

URL shortener

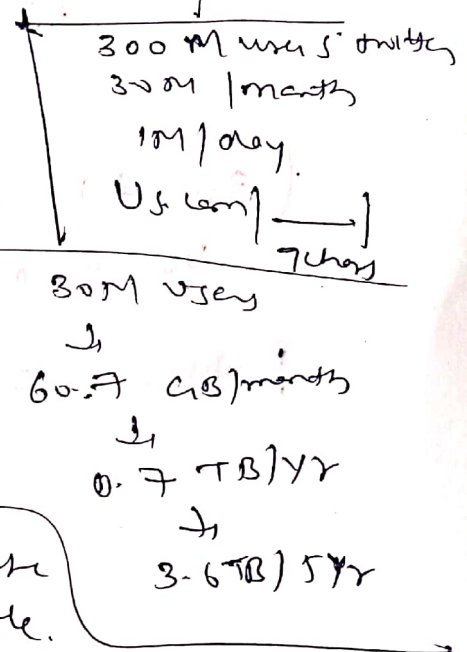
assumptions -

- ① what is the length of the URL?
- ② what is the volume of the traffic
- ③ Is the system single instance or should we scale it?
- ④ Who will use & how will use the system?

Data Capacity Model

- ① Long URL \rightarrow 2KB (2048 chars)
- ② Short URL \rightarrow 17 Bytes (17 chars)
- ③ Created at \rightarrow 7 Bytes (7 chars)
- ④ Expire at \rightarrow 7 Bytes (7 chars)

2.031 KB per entry in the Database table.



We should understand how much read/write a user will do on the above data. So that we can understand which type of d/b to use.

www.us.com (ABC12a) (short URL id)

We can use two algorithms for it

- ① Base62
- ② MD5 hash

decimal no.
or
integer input

\rightarrow Base62 \rightarrow

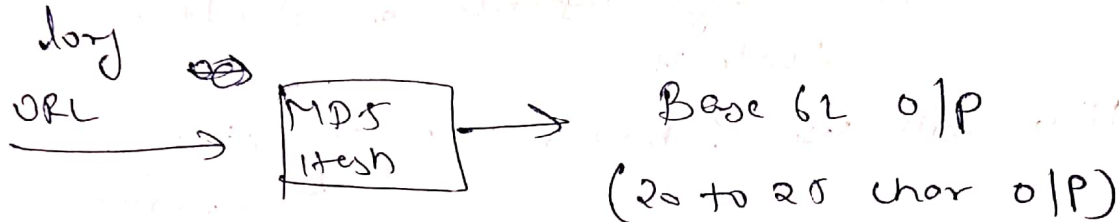
Base62 output
(contains capital
letter, alphabetical
letters, small, no)

In MD5 we also get

Base62 o/p.

but input is mostly different unique string.

We can give our long URL to the md5 hash

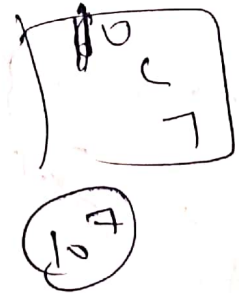


↓ but we need 7 chars
(So take first 7 chars of this output and use it as a unique id for our case)

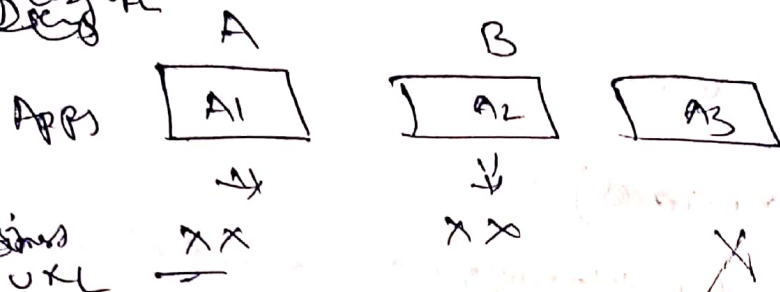
problem → Lot of collision in this case.

two & more can have same o/p but we need unique.

Base 62	VS	Base 10
A-Z - 26		1-10 → 10 ⁷ combinations
a-z - 26		
0-9 - 10 chars		
<u>62 chars</u>		
62 ⁷		
3.5 million		

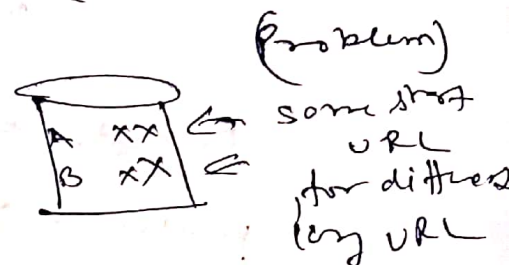


Long URL



Because of multiple servers (Systems)

How to resolve it?



Technique 2

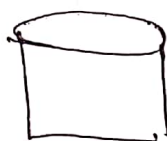
(Use MD5 and use hash)

MD5 (Long-URL)



AB1LCD78D7E2

insert to database



(Collision can happen)

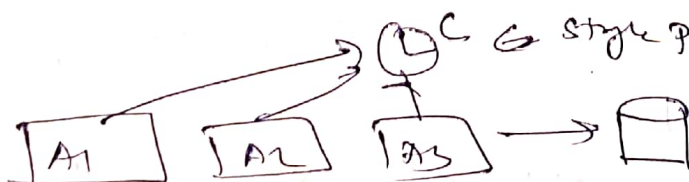
Need to check entry present in DB or not

Can't use MD5

Technique 3 -

Use Counter Variable -

take it as thread protected or synchronized



style point failure if this counter goes down

Technique 4 -

Use Zookeeper (a service having multiple counters and the mixed up architecture)

* Base 62 conversion code -

```
def base62_encode(decimal):
```

$S \Rightarrow$ "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz"

hash_str = ""

while dec > 0:

hash_str = S[dec % 62] + hash_str

dec /= 62

return hash_str

print base62_encode(999)

Q - what database to use?

RDBMS

+ ACID

Scaling is problematic
we can use sharding
but there are lot of users
(30M) & more read/write
per sec

DP level hrs are present
(if object present or not present
in database then only insert)

NO SQL

- Eventual consistency
we write something but
it takes sometime to
reflect to the other
node.

+ Highly available

+ Easy scaling

Q - How to insert mapping of long URL to short
URL into db?

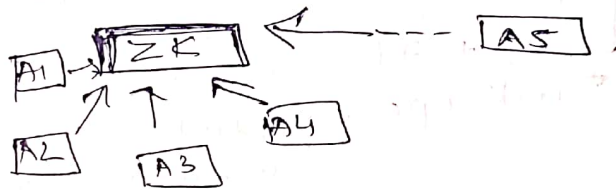
www.something \Leftarrow long URL

\rightarrow random \rightarrow 1234137 \rightarrow BLZ
AaBbCd1

www.us.com / AaBbCd1 \Leftarrow short URL

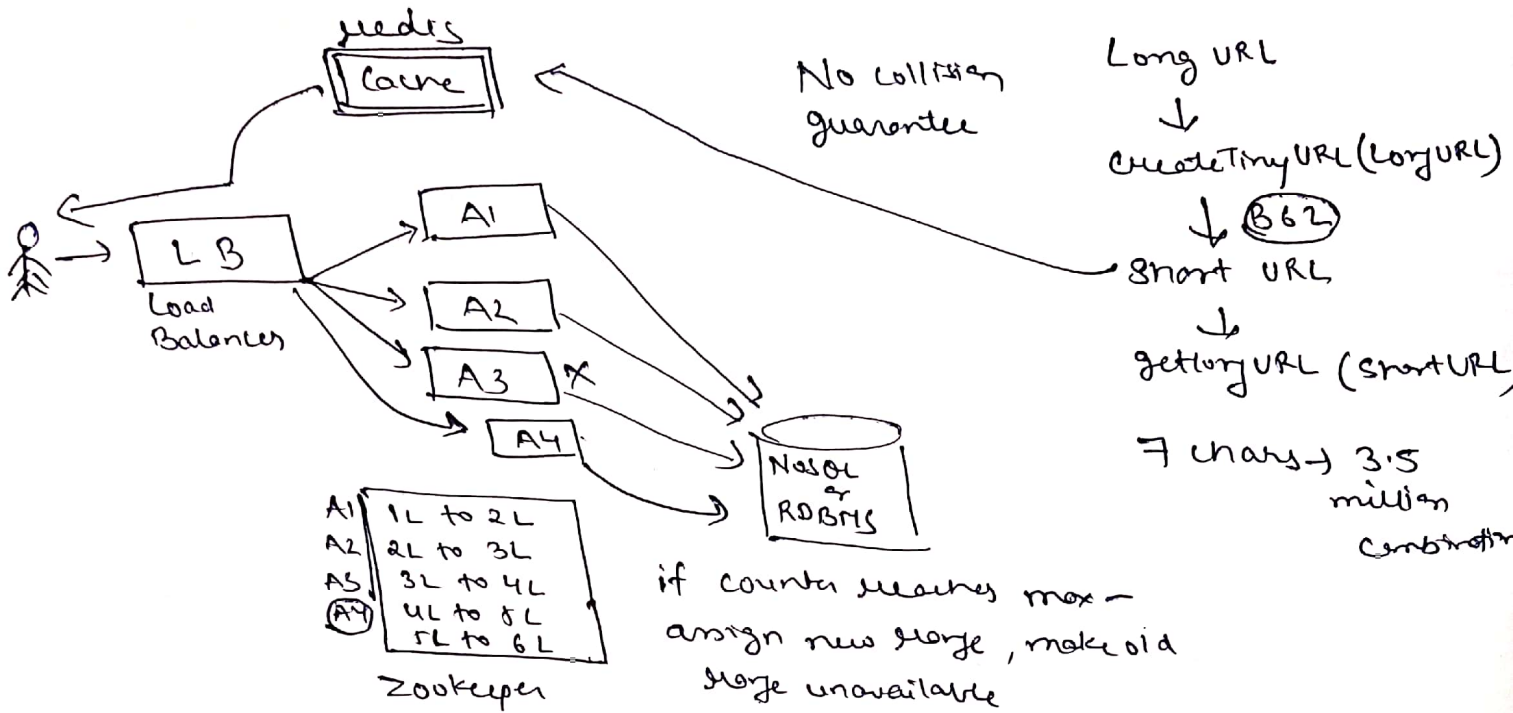
Zookeeper

- ① Distributed coordination service to manage multiple host or distributed host machine.
- ② It solves the coordination services between multiple apps or hosts or servers by its simple architecture
- ③ It lets developer concentrate on the core of business logic and not to worry about the coordination service.



- ⇒ Zookeeper adds the server in a distributed system
- ⇒ It names them,
- ⇒ Coordinates between them
- ⇒ and elects master in them
- ⇒ Checks which server is down
- ⇒ Also keeps some configuration information for all these hosts.
- ⇒ Also provides a coordinated file system where these hosts individual hosts can write the data

URL Shortener | Tiny URL | Bitly



APIs

createTinyURL(LongURL)

```
def B62(integer)
    return string
```

```
while n > 0:
    shorturl.append(map(n % 62))
    n = n / 62
```

long int ShortURLtoID(string ShortURL)

```
long int id = 0
```

```
for(int i=0; i < ShortURL.length(); i++)
```

```
if (ShortURL(i) <= 'z')
    id = id * 62 + ShortURL(i) - 'a'
```

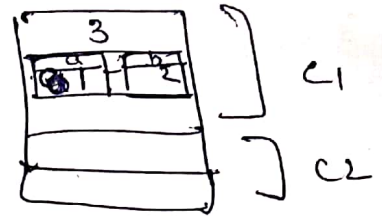
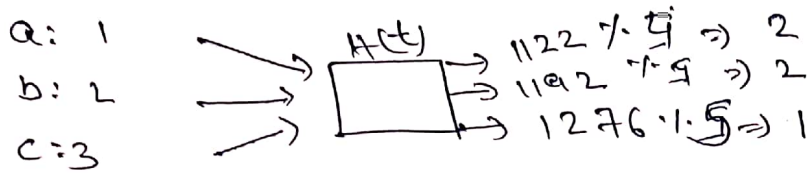
```
if ('A' <= ShortURL(i) && ShortURL(i) <= 'Z')
    id = id * 62 + ShortURL(i) - 'A' + 26
```

```
if ('0' <= ShortURL(i) && ShortURL(i) <= '9')
    id = id * 62 + ShortURL(i) - '0' + 52
```

```
}
```

Hashing & consistent Hashing

Key: value, Hash fn, Hash table.

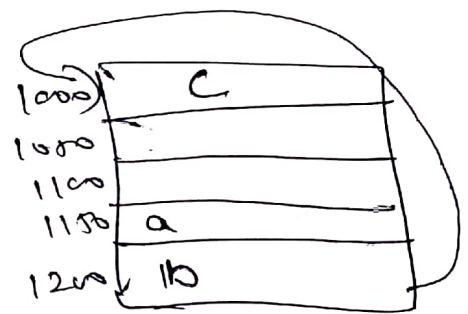
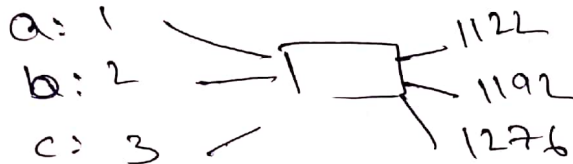


then error while retrieving

What if C2 goes down

Use consistent Hashing instead

generate random no., save to next big index



no position for
 C then go
 to index 1 again