7장. 분산 시스템을 위한 유일 ID 생성기 설계

분산 시스템을 위한 유일 ID에 대한 질문을 받으면, 일반적으로 "auto_increment 속성이 설정된 관계형 데이터베이스의 기본키를 쓰면 되지 않을까?"라는 생각을 한다.
하지만 분산 환경에서는 이런 접근 방법이 통하지 않을뿐더러, 여러 데이터베이스 서버를 쓰는 경우 지연 시간을 낮추기가 무척 힘들다.

1단계 | 문제 이해 및 설계 범위 확정

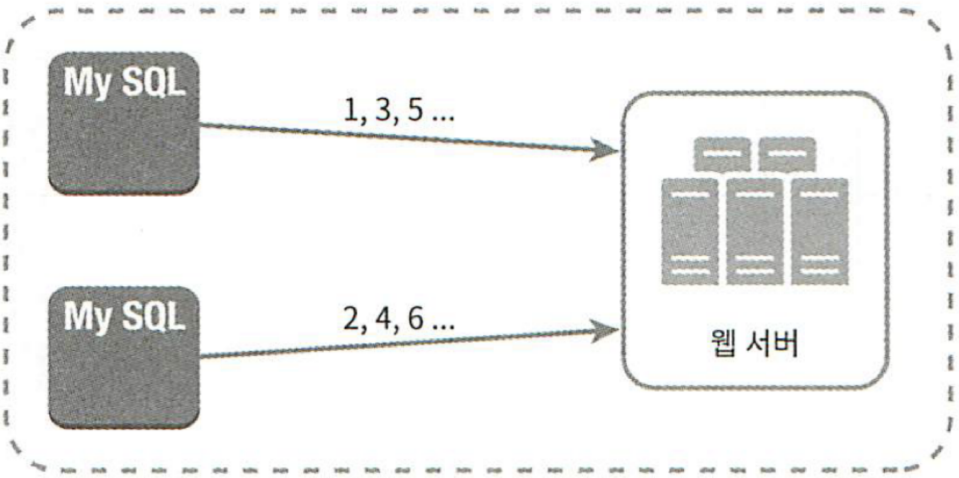
- ID는 유일해야 한다. (unique)
- ID는 숫자로만 구성되어있어야 한다.
- ID는 발급 날짜에 따라 정렬 가능해야한다.
- ID는 64비트로 표현될 수 있는 값이어야한다.
- ID는 발급 날짜에 따라 정렬 가능해야 한다.
- 초당 10,000개의 ID를 만들 수 있어야 한다.

2단계 | 개략적 설계안 제시 및 동의 구하기

분산 시스템에서 유일성이 보장되는 ID를 만드는 방법은 여러 가지다. 우리는 다음과 같은 선택지를 살펴볼 것 이다.

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

다중 마스터 복제(multi-master replication)



- 이 접근법은 데이터베이스의 auto_increment 기능을 활용하는 것이다. 다만 다음 ID의 값을 구할 때 1만큼 증가시켜 얻는 것이 아니라, k만큼 증가시킨다.
- 여기서 k는 현재 사용 중인 데이터베이스 서버의 수다.
  위의 그림의 예제를 보자. 어떤 서버가 만들어 낼 다음 아이디는, 해당 서버가 생성한 이전 ID 값에 전체 서버의 수 2를 더한 값이다.
- 이렇게 하면 규모 확장성 문제를 어느 정도 해결할 수 있는데, 데이터베이스 수를 늘리면 초당 생산 ID 수도 늘릴 수 있기 때문이다.

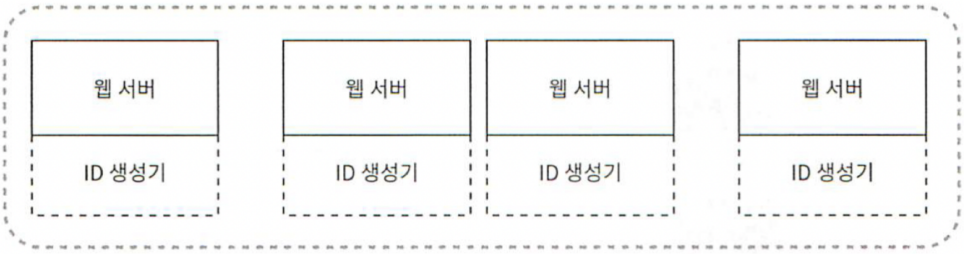
단점

- 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(Universally Unique Identifier)

UUID는 컴퓨터 시스템에 저장되는 정보를 유일하게 식별하기 위한 128비트짜리 수다. UUID 값은 충돌 가능성이 지극히 낮다.

- UUID값은 09c93e62-50b4-468d-bf8a-c07e1040bfb2와 같은 형태
- UUID는 서버 간 조율 없이 독립적으로 생성 가능



- 이 구조에서 각 웹 서버는 별도의 ID 생성기를 사용해 독립적으로 ID를 만들어낸다.

장점

- UUID를 만드는 것은 단순하다. 서버 사이의 조율이 필요 없으므로 동기화 이슈도 없다.
- 각 서버가 자기가 쓸 ID를 알아서 만드는 구조이므로 규모 확장도 쉽다.

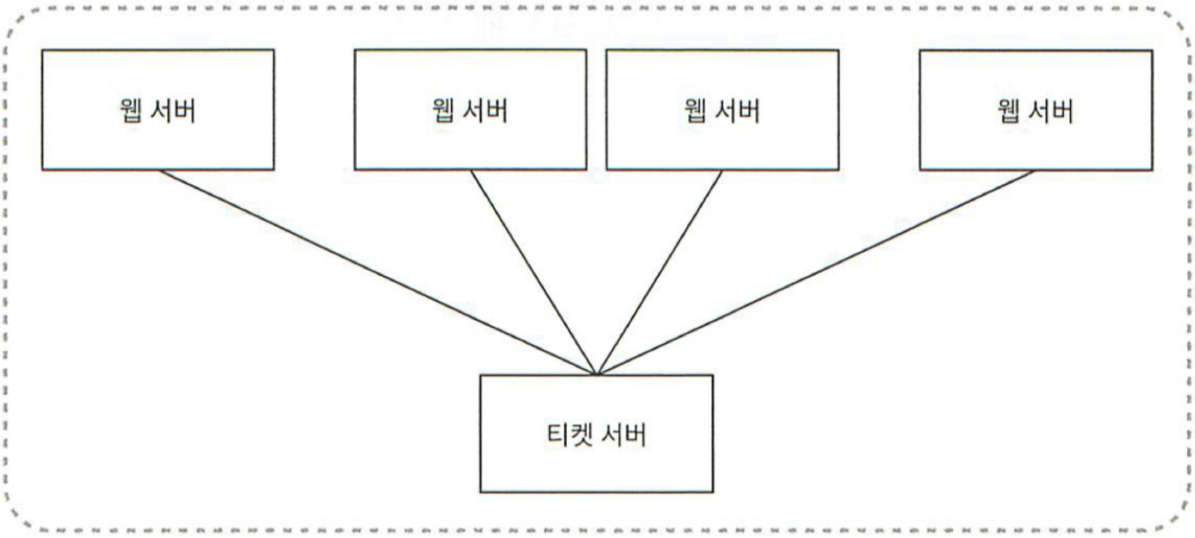
단점

- ID가 128비트로 길다. 이전의 ID 요구사항은 64비트이다.
- ID를 시간순으로 정렬할 수 없다.
- ID에 숫자가 아닌 값이 포함될 수 있다.

티켓 서버(ticket server)

이 아이디어의 핵심은 auto_increment 기능을 갖춘 데이터베이스 서버, 즉 티켓 서버를 중앙 집중형으로 하나만 사용하는 것이다.

- 플리커는 분산 기본 키를 만들어 내기 위해 이 기술을 이용



장점

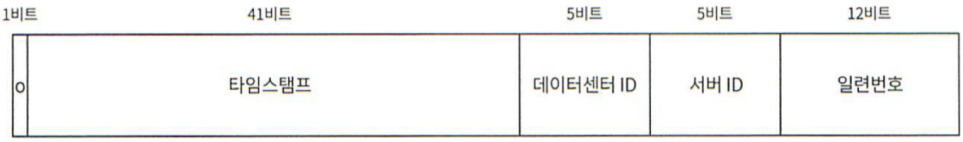
- 유일성이 보장되는 숫자로만 구성된 ID를 쉽게 만들 수 있다.
- 구현하기 쉽고, 중소 규모 애플리케이션에 적합하다.

단점

- 티켓 서버가 SPOF(Single-Point-of-Failure)가 된다. 이 서버에 장애가 발생하면, 해당 서버를 이용하는 모든 시스템이 영향을 받는다. 이 이슈를 피하려면 티켓 서버를 여러 대 준비해야 한다. 하지만 그렇게하면 데이터 동기화 같은 새로운 문제가 발생할 것이다.

트위터 스노플레이크(twitter snowflake) 접근법

- 트위터는 스노플레이크(snowflake)라고 부르는 독창적인 ID 생성 기법을 사용
- 이 접근법은 **분할 정복(divide and conquer)** 을 적용



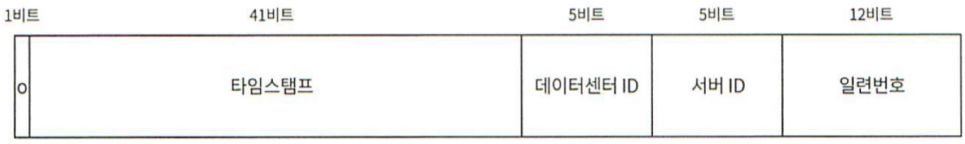
위의 그림은 우리가 생성할 64비트 ID의 구조

- 사인(sign) 비트 : 1비트를 할당한다. 지금으로서는 쓰임새가 없지만 나중을 위해 유보해 둔다. 음수와 양수를 구별하는 데 사용할 수 있을 것이다.
- 타임스탬프 : 41비트를 할당한다. 기원 시각(epoch) 이후로 몇 밀리초가 경과했는지를 나타내는 값이다. 본 설계안의 경우에는 기원 시각으로 트위터 스노플레이크 구현에서 사용하는 값 1288834974657(Nov 04, 2010, 01:42:54 UTC에 해당)을 이용할 것이다.

- 데이터센터 ID: 5비트를 할당한다. 따라서 25 = 32개 데이터센터를 지원할 수 있다.
- 서버 ID : 5비트를 할당한다. 따라서 데이터센터당 32개 서버를 사용할 수 있다.
- 일련번호 : 12비트를 할당한다. 각 서버에서는 ID를 생성할 때마다 이 일련번호를 1만큼 증가시킨다. 이 값은 1밀리초가 경과할 때마다 0으로 초기화 된다.

3단계 | 상세 설계

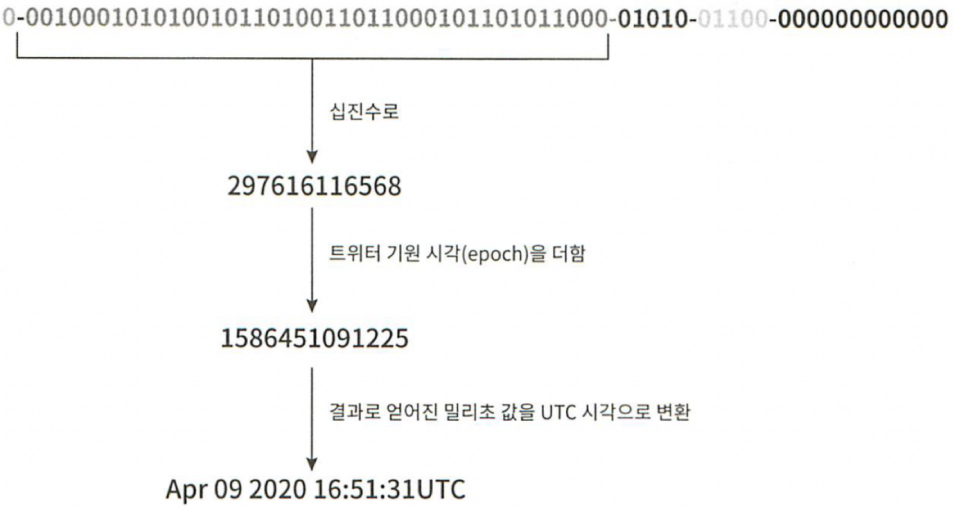
개략적 설계를 진행하며 우리는 분산 시스템에서 사용할 유일성 보장 ID 생성기를 설계할 수 있는 다양한 기술적 선택지를 살펴보았다. 그 가운데 스노플레이크 접근법을 사용하여 보다 상세한 설계를 진행해본다.



- 데이터센터 ID와 서버 ID는 시스템이 시작할 때 결정되며, 일반적으로 시스템 운영 중에는 변경 X
  - 데이터센터 ID나 서버 ID를 잘못 변경하게 되면 ID 추돌이 발생할 수 있으므로, 그런 작업을 해야 할 때는 신중

타임 스탬프

- 타임스탬프는 앞서 살펴본 ID 구조에서 가장 중요한 41비트를 차지
- 타임스탬프는 시간이 흐름에 따라 점점 큰 값을 갖게 되므로, 결국 ID는 시간 순으로 정렬 가능



- 위의 그림은 앞서 살펴본 ID 구조를 따르는 값의 이진 표현 형태로부터 UTC 시각을 추출하는 예제
- 이 방법을 역으로 적용하면 어떤 UTC 시각도 상술한 타임스탬프 값으로 변환 가능

일련 번호

- 12비트이므로, 212 = 4096개의 값을 가질 수 있음
- 어떤 서버가 같은 밀리초 동안 하나 이상의 ID를 만들어 낸 경우에만 0보다 큰 값을 가짐

4단계 | 마무리

추가로 논의할 수 있는 사항

- 시계 동기화(clock synchronization)
  - 하나의 서버가 여러 코어에서 실행될 경우나 여러 서버가 물리적으로 독립된 여러 장비에서 실행되는 경우, ID 생성 서버들이 전부 같은 시계를 사용한다고 할 수 없음 → 이러한 문제를 해결하는 가장 보편적인 수단으로는 NTP(Network Time Protocol)이 있음
- 각 절(section)의 길이 최적화
  - 예를 들어 동시성이 낮고 수명이 긴 애플리케이션에서는 일련번호 절의 길이를 줄이고 타임스탬프 절의 길이를 늘리는 것이 효과적
- 고가용성(high availability)
  - ID 생성기는 필수 불가결 컴포넌트이므로 아주 높은 고가용성을 제공해야함