Tiny Url

Statement:-

A tiny Url is a small Url (handy Url) for a large Url. With this tiny Url, share with other team members, typo error can be avoided.

Functional Requirement.

1) create a short Url for a given long Url.

2) Re-direct/Return long Url for the short Url input.

3) If multiple User trying to get the same long Url, short Url, for the given long Url, system should give the same short Url.

4) Once short Url is created for a long Url, it is used for that same long Url (forever) which means, that short Url cannot be used again for different long Url.

Non-Functional Requirement:-

1) System should be highly available.

2) Capture metrics & logs for analytics.

3) Auto scale.

Traffic & Capacity:-

Let's assume 10M short Url requests comes per month. Also 1:200 is the write-read ratio, which means for every 1 write there will be 200 reads. so it is a read heavy application.

Let's also assume size of the Url will be 500 bytes.

Write - requests per second = $\frac{10M}{30 \times 24 \times 60 \times 60} \simeq 4$ URLs / sec.

Read - request per second = $4 \times 200 = 800$ URLs/sec

Size of the URL = 500 bytes.

Storage:-

For one month, number of URLs = 10M,

So total storage = $10M \times 500$ bytes $\simeq 4.6$ GB / per month.

Let's Assume system should be available for 100 years, then total storage capacity will be.

$4.6 \times 100 \times 12 \simeq 5.4$ TB.

Cache:-

Number of reads per second = 800 URLs

So for month = $800 \times 30 \times 24 \times 60 \times 60 = 2$ billion requests

So cache limit will be to hold one month's URLs.

Cache capacity = $800 \times 30 \times 24 \times 60 \times 60 \times 500 \simeq 1$ GB.

Short-URL length:-

If it is decided to use Base-62 encoding,

Total number of URLs for 100 years = 120 Billion.

Total number of characters required in short URL = 7

$62^5 \simeq 1$ billion, $62^6 \simeq 56$ billion, but

$62^7 \simeq 3.5$ trillion.

How base-62 works?

Base-62 encoding, ~~performs hashing on~~ works on encoding the integers or alphanumerics.

~~eg: 1 2 3~~

So convert the ~~take~~ url' to decimal by hashing and then apply Base-62 encoding. ~~So, In~~

hash (long url) ⟶ Base 62 encoding (hash_value)

↓

short url.

Drawback of this approach, with hashing chances of getting duplicate is more, ~~and more~~ i.e. 2 or more ~~hash~~ url's can return the same value, and so hashing with encoding is not possible.

Let's try with another approach & which is MD-5 algorithm.

In this approach, MD5 algorithm is applied directly on long url and the encoded string is long and so need to trim the ~~encoded~~ encoded string ~~thereby~~ to get 7-character short url. Due to this trimming duplicates are possible. ~~So this~~

MD 5 (long url) ⟶ trim (encoded string) ⟶ Base 62 (trimmed encoded string)

↓

short url.

↑

if short url exists then take another set of trimmed encoded string.

The core problem of above approaches are getting the duplicate value before encoding for the same URL. If that duplicate value for same URL is resolved, then Base62 encoding works flawlessly. TO solve this duplicate issue can go with counter approach.

In this approach, get the minimum value that is required to generate 7-characters for short URL, and this minimum value be used as a starting range.

For example, Let's assume 100000000 will be the starting value for generating 7-char short URL.

For the first time value will be 100000000 and increment by a fixed number, in this case let's increment by 1 & apply Base-62 encoding for the hash value.

Base62 (100000000) ——→ P07AK(my tiny Url).

Second time

Base62 (100000001) ——→ AK7923t

So there won't be any duplicates i.e. no long short URL for 2 different long URL.

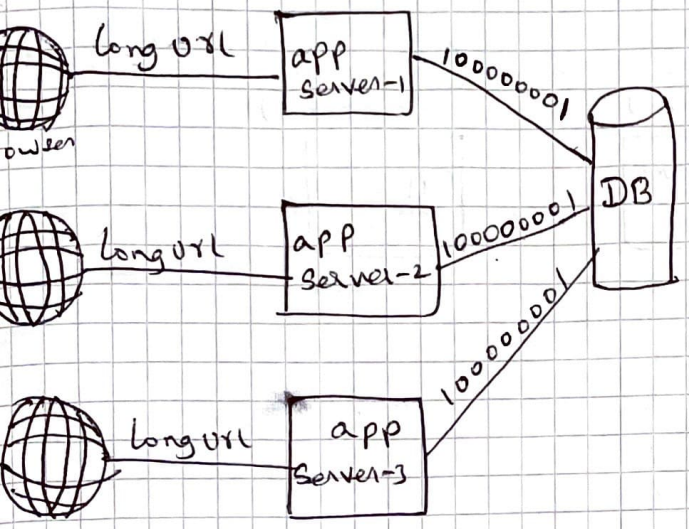
Problem with this approach will be,

If This Base62 encoding happens in the short URL application server, then there are few issues to be addressed.

1) What will happen, when the application server fails?

In this case, counter value will be lossed, and if there is only

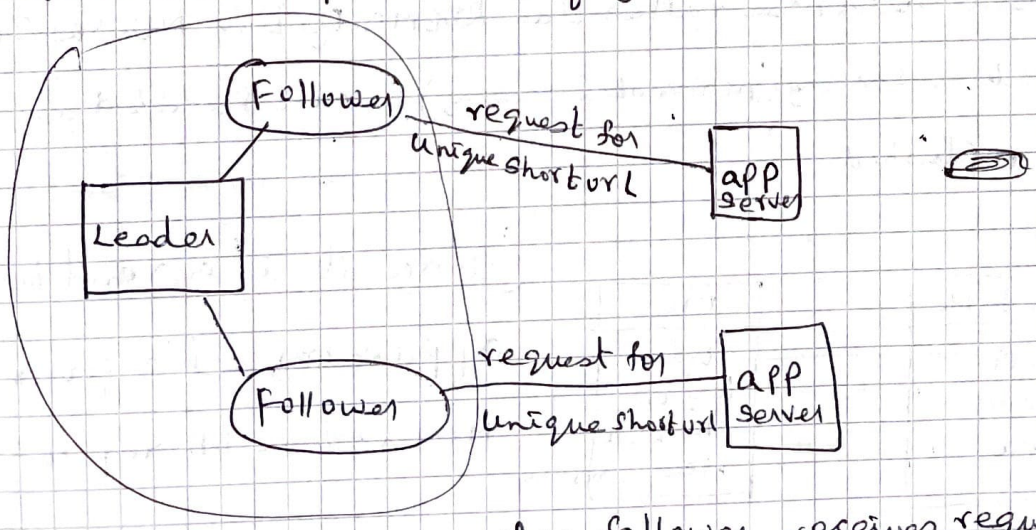
one application server, then it is a single point failure.
If we have multiple application server, and if it is maintain
the counter value is maintained in database, then parallel
processing of request will be a problem, ~~which~~ because of
updating the same counter value from the app server to the database
or different long URL, so duplicates can come up.



To avoid this issue, we can either go with

1) Server synchronisation (Apache zookeeper)
2) ~~K-G-S~~ Range counter approach.

With server synchronisation, we can go ~~ahead~~ use zookeeper, which will take the responsibility of generating the keys using counters

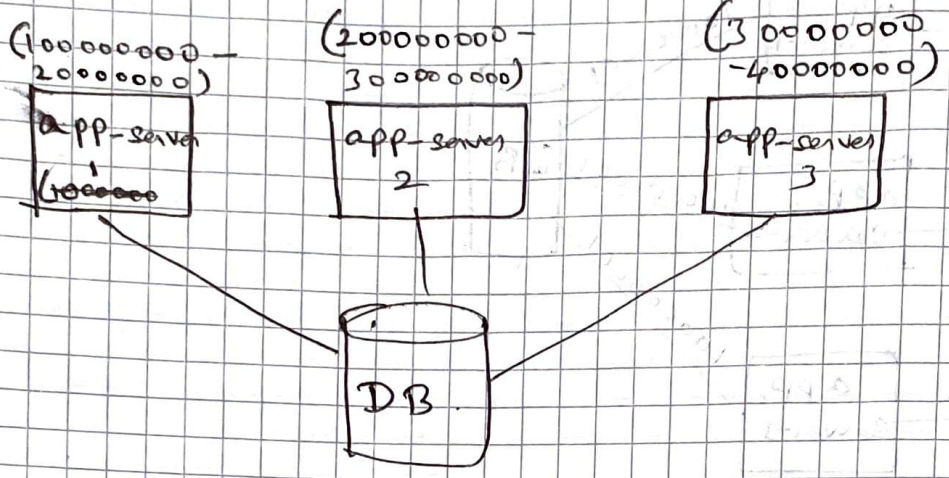


Co-ordination service,

When follower receives request, ~~leader~~ it will be forwarded to leader & leader will generate ~~for~~ counter increment tiny URL.

A Second approach is range based, where a ~~set of~~ range of counter value is ~~assigned~~ allocated to each application server, so that uniqueness will be achieved.

problem with this approach is, ~~it is~~ if one application server dies, then range allocated to that server will be wasted, in order to avoid this, use a db to keep track of used range values & un-used ones.

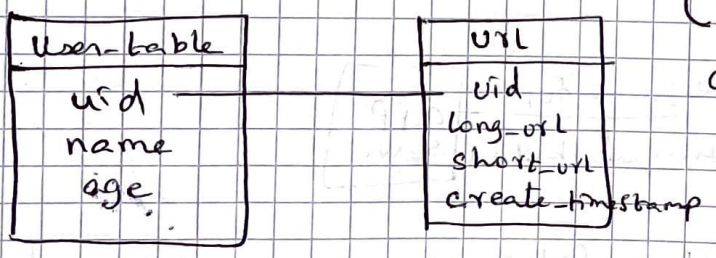


With these information, in mind let's discuss the design,

Let's start with DB, ~~first~~

Since it is write-read heavy application, and ~~data~~ record entities have no complex relationships, so it ~~ok~~ is better to use no-sql database rather than RDBMS. By this NO-SQL DB, even ~~RDB~~ auto-scaling (horizontal scaling) is more feasible & is easily accomplished.

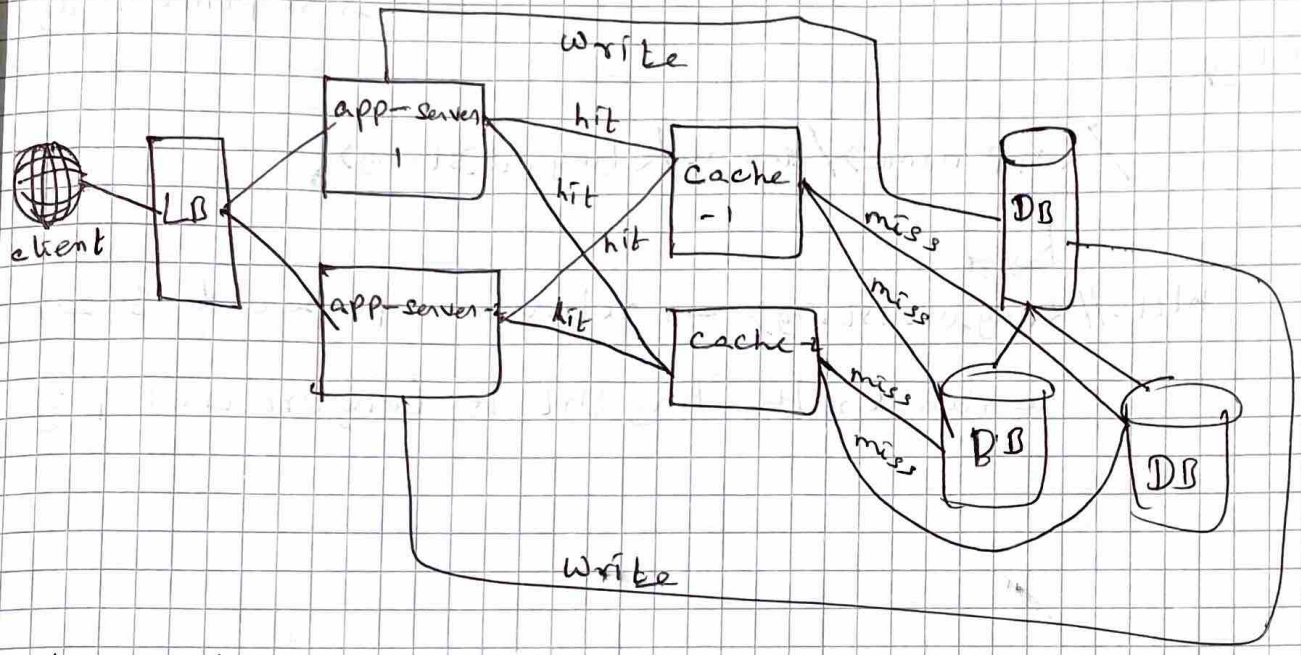
DB-Design :-



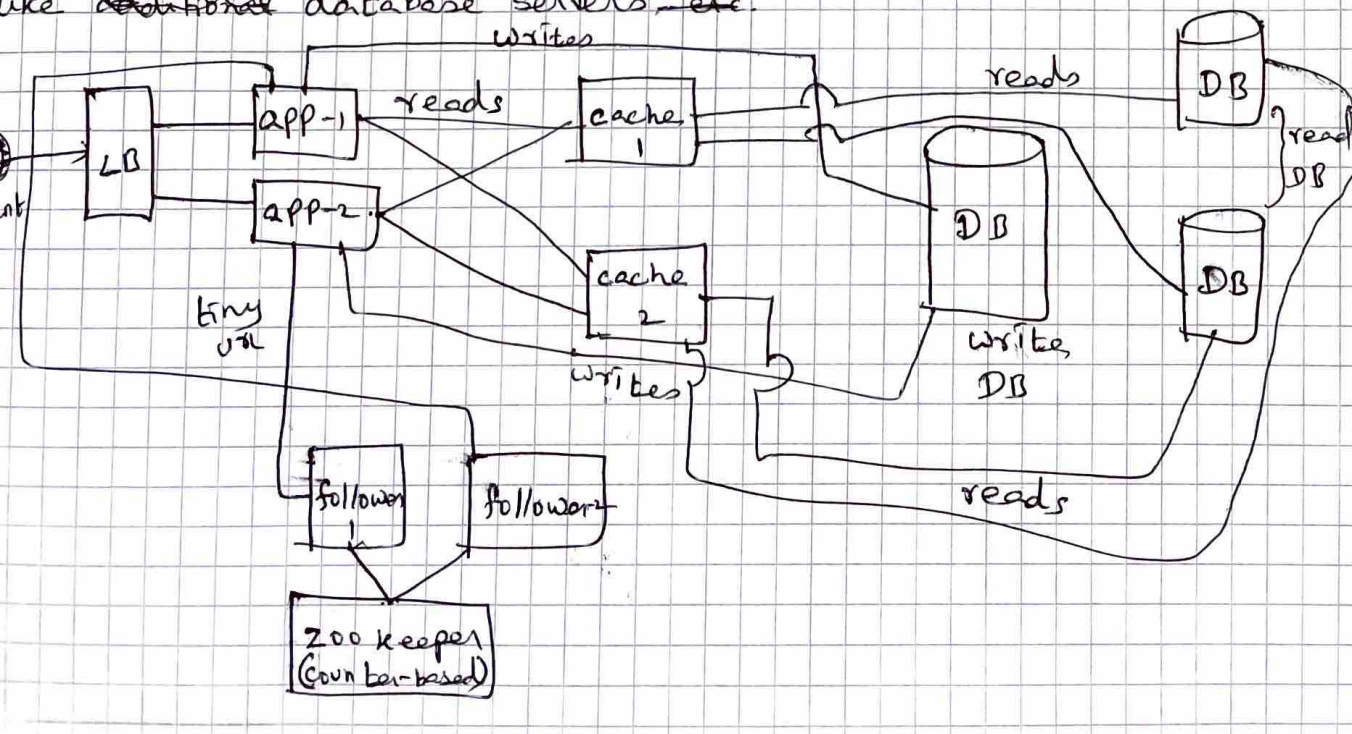
(Since it is a read-heavy application, let's go for Master-slave DB replica).

Now let's discuss about cache,

. As discussed earlier, per month there will be 2 billion requests
→ cache storage is decided to have 1 GB. It is required to store the short url for a long URL first in database, than in cache, because, in distributed system, especially for this concurrent request of tiny URL, duplicate short-url may be possible, if eventual consistency is followed over consistency. so consistency is on priority
of so let's go for write-around cache.



et's consider the tiny URL service, can go with zookeeper service for
iny URL generation, as range based approach needs more resource
like ~~additional~~ database servers: ~~etc~~.



Rest-API calls:- (HTTP request).

Post Request:-

/<app-name>/~~url~~tinyUrl → URL

body → {

user_id : <user_name">

Long_URL : <URL>

}

If successful returns response code 200 with tiny URL in response body

Get Request:-

/<app_name>/tinyurl/<tiny_url_string>

http://<tiny_url_string> → returns response code 302 & re-directs the tiny URL to Long URL web page