

7주차

12장. 채팅 시스템 설계

1단계 - 문제 이해 및 설계 범위 확정

면접관이 원하는 앱이 정확히 무엇인지 - 적어도 설계 대상이 1:1 채팅앱인지, 그룹 채팅 앱인지 정도는 - 알아내야 함

책에서 설계할 채팅 시스템

페이스북 메신저와 유사한 채팅 앱 설계

- 응답 지연이 낮은 일대일 채팅 기능
- 최대 100명까지 참여할 수 있는 그룹 채팅 기능
- 사용자의 접속 상태 표시 기능
- 다양한 단말 지원 → 하나의 계정으로 여러 단말 여러 동시 접속 지원
- 푸시 알림
- 5천만 DAU 처리 지원

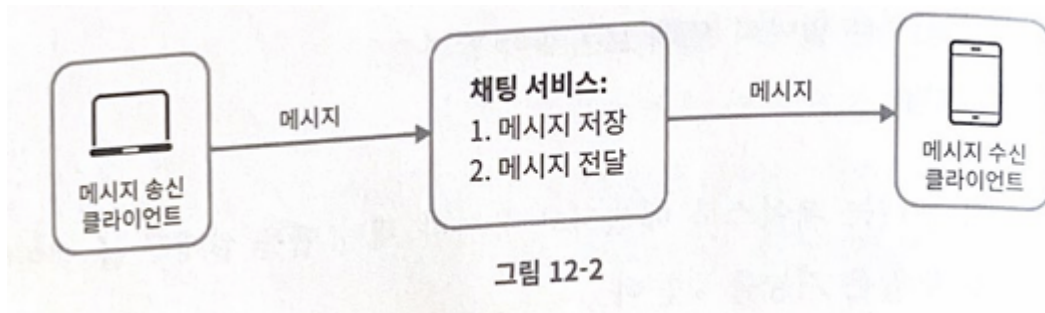
2단계 - 개략적 설계안 제시 및 동의 구하기

채팅 시스템의 경우 클라이언트는 모바일 앱 혹은 웹 어플리케이션이 됨

클라이언트는 서로 직접 통신하지 않고, 채팅 서비스와 통신을 진행 → 이때, 채팅 서비스는 아래의 기능을 제공해야 함

- 클라이언트들로부터 메시지 수신
- 메시지 수신자 결정 및 전달
- 수신자가 접속 상태가 아닌 경우에는 접속할 때까지 해당 메시지 보관

채팅을 시작하려는 클라이언트는 네트워크 통신 프로토콜을 사용하여 서비스에 접속 → 어떤 통신 프로토콜을 사용할지 역시 중요한 문제



1. 메시지 송신 클라이언트는 채팅 서비스에 HTTP프로토콜로 연결
2. 메시지 송신 클라이언트가 채팅 서비스에 수신 클라이언트에게 전달할 메시지를 전송 및 수신자에게 해당 메시지를 전달하라고 알림
 - a. 이때, 채팅 서비스와 Keep-alive 헤더를 사용하면 클라이언트와 서버 사이의 연결을 끊지 않고 계속 유지할 수 있으므로 효율적 (+ 핸드셰이크 횟수 줄이기도 가능)

단, HTTP는 클라이언트가 연결을 만드는 프로토콜이고, 서버에서 클라이언트로 임의의 시점에 메시지를 보내는 데는 쉽게 쓰일 수 없음 → 폴링, 롱 폴링, 웹 소켓 등 서버가 연결을 만드는 것처럼 동작할 수 있도록 하는 기법 사용

폴링

클라이언트가 주기적으로 서버에게 새 메시지가 있냐고 물어보는 방법

폴링 비용은 폴링을 자주하면 할 수록 오름 → 답해줄 메시지가 없는 경우, 서버 자원이 불필요하게 낭비된다는 문제 발생

롱 폴링

폴링의 비효율적인 부분을 보완하기 위한 방법. 클라이언트는 새 메시지가 반환되거나 타임아웃 될 때까지 연결을 유지, 클라이언트는 새 메시지를 받으면 기존 연결을 종료하고 서버에 새로운 요청을 보내 모든 절차를 다시 시작함

롱 폴링 약점

- 메시지를 보내는 클라이언트와 수신하는 클라이언트가 같은 채팅 서버에 접속하게 되지 않을 수도 있음
- 서버 입장에서는 클라이언트가 연결을 해제했는지 아닌지 알 좋은 방법이 없음
- 여전히 비효율적임
 - 메시지를 받지 않는 클라이언트도 타임아웃이 일어날 때마다 주기적으로 서버에 다시 접속할 것

웹 소켓

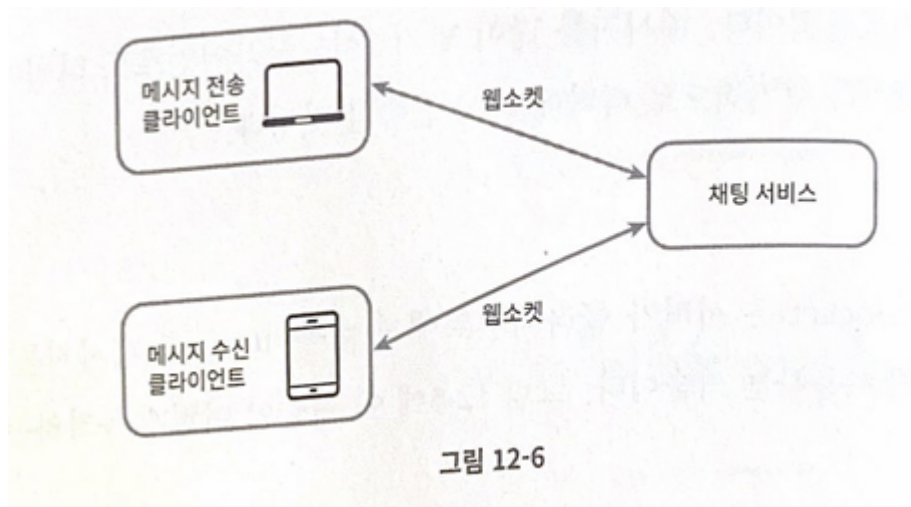
서버가 클라이언트에 비동기 메시지를 보낼 때 가장 널리 사용하는 기술.

웹 소켓 연결은 클라이언트가 시작 → 한번 맺어진 연결은 항구적이며 양방향

- 연결은 처음에는 HTTP 연결이지만 특정 핸드셰이크 절차를 거쳐 웹소켓 연결로 업그레이드 됨
- 항구적 연결이 만들어지고 나면 서버는 클라이언트에게 비동기적으로 메시지를 전송할 수 있음

웹 소켓은 일반적으로 방화벽이 있는 상황에서도 잘 동작함 → 80번, 443번처럼 HTTP나 HTTPS가 사용하는 포트 번호를 그대로 사용하기 때문

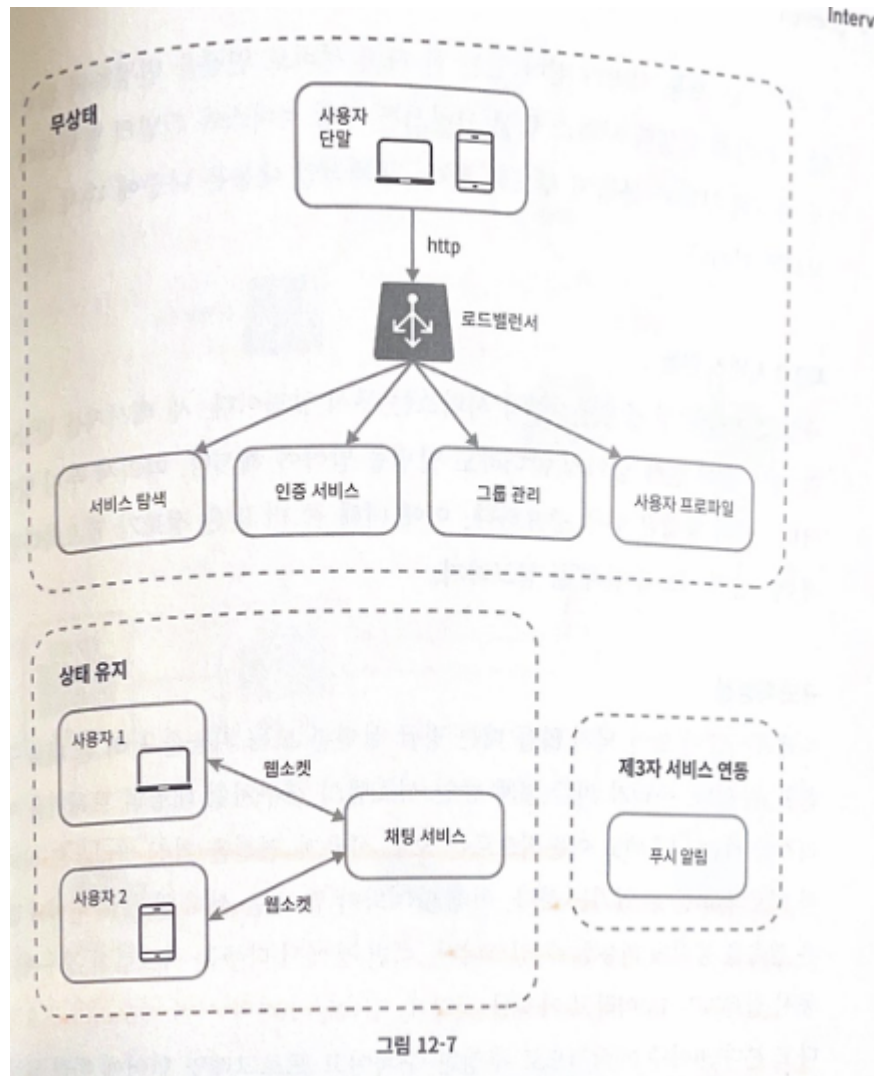
웹 소켓은 양방향 메시지 전송까지 가능하게 하므로 HTTP보다 선호됨



웹 소켓을 이용하면 메시지를 보낼 때나 받을 때 동일한 프로토콜을 사용할 수 있으므로 설계 뿐만 아니라 구현도 단순하고 직관적

단, 웹소켓 연결을 항구적으로 유지되어야 하므로 서버 측에서 연결 관리를 효율적으로 해야 함

개략적 설계안



채팅 시스템

- 무상태 서비스
- 상태 유지 서비스
- 제3자 서비스 연동

1) 무상태 서비스

이 설계안에서 무상태 서비스는 로그인, 회원 가입, 사용자 프로필 표시 등을 처리하는 전통적인 요청/응답 서비스가 됨 → 웹사이트와 앱이 보편적으로 제공하는 기능

- 무상태 서비스는 로드밸런서 뒤에 위치함
 - 로드밸런서: 요청을 그 경로에 맞는 서비스로 정확하게 전달하기

- 로드밸런서의 뒤에 오는 서비스는 모놀리틱 서비스일 수도 있고, 마이크로서비스일 수도 있음
- 서비스 탐색 서비스: 클라이언트가 접속할 채팅 서버의 DNS 호스트명을 클라이언트에게 알려주는 역할

2) 상태 유지 서비스

각 클라이언트가 채팅 서버와 독립적인 네트워크 연결을 유지해야 하므로 상태 유지가 필요한 서비스는 채팅 서비스가 됨

클라이언트는 보통 서버가 살아있는 한 다른 서버로 연결을 변경하지 않음 → 서비스 탐색 서비스는 채팅서비스와 긴밀히 협력, 특정 서버에 부하가 몰리지 않도록 함

3) 제3자 연동 서비스

채팅 앱에서 가장 중요한 제3자 연동 서비스 = 푸시 알림

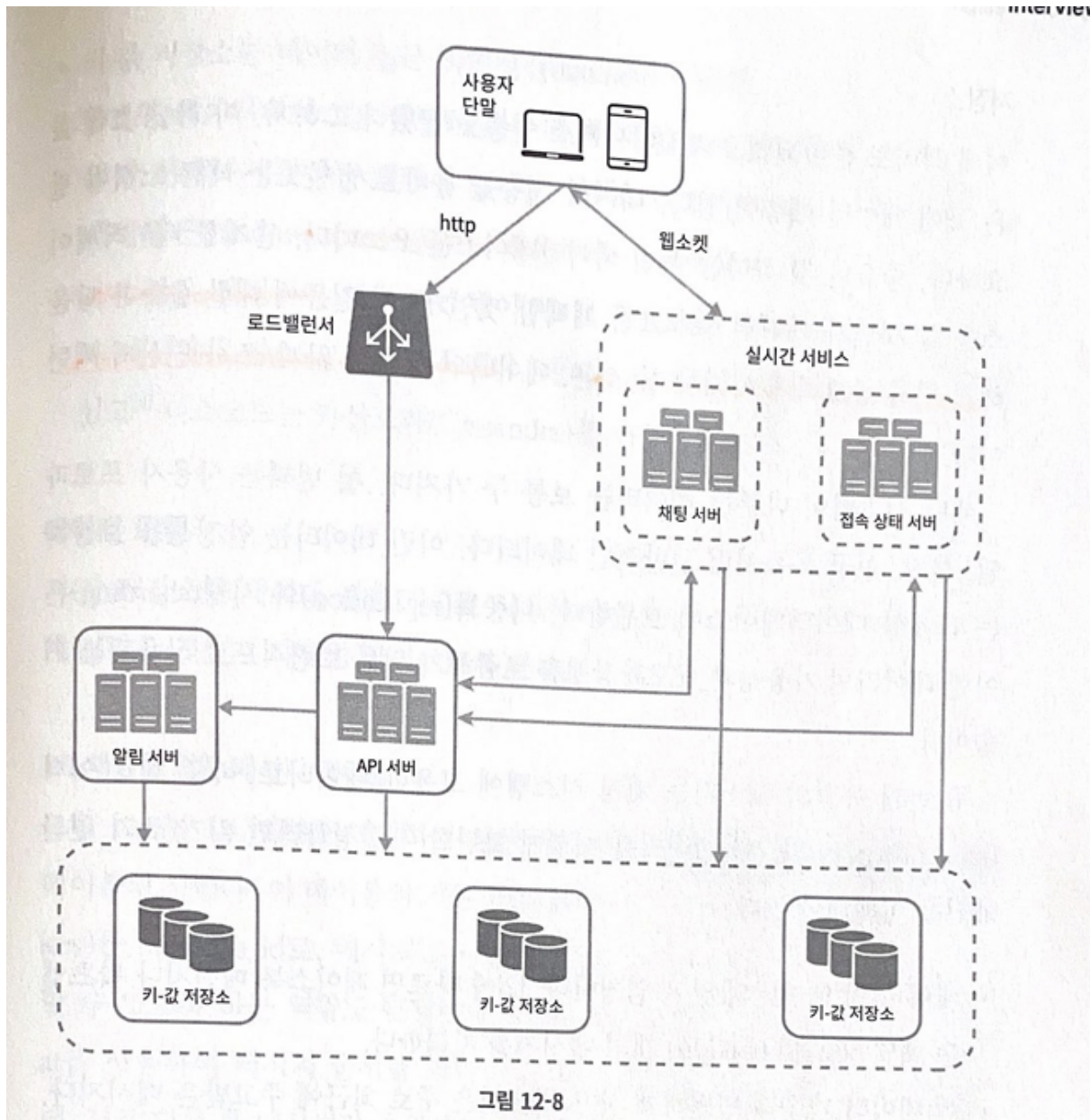
새 메시지를 받았다면 앱이 실행 중이지 않더라도 알림을 받아야 함

규모 확장성

대량의 트래픽을 처리해야 하는 경우에도 이론적으로는 모든 사용자 연결을 최신 클라우드 서버 한 대로 처리 가능 → 서버 한대로 얼마나 많은 접속을 동시에 허용할 수 있는지를 살펴 봐야 함

그러나 모든 것을 서버 한 대에 담은 설계안은 SPOF 등의 이유로 좋은 점수를 받기 어려움
그렇지만 서버 한 대만 갖는 설계안에서 출발하여 점차 다듬어 나가는 것은 괜찮음 ^^

지금까지 만든 설계안



- 채팅 서버는 클라이언트 사이에 메시지를 중계하는 역할을 담당
- 접속상태 서버는 사용자의 접속 여부를 관리
- API 서버는 로그인, 회원가입, 프로필 변경 등 그 외 나머지 전부를 처리
- 알림 서버는 푸시 알림을 보냄
- 키-값 저장소에는 채팅 이력을 보관 → 시스템에 접속한 사용자는 이전 채팅 이력을 전부 확인 가능

저장소

데이터베이스를 선택하기 위해 아래 두 가지를 중요하게 따져야 함

1. 데이터의 유형

2. 읽기/쓰기 연산의 패턴

1) 데이터 유형

채팅 시스템이 다루는 데이터는 보통 두 가지임

1. 사용자 프로필, 설정, 친구 목록 등의 일반적인 데이터

- 안전성을 보장하는 관계형 데이터베이스에 저장
- 다중화, 샤딩은 데이터의 가용성과 규모확장성을 보증하기 위해 보편적으로 사용됨

2. 채팅 시스템에 고유한 데이터 - 채팅 이력

- 데이터를 어떻게 보관할지 결정하려면 읽기/쓰기 연산 패턴을 이해해야 함

2) 읽기 쓰기 패턴

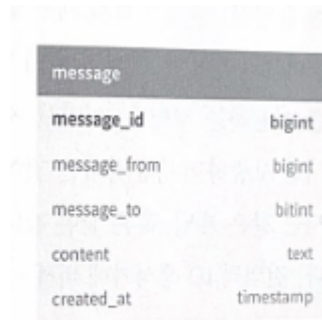
- 채팅 이력 데이터의 양은 엄청남 → 페메나 왓츠앱은 매일 600억 개의 메시지를 처리함
- 채팅 데이터 가운데 빈번하게 사용되는 것을 주로 최근에 주고 받은 메시지가 됨 → 대부분의 사용자는 오래된 메시지는 들여 보지 않음
- 사용자는 대체로 최근에 주고 받은 메시지 데이터만 보게 되는 것이 사실이나, 검색 기능을 이용하거나 특정 사용자가 언급된 메시지를 보거나, 특정 메시지로 점프하거나 하여 무작위적인 데이터 접근을 하게 되는 경우도 있음 → 데이터 계층은 이런 기능도 지원해야 함
- 1:1 채팅 앱의 경우 읽기:쓰기 비율은 대략 1:1

위를 지원하는 데이터베이스로 책에서는 키-값 저장소를 추천

- 키-값 저장소는 수평적 규모 확장이 가능
- 키-값 저장소는 데이터 접근 지연시간이 낮음
- 관계형 데이터베이스는 데이터 가운데 롱 테일에 해당하는 부분을 잘 처리하지 못하는 경향이 있음 → 인텐스가 커지면 데이터에 대한 무작위 접근을 처리하는 비용이 늘어남
- 이미 많은 안정적인 채팅 시스템이 키-값 저장소를 채택하고 있음

데이터 모델

1:1 채팅을 위한 메시지 테이블



message	
message_id	bigint
message_from	bigint
message_to	bigint
content	text
created_at	timestamp

그림 12-9

테이블의 기본키는 message_id로 메시지 순서를 쉽게 정할 수 있도록 하는 역할을 담당
created_at을 사용하여 메시지 순서를 정할 수는 없음 → 서로 다른 두 메시지가 동시에 만들어질 수도 있기 때문

그룹 채팅을 위한 메시지 테이블

(channel_id, message_id)의 복합 키를 기본 키로 사용함

channel_id는 파티션 키로도 사용 → 그룹 채팅에 적용될 모든 질의는 특정 채널을 대상으로 하기 때문

메시지 ID

message_id는 메시지들의 순서를 표현할 수 있어야 하므로 아래의 속성을 만족해야 함

- message_id의 값은 고유해야 함
- ID값은 정렬 가능해야 하며 시간 순서와 일치해야 한다. 즉, 새로운 ID는 이전 ID보다 큰 값이어야 한다.

아래 세 가지 방법으로 메시지 ID 생성이 가능함

- RDBMS의 경우 auto_incremeant 사용
- 스노플레이크 같은 전역적 64-bit 순서번호 생성기를 이용
- 지역전 순서 번호 생성기 이용
 - 지역적 = ID의 유일성이 같은 그룹 안에서만 보증되면 충분하다

- 메시지 사이의 순서는 같은 채널, 혹은 같은 1:1 채팅 세션 안에서만 유지되면 충분하므로 가능해짐

3단계 - 상세 설계

채팅 시스템의 경우 서비스 탐색, 메시지 전달 흐름, 사용자 접속 상태를 표시하는 방법 정도를 좀 더 자세히 살펴볼 만함

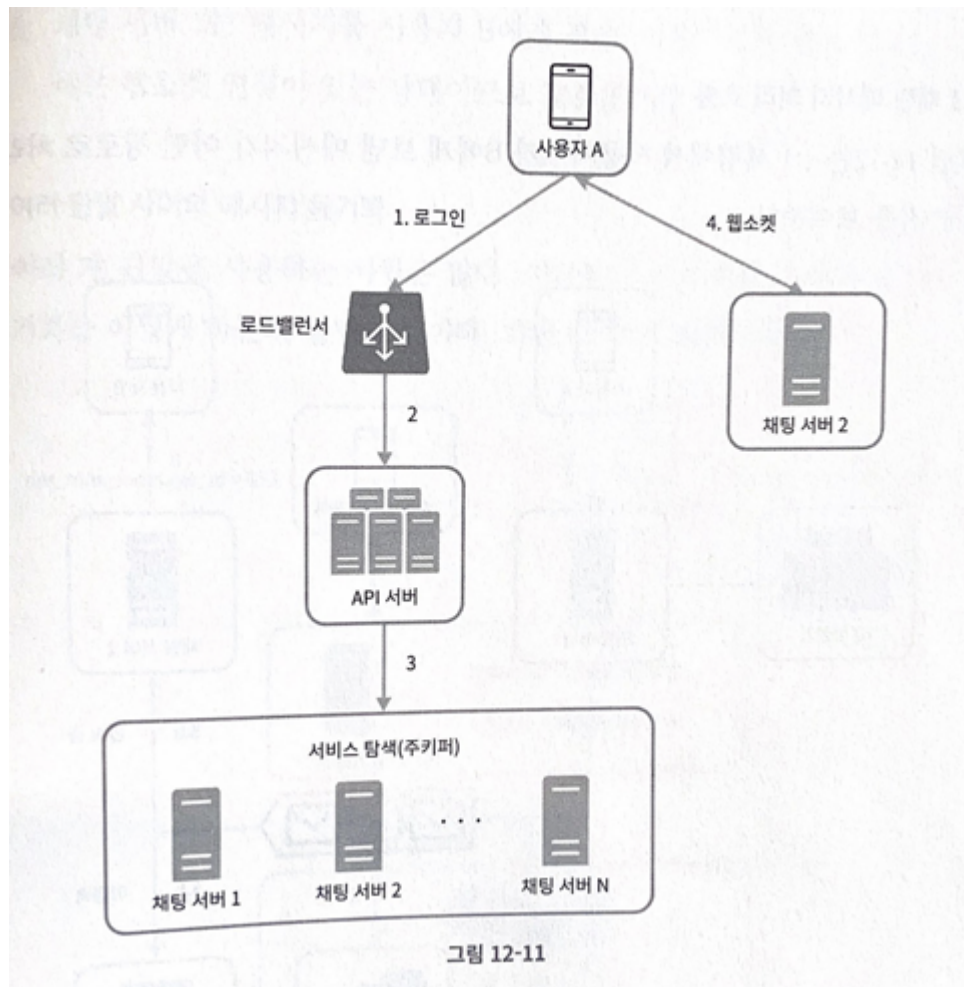
서비스 탐색

서비스 탐색 기능의 주된 역할은 클라이언트에게 가장 적합한 채팅 서버를 추천하는 것 → 클라이언트의 위치, 서버의 용량등이 기준이 됨

아파치 주키퍼 등이 서비스 탐색 기능 구현에 사용됨

- 사용 가능한 모든 채팅 서버를 여기 등록 시키고, 클라이언트가 접속을 시도하면 사전에 정한 기준에 따라 최적의 채팅 서버를 골라줌

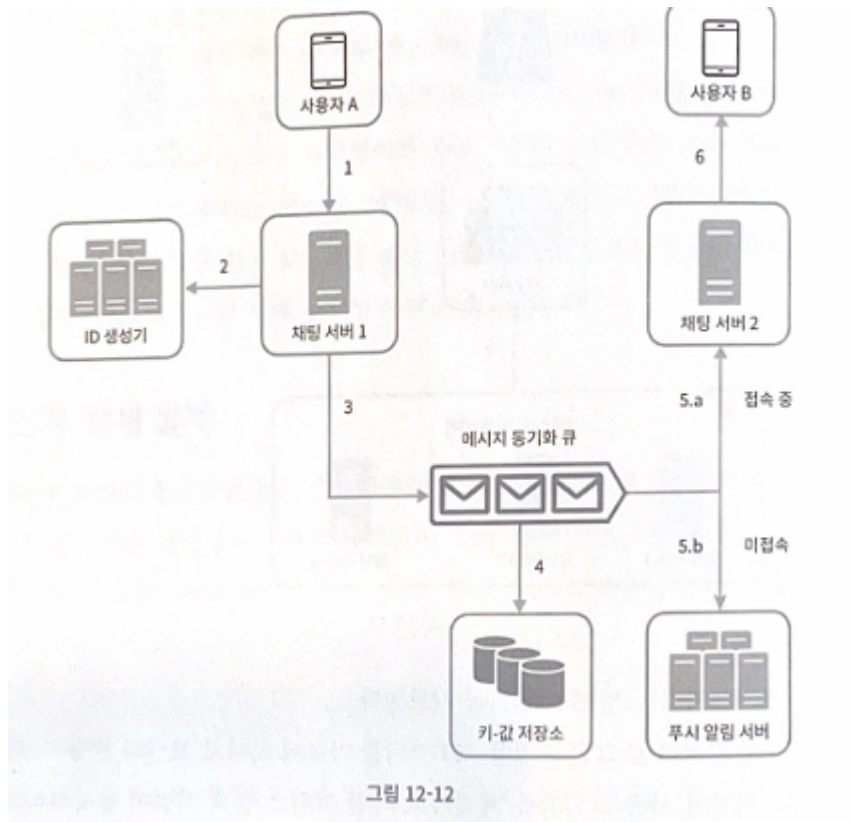
주키퍼로 구현한 서비스 탐색 기능의 동작 과정



1. 사용자 A가 시스템에 로그인을 시도
2. 로드밸런서가 로그인 요청을 API 서버들 가운데 하나로 보냄
3. API 서버가 사용자 인증을 처리하고 나면 서비스 탐색 기능이 동작 → 해당 사용자를 서비스할 최적의 채팅 서버를 찾음
 - a. 해당 예제에서는 서버2가 선택되어 반환됨
4. 사용자 A는 채팅 서버 2와 웹소켓 연결을 맺음

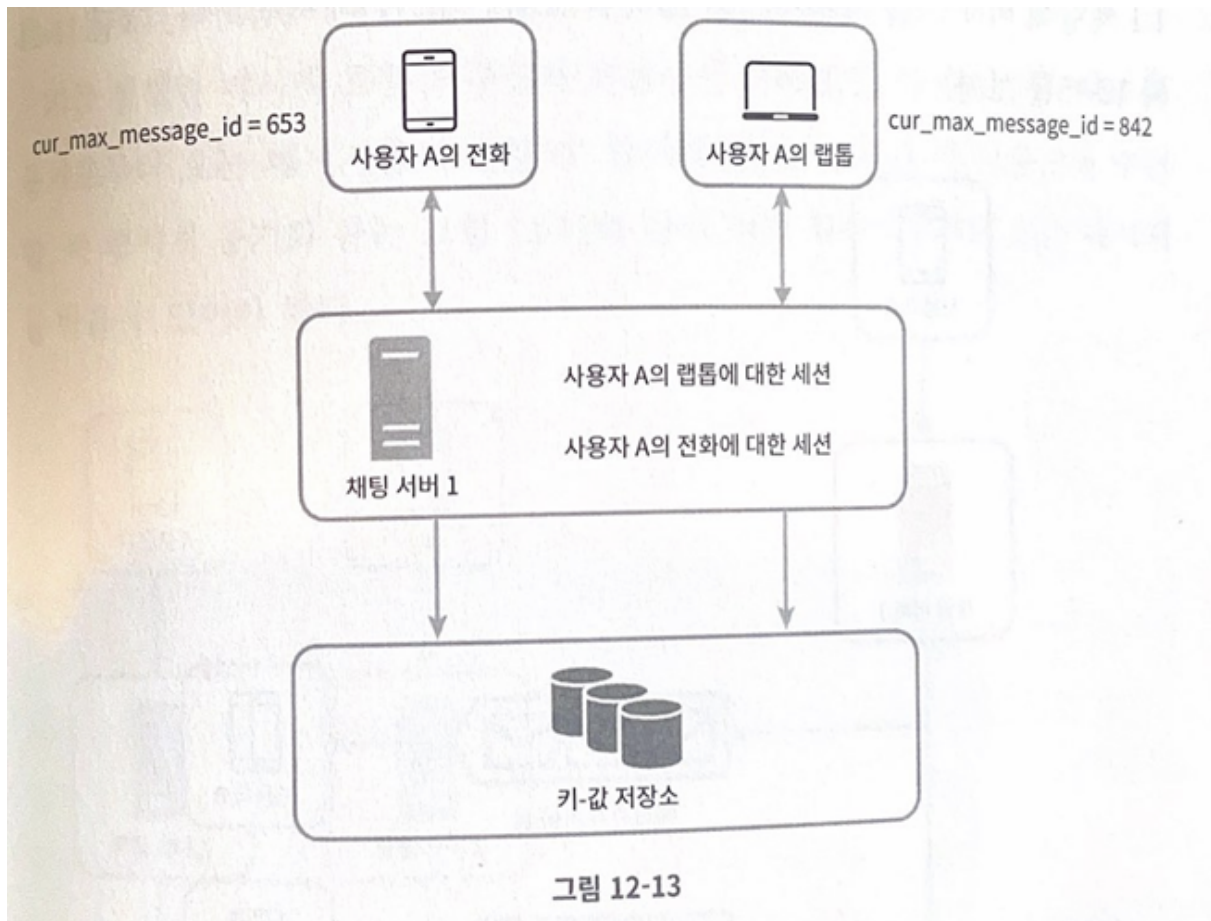
메시지 흐름

1:1 채팅 메시지 처리 흐름



1. 사용자 A가 채팅 서버 1로 메시지 전송
2. 채팅 서버 1은 ID 생성기를 사용, 해당 메시지의 ID를 결정
3. 채팅 서버 1은 해당 메시지를 메시지 동기화 큐로 전송
4. 메시지가 키-값 저장소에 보관됨
5. (a) 사용자 B가 접속 중인 경우 메시지는 사용자 B가 접속 중인 채팅 서버로 전송됨 (b) 사용자 B가 접속 중이 아니라면 푸시 알림 메시지를 푸시 알림 서버로 보냄
6. 채팅 서버 2는 메시지를 사용자 B에게 전송. 사용자 B와 채팅 서버 2 사이에는 웹 소켓 연결이 있는 상태이므로 그것을 사용

여러 단말 사이의 메시지 동기화



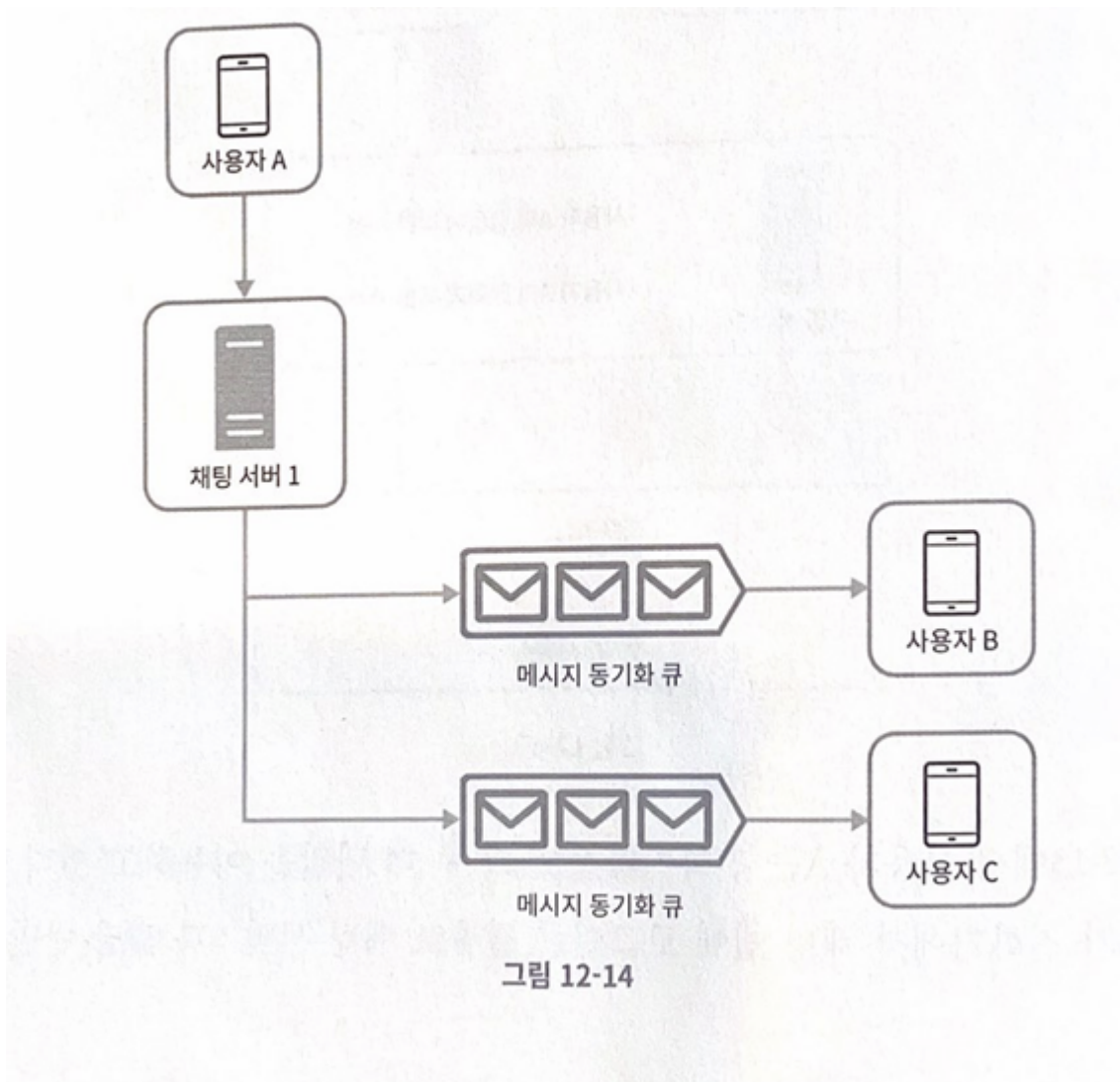
1. 사용자 A는 전화기와 랩톱의 두 대 단말을 사용 중
2. 사용자 A가 전화기에서 채팅 앱에 로그인한 결과로 채팅 서버1과 해당 단말 사이에 웹 소켓 연결이 만들어져 있고, 랩톱에서도 로그인한 결과로 별도 웹소켓이 채팅 서버 1에 연결되어 있음
3. 각 단말은 해당 단말에서 관측된 가장 최신 메시지의 ID를 추적하기 위한 `cur_max_message_id`라는 변수를 유지

다음의 두 조건을 만족하는 메시지는 새 메시지로 간주함

- 수신자 ID가 현재 로그인한 사용자 ID와 같다
- 키-값 저장소에 보관된 메시지로서, 그 ID가 `cur_max_message_id`보다 크다

`cur_max_message_id`는 단말마다 별도로 유지 관리하면 되므로 키 값 저장소에서 새 메시지를 가져오는 동기화 작업도 쉽게 구현 가능

소규모 그룹 채팅에서의 메시지 흐름



사용자 A가 보낸 메시지는 사용자 B와 C의 메시지 동기화 큐에 복사됨

- 동기화 큐는 사용자 각각에 할당된 메시지 수신함으로 생각해도 ok

위의 설계안은 소규모 그룹 채팅에 적합함

- 새로운 메시지가 왔는지 확인하려면 자기 큐만 보면 됨 → 메시지 동기화 플로우가 단순
- 그룹이 크지 않으면 메시지를 수신자별로 복사해서 큐에 넣는 작업의 비용이 문제 되지 않음

위챗은 이런 접근법을 사용 → 그룹 크기를 500명으로 제한

많은 사용자를 지원해야 하는 경우에는 똑같은 메시지를 모든 사용자의 큐에 복사하는 것이 바람직하지 않음

접속상태 표시

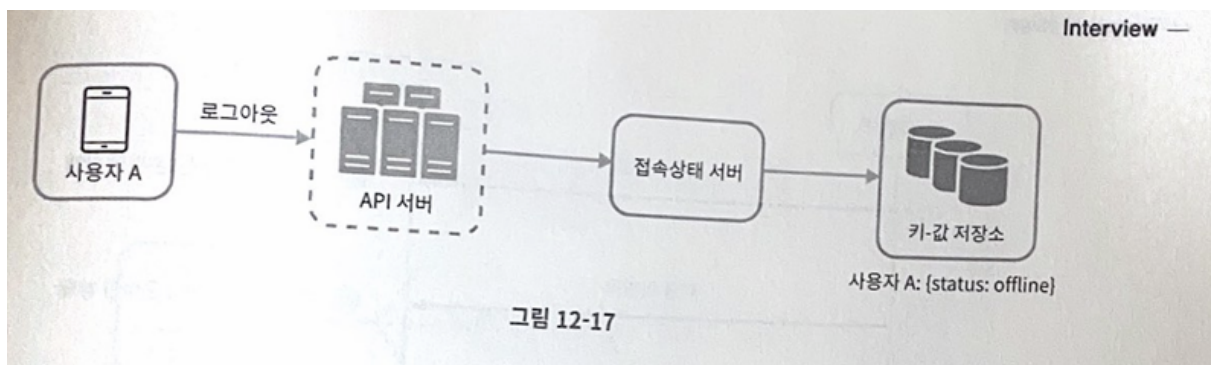
접속 상태 서버를 사용하여 사용자의 상태를 관리

사용자의 상태가 바뀌는 시나리오는 몇 가지가 존재

사용자 로그인

클라이언트와 실시간 서비스 사이에 웹 소켓 연결이 맺어지고 나면 접속 상태 서버는 A의 상태와 `last_active_at` 타임스탬프 값을 키-값 저장소에 보관 → 사용자의 상태가 접속 중인 것으로 표시됨

사용자 로그아웃



키-값 저장소에 보관된 사용자 상태가 online → offline으로 변경됨

절차가 끝나면 사용자의 상태는 접속 중이 아닌 것으로 변경됨

접속 장애

사용자의 인터넷 연결이 끊어지면 클라이언트와 서버 사이에 맺어진 웹소켓 같은 지속성 연결도 끊어지게 됨 → 상황에 대응할 수 있는 설계를 준비해야 함

장애 대응 방법

1. 사용자를 오프라인 상태로 표시 후 연결 복구 시 온라인으로 상태 변경

- 짧은 시간 동안 인터넷 연결이 끊어졌다 복구되는 일은 흔함
 - 이런 일이 벌어질 때 마다 사용자의 접속 상태를 변경하는 것은 바람직하지 X

2. 박동 검사 사용

- 온라인 상태의 클라이언트로 하여금 주기적으로 박동 이벤트를 접속 상태 서버로 보냄
 - 마지막 이벤트를 받은지 x초 이내에 또 다른 박동 이벤트 메시지를 받으면 해당 사용자의 접속 상태를 계속 온라인으로 유지하고 그렇지 않을 경우에만 오프라인으로 변경

상태 정보의 전송

상태 정보 서버는 발행-구독 (pub-sub) 모델을 사용하므로 각각의 친구 관계마다 채널을 하나씩 두게 됨

ex) 사용자 A의 접속 상태가 변경된 경우

1. 변경을 세 개의 채널 (A-B, A-C, A-D)에 작성
2. A-B는 구독자 B가, A-C는 구독자 C가, A-D는 구독자 D가 구독함

⇒ 친구관계에 있는 사용자가 상태 정보 변화를 쉽게 받아들일 수 있게 됨

단, 해당 방식은 그룹의 크기가 작을 때만 사용 가능

4단계 - 마무리

추가로 논의하면 좋을 내용

- 채팅 앱을 확장하여 사진이나 비디오 등의 미디어를 지원하도록 하는 방법
- 종단간 암호화
- 캐시
- 로딩 속도 개선
- 오류처리
 - 채팅 서버 오류

- 메시지 재전송

13장. 검색어 자동완성 시스템

1단계 - 문제 이해 및 설계 범위 확정

요구사항

- 빠른 응답 속도: 사용자가 검색어를 입력함에 따라 자동완성 검색어도 충분히 빨리 표시되어야 함
- 연관성: 자동완성되어 출력되는 검색어는 사용자가 입력한 단어와 연관된 것이어야 함
- 정렬: 시스템의 계산 결과는 인기도 등의 순위 모델에 의해 정렬되어 있어야 함
- 규모 확장성: 시스템은 많은 트래픽을 감당할 수 있도록 확장 가능해야 함
- 고가용성: 시스템의 일부에 장애가 발생하거나, 느려지거나, 예상치 못한 네트워크 문제가 생겨도 시스템은 계속 사용 가능해야 함

개략적 규모 추정

- 일간 능동 사용자(DAU)는 천만 명으로 가정
- 평균적으로 한 사용자는 매일 10건의 검색을 수행
- 질의 시, 평균적으로 20바이트의 데이터를 입력
 - 문자 인코딩 방법으로는 ASCII를 사용 → 1문자 = 1바이트
 - 질의는 평균적 4개의 단어로 이루어짐, 각 단어는 평균적으로 다섯 글자로 이루어짐
 - 질의당 평균 $4 \times 5 = 20$ 바이트
- 검색창에 글자 입력 시, 클라이언트는 검색어 자동완성 백엔드에 요청을 보냄 → 평균적으로 1회 검색당 20건의 요청이 백엔드로 전달

ex) dinner를 검색창에 입력할 경우 다음 여섯 개의 요청이 순차적으로 백엔드로 전달 됨

search?q=d

search?q=di

search?q=din

search?q=dinn

search?q=dinne

search?q=dinner

- 대략 초당 24,000건의 질의(QPS)가 발생
- 최대 QPS = QPSx2 = 48,000
- 질의 가운데 20%는 신규 검색어로 가정 → 0.4GB
 - 매일 0.4GB의 신규 데이터가 시스템에 추가됨

2단계 - 개략적 설계안 제시 및 동의 구하기

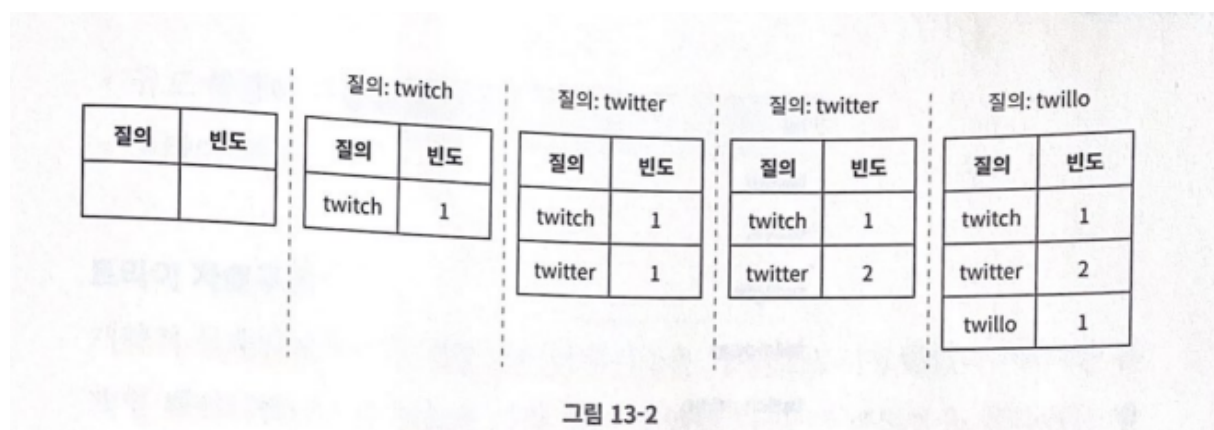
시스템은 개략적으로 두 부분으로 나누어짐

- 데이터 수집 서비스: 사용자가 입력한 질의를 실시간으로 수집하는 시스템
 - 단, 데이터가 많은 애플리케이션에 실시간 시스템은 그다지 바람직하지 않음
- 질의 서비스: 주어진 질의에 다섯 개의 인기 검색어를 정렬해 내놓는 서비스

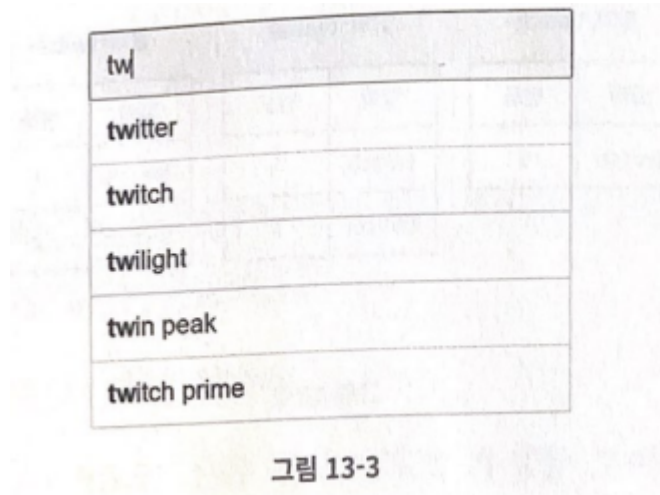
데이터 수집 서비스

데이터 수집 서비스의 동작 과정

1. 질의문과 사용 빈도를 저장하는 빈도 테이블이 존재
2. 초기에 빈도 테이블은 비어 있지만, 후에 사용자가 'twitch', 'twitter', 'twitter', 'twillo'를 순서대로 검색하면 상태가 변경됨



질의 서비스



A diagram showing a table with search suggestions. The table has two columns: 'query' and 'frequency'. The first row shows 'tw|' in the query column and an empty frequency column. The subsequent rows show suggestions: 'twitter', 'twitch', 'twilight', 'twin peak', and 'twitch prime'.

tw	
twitter	
twitch	
twilight	
twin peak	
twitch prime	

그림 13-3

- query: 질의문을 저장하는 필드
- frequency: 질의문이 사용된 빈도를 저장하는 필드

이 상태에서 사용자가 “tw”를 검색창에 입력하면 아래의 top5 자동완성 검색어가 표시되어야 함

- top5는 빈도 테이블에 기록된 수치를 사용하여 계산

```
select * from frequency_table where query like `prefix%` or
```

데이터의 양이 적을 때는 나쁘지 않지만, 데이터가 많아질 경우 데이터베이스 병목 가능성 O

3단계 - 상세 설계

상세 설계 & 최적화할 컴포넌트

- 트라이 자료구조
- 데이터 수집 서비스
- 질의 서비스
- 규모 확장이 가능한 저장소
- 트라이 연산

트라이 자료구조

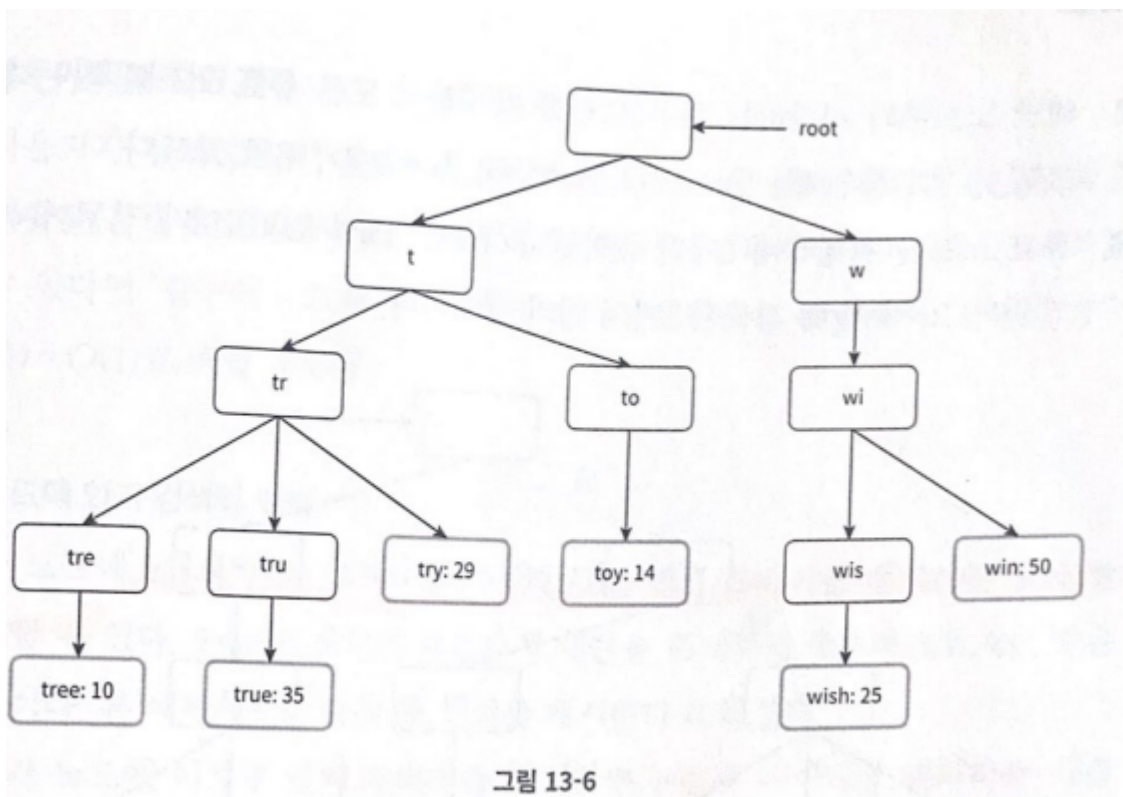
개략적 설계안: 관계형 데이터베이스를 사용, 가장 인기 있는 다섯 개의 질의문을 골라냄 → 비효율적

상세 설계안: 트라이를 사용하여 가장 인기 있는 다섯 개의 질의문을 골라냄

트라이 자료구조?

: 문자열들을 간략하게 저장할 수 있고, 문자열을 꺼내는 연산에 초점을 맞추어 설계된 자료구조

- 핵심 아이디어
 - 트리 형태의 자료구조
 - 트리의 루트 노드는 빈 문자열을 나타냄
 - 각 노드는 글자 하나를 저장하며, 26개의 자식 노드를 가질 수 있음
 - 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타냄



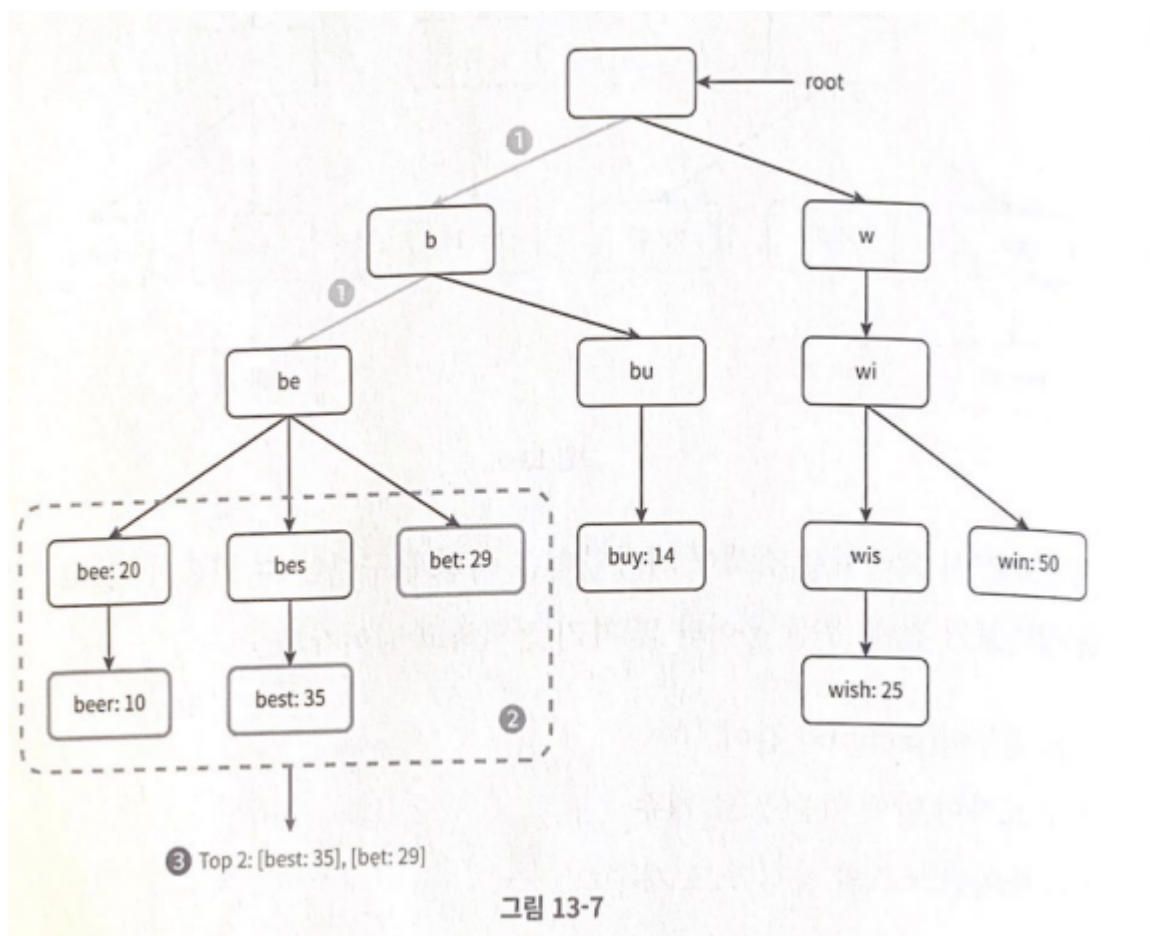
트라이로 검색어 자동완성을 구현하는 방법

- p: 접두어(prefix)의 길이
- n: 트라이 안에 있는 노드 개수
- c: 주어진 노드의 자식 노드 개수

가장 많이 사용된 질의어 K개 찾기

- 해당 접두어를 표현하는 노드를 찾음 → 시간 복잡도: $O(p)$
- 해당 노드부터 시작하는 하위 트리를 탐색, 모든 유효 노드를 찾음 → 시간 복잡도: $O(c)$
 - 유효 노드: 유효한 검색 문자열을 구성하는 노드
- 유효 노드들을 정렬하여 가장 인기 있는 검색어 k개를 찾음 → 시간 복잡도: $O(c \log c)$

ex) k = 2이고 사용자가 검색창에 'be'를 입력한 경우



1. 접두어 노드 'be'를 찾음
2. 해당 노드부터 시작하는 하위 트리를 탐색, 모든 유효 노드를 찾음

- 해당 예제에서는 [beer:10], [best:35], [bet: 29]가 유효 노드가 됨

3. 유효 노드를 정렬하여 2개만 골라냄

- [best: 35]와 [bet: 29]가 접두어 "be"에 대해 검색된 2개의 인기 검색어가 됨

위 알고리즘의 시간 복잡도는 각 단계에 소요된 시간의 합 $\rightarrow O(p) + O(c) + O(c \log c)$

알고리즘이 직관적이지만, 최악의 경우에는 k개의 결과를 얻기 위해 전체 트리를 다 검색해야 하는 경우가 생김

최적화 방안

- 접두어의 최대 길이 제한
- 각 노드에 인기 검색어를 캐시

1. 접두어 최대 길이 제한

사용자가 검색창에 긴 검색어를 입력하는 경우는 거의 없음 $\rightarrow P$ 값을 작은 상숫값으로 가정해도 OK

검색어의 최대 길이를 제한할 경우 "접두어 노드를 찾는" 단계의 시간 복잡도는 $O(p) \rightarrow O(\text{작은 상숫값}) = O(1)$ 로 변경됨

2. 노드에 인기 검색어 캐시

각 노드에 K개의 인기 검색어를 저장해두면 전체 트라이를 검색하는 일을 방지 가능

각 노드에 질의어를 저장할 공간이 많이 필요해지지만, 빠른 응답속도가 중요할 경우에는 좋은 방안이 됨

위의 두 최적화 기법을 적용하면 각 단계의 시간 복잡도가 $O(1)$ 로 변경되므로, 최고 인기 검색어 K개를 찾는 전체 알고리즘 복잡도가 $O(1)$ 로 변경됨

데이터 수집 서비스

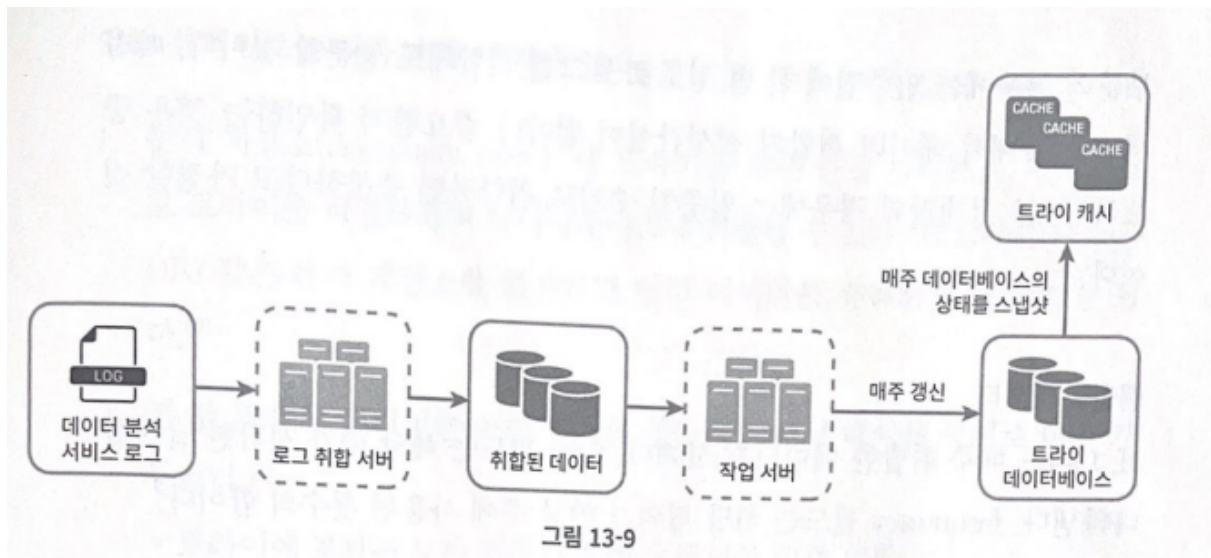
사용자가 검색창에 타이핑할 때 마다 실시간으로 데이터를 수정하는 방법의 문제점

- 매일 수천만 건의 질의가 입력 \rightarrow 그때마다 트라이 갱신할 경우 질의 서비스가 느려짐

- 트라이가 만들어지면 인기 검색어는 자주 바뀌지 않음 → 트라이를 자주 갱신할 필요 X

규모 확장이 쉬운 데이터 수집 서비스를 만들려면 데이터가 어디서 오고 어떻게 이용되는 지를 알아야 함

- 트라이를 만드는 데 쓰는 데이터는 데이터 분석 서비스나 로깅 서비스로부터 오게 됨



데이터 분석 서비스 로그

검색창에 입력된 질의에 관한 원본 데이터가 보관됨

- 새로운 데이터가 추가될 뿐 수정은 이루어지지 X
- 로그 데이터에는 인덱스를 걸지 않음

로그 취합 서버

로그는 보통 양이 엄청나고 데이터 형식도 제각각인 경우가 많음 → 데이터를 잘 취합 + 시스템이 쉽게 소비할 수 있도록 해야 함

데이터 취합 방식

: 서비스의 용례에 따라 달라짐

- 실시간 애플리케이션: 결과물을 빨리 보여줘야 함 → 취합 주기가 짧음
- 대부분 애플리케이션: 일주일에 한 번 정도로 로그 취합해도 충분

취합된 데이터

query	time	frequency
tree	2019-10-01	12000
tree	2019-10-08	15000
tree	2019-10-15	9000
toy	2019-10-01	8500
toy	2019-10-08	6256
toy	2019-10-15	8866

표 13-4

작업 서버

주기적으로 비동기적 작업을 실행하는 서버 집합

트라이 자료구조를 만들고 트라이 데이터베이스에 저장하는 역할을 담당

트라이 캐시

분산 캐시 시스템. 트라이 데이터를 메모리에 유지 → 읽기 연산 성능을 높임

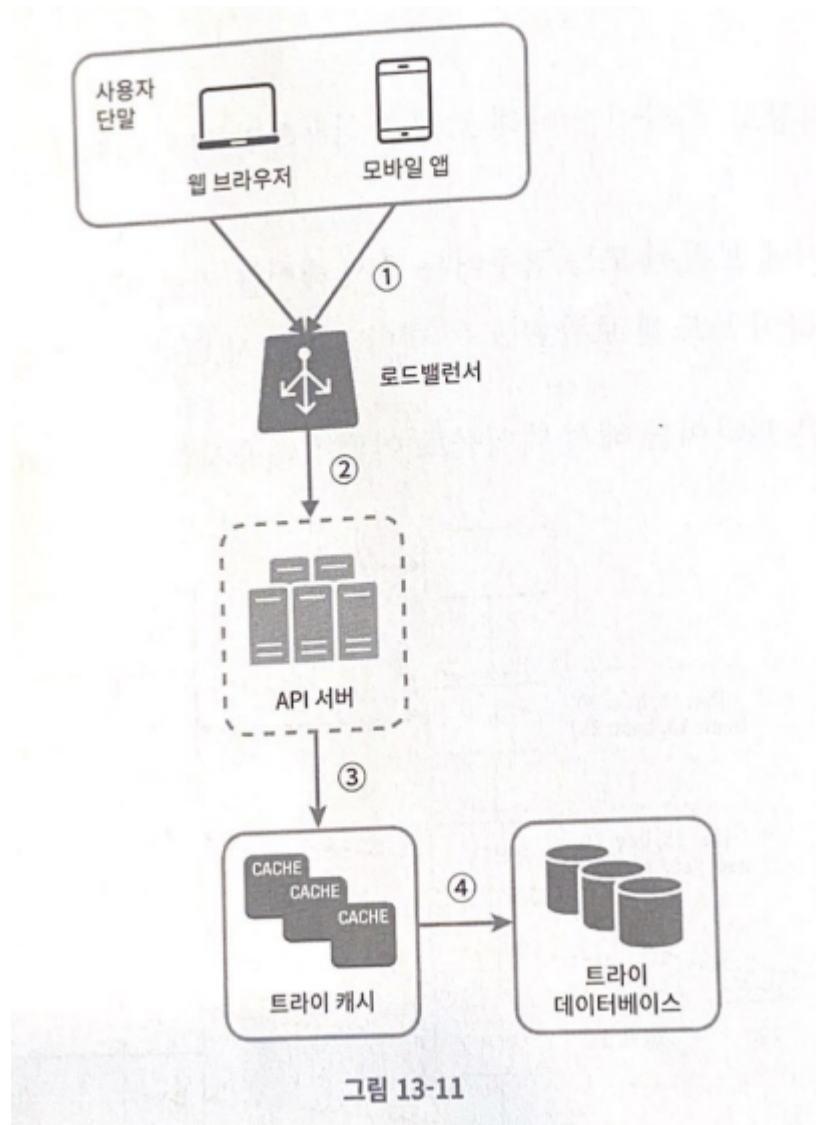
매주 트라이 데이터베이스의 스냅샷을 떼서 갱신

트라이 데이터베이스

지속성 저장소

1. 문서 저장소: 새 트라이를 매주 만들 것이므로, 주기적으로 트라이를 직렬화하여 데이터 베이스에 저장 가능
2. 키-값 저장소: 트라이는 아래 로직을 적용하면 해시 테이블 형태로 변환 가능
 - 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환
 - 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값으로 변환
 - 트라이 노드는 하나의 <키, 값> 쌍으로 변경

질의 서비스



1. 검색 질의가 로드밸런서로 전송됨
2. 로드밸런서는 해당 질의를 API 서버로 보냄
3. API 서버는 트라이 캐시에서 데이터를 가져와 해당 요청에 대한 자동완성 검색어 제안 응답을 구성함
4. 데이터가 트라이 캐시에 없는 경우, 데이터를 데이터베이스에서 가져와 캐시에 채움 → 같은 접두어에 대한 질의가 오면 캐시에 보관된 데이터를 사용해 처리 가능
 - 캐시 미스는 캐시 서버의 메모리가 부족하거나 캐시 서버에 장애가 있어도 발생할 수 있음

질의 서비스는 속도가 무척이나 중요함

질의 서비스 최적화 방법

- AJAX 요청: 웹 애플리케이션의 경우 브라우저는 보통 AJAX 요청을 보내어 자동완성된 검색어 목록을 가지고 옴
 - 요청을 보내고 받기 위해 페이지를 새로고침 할 필요가 없다는 장점을 가짐
 - 브라우저 캐싱: 대부분 애플리케이션의 경우 자동완성 검색어 제안 결과는 짧은 시간 안에 자주 바뀌지 않음
 - 제안된 검색어들을 브라우저 캐시에 넣어두면 후속 질의의 결과는 해당 캐시에서 바로 가져갈 수 있음
- ex) 구글 검색 엔진 - 제안된 검색어를 한 시간 동안 캐시해 둠
- 데이터 샘플링: N개의 요청 가운데 1개만 로깅하도록 하는 것
 - 대규모 시스템의 경우, 모든 질의 결과를 로깅하면 CPU 자원과 저장공간을 엄청나게 소모하게 된다는 단점 보완

트라이 연산

검색어 자동완성 시스템의 핵심 컴포넌트

트라이 생성

작업 서버가 담당, 데이터 분석 서비스의 로그나 데이터베이스로부터 취합된 데이터를 이용

트라이 갱신

1. 매주 한번 갱신하는 방법: 새로운 트라이를 만든 후, 기존 트라이를 대체
2. 트라이의 각 노드를 개별적으로 갱신하는 방법: 트라이 노드 갱신 시, 상위 노드도 함께 갱신
 - 트라이가 적을 때 고려해볼 수 O

검색어 삭제

트라이 캐시 앞에 필터 계층을 두고 부적절한 질의가 반환되지 않도록 구현

필터 계층을 두면, 필터 규칙에 따라 검색 결과를 자유롭게 변경할 수 있다는 장점이 존재

저장소 규모 확장

책에서 구현하는 시스템은 영어만 지원하므로 첫 글자를 기준으로 샤딩하는 방법을 생각할 수 있음

- 검색어를 보관하기 위해 두 대 서버가 필요하다면 'a'부터 'm'까지 글자로 시작하는 검색어는 첫번째 서버에 저장, 나머지는 두 번째 서버에 저장
- 세 대의 서버가 필요하다면 'a'부터 'i'까지는 첫 번째 서버에, 'j'부터 'r'까지는 두 번째 서버에, 나머지는 세 번째 서버에 저장

위의 방법은 영어 알파벳이 26자 밖에 없으므로 사용 가능한 서버가 최대 26대로 제한됨

서버의 대수를 늘리려면 샤딩을 계층적으로 진행 → 데이터를 각 서버에 균등하게 배분하지 못하는 문제 발생

- 과거 질의 데이터의 패턴을 분석하여 샤딩하는 방법으로 문제 해결 가능
 - 샤드 관리자가 어떤 검색어가 어느 저장소 서버에 저장되는지에 대한 정보를 관리

4단계 - 마무리

다국어 지원

- 트라이에 유니코드 데이터를 저장

국가별 인기 검색어 순위

- 국가별로 다른 트라이 사용
 - 트라이를 CDN에 저장, 응답속도를 높이는 방법도 O

실시간 검색어 추이 반영

- 현 설계안은 실시간 추이 반영에는 적합하지 않음
 - 작업 서버가 매주 한 번씩만 돌도록 되어 있으므로 시의 적절하게 트라이를 갱신할 수 없음

- 때에 맞추어 서버가 실행 되어도, 트라이 구성에 너무 많은 시간이 소요됨
- 생각해볼 수 있는 방법
 - 샤딩을 통해 작업 대상 데이터의 양을 줄임
 - 순위 모델을 변경 → 최근 검색어에 보다 높은 가중치를 줌
 - 데이터가 스트림 형태로 올 수 있다는 점, 즉 한번에 모든 데이터를 동시에 사용할 수 없을 가능성이 있다는 점을 고려