가상 면접 사례로 배우는

대규모 시스템 설계 기초

이정민

12. 채팅 시스템 설계

1단계) 문제 이해 및 설계 범위 확정

■ 요구사항 질의/응답

- 지원 내용(1:1 채팅 or 그룹 채팅) 둘 다 지원 / 대상(모바일 앱 or 웹)? 둘 다 지원
- 트래픽 규모? 일별 능동 사용자 수 (DAU: Daily Active User) 기준 5천만 명
- 그룹 채팅 인원 제한? 최대 100명 참가 / 메시지 길이 제한? 100,000자
- 중요 기능(첨부파일 등)? 1:1 채팅, 그룹 채팅, 사용자 접속상태 표시, 텍스트 메시지만
- 종단 간 암호화 지원? 없음 / 채팅 이력 보관 기간? 영원히

■ 요구사항

- 응답지연이 낮은 1:! 채팅, 최대 100명까지 그룹 채팅, 사용자 접속상태 표시
- 다양한 단말 지원, 하나의 계정으로 여러 단말 동시 접속 지원, 푸시 알림, 5천만 DAU

2단계) 개략적 설계안 제시 및 동의 구하기

■ 제공 기능

- 클라이언트들로부터 메시지 수신
- 메시지 수신자(recipient) 결정 및 전달
- 수신자가 접속(online) 상태가 아닌 경우에는 접속할 때까지 해당 메시지 보관
- 사용 프로토콜
  - 송신 : HTTP (keep-alive 헤더 사용 - 클라언트와 서버 간의 연결을 유지, TCP 접속 핸드세이크 횟수 줄임)
  - 수신 : 폴링, 롱 폴링, 웹 소켓 등 사용

    (HTTP는 클라이언트가 연결을 만드는 프로토콜로서 서버에서 클라이언트로 메시지 전송 시 사용이 힘듬)

2단계) 개략적 설계안 제시 및 동의 구하기

■ 폴링

- 클라이언트가 주기적으로 서버에게 새 메시지가 있느냐고 물어보는 방법
- 자주 할 수록 비용 증가
- 답해줄 메시지가 없는 경우 서버 자원이 불필요하게 낭비됨

2단계) 개략적 설계안 제시 및 동의 구하기

■ 롱 폴링

- 폴링의 단점인 비효율성을 개선하기 위한 기법
- 클라이언트는 새 메시지가 반환되거나 타임아웃 될 때까지 연결을 유지함
- 클라이언트는 새 메시지를 받으면 기존 연결을 종료하고 서버에 새로운 요청을 보내 재반복
- 단점
  - 메시지 송신/수신 클라이언트가 동일 채팅 서버가 아닐 수 있음
  - 서버에서 클라이언트 연결이 해제되었는지 알 수 있는 좋은 방법이 없음
  - 여전히 비효율적임 (메시지를 많이 받지 않는 경우도 타임아웃이 일어날 때 마다 주기적으로 서버 재접속함)

2단계) 개략적 설계안 제시 및 동의 구하기

■ 웹소켓

- 서버가 클라이언트에게 비동기 메시지를 보낼 때 가장 널리 사용하는 기술
- 클라이언트가 연결을 시작하고 한번 맺어진 연결은 항구적이며 양방향임
- 처음에는 HTTP 연결이고 핸드셰이크 절차를 거쳐 웹소켓 연결로 업그레이드됨
- 연결이 맺어지면 서버는 클라이언트에게 비동기적으로 메시지를 전송할 수 있음
- 웹소켓은 방화벽 환경에서도 잘 동작함(HTTP 80, HTTPS 443 기본포트 사용)
- 메시지 송/수신 시 웹소켓 사용하면 설계/구현이 단순하고 직관적임
- 단, 웹소켓 연결은 항구적으로 유지되어야 하기 때문에 서버특에서 연결관리를 효율적으로 해야 함

2단계) 개략적 설계안 제시 및 동의 구하기

- 개략적 설계안
  - 무상태 서비스 : 로그인, 회원가입, 사용자 프로파일 표시 등 처리
  - 상태 유지 서비스 : 채팅 서비스 (채팅 서버와 각 클라이언트가 독립적인 네트워크 연결을 유지)
  - 제3자 서비스 연동 : 푸시 알림
  - 규모 확장성 : p. 207) 그림 12-8
  - 저장소
    - 일반적인 데이터 : 사용자 프로파일, 설정, 친구 목록 - 안정성을 보장하는 관계형 데이터베이스에 보관
    - 채팅 이력 데이터 : 데이터 양이 엄청남. 최근 메시지만 빈번하게 사용. 검색 기능 및 무작위 데이터 접근 할 수 있음. 1:1 채팅의 경우 읽기:쓰기 비율은 1:1 => 키-값 저장소 추천 (수평적 규모확장 쉬움, 데이터 접근 지연시간이 낮음. RDB 문제, 레퍼런스 많음)

2단계) 개략적 설계안 제시 및 동의 구하기

■ 데이터 모델

- 1:1 채팅을 위한 메시지 테이블
  - message : message_id(pk, bigint), message_from(bigint), message_to(bigint), content(text), created_at(timestamp)
- 그룹 채팅을 위한 메시지 테이블
  - group_message : channel_id(pk, big_int), message_id(pk, big_int), message_to(bigint), content(text), create_at(timestamp)
- 메시지 ID
  - message_id 값은 고유해야 함. id 값은 정렬 가능해야 하며 시간 순서와 일치해야 함
  - RDBMS의 경우 auto_increment 사용 가능하나, NoSQL에서는 미지원함
  - 전역적 시퀀스 번호 생성기(제7장) 사용 or 지역적 시퀀스 번호 생성기 사용

3단계) 상세 설계

■ 메시지 흐름

- 1:1 채팅 메시지 처리 흐름 : p. 212) 그림 12-22
- 여러 단말 사이의 메시지 동기화 : p. 213) 그림 12-23
- 소규모 그룹 채팅에서의 메시지 흐름
  - 사용자 별 할당 된 메시지 동기화 큐(메시지 수신함) 사용. 자신에게 할당된 큐만 확인하므로 처리가 간단함
  - 그룹이 크지 않은 경우 메시지를 수신자별로 큐에 복사하는 비용이 크게 문제가 되지 않음

3단계) 상세 설계

■ 접속상태 표시

- 접속상태 서버를 통해 사용자의 상태를 관리
- 사용자 로그인 : 클라이언트와 웹소켓 연결이 맺어지면 사용자는 접속중으로 표시 (last_active_at)
- 로그아웃 : 로그아웃 API 사용. 사용자 상태가 online->offline
- 접속 장애 : 클라이언트로부터 heartbeat를 받아 일정 시간 미수신 시 오프라인 상태로 변경
- 상태정보의 전송 : 각 친구 관계마다 Pub/Sub 채널 사용(크기가 작을 때 효과적). 특정 시점에만 상태정보를 읽게 하거나, 수동으로 친구리스트의 접속상태 정보를 갱신하도록 유도

4단계) 마무리

■ 추가 논의사항

- 사진이나 비디오 등의 미디어 지원 : 압축 방식, 클라우드 저장소, 섬네일 생성 등
- 종단 간 암호화 : 메시지 발신인과 수신자 이외에 아무도 메시지 내용을 볼 수 없는 기법
- 캐시 : 클라이언트에 이미 읽은 메시지를 캐싱
- 로딩 속도 개선 : 사용자의 데이터, 채널 등을 지역적으로 분산하는 네트워크 구축 등
- 오류 처리
  - 채팅 서버 오류 : 채팅 서버 다운 시 클라이언트에게 새로운 서버를 배정하고 다시 접속할 수 있도록 서비스
  - 메시지 재전송 : 재시도(retry), 큐 등 메시지의 안정적 전송을 보장하는 기법