

4

14장 유튜브 설계

유튜브는 간단히 생각하는 것과 달리 매우 복잡하다.

유튜브에 대한 놀라운 통계 자료가 있다. 2020년 기준,

1. 월간 능동 사용자 수 : 2십억 ++
2. 매일 재생되는 비디오 수 : 5십 억 ++
3. 미국 성인 가운데 73%가 유튜브를 이용한다.
4. 5천만명의 창작자
5. 유튜브의 광고 수입은 2019년 기준 150억 달러
6. 모바일 인터넷 트래픽 중 37% 점유
7. 80개 언어로 이용 가능

1단계 문제 이해 및 설계 범위 확정

간단하게 추린 적절한 요구사항은 다음과 같을 것이다.

1. 비디오 Upload / Streaming 기능
2. Multi platform 지원
3. 하루 사용자 수는, 5백만
4. 하루 평균 사용 시간은 30분 정도
5. 다국어 지원
6. 현존하는 대부분의 비디오 해상도 지원
7. 암호화 필요

8. 비디오 크기는 최대 1GB
9. 클라우드 서비스 이용 가능

따라서 진정한 요구사항은

- 빠른 비디오 업로드
- 원활한 비디오 재생
- 재생 품질 선택 가능
- 낮은 인프라 비용
- 높은 가용성과 규모 확장성 및 안정성
- 다양한 플랫폼 지원

개략적 규모는

- DAU(일간 능동 사용자) 수는 5백만
- 한 사용자는 하루 평균 5개 비디오 시청
- 10% 사용자가 하루에 1비디오 업로드
- 비디오 평균 크기는 300mb
 - 따라서 매일 새로 요구되는 저장 용량은 $5m * 10\% * 300MB = 150TB$
- CDN 비용
 - 클라우드 CDN을 통해 비디오를 서비스할 경우, CDN에서 나가는 데이터 양에 따라 과금한다.
 - AWS의 CF를 사용할 경우 1GB 당 \$0.02가 소요된다.
 - 비디오 저장을 위해 매일 새로 요구되는 저장 요금은
 - $DAU(5m) * 5 \text{ video} * 0.3GB * \$0.02 = \$150,000$ 이다.

위 추정 결과에 따라 CDN을 통해 서비스를 지원하려고 하면, 엄청난 서비스 비용을 감당해야 한다. 비용을 줄이기 위한 방법은 상세 설계에서 논의한다.

2단계 개략적 설계안 제시 및 동의 구하기

여기서 제시하는 설계안의 CDN과 Blob Storage는 클라우드 서비스를 사용하기로 한다.

왜 클라우드 서비스를 쓰는가에 대해선.. 그냥 어렵고 복잡하고 설계가 중요한거지 구현이 중요한게 아니라고 말할 수 있다.

개략적으로 보면 이 시스템은 다음 3 가지의 컴포넌트로 구성된다.

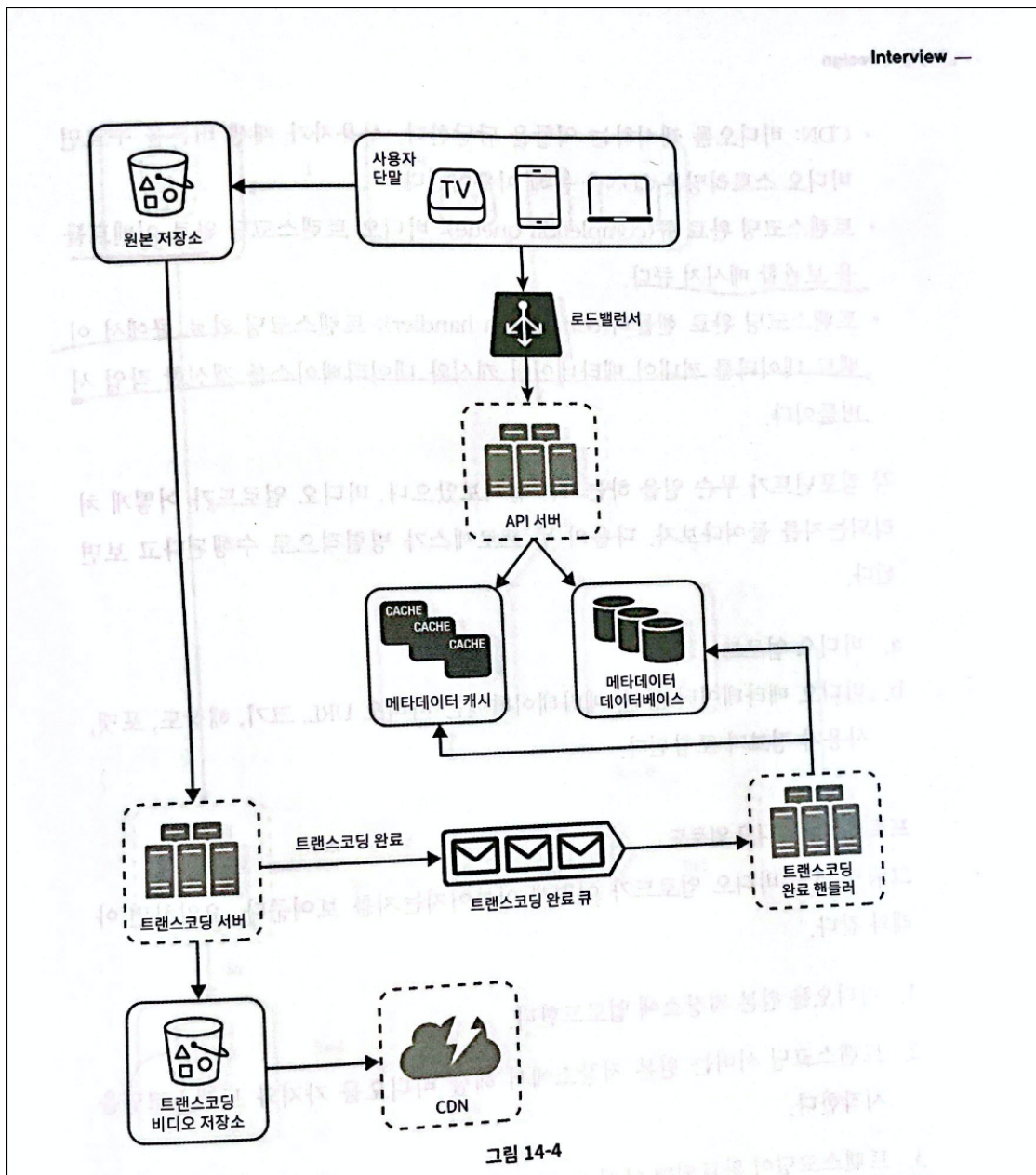
1. 단말(Client) : 컴퓨터 / 모바일 / TV를 통해 시청할 수 있다.
2. CDN : 비디오는 CDN에 저장된다. 재생 버튼을 누르면 CDN을 통해 스트리밍이 이루어진다.
3. API 서버 : 스트리밍을 제외한 모든 요청은 API 서버가 처리한다.

만약 면접관이 다음 두 영역을 설계해줄 것을 요청 했다고 하면

- 비디오 업로드 절차
- 비디오 스트리밍 절차

이 두 영역에 대한 설계를 시작하면 된다.

비디오 업로드 절차



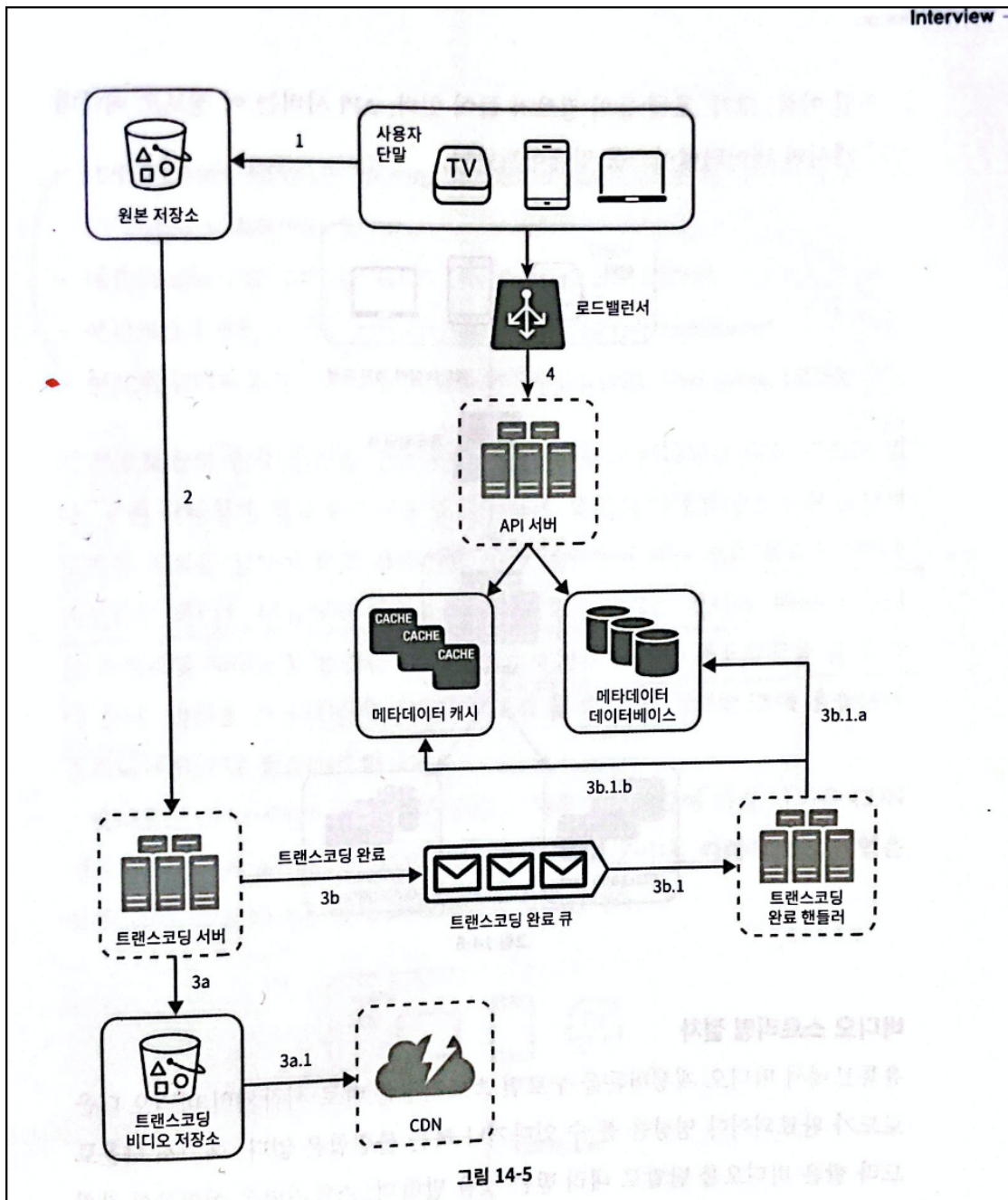
- 사용자 : 유튜브를 이용하는 사람
- 로드밸런서 : API 서버로 고르게 요청을 분산하는 역할
- API 서버 : 비디오 스트리밍을 제외한 다른 모든 요청을 처리한다
- 메타데이터 데이터베이스 : 비디오의 메타데이터를 보관한다. 샤딩과 다중화를 적용하여 성능 및 가용성 요구사항을 충족한다.
- 메타데이터 캐시 : 성능을 높이기 위해 비디오 메타데이터와 사용자 객체는 캐시한다.
- 원본저장소 : 원본 비디오를 보관할 대영 이진 파일 저장소(BLOB) 시스템이다.

- 트랜스코딩 서버 : 트랜스 코딩은 인코딩이라고도 불리며, 비디오 포맷이나 압축 등 변환을 제공하며 단말이나 대역 폭 요구 사항에 맞춘 비디오 스트림을 제공하기 위해 사용된다.
- 트랜스코딩 비디오 저장소 : 인코딩이 완료된 비디오 BLOB 저장소다.
- CDN : 비디오를 캐시하는 역할을 한다. 사용자가 재생 버튼을 누르면 비디오 스트리밍은 CDN을 통해 이루어진다.
- 트랜스코딩 완료 큐 : 인코딩 완료 이벤트를 보관할 메세지 큐다.
- 트랜스코딩 완료 핸들러 : 인코딩 완료 큐에서 이벤트 데이터를 꺼내 메타데이터 캐시와 데이터베이스를 갱신할 작업 서버들이다.

이제 각 컴포넌트가 수행할 역할을 살펴보았으니, 업로드 처리 프로세스를 살펴본다. 다음의 두 프로세스가 병렬적으로 실행된다고 보면 된다.

- 비디오 업로드.
- 비디오 메타데이터 갱신, 메타데이터는 포맷 URL 크기 해상도 등이 포함된다.

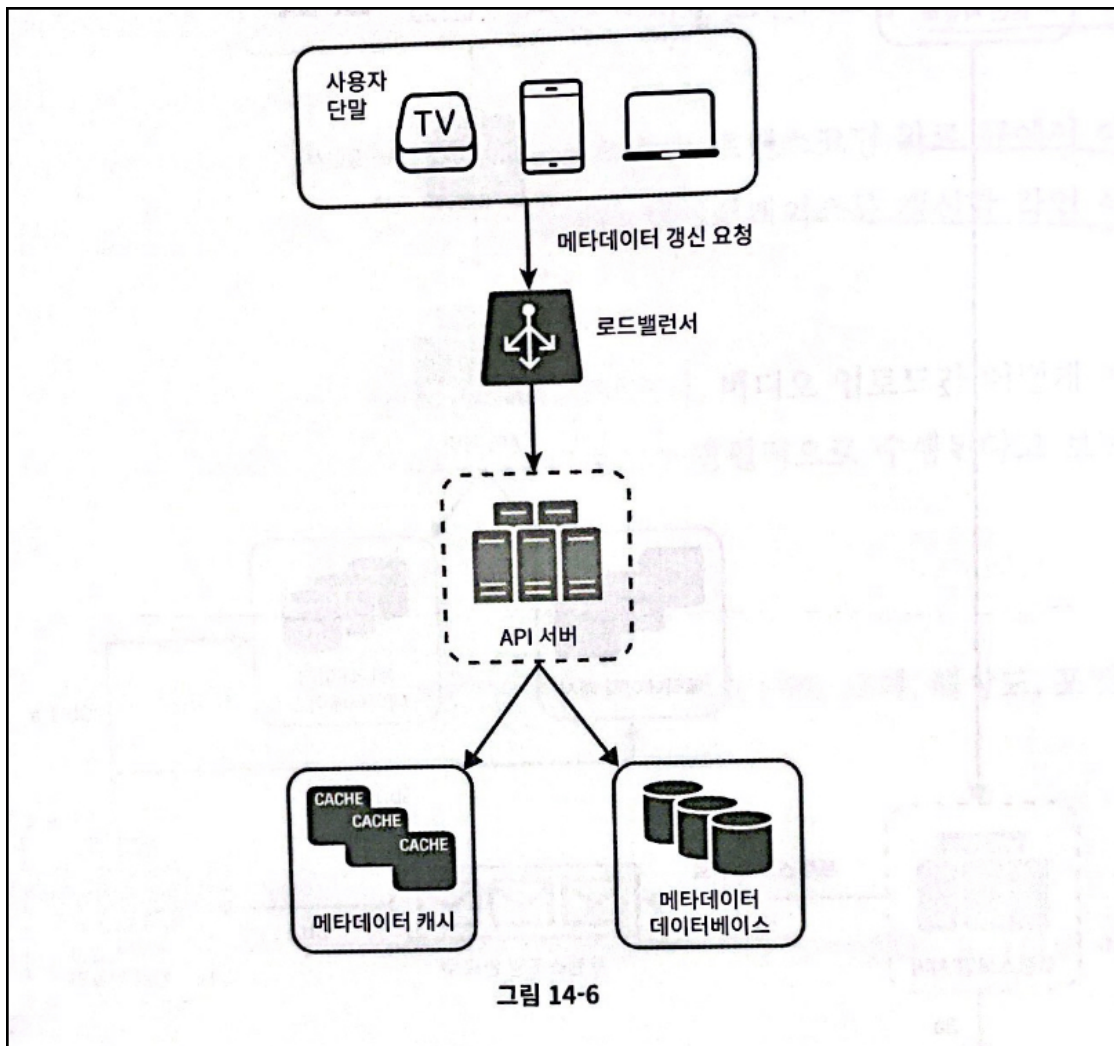
프로세스 a: 비디오 업로드



1. 비디오를 원본 저장소에 업로드한다.
2. 인코딩(=트랜스 코딩, 이하 인코딩이라고 부르기로 한다.) 서버는 원본 저장소에서 비디오를 가져와 인코딩을 수행한다.
3. 인코딩이 완료되면, 두 절차가 병렬적으로 실행된다.
 - a. 완료된 비디오를 인코딩 저장소로 병렬 업로드한다.
 - b. 인코딩 완료 이벤트를 큐에 넣는다.
 - i. 인코딩이 끝난 비디오를 CDN에 올린다.

- ii. 완료 핸들러가 이벤트 데이터를 큐에서 꺼낸다.
 - iii. 완료 핸들러가 메타데이터 데이터베이스와 캐시를 갱신한다.
4. API 서버가 단말에 비디오 업로드가 끝나 스트리밍이 준비 됐음을 알린다.

프로세스 b: 메타데이터 갱신



원본 저장소에 파일이 업로드 되는 동안, 단말은 병렬적으로 비디오 메타 데이터 갱신 요청을 API 서버에 보낸다. 이 요청이 포함된 데이터는 파일이름 / 크기 / 포맷 등 정보가 있다. API 서버는 이 정보로 메타데이터 캐시와 데이터베이스를 업데이트 한다.

비디오 스트리밍 절차

유튜브에서 비디오 재생 버튼을 누르면 자동으로 스트리밍이 시작된다. 다운로드 받을 필요는 없다.

유튜브 스트리밍을 이해하기 위해선, 스트리밍 프로토콜에 대해서 알아야 한다. 스트리밍 프로토콜은 비디오 스트리밍을 위해 데이터를 전송할 때 쓰이는 표준 통신방법이다. 널리 사용되는 것들은 다음과 같다.

- MPEG-DASH
- Apple의 HLS
- 마이크로 소프트의 Smooth Streaming
- Adobe HTTP Dynamic Streaming, HDS

전부 알아야 할 필요는 없지만, 프로토콜마다 지원하는 비디오 인코딩도 다르고 플레이어도 다르다. 따라서 서비스 용법에 맞는 프로토콜을 잘 선택해야 한다.

비디오는 CDN에서 바로 스트리밍된다.

사용자의 단말에서 가장 가까운 CDN 에지 서버가 비디오 전송을 담당할 것이다. 전송지연은 아주 낮다.



CDN ...에 대하여

간단하게 CDN은 일반적으로 지리적으로 가장 가까운 네트워크(edge 서버)나 트래픽이 가장 여유로운 곳을 찾아 네트워크 트래픽을 분산시켜주는 기술을 의미한다.

Origin 서버의 Contents를 Cache 서버에 보존하여 정적 객체를 클라이언트로 보내주기 때문에 전송 지연(Latency)가 낮아 사용성이 좋다.

3단계 상세 설계

비디오 트랜스코딩

비디오를 녹화하면 해당 비디오를 특정 포맷으로 저장한다. 보통 클라이언트의 단말과 비디오의 포맷이 호환되어야 재생되기 때문에 포맷을 변경하는 일은 비디오 스트리

밍 서비스에 있어서 꼭 필요한 일이다.

보통 비디오가 다른 단말에서 호환되는 비트레이트와 포맷으로 저장되어야 하는데, 비트레이트는 비디오를 구성하는 비트가 얼마나 빨리 처리되는지를 나타내는 단위다. 그래서 일반적으로, 비트레이트가 높다는 것은 고해상도를 의미한다.

비디오 인코딩은 다음과 같은 의미를 가지는데,

- 원본 비디오는 용량이 크다. 압축이나 포맷팅을 통해 크기를 줄일 필요가 있다.
- 대부분의 브라우저는 특정 포맷만을 지원한다.
- 네트워크 속도에 따라 저화질 비디오와 고화질 비디오를 다르게 지원할 필요가 있다.
- 모바일의 경우 네트워크 상황이 수시로 변할 수 있기 때문에 수동으로 변경될 필요가 있어, 이를 지원할 소스를 준비해야 한다.

인코딩은 두 부분으로 구성되었다.

- 컨테이너 : 비디오 파일, 오디오, 메타 데이터를 담는 바구니 같은 것이다.
- 코덱 : 비디오 화질은 보존하면서, 파일 크기를 줄일 목적으로 고안된 압축 및 압축 해제 알고리즘이다. H.264 / HEVC 등이 있다.

유형 비순환 그래프(DAG) 모델

비디오 인코딩은 컴퓨터 리소스를 많이 사용하는 작업이다. 그리고 제작자마다 다른 프로세스를 가진다. 각기 다른 프로세스를 지원하기 위해서, 작업을 병렬로 진행하고 적당한 프로세스 추상화를 통해 작업을 커스텀할 수 있도록 지원해야 한다.

예를들면 페이스북은, 스트리밍 비디오 엔진에 유형 비순환 그래프를 도입하여 작업을 단계별로 배열할 수 있도록 하여 작업들이 순차적 또는 병렬적으로 실행되도록 하고 있다. 본 설계안에서도 이와 비슷하게 동작하도록 설계할 것이다.

원본 비디오는 비디오 / 오디오 / 메타데이터로 나뉘어 처리된다. 각 비디오 인코딩시 처리되는 작업들은

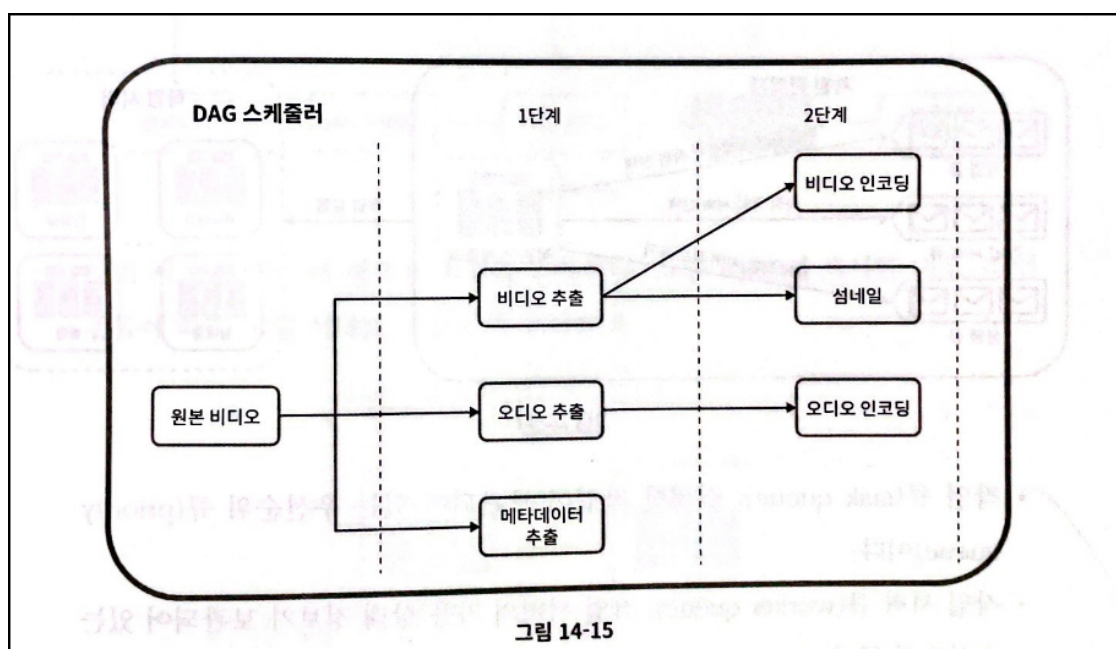
- 검사(inspection) : 비디오 자체의 유효성과 품질을 검사한다.

- 비디오 인코딩 : 비디오를 다양한 해상도 / 코덱 / 비트레이트 조합으로 인코딩하는 작업이다. 인코딩 과정을 거친 비디오들은 서비스에서 지원하는 해상도에 따라 360 / 480 / 720 / 1080... 등으로 변환된다.
- 썸네일 : 사용자가 업로드한 비디오에서 자동 추출된 이미지로 유저들에게 보여줄 대표 이미지가 된다.
- 워터마크 : 비디오 식별정보(누구의 비디오인지) 오버레이 형식으로 띄워준다.



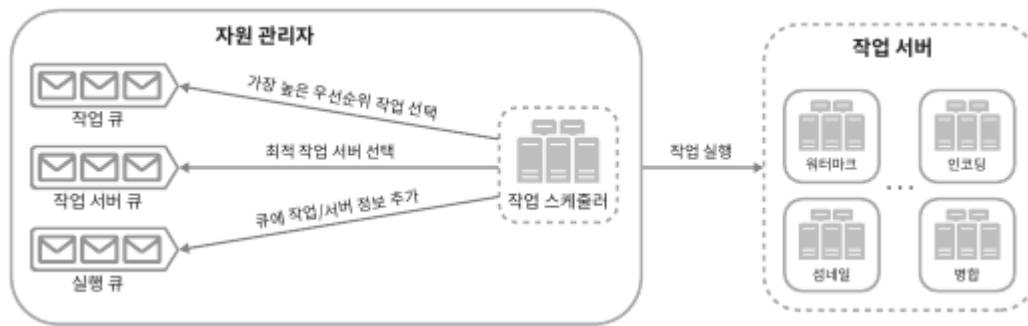
비디오 인코딩 아키텍처는 위와 같은데, 대부분은 텍스트 그대로의 작업을 진행하지만 살펴볼만한 건 DAG 스케줄러다.

DAG(Directed Acyclic Graph)는 **관계와 의존성을 가진 작업들의 집합**이다. 전처리기에서 생성한 DAG의 작업 스케줄링이 바로 DAG 스케줄러다. DAG 스케줄러는 DAG 그래프를 몇 단계로 분할한 다음 그 작업들을 자원 관리자의 작업 큐에 넣는다.



하나의 DAG그래프가 어떻게 작업되는지 보여주는 것이 그림 14-15이다.

첫 번째 단계에서 비디오와 오디오 메타데이터를 분리하고, 두번째 에서는 파일을 인코딩하고 섬네일을 추출하고 오디오 파일 또한 인코딩한다.



자원 관리자는 세 개의 큐와 작업 스케줄러로 자원을 효과적으로 배분한다.

- 작업 큐 : 실행 작업이 보관된 우선순위 큐다.
- 작업 서버 큐 : 작업 서버의 가용 상태 정보가 들어있다.
- 실행 큐 : 현재 실행 중인 작업 및 작업 서버 정보가 들어있다.
- 작업 스케줄러 : 최적의 작업 서버 및 서버 조합을 골라 작업 서버가 작업을 수행하도록 지시한다.

작업 관리자는 다음과 같이 돌아간다.

1. 작업 큐에서 가장 우선순위가 높은 큐를 꺼낸다.
2. 작업 관리자는 해당 작업을 실행하기 적합한 서버를 고른다.
3. 작업 스케줄러는 해당 작업 서버에게 작업 서버를 지시한다.
4. 작업 스케줄러는 해당 작업이 어떤 서버에 할당 됐는지 정보를 실행 큐에 넣는다.
5. 작업 스케줄러는 작업이 완료되면, 해당 작업을 실행 큐에서 제거한다.

그리고 작업 실행 서버는 적합한 작업을 실행하고.. 임시 저장소는 임시적으로 보관할 데이터를 넣어놓고... 이런 작업과정들을 거쳐 인코딩된 비디오가 생성된다.

시스템 최적화

- 속도 최적화 : 비디오 병렬 업로드

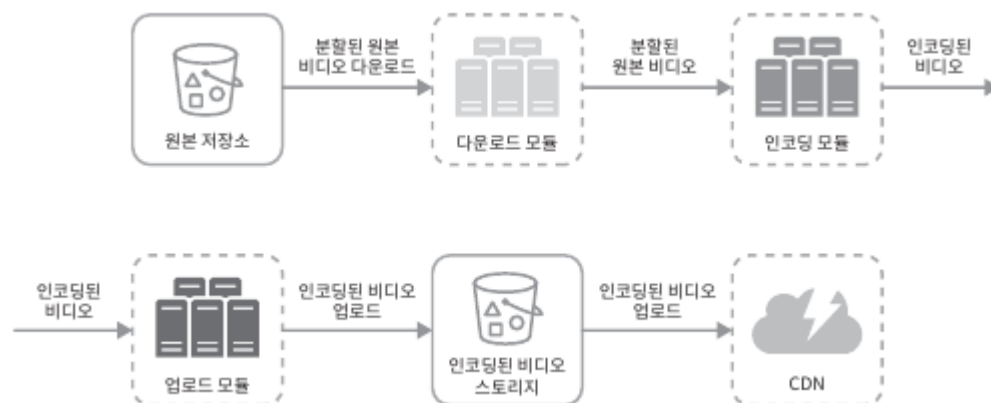
GOP 로 분할하여 비디오를 병렬로 업로드 하면 더 빠르다.. 가장 대표적인 예시로, AWS의 멀티파트 업로드가 있다.

- 속도 최적화 : 업로드 센터를 사용자 근거리에 지정

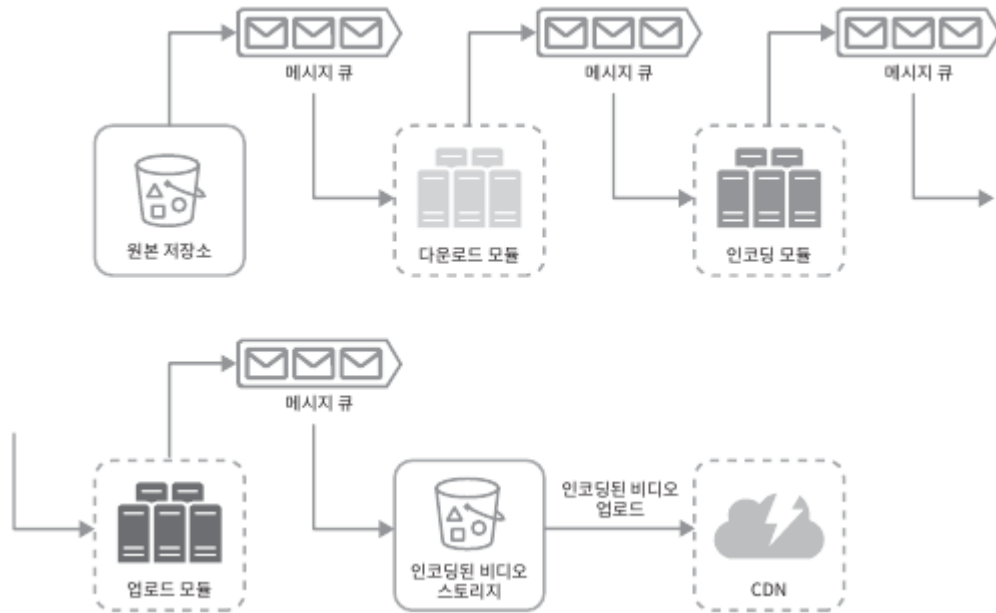
업로드 엣지 센터를 여러곳을 두어 세계 곳곳의 네트워크 최적화를 진행하는 것도 좋다.

- 속도 최적화 : 모든 절차를 병렬화

낮은 응답지연을 달성하기 위해 느슨하게 결합된 시스템을 만들어 병렬성을 높이는 것도 좋다. 다만, 이를 위해서는 설계안을 조금 변경해야 한다.



비디오를 원본 저장소에서 CDN으로 옮기는 절차를 살펴보면 대부분 이전 단계의 결과에 따라 진행되는 긴밀한 결합 단계가 있다. 각 컴포넌트간 의존성이 높을수록 병렬적 처리를 진행하기가 어렵다.



시스템의 결합도를 낮추는데 가장 효과적인 방법 중 하나는 바로 메세지 큐이다. 메세지 큐를 도입하기 전에는

- 큐 도입 전 인코딩 모듈은 다운로드 모듈 작업이 끝나기를 기다려야 했다.
- 큐 도입 후 인코딩 모듈은 다운로드 모듈 작업이 끝나기를 기다릴 필요 없이, 큐에 보관된 이벤트 각각을 병렬로 처리할 수 있게 됐다.

• 안정성 최적화 : 미리 사인된 업로드 URL

허가 받은 사용자만이 올리도록 설정할 수도 있고, 서버가 아닌 유저의 리소스를 사용하여 클라이언트에서 바로 업로드 한다는 점에서 좋다.

• 안정성 최적화 : 비디오 보호

DRM을 통해 디지털 저작권 관리를 할 수도 있다.

혹은 워터마크를 달아.. 내 저작권을 보호할 수도 있다.

• 비용 최적화

기본적으로 동영상 서비스는 비용이 많이 들어가는 서비스다. CDN은 그나마 저렴하지만, 이 CDN 비용마저도 천문학적일 수 있다.

유튜브의 비디오 스트리밍은 룬테일이라고 한다. 인기 있는 건 번번히 재생되는 반면, 인기 없는 건 거의 재생이 안된다고 한다. 이를 통해 최적화를 시도할 수 있다.

1. 인기 비디오는 CDN을 통해 재생하되, 다른 비디오는 비디오 재생서버를 통해 재생한다.
2. 인기 없는 비디오는 인코딩 과정이 필요 없거나 절차를 일부 생략할 수도 있다.
3. CDN을 직접 구축하고 인터넷 서비스 제공자(ISP)와 제휴하여 서비스를 제공한다. 다만, CDN 직접 구축하는 것은 초대형 프로젝트로서.. 초 거대 기업이 아니라면 어려운 개념이다.

오류 처리

재해복구 시스템이 대형 시스템에선 불가피하다.

시스템 오류는 두 가지가 있다.

- 회복 가능 오류 : 특정 비디오 세그먼트를 인코딩 실패 했다면가 하는 오류는 회복 가능한 종류라 재시도 정책을 구성하면 해결된다.
- 회복 불가능 오류 : 포맷이거나 기존 서비스에서 아예 지원하지 않는 포맷이라면 회복 불가능하여 인코딩 서버가 죽어버릴 수 있다. 해당 포맷 작업을 분리하던가 작업을 중단하여 적절 오류를 제공해야 한다.

따라서 적당한 해결 방법을 정리할 수 있다.

- 업로드 오류 ⇒ 재시도 정책 구성
- 비디오 분할 오류 ⇒ 낮은 버전의 클라이언트가 GOP 경계에 따라 비디오를 분할하지 못하는 경우라면 비디오 서버로 전송하고, 해당 서버가 처리하도록 한다.

- 인코딩 오류 ⇒ 재시도 정책 구성
- 전처리 오류 ⇒ DAG 그래프 재생성
- DAG 스케줄러 오류 ⇒ 작업 재배포
- 큐 장애 ⇒ 레플리카 운영
- 작업 서버 장애 ⇒ 다른 서버에서의 작업 시도
- API 서버 장애 ⇒ 신규 요청은 다른 API 서버로 트래픽 이동
- 메타데이터 캐시 서버 장애 ⇒ 데이터가 다중화 되었다면, 다른 노드에서 정보를 가져올 수 있을 것이다.
- 메타데이터 데이터베이스 서버 장애
 - ⇒ 주 서버가 죽었다면 부 서버 가운데 하나를 주 서버로 돌린다.
 - ⇒ 부 서버가 죽었다면 다른 부 서버를 통해 읽기 연산을 처리하고 죽은 서버는 새 것으로 교체

4단계 마무리

추가적으로 다음과 같은 논의를 할 수 있다.


- API 규모 확장성 논의
- 데이터베이스 계층 규모 확장성 확보 ⇒ 다중화와 샤딩
- 라이브 스트리밍 ⇒ 여기서 논의하긴 의미 없는 듯하여 생략
- 비디오 삭제 ⇒ 저작권 위반한 비디오 선정적 비디오 등 삭제 정책을 구성해야 한다.

-
- 이미지 참조

[https://github.com/Meet-Coder-Study/book-system-design-interview/blob/master/14장/\[8주차\]_14장_유튜브 설계_김광훈.md](https://github.com/Meet-Coder-Study/book-system-design-interview/blob/master/14장/[8주차]_14장_유튜브 설계_김광훈.md)

14장. 유튜브 설계

사내 스튜디오 진행하는 가상 면접 사례로 배우는 대규모 시스템 설계 기초 14장. 유튜브 설계 에 대해 공부한 내용을 정리해보려고 한다.

 <https://velog.io/@ony/14장.-유튜브-설계>



14장. 유튜브 설계

가상 면접 사례로 배우는 대규모 설계 시스템 기초