Description

1. Scope / MVP –

functional requirements

a.) Given a url, shorten url as output

b.) Given a url with custom string, → return short url with custom string as suffix

c.) Given short ~~url~~ url, → return back original url.

Non-functional Requirements

a.) system should be highly available.

b.) minimum latency.

2. Scale and ~~storage~~ estimates

Let's say 1 Day = 1000,000 requests for url shortening

WRITE Requests / month.

30 days = 30 × 1000,000

= 3 × 1000 × 1000

= 30×10^6 = 30 Million.

Let's say on an average a shorten url is accessed around ≃ 1000 times.

Read Request / month

= $30 \times 10^6 \times 1000$

= 30×10^9

= 30 billion

So, read requests are more than write requests.

Storage requirements. let's say we keep the data in our database for an year.

30 million × 12 = 36×10^7

Let's on ~~avg~~ average each url is 200 chars long = 200 bytes.

= $36 \times 10^7 \times 200$ bytes

= 72×10^9 bytes = 72×10^6 KB = 72×10^3 MB

= 72 GB ≃ 100 GB

QPS - query per second (write) $= \frac{\cancel{30} \times 10^6}{\cancel{30} \times 24 \times 60 \times 60} = \frac{10^4}{24 \times 6 \times 6} \cong 15$ write/sec

QPS (Read) $= 15 \times 1000$
$= 15K/sec$ (Read requests)

(3.) Design Rules

CAP theorem → . Two important points to meet here is.

available → service should be up all the time so that url redirection is working when requested.

system should be partition tolerant, so that it continues to function even in case of break in links.

Consistent → cache would be eventually consistent for the most frequently used url's. In case something is not available in cache, same will be fetched / looked up from DB.

Also, both DB's will sync with each other periodically.

(4.) Design API's.

(a) generateShortUrl (Request r) {
→ check option suffix string
→ call service layer to generate short url for input url.
→ return a base 64 encoded representation.
}
→ append suffix string if present.

write original and shorturl in DB.

```
(b.)  fetchLongUrl ( Request r ) {
          → search the short url in cache map, see if
            it is present and return back corresponding
            large url.
          → If not present in cache, get the long url from
            DB and persist in cache also.
      }   → return the results to the user browser.
```

— Questions

1.) First of all, we can have some hash which generates a unique hash of our input string and then we can further call base64 or. base128 encoding to get a unique short url string as output.

~~base64~~ ~~#11164~~

2.) for low latency, we will have our application instances deployed on all geographical locations. and then the load balancer (BIG IP) F5 technique and distribute the requests to the instance which is closest to the user location. Also, we will cache most frequently used URL's to return the results quickly.

3.) Even if the machine dies / or cache is invalidated, still there are multiple DB instances to get the results. Data is replicated to more than one instance to ensure high availability.

4.) System is consistent → This service will maintain the URL's in DB and cache (for frequently used). and cache and other DB's will sync periodically. The system will be always available to ensure the user is returned back original url for given short url.

5.) We have multiple instances of application, databases and load balancer. This is done to ensure high availability and no single point of failure in the system. All components have maintained clusters.