[15장] 구글 드라이브설계

가상 면접 사례로 배우는 대규모 시스템 설계 기초

이민석 / unchaptered

https://inblog.ai/monthly-cs
https://github.com/monthly-cs/2024-03-system-design-interview-1

요구사항 파악 및 정리

파악

- 파일 업로드/다운로드
  - 파일 업로드 : 드래그 앤 드랍 방식 지원
- 파일 동기화 : 여러 단말 간 파일 동기화 및 파일 공유
- 파일 갱신 이력 조회 : 파일 편집 및 새 공유 시
- 알림 기능
- 앱/웹 지원
- 성능 요구사항
  - 안정성 필요
  - 빠른 동기화 속도
  - 네트워크 대역폭 및 규모 확장성
  - 높은 가용성

개략적 규모 추정

1. 가입 사용자 50,000,000명(천만명)
2. 일간 능동 사용자(DAU)는 10,000,000명(천만명)
3. 모든 사용자에게 10GB의 무료 저장공간
4. 매일 각 사용자가 2개의 파일을 업로드하며 개당 500KB
5. 읽기 및 쓰기 비율은 1:1
6. 필요한 저장공간 = 50,000,000 × 10GB = 50PB
7. 업로드 API QPS = 10,000,000 × 2회 ÷ 24시간 ÷ 3600초 = 약 240
8. 최대 QPS = QPS × 480

단일 서버

[서버 구조]

1. 웹서버(WebServer)와 데이터베이스(MySQL) 그리고 스토리지
2. 스토리지에는 계층적 구조를 사용하여 데이터를 저장

[필요한 API]

1. 파일 업로드 API
2. 파일 다운로드 API
3. 파일 갱신 히스토리 API

단일 서버의 제약 - 1

용량이 부족할 때

1. 전략을 세우고 데이터를 샤딩하여 여러 서버에 **분산 저장**
2. **Amazon S3를 사용하면 확장성, 가용성, 보안 및 성능 부분을 몇가지 설정으로 지원받을 수 있음**
    a. 동일 리전 다중화
    b. 멀티 리전 다중화 : 굿

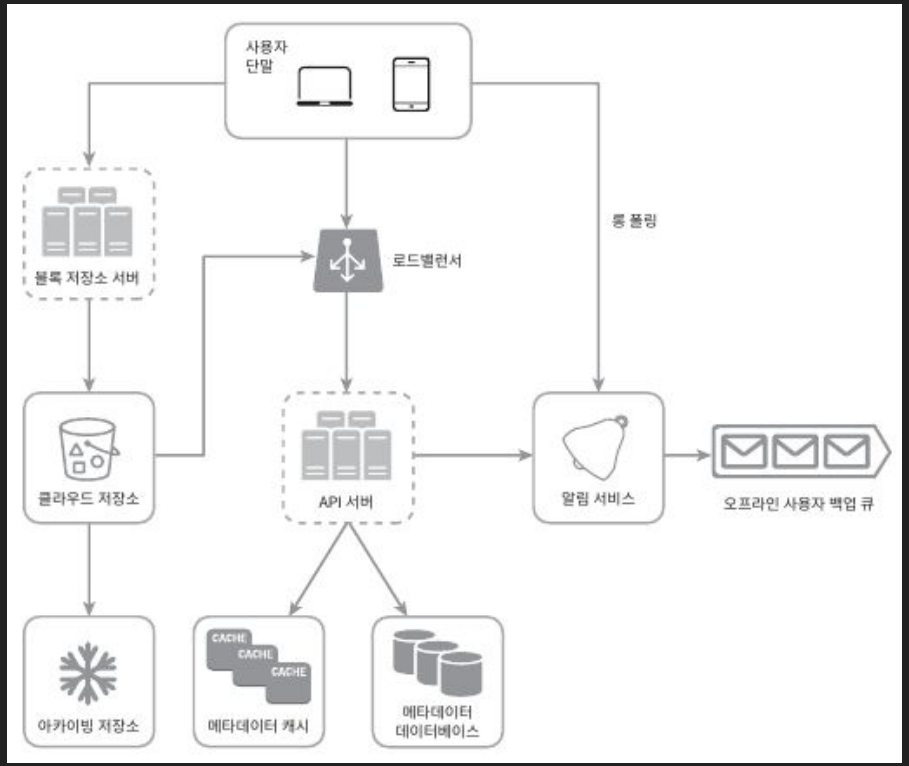
단일 서버의 제약 - 2

- **로드 밸런서** : N대의 서버로 부하 분산
- 웹 서버 : N대의 서버로서 수평적 확장 가능
- 메타데이터 데이터베이스 : Amazon S3와 분리하여 다중화, 샤딩 정책을 적용하여 SPoF를 회피
- 파일 저장소 : Amazon S3를 사용하며, 2개 이상의 리전에 데이터를 다중화

동기화 충돌

- *약간 Git Conflict와 유사한 것 같기도…*
- **두 명 이상의 사용자가 같은 파일이나 폴더를 동시에 업데이트 시도**
  - 먼저 처리된 변경은 성공
  - 나중에 처리된 변경은 충돌(Conflict)
- 충돌이 발생한 경우, 사용자가 두 파일을 합칠지(Merge) 혹은 대체할지(Replace) 결정

개략적인 설계안



https://velog.io/@ony/15%EC%9E%A5.-%EA%B5%AC%EA%B8%80-%EB%93%9C%EB%9D%BC%EC%9D%B4%EB%B8%8C-%EC%84%A4%EA%B3%84

- 사용자 단말
- 블록 저장소 서버
  - 파일을 여러 개의 블록으로 나누어 저장하는 부분
  - *왜 필요한거지..?*
- 클라우드 저장소
- 아카이빙 저장소
- API 서버
- 알림 서비스
  - 이벤트의 발행 및 구독
- 오프라인 사용자 백업 큐
  - 비접속 사용자를 위한 알림 저장

3단계 | 상세 설계

블록 저장소 서버

큰 파일의 갱신은 전체 파일 전송 시, **대량의 네트워크 대역폭 소비**

이를 개선하기 위해서 아래의 최적화 기법 혹은 동적 절차 사용

- 델타 동기화(**Delta Sync**) : 파일이 수정된 블록만 동기화
- 압축 : 블록 단위로 압축 시 데이터 크기를 줄임, 파일 유형에 따라서 압축 알고리즘 결정

- 동적 절차 : 파일을 작은 블록으로 분할 → 각 블록을 압축 → 암호화 → (델타 동기화 -> 갱신 블록만) -> 저장소 전송

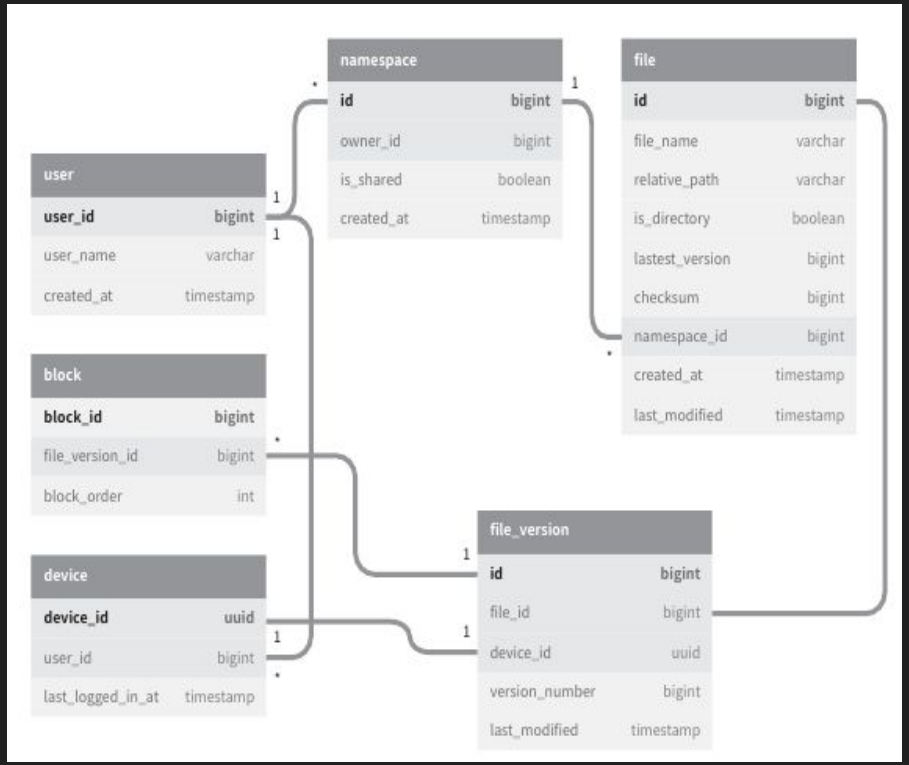
높은 일관성 요구사항, 강한 일관성

강한 일관성 모델 지원 필요

1. 메모리 캐시: 최종 일관성 모델
   a. 캐시에 보관된 사본과 데이터베이스 원보닝 일치해야함
   b. 데이터에 보관된 원본이 변경 시, 일부 캐시를 무효화

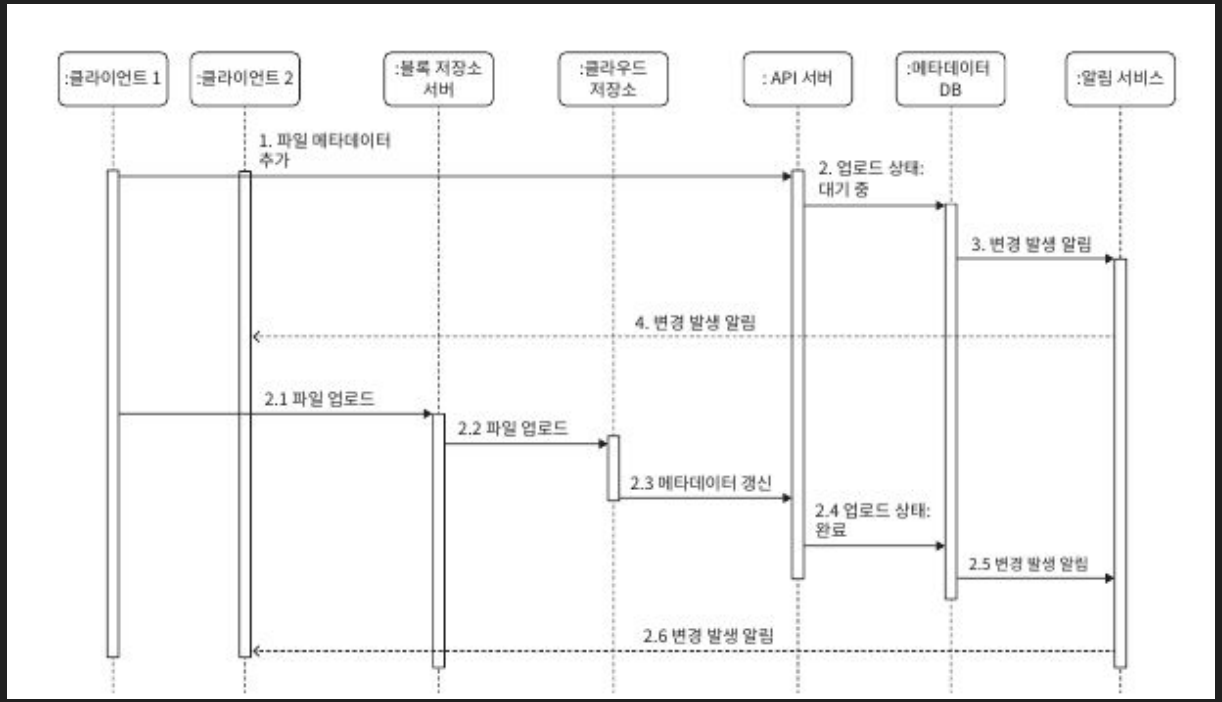
2. 관계형 데이터베이스 (ACID 보장 : 데이터베이스의 트랜잭션 안정성 보장)

메타데이터 데이터베이스 설계



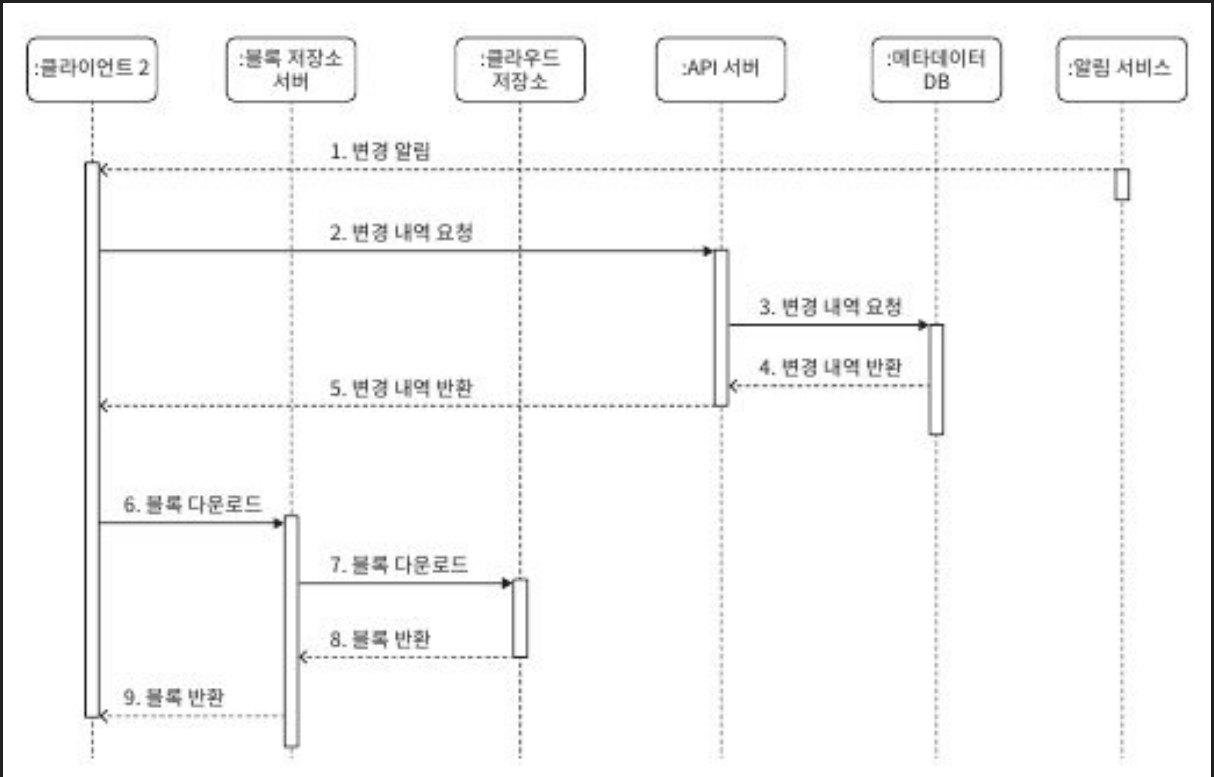
https://velog.io/@ony/15%EC%9E%A5.-%EA%B5%AC%EA%B8%80-%EB%93%9C%EB%9D%BC%EC%9D%B4%EB%B8%8C-%EC%84%A4%EA%B3%84

업로드 절차



https://velog.io/@ony/15%EC%9E%A5.-%EA%B5%AC%EA%B8%80-%EB%93%9C%EB%9D%BC%EC%9D%B4%EB%B8%8C-%EC%84%A4%EA%B3%84

다운로드 절차



https://velog.io/@ony/15%EC%9E%A5.-%EA%B5%AC%EA%B8%80-%EB%93%9C%EB%9D%BC%EC%9D%B4%EB%B8%8C-%EC%84%A4%EA%B3%84

비디오 트랜스코딩 아키텍처

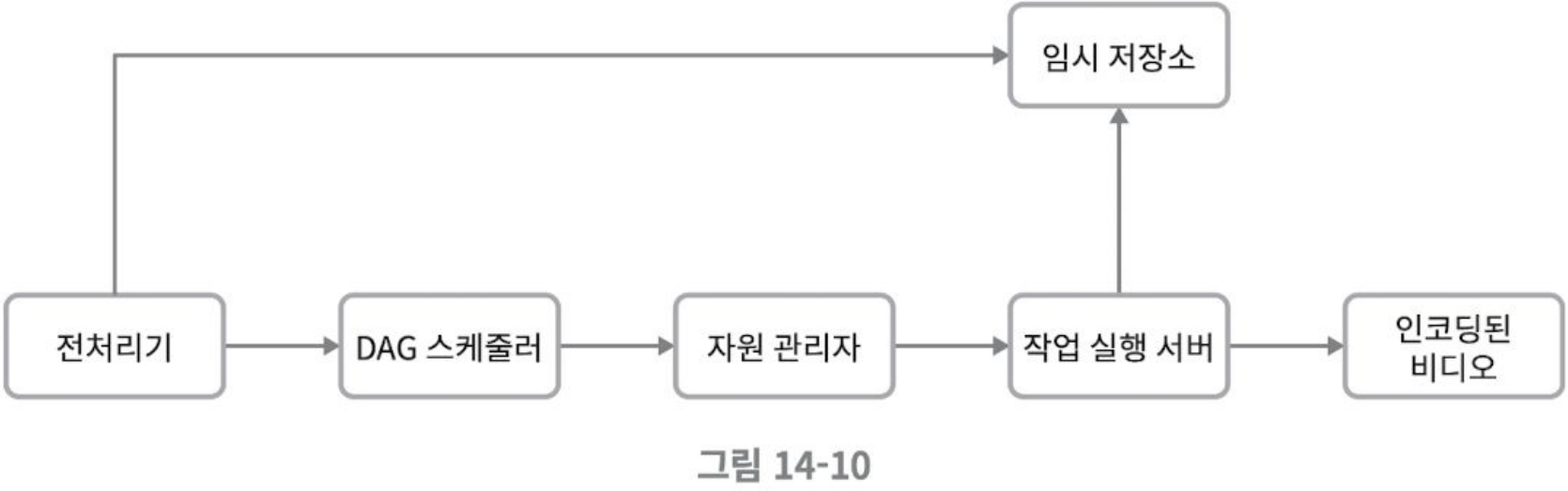


그림 14-10

https://velog.io/@nnagman/TIL%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-14%EC%9E%A5-%EC%9C%A0%ED%8A%9C%EB%B8%8C-%EC%84%A4%EA%B3%84#directed-acyclic-graphdag

3단계 | 상세 설계

알림 서비스

파일 일관성 유지를 위해, **파일 수정 시 알림이 필요함**

1. **롱 폴링(Long Polling)** : 이번 설계안에서 선택
2. 웹 소켓(Web Socket)

왜 롱폴링인가?

1. 양방향 통신 불필요
2. 특정 파일에 대한 변경을 감지하면 해당 연결을 중단
3. 메타데이터 데이터베이스로부터 파일의 최신내역 다운로드
4. 다운로드 완료 혹은 연결 타임아웃 발생 시에는 롱 폴링 연결을 복원

3단계 | 상세 설계

저장소 공간 절약

각 블록의 해시값을 활용한 중복 제거

지능적 백업 전략 필요

- 오래된 파일은 아카이빙
- 파일 버전에 상한선을 저장
- 중요한 버전만 저장..?

3단계 | 상세 설계

장애 처리

스토리지 장애 : 멀티 리전 다중화를 통해서 보관 중이기에, 다른 리전의 데이터로 복구

블록 스토리지 서버 장애 : 한 블록 스토리지 서버의 장애가 발생하면, 다른 서버 이를 이어서 받아서 작업 진행 필요

데이터베이스 장애 : 주/부 데이터베이스를 사용할 것, 장애 발 생시 대체

알림 서비스 장애 : 1대의 알림 서비스로는 장애를 복구하는데 오래걸리기 때문에, *N개의 알림서버로 다중화하는 것은 어떤지…*

오프라인 사용자 백업 큐 장애 : 기본적으로 다중화는 필수, 이후 백업 큐로 구독관계를 재설정

추가 논의사항

- 블록 서버를 거치지 않고 클라우드를 사용 시
  - 여러 플랫폼(OS)에 이를 구현하는데 들어가는 리소스
  - 여러 플랫폼/클라이언트이 해킹 당할 가능성

- 접속 상태를 관리하는 로직의 재사용성
  - 여러 서비스에서 재활용이 가능해보임

감사합니다.