Introduction to Caching and Caching Strategies

Have you ever experienced a website taking longer time to load on the first visit but significantly faster on subsequent visits? This is related to caching: An idea to improve the performance of distributed systems. In this blog, we will discuss the caching concept, how it works, and what are the important types of caching.

What is Caching?

Caching is a process of storing data in a temporary storage location called **cache** to improve the speed of data access. Now the question is: What is cache? A cache is a high-speed storage that stores a small proportion of critical data so that future requests for that data can be served faster.

Let's understand this from another perspective! Accessing data from the main database will take a good amount of time (due to disk I/O or network latency). So when data is accessed or processed for the first time, the system will store that data in a faster memory i.e. cache. If there is a further request for the same data, the system will serve it directly from the cache.

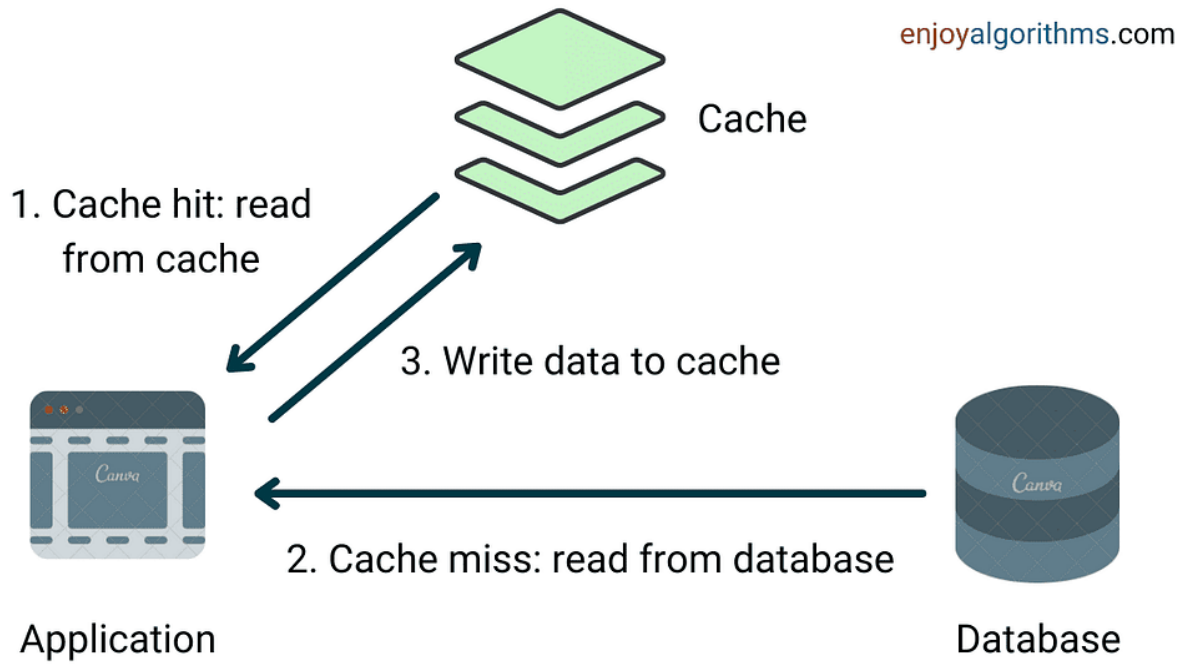
Cache memory is very costly, so its size is much small compared to the size of the main database. So what type of data should we store in the cache? One basic idea is to store the most frequently accessed data. Why? We highly recommend to explore the reason.

So one of the critical goals of caching is to make data retrieval fast by accessing frequently accessed data from the cache instead of retrieving it again from the main database. This will take significantly less time, which will improve the performance.

How Caching Works?

Now let's understand how caching works in practice when a user requests some data.

1. The system will first check the cache memory to see if the requested data is already there. If data is present (**cache hit**), the system will return the data directly from the cache. As mentioned earlier, cache access is much faster compared to the main database access.
2. If requested data is not present (**cache miss**) in the cache, the system will retrieve the data from the original database. As discussed earlier, this process is slower compared to the cache access.
3. Now, after fetching data from the database, the system will store the fetched data in the cache for future use, so that subsequent requests for the same data can be served more quickly.



The above idea is one of the commonly used caching strategies: [cache-aside strategy]. Here cache and database are independent and it is the responsibility of the application code to manage operations on cache and database. This is useful when we are dealing with read-heavy systems.

There can be other types of caching strategies like read-through strategy, write-around strategy, write-back strategy and write-through strategy. We will discuss these concepts later in this blog.

Note: Cache hits and misses are critical metrics for measuring cache performance.

- A high cache hit rate means the cache is effectively storing and retrieving the data.
- On the other hand, a high cache miss rate implies that the cache is not used effectively, which can result in slower performance. In this situation, we can think to adjust cache size or use a different replacement policy, or some other techniques to minimize cache miss.

Real-life Examples of Caching

- **Web browsers** use caching to store frequently accessed HTML, CSS, JavaScript, and images. This allows the browser to quickly retrieve these resources, rather than retrieving them from the server each time.
- **Content Delivery Networks (CDNs)** store static files like images and videos and serve them from locations closer to the user. This reduces the time it takes to retrieve data and reduces latency.
- **DNS caching** helps to improve the speed of accessing web pages by storing the IP address of a domain name in a cache. In other words, instead of performing a DNS query every time, an IP address can be quickly retrieved from the cache.

Cache Eviction Policy

Cache eviction policies are algorithms that manage data stored in a cache. When the cache is full, some data needs to be removed in order to make room for new data. So cache eviction policy determines which data to remove based on certain criteria. There are several cache eviction policies:

- Least Recently Used (LRU): Removes least recently used data.
- Least Frequently Used (LFU): Removes least frequently used data.
- Most Recently Used (MRU): Removes most recently used data.

- Random Replacement (RR): Randomly selects a data item and removes it to make space.
- First In First Out (FIFO): This algorithm maintains a queue of objects in the order that they were added into the cache. When a cache miss occurs, one or more objects are removed from the head of the queue, and a new object is inserted at the tail.

Solution of Cache Invalidation using various Caching Strategies

When data in the database is constantly being updated, it is important to ensure that the cache is also updated to reflect these changes. Otherwise, the application will serve outdated or stale data. So, we use cache invalidation techniques to maintain the consistency of the cache.

We mostly use five types of caching strategy to solve the problem of cache invalidation: Cache-aside strategy, Read-through strategy, Write-through strategy, Write-around strategy and Write-back strategy. We have already discussed the cache-aside strategy in the above section. So let's discuss the rest of them.

Read-through strategy

In this strategy, cache is present between the application code and the database. In case of a cache miss, it retrieves data from the database, stores that data in the cache and returns it to the application. So read-through caches are also one of the good choices for read-heavy systems.

Both cache-aside and read-through look similar, but there are differences. In cache-aside, application code is responsible for retrieving data from the database and storing it in the cache. In read-through, this is supported by the cache provider. So this strategy simplifies the application code by abstracting away the complexity of cache management.

In both strategies, the most basic approach is to write directly to the database. Due to this, cache will be inconsistent with the database. So

what is the solution?

- One idea is to use the time to live (TTL), so cache will continue serving stale data until expiry of TTL.
- If we need to ensure data consistency, we can follow the write strategy discussed below.

Write through strategy

In write-through cache, writes are first made to the cache and then to the database. If both writes are successful, write operation is considered successful. This will ensure that cache and database remain consistent and reduces the risk of data loss in case of a crash or system disruption.

- Write-through cache comes with the trade-off of increased write latency. This is because data must be written to two separate places! So this strategy is not a good choice for write-heavy applications.
- This is best for applications that frequently re-read data. Despite the increased write latency, this approach offers lower read latency and consistent data, which can compensate for the longer write time.

Write around strategy

In write-around cache, write operations bypass the cache and go directly to the database rather than being written to the cache first. This technique does not ensure that data in the cache is always up-to-date because updated data may not be available in the cache.

- This is used in systems where write performance is more critical than read performance, or where write activity is heavy and data is not frequently accessed. In such cases, cache is only updated when the data is read. This will reduce the number of write operations on the cache and the risk of cache overloading.
- This is not suitable for applications that frequently write and re-read the most recent data. In this situation, cache misses will occur frequently, which can cause slower read times or higher read latency.

Write back strategy

The write-back cache is used in systems with high write activity to improve write performance. Writes are temporarily stored in a cache layer, where they are quickly verified and then asynchronously written to the database. This results in lower write latency and higher write throughput.

However, this technique carries the risk of data loss if cache layer fails, because cache is the only copy of the written data. To minimize this risk, it is recommended to have multiple cache replicas that acknowledge the write. This way, if one cache fails, data can still be recovered from another cache.

Different types of caching used in system design

Let's move forward to understand some popular approaches to caching used in distributed systems. We apply these approaches at various stages to improve performance and scalability.

Browser caching

If you're wondering how websites load quickly on subsequent visits, one of the reasons is browser caching. This idea temporarily stores resources like images, HTML and JavaScript files within a cache in a web browser.

When you revisit the same website, the browser will retrieve these resources from the cache, rather than downloading them from the network again. This will result in a faster page load time and reduce the amount of data that needs to be transmitted. This is also known as **client-side caching.**

Browser cache has limited capacity and is set to store resources for a specific time duration. When the cache reaches its capacity or resources reach their expiration date, the browser will automatically clear the cache and retrieve updated resources from the network during the next visit. Users can also manually clear their browser cache if it becomes full.

Web server caching

Web server caching is used to improve performance by storing resources on the server side. This reduces the load on the server. There are several ways to implement web server caching i.e. reverse proxy cache and key-value store such as Memcached or Redis.

The **reverse proxy cache** acts as an intermediary between the browser and the web server. When a user makes a request, the reverse proxy cache checks if it has a copy of the requested data. If yes, it will serve a cached version to the user rather than forwarding the request to the web server.

We can also use **the key-value database** to cache application data. These databases are typically accessed by the application code. Unlike reverse proxies, which cache HTTP responses for specific requests, key-value databases can cache any user-specific data or frequently accessed data based on need.

Content Delivery Network (CDN)

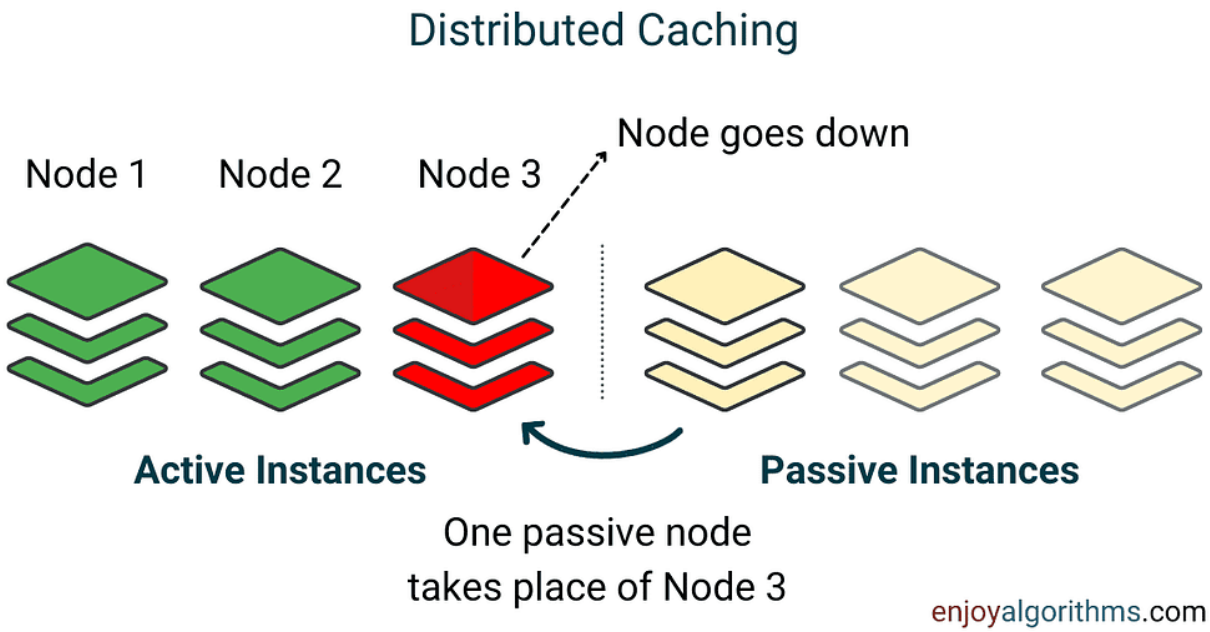
Content Delivery Network (CDN) is designed to improve the delivery speed of static content like web pages, images, videos, and other media files. These proxy servers are located in strategic locations around the world to reduce the distance between the end user and the origin server (reducing latency).

When a user requests content from a website that uses a CDN, CDN fetches the content from the origin server and stores a copy of it. If the user requests the same content again, CDN serves content directly rather than fetching it again from the origin server.

Think CDN like a chain of grocery stores: Instead of going all the way to farms where food is grown, which could be hundreds of miles away, customers can go to their local grocery store. Grocery store stocks food from faraway farms, allowing customers to get the food they need in a matter of minutes, rather than days.

Distributed caching

Distributed caching is the practice of using multiple caching servers that are spread across a network. Unlike traditional caches, which are usually limited to the memory of a single machine, distributed cache can scale beyond the memory limits of a single machine by linking together multiple machines (distributed clusters).



In distributed caching, each caching server maintains a portion of the cached data, and requests for data are directed to the appropriate server based on a hashing algorithm or some distribution strategy.

- Distributed caching is useful in an environment of high data access because distributed architecture can incrementally add more machines to the cluster. This will increase the cache size with the growth in data.
- Distributed caching provides several benefits: faster response times, scalability, increased availability, and better fault tolerance. It is used in large-scale applications and cloud-based architectures.

Advantages of caching

1. By storing frequently accessed data, caching reduces the need to retrieve data from slower data sources. This will result in faster access, quicker response times and better user experience.
    - Since the data is stored locally in the cache, it can be retrieved quickly without having to traverse the network. So caching will **reduce latency**.
    - Caching **increases throughput** because it provides much higher request rates compared to the main database. For example, a cache instance can serve hundreds of thousands of requests per second.
2. Caching offloads the backend databases by serving frequently requested data from the cache. Due to this, database servers can handle more requests simultaneously. If required, cache layer can scale horizontally by distributing the workload across multiple cache servers. So caching **improves scalability**.
3. By reducing the load on backend databases, caching **reduces the overall cost** of the system, especially if the backend database charges per throughput.
4. By storing relevant data in the cache, users can continue to access data or content even when the network connection is unreliable or completely offline. This is particularly useful for mobile applications. For example, if you install the EnjoyAlgorithms website as a progressive web app on mobile, you can access the content even if you are offline.

Disadvantages and limitations of caching

- When there is a data update in the main database, the cached copy will be outdated. So there is a risk of data inconsistency. For this, we need to carefully choose perfect cache invalidation or cache expiration strategies to ensure data consistency.
- Cache misses introduce latency! So it's important to keep them low compared to cache hits to maximize the benefits of cache. If cache misses are not well managed, the caching system can become

nothing more than overhead!

- When a cache is initially empty, it needs to "warm up" by populating it with frequently accessed data. This can cause temporary performance degradation because the cache needs time to build up its contents. During this time, users may experience slower response times until the cache becomes populated with the most accessed data.
- The caching only improves the performance of read operations. When it comes to the write operation, most of the time system needs to perform write on both cache and the main database. This process will not improve write performance.

Conclusion

In summary, caching is a useful technique for improving performance, reducing cost, and increasing the scalability of a system by storing frequently accessed data in fast storage called cache!

Thanks to Chiranjeev and Navtosh for their contribution in creating the first version of this content. If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy system design!