**Problem Statement**

Your manager has asked you to create a post for the company social media handles and add a link to the team recruitment form. The link to the form is very long and ends up consuming more characters. Your colleague advises you to shorten the link before posting to promote good digital hygiene, and the shortened link would also increase the Click-Through rate. You now want to design a URL Shortener to solve this issue.

What would be the critical points to be focused on for the designing of the URL Shortener system?

1.Features (functional req)

i)what is the length of the short url

ii)what kind of domains that I get (blocking etc)

2.Estimates

i) how many queries per second load does your system support ?

ii)what are the memory req ? CPU req ?

3.Design Goals (Non-functional req)

i)what kind of availability ?

ii)how consistency is important ?

iii)what is the partition tolerance of your sys ? etc .

4)HLD (High level design of the entire sys)

i)what are various components ? where are caches ? where are load balancers ? where are database servers ? what kind of databases ? what is relationship ? how do they interact ?

5)Scaling

how I’ll be scaling the sys

# Features (F.R)

1)shorten the given URL and generate alias

2)redirect to the original URL

3)should have an expiry to shorten URL

# Design goals (N.F.R)

1)High avalability

2)Low latency : redirection should happen in real time

3)Random urls should be generated (not predictable)

# Estimates and Constraints

i)what should be the throughput (traffic estimates query/ sec)

ii)latency concerns ?

iii)read write ratio for databases ?

iv)storage req

v)memory req

Throughput

lets say R:W = 100 :1 people accessing the short link are more than creating the link (read-intensive app)

Assume 500 M url shortenings every month , redirections will be 500 M x 100 = 50 B redictions

Queries per sec = 50 B / (30 x 24 x 60 x 60 ) = 20k queries / sec (read through put )

Write though put => 20,000 / 100 = 200 writes/sec (based on the ratio) (write through put )

Storage req

Assume we need to store urls for 5 years(default expiry)

500 M per month

Let say each url is around 500 bytes

500 M x 5 x 12 x 15 = 15 TB (tera bytes)

Primary mem (RAM)

20% of req to be in RAM for a day

20k/sec - > 20k x 60 x 60 x 24 = 1.7 B

1.7 B x 500 bytes x 20 % = 170 GB per day

# High level design

1)System APIs and Database decisions

i)Create url API : take long url return the shorten url to user

Parameters :

Api-key : A secret key

User name

Long url

Expiry date (default is 5 years)

ii)Delete url API

Parameters :

Api-key

Long url

iii)login / logout API

iv) Create user API ( registering )

v)Redirection service

Database

Service is read heavy , no strong relation ship b/w the data , no need of joins or any complex queries

So we may not require a relational DB , we can go with the NoSQL key-valued like MongoDB

Tables

i)URL

orginal\_url

created\_date

expiry\_date

user\_id

short\_url

ii)User

name

email

created\_at

api-key

2)Basic algorithm to shorten the url

i)Long url to short url

ii)No of collisions less (2 randomly generated url strings should not collide with each other , should be unique )

Approach 1 (Encode)

i)Take hash of the long url using widely used hash function MD5 , given a string it produces the 128 bit

hash value

ii) Encode the hash value into the characters such as [a-z A-Z 0-9] small, capital characters and digits 26+26+10 = 62 so Base62 encoding, if we include ‘+’ , ‘-‘ , it will be Base64 encoding

iii)from this take the no of characters from the first as per given requirement, if it is 6 long we would have 6^62 => 6 Billion possible permutations which is good for collison free.

How to handle collisions?

If the generated shrotened url already exists, create new one.

Approch 2 (Offline key generation)

As approach 1 is in application main flow , we can separate this also as another service.

Using a microservice cron job (to run periodically at fixed times, dates, or intervals.)

3)Purging / Deleting the expiry URL

Default expiry of 5 years taken, use cron job may be on a monthly basis to check the expired url

and delete, deleting should happen in all places db , read-replicas and the cache server also

# Scalability

Scaling the databases

Just mention range based, hash based and the problem and go for the consistent hashing.

As our system is read-intensive, split the db servers such that certain servers are only dedicated for reading, these should be in sync through replication.

Maintian the eventual consistency

Caching

20% are high-frequency urls as discussed, it needs 170 GB which can fit in single cache server. We can choose the best eviction policy

Load balancers

3 scenarios

i)between clent and application servers

ii)between application and data base servers

iii) between application and one cache servers

# Design and Diagram

