주변 친구 (240107)

▼ 지난 회차 토론 복습

왜 읽기 연산이 많은 시스템에 RDB가 적합한가?

- **특정한 조건**에 맞는 데이터를 검색할 때 빠르게 검색할 수 있기 때문
- 일관성 있는 데이터를 제공할 수 있음

> 🔥 본인 위치 정보 접근 권한을 허용한 사용자에 한해 인근 친구 목록을 보여주자

근접성 서비스와 다른 점?

- 근접성 서비스의 사업장 주소는 **정적**. 주변 친구의 위치는 **동적**

▼ 1️⃣ 설계 범위 확정

거리	정책	기타
`주변에 있다`의 기준?	이동 이력 보관 여부	앱 사용자 수
거리 산정 방식(ex: 직선 거리)	비활성 상태 사용자 표시 여부	기능 사용자 수
	사생활 및 데이터 보호법	

▼ 2️⃣ 요구사항 정리

기능 요구사항

- 주변 친구 확인
  - 친구와의 거리
  - 마지막 갱신 시각
- 친구 목록 n초마다 갱신

비기능 요구사항

낮은 지연시간	안정성	결과적 일관성

▼ 3️⃣ 개략적 설계

규모 추정

- `주변 친구`는 8km 반경 이내 친구이다.
- 친구 위치 정보는 30초 주기로 갱신한다.
- 주변 친구 검색 평균 사용자는 매일 1억명이다.
- 동시 접속 사용자 수는 DAU의 10%, 즉 1000만명이다.
- 한 사용자는 평균 400명의 친구를 갖는다. 와
- 페이지 당 20명의 주변 친구를 표시한다.

🔎 QPS 계산하기

- **DAU** : 1억
- **동시 접속 사용자** : 1억 * 10% = 1000만
- 30초마다 자기 위치 시스템에 전송
- 위치 정보 갱신 QPS : 1000만 / 30 ~= **334,000**

고려할 점

- 모든 활성 상태 사용자의 위치 변화 내역 수신
- 사용자 위치 변경 내역 수신마다 해당 사용자의 모든 활성 상태 친구를 찾아서 변경 내역 전달
- 두 사용자 간 거리가 특정치보다 먼 경우 변경 내역을 전송하지 않음

⇒ 큰 규모에 적용 어려움

기본 설계안

로드밸런서

- RESTful API 서버와 양방향 유상태 웹소켓 서버에 부하를 고르게 분산

RESTful API 서버

- 무상태 서버 클러스터
- 사용자, 친구 관리 / 인증, …

웹소켓 서버

- 친구 위치 정보 변경을 거의 실시간에 가깝게 처리
- 각 클라이언트는 서버 한 대와 웹소켓 연결을 유지

레디스 (위치 정보 캐시)

- 활성 상태 사용자의 가장 최근 위치 정보를 캐싱
- 정보 갱신 시 TTL도 함께 갱신

사용자 데이터베이스

- 사용자 데이터
- 사용자의 친구 관계 정보

위치 이동 이력 데이터베이스

- 위치 변동 히스토리 보관

레디스 Pub/Sub 서버

- 초경량 메시지 버스
- **동작 방식**
  - 위치 정보 갱신 이벤트 발행
  - 레디스 Pub/Sub 채널에 저장
  - 해당 사용자의 친구와 연결된 웹소켓 연결 핸들러가 채널을 구독
  - 구독자들의 웹소켓 연결 핸들러가 호출
  - 수신할 친구가 활성 상태면 거리를 다시 계산
  - 새로 계산한 거리가 반경 이내면 갱신된 위치와 시각을 단말에 전달
- **주기적 갱신**
  - 클라이언트가 위치 변경 사실을 로드밸런서에 전송
  - 로드밸런서는 위치 변경 내역을 클라이언트와 웹소켓 서버 사이 설정된 연결을 통해 웹소켓 서버로 전달
  - 웹소켓 서버는 해당 이벤트를 위치 이동 이력 데이터베이스에 저장
  - 웹소켓 서버는 새 위치를 위치 정보 캐시에 보관하고 TTL을 갱신
  - 웹소켓 서버는 웹소켓 연결 핸들러 안의 변수에 위치 반영

- 웹소켓 서버는 레디스 Pub/Sub 서버의 해당 사용자 채널에 새 위치 발행
- 레디스 Pub/Sub 채널에 발행된 새로운 이베느는 모든 구독자에게 브로드캐스트
- 각 구독자의 웹소켓 연결 핸들러가 친구의 위치 변경 이벤트를 수신
- 웹소켓 연결 핸들러가 위치한 웹소켓 서버는 사용자 간 거리를 새로 계산
- 검색 반경을 넘지 않는다면 단말로 변경된 정보 전송 / 넘으면 보내지 않음

API 설계

웹소켓

[서버] 주기적인 위치 정보 갱신

- request: 위도, 경도, 시각

[클라이언트] 갱신된 친구 위치 수신

- response: 친구 위치 데이터, 변경된 시각

[서버] 웹소켓 초기화

- request: 위도, 경도, 시각
- response: 친구들의 위치 데이터

[클라이언트] 새 친구 구독

- request: 친구 ID
- response: 가장 최근 위도, 경도, 시각

[클라이언트] 구독 해지

- request: 친구 ID

HTTP

친구 추가/삭제

데이터 모델

위치 정보 캐시

활성 상태 친구의 가장 최근 위치

key	value

사용자 ID	{위도, 경도, 시각}

왜 데이터베이스를 사용하지 않는가?

- **현재 위치**만을 이용하기 때문
- 영속성 보장 필요가 없음
- Redis의 읽기/쓰기 연산 속도가 매우 빠르고 TTL을 지원하기 때문

위치 이동 이력 데이터베이스

Column	Description
user_id	사용자 ID
latitude	위도
longitude	경도
timestamp	시각

- 쓰기 부하 감당
- 수평적 규모 확장 가능
- 샤딩이 필요하다면 사용자 ID를 기준으로 샤딩하자!

▼ 4️⃣ 상세 설계

> 🛠️ 규모를 늘려 나가면서 병목과 해결책을 찾는 데 집중해보자

규모 확장성

웹소켓 서버

- **유상태** 서버라 기존 서버를 제거할 때 주의해야 한다.
  - 기존 연결 종료 후 노드 실제거
    - 1) 로드밸런서가 인식하는 노드 상태를 `연결 종료 중(draining)`으로 변경해두기
    - 2) 해당 서버로 더이상 새로운 웹소켓 연결이 만들어지지 않음
    - 3) 시간이 흘러 모든 연결이 종료되면 서버를 제거하기

- 유상태 서버 클러스터 규모 확장을 위해 좋은 로드밸런서가 있어야 함.

  그리고 대부분의 클라우드 로드밸런서는 잘 해줍니다.

클라이언트 초기화

`모바일 클라이언트 - 웹소켓 클러스터`는 지속성 웹소켓 연결을 맺는다.

웹소켓 연결이 초기화되면 어떤 일이 일어날까?

1. 클라이언트는 모바일 단말의 위치 정보를 전송한다.
2. 웹소켓 연결 핸들러가 정보를 받는다.
3. 위치 정보 캐시에 보관된 사용자의 위치를 갱신한다.
4. 위치 정보는 연결 핸들러 내 변수에 저장한다.
5. 사용자 데이터베이스에서 사용자의 모든 친구 정보를 가져온다.
6. 위치 정보 캐시에 일괄 요청을 보내 모든 친구 위치를 한번에 가져온다.
7. 친구 위치 각각에 대해 사용자와의 거리를 계산하여 주변에 있는 친구의 정보를 웹소켓 연결을 통해 클라이언트에 반환한다.
8. 웹소켓 서버는 각 친구의 레디스 서버 Pub/Sub 채널을 구독한다.
9. 사용자의 현재 위치를 레디스 Pub/Sub 서버의 전용 채널을 통해 모든 친구에게 전송한다.

사용자 데이터베이스

사용자 상세 정보, 친구 관계 데이터

사용자 ID를 기준으로 샤딩하여 수평적 규모 확장을 하자!

위치 정보 캐시

최대 메모리 사용량

- 각 키에 TTL이 설정되어 있으므로 일정 한도내로 유지 가능
- 1000만명의 사용자가 동시에 활성화, 위치 정보 보관에 각 100byte가 필요해도 1대 서버로 캐시 가능
- **단, 30초마다 변경된 정보를 전송하기 때문에 초당 연산 수 약 334K** 버거워요

샤딩

- 사용자 ID를 기준으로 여러 서버에 샤딩 가능
- 부하 분산 가능

가용성

- 각 샤드에 보관하는 위치 정보를 대기(standby) 노드에 복제
- 주(primary) 노드에 장애가 발생하면 대기 노드를 승격시켜 장애 시간을 줄임

레디스 Pub/Sub 서버

> ### Why Redis?
>
> 채널을 만드는 비용이 아주 저렴하기 때문 !
> - 구독자가 없는 채널로 전송된 메시지는 버려지는데, 이 과정에 서버에 거의 부하가 가지 않음
> - 채널 1개 유지 위해 구독자 관계 추적을 위한 해시 테이블, 연결 리스트 필요. 아주 소량의 메모리 사용
> - 오프라인 사용자의 경우 채널 생성 이후 CPU 자원을 전혀 사용하지 않음

동작 방식

1. 주변 친구 기능을 사용하는 모든 사용자에게 채널 1개를 부여한다.
2. 주변 친구 기능 사용자의 앱은 초기화 시 모든 친구의 채널과 구독 관계 설정
    a. 친구의 상태 변경에 따라 구독/구취 할 필요가 없어서 단순해짐
    b. **더 많은 메모리를 사용함**에 유의할 것

🔎 **얼마나 많은 서버가 필요한가?**

1. 메모리 사용량
    - 모든 사용자에게 채널 1개 할당 = 10억 * 10% = **1억 개**
    - 한 사용자의 주변 친구 100명이 기능을 사용, 1명 추적을 위해 20byte 포인터를 저장해야 한다면
        - 모든 채널 저장 필요 메모리 = 1억 * 20byte = **200GB**

    ⇒ 100GB 메모리를 설치할 수 있는 레디스 서버를 사용한다면 **2대**가 필요하다.
2. CPU 사용량

- Pub/Sub 서버가 구독자에게 전송해야 하는 위치 정보 업데이트 양 = 초당 1400만 건
- 보수적으로 1대가 감당 가능한 구독자 수를 10만이라고 가정
    - 필요한 서버 수 = 1400만 / 10만 = **140대**

⇒ **레디스 Pub/Sub 서버의 병목은 메모리가 아니라 CPU 사용량이다 !**

⇒ **즉, 분산 레디스 클러스터가 필요하다 !**

⭐ 분산 레디스 Pub/Sub 클러스터

사용자 ID를 기준으로 샤딩하면 되지만… 운영을 매끄럽게 하고 싶어요

서비스 탐색 컴포넌트를 도입하자. (etcd, zookeeper, …)

1. 가용 서버 목록을 유지 / 갱신하는 데 필요한 UI or API

   키: /config/pub_sub_ring

   값: [”p_1”, “p_2”, “p_3”, “p_4”] 레디스 Pub/Sub 서버로 구성된 해시 링
2. 웹소켓 서버로 하여금 값에 명시된 레디스 Pub/Sub 서버에서 발생한 변경 내역 구독

레디스 클러스터 속성

1. Pub/Sub 채널에 전송되는 메시지는 채널의 모든 구독자에게 전송되고 나면 삭제된다.
2. Pub/Sub 서버는 채널에 대한 상태 정보를 보관한다.

⇒ 레디스 Pub/Sub 서버 클러스터는 유상태 서버 클러스터로 취급하는 게 바람직하다.

불필요한 크기 변화를 피하도록 오버 프로비저닝하자.

레디스 클러스터 규모 조정 시 유의사항

1. 클러스터의 크기를 조정하면 많은 채널이 다른 서버로 이동하면서 엄청난 재구독 요청이 발생할 것이다.
2. 재구독 요청을 처리하다보면 클라이언트가 보내는 위치 정보 변경 메시지 처리가 누락될 수 있다.
3. 서비스의 상태가 불안정해질 수 있으므로 클러스터 크기 조정은 시스템 부하가 가장 낮은 시간을 골라서 해야 한다.

클러스터 크기 조정 절차

1. 새로운 링 크기 계산
    a. 크기가 늘어나는 경우 새 서버를 준비

2. 해시 링의 키에 매달린 값을 새로운 내용으로 갱신
3. 대시보드 모니터링

운영 유의사항

1. Pub/Sub 서버에 장애 발생 시 모니터링 소프트웨어가 온콜 엔지니어에게 경보 발송
2. 온콜 담당자는 경보를 받으면 서비스 탐색 컴포넌트의 해시 링 키에 매달린 값을 갱신하여 장애가 발생한 노드를 대기 노드와 교체
3. 교체된 사실을 모든 웹소켓 서버에 통지
4. 각 웹소켓 서버는 실행 중인 연결 핸들러에게 새 Pub/Sub 서버의 채널을 다시 구독하도록 알림
5. 각 연결 핸들러는 모든 채널을 해시 링과 대조하여 새 서버로 구독 관계 재설정

친구 관련 이슈

친구 추가

1. 새 친구가 추가됨
2. 클라이언트에 연결된 웹소켓 서버의 연결 핸들러에게 알림(callback)
3. 새 친구의 Pub/Sub 채널을 구독함
4. 웹소켓 서버는 해당 친구가 활성화 상태인 경우 가장 최근 위치 정보를 응답 메시지에 담아 보냄

친구 삭제

1. 친구를 삭제함
2. 클라이언트에 연결된 웹소켓 서버의 연결 핸들러에게 알림(callback)
3. 해당 친구의 Pub/Sub 채널 구독을 취소함

친구가 많은 사용자

친구가 많은 사용자는 시스템 성능에 영향을 줄까?

최대 친구 수에 상한(5000명)이 있고, 친구 관계는 양방향이라고 가정

- 친구들의 위치가 변경될 때 생기는 부하는 웹소켓 서버가 나눠 처리하므로 핫스팟 문제 발생 X
- 많은 친구를 둔 사용자의 채널이 존재하는 Pub/Sub 채널의 경우 조금 더 많은 부하 발생, 막대한 부담까진 아니다~

+) 주변의 임의 사용자

1. 지오해시에 따라 구축된 Pub/Sub 채널 풀 만들기
2. 해당 격자 안에 있는 모든 사용자는 해당 격자에 할당된 채널 구독하기

+) 경계 부근에 있는 사용자 처리를 위해 주변 격자를 담당하는 채널도 구독하기

레디스 Pub/Sub의 대안

얼랭? (Erlang)

마이너한 언어라서 채용이 쉽지 안하 😭

장점

- 고도로 분산된 병렬 애플리케이션을 위해 고안됨
- 경량 프로세스. 프로세스 생성 비용은 리눅스에 비해 매우 저렴
- 여러 서버로 분산하기 쉬움
- 운영 부담이 낮음
- 배포 도구가 강력함

우리 설계안에서는 …

- 웹소켓 서비스를 얼랭으로 구현
- 레디스 Pub/Sub 클러스터를 분산 얼랭 애플리케이션으로 대체
- 친구 관계에 있는 사용자의 얼랭 프로세스와 구독 관계 설정 및 위치 변경 내역 수신

▼ 5️⃣ 마무리

사용자의 위치 정보 변경 내역을 다른 사람에게 어떻게 효율적으로 전달할 것인가?

- 웹소켓 : 클라이언트-서버 간 실시간 통신을 지원
- 레디스 : 위치 데이터의 빠른 I/O 지원
- 레디스 Pub/Sub : 사용자 위치 정보 변경 내역을 모든 친구에게 전달하는 라우팅 계층

규모가 커질수록 각 컴포넌트는 어떻게 대응할 수 있을까?

친구가 많은 사용자에게 발생할 수 있는 성능 문제는 뭐가 있을까?

토론 주제

- 클라우드 로드밸런서는 어떻게 유상태 서버 클러스터의 규모 확장을 지원할까요?