 1 byte, billing_details: 50 bytes, address: 100 bytes, id: 4 bytes, registration_date: 4 bytes (epoch)
-> 256 bytes /user * 20 millions : **5GB database**

Programs DB (for 1 day):
While programs are available for 24h only, we may want to keep historical data for 5 years.

24 programs /day:
id: 4 bytes, title: 20 bytes, description: 512 bytes, tags (4/program): 80 bytes, time: 4 bytes, duration: 4bytes, video_link: 32 bytes, thumbnail_link: 32 bytes
-> 1024 bytes/program * 24 * 365 * 5 = **40MB database**

Users historical activities:
- id: 4 bytes, user_ref: 32 bytes, program_ref: 32 bytes, time: 32 bytes (epoch), rating: 1 bytes
-> 101 bytes / activity.
2 program/day/user * 20M * 5 years = **7TB database**

Interfaces

Users management:

- register (session_id, username, password, email): create a new user
- login (session_id, username, email): to authenticate to the service
- CRUD operations: create, read, update, delete operations (e.g. preferences, address,...).

Subscriptions:

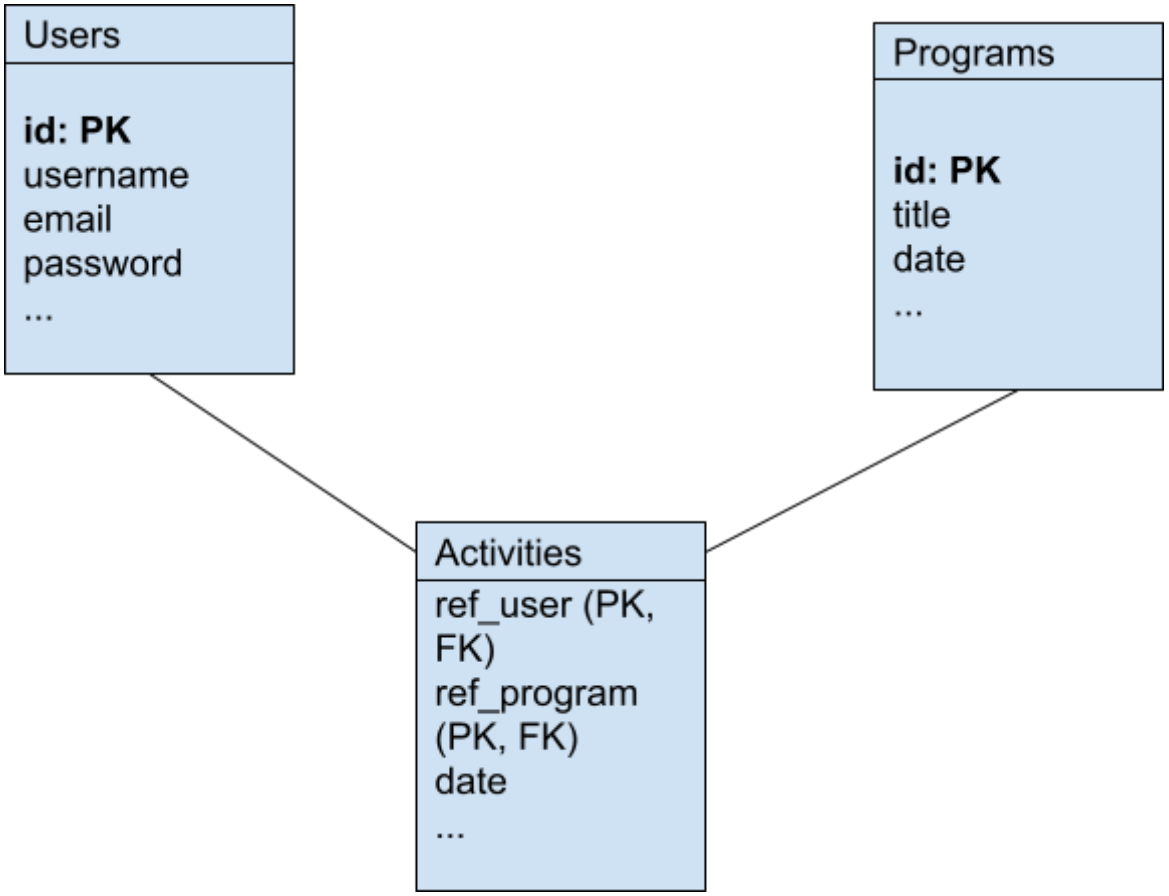
- update_CB (session_id, CB_number, CVV, card_holder, expiration_date): update credit card details
- Subscribe (session_id, subscription_level): update subscription level.

Watch:

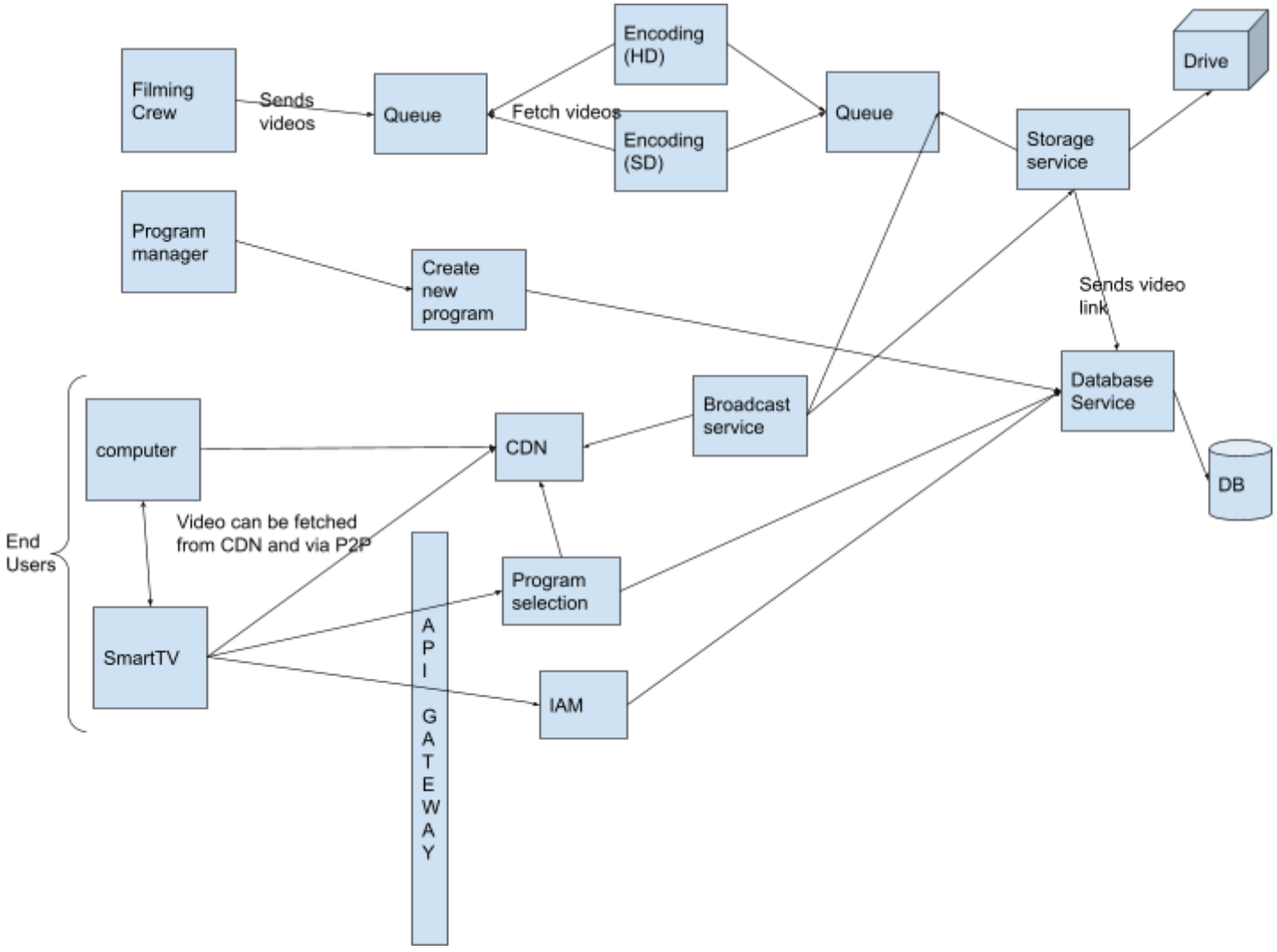
- Watch_channel (session_id, channel_id, program_id, start_time): select what program to watch now.

Data models

CAP theorem:only two out of Availability, consistency and partition tolerence.
- Near real time service
- billing system.
- Scale of database is acceptable (< 8TB). For growth, database can also be partitioned using DB Sharding with consistent hashing on primary keys.
- Fairly stable schema
-> **RDBMS** (e.g. MySQL) which offers Availaibility and Consistency.



High level design



Explanations:

- Filming crew who is on the site of the event sends videos to a queue.
- To address latency, the queue is in memory, with high availability cluster (e.g. REDIS)
- Multiple encoding services fetch the videos to encode them in different formats/quality. Then they publish the encoded videos to a new queue, also in cache (REDIS).
- The encoded videos queue is consumed by two services in parallel:
    - Storage service to write the video on disk and store data in the DB,
    - Broadcast service that immediately dispatch to CDN.
    - This way, The potential lower performances on disk writting is not in the critical path of broadcasting near real time.

- When end users want to watch a new program, they contact the Program selection service that redirect them to the nearest CDN node. The CDN can also be part of a P2P network with local users, so the system can easily scale during peak times: traffic goes directly between users and only API calls go to our systems.