1. How to design the API?

Iniitial design (wrong)

A diagram of a webserver

Description automatically generated

What are some problems and solutions here?

Problems with above design :

1. There is only one WebServer which is single point of failure (SPOF)
2. System is not scalable
3. There is only single database which might not be sufficient for 60 TB of storage and high load of 8000/s read requests

To cater above limitations we :

1. Added a load balancer in front of WebServers
2. Sharded the database to handle huge object data
3. Added cache system to reduce load on the database.

**Actual good design here**

A diagram of a computer process

Description automatically generated

**Database Design**

API is very easy

Database schema

**Data Related to user**

1. **User ID:** A unique user id or API key to make user globally distinguishable
2. **Name:** The name of the user
3. **Email:** The email id of the user
4. **Creation Date:** The date on which the user was registered

**Data Related to ShortLink**

1. **Short Url:** 6/7 character long unique short URL
2. **Original Url:** The original long URL
3. **UserId:** The unique user id or API key of the user who created the short URL

**2 ways to shorten the url here**

a.)***URL Encoding randomly***

b) URL encoding through ***base62*** or MD5 coding

c) Using a global counter

Technqiue 1: Use url encoding random choice for each character here

We could just make a**random choice for each character** and check if this tiny url **exists** in DB or not. If it doesn’t exist return the tiny url else continue rolling/retrying.As more and more 7 characters short links are generated in Database, we would require 4 rolls before finding non-existing one short link which will slow down tiny url generation process.

* Con: probability of collision is higher due to the overlapping bits

**How to scale technique 1 (Short url from random numbers)**

[***MongoDB***](https://docs.mongodb.com/manual/sharding/#hashed-shard-keys) supports distributing data across multiple machines using shards. Since we have to support large data sets and high throughput, we can leverage sharding feature of MongoDB.

Once we generate 7 characters long tiny url, we can use this tinyURL as shard key and employ [sharding strategy](https://docs.mongodb.com/manual/sharding/#sharding-strategy) as hashed sharding. MongoDB automatically computes the hashes (Using tiny url ) when resolving queries. Applications do **not** need to compute hashes. Data distribution based on hashed values facilitates more even data distribution, especially in data sets where the shard key changes monotonically.

To speed up reads can also

**How do we scale up Mongodb?**

[***MongoDB***](https://docs.mongodb.com/manual/sharding/#hashed-shard-keys) use shards to spread data here

1. Use hash-based sharding

- Use 7 char tiny url as shard key, automatically computes the hashes (Using tiny url ) when resolving queries.

- use hash-based sharding here

2. Use range-based sharding

Scaling technique 2: by using auto increment counter here

1. We used a counter (A large number) and then converted it into a base62 ***7 character tinyURL***.As counters always get incremented so we can get a new value for every new request (Thus we don’t need to worry about getting same tinyURL for different long/ori

**Technique 2: Base 62 vs MD5**

Generate

What’s base62 conversion

1. **base 62** are [0–9][a-z][A-Z] so 62 bases here we have so this is quite important here

2. 62^7 ~= 3500 billion URLs which is quite enough in

**MD5 here**

The length is 128 bits here, the

**Gives morethan** 7 characters

* MD5 can create a lot of collisions. For two or many different long URL inputs we may get the same unique id for a short URL and that could cause data corruption.
* So we need to perform some checks to ensure that this unique id doesn’t exist in the database already.

**How to deal with collision?**

When a URL is shortened, we will use a hash function like MD5 or SHA256 to generate a hash and store the first 6 characters. If there’s a collision, we can choose the next 6 characters or add a random character at the end.

2 appraoches to shortening here

1. **Short url from random numbers:**

As more and more 7 characters short links are generated in Database,

2. Short url from base conversino

Converr a base 10 number 125 into the 62 you were referring to here

**How to scale using technique 2 hash base64 here with sql**

**Database instances here**

**How to scale with SQL?**

1. Each partition on separate sql server here

2. Must choose the right sharding key,

3. We can use sharding key as auto-incrementing counter and divide them into ranges for example from 1 to 10M, server 2 ranges from 10M to 20M, and so on.

Process

1. Start with 100 db instances, when any instance reaches maximum limit (10M), we can stop saving objects there and spin up a new server instance.

**Database structure**

CREATE TABLE **tinyUrl** (  
id [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) NOT NULL, AUTO\_INCREMENT  
shortUrl VARCHAR(7) NOT NULL,  
originalUrl VARCHAR(400) NOT NULL,  
userId VARCHAR(50) NOT NULL,  
automatically on primary-key column  
-- INDEX (shortUrl)  
-- INDEX (originalUrl)  
);

**Technique 3: Using a global counter**

**On-the-Fly URL Shortening with Using a Global Counter:**

* **Strategy:** You maintain a global counter and each time a new URL needs to be shortened, you increment the counter and use the value to generate a unique short URL.
* **Trade-offs:** The advantage is that this method is simple and guarantees uniqueness without needing to check for collisions. The disadvantage is that maintaining a global counter in a distributed system can be challenging due to synchronization issues.

***How will you maintain a global counter in distributed system?***

**Centralized Counter:** This is the simplest approach where you maintain the counter on a single machine (or a small number of machines for redundancy). Every time a new URL needs to be shortened, the service contacts this machine to get the next counter value. The disadvantage is that this machine can become a bottleneck if the request rate is high, and it also represents a single point of failure.

**Distributed Counter with Locks:** In this approach, the counter is distributed across multiple machines and a lock is used to ensure that only one machine can increment the counter at a time. This can help increase availability and reduce the risk of a single point of failure, but it can also introduce latency due to the need for acquiring and releasing locks.

**Sharded Counters:** You could shard the counter across multiple machines. Each shard manages a portion of the counter. For example, if you have two shards, the first shard could manage odd numbers and the second shard could manage even numbers. This can help increase throughput, but it also increases complexity.

**Using Distributed Data Stores:** You can use distributed data stores that provide atomic counters like ZooKeeper, etcd or a distributed database like Google’s Spanner. These systems provide primitives to handle distributed counters with consistency.

**Why use zooKeeper**

Wecan use a distributed service [Zookeeper](https://zookeeper.apache.org/) to solve the various challenges of a distributed system like a race condition, deadlock, or particle failure of data. Zookeeper is basically a distributed coordination service that manages a large set of hosts. It keeps track of all the things such as the naming of the servers, active database servers, dead servers, configuration information (*Which server is managing which range of counters*)of all the hosts.

Technique 3: Using MD5 conversion text

**MD5 Conversion next**

The **MD5 message-digest algorithm** is a widely used [hash function](https://en.wikipedia.org/wiki/Hash_function) producing a 128-[bit](https://en.wikipedia.org/wiki/Bit) hash value(or 32 [hexadecimal](https://en.wikipedia.org/wiki/Hexadecimal) digits). We can use these 32 hexadecimal digit for generating 7 characters long tiny url.

How do we scale this?

**How to check whether short URL is present in database or not ?**

**Solution** : When we get tiny url (For example **1L9zO9O**) we can use ***base62ToBase10***function to get the counter value (**100000000000**). Once we have this values we can get which database this counter ranges belongs to from zookeeper(Let’s say it database instance 1 ) . Then we can send SQL query to this server (Select \* from tinyUrl where id=10000000000111).This will provide us sql row data (\*if present)

2. How to resolve the hash collision situation

Using base 62 encoding would be really good for this as mentinoed here. And that’s what we want basically using this method

Basically using this

* From its name, base 62 is a way of using 62 characters for encoding. The mappings are: *0-0, ..., 9-9, 10-a, 11-b, ..., 35-z, 36-A, ..., 61-Z, where* ‘a’ stands for 10, ‘Z’ stands for 61, etc.
* 1115710 = 2 x 622 + 55 x 621 + 59 x 620 = [2, 55, 59] -> [2, T, X] in base 62 representation. Figure 6 shows the conversation process.

**3. Diff between mdn5 and base62 hashing?**

Basically the above would work given the solution below here

Base 62 8 characters long here,

MD5 used as an encrypter or decrypter here, very useful in both case scneraios.

**What’s the technique 4? (this is the last approach)**

We can have a standalone **Key Generation Service (KGS)** that generates random seven-letter strings beforehand and stores them in a database (let’s call it key-DB). Whenever we want to shorten a URL, we will take one of the already-generated keys and use it. This approach will make things quite simple and fast. Not only are we not encoding the URL, but we won’t have to worry about duplications or collisions. KGS will make sure all the keys inserted into key-DB are unique

We should have 2 tables

1. 1 for used key and 1 for unused keys here

**What problem are there?**

**Can concurrency cause problems?** As soon as a key is used, it should be marked in the database to ensure that it is not used again. If there are multiple servers reading keys concurrently,

How do we deal with a case where multiple requests come in at the same time

4. Would a failure be caused when multiple requests with same url come in at the same time?

<https://medium.com/@rishabh171192/system-design-url-shortener-22bd48bf7663>

Bad solution: In this we can maintain an Incremental counter for each request. Based on counter value we will create a short url and dump into db.

Also not good

Use zookeeper here

We need to set up some synchronisation between servers with some predefined configurations. ZooKeeper can solve this problem as its service provided by apache to maintain configuration, sync and coordination in distributed systems. ZooKeeper is widely used in large scale applications and some deployment configurations. Kubernetes also makes use of ZooKeeper for coordination between parent and child nodes.

Follow up questions

***How do you track collisions in your service?***

**Monitor Database Performance:** If your database queries are slowing down or if you’re seeing increased latency, it could be a sign that your hash space is crowded and it’s time to consider expanding it.