Random ID Generator

Let's continue our system design interview questions discussion. If you are new to this series, you can check our previous posts (http://blog.gainlo.co/index.php/category/system-design-interview-questions/). Basically, each week we are going to pick several interesting interview questions and provide in-depth analysis.

It's worth to note that the post is not about giving you something like a standard answer. Instead, we focus more on analyzing the problem and how to come up with reasonable approaches. This is even more true for system design interviews because the question can be extremely open-ended.

This week, we will talk about how to a random ID generator. We will cover a bunch of topics including scaling the ID generator and pros and cons of each approach.

Random ID Generator

In case you don't know what an ID generator is, let me briefly explain this here. Suppose you are building a social network product like Twitter, you need to store each user in your database with a user ID, which is unique and used to identify each user in the system.

In some systems, you can just keep incrementing the ID from 1, 2, 3 ... N. In other systems, we may need to generate the ID as a random string. Usually, there are few requirements for ID generators:

- **They cannot be arbitrarily long.** Let's say we keep it within 64 bits.
- **ID is incremented by date.** This gives the system a lot of flexibility, e.g. you can sort users by ID, which is same as ordering by register date.

There can be some other requirements, especially when you want to scale the system to support millions or even billions of users.

Single machine

As we suggested previously (http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/), it's good to start with something simple and keep optimizing it later, which is more true when the question is broad. If I got this question in a system design interview, most likely I'll start with a single machine design. Not only is this easy to design, but it's good enough for most of the cases.

In the simplest case, we can just keep incrementing ID from 1, 2, 3 ... N, which in fact is one of the most popular ways to generate ID in many real life projects. If user A's ID is larger than user B, then we know that A registered later.

However, this approach is hard to scale. Let's say after one year there are too many users everyday that we have to scale the database to multiple instances. You'll see that this approach won't work because it may generate duplicate IDs for different users.

3rd party service

To scale the ID generator to multiple machines, one natural solution is to keep a single separate server that is only responsible for ID generation. More specifically, when a user registers to the product, for whichever database that handles this request, it'll connect to the 3rd party server to ask for a random ID. Since all the ID generation is handled in a single server, there's no risk of generating duplicate IDs.

However, the downside of this solution is obvious. Suppose the product is so popular that there can be a huge number of people registering within a single second, the 3rd party server will soon become the bottleneck. The server may either block registration or just crash.

Multiple machine solution

Thus, we have to scale the ID generation to multiple servers. If we want to have no communication between ID generation servers, each server itself should be able to generate unique IDs that are incremented by time. It should be quite natural to think about using timestamp to generate IDs.

Since within a single timestamp there can also be multiple users, we could solve this with two approaches.

1. We assign a server ID to each ID generation server and the final ID is a combination of timestamp and the server ID.
2. We can also allow multiple requests within a single timestamp on a single server. We can keep a counter on each server, which indicates how many IDs have been generated in the current timestamp. So the final ID is a combination of timestamp, serverID and the counter.

As mentioned previously, the ID cannot be arbitrarily long so that the counter may end up with only 8 bits for instance. In this case, the server can handle 256 requests within a single timestamp at most. If it frequently exceeds this limit, we need to add more instances.

In fact, this solution is what Twitter does to solve the problem. They open sourced their ID generator called Snowflake (https://blog.twitter.com/2010/announcing-snowflake).

Clock synchronization

We ignored a crucial problem in the above analysis. In fact, there's a hidden assumption that all ID generation servers have the same clock to generate the timestamp, which might not be true in distributed systems.

In reality, system clocks can drastically skew in distributed systems, which can cause our ID generators provide duplicate IDs or IDs with incorrect order. Clock synchronization is out of the scope of this discussion, however, it's important for you to know such issue in this system. There are quite a few ways to solve this issue, check NTP (https://en.wikipedia.org/wiki/Network_Time_Protocol) if you want to know more about it.

Summary

Random ID generator is a very practical issue in many real life projects. Similar to many other systems, it seems to be easy at first glance, but there are quite a lot issues when scaling to multiple machines. If you want to know more about this topic, you can also check Flickr's ticket servers (http://code.flickr.net/2010/02/08/ticket-servers-distributed-unique-primary-keys-on-the-cheap/), which is another approach besides Snowflake.

Related Posts:

1. **Design Facebook Chat Function (http://blog.gainlo.co/index.php/2016/04/19/design-facebook-chat-function/)**
2. **Design a Recommendation System (http://blog.gainlo.co/index.php/2016/05/24/design-a-recommendation-system/)**
3. **Design eCommerce Website (Part I) (http://blog.gainlo.co/index.php/2016/08/22/design-ecommerce-website-part/)**
4. **Design eCommerce Website (Part II) (http://blog.gainlo.co/index.php/2016/08/28/design-ecommerce-website-part-ii/)**

5 thoughts on "Random ID Generator"

MATIAS SM
June 8, 2016 at 7:27 pm (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/#comment-1140)

I think you should remove the "random" part of the title, since you are discussing a plain id generator (not a random one).

"A random-number generator (RNG) is a computational or physical device designed to generate a sequence of numbers or symbols that can not be reasonably predicted better than by a random chance."

o.co/index.php/2016/06/07/random-id-generator/?replytocom=1140#respond)

RICK MAC GILLIS (HTTP://RICKMACGILLIS.COM)
June 8, 2016 at 11:54 pm (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/#comment-1142)

My solution to the problem was to use a Unique key on the database so that the ID is always unique. I'd also use a hardware RNG to generate a number in the 64 bit integer range. If the row won't insert because of it having a duplicate key, then the DB will throw an error. We could try to insert it again under another number, or we could just pass the very rare error message to the end user so they can resubmit.

The trade-off is that the DB will have to verify that that ID is unique, and thus it could cost an O(log n) overhead from a binary search with each insert. Another trade-off is that we wouldn't know when they registered by looking at the ID.

On the upside, the hardware module will give an excellent entropy source, thus reducing ID collisions, and it's far less involved to set up the system this way as you don't need to account for timestamps, server ID, etc. As a 64 bit integer is far larger than the total Earth population, then the risk of an ID collision is already immensely small.

My example favors time over space, as we could still use the traditional createdAt, updatedAt, (and possibly deletedAt for soft-deletes), to handle the timestamps for actions on the account.

o.co/index.php/2016/06/07/random-id-generator/?replytocom=1142#respond)

JAKE
June 9, 2016 at 10:10 pm (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/#comment-1148)

Hi Rick,

Your solution works if we don't care about whether IDs are sorted by registration date.

index.php/2016/06/07/random-id-generator/?replytocom=1148#respond)

LINUAS
September 17, 2017 at 9:35 am (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/#comment-11964)

Why is this a requirement of a Random ID Generator:
"They cannot be arbitrarily long"

.co/index.php/2016/06/07/random-id-generator/?replytocom=11964#respond)

LINUAS
September 17, 2017 at 10:01 am (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/#comment-11966)

The post mentions "In reality, system clocks can drastically skew in distributed systems, which can cause our ID generators provide duplicate IDs", how could this happen due to Clock Synchronization?
If each machine still uses "combination of timestamp, serverID and the counter." I can understand IDs with incorrect order exist but how do duplicates come in place?

.co/index.php/2016/06/07/random-id-generator/?replytocom=11966#respond)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

‹ Second Largest Element of a Binary Search Tree (http://blog.gainlo.co/index.php/2016/06/03/second-largest-element-of-a-binary-search-tree/)

Flatten a Linked List › (http://blog.gainlo.co/index.php/2016/06/12/flatten-a-linked-list/)

CATEGORIES

Coding Interview Questions (http://blog.gainlo.co/index.php/category/coding-interview-questions/)

Common Pitfalls (http://blog.gainlo.co/index.php/category/common-pitfalls/)

Common Questions (http://blog.gainlo.co/index.php/category/common-questions/)

Facebook Interview Questions (http://blog.gainlo.co/index.php/category/facebook-interview-questions/)

Mock Interview Thoughts (http://blog.gainlo.co/index.php/category/mock-interview-thoughts/)

Others (http://blog.gainlo.co/index.php/category/others/)

System Design Interview Questions (http://blog.gainlo.co/index.php/category/system-design-interview-questions/)

The Complete Guide to Google Interview Preparation (http://blog.gainlo.co/index.php/category/google-interview-preparation/)

Uber Interview Questions (http://blog.gainlo.co/index.php/category/uber-interview-questions/)