Case Study - System Design of Hotstar

Frequently Asked Interview Questions

Q1: Given that different clients have different internet speeds, how do you ensure a good experience for all?

Answer: If clients have different speeds, and they all had to download the same heavy video, then obviously the slower clients will have no option but to keep buffering till they have some part of the video to show. In general, the rate at which video plays is always going to be higher than the rate at which those bits can be downloaded and hence, the client for slow internet will lag.

So, what do we do? Can we play with the resolution of the video? A smaller resolution video would be low on quality, but would require much less number of bits to be downloaded to show the same scenes. That way, even the slower connections can show video without lag.

You would however want to give the best resolution possible at the speed of the internet available. For example, if you know 1080p cannot be loaded at the current speed, but 720p can be loaded, then you would not want to degrade the experience by loading 240p. Note that the internet speed can be changing as well. So, you'd need to do detection continuously and adapt to the best resolution possible at the current internet speed available.

Most OTTs do exactly the above by using **Adaptive bitrate streaming (ABS)**. ABS works by detecting the user's device and internet connection speed and adjusting the video quality and bit rate accordingly.

The Adaptive Bitrate Streaming (ABS) technology of Hotstar detects Internet speed by sending data packets to the user's device and measuring the time it takes to respond. For example, if the response time is low, the internet connection speed is good; hence, Hotstar servers can stream a higher video quality. On the other hand, if the response time is high, it indicates a slow internet connection; hence, Hotstar servers can stream a lower video quality. This way, Hotstar ensures that all its viewers have an uninterrupted streaming experience, regardless of device or internet speed.

But is the internet speed alone enough? For example, if I am on a good internet connection but on mobile, does it make sense to stream in 1080p resolution? At such a small screen size, I might not even be able to decipher the difference between 480p, 720p and 1080p resolution. And if so, streaming 1 080p is bad for my data packs :P

Hotstar can detect a user's client device to provide an optimal streaming experience. Hotstar uses various methods, such as user-agent detection and device fingerprinting.

User-agent detection involves analyzing the string sent by a user's browser when they visit a website. This string contains information about the browser version and operating system, which Hotstar can use to identify the device type.

Device fingerprinting works by analyzing specific parameters of a user's device, such as screen size, plugins installed, and time zone settings. Then Hotstar uses this data to create a unique "fingerprint" for each user's device to identify their device type.

Using these two methods, Hotstar is able to accurately identify the user's device and provide an optimal streaming experience. This ensures that users are able to enjoy uninterrupted viewing, no matter which device they are using.

Q2: For non-live/recorded content, what optimizations would you do on the client to ensure smoother loading?

Observation #1: It does not make sense to load the entire file on the client. If I decide to view 5 movies, watch their first 5 mins and end up closing them, I would end up spending a lot of time and data downloading 5 entire movies.

So, we do what Torrent does. We break the file into small chunks (remember HDFS?). This way, a client can only request for a specific chunk (imagine 10:00 - 11:00 min chunk).

Observation #2: But how does the client know which chunk to request? Or if you notice, as you try to scroll to the future/past timestamp on the video, it shows a preview. How would it do that? When you first click on a video, the metadata of these chunks (timestamp boundaries, chunk_id, a low res thumbnail per chunk) can be brought to the client (and the client caches it). This could enable the jumping from one timestamp to another.

Observation #3: Ok, but what happens when I am done with the current chunk? If I would have to wait for the next chunk to be downloaded, then it would lead to a buffering screen at the end of every chunk. That’s not a good experience, is it? How do we solve that?
What if we pre-load the next chunk (or next multiple chunks) as we near the end of the current chunk. For example, if the current chunk is 60 seconds, then we might start to preload the next chunk when I am at 30 seconds / 40 seconds. It happens in the background, and hence you’d never see buffering.

Obviously, I cannot cache too many chunks as it takes up my RAM space. So, I keep evicting from cache chunks I have seen in the past (or just simple LRU on chunks).

Note that the chunk downloaded is of the right bitrate, depending on the adaptive bit rate we talked about above. So, it is possible I download chunk 1 of very high quality, but if my internet speed has degraded, the next downloaded chunk is of lower quality.

Chunks make it easier to upload files (easier to retry on failure, easier to parallelise uploads) and make it easier to download (2 chunks can be downloaded in parallel).

And obviously, it's better to fetch these chunks from CDN for the right resolution of the file instead of directly from S3/HDFS.

More to think about here(Homework): Is there a better way of creating chunks other than just timestamps? For example, think of the initial cast introduction scenes which most people skip. If most people skip that, how would you break that down into chunks for most optimal data bandwidth utilization? Also, what future chunks to load then? Or could you create chunks by scenes or shot selection?

Observation #4: If you notice, if Netflix has 10,000 TV shows, most people are watching the most popular ones at a time. There is a long tail that does not get watched often. For most popular shows, can we do optimisations to ensure their load time is better?
What if we did server side caching of their metadata, so that it can be returned much faster. In general, LRU caching for movies/TV show metadata does the job.

Summary: Hotstar uses various technologies and techniques to ensure that viewers can access high-quality video streams quickly, reliably, and without interruption. To further optimize the user experience for non-live content, Hotstar could employ the following optimizations:

- **Chunking:** Dividing video content into small sections or "chunks" allows for improved streaming and delivery of video files.
- **Pre-emptive loading** of the future chunks.
- **Browser Caching** of relevant metadata and chunks to enable a smoother viewing experience.
- **Content Delivery Network (CDN):** A CDN can help distribute the load of delivering video files to users by caching content closer to the users. It can significantly reduce the distance data needs to travel, thereby reducing load times. The browser can download resources faster as they are served from a server closer to the user.
- **Server-side Caching:** By caching frequently-accessed video files and metadata, the system can reduce the number of times it has to retrieve data from a slower storage system like disk or network storage.
- **Encoding optimization:** Using video encoding, Hotstar reduces the size of the video files without affecting the perceived quality.
- **Adaptive Bitrate Streaming:** This technique allows the video player to adjust the video quality based on the user's network conditions.
- **Minimizing HTTP requests:** By reducing the number of resources that need to be loaded, the browser can load the page faster. It can be done by consolidating files, using CSS sprites, and lazy loading resources.

Q3: In live streaming, given a bunch of clients could be lagging by a few seconds/minutes, how do you handle those?

Answer:

Now that recorded videos are done, let's think about how this should work for a live streaming use case.

For the recorded case, we had the entire file with us upfront. Hence, we could create chunks, transform to different resolutions as an offline task at our convenience. Publish chunk metadata. And then make the video live once all of it was ready.

For live use cases, we are getting video on the fly. While a delay of a minute is fine, you wouldn't want the delay to be higher than that.

Problem statement: How does a client tell CDN/Hotstar where does it need to get the next sequence of bytes from?

Imagine if we do create chunks. If chunks are large (1-2 mins), then that means, our clients will have large lag. They will have to wait for the 2 min chunk to be created (which is only after events in those 2 mins have occurred) and then that chunk goes to the client via CDN.
Imagine we create 15-30 second chunks. This is also called stream segmentation.

Steps involved:

- From the source, the video gets to the Hotstar backend using RTMP.
- For every 15-30 second chunk, immediately, jobs are scheduled to transform these chunks to different resolutions. Metadata for these chunks is updated in primary DB (In memory cache and MySQL?).
- Via another job, these chunks are uploaded to CDNs.

On the client side,

- You request for chunk metadata (Backend can control - till how long back I send the metadata for, and what you send in metadata)
- Based on metadata, you request for the most recent chunk from CDN.
- You keep asking for incremental metadata from the backend, and keep going to CDN for the next chunk. *[Note that in the case of recorded videos, you need not have asked for incremental metadata again and again].*

The segmentation in the above case helps with:

- Smoother handling of network lag, flaky internet connection.
- Being able to support lagging clients (not every client needs to be at the same timestamp).

- Being able to show history of chunks in the past, so you can scroll to earlier chunks if you want to watch a replay.

Do note that there are clients like Scaler class streaming / Zoom / Meet where delay has to be less than 2-3 seconds. However, you don't need to support lagging clients there. You cannot scroll to a history of the meeting. You might not even be recording the meeting.
The number of clients would be limited. That is a very different architecture.

Homework: Think about how Google Meet streaming would work if you were building it.

Size of a segment: It's essential to balance the number of segments you create and the size of each segment. Too many segments will mean a lot of metadata, while too few segments will make every segment larger and hence increase the delay.

In addition, Hotstar also utilized **AI** and **data mining techniques** to identify trends in the streaming behavior of its users to improve the user experience. For example, they scale the number of machines up and down based on certain events (since autoscaling can be slow). Dhoni coming to bat increases the number of concurrent viewers, so ML systems can detect that pattern and scale to a larger number of machines beforehand.

Do note that **CDN** delivers the chunks of video fast as it's at the heart of the design.

To sum it up, Hotstar's system design uses techniques like chunking, encoding, dynamic buffer management, CDN, ABS, and AI-powered platforms; it ensures users have an enjoyable streaming experience regardless of their device or internet connection speed.

Q4: How do you scale from 100 clients to 50 million clients streaming simultaneously?

Let’s look at what are the common queries that scale as the number of users concurrently live streaming increases.

In a live stream, as discussed above, 3 things happen:
1. **Upload:** Get the video from the source, encode and transform it to different resolutions and update metadata of chunks. This is independent of the number of users watching.
2. **Metadata fetch:** Fetch updated metadata of chunks to be loaded, so that clients can keep requesting the right stream of data from the CDN, from the right checkpoint.
3. **Streaming the actual video:** from the CDN.

2 and 3 scales with the number of concurrent users. [Note that we have assumed a MVP here, so no chat/messaging feature assumed].
For 2, you’d need to scale the number of appservers and number of caching machines which store these metadata. However, the maximum amount of load is generated due to 3. As it is a massive amount of data being sent across to all the concurrent users.

CDN infrastructure plays a major role here. Akamai (Hotstar's CDN provider) has done a lot of heavy lifting to let Hotstar scale to the number of concurrent users that they have. A CDN stores copies of web assets (videos, images, etc.) on servers worldwide so that users can quickly access them no matter where. As the number of users of Hotstar increases, the CDN will have to scale accordingly.

If most of the users are expected to be from India, then for edge servers (CDN machines closer to the user) that are close to India region, more capacity is added there (more number of machines). Clients do an ANYCAST to connect to the nearest available edge server.

Additional Resources

- Scaling hotstar.com for 25 million concurrent viewers
- Building a scalable data platform at Hotstar