Designing Tiny URL / URL shortener

1. **Requirements clarifications – functional and non-functional requirements**

   Ask clarifying questions to the interviewer to learn the exact scope of the system.

   Functional requirements:

   a. The system must be capable of converting the long URLs to a short/ tiny readable URL
   b. When clicking the short URL, the page should automatically redirect to the longer URL page with minimum acceptable latency

   Non-functional requirements:

   a. Scalability - The system must be capable of growing and managing high volumes of data or high volumes of reads and writes as the number of users increase over the time without affecting the actual performance.
   b. Availability - In simple terms availability of a system is the time the system remains operational. In order for the system to remain highly available, it must be in operational state for a greater amount of time. This system in particular must be highly available because the users will be trying to access the webpage/link using their tiny URL but if the system is not available enough, then all the redirections would fail. So we have to design the system in a way that this condition is taken care of.
   c. Consistency - This is the property of maintaining coherence between the various database servers and between database and cache. If the URL is changed, then this change must be updated on all the database servers and also on the cache i.e., cache invalidation must be made. This system does not need to be highly consistent, as once we write the mapping between long URL and short URL in the database, we would just be making many reads on the database and not make constant changes that require updates on all servers and caches and maintaining coherence between them.

2. **System interface Definition – APIs**

   create_url(user_name, original_url, unique_id)

   delete_url(user_name, unique_id, short_url)

change_url(user_name, unique_id, original_url, short_url)

Parameters:

a. user_name : This is the username of the account that is requesting tiny url
b. Unique_id : This is some id generated while authentication and this id is unique for each user (just to maintain correctness in recording the data in case there are duplicate usernames)
c. Original_url: This is the url that needs to be converted or shortened
d. Short_url: It is the shortened url already created, which the user is trying to delete or change to

3. Capacity estimation

Here we estimate the capacity of our system based on the following:

a. Load of the Data we are dealing with, here, the approximate length of long URL and the length of the short URLs we are going to store in our database.

   Example:

   If 1 short URL takes 50 bytes and long URL ranges between 150-200 bytes, then for one URL mapping, we would store 250 bytes of data max

b. Number of requests - It is the number of URL conversion request which we get in a day

   Example:

   400 requests per day

c. Number of reads and writes - The number of writes indicates the number of requests for conversion per day. The number of reads indicates the number of redirections per day.

   We have already estimated the number of writes per day. We can assume that the number of reads could be at least 100 times more than writes. This is because once a shortened URL is given, all the users who have it so far irrespective of newly created requests, will try to access short URLs, so database reads for the corresponding long URLs.

   Example:

Say we get 400 * 100 = 40k per day

d. Network bandwidth

(Not sure how to estimate this as of now)

Therefore, the total capacity estimation in terms of Queries Per Second (QPS) would be:

400 * 50/ (24 hrs * 60 min * 60 seconds) requests per second (approx)

And

400 * 250 * 100 / (24 hrs * 60 min * 60 seconds) redirections per second (approx)

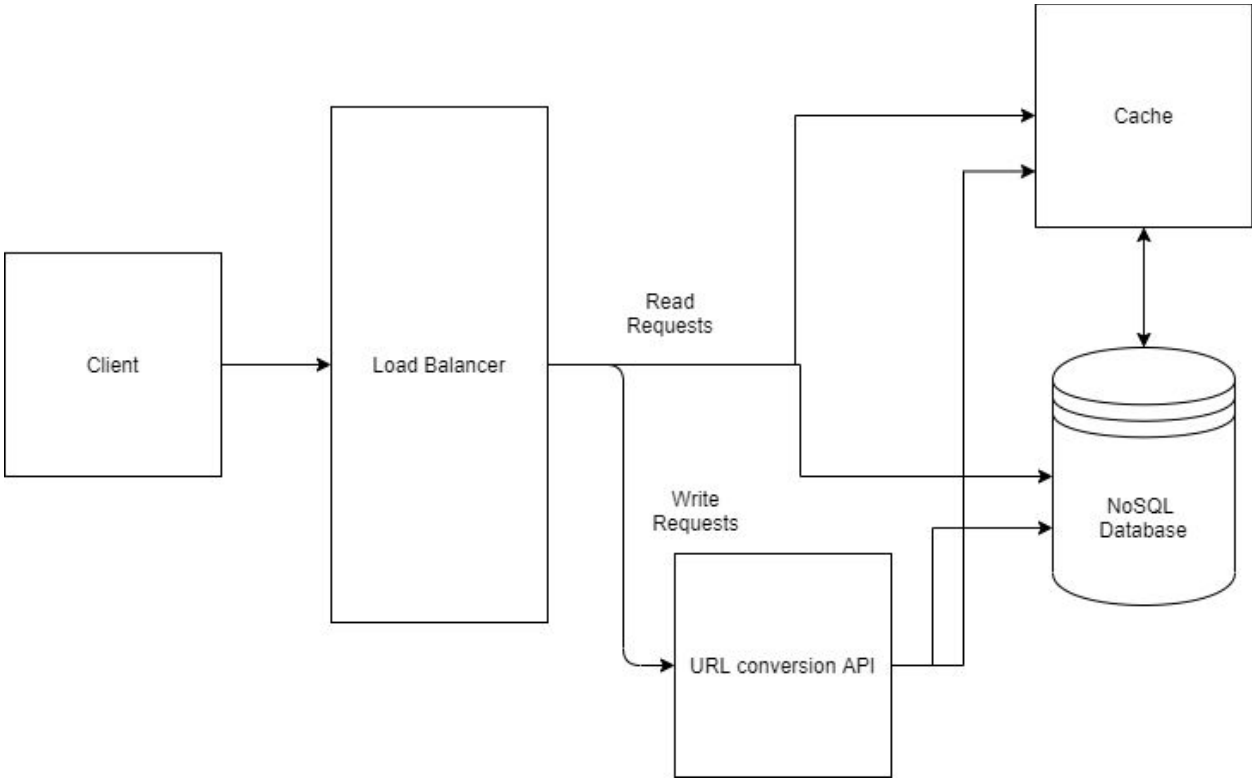
Storage estimation:

If we plan to store all these URLs for period of 5 years, then the amount of data that will be stored at a point would be:

4. Data Model

Since we are storing billions of data and for storing we just need the format of mapping the original_url to a short_url. A No-SQL key-value pair kind of a storage will be a good choice.

5. High-level design



6. Detailed design and discussion

7. Identifying the bottlenecks

If the database server is down, then it could cause a single-point of failure as none of the redirects would work. This can be overcome by having a cluster type architecture for the database, where we will have master and slaves. If the master goes down, then the requests must start forwarding to the slave without exposing it to the user. For this consistency between master and slaves must be maintained.