6주차

7장

분산 시스템에서 사용될 유일 ID 생성기 설계

Auto increment 속성이 설정된 관계형 데이터베이스의 기본 키를 쓰면 되겠다고 생각할 수 있지만 이 접근법은 분산환경에서 사용할 수 없다.

1단계 : 문제 이해 및 설계 범위 확정

- ID는 유일해야 한다.
- ID는 숫자로만 구성되어있어야 한다.
- ID는 발급 날짜에 따라 정렬 가능해야한다.
- ID는 64비트로 표현될 수 있는 값이어야한다.
- ID는 발급 날짜에 따라 정렬 가능해야 한다.
- 초당 10,000개의 ID를 만들 수 있어야 한다.

2단계 : 개략적 설계안 제시 및 동의 구하기

- 다중 마스터 복제 (multi-master replication)
- UUID(Universally Unique Identifier)
- 티켓 서버(ticket server)
- 트위터 스노플레이크 접근법

다중 마스터 복제

다중 마스터 복제는 데이터베이스의 auto_increment 기능을 활용한다.

현재 사용 중인 데이터벰이스 서버의 수를 k라고 하자. 다음 ID 값을 구할 때 이전 키에서 k 만큼 더해주는 방법이다. 서버마다 시작하는 숫자를 달리하면 다음에 생성되는 키들도 전부 유니크해질 것.

장점

- 데이터베이스 수를 늘리면 초당 생산 가능 ID 수도 늘어난다

한계

- 여러 데이터 센터에 걸쳐 규모를 늘리기 어렵다.
- ID의 유일성은 보장되겠지만 그 값이 시간 흐름에 맞추어 커지도록 보장할 수는 없다.
- 서버를 추가하거나 삭제할 때도 잘 동작하도록 만들기 어렵다.

UUID

- UUID: 컴퓨터 시스템에 저장되는 정보를 유일하게 식별하기 위한 128비트 짜리 수
- UUID 값은 충돌 가능성이 지극히 낮다.

장점

- UUID를 만드는 것은 단순하다.
- 동기화 이슈도 없다.
- 각 서버가 자기가 쓸 ID를 알아서 만드는 구조이므로 규모 확장이 쉽다.

단점

- ID가 128비트로 길다. 이번 문제에서는 쓸 수 없음
- ID를 시간순으로 정렬할 수 없다.
- ID에 숫자가 아닌 값이 포함될 수 있다.

티켓 서버

이 방법은 auto_increment 기능을 갖춘 데이터베이스 서버를 티켓서버로 하고, 중앙 집중형으로 하나만 사용하는 것이다.

장점

- 숫자로만 구성된 ID를 쉽게 만들 수 있다.
- 구현하기 쉽고, 중소 규모 애플리케이션에 적합하다.

단점

- 티켓서버가 단일장애지점이 된다. 결국 이 이슈를 해결하기 위해 티켓 서버를 여러 대 준비하게 되는데, 그렇게 하면 데이터 동기화같은 새로운 문제가 발생한다.

트위터 스노플레이크 접근법

- 타임스탬프가 제일 앞에 있기 때문에 시간에 따른 정렬이 가능하다.
- 타임스탬프는 밀리초, 즉 69년만 측정 가능하다.
- 일렬번호가 의미하는 것 : 한 서버가 1ms 내에 최대 2^12개의 ID를 만들 수 있다.
    - 동시성이 적은 어플리케이션의 경우 일렬번호를 줄일 수 있다.
    - 또 데이터센터와 서버 수도 적절히 예측해 조절한다.

3단계 : 상세 설계

개략적 설계를 진행하면서 우리는 분산 시스템에서의 ID 생성기를 설계하는 데 사용할 다양한 기술적 선택지를 봤다. 문제의 요구사항을 만족시키기 위해 트위터의 snowflake 기법을 사용해서 상세 설계.

4단계 : 마무리

ID 생성기 구현에 쓰일 수 있는 4가지 전략

- 다중 마스터 복제
- UUID
- 티켓 서버
- 트위터의 snowflake

그리고 이번 문제의 요구사항에 적합한 snowflke를 선택했다.

설계 이후에 추가로 논의할 수 있는 주제들은 다음과 같다.

- 시계 동기화: 이번 설계를 진행하면서 우리는 ID 생성 서버들이 전부 같은 시계를 사용한다고 가정하였다. 하지만 이런 가정은 하나의 서버가 여러 코어에서 실행될 경우 유효하지 않을 수 있다. 여러 서버가 물리적으로 독립된 여러 장비에서 실행되는 경우에도 마찬가지다.
- 각 섹션의 길이 최적화: 예를 들어 동시성이 낮고 수명이 긴 애플리케이션이라면 일련번호 절의 길이를 줄이고 타임스탬프 절의 길이를 늘리는 것이 효과적일 수 있다.
- 고가용성: ID 생성기는 필수 불가결 컴포넌트 이므로 아주 높은 가용성을 제공해야 할 것이다.

8장 URL 단축기 설계

1단계 : 문제 이해 및 설계 범위 확장

개략적 추정

- 쓰기 연산 : 매일 1억 개의 단축 URL 생성
- 초당 쓰기 연산 : 1억/24/3600 = 1160
- 읽기 연산 : 읽기 연산과 쓰기 연산 비율을 10:1 이라고 하자. 그 경우 읽기 연산은 초당 11600회 발생.
- 축약 전 URL 길이 100이라 하자.
- 10년 동안 필요한 저장 용량은 3650억 x 100바이트 = 36.5TB 이다.

2단계 : 개략적 설계안 제시 및 동의 구하기

API 엔드포인트

URL 단축기는 기본적을 두 개의 엔드포인트 필요

- URL 단축용 엔드포인트 : 새 단축 URL을 사용하고자 하는 클라이언트는 엔드포인트에 단축할 URL 을 인자로 실어서 POST 요청을 보내야 함.
- URL 디렉션용 엔드포인트 : 단축 URL에 대해서 HTTP 요청이 오면 URL로 보내주기 위한 용도의 엔드포인트.

URL Redirection

- 301 (Paermanently Moved)
  - 영구적으로 URL이 리다이렉션되기 때문에 브라우저는 반환된 URL을 캐싱한다.
  - 그리고 나중에 요청을 보낼 때 브라우저 단에서 URL을 바꿔서 보낸다.
  - 이거 테스트 환경에서 잘 못 쓰면 많이 고생한다.
- 302 (Found)
  - 일시적으로 리다이렉션된다
  - 매번 URL 단축기 서버를 거쳐 가기 때문에 트래픽 분석하기 좋다.

URL 단축

- 긴 URL을 해싱해서 짧은 URL을 만든다.

- 짧은 URL을 긴 URL로 복원해야 한다.

3단계 : 상세 설계

데이터 모델

- 해시 테이블 : 초기 전략으로는 괜찮지만 메모리는 제한이 있고 비싸다.
- RDBMS : 단축 URL, 원래 URL 쌍으로 저장.

해시 함수

- 해시 값 길이
  - 0-9, a-z, A-Z 의 총 62개의 문자 사용.
  - 길이 정하려면 62^n ≥ 3650억인 n의 최소값 찾아야 한다.
- 해시 후 충돌 해소
  - n=7 일 때, 원래 URL을 7글자로 줄여야함.
  - CRC32, MD5, SHA-1와 같은 해시 함수 이용.
  - 7글자 이상일 경우, 처음 7글자만 사용 → 충돌 확률 증가.
  - 충돌 발생시 충돌 해소될 때까지 사전에 정한 문자열을 해시값에 덧붙임.

    → 단축시 한 번 이상 DB에 질의해야하므로 오버헤드 크다.

  → 블룸필터를 사용한다!
- base-62 변환
  - 해시값에 쓸 수 있는 62개 값을 이용하여 진법 변환.
  - 유일성 보장 ID 생성기 필요. 충돌도 아예 불가능.

URL 단축기 상세 설계

1. 긴 URL 입력 받음.
2. DB에 해당 URL 있는지 검사.
3. DB에 있다면 해당 URL에 대한 단축 URL 만든적 있는지 검사. 이후 클라이언트에 단축 URL 반환.
4. DB에 없는 경우 해당 URL은 새로 접수된 것이므로 유일한 ID생성.
5. 62진법 변환 적용.

6. ID, 단축 URL, 원래 URL로 새 DB 레코드 만든 후 단축 URL 클라이언트에 전달.