Requirements

→ 5 million new URL shortening requests / month

→ 100 : 1 Read-write ratio

→ Postgres DB.

Calculating storage :-

→ Log URL → https://www.google.com

22 chars.

Lets assume that max length of long URL be 2048 chars.

⇒ $2 \times 2^{10} = 2$ KB

→ Let's make short_url be of length 7 chars. ⇒ 7B

https://www.chota-url.com/ wX9yMpA

7 chars.

mapping in database :-

short_url	long_url
wX9yMp	https://... google.com
⋮	⋮

→ Created_at → 7 Bytes (7 chars)

→ Expiry_at → 7 Bytes (7 chars)

Total storage = 2.021 KB / url shortening

So for 5 mn requests / month

$\Rightarrow$ storage $= (5 \times 10^6) \times (2.021\ KB)$ / month

$= 10.105 \times 10^6\ KB$ / month

$= \underline{10.105\ GB/month}$

$= 10.105 \times 12\ GB/year$

$= 121.26\ GB/year$

$\approx \underline{0.12\ TB/year}$

$\longrightarrow$ Generating 7 chars unique random id

Approach 1: Use B_{10} (base 10)

(0 to 9) = 10 unique values

total combinations for 7 chars long string $= \underbrace{10 \quad 10 \quad 10 \quad \ldots \quad 10}_{7}$

$= 10^7$

$= 10$ mn

$\because$ we have 5 mn req / month

we can only create unique short url for 2 months

$\Rightarrow$ NOT a good approach $\Rightarrow$ Collisions after 2 months

Approach 2: Use B_{62} (base 62)

(0 - 9) = 10 values

(A - Z) = 26 values

(a - z) = 26 values

62 unique values

total combinations = $\underbrace{62 \quad 62 \quad \ldots \quad 62}_{7}$

$= 62^7$

$= 3.52$ trillion

for 5 mn req / month

$\Rightarrow$ 60 mn req / year

total unique combinations = $\frac{3.52 \text{ trillion}}{60 \text{ mn}}$

$= \frac{3.52 \times 10^{13}}{60 \times 10^6} = \frac{352 \times 10^{11}}{6 \times 10^7}$

$= 58.6 \times 10^4$ years

$\approx$ 50,000 years

we can generate unique ids for 50,000 years using approach

Mapping shorturl & longurl in DB :-

Generate 7 random numbers each can range from (0 - 61)

eg: 0 12 61 51 49 7 33

↓ ↓ ↓ ↓ ↓ ↓ ↓

a l 9 Z . . .

"al9ZpQt" → shorturl id

DB

short	long
al9zpQt	→ google.com

for another turn, we can get same random nos for different long_url.
if we simply insert in DB ⇒ COLLISION

So before inserting into DB,

a) check if short-url exists

b) If yes

(i) generate another random nos.

↓

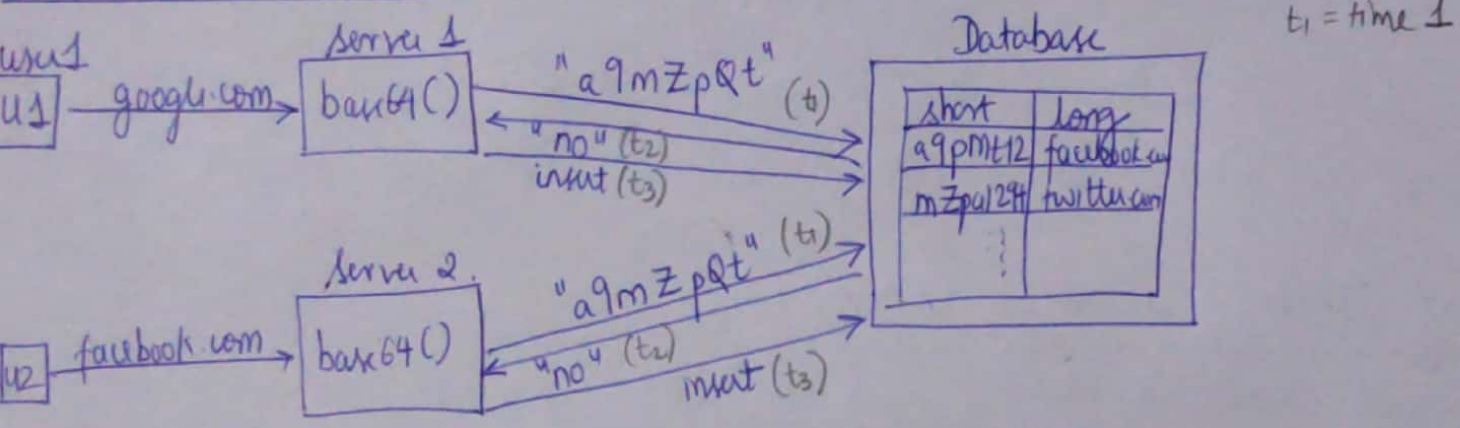
do step (a)

c) else:

(i) insert into db.

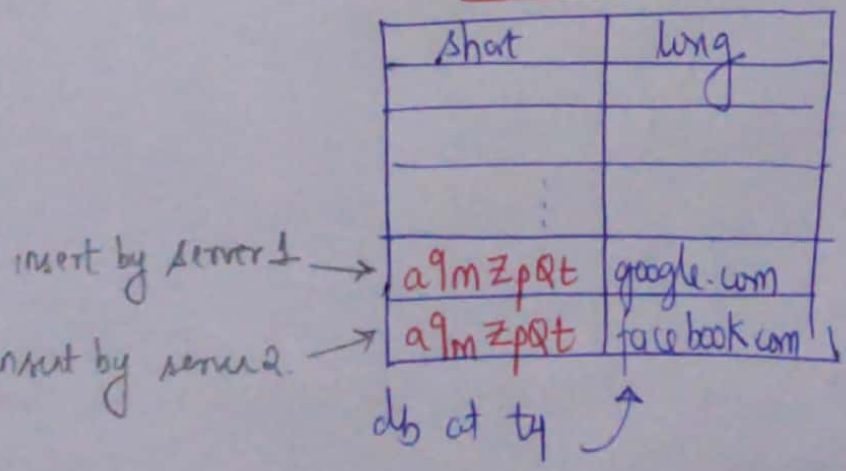
→ This approach works well with centralized DB server, since it will execute write request serially one at a time.

For distributed server :-



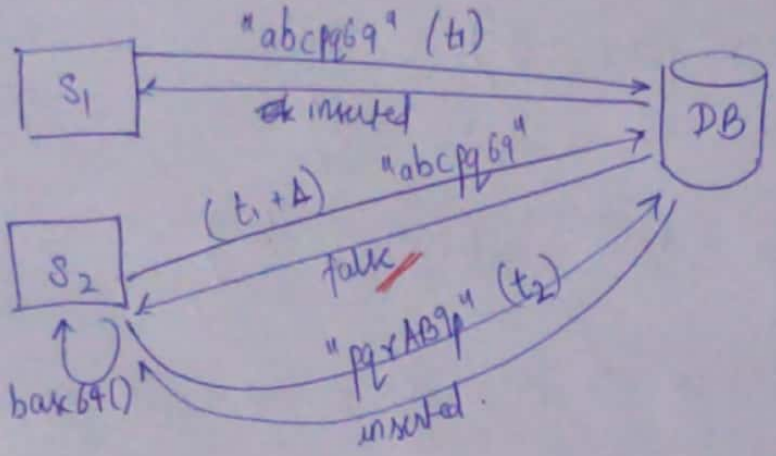
at time t4 db will have two same short urls mapping different long urls

⇒ collision.



To avoid above collision problem, we can use in distributed servers

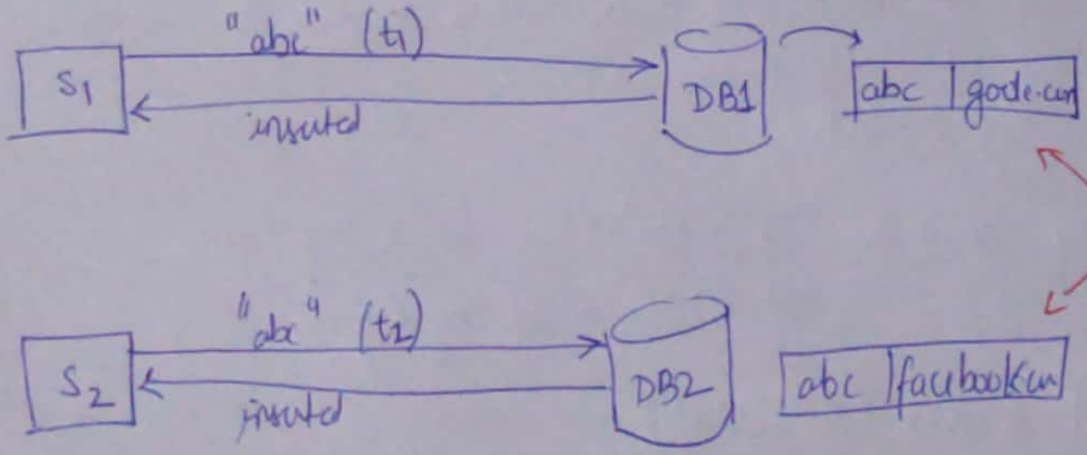
RDBMS function : "Insert if does'nt exist"



Problem 1

The above solution works well when we have CENTRALIZED DB

if we implement distributed DB, chances of data inconsistency can arise



Problem 2

The above solution works well when no. of ~~request~~ entries in db is less. As the entries increases, the shorturl might already present in db & server will again generate base64.random id.

This can be time taking if no of db request increase (i.e to many collisions)

To solve above 2 problems :-

→ We can assign a counter to each of the server
→ Each server will have a value which be reflected in counter & each server will have counter range of 1mn.

e.g: server2 : value = 1 (server number - 1)

counter range = [1,000,000 - 2,000,000)

= [value * 1mn to (1+value) * 1m)

Now this counter will increase it's value by 1 on each WRITE request

Calculating time required for each server to end it's counter range

assumption :- - We have 10+ servers in distb servers.
- write req are uniformly distb across servers by load balancer
- 100 : 1 read-write ratio.
- 5 mn req / month.

$\Rightarrow \frac{5 \times 10^6}{100}$ write req / month $\Rightarrow 5 \times 10^4 \times 12$ write req / year
$\Rightarrow 6 \times 10^5$ write req / year.

If we have M servers (distributed)
then each server will have $\frac{6 \times 10^5}{M}$ write req / year.

Time to complete counter range by each server $= \frac{1 \times 10^6}{\frac{6 \times 10^5}{M}} = 1.66 \times M$ years

if no. of servers = 10 $\Rightarrow$ M = 10 $\Rightarrow$ Each server will take 16 year to end it's counter

This solution suits best for our application. ∵ No collision