분산 시스템을 위한 유일 ID 생성기 설계

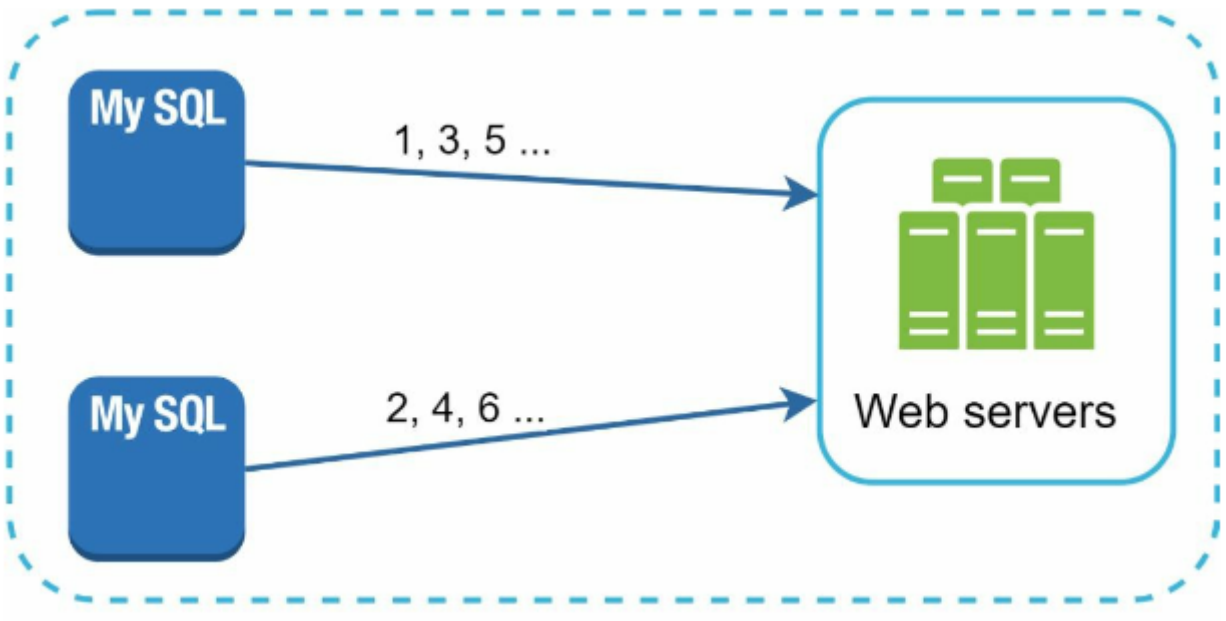
요구 사항

- 유일한 ID
- 숫자로만 구성되는 값
- 64비트로 표현될 수 있는 값
- 발급 날짜에 따라 정렬 가능한 값
- 초당 10,000개의 ID를 만들 수 있어야한다

개략적 설계안 제시 및 동의 구하기

다중 마스터 복제

- Auto Increment 기능을 활용하여, 현재 사용중인 데이터베이스 서버의 수 k 만큼씩 증가시킴
- 서버의 수를 증가시킴으로써 초당 생성가능한 ID 수를 늘릴 수 있음
    - 이를 통해 규모 확장성 문제를 어느 정도 해결 가능

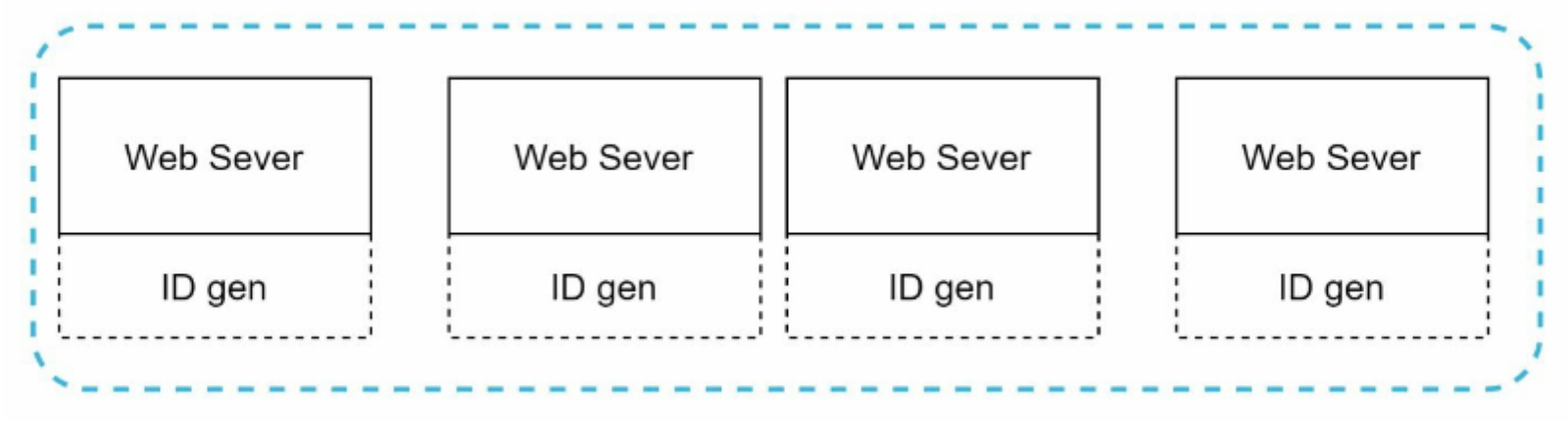


단점

- 여러 데이터 센터에 걸쳐 규모를 늘리기 어렵다
- ID의 유일성을 보장되지만, 그 값이 시간 흐름에 맞추어 커지도록 보장할 수 없음
- 서버를 추가하거나 삭제할 때, 잘 동작하도록 만들기 어렵다

UUID

- UUID는 컴퓨터 시스템에 저장되는 정보를 유일하게 식별하기위한 128bit 크기의 수
- 충돌 가능성은 지극히 낮으며, 초당 10억개의 UUID를 100년동안 계속 만들지 않는 이상 중복의 가능성이 50% 이하이다
- 서버 간 조율 없이 ID 생성이 가능하다



장점

- UUID 생성은 단순하며, 동기화 이슈도 없다
- 각 서버가 알아서 ID를 만드는 구조이므로, 규모 확장도 쉽다

단점

- ID가 128bit로 길다
- 시간순으로 정렬할 수 없다
- 숫자가 아닌 값이 ID에 포함될 수 있다

티켓 서버

- auto increment 기능을 가진 하나의 서버에서 ID를 return 해주는 방식이다

장점

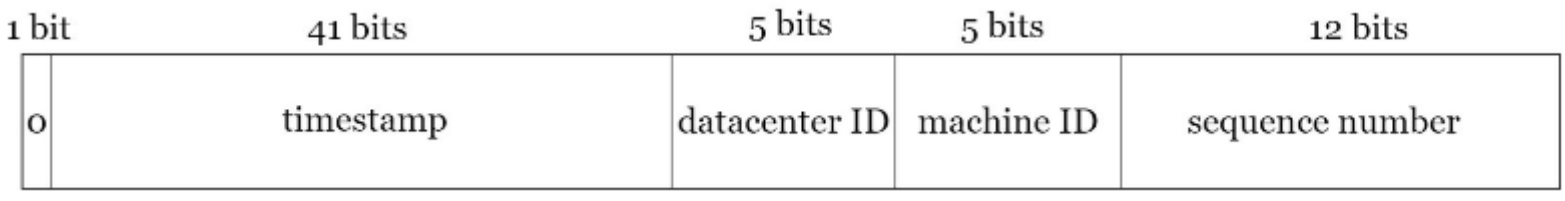
- 유일성이 보장되는 오직 숫자로만 구성된 ID를 쉽게 만들 수 있다
- 구현하기 쉽고, 중소 규모의 애플리케이션에 적합

단점

- SPOF가 될 수 있다
- SPOF를 피하기 위해 여러 서버를 준비한다면, 동기화의 문제가 생긴다

트위터 스노우플레이크 접근법

트위터 스노우 플레이크 접근법은 하나의 아이디를 여러 섹션으로 나누어, 의미를 부여한다

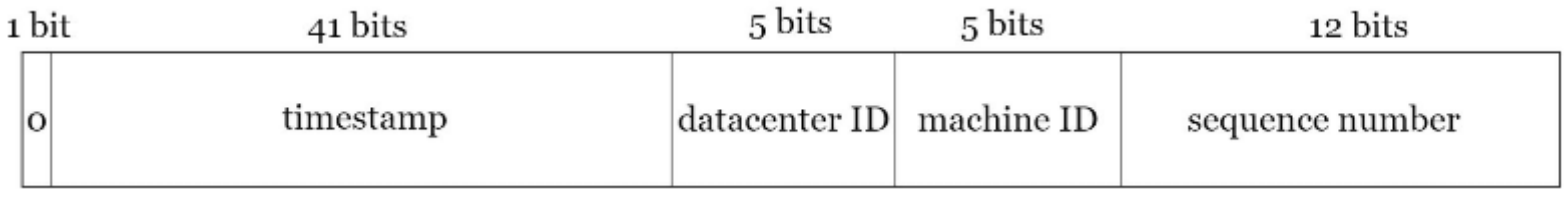


- Sign bit : 1비트를 할당한다. 음수와 양수를 구별하는데 사용한다. 현재는 필요없지만 추후를 위해 남겨둔다.
- Timestamp : 41비트를 할당한다. 기원 시각 (epoch) 이후로 몇 밀리초가 경과했는지를 나타내는 값이다.
- 데이터센터 ID : 5비트를 할당한다. 32개의 데이터센터를 지원가능.
- 서버 ID : 5비트를 할당한다. 각 데이터센터당 32개의 서버를 지원가능.
- 일련번호 : 12비트를 할당한다. 각 서버에서는 ID를 생성할 때마다, 일련번호를 1만큼 증가시키고 1밀리초가 경과할 때 마다 0으로 초기화 시킨다.

상세 설계

트위터 스노우플레이크 접근법을 사용하여, 상세 설계를 진행

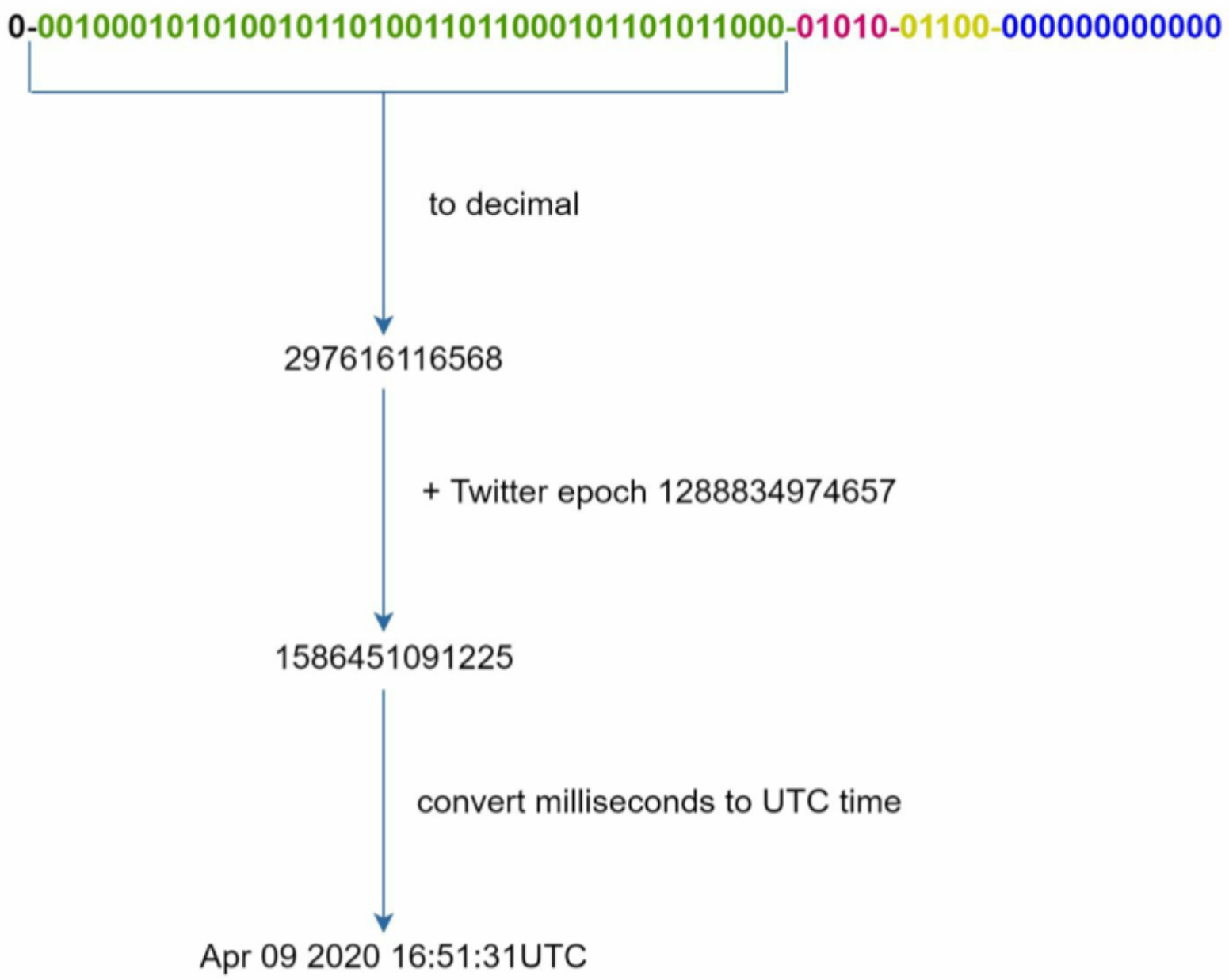
구조



- 데이터 센터 ID와 서버 ID는 시스템 시작시 결정되며, 운영되는 동안 변경되지 않는다
- 데이터 센터나 서버 ID를 잘못 변경하면, 충돌이 발생할 수 있으므로 신중히 변경해야한다
- 타임 스탬프나 일련번호는 ID 생성기가 동작하는 동안 만들어지는 값이다

타임 스탬프

- 타임스탬프는 41bit를 차지하고 있음
- 시간이 흐름에 따라 점점 큰 값을 가지기 때문에 정렬 가능
- 아래는 UTC 시각을 추출하는 예제



- 41bit는 최대 69년을 표현할 수 있으므로, 69년이 지난다면 기원 시각 또는 ID 체계를 변경해야한다

일련 번호

- 일련번호는 12bit이기 때문에, 4096개를 만들 수 있다
- 밀리초동안 하나 이상의 ID를 만드는 경우에만, 0보다 큰 값을 갖게될 것이다

마무리

스노플레이크 방식은 모든 요구사항을 만족하면서, 분산 환경에서 규모 확장이 가능했기 때문에 선택했다.

아래 사항들을 추가로 살펴보면 좋을 것.

- 시계 동기화
  - 이번 설계를 진행하면서 우리는 ID 생성 서버들이 전부 같은 시계를 사용한다고 가정하였다
  - 이런 가정은 하나의 서버가 여러 코어에서 실행될 경우 유효하지 않을 수 있다
  - 여러 서버가 물리적으로 독립된 여러 장비에서 실행되는 경우에도 마찬가지
  - 이러한 문제를 해결하기 위해 NTP (Network Time Protocol) 이 가장 보편적인 해결 수단이니 참고해보자
- 각 section의 길이 최적화
  - 동시성이 낮고, 수명이 긴 애플리케이션이라면 일련 번호 section의 길이를 줄이고 타임스탬의 길이를 늘리는 것이 좋을 것이다
- 고가용성
  - ID 생성기는 필수이므로, 높은 고가용성을 지원해야한다

URL 단축기 설계

우리는 이번 섹션에서 다음과 같은 기능을 가진 URL 단축기를 설계할 것이다.

- URL 단축 : 주어진 긴 URL을 훨씬 짧게 줄인다.
- URL 리디렉션 : 축약된 URL로 HTTP 요청이 오면 원래 URL로 안내한다.
- 높은 가용성과 규모 확장성, 장애 감내가 요구된다.

개략적 추정

- 쓰기 연산 : 매일 1억개의 단축 URL을 생성
- 초당 쓰기 연산 : 1억 / 24 / 3600 = 1,160개
- 읽기 연산 : 읽기와 쓰기의 비율은 10 : 1 이라고 대략 가정하자. 그럼 대략 11,160 개.
- URL 단축 서비스를 10년간 운영해야한다면, 1억 x 365 x 10 = 3,650억개의 레코드를 보관
- 축약 전 URL의 평균 길이를 100이라 가정
- 10년 동안 필요한 용량은 365,000 억바이트 = 36.5TB 이다.

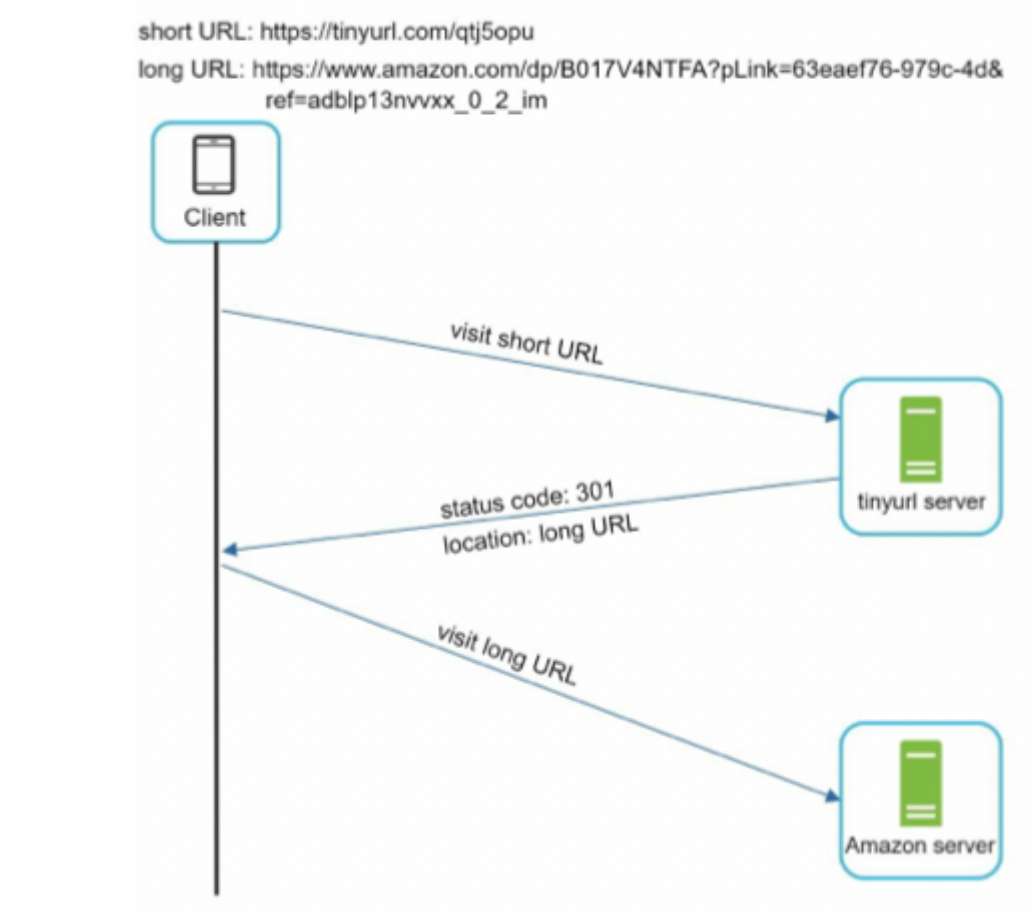
개략적 설계안 제시 및 동의 구하기

API 엔드 포인트

- 단축용 엔드포인트
  - 새 단축 URL을 생성하고자하는 클라이언트는 POST 요청을 통해 단축할 URL을 인자로 실어서 보내야 함.
  - POST `/api/v1/data/shorten`
  - 인자 : { longUrl : longUrlString }
  - 반환 : 단축된 URL
- 리디렉션용 엔드포인트
  - 단축 URL에 대해서 HTTP 요청이 오면, 원래 URL로 보내주기 위한 용도의 엔드포인트.
  - GET `/api/v1/shortUrl`
  - 반환 : HTTP 리디렉션 목적지가 될 원래 URL

URL 리디렉션

브라우저에 단축 URL을 입력하면 발생하는 일을 아래에서 살펴보자.



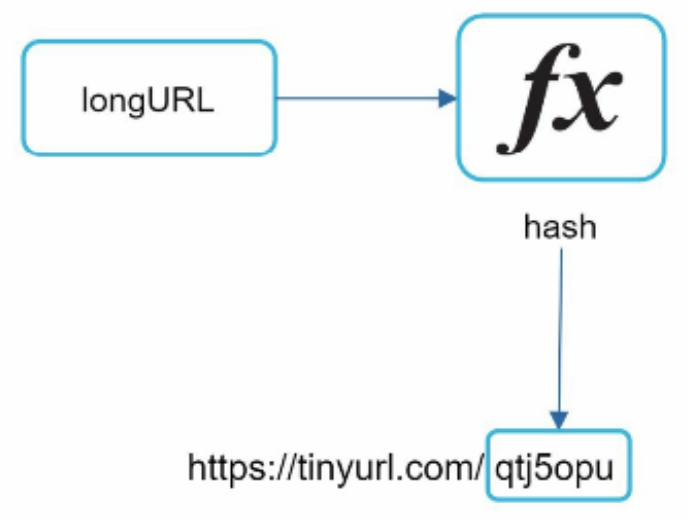
- 301 Permanently Moved :
  - 이 응답은 해당 URL에 대한 요청 처리 책임이 영구적으로 Location 헤더에 반환된 URL로 이전되었다는 응답이다.
  - 영구적으로 이전되었기 때문에, 브라우저에서 캐시한다.
- 302 Found
  - 이 응답은 일시적으로 Location 헤더에 반환된 URL로 이전되었다는 응답이다.
  - 클라이언트는 언제나 단축 URL 서버에 요청을 먼저 보내고, 리디렉션된다.

서버 부하를 줄이는 것이 중요하다면 301 Permanently moved를 사용하는 것이 좋다. 하지만 트래픽 분석이 필요할 때는, 302 Found를 쓰는 게 유리하다.

- URL 리디렉션을 구현하는 가장 직관적인 방법은 해시테이블을 이용해, < 단축 URL, 원래 URL > 의 쌍을 저장하는 방법이다.
  - 원래 URL : hashTable.get(단축 URL)
  - 301 또는 302 응답 Location 헤더에 원래 URL을 넣어 전송

URL 단축

단축 URL을 `https://tinyurl.com/{hashValue}` 라고 한다면, 중요한 없은 긴 URL을 hashValue에 대응시키는 hash function을 찾는 것이다.

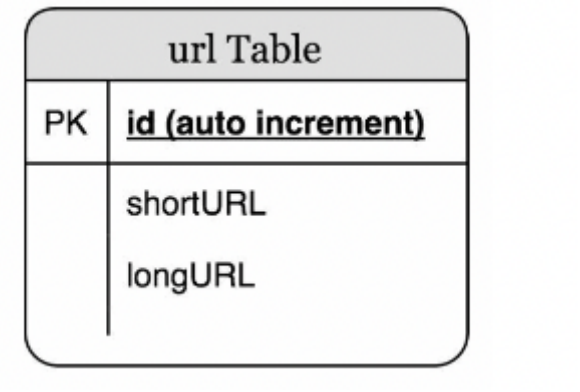


- 해시함수는 다음 요구 사항을 만족해야한다.
  - 긴 URL이 다른 값이면, 해시값도 달라야한다.
  - 계산된 해시값은 원래 값으로 원복될 수 있어야한다.

상세 설계

데이터 모델

해시테이블에 저장하기에는 문제가 있다. 그렇기에 < 단축 URL, 원래 URL > 을 관계형 데이터베이스에 저장하는 것이 나을 것이다.



- 간단하게 id, shortURL, longURL 을 저장하는 url table을 설계했다.

해시 함수

해시 함수는 원래 URL을 단축 URL로 변환하는데 사용될 것이다.

해시 값 길이

- hashValue는 [0-9, a-z, A-Z]의 문자들로 구성되며, 사용할 수 있는 문자의 개수는 62개이다.
- hashValue를 정하기 위해, 62^n ≥ 3650억인 n의 최소값을 정해야한다.
- n = 7 이면, 3.5조개의 URL을 만들 수 있기 때문에 hashValue를 7로 두자.

해시 후 충돌 해소

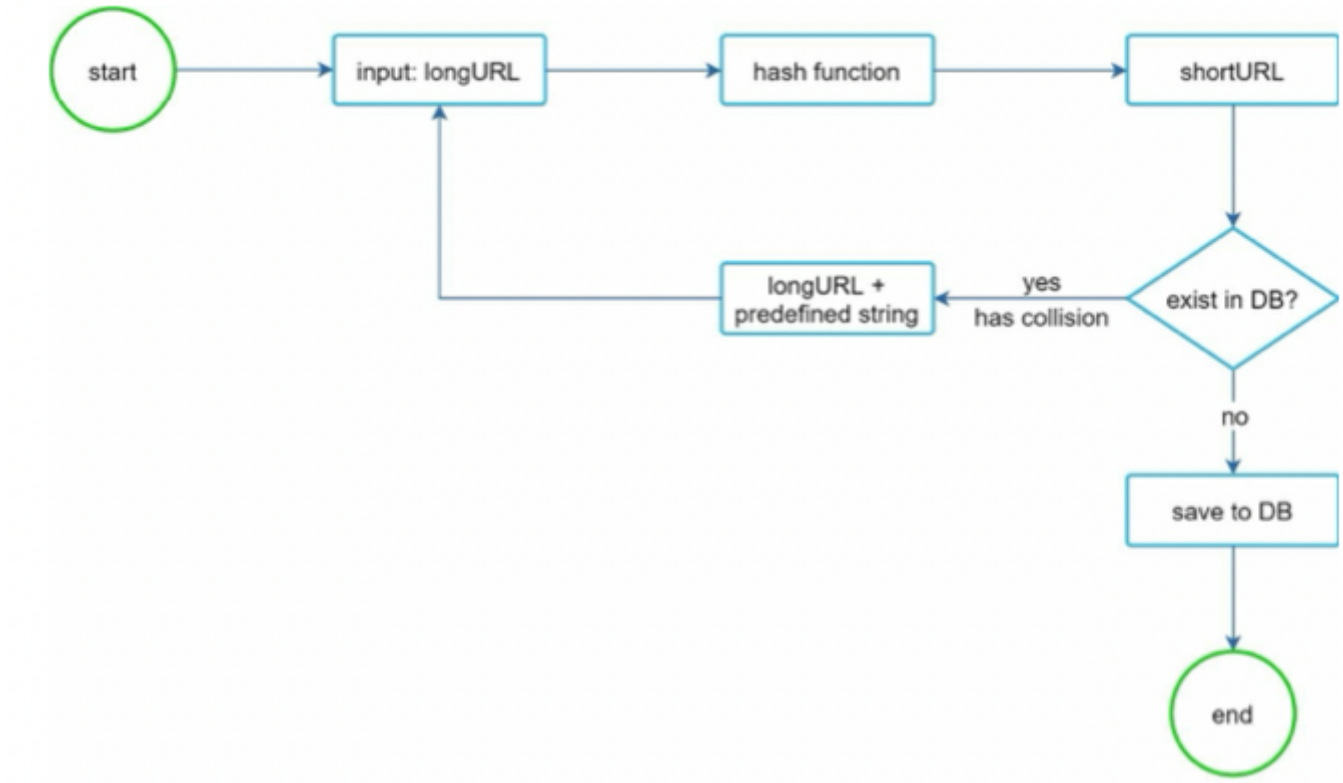
긴 URL을 줄이려면, 원래 URL을 7글자로 줄이는 해시함수가 필요하다.

- 자주 사용되는 해시함수는 보통 아래와 같다.

Hash function	Hash value (Hexadecimal)
CRC32	5cb54054
MD5	5a62509a84df9ee03fe1230b9df8b84e
SHA-1	0eeae7916c06853901d9ccbefbfcaf4de57ed85b

하지만, 위의 해시함수들은 가장 짧은 해시값조차도 7보다 길이가 길다.

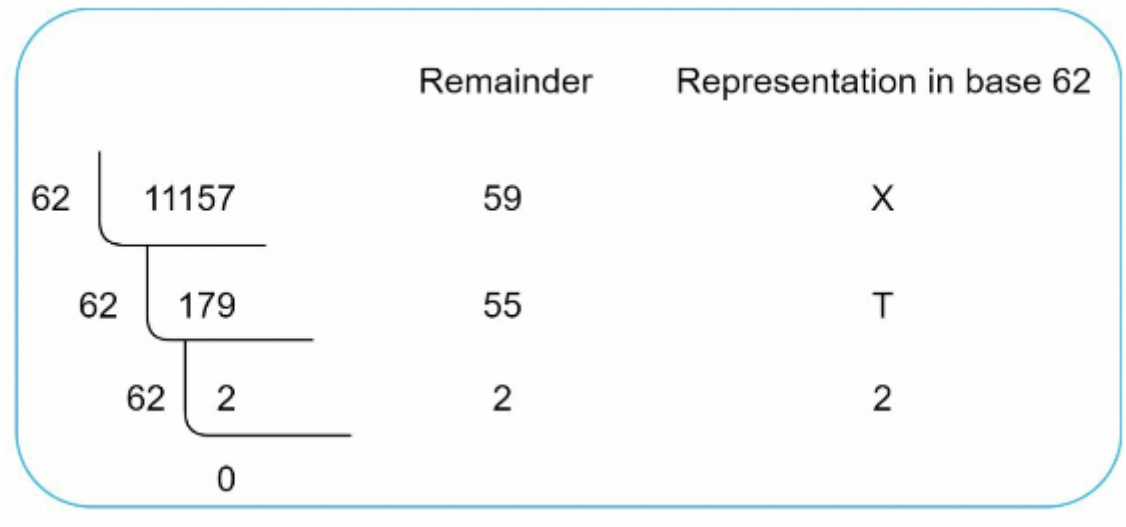
- 이 문제를 해결하는 방법은 계산된 해시 값에서 처음 7개 값만 사용하는 것이다.
- 하지만 충돌 확률 이 높아지기 때문에, 충돌이 발생했을 때 사전에 정한 문자열을 해시값에 덧붙이는 방식을 사용한다.



위 방법을 통해 충돌 해소는 할 수 있지만, 단축 URL을 생성할 때마다 데이터베이스 질의가 필요하므로 성능을 높일 수 있다. 데이터베이스 대신 블룸필터를 사용하면 성능을 높일 수 있다.

base-62 변환

- 62진법은 수를 표현하기 위해 총 62개의 문자를 사용하는 진법이다. 따라서 0은 0으로, 9는 9로, 10은 a로 … 61은 Z로 표현하도록 할 것임.
  - ‘a’ = 1010, ‘Z’ = 6110
- 11157 =2x62^2 + 55x62^1 + 59x62^0 =[2,55,59]->[2,T,X] 이다.



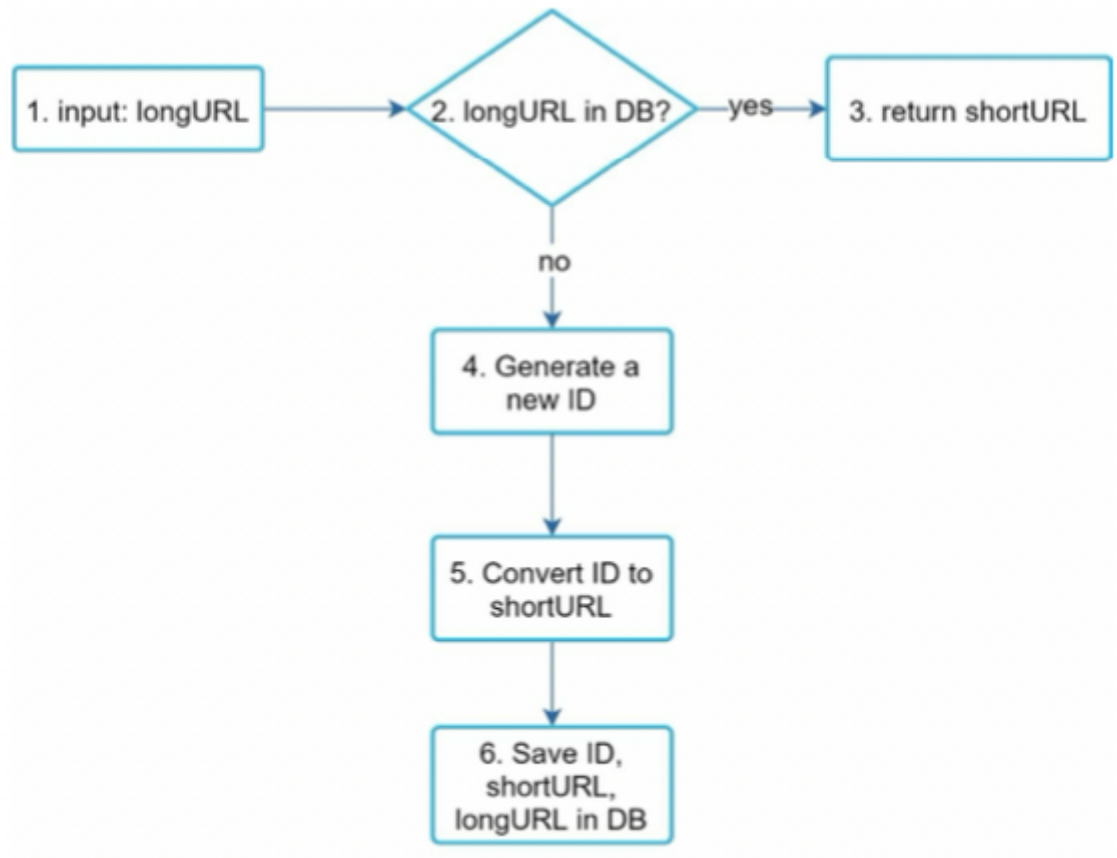
- 위 방식에 따라, 단축 URL은 `https://tinyurl.com/2TX` 와이고 같음.

두 접근법 비교

해시 후 충돌 해소 전략	base-62 변환
단축 URL의 길이가 고정됨.	단축 URL의 길이가 가변적. ID 값이 커지면 같이 길어진다.
유일성이 보장되는 ID 생성기가 필요치 않음.	유일성 보장 ID 생성기가 필요
충돌이 가능해서 해소 전략이 필요.	ID의 유일성이 보장된 후에야 적용 가능한 전략이라 충돌은 아예 불가능.
ID로부터 단축 URL을 계산하는 방식이 아니라서 다음에 쓸 수 있는 URL을 알아내는 것이 불가능.	ID가 1씩 증가하는 값이라고 가정한다면 다음에 쓸 수 있는 URL이 무엇인지 쉽게 알아낼 수 있어서 보안상 문제가 될 소지가 있음.

단축기 상세 설계

62 진법을 통해 단축기 상세 설계를 진행해보면, 아래와 같다.

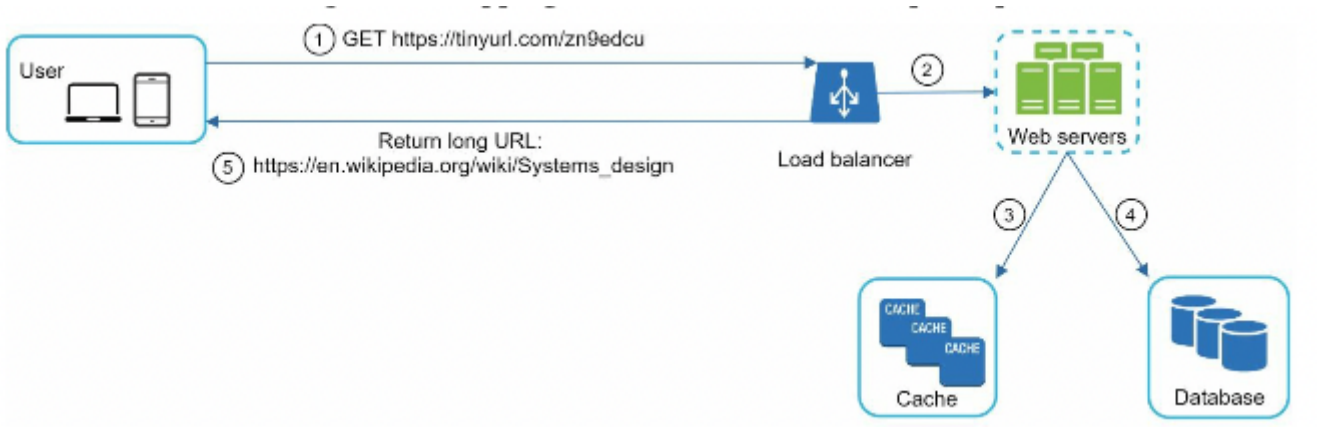


1. 입력으로 긴 URL을 받는다.
2. 데이터베이스에 URL이 있는지 검사.
3. 있다면, URL을 데이터베이스에서 가져와 클라이언트에 반환한다.
4. 데이터베이스에 없는 경우, 유일한 ID를 생성한다. 이 ID를 데이터베이스의 기본 키로 사용된다.
5. 62진법 변환을 적용, ID를 단축 URL로 만든다.
6. ID, 단축 URL, 원래 URL을 데이터베이스의 새 레코드로 만든 후 단축 URL을 클라이언트에 전달한다.

아래의 예시가 생성되는 데이터베이스의 새 레코드이다.

id	shortURL	longURL
2009215674938	zn9edcu	https://en.wikipedia.org/wiki/Systems_design

리디렉션 상세 설계



- 쓰기보다 읽기를 더 자주하는 시스템이기 때문에, < 단축 URL, 원래 URL > 을 쌍으로 캐시에 저장하여 성능을 높였다.
- 로드 밸런스의 동작은 아래와 같이 요약가능하다.

1. 사용자가 단축 URL을 클릭한다.
2. 로드밸런서가 해당 클릭으로 발생한 요청을 웹 서버에 전달한다.
3. 캐시에 있는 경우, 캐시에 바로 꺼내서 클라이언트에 전달한다.
4. 없는 경우 데이터베이스에서 꺼낸다. 만약에 데이터베이스에 전달하지 않는다면, 사용자가 잘못된 URL을 입력한 경우일 것이다.
5. 데이터베이스에서 꺼낸 URL을 캐시에 저장하고, 사용자에게 전달한다.

마무리

- 처리율 제한 장치
  - 엄청난 URL 단축 요청이 올 경우 서버가 무력화될 수 있다. IP 주소 등의 필터링 규칙을 통해 처리율 제한 장치를 두도록 하자.
- 웹 서버의 규모 확장
  - 웹 계층은 무상태 계층이기 때문에, 자유롭게 증설 및 확장할 수 있다.
- 데이터베이스의 규모 확장
  - 데이터베이스를 다중화하거나 샤딩하여 규모 확장성을 달성할 수 있다.
- 데이터 분석 솔루션
  - 성공적인 비지니스를 위해, 어떤 링크를 얼마나 많은 사용자가 클릭했는지 언제 클릭했는지 중요한 정보를 알아낼 수 있다.
- 가용성, 데이터 일관성, 안정성
  - 대규모 시스템이 성공적으로 운영되려면, 위 속성들을 갖추어야한다.