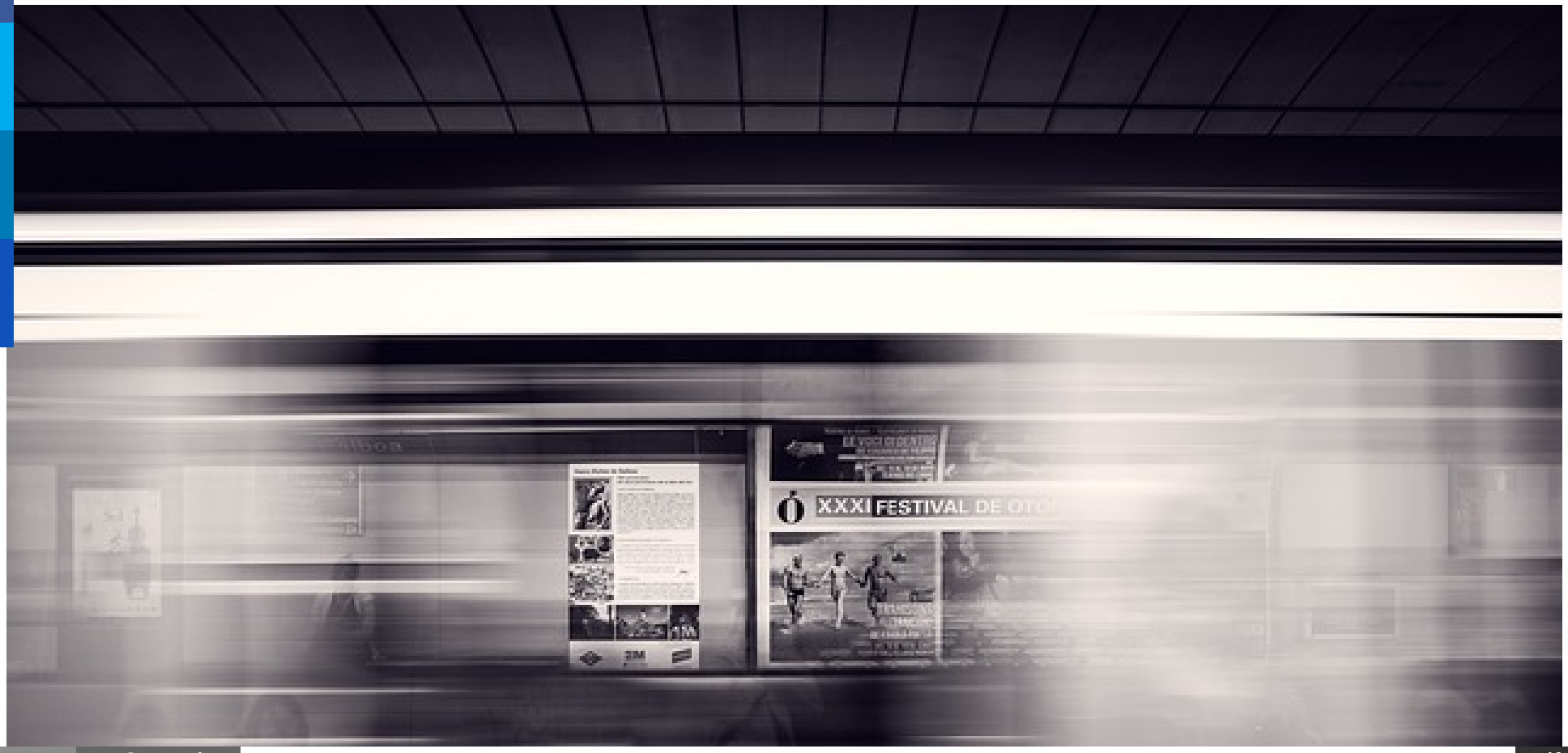
What? Interview coaching from Googlers!

I want to know more



Design a Cache System

Similar to our previous posts, we would like to select system design interview questions that are popular and practical so that not only can you get ideas about how to analyze problems in an interview, but learn something interesting at the same time.

If you have no idea about system design interviews, I'd recommend you read this tutorial first. In this post, we are addressing the problem – how to design a cache system. Topics covered by this post include:

- LRU cache
- Eviction policy,
- Cache concurrency
- Distributed cache system

Problem

How to design a cache system?

Cache system is a widely adopted technique in almost every applications today. In addition, it applies to every layer of the technology stack. For instance, at network area cache is used in DNS lookup and in web server cache is used for frequent requests.

In short, a cache system stores common used resources (maybe in memory) and when next time someone requests the same resource, the system can return immediately. It increases the system efficiency by consuming more storage space.

LRU

One of the most common cache systems is LRU (least recently used). In fact, another common interview question is to discuss data structures and design of an LRU cache. Let's start with this approach.

The way LRU cache works is quite simple. When the client requests resource A, it happens as follow:

- If A exists in the cache, we just return immediately.
- If not and the cache has extra storage slots, we fetch resource A and return to the client. In addition, insert A into the cache.
- If the cache is full, we kick out the resource that is least recently used and replace it with resource A.

The strategy here is to maximum the chance that the requesting resource exists in the cache. So how can we implement a simple LRU?

LRU design

An LRU cache should support the operations: lookup, insert and delete. Apparently, in order to achieve fast lookup, we need to use hash. By the same token, if we want to make insert/delete fast, something like linked list should come to your mind. Since we need to locate the least recently used item efficiently, we need something in order like queue, stack or sorted array.

To combine all these analyses, we can use queue implemented by a doubly linked list to store all the resources. Also, a hash table with resource identifier as key and address of the corresponding queue node as value is needed.

Here's how it works. when resource A is requested, we check the hash table to see if A

exists in the cache. If exists, we can immediately locate the corresponding queue node and return the resource. If not, we'll add A into the cache. If there are enough space, we just add a to the end of the queue and update the hash table. Otherwise, we need to delete the least recently used entry. To do that, we can easily remove the head of the queue and the corresponding entry from the hash table.

Eviction policy

When the cache is full, we need to remove existing items for new resources. In fact, deleting the least recently used item is just one of the most common approaches. So are there other ways to do that?

As mentioned above, The strategy is to maximum the chance that the requesting resource exists in the cache. I'll briefly mention several approaches here:

- Random Replacement (RR) – As the term suggests, we can just randomly delete an entry.
- Least frequently used (LFU) – We keep the count of how frequent each item is requested and delete the one least frequently used.
- W-TinyLFU – I'd also like to talk about this modern eviction policy. In a nutshell, the problem of LFU is that sometimes an item is only used frequently in the past, but LFU will still keep this item for a long while. W-TinyLFU solves this problem by

calculating frequency within a time window. It also has various optimizations of storage.

Concurrency

To discuss concurrency, I'd like to talk about why there is concurrency issue with cache and how can we address it.

It falls into the classic reader-writer problem. When multiple clients are trying to update the cache at the same time, there can be conflicts. For instance, two clients may compete for the same cache slot and the one who updates the cache last wins.

The common solution of course is using a lock. The downside is obvious – it affects the performance a lot. How can we optimize this?

One approach is to split the cache into multiple shards and have a lock for each of them so that clients won't wait for each other if they are updating cache from different shards. However, given that hot entries are more likely to be visited, certain shards will be more often locked than others.

An alternative is to use commit logs. To update the cache, we can store all the mutations into logs rather than update immediately. And then some background processes will

execute all the logs asynchronously. This strategy is commonly adopted in database design.

Distributed cache

When the system gets to certain scale, we need to distribute the cache to multiple machines.

The general strategy is to keep a hash table that maps each resource to the corresponding machine. Therefore, when requesting resource A, from this hash table we know that machine M is responsible for cache A and direct the request to M. At machine M, it works similar to local cache discussed above. Machine M may need to fetch and update the cache for A if it doesn't exist in memory. After that, it returns the cache back to the original server.

If you are interested in this topic, you can check more about Memcached.

Summary

Cache can be a really interesting and practical topic as it's used in almost every system

nowadays. There are still many topics I'm not covering here like expiration policy.

If you want to know more about similar posts, check our system design interview questions collection.

Subscribe

We'll email you when there are new posts here.

Enter your email address here...

Subscribe

Related Posts:

1. **Design a Key-Value Store (Part I)**
2. **Design a Key-Value Store (Part II)**
3. **Design eCommerce Website (Part II)**
4. **Dropbox Interview – Design Hit Counter**

LEAVE A REPLY

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

Comment

Name *

Email *

Website

Post Comment

Duplicate Elements of An Array

Design a Recommendation System