14장 - 유튜브 설계

- 유튜브 시스템 설계 (= 넷플릭스, 훌루 같은 비디오 플랫폼 설계하는 문제에도 적용 가능)
- 유튜브에 대한 통계자료
  - MAU : 2십 억(2billion)
  - 매일 재생되는 비디오 수 : 5십 억(5billion)
  - 미국 성인 가운데 73% 유튜브 이용
  - 5천만(50million) 명의 창작자
  - 유튜브의 광고 수입은 2019년 기준 150억(15.1billion) 달러, 2018년 대비 36% 증가
  - 모바일 인터넷 트래픽 가운데 37%를 유튜브가 점유
  - 80개 언어로 이용 가능

1단계 : 문제 이해 및 설계 범위 확정

- 유튜의 전체 기능은 너무 많으므로 적절한 질문을 통해 설계 범위를 줄여야 함
  - `댓글`, `공유`, `좋아요`, `재생목록`, `구독`, 등. 너무 많음 이를 45 ~ 60분 안에 설계하기는 불가능

> 지원자: 어떤 기능이 가장 중요한가요?
>
> 면접관: 비디오를 올리는 기능과 시청하는 기능입니다.
>
> 지원자: 어떤 클라이언트를 지원해야 하나요?
>
> 면접관: 모바일 앱, 웹 브라우저, 스마트 TV 입니다.
>
> 지원자: 일간 능동 사용자 수는 몇 명입니까?
>
> 면접관: 5백만(5million)입니다.

지원자: 사용자가 이 제품에 평균적으로 소비하는 시간은 얼마인가요?

면접관: 30분입니다.

지원자: 다국어 지원이 필요한가요?

면접관: 네 어떤 언어로도 이용 가능해야 합니다.

지원자: 어떤 비디오 해상도를 지원해야 할까요?

면접관: 현존하는 비디오 종류와 해상도를 대부분 지원해야 합니다.

지원자: 암호화가 필요할까요?

면접관: 네.

지원자: 비디오 파일 크기에 제한이 있습니까?

면접관: 작은 비디오, 중간 크기 비디오에 초점을 맞추도록 합시다.
비디오 크기는 최대 1GB로 제한합니다.

지원자: 아마존이나 구글, 마이크로소프트가 제공하는 클라우드 서비스를 활용해도 될까요

면접관: 좋은 질문입니다. 모든 걸 바닥부터 쌓아 올리는 것은 대부분 회사에게는 비현실적
활용할 수 있다면 하는 것이 바람직할 것입니다.

요구사항

- **빠른 비디오 업로드**
- **원할한 비디오 재생**
- **재생 품질 선택 가능**
- **낮은 인프라 비용(infrastructure cost)**
- **높은 가용성과 규모 확장성, 그리고 안정성**
- **지원 클라이언트 : 모바일 앱, 웹 브라우저, 스마트 TV**

개략적 규모 추정

- DAU 5백만 (tmillion)
- 한 사용자당 평균 5개 비디오 시청
- 10% 사용자는 하루에 1비디오 업로드
- 비디오 평균 크기 300MB
- 비디오 저장을 위해 매일 새로 요구되는 저장 용량 = 5백만 * 10% * 300MB = 150TB
- CND 비용
  - 클라우드 CDN을 통해 비디오를 서비스할 경우 CDN에서 나가는 데이터의 양에 따라 과금
  - 아마존 CloudFront를 CDN 솔루션으로 사용할 경우, 트래픽 전부가 미국에서 발생한다고 가정했을 때
    1GB당 $0.02의 요금 발생 (그림 14-2 참고, 문제를 단순화하기 위해
    비디오 스트리밍 비용만 고려)

Per Month	United States & Canada	Europe & Israel	South Africa, Kenya, & Middle East	South America	Japan	Australia	Singapore, South Korea, Taiwan, Hong Kong, & Philippines	India
First 10TB	$0.085	$0.085	$0.110	$0.110	$0.114	$0.114	$0.140	$0.170
Next 40TB	$0.080	$0.080	$0.105	$0.105	$0.089	$0.098	$0.135	$0.130
Next 100TB	$0.060	$0.060	$0.090	$0.090	$0.086	$0.094	$0.120	$0.110
Next 350TB	$0.040	$0.040	$0.080	$0.080	$0.084	$0.092	$0.100	$0.100
Next 524TB	$0.030	$0.030	$0.060	$0.060	$0.080	$0.090	$0.080	$0.100
Next 4PB	$0.025	$0.025	$0.050	$0.050	$0.070	$0.085	$0.070	$0.100
Over 5PB	$0.020	$0.020	$0.040	$0.040	$0.060	$0.080	$0.060	$0.100

그림 14-2

  - 매일 발생하는 요금은 5백만 * 5비디오 * 0.3GB * $0.02 = $150,000
  - 15만 달러의 비용은 적지 않은 금액이다. **비용을 줄이는 방법에 대해서는 상세 설계**에서 나온다.

2단계 : 개략적 설계안 제시 및 동의 구하기

- 앞으로 제시하는 설계안은 `CDN`과 `BLOB 스토리지`의 경우 클라우드를 활용한다. (면접관의 동의를 구함)

> *CND과 BLOB 스토리지를 직접 만들지 않는 이유*
>
> - 시스템 설계 면접은 모든 것을 밑바닥부터 만드는 것과 관계 없음
>   - 주어진 시간 안에 적절한 기술을 골라 설계를 마치는 것이 기술 각각이 어떻게 동작하는지 상세히 설명하는 것보다 중요.
>   - 비디오를 저장하기 위해 BLOB 저장소를 쓸 것이라면 그 사실만 언급해도 됨
>   BLOB 저장소를 어떻게 구현할 지 살세히 설계하는 것은 지나침
> - 규모 확장이 쉬운 BLOB 저장소나 CDN을 만드는 것은 지극히 복잡할 뿐 아니라 많은 비용 발생
>   - 넷플릭스나 페이스북 같은 공룡 기업도 모든 것을 스스로 구축하지 않음
>   - 넷플릭스는 AWS사용, 페이스북은 아카마이(Akamai) CDN을 이용

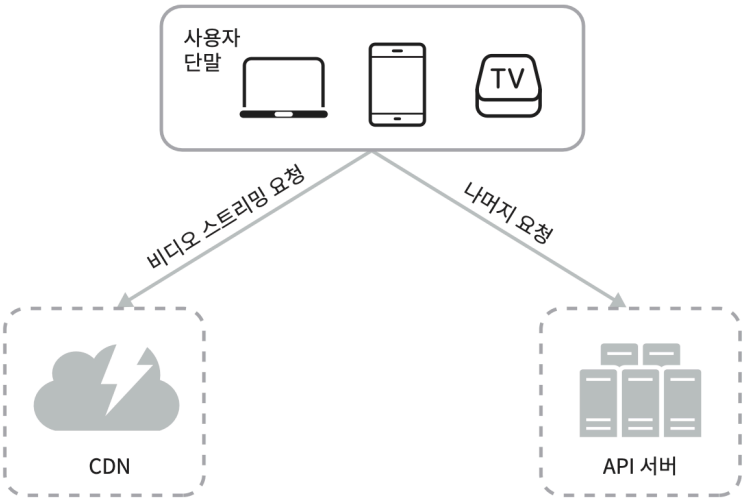

그림 14-3

이 시스템은 위 그림과 같이 세 개의 컴포넌트 사용

- 단말(clien) : 컴퓨터, 모바일, 스마트 TV를 통해 유뷰트 시청 가능
- CND : 비디오는 CND에 저장. 재생 버튼을 누르면 CDN으로부터 스트리밍이 이루어짐
- API 서버 : 비디오 스트리밍을 제외한 모든 요청은 API 서버가 처리.
    - 피드 추천(feed recommendation), 비디오 업로드 URL 생성, 메타데이터 데이터베이스와 캐시 갱신,
      사용자 가입 등등이 API 서버가 처리

다음 두 영역을 설계해본다.

- 비디오 **업로드**
- 비디오 **스트리밍**

비디오 업로드 절차

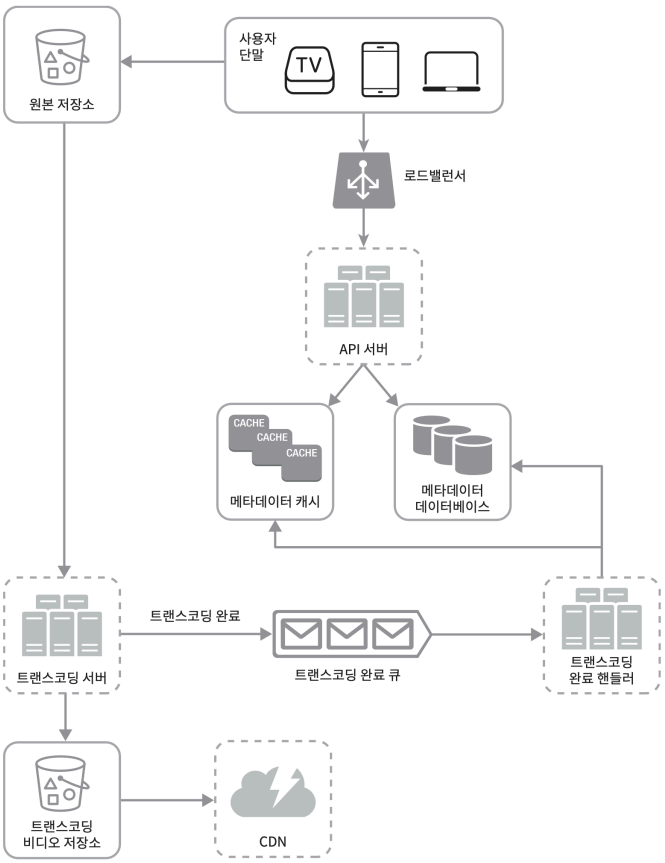

그림 14-4

업로드 절차에서 사용되는 컴포넌트

- **사용자** : 컴퓨터나 모바일 폰, 혹은 스마트 TV를 통해 유튜브를 시청하는 이용자
- **로드 밸런서** : API 서버 각각으로 고르게 요청을 분산하는 역할 담당
- **API 서버** : 비디오 스트리밍을 제외한 다른 모든 요청 처리
- **메타데이터 데이터베이스** : 비디오의 메터데이터를 보관
  - 샤딩(sharding)과 다중화(replication) 적용해 성능 및 가용성 향상
- **메타데이터 캐시**
  - 성능을 높이기 위해 `비디오 메타데이터` 와 `사용자 객체` 캐시
- **원본 저장소**
  - 원본 비디오를 보관할 대형 이진 파일 저장소(BLOB) 시스템
  - [6]에 따르면 저장소는 이진 데이터를 하나의 개체로 보관하는 데이터베이스 관리 시스템
- **트랜스코딩 서버**
  - 비디오 인코딩이라 부르기도 함

    - 비디오의 포맷을 변환하는 절차
    - 단말이나 대역폭 요구사항에 맞는 최적의 비디오 스트림을 제공하기 위해 필요
- **트랜스코딩 비디오 저장소**
    - 트랜스코딩이 완료된 비디오를 저장하는 BLOB 저장소
- **CDN**
    - 비디오를 캐시하는 역할 담당
    - 사용자가 재생 버튼을 누르면 비디오 스트리밍은 CDN을 통해 이루어짐
- **트랜스코딩 완료 핸들러**
    - 트랜스코딩 완료 큐에서 이벤트를 꺼내 메타데이터 캐시와 데이터베이스를 갱신할 작업 서버들

비디오 업로드 처리 프로세스

a. 비디오 업로드.

b. 비디오 메타데이터 갱신. 메타데이터에는 비디오 URL, 크기, 해상도, 포켓, 사용자 정보 포함

프로세스 a: 비디오 업로드

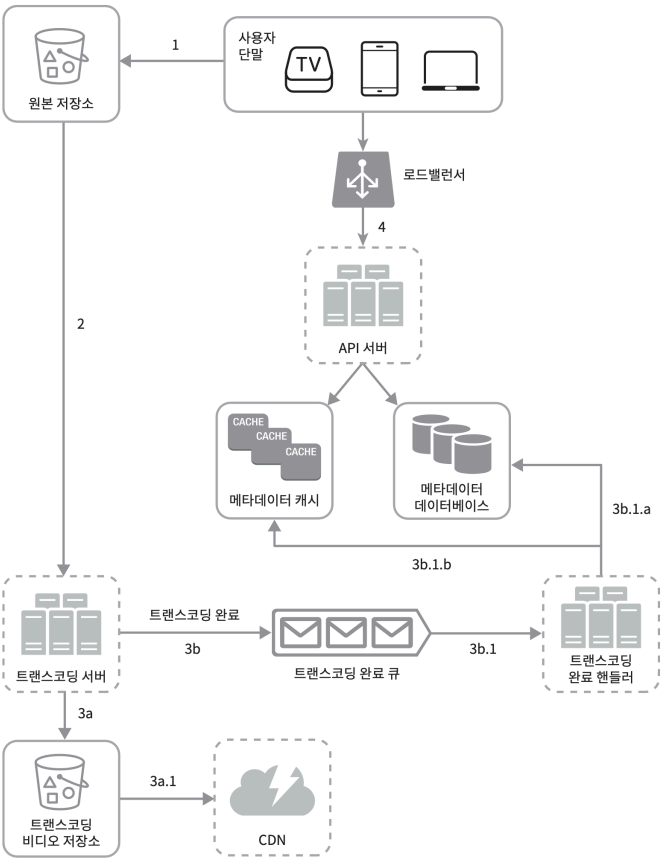

그림 14-5

1. 비디오를 원본 저장소에 업로드
2. 트랜스코딩 서버는 원본 저장소에서 해당 비디오를 가져와 트랜스코딩 시작
3. 트랜스코딩이 완료되면 아래 두 절차가 병렬적으로 수행
    a. 완료된 비디오를 트랜스코딩 비디오 저장소로 업로드
    b. 트랜스코딩 완료 이벤트를 트랜스코딩 완료 큐에 넣는다.
        i. 트랜스코딩이 끝난 비디오를 CDN에 올린다.
        ii. 완료 핸들러가 이벤트 데이터를 큐에서 꺼낸다.
        iii. `3b.1.a`, `3b.1.b` 완료 핸들러가 메타데이터 데이터베이스와 캐시를 갱신한다.
4. API 서버가 단말에게 비디오 업로드가 끝나서 스트리밍 준비가 되었음을 알린다.

프로세스 b: 메타데이터 갱신

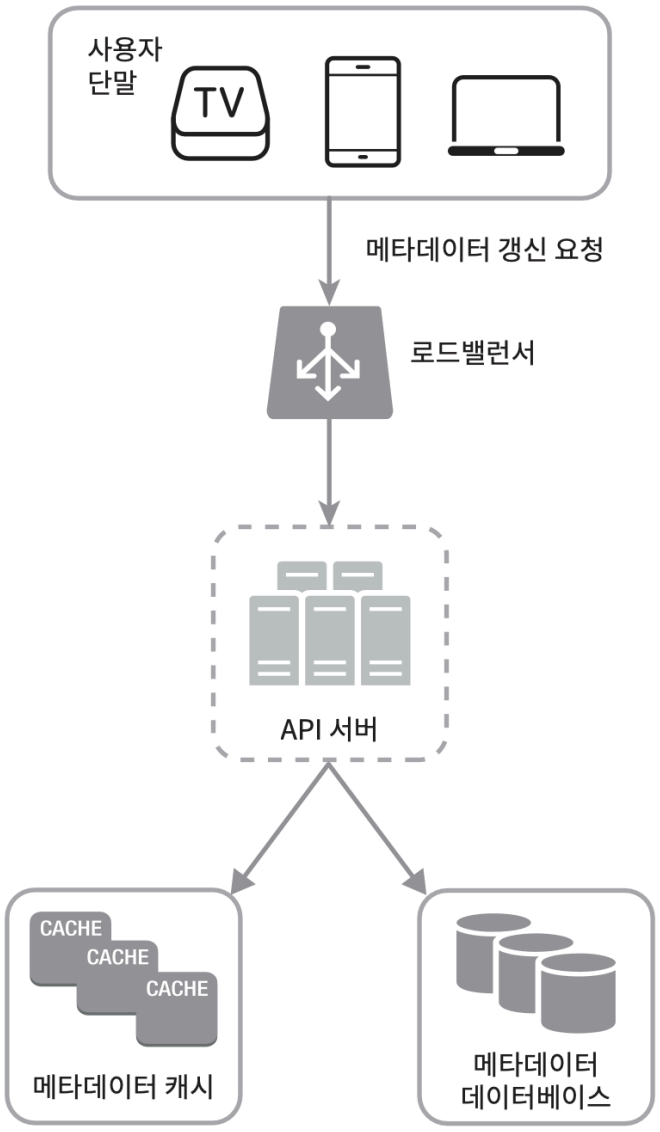

그림 14-6

- 원본 저장소에 파일이 업로드되는 동안, 단말은 병렬적으로 비디오 메타데이터 갱신 요청을 API 서버로 보냄
  - 이 요청에 포함된 메타데이터에는 `파일 이름`, `크기`, `포맷` 등의 정보가 들어 있음
  - API 서버는 이 정보로 메타데이터 캐시와 데이터베이스 업데이트

비디오 스트리밍 절차

- 스트리밍은 사용자의 단말이 원격지의 비디오로부터 지속적으로 비디오 스트림을 전송받아 영상을 재생하는 것을 뜻한다.
  - 다운로드가 완료되어야 영상을 볼 수 있다거나 하는 불편함이 없음

- 비디오 스트리밍을 다루기 전 **스트리밍 프로토콜(streaming protocol)**을 살펴보면 다음과 같다.
  - **비디오 스트리밍을 위해 데이터를 전송할 때 쓰이는** 표준화된 **통신방법** 이다.
  - 널리 사용되는 스트리밍 프로토콜
    - **MPEG-DASH**
      - **MPEG** : Moving Picture Experts Group의 약어
      - **DASH** : Dynamic Adaptive Streaming over HTTP의 약어
    - **애플 HLS**. HLS : HTTP Live Streaming의 약어
    - **마이크로소프트 스무드 스트리밍(Microsoft Smooth Streaming)**
    - **어도비 HTTP 동적 스트리밍(Adobe HTTP Dynamic Streaming, HDS)**
  - 프로토콜마다 지원하는 비디오 인코딩이 다르고 플레이어도 다르다.
    따라서 서비스의 용례에 맞는 프로토콜을 잘 골라야 한다.
  - 비디오 스트리밍 프로토콜에 대해 더 자세히 알고 싶으면 [7]을 살펴보자.
- 비디오는 CDN에 바로 스트리밍

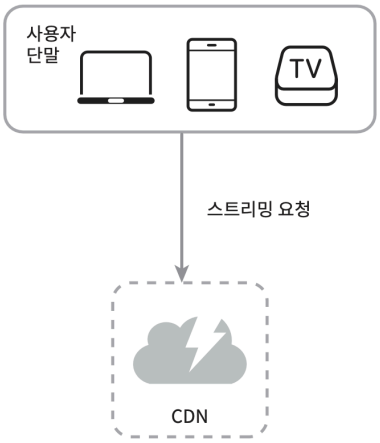

그림 14-7

  - 사용자의 단말에 **가장 가까운 CDN 엣지 서버**(edge server)가 비디오 전송을 담당 (**전송지연이 낮음**)

3단계 : 상세 설계

- 앞서 설계한 `비디오 업로드`와 `비디오 스트리밍` 부분에 대해 **최적화**를 진행하고, **오류 처리 매커니즘 소개**한다.

비디오 트랜스코딩

- 비디오를 녹화하면 단말(보통 전화나 카메라)은 해당 비디오를 특정 포맷으로 저장한다.
  - 이 비디오가 다른 단말에서도 순조롭게 재생되려면 다른 단말고 호환되는 **비트레이트(bitrate)**와 포맷으로 저장되어야 한다.
  - 비트레이트 : 비디오를 구성하는 비트가 얼마나 빨리 처리되어야 하는지를 나타내는 단위
    - 일반적으로 비트레이트가 높은 비디오가 고화질 비디오
    - 높은 비디오 스트림을 정상 재생하려면 높은 성능의 컴퓨팅 파워와 인터넷 회신 속도가 빨라야 함
- 비디오 트랜스코딩은 다음과 같은 이유로 중요
  - **가공되지 않은 원본 비디오(raw video)는 저장 공간을 많이 차지**
    - 초당 60프레임으로 녹회된 HD 비디오는 수백 GB 저장공간을 차지할 수 있음
  - **상당수의 단말과 브라우저는 특정 종류의 비디오 포맷만 지원**
    - 호환성 문제를 해결하라면 하나의 비디오를 여러 포맷으로 인코딩해 두는 것이 좋음
  - **사용자에게 끊김 없는 고화질 비디오를 재생하려면**
    - 네트워크 대역폭이 충분하지 않다면 저화질로, 대역폭이 충분하다면 고화질로 보내는 것이 좋음
- 인코딩 포맷은 다양하지만 대부분 다음 두 부분으로 구성

  ▼ **컨테이너(container)**

  - `비디오 파일`, `오디오`, `메타데이터`를 담는 바구니 같은 것
  - 컨테이너 포맷은 `.avi`, `.mov`, `.mp4` 같은 파일 확장자를 보면 알 수 있음

  ▼ **코덱(codec)**

  - 비디오 화질은 보존하면서 파일 크기를 줄일 목적으로 고안된 압축 및 압축 해제 알고리즘이다.
  - 가장 많이 사용되는 비디오 코덱으로는 `H.264`, `VP9`, `HEVC`가 있음

유향 비순환 그래프(DAG, Directed Acyclic Graph) 모델

- 각기 다른 유형의 비디오 프로세싱 파이프라인을 지원하고, 처리 과정의 병렬성을 높이기 위해 적절한 수준의 **추상화를 도입**하여 **클라이언트 프로그래머**로 하여금 **실행할 작업(task)을 손수 정의할 수 있도록** 해야 한다.
  - 비디오 트랜스코딩 자체가 비용이 비쌈 (컴퓨팅 자원, 시간 등)
  - 콘텐츠 창작자 별 자기만의 비디오 프로세싱 요구사항이 있을 수 있음
    - 워터마크, 섬네일 이미지, 화질 등
  - ex) **페이스북** 스트리밍 비디오 엔진은 **유향 비순환 그래프(DAG) 프로그래밍 모델 도입 (유연성, 병렬성)**
- 다음 그림처럼 이 책의 설계에서도 DAG 모델을 도입해 유연성과 병렬성을 달성한다.

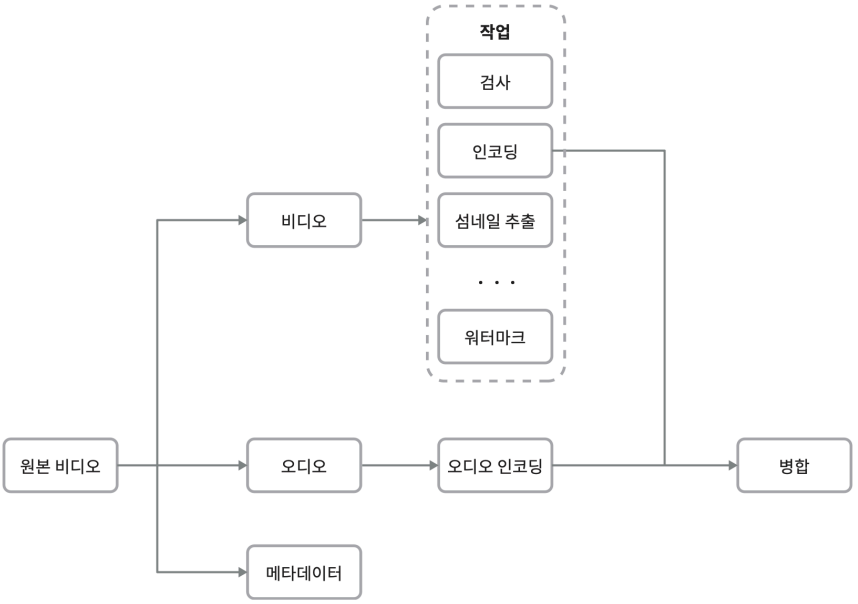

그림 14-8

- 위 그림에서 원본 비디오는 비디오, 오디오, 메타데이터의 세 부분으로 나눠 처리한다.

> ### 비디오 부분에 적용되는 작업

  - **검사(inspection)** : 좋은 품질의 비디오인지, 손상은 없는지 확인하는 작업
  - **비디오 인코딩(video encoding)** : 다양한 해상도, 코덱, 비트레이트 조합으로 인코딩하는 작업

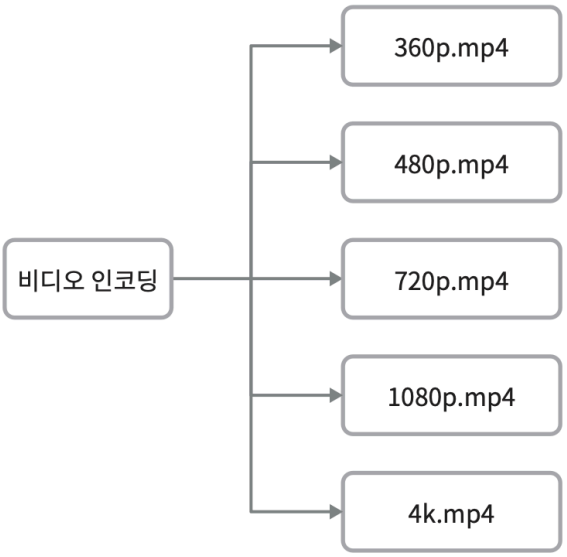

그림 14-9

비디오 인코딩의 결과물 사례

- **섬네일(thumbnail)** : 사용자가 업로드한 이미지나 비디오에서 자동 추출된 이미지로 섬네일 생성
- **워터마크(watermark)** : 비디오에 대한 식별정보를 이미지 위에 오버레이 형태로 띄워 표시하는 작업

비디오 트랜스코딩 아키텍처

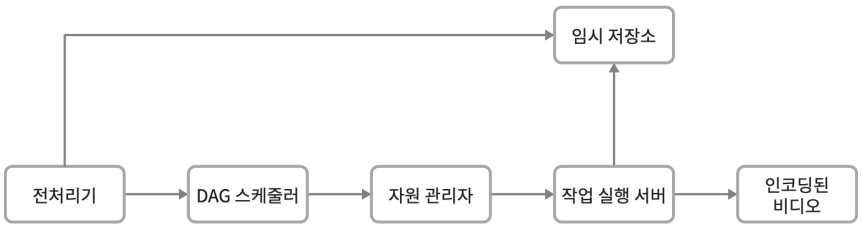

그림 14-10

- 다섯 개의 주요 컴포넌트로 구성
  - **전처리기(preprocessor)**
  - **DAG 스케줄러**
  - **자원 관리자(resource manager)**
  - **작업 실행 서버(resource worker)**
  - **임시 저장소(temporary storage)**

- 이 아키텍처가 동작한 결과로 인코딩된 비디오가 만들어진다.

전처리기

- 전처리가 담당하는 세 가지 일

1. **비디오 분할(video splitting)**
    - 비디오 스트림을 GOP(Group of Pictures)라고 불리는 단위로 쪼갬
    - GOP는 특정 순서로 배열된 프레임 그룹
    - 하나의 GOP는 독립적으로 재생 가능하며, 길이는 보통 몇 초 정도
    - 어떤 종류의 오래된 단말이나 브라우저는 GOP 단위의 비디오 분할을 지원하지 않음
        - 그런 단말의 경우 전처리가 비디오 분할을 대신 함
2. **DAG 생성**
    - 클라이언트 프로그래머가 작성한 설정 파일에 따라 DAG를 만듦

그림 14-12

2개 노드와 1개 연결선으로 구성된 DAG

그림 14-13 (출처: [9])

왼쪽 DAG는 이 두 개의 설정 파일로부터 생성된 것 [9]

3. **데이터 캐시**
    - 전처리기는 분할된 비디오의 캐시임
    - 안정성을 높이기 위해 전처리기는 GOP와 메타데이터를 임시 저장소에 보관
    - 비디오 인코딩이 실패하면 시스템은 이렇게 보간된 데이터를 활용해 인코딩 재개

DAG 스케줄러

- DAG 그래프를 몇 개 단계로 분할한 다음 각각을 자원 관리자의 작업 큐에 넣음

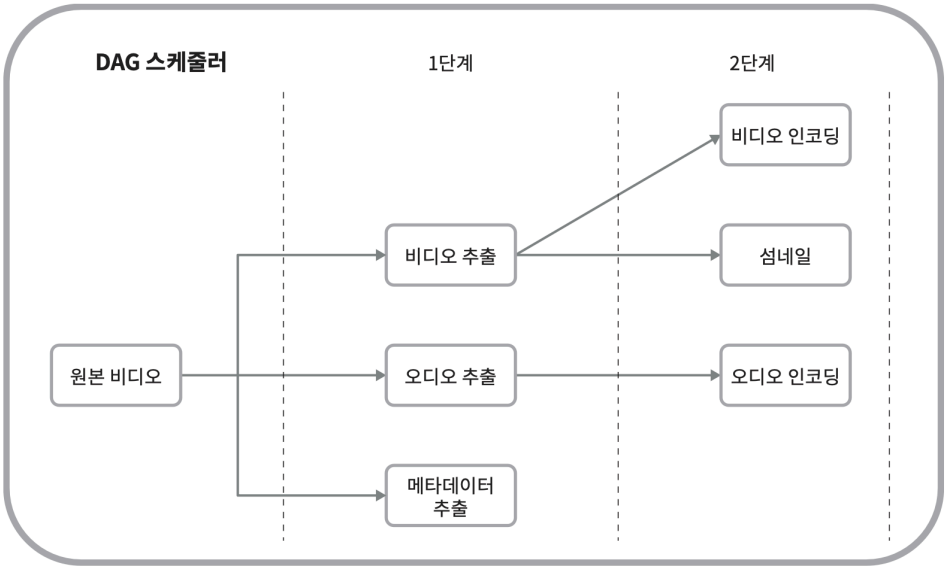

그림 14-15

하나의 DAG 그래프를 2개 작업 단위 로 쪼갠 사례

- 첫 번째 단계 : 비디오, 오디오 메타데이터 분리
- 두 번째 단계 : 해당 비디오 파일을 인코딩, 섬네일 추출, 오디오 파일 인코딩

자원 관리자

- 자원 배분을 효과적으로 수행하는 역할 담당
- 다음 그림처럼 세 개의 큐 와 작업 스케줄러 로 구성

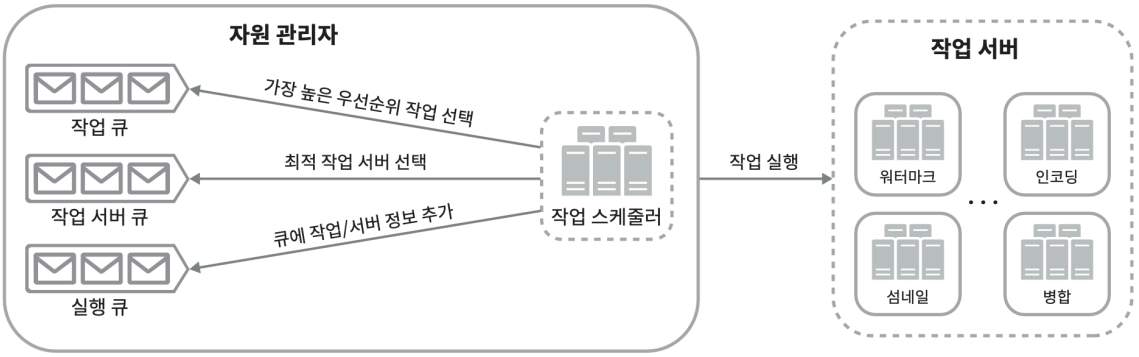

그림 14-17

- **작업 큐(task queue)** : 실행할 작업이 보관되어 있는 우선순위 큐
- **작업 서버 큐(worker queue)** : 작업 서버의 가용 상태 정보가 보관되어 있는 우선순위 큐

- **실행 큐(running queue)** : 현재 실행 중인 작업 및 작업 서버 정보가 보관되어 있는 큐
- **작업 스케줄러** : 최적의 작업/서버 조합을 골라, 해당 작업 서버가 작업을 수행하도록 지시하는 역할

- 작업 관리자는 다음과 같이 동작함
  - 작업 큐에서 가장 높은 우선순위의 작업을 꺼냄
  - 해당 작업을 실행하기 적합한 작업 서버를 고름
  - 해당 작업 서버에게 작업 실행 지시
  - 해당 작업이 어떤 서버에게 할당되었는지에 관한 정보 실행 큐에 삽입
  - 작업이 완료되면 해당 작업을 실행 큐에서 제거

작업 서버

- DAG 정의된 작업 수행, 작업 종류에 따라 작업 서버도 구분하여 관리해야 됨

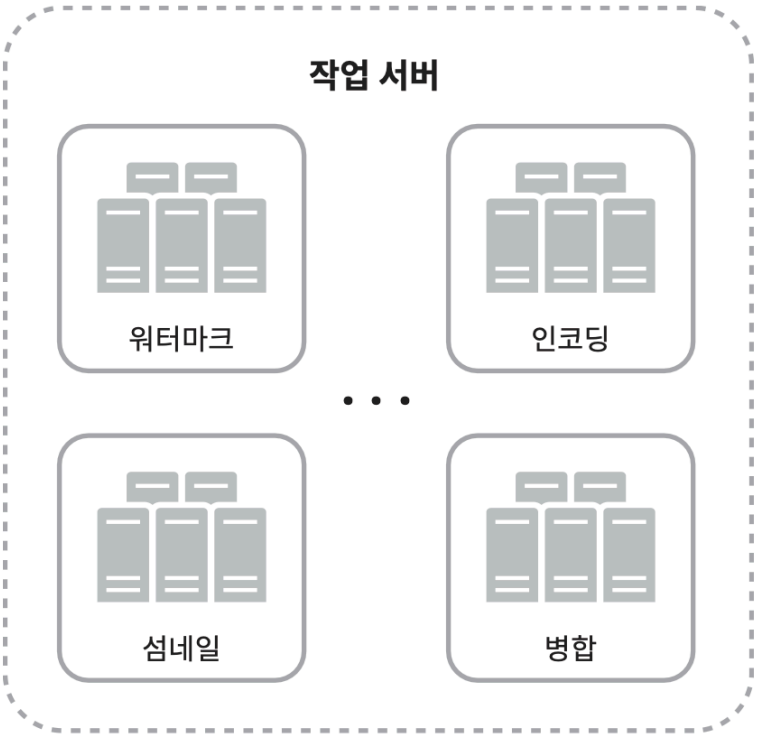

그림 14-19

임시 저장소

- `저장할 데이터의 유형`, `크기`, `이용 빈도`, `데이터 유효기간` 등에 따라 **사용할 저장소가 달리진다.**
  - ex)

    메타데이터는 작업 서버가 빈번히 참조하는 정보이고, 크기도 작다.
    따라서
    메모리에 캐시해 두면 좋을 것이다.
    하지만 비디오/오디오 데이터는 BLOB 저장소에 두는 것이 좋다.

인코딩된 비디오

- 인코딩 파이프라인의 최종 결과물 `temp_720p.mp4`와 같은 이름을 갖음

시스템 최적화

- `속도`, `안정성`, `비용` **최적화** (비디오 업로드와 스트리밍, 트랜스코딩 절차는 위에서 다룸)

속도 최적화: 비디오 병렬 업로드

- 하나의 비디오는 다음 그림처럼 작은 GOP들로 분할할 수 있음

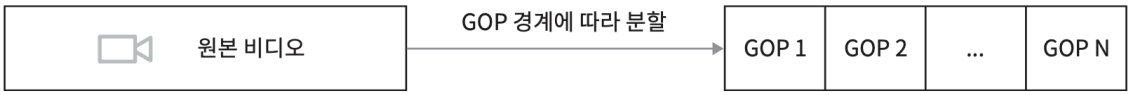

그림 14-22

- 이렇게 분할한 GOP를 병렬적으로 업로드하면 설사 일부가 실패해도 빠르게 업로드를 재개할 수 있음
  - 비디오를 GOP 경계에 맞춰 분할하는 작업을 단말이 수행하면 다음 그림처럼 업로드 속도를 높일 수 있음

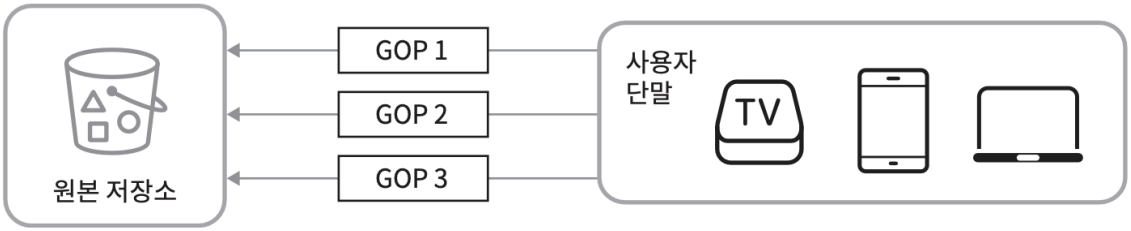

그림 14-23

속도 최적화: 업로드 센터를 사용자 근거리에 지정

- 업로드 센터를 여러 곳에 두어, 미국 거주자는 북미 지역, 아시아 사용자는 아시아 업로드 센터 이런 식으로

속도 최적화: 모든 절차를 병렬화 - 이를 위해서는 지금까지 설계안을 변경해야 함

- 느슨하게 결합된 시스템을 만들어 병렬성을 높이는 방법 - CDN으로 옮기는 절차를 다시 확인해보자
- 메시지 큐 도입 전 - 다음 그림과 같이 동작

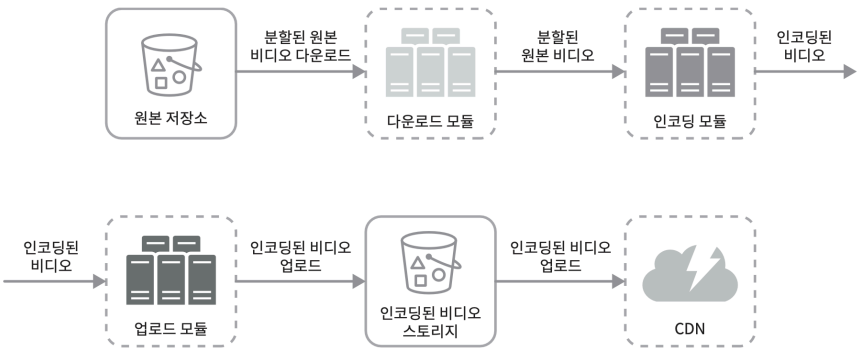

그림 14-25

  - 어떤 단계의 결과물은 이전 담계의 결과물을 입력으로 사용하여 만들어 짐 이런 의존성이 있으면 벙렬성을 높이기 어려움
  - 이 시스템의 결합도를 낮추기 위해 다음 그림과 같이 메시지 큐를 도입
- 메시지 큐 도입 후 - 다음 그림과 같이 동작

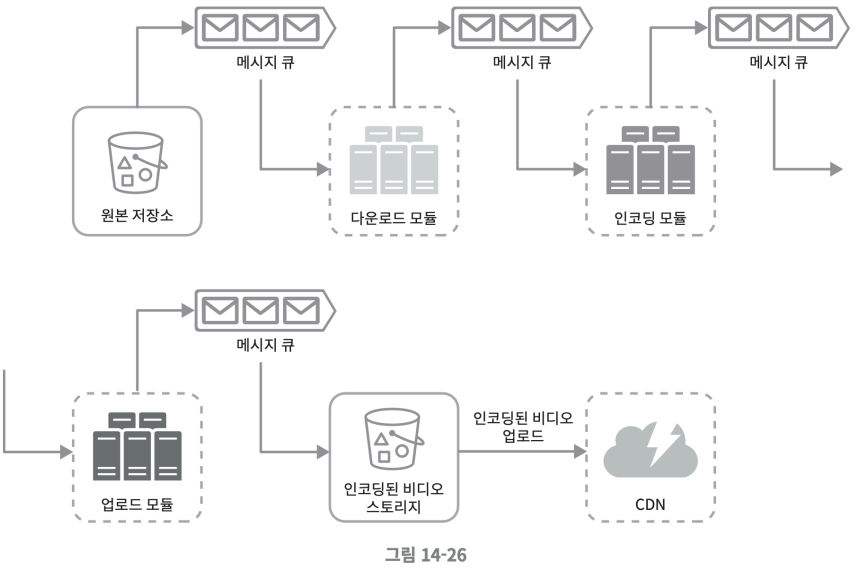

그림 14-26

- 메시지 큐를 도입하기 전에 인코딩 모듈은 다운로드 모듈의 작업이 끝나기를 기다려야 했다.
- 메시지 큐를 도입한 뒤 인코딩 모듈은 다운로드 모듈의 작업이 끝나기를 더 이상 기다릴 필요가 없음
  메시지 큐에 보관된 이벤트 각각을 인코딩 모듈은 병렬적으로 처리 가능

안정성 최적화: 미리 사인된 업로드 URL

- 모든 제품의 가장 중요한 측면
- 허가받은(authorized) 사용자만이 올바른 장소에 비디오 업로드 가능

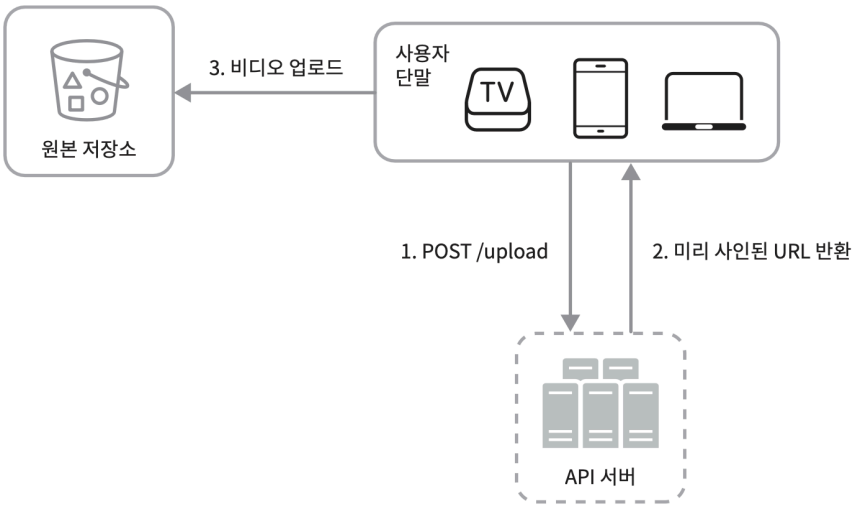

그림 14-27

미리 사인된(pre-signed) 업로드 URL 이용

- 이를 위해 업로드 절차는 다음과 같이 변경
  1. 클라이언트는 HTTP 서버에 POST 요청을 하여 미리 사인된 URL을 받는다.
     - 해당 URL이 가리키는 객체(object)에 대한 접근 권한이 이미 주어져 있는 상태
     - **미리 사인된(pre-signed) URL**이라는 용어는 **아마존 S3에서 쓰이는 용어**
       - 다른 클라우드 업체는 다른 이름을 사용할 수 있음
         ex.
         애저(Azure)에서 BLOB 저장소는 `접근 공유 시그니처(Shared Access Signature)`
  2. API 서버는 미리 사인된 URL을 돌려줌
  3. 클라이언트는 해당 URL이 가리키는 위치에 비디오 업로드

안정성 최적화: 비디오 보호

- 비디오의 저작권을 보호하기 위해 다음 세 가지 선택지 중 하나를 고를 수 있음

1. **디지털 저작권 관리(DRM, Digital Rights Management) 시스템 도입**
   - 애플의 페어플레이(FairPlay), 구글의 와이드바인(Widevine), 마소의 플레이레디(PlayReady)가 있음
2. **AES 암호화**
   - 비디오를 암호화하고 접근 권한을 설정하는 방식
   - 암호화된 비디오는 재생 시에만 복호화, 허락된 사용자만 암호화된 비디오를 시청할 수 있음
3. **워터마크**
   - 비디오 위에 소유자 정보를 포함하는 이미지 오버레이를 올려놓는 것. (ex. 회사 로고 같은)

비용 최적화

- CDN은 시스템의 핵심 부분, 하지만 비쌈, 데이터의 크기가 클 수록 더 비쌈
- 유튜브의 비디오 스트리밍은 롱테일 분포([11], [12])를 따른다.
  인기 잇는 비디오는 빈번히 재생되지만, 나머지는 거의 보는 사람이 없다.

- 이 부분을 착안해 몇 가지 최적화를 시도할 수 있음

1. **인기 비디오는 CDN**을 통해 재생하되 **다른 비디오는 비디오 서버**를 통해 재생

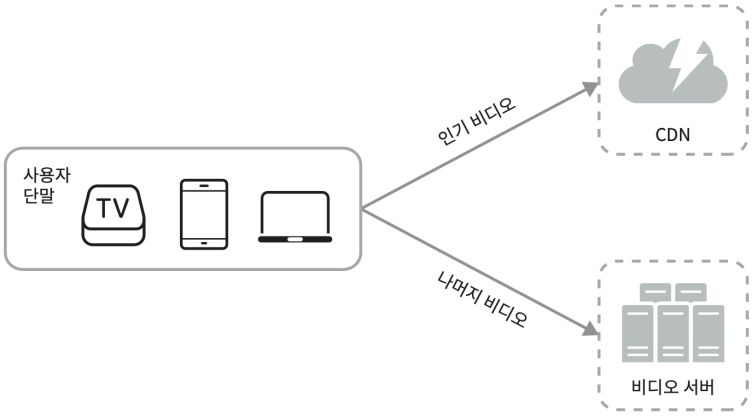

그림 14-28

2. 인기가 별로 없는 비디오는 인코딩 할 필요가 없을 수 있음. - 짧은 비디오라면 필요할 때 인코딩 후 재생
3. 어떤 비디오는 특정 지역에서만 인기가 높음. - 이런 비디오는 다른 지역에 옮길 필요가 없음
4. CDN을 직접 구축하고 인터넷 서비스 제공자(ISP, Internet Service Provider)와 제휴
    - CDN을 직접 구축하는 것은 초대형 프로젝트다. (대규모 스트리밍 사업자라면 필요할 수도?)
    - ISP로는 컴캐스트(Comcast), AT&T, 버라이즌(Verizon) 등이 있고,
      ISP는 전세계 어디나 있으며 사용자와 가깝다.
      이들과 제휴하면 사용자 경험도 향상시킬 수 있고, 인터넷 사용 비용도 절약할 수 있다.

- 이 모든 최적화는 콘텐츠 인기도, 이용 패턴, 비디오 크기 드의 데이터에 근거한 것이다.
    - 최적화 시도 전 시청 패턴을 분석하는 것이 중요 - [12], [13] 참고

오류 처리

- 시스템 오류의 두 가지 종류

1. **회복 가능 오류(recoverable error)**

- 특정 비디오 세그먼트를 트랜스코딩하다 실패했다든가 하는 오류는 회복 가능한 오류에 속함
- 일반적으로 이런 오류는 몇 번 재시도(retry)하면 해결 가능
- 단, 계속해서 실패하고 복구가 어렵다 판단되면 클라이언트에게 적절한 오류 코드 반환해야 함

2. **회복 불가능 오류(non-recoverable error)**
    - 비디오 포맷이 잘못되었다거나 하는 회복 불가능한 오류가 발견하면 시스템은 해당 비디오에 대한 작업을 중단하고 클라이언트에게 적절한 오류 코드를 반환해야 함

- 시스템 컴포넌트 각각에 발생할 수 있는 오류에 대한 전형적 해결 방법은 다음과 같다.
    - `업로드 오류` **:** 몇 회 재시도
        -
    - `비디오 분할 오류`
        - 낡은 버전의 클라이언트가 GOP 경계에 따라 비디오를 분할하지 못하는 경우라면 전체 비디오를 서버로 전송하고 해당 비디오 분할을 처리하도록 함
    - `트랜스코딩 오류` : 재시도
    - `전처리 오류` : DAG 그래프 재생성
    - `DAG 스케줄러 오류` : 작업을 다시 스케줄링
    - `자원 관리자 큐에 장애 발생` : 사본(replica)을 이용
    - `작업 서버 장애` : 다른 서버에서 해당 작업을 재시도
    - `API 서버 장애` : API 서버는 무상태 서버이므로 신규 요청은 다른 API 서버로 우회
    - `메타데이터 캐시 서버 장애`
        - 데이터는 다중화되어 있으므로 다른 노드에서 데이터를 가져올 수 있음
        - 장애가 난 캐시 서버는 새로운 것으로 교체
    - `메타데이터 데이터베이스 서버 장애`
        - 주 서버가 죽었다면 부 서버 가운데 하나를 주 서버로 교체
        - 부 서버가 죽었다면 다른 부 서버를 통해 읽기 연산을 처리하고 죽은 서버는 새 것으로 교체

4단계 : 마무리

추가로 해볼 논의

- **API 계층의 규모 확장성 확보 방안**
  - API 서버는 무상태 서버이므로 수평적 규모 확장이 가능하다는 사실을 언급하면 좋음
- **데이터베이스 계층의 규모 확장성 확보 방안**
  - 데이터베이스의 다중화와 샤딩 바업에 대해 이야기하자.
- **라이브 스트리밍(live streaming)**
  - 비디오를 실시간으로 녹화하고 방송하는 절차
  - `라이브`와 `비-라이브` 스트리밍 시스템 간에는 **비슷한 점**으로 `비디오 업로드`, `인코딩`, `스트리밍`이 필요
    다른 점으론 다음과 같음
    - **라이브**의 경우에는 응답 지연이 좀 더 낮아야 함. 따라서 **스트리밍 프로토콜 선정에 유의**해야 함
    - **라이브**의 경우 **작은 단위의 데이터를 실시간으로 빨리** 처리해야 하기 때문에 **병렬화는 떨어짐**
    - **라이브**의 경우 **오류 처리 방법을 달리**해야 함. **많은 시간이 걸리는 방안은 사용할 수 없음**
- **비디오 삭제(takedown)**
  - 저작권을 위반한 비디오, 선정적 비디오, 불법적 행위에 관계된 비디오는 내려야 한다.
  - 내릴 비디오는 `업로드 과정에서 식별`해 낼 수도 있지만, `사용자의 신고 절차`를 통해 판별할 수도 있음

Chapter 14: DESIGN YOUTUBE

1. YouTube by the Numbers
2. YouTube Demographics
3. CloudFront Pricing

4. Netflix on AWS
5. Akamai homepage
6. Binary large object
7. Streaming Protocols for Live Broadcasting: Everything You Need to Know
8. SVE: Distributed Video Processing at Facebook Scale
9. Weibo video processing architecture
10. Delegate access with a shared access signature
11. Seattle Conference on Scalability: YouTube Scalability
12. Understanding the Characteristics of Internet Short Video Sharing: YouTube as a Case Study
13. Content Popularity for Open Connect