Experiment 4

Student Name: Mayank Madan **UID:** 23BCS10894
Branch: CSE **Section/Group:** KRG 3_B
Semester: 6th **Date of Performance:**04/02/2026
Subject Name: System Design **Subject Code:** 23CSH-314

1. Aim

To design and analyze a scalable OTT (Over-The-Top) video streaming platform similar to Netflix/AmazonPrime that allows users to register, subscribe, search, and stream video content efficiently while ensuring high availability, low latency, and large-scale scalability using distributed system concepts**.**

2. Theory

An OTT (Over-The-Top) platform delivers video content over the internet without requiring traditional cable or satellite services. Modern OTT platforms are designed using microservices architecture and distributed systems to handle millions of concurrent users.

Such systems prioritize:

- High availability
- Low latency
- Scalability
- Fault tolerance

Video streaming platforms use:

- HLS (HTTP Live Streaming)
- DASH (Dynamic Adaptive Streaming over HTTP)

These protocols support Adaptive Bitrate Streaming (ABR), where video quality changes automatically according to user bandwidth.

In distributed systems, the CAP Theorem states that a system can provide only two of the following three guarantees:

- Consistency
- Availability
- Partition Tolerance

In OTT systems, Availability is prioritized over Consistency, because video playback should not stop even if minor metadata delays occur**.**

Tools Used

1. **Draw.io –** For architecture diagram
2. **MySQL –** For user and subscription data
3. **MongoDB (NoSQL) –** For video metadata storage
4. **ElasticSearch –** For fast search functionality
5. **Apache Kafka –** For asynchronous event streaming
6. **Redis / CDN Cache –** For caching
7. **Blob Storage (Amazon S3 equivalent) –** For storing video files

4. System Requirements

A. Functional Requirements

1. User should be able to create an account.
2. User should be able to log in securely.
3. User should be able to subscribe to plans.
4. User should be able to search movies/TV shows.
5. User should be able to stream videos in multiple resolutions (480p, 720p, 1080p, 4K).
6. System should support adaptive bitrate streaming.
7. User should be able to view thumbnails and metadata.
8. (Optional) System may provide recommendations.

B. Non-Functional Requirements

1. Scalability

- 200–300 million active users
- Around 20,000 videos
- Support for millions of concurrent streams
- Horizontal scaling of services

2. Availability vs Consistency

- Availability is more important than consistency
- Video playback must not stop
- Slight metadata delay is acceptable
- Payment service requires strong consistency

Latency

- Target latency: 50–80 ms
- Minimal buffering
- CDN reduces delay

5. High Level Design (HLD)

The system follows a microservices architecture**.**

Components:

1. API Gateway
    - Handles authentication
    - Routes requests
    - Rate limiting
2. User Service
    - Registration and login
    - JWT authentication
    - Uses MySQL
3. Subscription & Payment Service
    - Manages plans
    - Ensures transactional consistency
4. Search Service
    - Uses ElasticSearch
5. Video Metadata Service
    - Stores title, description, thumbnails (NoSQL DB)
6. Video Processing Pipeline
    - Videos stored in Blob Storage
    - Chunker splits videos into small segments
    - Encoder creates multiple resolutions
    - Kafka manages asynchronous processing
7. Streaming Service
    - Generates HLS (.m3u8) or DASH (.mpd) files
    - Client fetches chunks dynamically
8. CDN
    - Caches video chunks
    - Reduces server load
    - Improves performance

Video Streaming Workflow

1. User clicks on Play.
2. Client requests manifest file (.m3u8 / .mpd).
3. Manifest contains different bitrate URLs.
4. Client checks network speed.
5. Suitable resolution chunks are selected.
6. CDN serves cached chunks.
7. Video plays smoothly using adaptive bitrate.

7. Scalability Strategy

- Horizontal scaling of microservices
- Load balancing at API Gateway
- Kafka decouples processing
- CDN reduces origin traffic
- Stateless services enable easy scaling
- Blob storage handles large files efficiently

8. Result

A scalable and highly available OTT streaming system architecture was successfully designed and analyzed. The system ensures high availability, low latency, adaptive streaming, and supports millions of concurrent users using distributed system principles.

9. Learning Outcomes

1. Learned real-world OTT architecture design.
2. Understood functional vs non-functional requirements.
3. Gained knowledge of HLS/DASH and adaptive streaming.
4. Understood CAP theorem trade-offs.
5. Learned the role of Kafka, CDN, and encoding pipeline.
6. Improved understanding of scalable distributed systems.