1) SYSTEM DESIGN OF URL SHORTENING SERVICE E.g TinyURL

1) Requirements & Clarifications :
(i)Design URL shortening service that converts long URL into unique short (6 characters length) URL
(ii) It can optionally have custom alias
(iii) Read : Write ratio is 100 : 1
(iv) Monthly 500M new urls are created.
(v) it can optionally have expiration date time

2) Capacity Estimation :

(i) Traffic / Query Per Second (QPS) ??
Write(QPS) :
500 M / 30(days) * 24(hours) * 3600(seconds) = 200 / seconds

Read(QPS):
100 * 500M / 30 * 24 * 3600
= 20,000 / seconds

(ii) Storage Required ??
Assuming 5 years and each object is of 500 bytes
5 * 12 (months) * 500 (bytes) = 15 TB

(iii) Network Bandwidth ??
Write (incoming request to server) :
200 (URL per second) * 500 (bytes)
= approx 10 KB / sec

Read (outgoing requeet from server) :
20,000 (URL per second) * 500 (bytes)
= approx 10 MB / sec

(iv) Memory Estimation (caching) ??
Considering 80:20 rule , that is 20% Redirecting generating 80% traffic

0.2 * 20,000(URL per second) * 24(hours) * 3600 (seconds) * 500 (bytes)
= approx 176 GB

note: less than 176GB will be needed, as majorly requests wiill be duplicates

3) System Interface Definition
createURL(api_dev_key, original_url, custom_alias=None,expiry_Date=None)

deleteURL(api_dev_key, short_url)

4) Define Data Modeling
observations are as below
(i) Read heavy
(ii) billions of records to be stored
(iii) no major relationship other than which user has created the url
(iv) object size 500 kb

we can have NO-SQL database considering above reasons.

USER
userid int 10 (pk)
username varchar 50
creationDate dateTime
lastLogin dateTime

URL
hashCode varchar 6 (pk)
originalUrl varchar 500
expiryDate dateTime
userid int 10 (fk)

5) High Level Design
Creating URl Logic:
We can use Key Generation Service (KGS) that will generate unique keys before hand and store in key DB.

key DB will have NotUsedUrl and UsedUrl columns.
We can periodically load some keys in memory of KGS and application servers.

Redirecting Url logic:
HTTP 302 Redirect status if key found and adding original url in location.
HTTP 404 status if key not found service

6) Detailed Design
We can add load balancer before application server.
We can add Cache server after application server.
Load Balancer can be placed in between cache servers and Database.
Cleanup service can be added to delete the deleted URL's from both Database and Key Db.

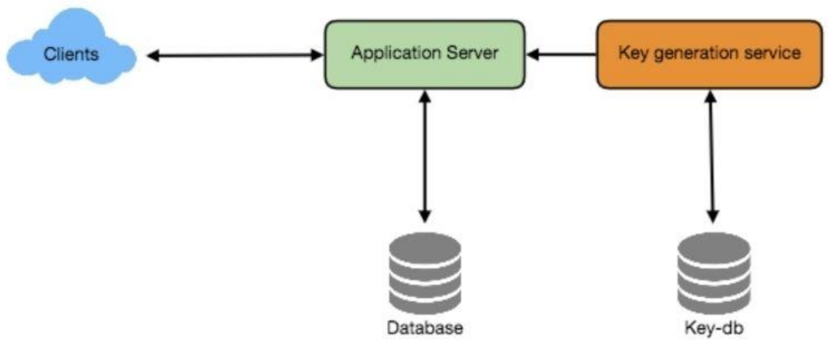
7) Identification and resolving bottlenecks
Single point of failure?
We can add standby load balancer and standby key Db, standby KGS service to avoid single point of failure

Data partitioning?

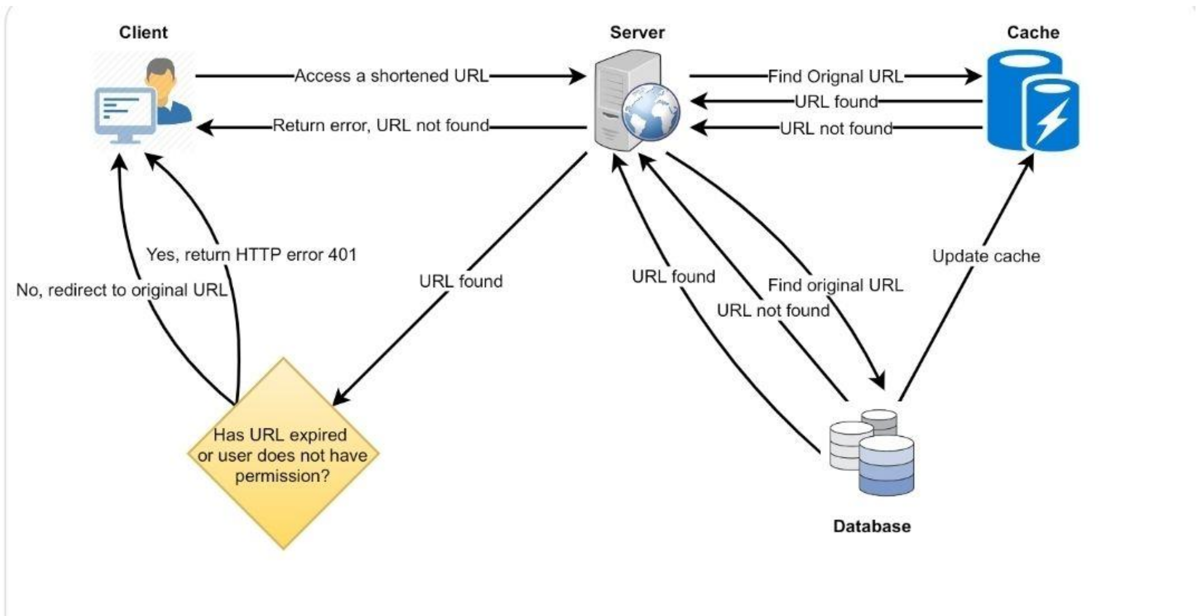
We can use consistent hashing for data partitioning

Data Replication Services?

As our system is READ heavy and AVAILABILITY is important, so we can add Replication services to avoid shutdown.



High level system design for URL shortening



Request flow for accessing a shortened URL