

What? Interview coaching from Googlers!

I want to know more (http://www.gainlo.co/?utm_source=blog&utm_medium=banner <http%3A//blog.gainlo.co/index.php/20>)



Create a TinyURL System

If you have started preparing for system design interviews, you must have heard of one of the most popular question – create a TinyURL (<http://tinyurl.com/>) system.

This week, Gainlo (http://www.gainlo.co/?utm_source=blog&utm_medium=create%20a%20tinyurl%20system&utm_campaign=blog) team has brainstormed tons of ideas about this interview question. I'd like to summarize how we'll analyze this problem with high-level thoughts here.

If you are new to system design interview, I'd highly recommend you take a look at 8 Things You Need to Know Before a System Design Interview (<http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/>) first, which provides general strategies that can help you get started.

Question

How to create a TinyURL (<http://tinyurl.com/>) system?

If you are not familiar with TinyURL, I'll briefly explain here. Basically, TinyURL is a URL shortening service, a web service that provides short aliases for redirection of long URLs. There are many other similar services like Google URL Shortener (<https://goo.gl/>), Bitly (<http://bit.ly>) etc..

For example, URL <http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/> (<http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/>) is long and hard to remember, TinyURL can create an alias for it – <http://tinyurl.com/j7ve58y> (<http://tinyurl.com/j7ve58y>). If you click the alias, it'll redirect you to the original URL.

So if you would design this system that allows people input URLs with short alias URLs generated, how would you do it?

High-level idea



Let's get started with basic and high-level solutions and we can keep optimizing it later on.

At first glance, each long URL and the corresponding alias form a key-value pair. I would expect you to think about something related to hash (https://en.wikipedia.org/wiki/Hash_function) immediately.

Therefore, the question can be simplified like this – given a URL, how can we find hash function F that maps URL to a short alias:

$$F(\text{URL}) = \text{alias}$$

and satisfies following condition:s

1. Each URL can only be mapped to a unique alias
2. Each alias can be mapped back to a unique URL **easily**

The second condition is the core as in the run time, the system should look up by alias and redirect to the corresponding URL quickly.

Basic solution

To make things easier, we can assume the alias is something like `http://tinyurl.com/<alias_hash>` and `alias_hash` is a fixed length string.

If the length is 7 containing [A-Z, a-z, 0-9], we can serve $62^7 \approx 3500$ billion URLs. It's said that there are ~644 million URLs (<https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=how%20many%20urls%20are%20there>) at the time of this writing.

To begin with, let's store all the mappings in a single database. A straightforward approach is using `alias_hash` as the ID of each mapping, which can be generated as a random string of length 7.

Therefore, we can first just store <ID, URL>. When a user inputs a long URL "http://www.gainlo.co", the system creates a random 7-character string like "abcd123" as ID and inserts entry <"abcd123", "http://www.gainlo.co"> into the database.

In the run time, when someone visits `http://tinyurl.com/abcd123`, we look up by ID "abcd123" and redirect to the corresponding URL "http://www.gainlo.co (http://www.gainlo.co)".

Performance VS flexibility



There are quite a few follow-up questions for this problem. One thing I'd like to further discuss here is that by using GUID (https://en.wikipedia.org/wiki/Globally_unique_identifier) (Globally Unique Identifier) as the entry ID, what would be pros/cons versus incremental ID in this problem?

If you dig into the insert/query process, you will notice that using random string as IDs may sacrifice performance a little bit. More specifically, when you already have millions of records, insertion can be costly. Since IDs are not sequential, so every time a new record is inserted, the database needs to go look at the correct page for this ID. However, when using incremental IDs, insertion can be much easier – just go to the last page.

So one way to optimize this is to use incremental IDs. Every time a new URL is inserted, we increment the ID by 1 for the new entry. We also need a hash function that maps each integer ID to a 7-character string. If we think each string as a 62-base numeric, the mapping should be easy (Of course, there are other ways).

On the flip side, using incremental IDs will make the mapping less flexible. For instance, if the system allows users to set custom short URL, apparently GUID solution is easier because for whatever custom short URL, we can just calculate the corresponding hash as the entry ID.

Note: In this case, we may not use random generated key but a better hash function that maps any short URL into an ID, e.g. some traditional hash functions like CRC32 (<http://www.accuhash.com/what-is-crc32.html>), SHA-1 (<https://en.wikipedia.org/wiki/SHA-1>) etc..

Cost

I can hardly not ask about how to evaluate the cost of the system. For insert/query, we've already discussed above. So I'll focus more on storage cost.

Each entry is stored as <ID, URL> where ID is a 7-character string. Assuming max URL length is 2083 characters (<https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=max%20url%20length>), then each entry takes $7 * 4$ bytes + 2083 * 4 bytes = 8.4 KB. If we store a million URL mappings, we need around 8.4G storage.

If we consider the size of database index (https://en.wikipedia.org/wiki/Database_index) and we may also store other information like user ID, date each entry is inserted etc., it definitely requires much more storage.

Multiple machines

Apparently, when the system has developed to certain scale, a single machine is not capable to store all the mappings. How do we scale with multiple instances?

The more general problem is how to store hash mapping across multiple machines. If you know distributed key-value store (https://en.wikipedia.org/wiki/Distributed_data_store), you should know that this can be a very complicated problem. I will discuss only high-level ideas here and if you are interested in all those details, I'd recommend you read papers like Dynamo: Amazon's Highly Available Key-value

Store (<http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>).

In a nutshell, if you want to store a huge amount of key-value pairs across multiple instances, you need to design a lookup algorithm that allows you to find the corresponding machine for a given lookup key.

For example, if the incoming short alias is <http://tinyurl.com/abcd123>, based on key "abcd123" the system should know which machine stores the database that contains entry for this key. This is exactly the same idea of database sharding ([https://en.wikipedia.org/wiki/Shard_\(database_architecture\)](https://en.wikipedia.org/wiki/Shard_(database_architecture))).

A common approach is to have machines that act as a proxy, which is responsible for dispatching requests to corresponding backend stores based on the lookup key. Backend stores are actually databases that store the mapping. They can be split by various ways like use $\text{hash}(\text{key}) \% 1024$ to divide mappings to 1024 stores.

There are tons of details that can make the system complicated, I'll just name a few here:

- Replication ([https://en.wikipedia.org/wiki/Replication_\(computing\)](https://en.wikipedia.org/wiki/Replication_(computing))). Data stores can crash for various random reasons, therefore a common solution is having multiple replicas for each database. There can be many problems here: how to replicate instances? How to recover fast? How to keep read/write consistent?
- Resharding. When the system scales to another level, the original sharding algorithm might not work well. We may need to use a new hash algorithm to reshard the system. How to reshard the database while keeping the system running can be an extremely difficult problem.
- Concurrency ([https://en.wikipedia.org/wiki/Concurrency_\(computer_science\)](https://en.wikipedia.org/wiki/Concurrency_(computer_science))). There can be multiple users inserting the same URL or editing the same alias at the same time. With a single machine, you can control this with a lock. However, things become much more complicated when you scale to multiple instances.

Summary

I think you've already realized that the problem is not complicated at first glance, but when you dig into more details especially consider scale issues, there can be a lot of follow-up problems.





As we said in earlier post (<http://blog.gainlo.co/index.php/2016/02/17/system-design-interview-question-how-to-design-twitter-part-1/>), specific techniques used is not that important. What really matters is the high-level idea of how to solve the problem. That's why I don't mention things like Redis (<http://redis.io/>), RebornDb (<http://reborndb>) etc..

Again, there are infinite ways to extend this problem further. I'd like to use this post as a starting point for you to get familiar with system design interview.

If you find this post helpful, I would really appreciate if you can share it with your friends. Also you can check more system design interview questions (<http://blog.gainlo.co/index.php/category/system-design-interview-questions/>) and analysis here.

The post is written by [Gainlo \(http://www.gainlo.co/?utm_source=blog&utm_medium=footer-link http%3A//blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/&utm_campaign=blog\)](http://www.gainlo.co/?utm_source=blog&utm_medium=footer-link) - a platform that allows you to have mock interviews with employees from Google, Amazon etc..

I'd like to learn more (http://www.gainlo.co/?utm_source=blog&utm_medium=footer http%3A//blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/)

 (<http://www.facebook.com/sharer.php?u=http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/>)  (<http://twitter.com/share?url=http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/&text=Create+a+TinyURL+System+>)  (<http://www.linkedin.com/shareArticle?mini=true&url=http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/>)  (<http://reddit.com/submit?url=http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/&title=Create+a+TinyURL+System>)