**YouTube**

1. Solidify the requirements – both Functional and Non-Functional

* Functional Requirements:
  + **Stream Videos**
  + **Upload Videos**
  + **View Videos**
  + **View Thumbnails**
  + **Search Videos according to titles**
  + **Add Comments to Videos**
  + **Like & Dislike Videos**
* Non-Functional Requirements:
  + **Highly Availability**: By Incorporating Fault tolerance (failover mechanisms for load balancers) and Repetition of System Bottlenecks.
  + **Good Performance**: A smooth streaming experience leads to better performance overall.
  + **Scalability**: By Incorporating Horizontal Scaling.
  + **Reliability**: Content uploaded to the system should not be lost or damaged.
  + **Consistency can take a hit:**Consider an example where a creator uploads a video. Not all users subscribed to the creator’s channel should immediately get the notification for uploaded content.

1. Scope the Problem

* What kind of clients? Mobile Apps, Web Browsers, Smart TVs?

1. Capacity/ Resource Estimation
2. Traffic Estimates
3. DAU – 500M
4. Total Users – 1.5B
5. Uploads/ min – 500 hours of video/ min
6. View/ Upload ratio – 300 : 1
7. Average Video length – 5 mins
8. Size of an average (5 min) video before processing/ encoding (compression, format changes and so on) – 600MB
9. Size of an average video after encoding (using different algorithms for different resolutions like MPEG-4 and VP9) – 30MB
10. Daily User handling limit of a server (assume that a typical server handles 8000 requests per second) – 691200000
    * Estimated storage requirement/min – (500 x 60 x 30MB) / 5 = 180GB/min
    * Views/ min –
    * Total Servers Required – 500M/ 8000 = 62500 servers (Assuming that each 500M users only gives 1 request per day)
11. Storage Estimates
12. Time Duration for which objects are required to be stored – 1 year

* Total Storage Required for a year = 365 x 24 x 60 x 180GB/min = 94.608PB

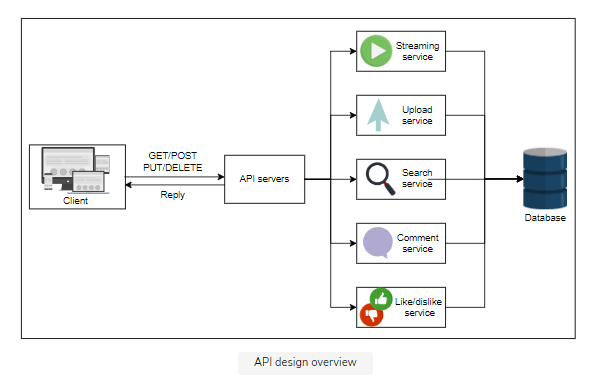
1. Bandwidth Estimates

* Assume uploaded data bandwidth per minute – 50 MB/min
* Incoming Data (uploads) – (500 x 60 x 50MB/min) x (8/60) = 200Gbps
* Assume viewed data bandwidth per min – 10MB
* Outgoing Data (views) – (200/5) x 300 = 12Tbps

1. Memory (Cache) Estimates
2. 80/20 Rule? Yes

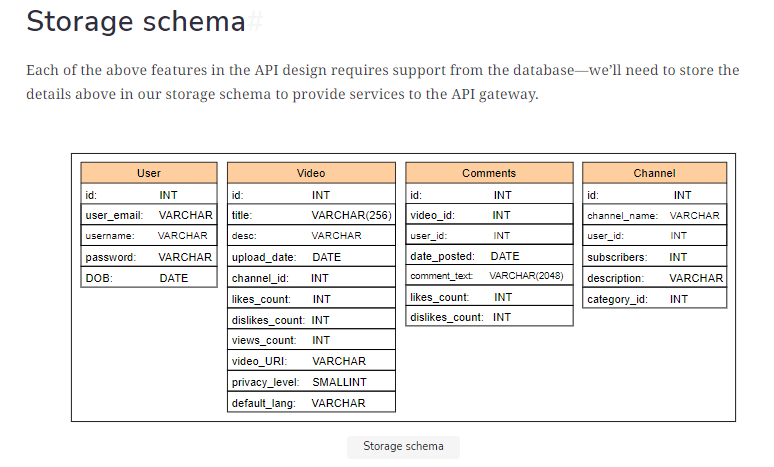
* Cache Size =

1. System API’s

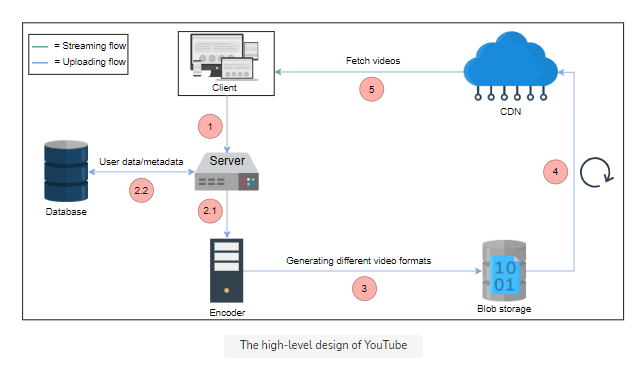


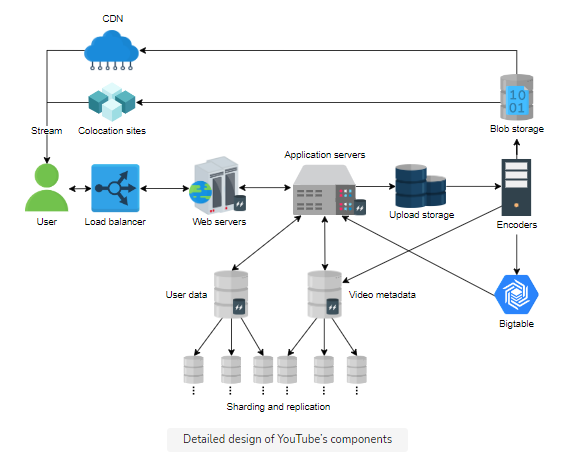
* uploadVideo(user\_id, video\_file, category\_id, title, description, tags, default\_language, privacy\_settings)
* streamVideo(user\_id, video\_id, screen\_resolution, user\_bitrate, device\_chipset)
  + The server can best optimize the video if the user's screen resolution is known.
  + The transmission capacity of the user is required to understand which quality of video chunks should be transferred to the client or user.
  + Many YouTube users watch content on handheld devices, which makes it important to know the handling capabilities of these devices to better serve the users.
  + **The server will store different qualities of the same video in its storage and serve users based on their transmission rate.**
* searchVideo(user\_id, search\_string, length, quality, upload\_date)
* viewThumbnails(user\_id, video\_id)
* likeDislike(user\_id, video\_id, like)
* commentVideo(user\_id, video\_id, comment\_text)

1. Database Design



1. Present the building blocks of the Design
2. Databases: to store the metadata of videos, thumbnails, comments, and user-related information.
3. Load Balancers: to distribute millions of incoming clients request among the pool of available servers.
4. CDN: used to effectively deliver content to end users, reducing delay and burden on end-servers.
5. Blob Storage: for storing all videos on the platform.
6. Servers: basic requirement to run application logic and entertain user requests.
   1. Web servers: Web servers take in user requests and respond to them. These can be considered the interface to our API servers that entertain user requests.
   2. Application server: The application and business logic reside in application servers. They prepare the data needed by the web servers to handle the end users’ queries.
7. User and metadata storage: Since we have a large number of users and videos, the storage required to hold the metadata of videos and the content related to users must be stored in different storage clusters. This is because a large amount of not-so-related data should be decoupled for scalability purposes.
8. BigTable: For each video, we’ll require multiple thumbnails. Bigtable is a good choice for storing thumbnails because of its high throughput and scalability for **storing key-value data. Bigtable is optimal for storing a large number of data items each below 10 MB.** Therefore, it is the ideal choice for YouTube’s thumbnails.
9. Encoders: Encoders and transcoders compress videos and transform them into different formats and qualities to support varying numbers of devices according to their screen resolution and bandwidth.
10. Upload storage: The upload storage is temporary storage that can store user-uploaded videos.
11. CDN and colocation sites: CDNs and colocation sites store popular and moderately popular content that is closer to the user for easy access. Colocation centers are used where it’s not possible to invest in a data center facility due to business reasons.
12. Propose a Design Diagram and get an agreement



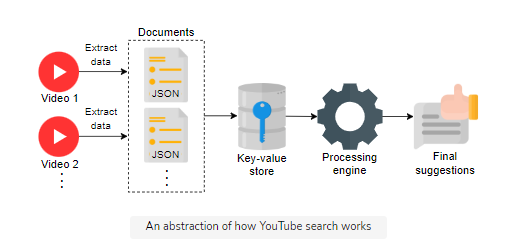


8. Design Workflow

1. The user can upload a video by connecting to the web servers. The web server can run Apache or Lighttpd. Lighttpd is preferable because it can serve static pages and videos due to its fast speed.
2. Requests from the web servers are passed onto application servers that can contact various data stores to read or write user, videos, or videos’ metadata.

* Why the need of separate Application and Web server?
  + we want to decouple clients’ services from the application and business logic.
  + different programming languages can be used on this layer to perform different tasks efficiently. For example, the C programming language can be used for encryption.
  + gives us an additional layer of caching, where the most requested objects are stored on the application server while the most frequently requested pages will be stored on the web servers.

1. Multiple storage units are used. Let’s go through each of these:
   * Upload storage is used to store user-uploaded videos before they are temporarily encoded.
   * User account data is stored in a separate database, whereas videos metadata is stored separately. The idea is to separate the more frequently and less frequently accessed storage clusters from each other for optimal access time. We can use MySQL if there are a limited number of concurrent reads and writes. However, as the number of users—and therefore the number of concurrent reads and writes—grows, we can move towards NoSQL types of data management systems.
   * Since Bigtable is based on Google File System (GFS), it is designed to store a large number of small files with low retrieval latency. It is a reasonable choice for storing thumbnails.
2. The encoders generate thumbnails and also store additional metadata related to videos in the metadata database. It will also provide popular and moderately popular content to CDNs and colocation servers, respectively.
3. Because YouTube is storage intensive, sharding different storage services will effectively come into play as we scale and do frequent writes on the database. At the same time, Bigtable has multiple cache hierarchies. If we combine that with GFS, web- and application-level caching will further reduce the request processing latency.
4. Specific Design Components:
   * + **YouTube Search**
       - Each new video uploaded to YouTube will be processed for data extraction. We can use a JSON file to store extracted data, which includes the following: Title of the Video, Channel name, Description of the video, The content of the video, possibly extracted from the transcripts, Video length, Categories.
       - Each of the JSON files can be referred to as a document. Next, keywords will be extracted from the documents and stored in a key-value store.
       - The key in the key-value store will hold all the keywords searched by the users.
       - Value in the key-value store will contain the occurrence of each key, its frequency, and the location of the occurrence in the different documents.
       - When a user searches for a keyword, the videos with the most relevant keywords will be returned.

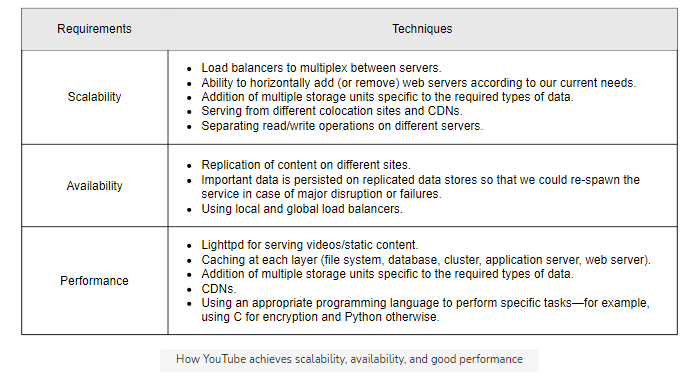


* The approach above is simplistic, and the relevance of keywords is not the only factor affecting search in YouTube.
* In reality, a number of other factors will matter. The processing engine will improve the search results by filtering and ranking videos. It will make use of other factors like view count, the watch time of videos, and the context, along with the history of the user, to improve search results.

1. Design Evaluation:
2. Availability
   * **Replication of Servers**
   * **Replication of Data Centers**
   * **Global Servers Load Balancing (GSLB)**: Our design uses global server load balancing (GSLB) to handle our system traffic. It ensures intelligent request distribution among different global servers, especially in the case of on-site failures.
   * **Local Load Balancing:** Local load balancers can exclude any dead servers
3. Scalability

* **Horizontal database sharding**: Our design is scalable because our data can easily be distributed among horizontally sharded databases. We can employ a consistent hashing scheme to balance the load between the application and database layers.
  + **Choice of NoSQL database**

1. Low Latency/Smooth streaming
   * **Distributed Cache** - Geographically distributed cache servers at the ISP level to keep the most viewed content.
   * **Cache at local servers** - Using caching at various layers via a distributed cache management system.
   * **CDNs** - A CDN deploys its services in close vicinity to the end users for low-latency services.
   * **Choice of Appropriate Storage** - Choosing appropriate storage systems for different types of data. For example, we’ll can use Bigtable for thumbnails, blob storage for videos, and so on.
2. Reliability
   * **Data partitioning**: Through data partitioning, the non-availability of one type of data will not affect others.
   * **Fault Tolerance**: We can use redundant hardware and software components for fault tolerance.
   * **Heartbeat Protocol**: We can use the heartbeat protocol to monitor the health of servers and omit servers that are faulty and erroneous.
   * **Consistent Hashing:** We can use a variant of consistent hashing to add or remove servers seamlessly and reduce the burden on specific servers in case of non-uniform load.



11. Additional Details:

1. Load Balancers:
   * We could use a simple Round Robin approach that distributes incoming requests equally among backend servers.
   * A problem with Round Robin LB is that we do not consider the server load. As a result, if a server is overloaded or slow, the LB will not stop sending new requests to that server. To handle this, a more intelligent LB solution can be placed that periodically queries the backend server about its load and adjusts traffic based on that.
2. Caching:

* **Which Cache**: We can use any off-the-shelf solution like Memcached, which can store full URLs with their respective hashes.
* **How much cache memory should we have?** We can start with 20% of daily traffic and, based on clients’ usage patterns, we can adjust how many cache servers we need.
* **Which cache eviction policy would best fit our needs?** When the cache is full, and we want to replace a link with a newer/hotter URL, how would we choose? Least Recently Used (LRU) can be a reasonable policy for our system.

12. Trade – Offs

1. Consistency
   * Our solution prefers high availability and low latency. However, strong consistency can take a hit because of high availability (see the [CAP theorem](https://www.educative.io/edpresso/what-is-the-cap-theorem)).
   * Nonetheless, for a system like YouTube, we can afford to let go of strong consistency. This is because we don’t need to show a consistent feed to all the users.
   * For example, different users subscribed to the same channel may not see a newly uploaded video at the same time.
   * **It’s important to mention that we’ll maintain strong consistency of user data. This is another reason why we’ve decoupled user data from video metadata.**
2. Distributed Cache
   * We prefer a distributed cache over a centralized cache in our YouTube design. This is because the factors of scalability, availability, and fault-tolerance, which are needed to run YouTube, require a cache that is not a single point of failure. This is why we use a distributed cache.
   * Since YouTube mostly serves static content (thumbnails and videos), Memcached is a good choice because it is open source and uses the popular Least Recently Used (LRU) algorithm.
3. Bigtable versus MySQL
   * Why did we choose MySQL and Bigtable?
     1. The primary reason for the choice is performance and flexibility. The number of users in YouTube may not scale as much as the number of videos and thumbnails do. Moreover, we require storing the user and metadata in structured form for convenient searching. Therefore, MySQL is a suitable choice for such cases.
     2. However, the number of videos uploaded and the thumbnails for each video would be very large in number. Scalability needs would force us to use a custom or NoSQL type of design for that storage. One could use alternatives to GFS and Bigtable, such as HDFS and Cassandra.
4. Public versus private CDN

* CDNs can be private or public.
* The choice is more of a cost issue than a design issue. However, for areas where there is little traffic, YouTube can use the public CDN because of the following reasons:
  + Setting up a private CDN will require a lot of CAPEX.
  + For rather little viral traffic in certain regions, there will not be enough time to set up a new CDN.
  + There may not be enough users to sustain the business.
* However, YouTube can consider building its own CDN if the number of users is too high, since public CDNs can prove to be expensive if the traffic is high. Private CDNs can also be optimized for internal usage to better serve customers.

1. Duplicate videos
   * Duplication can be solved with simple techniques like locality-sensitive hashing. However, there can be complex techniques like Block Matching Algorithms (BMAs) and phase correlation to find duplications. Implementing this solution can be quite complex in a huge database of videos. We may have to use technologies like artificial intelligence (AI).