가상 면접 사례로 배우는

대규모 시스템 설계 기초

이정민

15. 구글 드라이브 설계

1단계) 문제 이해 및 설계 범위 확정

■ 요구사항 파악

- 중요한 기능은? 파일 업로드/다운로드, 파일 동기화, 알림 기능 / 지원 대상? 모바일 앱, 모바일 웹
- 암호화 지원? 필요함 / 파일 크기 제한? 10GB 제한 / 사용자 수? 일간 능동 사용자 기준 천만 명

■ 요구사항 정리

- 기능적 요구사항
  - 파일 추가 (파일을 구글 드라이브 안으로 drag-and-drop), 파일 다운로드, 여러 단말 간 파일 동기화, 파일 공유
  - 파일 갱신 이력 조회, 파일이 편집되거나 새로 공유되었을 때 알림 표시 편집 및 협업 기능 제외
- 비기능 요구사항
  - 안정성(데이터 손실X), 빠른 동기화 속도, 네트워크 대역폭, 규모 확장성(많은 양의 트래픽 처리), 높은 가용성

1단계) 문제 이해 및 설계 범위 확정

■ 개략적 추정치

- ○ 가정
  - 가입 사용자 : 5천만명, DAU : 천만명
  - 모든 사용자에게 10GB 무료 저장공간 할당
  - 매일 각 사용자는 평균 2개의 파일을 업로드. 평균 파일 크기는 500KB
  - 읽기:쓰기 비율은 1:1명
- ○ 추정치
  - 필요한 저장공간 총량 = 500PB (5천만 사용자 * 10GB)
  - 업로드 API QPS = 약 240QPS (1천만 사용자 * 2회 업로드/24시간/3600초)
  - 최대 QPS = 약 480QPS (QPS * 2)

2단계) 개략적 설계안 제시 및 동의 구하기

- 서버 한대로 시작
  - 파일을 올리고 다운로드하는 과정을 처리할 웹 서버 (Apache Web Server)
  - 사용자 데이터, 로그인 정보, 파일 정보 등을 보관할 데이터베이스 (MySQL)
  - 파일을 저장 할 저장소 시스템. 파일 저장을 위해 1TB 공간 사용
    - drive 디렉토리 하위에 namespace가 되는 하위 디렉토리 (사용자 별 파일 보관)

2단계) 개략적 설계안 제시 및 동의 구하기

■ API

- 파일 업로드 API : (예) https://api.example.cm/files/upload?uploadType=resumable
  - 단순 업로드(파일 크기가 작을 때), 이어 올리기(파일 사이즈가 크고, 중단 될 가능성이 큰 경우)
  - 파라미터 : uploadType - 업로드 유형, data - 업로드 할 로컬 파일
- 파일 다운로드 API : (예) https://api.example.com/files/download
  - 파라미터 : path - 다운로드할 파일의 경로
- 파일 갱신 히스토리 API : (예) https://api.example.com/files/list_revisions
  - 파라미터 : path - 갱신 히스토리를 가져올 파일의 경로, limit - 히스토리 길이의 최대치

2단계) 개략적 설계안 제시 및 동의 구하기

- 한 대 서버의 제약 극복
  - 파일 시스템 용량 제한 문제 : 데이터를 샤딩하여 여러서버에 분산 저장
  - 아마존 S3(Simple Storage Service, 규모확장성, 가용성, 보안 성능 제공 객체 저장소 서비스)
    - 동일 지역 내 다중화 및 여러 지역에 걸친 다중화 지원

2단계) 개략적 설계안 제시 및 동의 구하기

- 개선 방안 : p. 285 그림 15-7
  - 로드밸런서 : 네트워크 트래픽 분산
  - 웹 서버 : 수평적 확장
  - 메타데이터 데이터베이스 : 파일저장 서버와 분리하여 다중화, 샤딩 정책 적용(SOF 회피)
  - 파일 저장소 : S3를 파일 저장소로 사용, 두개 이상의 저역에 데이터를 다중화

2단계) 개략적 설계안 제시 및 동의 구하기

■ 동기화 충돌

- 두명 이상의 사용자가 같은 파일이나 폴더를 동시에 업데이트 하려는 경우
- 먼저 처리되는 변경은 성공, 나중에 처리된 변경은 충돌이 발생한 것으로 처리
- 충돌이 발생한 시점에 시스템에는 같은 파일의 두가지 버전이 존재 (충돌 발생 사용자의 로컬 버전)
  - 사용자가 두 파일을 합칠지 대체할 지 판단하여 결정

2단계) 개략적 설계안 제시 및 동의 구하기

- 개략적 설계안 : p. 287 그림15-10
  - 사용자 단말
  - 블록 저장소 서버(block server) : 파일 블록을 클라우드 저장소에 업로드하는 서버
    - 클라우드 환경에서 데이터 파일을 저장하는 기술. 파일을 여러개의 블록으로 나누어 저장(해시값 할당)
  - 클라우드 저장소 : 파일은 블록 단위로 나눠져 저장
  - 아카이빙 저장소 : 오랫동안 사용되지 않은 비활성(inactive) 데이터 저장
  - 알림 서비스(파일 최신 상태 알림), 오프라인 사용자 백업 큐(클라이언트 접속 시 동기화)

3단계) 상세 설계

■ 블록 저장소 서버

- 정기적으로 큰 파일의 갱신이 일어날 때 마다 전체 파일 전송 시 대량의 네트워크 대역폭 소비
- 최적화 기법
  - 델타 동기화(delta sync) : 파일 수정 시 전체 파일 대신 수정이 일어난 블록만 동기화
  - 압축 : 블록 단위로 압축 시 데이터 크기를 줄임. 파일 유형에 따라 압축 알고리즘 결정
- 동작 절차
  - 파일을 작은 블록으로 분할 - 각 블록을 압축 - 암호화 - (델타 동기화 전략 - 갱신된 블록만) - 저장소로 전송

3단계) 상세 설계

■ 높은 일관성 요구사항

- 강한 일관성 모델(동일 파일이 단말이나 사용자에 따라 다르게 보이는 것을 허용하지 않음) 지원 필요
- 메모리 캐시(보통 최종 일관성 모델)
  - 캐시에 보관된 사본과 데이터베이스의 원본이 일치 해야 함
  - 데이터베이스에 보관된 원본이 변경되면 캐시의 사본을 무효화해야 함
- 관계형 데이터베이스(ACID 보장)
  - NoSQL 데이터베이스는 동기화 로직을 애플리케이션에서 보장해야함

3단계) 상세 설계

- 메타데이터 데이터베이스
  - 스키마 설계안 : p. 291 그림 15-13

3단계) 상세 설계

■ 업로드 절차

○ 동작 절차 : p. 291 그림 15-14

3단계) 상세 설계

■ 다운로드 절차

- 동작 절차 : p. 294 그림 15-15
- 네트워크에 연결된 상태가 아닐 경우 데이터는 캐시에 보관. 클라이언트 접속시 처리

3단계) 상세 설계

■ 알림 서비스

- 파일 일관성 유지
  - 클라이언트는 로컬에서 파일 수정을 감지하는 순간 다른 클라이언트에 알려서 충돌 가능성을 줄여야 함
- 사용 프로토콜 : 롱 폴링(long polling)
  - 양방향 통신 불필요
  - 특정 파일에 대한 변경을 감지하면 해당 연결을 끊음. 메타데이터 서버로부터 파일의 최신내역을 다운로드
  - 다운로드가 끝났거나 연결 타임아웃 시 새 요청을 통해 롱 폴링 연결을 복원하고 유지

3단계) 상세 설계

■ 저장소 공간 절약

- 파일의 다수 버전을 다수의 데이터 센터에서 모두 자주 백업할 경우 저장 용량이 빨린 소진됨
- 공간 절약 기법
  - 중복 제거(de-dupe) : 중복(해시값 비교)된 파일 블록을 계정 차원에서 제거
  - 지능적 백업 전략 : 한도 설정(보관 파일 갯수 제한), 중요 버전만 보관(불필요 버전 사본 제거)
  - 자주 사용되지 않는 데이터는 아카이빙 저장소로 이관 (몇달~수년간 이용되지 않는 데이터)

3단계) 상세 설계

■ 장애 처리

- 로드밸런서 장애 / 블록 저장소 서버 장애 / 클라우드 저장소 장애
- API 서버 장애 / 메타데이터 캐시 장애 / 메타데이터 데이터베이스 장애
- 오프라인 사용자 백업 장애
- 알림 서비스 장애
  - 한대의 알림 서버 장애 발생 시 대량의 사용자의 롱 폴링 연결을 재생성해야 함 (롱 폴링 복구는 상대적으로 느림)

4단계) 마무리

■ 추가 논의사항

- 파일을 블록 저장소 서버를 거치지 않고 클라우드 저장소에 직접 업로드한다면?
  - 업로드 시간은 빨라 질 수 있으나
  - 분할, 압축, 암호화 로직을 클라이언트에 두어야 하므로 플랫폼(iOS, 안드로이드, 웹 등) 별 구현이 필요함
  - 클라이언트가 해킹 당할 가능성이 있어 암호화 로직을 클라이언트에 두는 것은 적절치 않음
- 접속상태를 관리하는 로직을 별도 서비스로 옮기는 것 (다른 서비스들과 공유 가능)