6주차 - 이진성

7장 분산시스템을위한유일 ID 생성기설계

auto_increment와 같은 ID 새엉기는 분산시스템의 ID 생성 문제를 효과적으로 해결할 수 없고 또한 생성시의 delay를 낮추기 어렵다

문제 이해와 및 설계 범위 확정

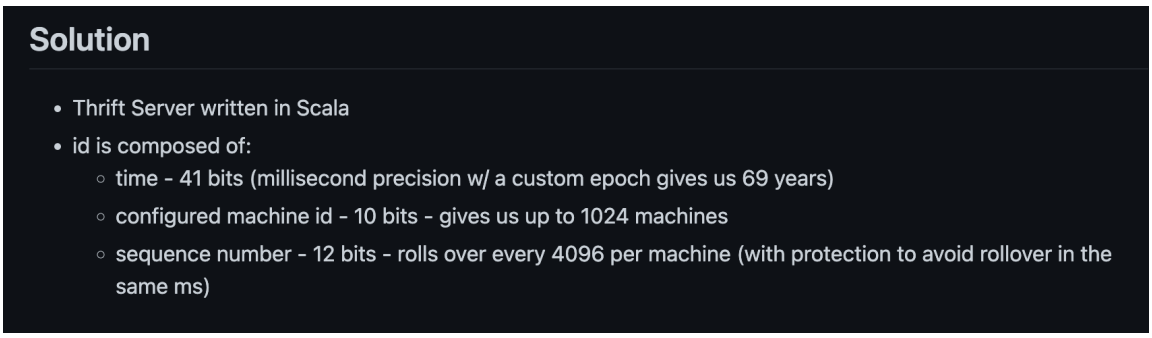
적절한 질문

- ID가 가져야하는 특서
- 순서성 : 완전하게 지킬 필요는 없음
- ID의 형식 : 숫자로만 구성
- 시스템 규모 : 초당 1만개

개략적인 설계안 제시 및 동의 구하기

- 다중 마스터 복제
  - auto increment를 사용하지만, 마스터 클러스터의 개수(K)만큼 증가함.
  - 정확하게 순서를 지켜 생성 가능
  - 여러 데이터 센터의 규모를 늘리기 어려움
  - ID 유일성은 보장되지만, 시간에 흐름에 맞추어 커지도록 보장할 수 없다
  - 서버를 추가해도 삭제할 때도 잘 동작학도록 만들기 어려움
- UUID
  - UUID가 충돌이 날 확률을 50%로 만들려면 초당 10억개씩 생성해야할 정도로 어려움
  - 장점
    - 동기화 이슈가 없어서 그냥 생성 가능
    - 규모의 확장도 무한함
  - 단점

        - 키의 길이가 128비트
        - 시간순 정렬 불가능
        - 숫자가 아닌 값이 포함될 수 있음
- 티켓 서버
    - Flickr - 분산 기본 키를 만들기 위한 기술
    - 장점
        - 유일 성이 보장되는 숫자로되는 ID 구성 가능
        - 구현하기 쉽고 중소 규모 애플리케이션애 적합
    - 단점
        - SPOF가 될 수 있음 - 티켓서버가 장애나면 전체 시스템이 멈춘다. → 새로운 문제 야기
- 트위터의 snowflake 접근법
    - GitHub - twitter-archive/snowflake at b3f6a3c6ca8e1b6847baa6ff42bf72201e2c2231
    - https://github.com/anthonynsimon/timeflake



    - 각개격파(divide conquer)방식을 통해 키를 여러 섹션으로 나누고 생성하는 방식
    - [1Bit - Sign | 41bit Timestamp | 5bit - datacenter | 5bit - server id | 12bit - Seq ID]

상세 설계

- 타임스탬프 - 41bit = [현재 시간 - twitter 기원 시각 밀리초 기준] ⇒ 69년
- 일련번호 - 12bit ⇒ 4096개

마무리

추가로 논의해야할 부분

- 시계 동기화
- 다중 코어 장비에서는 완전하게 시계를 동기화할 수 없다. NTP는 이를 해결할 수 있는 보편적 수단이다.
- 각 절 길이 최적화 : 동시성을 낮추고 수명이 긴 애플리케이션을 만들 있다.
- 고가용성 ID는 필수 불가결(mission critical) 컴포넌트이므로 아주 높은 가용성을 제공해야한다.

8장 - URL 단축기설계

설계 범위 확정

1. URL 단축기의 동작 예시
2. 트래픽 규모
3. 길이
4. URL 포함 문제 제한
5. URL 시스템에서 지우거나 갱신

추정

1. 매일 1억개의 URL생성
2. 초당 쓰기 연산 1160
3. URL 단축 서비스 10년간 - 3650억 레코드
4. URL 평균 길이 100
5. 10년치 데이터의 용량 36.5TB

개략적인 설계안 제시 및 동의 구하기

API 엔드포인트

1. URL 단축용 API
2. 원래 URL을 보내주기 위한 엔드포인트

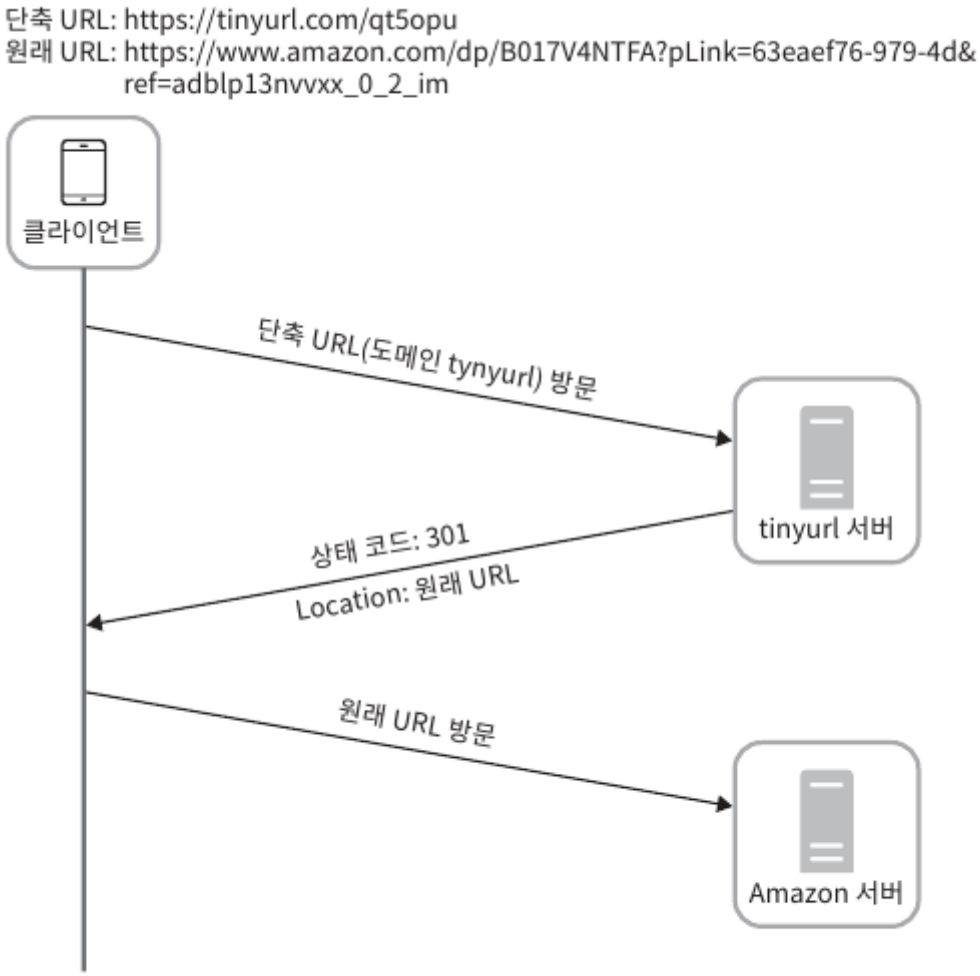


그림 8-2

URL 리디렉션

- 301 : 이 응답은 해당 URL에 대한 처리 책임을 영구적으로 Location 헤더의 값으로 이 전되었다는 뜻 - 이후에는 브라우저에 캐시될 수 있음
- 302 : 일시적으로 Location 헤더로 보냄

또한 기본적으로 url에 대해서 해시 함수가 있어야한다.

상세 설계

데이터 모델

- 인메모리 데이터는 비싸기 때문에 실제 시스템에 쓰이기엔 곤란하다.'
- 테이블 { id, shortUrl, longUrl } 형태로 구성한다.

해시함수

- 원래 URL → 단축 URL로 변환하는데 쓰임
- value range : [0-9, a-z, A-Z] 로 구성됨
- length : 7이면 충분할 듯?
- 해시함수는 기본적으로 7자리 이상인데, 어떤 함수던 앞의 7자리만 따서 쓰면 괜찮음.

해시함수 충돌 해소

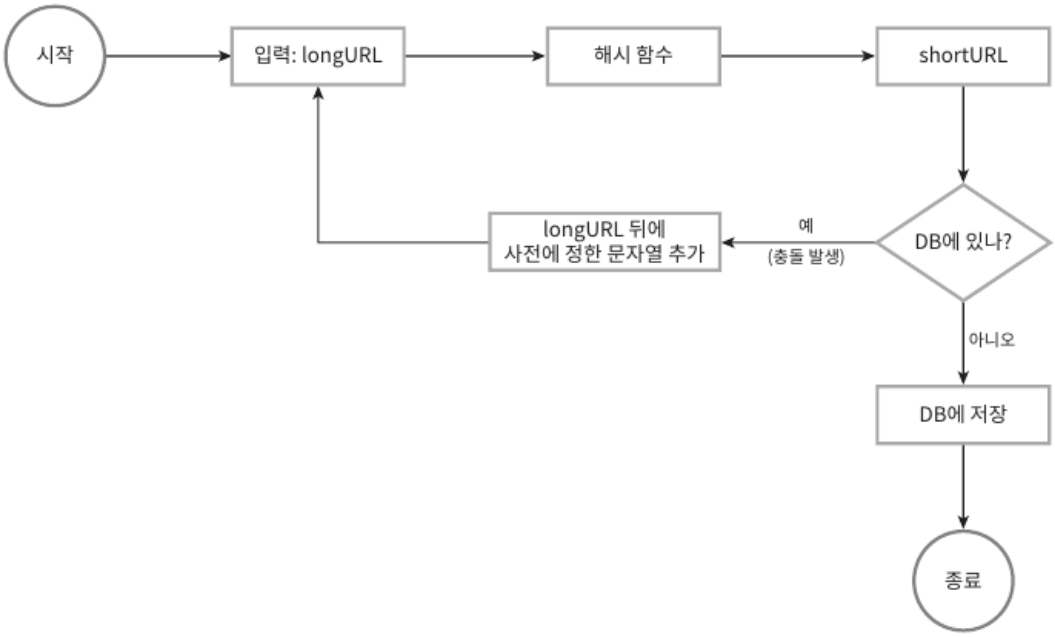


그림 8-5

- 해시함수는 특성상 충돌이 발생할 수 있는데 충돌이 발생한 경우를 DB 질의를 이용하여 체크할 수 있음.
- 충돌이 발생한경우 사전의 정의된 문자열을 longurl위에 붙혀 다시 해시함수를 돌림.
    - salt를 사용하지 않은 경우? → salt를 사용하면 항상 다른 해시 결과가 나오므로 같은 url에 대해서도 계속해서 다른 결과를 발생시킬 수 있음.

base-62활용법

흔히 사용되는 방법으로 62진법으로 shorturl을 표현하는 것, url을 기반해서 만들지 않고(너무 충돌이 쉽게 발생할 수 있으므로) → 유일성보장 id generator를 이용하여 만듬

- 이런 경우 long url에 대해서 여려벌의 shorturl이 발생할 수 있으나, hash collision은 나지 않으므로 해소할 필요가 없어짐
- 하지만 longurl에 대해서도 index를 걸거나 하는 방식으로 사전에 질의하여 같은 url을 반환할 수 있도록 개선이 가능함 → 비교적 많은 데이터 스토어 용량 요구

URL shortener design

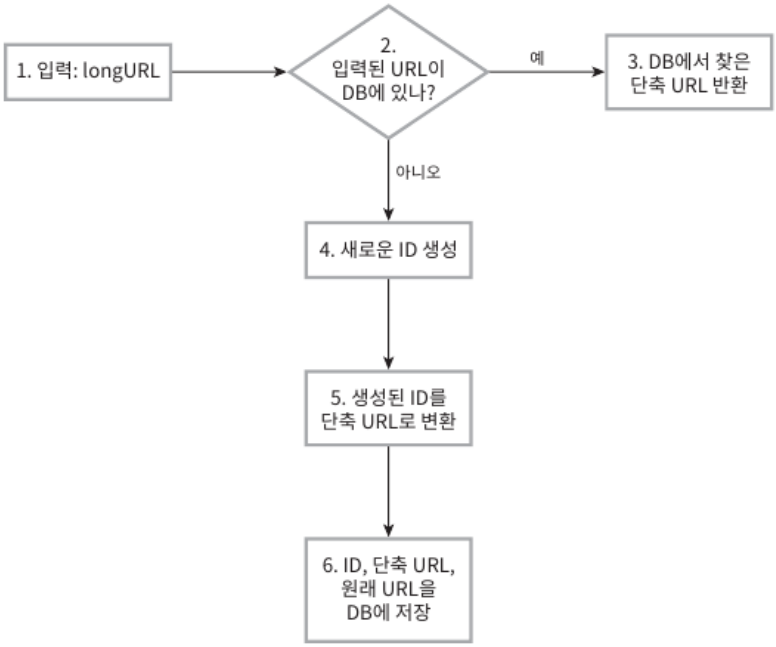


그림 8-7

특별히 볼 내용은 없고, 있으면 반환하고 없으면 생성하여 반환한다는 것

URL redirection design

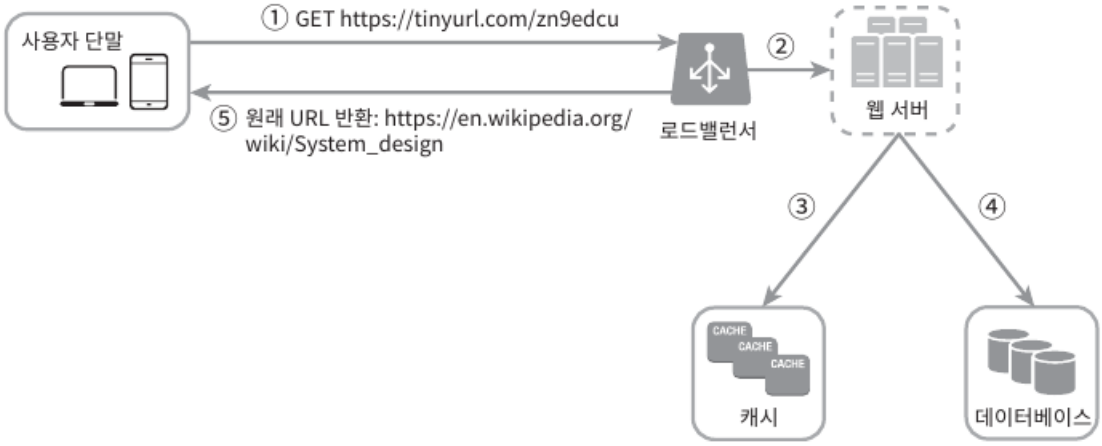


그림 8-8

1. 캐시와 데이터베이스를 활용하여 반환 함

논의 해볼만한 주제

- ratelimiter
- stateless
- sharding
- analytics
- consistency, availability, durability에 대해서..