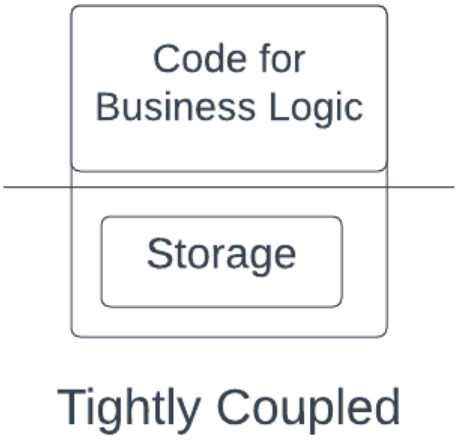
System Design - Caching

Agenda:

1. Whatsapp Group
2. Curriculum doc
3. AppsServer Layer
4. Caching
    - Where?
    - Invalidation?
    - Eviction?
    - Local caching case study

In the last class, we learned that when we type something on the browser(or search for a website specifically), the first thing browser has to do is talk to a DNS and figure out which IP address the browser should be talking to, and it communicates with the machine at that particular IP. And when you go from one machine to multiple machines, you need a load balancer to distribute traffic uniformly, but after a point when the amount of information cannot fit into a single machine, then the model needs to shard. It can be done using consistent hashing.

However, the machines had both the code and storage in the previous model. **Do you think it is a good model?**



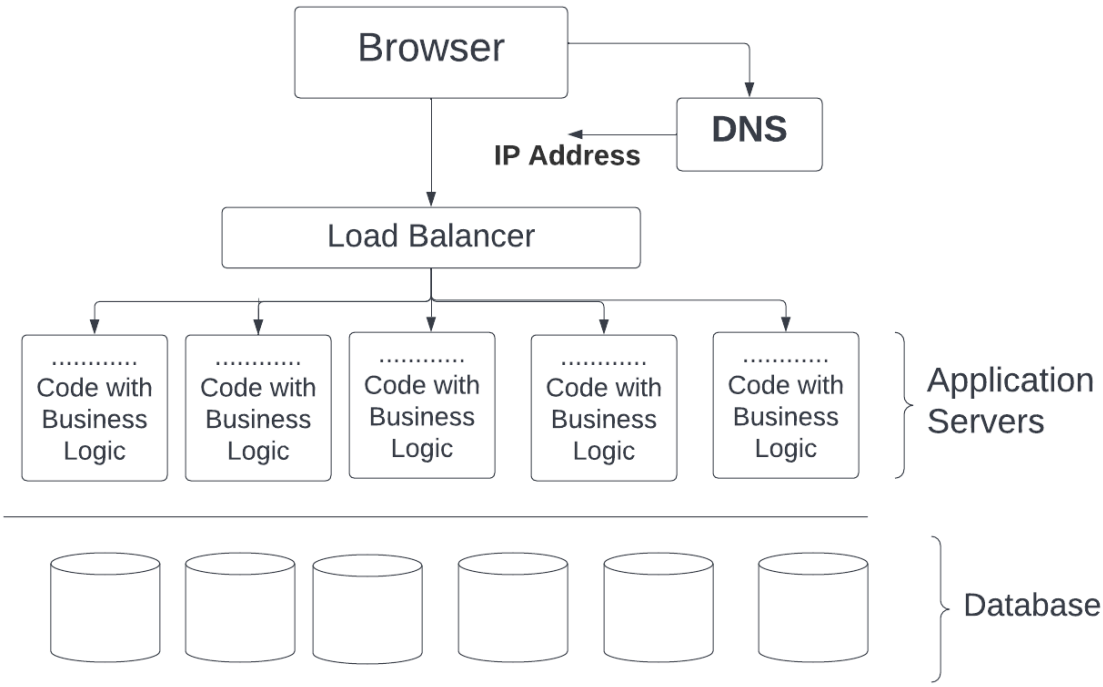
It's not, the reasons being:

1. Code and database are tightly coupled, and code deployments cause unavailability.
2. Fewer resources are available for the code since the database will also use some of the resources.

So it is better to decouple code and storage. However, the only **downside of decoupling** is the additional latency of going from one machine to another (code to the database).

So it can be concluded that it is not ideal for storing code and database on the same machine. The approach is to separate the code and storage parts to increase efficiency.

Different machines storing the same code running simultaneously are called **Application Server** Machines or App Servers. Since they don't store data and only have code parts, they are stateless and easily scalable machines.



Caching

The process of storing things closer to you or in a faster storage system so that you can get them very fast can be termed caching.
Caching happens at different places. The first place we start caching is the browser. Browser stores/caches things that you need to access frequently. Now let's look at different levels of caching.

1: In-Browser caching

We can cache some IPs so that browser doesn't need to communicate with the DNS server every time to get the same IP address. This caching is done of smaller entries that are likely not to change very often and is called **in-browser caching**. Browser caches DNS and static information like images, videos, and JavaScript files. This is why a website takes time to load for the first time but loads quickly because the browser caches the information.

2: CDN (Content Delivery Network)

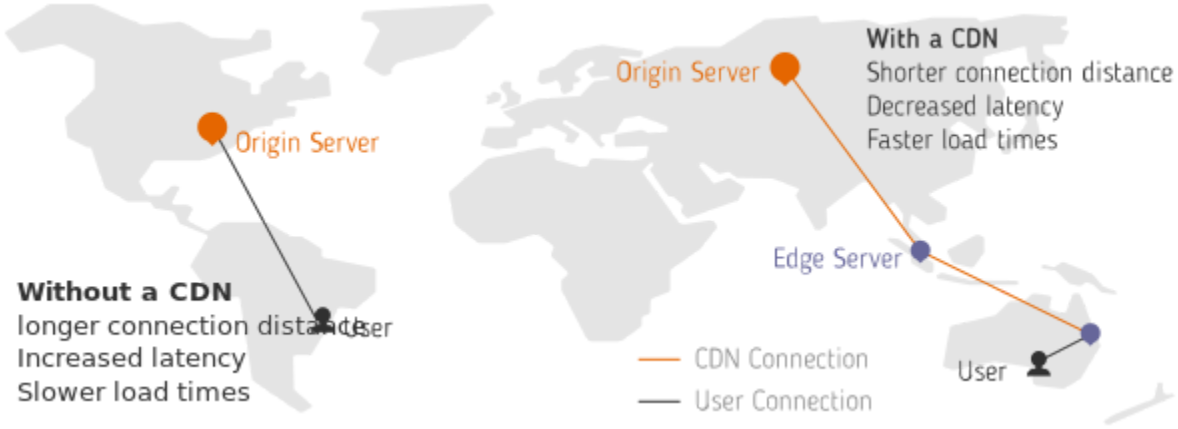
I will discuss fetching images/multimedia from the website (like **del.icio.us**). You have the browser in some region(say India), and you must fetch the files from the servers located in another region(say the US). When you try to access from your browser, a request is made to the load balancer, and then it goes to the application server and requests files from the file storage. You know that transferring files and other data will be fast for the machines in the same region. But it can take time for machines located on different continents.

From the website perspective, users worldwide should have a good experience, and these separate regions act as a hindrance. **So what's the solution?**

The solution for the problem is CDN, Continent Delivery Network. Examples of CDN are companies like

- Akamai
- Cloudflare
- CloudFront by Amazon
- Fastly

These companies' primary job is to have machines worldwide, in every region. They store your data, distribute it to all the regions, and provide different CDN links to access data in a particular region. Suppose you are requesting data from the US region. Obviously, you can receive the HTML part/ code part quickly since it is much smaller than the multimedia images. For multimedia, you will get CDN links to files of your nearest region. Accessing these files from the nearest region happens at a much larger pace. Also, you pay per use for using these CDN services.



One question you might think is **how your machine talks to the nearest region only**(gets its IP, not of some machine located in another region), when CDN has links for all the regions. Well, this happens in two ways:

1. A lot of ISP have CDN integrations. Tight coupling with them helps in giving access to the nearest IP address. For example, Netflix's CDN does that.
2. Anycast (https://www.cloudflare.com/en-gb/learning/cdn/glossary/anycast-network/)

This CDN process to get information from the nearest machine is also a form of caching.

3: Local Caching

It is caching done on the application server so that we don't have to hit the database repeatedly to access data.

4: Global Caching

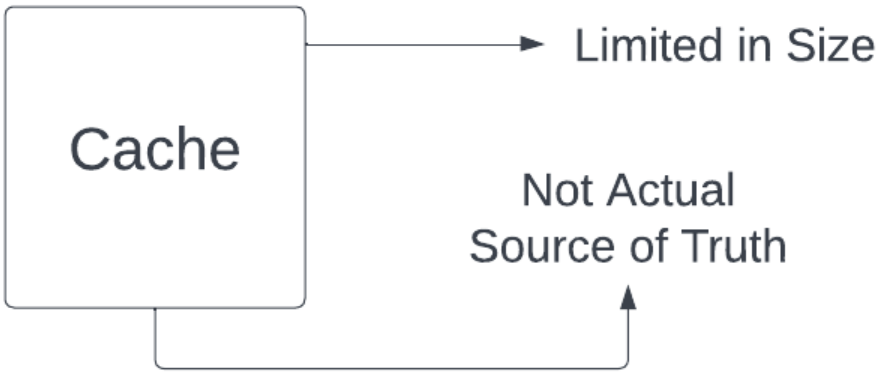
(This will be discussed in more detail in the next class)
This is also termed In-memory caching. In practice, systems like Redis and Memcache help to fetch actual or derived kinds of data quickly.

Problems related to caching

There are also two things related to the cache that you can derive from the discussions so far.

- Cache is limited in size

- It is not the actual source of truth; that is, the actual data is somewhere else. It stores a replica of data.



There are a few problems that you may face:

- Data can become stale and inconsistent with time (Data in Database - actual source of truth - changes. But is not reflected in cache)
- The cache can become full due to its small storage capacity.

So what do you think will be the solution to these two problems:

1. What do you have that doesn't become inconsistent?
2. How can you add entries if the cache is full?

We will be discussing ways to prevent these problems.

Case Invalidation Strategy

One solution that is proposed so that cache doesn't become stale is

TTL (Time to Live)

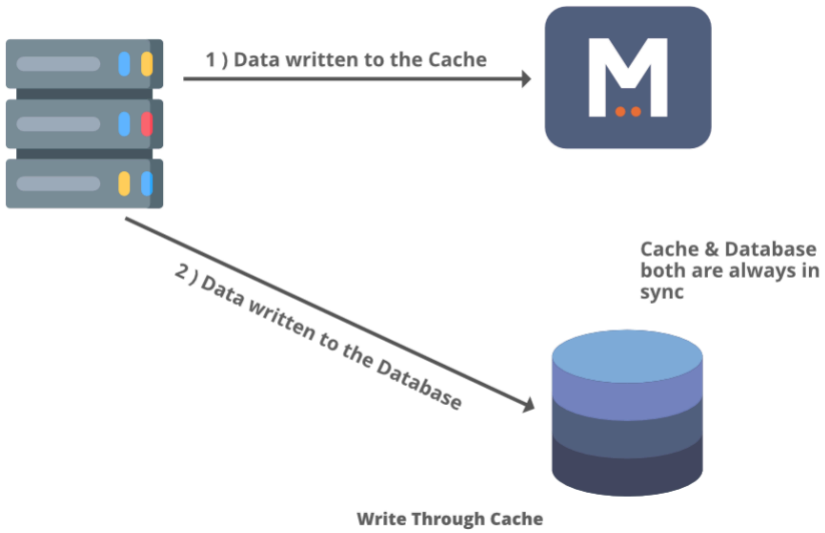
This strategy can be used if there is no problem with the cache being invalid for a very short time, so you can have a periodic refresh. Entries in the cache will be valid for only a period. And after that, to again get the entries, you need to fetch them again.
So, for example, if you cache an entry X at timestamp T with TTL of 60 seconds, then for all requests asking for entry X within 60 seconds of T, you read directly from cache. When you go asking for entry X at timestamp T+61, the entry X is gone and you need to fetch again.

We will look at more cache invalidation strategies in this doc through case studies.

Keeping cache and DB in sync

This can be done by the strategies like Write through cache, Write back cache, or Write around the cache.

Write through cache: Anything to be written is database passes from cache first(there can be multiple cache machines), storing it (updating cache), and then updating it to the database and returning success. If failed, changes will be reverted in the cache.
It makes the writing slower but reads much faster. For a read-heavy system, this could be a great approach.



There are other methodologies as well, like

Write back cache: First, the write is written in the cache. The moment write in the cache succeeds, you return success to the client. Data is then synced to the database asynchronously (without blocking current ongoing request).. The method is preferred where you don't care about the data loss immediately, like in an analytic system where exact data in the DB doesn't matter, and analytical trends analysis won't be affected if we lose data or two. It is inconsistent, but it will give very high throughput and very low latency.

Write around cache: Here, the writes are done directly in the database, and the cache might be out of sync with the database. Hence we can use TTL or any similar mechanism to fetch the data from the database to cache to sync with it.

Now let's talk about the second question: *How can you add entries if the cache is full?*

Well, for this, you be using an eviction strategy.

Cache eviction

There are various eviction strategies to remove data from the cache to make space for new writes. Some of them are:

- FIFO (First In, First Out)
- LRU (Least Recently Used)
- LIFO (Last In, First Out)
- MRU (Most Recently Used)

The eviction strategy must be chosen based on the data that is more likely to be accessed. The caching strategy should be designed in such a way that you have a lot of cache hits than a cache miss.

For the next class

Problem statements for you to think?
Que1: Local caching of test data on the app server. How to invalidate on the update?
Que2: How is facebook's newsfeed calculated, and how can you make it very fast?
(They both require a different kind of caching and will be discussed in the next class**)**

SUMMARY

1. It is not a good idea to have applications code and database on the same machine; hence we separated them.
2. Next, you learned caching and why it is necessary to get back to the user as soon as possible. Caching can happen at multiple levels. It can happen within the browser, on the CDN layer, or inside our systems, where you can cache data locally in app servers or have a global cache like central in-memory storage.
3. However, you still need to look at how you will invalidate the cache so that data should not be stale and the cache doesn't run out of space. For this TTL, write back, write through and write around can be used.

4. Lastly, you learned how to evict data so that you can store the most relevant information in the cache.
5. For the next class: Assignment to invalidate the cache when the test data changes.