The document titled "System Design: YouTube - Grokking Modern System Design Interview for Engineers & Managers" from Educative.io provides a detailed guide on designing the YouTube service as part of a system design interview preparation course. The course is structured into several key lessons and sections, each focusing on different aspects of system design.

### **Introduction**

The document starts by introducing the course and its goals, emphasizing the importance of understanding abstractions, non-functional system characteristics, and back-of-the-envelope calculations. It also highlights the foundational building blocks of system design, such as the Domain Name System (DNS), load balancers, databases, and various distributed systems components.

### **Key Sections and Topics**

1. **Abstractions and Non-functional System Characteristics**: These sections cover the essential concepts and criteria needed to design scalable and efficient systems.
2. **Back-of-the-envelope Calculations**: This section is aimed at helping engineers make quick, approximate calculations to estimate system requirements and capacities.
3. **Building Blocks**:
   * **Domain Name System (DNS)**
   * **Load Balancers**
   * **Databases**
   * **Key-value Store**
   * **Content Delivery Network (CDN)**
   * **Sequencer**
   * **Distributed Monitoring**
   * **Monitoring Server-side and Client-side Errors**
   * **Distributed Cache**
   * **Distributed Messaging Queue**
   * **Pub-sub System**
   * **Rate Limiter**
   * **Blob Store**
   * **Distributed Search**
   * **Distributed Logging**
   * **Distributed Task Scheduler**
   * **Sharded Counters**

### **Designing YouTube**

The main focus of the document is the design of YouTube, broken down into several detailed lessons:

1. **Requirements**: Identifying both functional and non-functional requirements of the YouTube system. This includes understanding the volume of video content uploaded and the necessary features for user interaction.
2. **Design**: Detailed explanation of how to design the YouTube service, including API design, database schemas, and the search functionality. This lesson covers the core components and their interactions to ensure a smooth user experience.
3. **Evaluation**: Analysis of how the proposed design meets the identified requirements, scalability considerations, and potential future challenges.
4. **Reality is More Complicated**: Discussion on the complexities involved in delivering video content efficiently, handling network congestion, and other real-world issues that YouTube faces.
5. **Quiz**: A quiz to reinforce the major concepts learned, focusing on designing similar systems like Netflix.

### **Additional System Design Examples**

The document also includes brief sections on designing other popular services, providing a broader perspective on system design:

* **Google Maps**
* **Proximity Service / Yelp**
* **Uber**
* **Twitter**
* **Newsfeed System**
* **Instagram**
* **URL Shortening Service / TinyURL**
* **Web Crawler**
* **WhatsApp**
* **Typeahead Suggestion**
* **Collaborative Document Editing Service / Google Docs**

### **Concluding Remarks**

The course concludes with a discussion on spectacular failures in system design, providing insights into common pitfalls and how to avoid them.

Overall, the document serves as a comprehensive guide for engineers and managers preparing for system design interviews, offering practical insights and detailed examples of designing large-scale systems like YouTube.

The document "Requirements of YouTube's Design - Grokking Modern System Design Interview for Engineers & Managers" outlines the essential requirements and estimations needed for designing YouTube as part of a system design interview preparation course. Here’s a detailed summary:

### **Overview**

The document is part of a course that aims to help engineers and managers prepare for system design interviews by providing a comprehensive understanding of the requirements for designing a complex system like YouTube.

### **Sections Covered**

1. **Functional Requirements**
2. **Non-functional Requirements**
3. **Resource Estimation**
4. **Storage Estimation**
5. **Bandwidth Estimation**
6. **Number of Servers Estimation**
7. **Building Blocks**

### **Functional Requirements**

The system must be capable of:

* Streaming videos.
* Uploading videos.
* Searching videos by titles.
* Liking and disliking videos.
* Adding comments to videos.
* Viewing thumbnails.

### **Non-functional Requirements**

* **High Availability:** The system should have an uptime of 99% or higher.
* **Scalability:** The system must handle increasing user demands without performance degradation.
* **Good Performance:** Ensuring a smooth streaming experience.
* **Reliability:** Uploaded content should not be lost or corrupted.

### **Resource Estimation**

Given the scale of YouTube, resource estimation involves:

* **Storage Resources:** Required for storing uploaded and processed content.
* **Processing Resources:** Web/application servers to handle concurrent user requests.
* **Bandwidth:** Required for both uploading and downloading video content.

### **Detailed Estimations**

#### **Storage Estimation**

* Assumptions:
  + Total number of users: 1.5 billion.
  + Active daily users: 500 million.
  + Average video length: 5 minutes.
  + Size before processing: 600 MB.
  + Size after encoding: 30 MB.
* Formula:
  + **Storage Requirement** = Total upload/min × Storage/min.
  + For example, if 500 hours of video are uploaded per minute, the storage needed is calculated as: Storage per minute=500 hours×60 minutes/hour×6 MB/minute=180,000 MB or 180 GBStorage per minute=500 hours×60 minutes/hour×6 MB/minute=180,000 MB or 180 GB

#### **Bandwidth Estimation**

* Assumptions:
  + Upload  
    ratio is 1:300.
  + Average video requires 10 MB bandwidth per minute for streaming.
* Formula:
  + **Bandwidth Requirement** = Total content/min × Transmission size/min.
  + For example, if 500 hours of video are uploaded per minute: Bandwidth per minute=500 hours×60 minutes/hour×50 MB/minute=1,500,000 MB or 1,500 GB/minuteBandwidth per minute=500 hours×60 minutes/hour×50 MB/minute=1,500,000 MB or 1,500 GB/minute

#### **Number of Servers Estimation**

* Assumption:
  + Each server handles 8,000 requests per second.
  + Total active users: 500 million.
* Calculation:
  + **Number of servers required**: Total active usersQueries handled per server=500,000,0008,000=62,500 serversQueries handled per serverTotal active users​=8,000500,000,000​=62,500 servers

### **Building Blocks**

Key components required for the high-level design of YouTube:

* **Databases:** Store metadata of videos, thumbnails, comments, and user information.
* **Blob Storage:** Store video files.
* **Content Delivery Network (CDN):** Efficiently deliver content to users, reducing latency.
* **Load Balancers:** Distribute incoming client requests among available servers.
* **Servers:** Run application logic and handle user requests.
* **Encoders and Transcoders:** Compress and transform videos into various formats and qualities.

### **Conclusion**

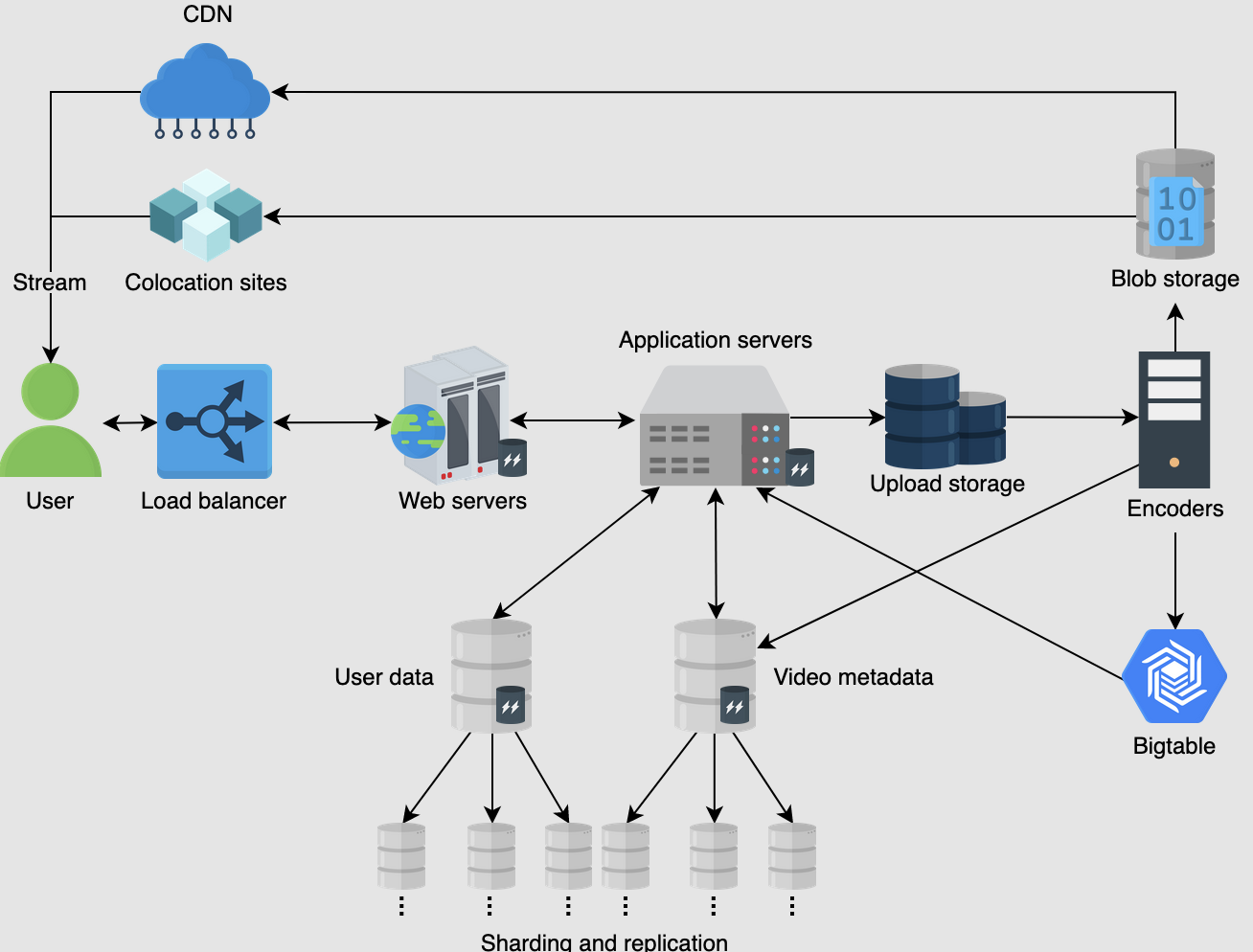
The document provides a thorough analysis of the requirements and estimations needed for designing a scalable, reliable, and efficient system like YouTube. It emphasizes understanding both functional and non-functional requirements, performing resource estimations, and identifying key building blocks essential for the system’s architecture.

The document "Design of YouTube - Grokking Modern System Design Interview for Engineers & Managers" provides a comprehensive guide to designing YouTube's architecture. It details the high-level and detailed design considerations, API design, storage schema, and search functionalities required for building a scalable and efficient video-sharing platform.

### **Key Sections Covered**

1. **High-level Design**
2. **API Design**
3. **Storage Schema**
4. **Detailed Design Components**
5. **Design Flow and Technology Usage**
6. **YouTube Search**

### **High-level Design**



The high-level design outlines how various components interact to meet YouTube's functional and non-functional requirements. The design includes:

1. **Video Upload Workflow**: Users upload videos to the server, which stores metadata in a database and hands over the video to the encoder for compression and transformation into multiple resolutions. Videos are stored in blob storage and popular videos may be forwarded to the CDN for low-latency streaming.
2. **Streaming Workflow**: The CDN provides low-latency streaming for end users, supported by a combination of load balancers, web servers, application servers, and storage components.

### **API Design**

APIs are designed to handle core YouTube functionalities:

1. **Upload Video**:
   * Endpoint: POST /uploadVideo
   * Parameters: user\_id, video\_file, category\_id, title, description, tags, default\_language, privacy\_settings.
2. **Stream Video**:
   * Endpoint: GET /streamVideo
   * Parameters: user\_id, video\_id, screen\_resolution, user\_bitrate, device\_chipset.
3. **Search Videos**:
   * Endpoint: GET /searchVideo
   * Parameters: user\_id, search\_string, length, quality, upload\_date.
4. **View Thumbnails**:
   * Endpoint: GET /viewThumbnails
   * Parameters: user\_id, video\_id.
5. **Like/Dislike Video**:
   * Endpoint: GET /likeDislike
   * Parameters: user\_id, video\_id, like.
6. **Comment Video**:
   * Endpoint: GET /commentVideo
   * Parameters: user\_id, video\_id, comment\_text.

### **Storage Schema**

The storage schema supports the API design by storing necessary details:

1. **Video Metadata**: Information about videos including title, description, tags, and more.
2. **User Data**: Information related to user accounts and activities.
3. **Thumbnails**: Thumbnails generated for each video are stored in Bigtable for efficient retrieval.

### **Detailed Design Components**

1. **Load Balancers**: Distribute user requests among web servers.
2. **Web Servers**: Handle user requests and interface with API servers.
3. **Application Servers**: Execute business logic and prepare data for web servers.
4. **User and Metadata Storage**: Separate storage clusters for user data and video metadata for optimal access time.
5. **Bigtable**: Stores thumbnails efficiently due to its high throughput and scalability.
6. **Upload Storage**: Temporary storage for user-uploaded videos.
7. **Encoders**: Compress and transcode videos into various formats and generate thumbnails.
8. **CDN and Colocation Sites**: Store and serve popular and moderately popular content close to users.

### **Design Flow and Technology Usage**

1. **Video Upload**: Users connect to web servers to upload videos, which are temporarily stored and then encoded.
2. **Data Separation**: User data, video metadata, and thumbnails are stored in separate databases for optimal performance.
3. **Encoders and CDNs**: Encoders generate thumbnails and compress videos, which are then stored in CDNs and colocation sites for efficient streaming.
4. **Streaming**: Users stream videos from the nearest CDN or colocation site.

### **YouTube Search**

1. **Data Extraction**: New videos are processed to extract data such as title, channel name, description, and content.
2. **Keyword Storage**: Extracted keywords are stored in a key-value store, where the key holds the keywords and the value contains their occurrences and locations.
3. **Search Algorithm**: When users search for keywords, the system returns videos with the most relevant keywords, enhanced by factors such as view count, watch time, and user history for better search results.

### **Conclusion**

The document provides an in-depth analysis of YouTube's design, covering high-level and detailed design aspects, API functionalities, and search mechanisms. The comprehensive approach ensures scalability, reliability, and efficient performance, making it a valuable resource for engineers and managers preparing for system design interviews.

The document "Evaluation of YouTube's Design - Grokking Modern System Design Interview for Engineers & Managers" provides a comprehensive evaluation of YouTube's design, focusing on how the proposed solution fulfills various requirements, the trade-offs involved, and considerations for future scaling. Here’s a detailed summary:

### **Key Sections Covered**

1. **Fulfilling Requirements**
2. **Trade-offs**
3. **Consistency**
4. **Distributed Cache**
5. **Bigtable vs. MySQL**
6. **Public vs. Private CDN**
7. **Duplicate Videos**
8. **Future Scaling**
9. **Vitess System for Scalability**
10. **Web Server Considerations**

### **Fulfilling Requirements**

The design aims to fulfill the primary requirements of smooth streaming (low latency), scalability, availability, and reliability.

1. **Low Latency/Smooth Streaming**:
   * Geographically distributed cache servers at the ISP level to store the most viewed content.
   * Appropriate storage systems for different data types (e.g., Bigtable for thumbnails, blob storage for videos).
   * Caching at various layers via a distributed cache management system.
   * Utilization of Content Delivery Networks (CDNs) to serve videos with low latency.
2. **Scalability**:
   * Horizontal scalability of web and application servers.
   * Potential restructuring of MySQL storage for better scalability.
   * Use of sharding and other techniques to handle increased traffic.
3. **Availability**:
   * Data replication across multiple servers and data centers to avoid single points of failure.
   * Local and global load balancers to manage traffic and handle server or data center failures.
4. **Reliability**:
   * Data partitioning and fault-tolerance techniques to ensure system reliability.
   * Use of redundant hardware and software components.
   * Monitoring server health and using consistent hashing to manage server loads.

### **Trade-offs**

* **Consistency**: The design prioritizes high availability and low latency over strong consistency, as per the CAP theorem. Strong consistency is maintained only for user data, while video metadata can afford eventual consistency.

### **Distributed Cache**

A distributed cache is preferred over a centralized one to ensure scalability, availability, and fault-tolerance. Memcached is chosen due to its open-source nature and use of the Least Recently Used (LRU) algorithm, suitable for YouTube’s access patterns.

### **Bigtable vs. MySQL**

* **Bigtable**: Chosen for its scalability and performance in storing large numbers of videos and thumbnails.
* **MySQL**: Used for structured data like user information and metadata for efficient searching. The design acknowledges potential scaling limitations with MySQL.

### **Public vs. Private CDN**

The decision between using public or private CDNs depends on cost considerations and traffic volume:

* **Public CDN**: Suitable for regions with low traffic due to lower CAPEX requirements.
* **Private CDN**: Considered for high-traffic regions to optimize performance and cost.

### **Duplicate Videos**

The design does not initially address duplicate videos, which can waste storage and raise copyright issues. Techniques like locality-sensitive hashing, Block Matching Algorithms (BMAs), and AI can be employed to handle duplication.

### **Future Scaling**

The document discusses limitations and potential design changes needed to handle increased traffic:

* **Infrastructure Scaling**: Scaling web servers, application servers, and data stores, along with implementing load balancers and distributed caches.
* **Sharded Databases**: Addressing potential choke points in MySQL by using sharding and client modifications.
* **Vitess System**: A solution developed by YouTube to manage database scalability while maintaining ACID properties. It abstracts database layers, giving the illusion of a single database to the client, thus simplifying scaling and maintaining performance.

### **Vitess System for Scalability**

Vitess allows for database partitioning and scalability while using MySQL for structured data management. It handles database-client complexity and provides the performance of a NoSQL system.

### **Web Server Considerations**

Custom web servers may be developed for large-scale services like YouTube, as commercial or open-source solutions might not meet specific needs. Google's ESF is an example of a custom web server used for YouTube.

### **Conclusion**

The document provides a thorough evaluation of YouTube's design, addressing key requirements, trade-offs, and future scaling considerations. The proposed design focuses on achieving low latency, high availability, scalability, and reliability while managing the complexities of a large-scale video-sharing platform.

The document titled "The Reality Is More Complicated - Grokking Modern System Design Interview for Engineers & Managers" delves into the complexities and optimizations involved in effectively delivering content to end users on YouTube. It outlines the necessary steps and techniques to maintain high quality of experience (QoE) while managing storage and network demands. Here’s a detailed summary:

### **Key Sections Covered**

1. **Introduction**
2. **Encode**
3. **Deploy**
4. **Recommendations**
5. **Deliver**
6. **Adaptive Streaming**
7. **Potential Follow-up Questions**

### **Introduction**

The document begins by emphasizing the importance of optimizing storage and network usage while ensuring a good QoE for end users. It introduces three crucial steps for providing effective service: Encode, Deploy, and Deliver.

### **Encode**

Encoding is critical for reducing the size of raw video files and optimizing them for different end devices. Key points include:

* **Multiple Encoding Schemes**: Raw videos are encoded using various schemes suitable for different devices, resulting in multiple encoded files per video.
* **Per-segment Encoding**: Videos are divided into smaller segments, each encoded individually based on content detail. This reduces storage requirements and saves bandwidth during deployment and streaming.
* **Granular Encoding**: Dynamic segments with more detail are encoded differently than less dynamic segments, optimizing quality and storage.

### **Deploy**

Content deployment involves bringing content closer to users to reduce latency and network burden. Strategies include:

* **Geographic Distribution**: Popular video chunks are stored in CDNs, ISPs' points of presence (PoPs), and internet exchange points (IXPs) to facilitate quick access.
* **Flash and Storage Servers**: Flash servers hold popular content for low-latency delivery, while storage servers hold less popular content.
* **Off-peak Transfer**: Content transfer to ISPs occurs during off-peak hours to avoid network congestion.

### **Recommendations**

YouTube recommends videos to users based on their profile, considering factors like interests, viewing history, and interactions. The recommendation process involves:

* **Candidate Generation**: Filtering millions of videos down to hundreds based on user history and context.
* **Ranking**: Further filtering and ranking videos based on features and user interests using machine learning algorithms.

### **Deliver**

Content delivery involves redirecting users to the nearest available video chunks. Popular content is served from CDNs, while non-popular content is served from colocation sites or data centers. The process ensures reduced latency and efficient content delivery.

### **Adaptive Streaming**

Adaptive streaming adjusts video quality based on user bandwidth and device capabilities. Key parameters include:

* **Available Bandwidth**: End-to-end bandwidth from CDN/servers to the client.
* **Device Capabilities**: User device specifications.
* **Encoding Techniques**: Different encoding schemes used.
* **Buffer Space**: Available buffer space at the client.

### **Potential Follow-up Questions**

The document addresses several follow-up questions related to YouTube's design, including:

1. **Quantifying Availability**: Discussing fault tolerance and system resilience to ensure high availability.
2. **Designing for Average Behavior**: Ensuring horizontal scalability to handle varying user profiles.
3. **Resource Estimation**: Explaining the omission of resource estimation for comments and likes.
4. **Managing Load Spikes**: Leveraging public cloud elasticity for unexpected load spikes.
5. **Global Network Deployment**: Using Google's network for connecting data centers and CDN sites.
6. **Audio/Video Encoding Details**: Leaving out detailed encoding algorithms for further exploration.
7. **Specialized Hardware**: Discussing the use of specialized hardware for specific computations.
8. **Client-side Compression**: Considering fast, lossless compression at the client-side during content upload.
9. **Additional Benefits of File Chunks**: Highlighting benefits beyond adaptive bitrates, such as parallel preprocessing for live streams.

### **Conclusion**

The document provides a thorough exploration of the complexities in designing YouTube, emphasizing the importance of encoding, deployment, and delivery techniques to ensure efficient content distribution and high QoE for end users. It also addresses potential challenges and follow-up questions, offering insights into the practical aspects of large-scale system design.

ChatGPT can make mistakes. Check important info.

ChatGPT