A Crash Course in Caching - Part 1

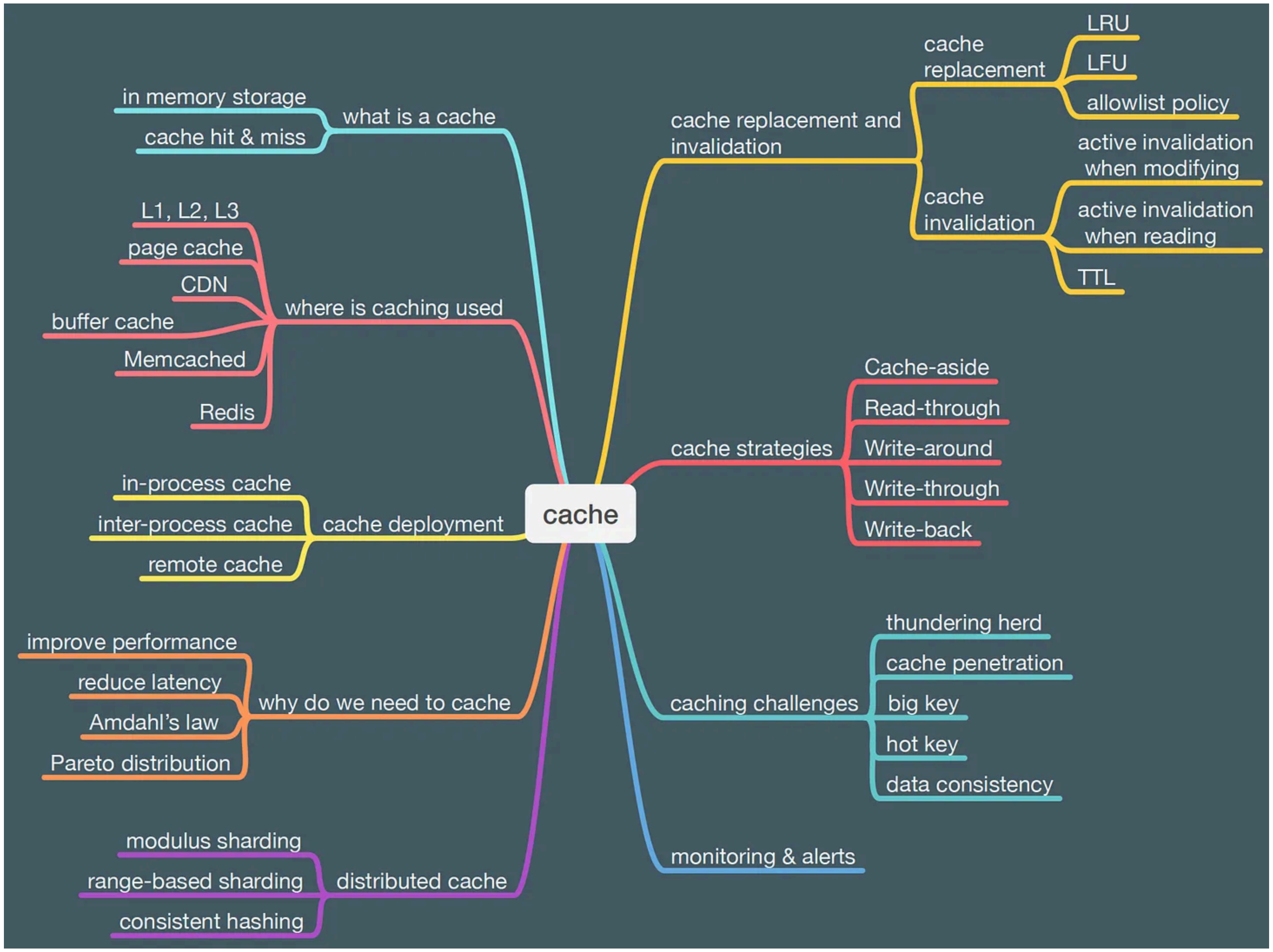
ALEX XU
MAR 15, 2023 · PAID

381 12 11 Share

Introduction

Caching is a fundamental technique in computing that enables quick retrieval of frequently accessed data. A study conducted by Amazon found that increasing page load time by just 100 milliseconds costs 1% in sales. By storing frequently accessed data in faster storage, usually in memory, caching improves data retrieval speed and improves overall system performance. This highlights the significant impact caching can have on the user experience and ultimately on business success.

The following is a list of the topics we will cover in this series.

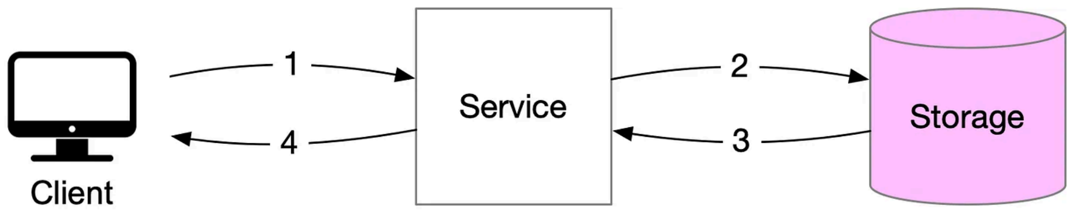


What is a cache

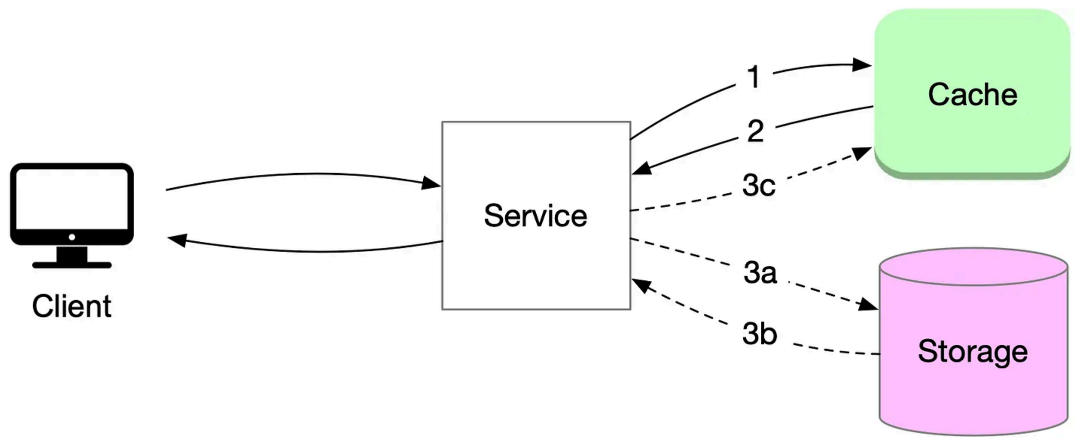
Definition: according to Wikipedia: "*In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be* *the result of an earlier computation or a copy of data stored elsewhere*".

Let's look at a concrete example.

When a client/server request is made without caching, the client sends a request to the service, which then retrieves data from storage and sends it back to the client. However, retrieving data from storage can become slow and overloaded with high volumes of traffic.



Adding a cache to the system can help overcome these limitations. When data is requested, the service first checks the cache for the data. If the data is present in the cache, it is quickly returned to the service. If the data is not found in the cache, the service retrieves it from storage, stores it in the cache, and then responds to the client. This allows for faster retrieval of frequently accessed data since cache usually stores data in memory in a data structure optimized for fast access.



Cache systems and storage systems differ in several ways. First, cache systems store frequently accessed data in memory. Storage systems store data on disks. Since memory is more expensive than disk, cache systems typically have much smaller data capacity than storage systems. This means that cache systems can store only a subset of the total data set.

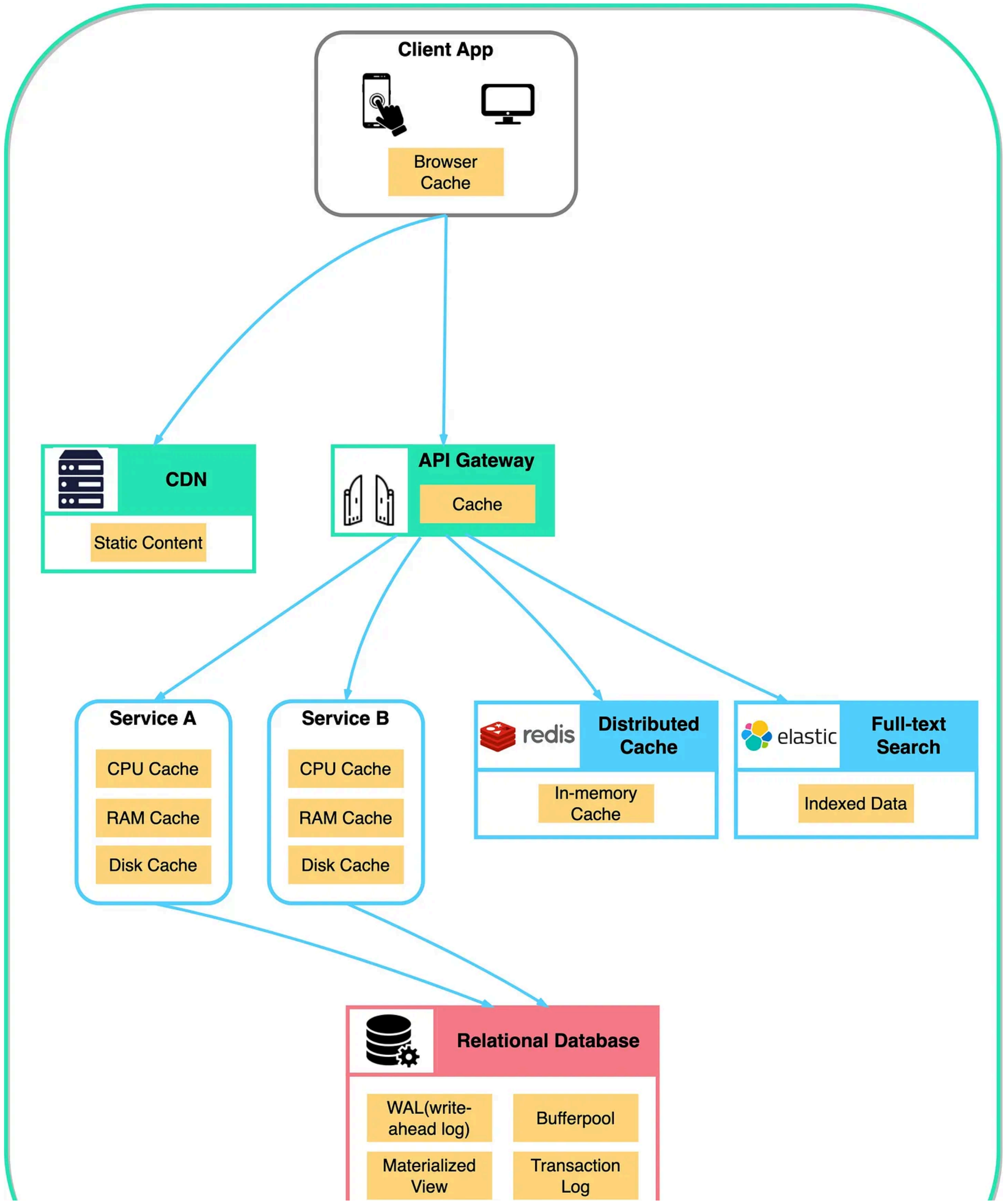
Second, the data stored in cache systems are not designed for long-term data persistence and durability. Cache systems are used to improve performance. They do not provide durable data storage. In contrast, storage systems are designed to provide long-term data persistence and durability, ensuring that data is always available when needed.

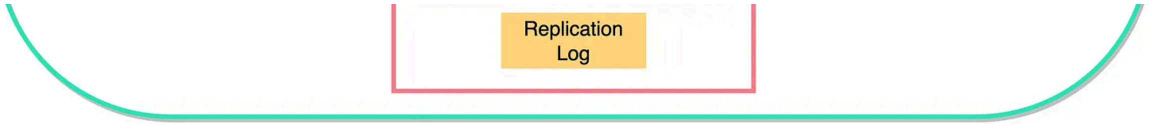
Finally, cache systems are optimized for supporting heavy traffic and high concurrency. By storing data in memory, cache servers can respond quickly to incoming requests, which is crucial for high-traffic websites or applications. Storage systems, on the other hand, are better suited for durably storing and managing large amounts of data.

If the data that is requested is available in the cache, it is known as a **cache hit**, otherwise, it is known as a **cache miss**. Because accessing data from the cache is significantly faster than accessing it from storage, the system overall operates more efficiently when there are more cache hits. The effectiveness of a cache is measured by the **cache hit ratio**, which is the number of cache hits divided by the number of cache requests. Higher ratios indicate better overall performance.

Where is caching used

While the previous example showcased caching in client/server computing, it is worth noting that caching is extensively used in modern computer and software systems.


Client App
Browser Cache
CDN
Static Content
API Gateway
Cache
Service A
CPU Cache
RAM Cache
Disk Cache
Service B
CPU Cache
RAM Cache
Disk Cache
redis
Distributed Cache
In-memory Cache
elastic
Full-text Search
Indexed Data
Relational Database
WAL(write-ahead log)
Bufferpool
Materialized View
Transaction Log



Modern computers utilize multiple levels of cache, including the L1, L2, and L3 caches, to provide fast access to frequently used data for the CPU.

The Memory Management Unit (MMU) is responsible for mapping virtual memory addresses to physical memory addresses. The MMU contains a specialized cache called the Translation Lookaside Buffer (TLB), which stores the most recently used address translations to speed up the address translation process.

The operating system employs a page cache in the main memory to enhance overall system performance. The page cache stores frequently accessed data pages and reduces the number of disk accesses, which can slow down the system.

By utilizing these different levels of cache, modern computers can improve their overall performance and efficiency.

In software systems, caching plays a crucial role in enhancing performance and reducing network latency.

Browsers use a cache to store frequently accessed website images, data, and documents, resulting in faster load times and a smoother browsing experience. This cache is commonly known as the browser cache.

Content Delivery Networks (CDNs) are another form of caching used to deliver static resources such as images, videos, CSS files, and other multimedia content. CDNs are a geographically distributed network of proxy servers that work together to deliver content from the nearest server to the user's location. This significantly reduces the time taken to access the content, resulting in a faster-loading website.

Caching in databases is essential for improving performance and reducing execution overhead. Some typical caches found in databases include the buffer cache, result cache, query cache, metadata cache, and session cache. These caches store frequently accessed data blocks, query results, metadata, and session-specific information in memory to reduce disk reads and query execution time, resulting in faster query response times and a better user experience.

Two widely used caching systems are Memcached and Redis. They are open-source, high-performance, distributed caching systems that can be used to store and retrieve data quickly.

Real-world use cases for caching include storing historical emails locally to avoid repeatedly pulling the same data from the server, caching popular tweets on social media websites like Twitter, and preloading and caching product information for flash sale systems to prevent excessive database pressure.

Caching is an effective solution for scenarios where data changes infrequently, the same data is accessed multiple times, the same output is produced repeatedly, or the results of time-consuming queries or calculations are worth caching.

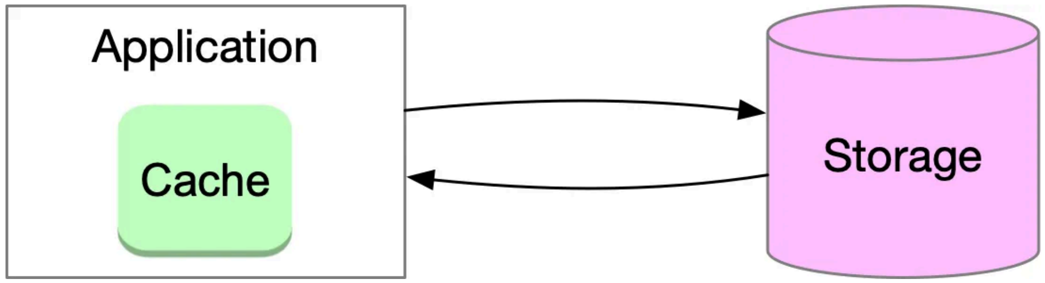
While there are many different types of caches in hardware and software, this article will focus on the general caching use cases in system design, exploring its principles, strategies, and problems.

Cache deployment

Caching is typically classified into three main categories based on where it is deployed: in-process cache, inter-process cache, and remote cache.

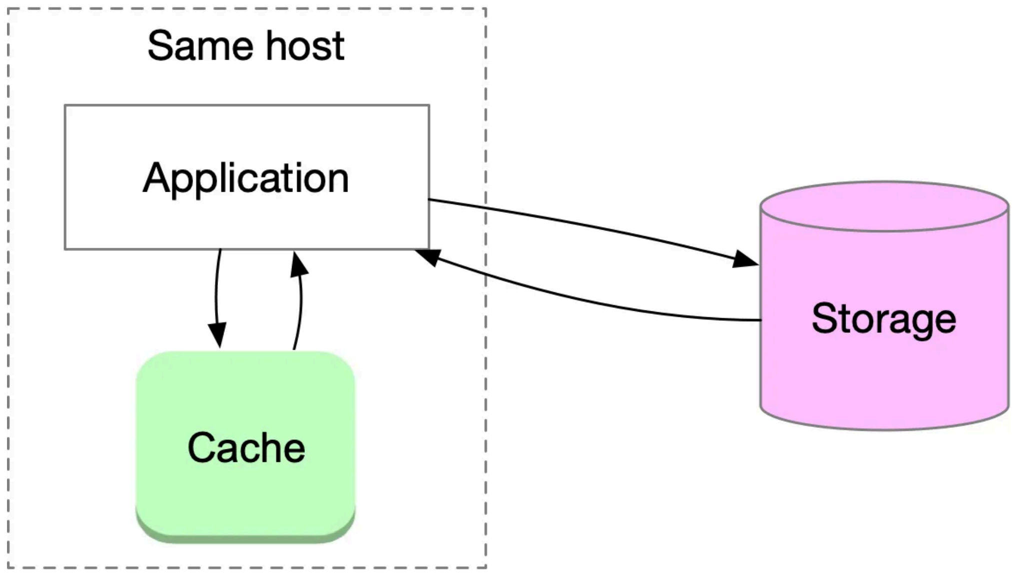
In-process cache

In-process cache is a type of cache that is located within the application itself. It provides high read and write performance with zero network cost. However, it has limited data capacity that is restricted by the memory size, and cached data is lost if the process restarts. In-process cache can be implemented as a library, such as Google Guava Cache.



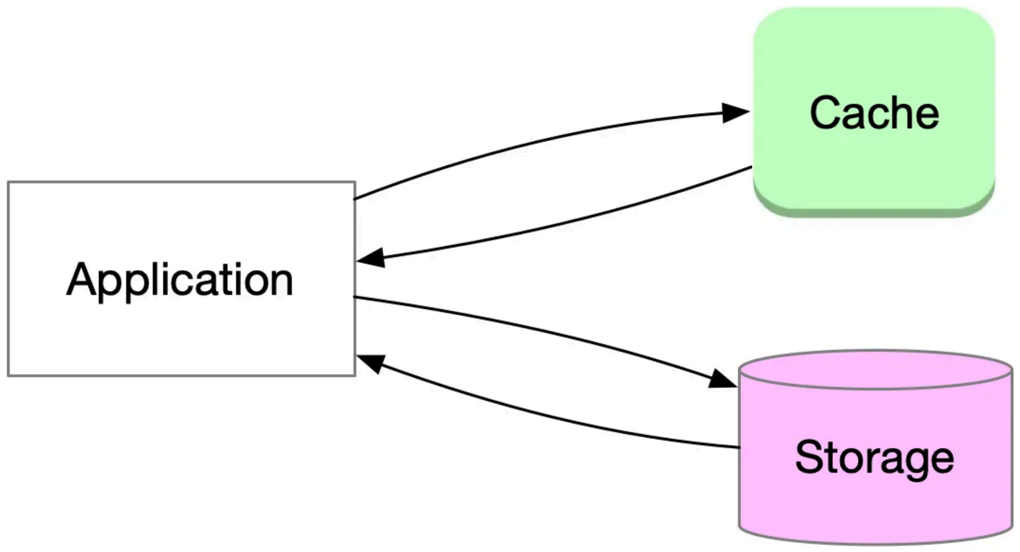
Inter-process cache

Inter-process cache, also known as local cache, runs in a separate process on the local machine. It has fast read and write speed, no network cost, and no data loss when the application process restarts. However, having the business logic and cache on the same host can complicate operation and maintenance costs, and cached data will be lost if the host crashes. Inter-process cache is usually implemented as locally deployed third-party storage, such as RocksDB, Memcached, and Redis in standalone mode.



Remote cache

Remote cache is deployed on a separate machine from the application, usually composed of multiple machines deployed together. It offers better scalability, ease of deployment and maintenance, and the ability to be shared by multiple applications. However, deploying a remote cache requires dedicated resources such as hardware, software, etc. Accessing data from a remote cache requires network communication, which can increase latency and reduce performance. Examples of remote cache include Memcached and Redis deployed on remote servers.

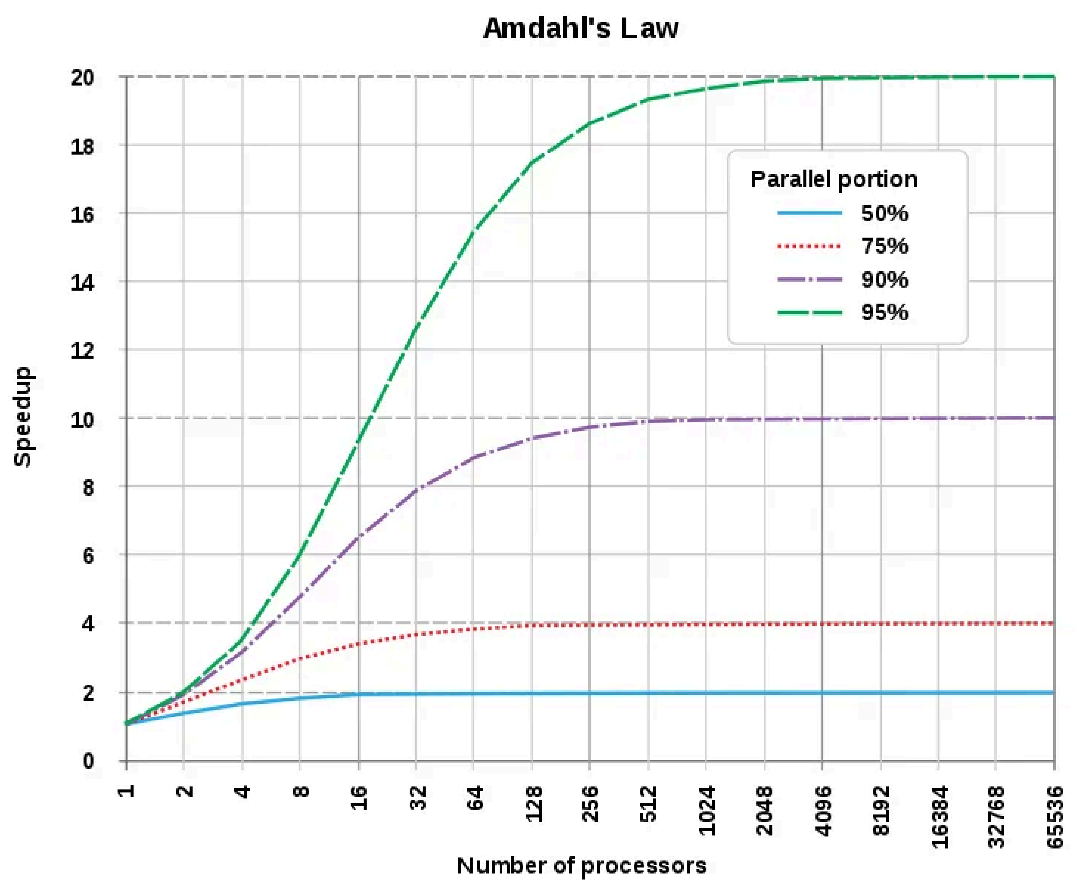


Why do we need to cache

Caching is a crucial technique in computing to improve performance and reduce latency. By storing frequently accessed data in a cache, the system can retrieve it more quickly and efficiently than accessing it from the authoritative storage.

When a request for data is made, the cache checks to see if it already has a copy of that data. If it does, the data is returned immediately, without accessing the storage. This means that subsequent requests for the same data can be served quickly, without incurring the overhead of accessing the storage again. This results in reduced latency and faster response times.

Amdahl's law states that the faster the cache is, the more speedup can be achieved. This means that by optimizing the performance of the cache, the system can achieve greater speed and efficiency.



Source: Wikipedia

On the other hand, the data access patterns likely follow the Pareto distribution (80/20 rule), where a small amount of data is responsible for a large amount of traffic. By caching this small amount of frequently accessed data, the cache can intercept most of the requests, reducing the load on the storage system. This can significantly improve the system's throughput and performance.

However, caching comes with some costs. It requires more resources and adds complexity to the system's implementation. In a distributed environment, maintaining data consistency and availability becomes a challenge, and edge cases must be considered.

Cache replacement and invalidation

Optimizing cache hits requires anticipating data access patterns and preloading the cache as much as possible. However, predicting these patterns accurately can be a challenging task. The principle of locality can be helpful in this regard.

Temporal locality refers to the tendency for recently accessed data to be accessed again in the near future. For example, in a social media application, hot tweets are likely to be frequently accessed.

Spatial locality, on the other hand, refers to the tendency for data that is close to recently accessed data to be accessed soon after. For example, in a database, when a value is taken from a sequence, it is likely that the next values in the sequence will be accessed later.

When traffic follows these patterns of locality, cache usage can be optimized, leading to more cache hits and improved performance. However, the capacity of the cache also affects its performance. A larger cache capacity allows for more data to be cached and for the data to remain in the cache for a longer period of time, resulting in better cache usage. This comes at the cost of increased space requirements and a tradeoff between space and cost.

Cache replacement

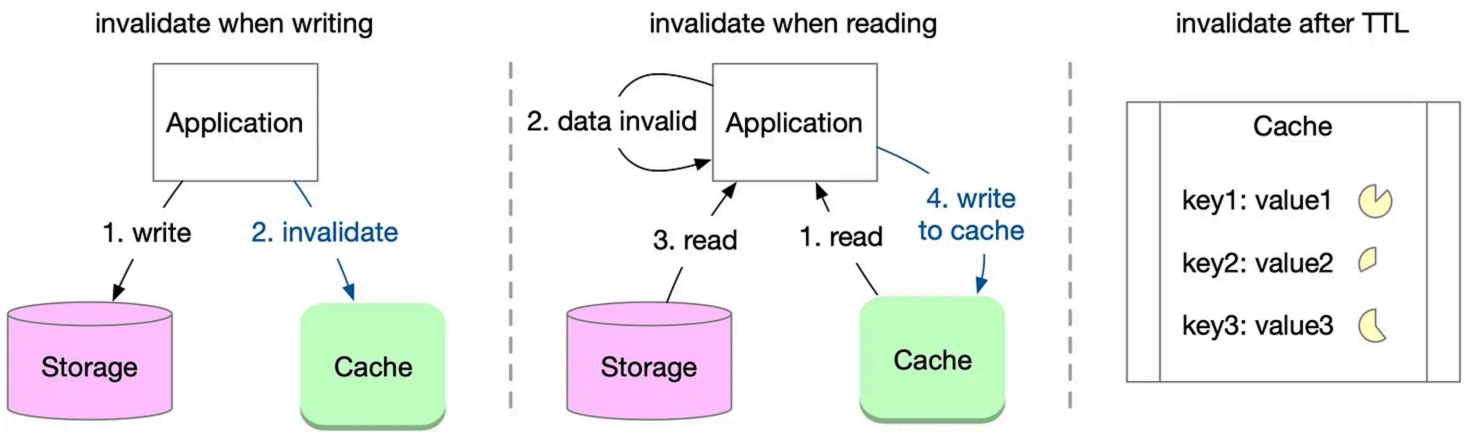
In practice, the cache has a limited capacity, so it's not possible to store all of the data in the cache. The best approach is to store the most frequently accessed data in the cache in order to maximize its effectiveness.

There are several cache replacement policies, including the Least Recently Used (LRU) policy, which discards the least recently used items first based on their last accessed timestamp. This policy is suitable for caching hot keys. The Least Frequently Used (LFU) policy, on the other hand, discards the least frequently used items first based on their usage count. This policy is often used for caching hot tweets, and is often used in conjunction with LRU to filter the most recent popular tweets.

The allowlist policy is another approach, where data items in the allowlist will not be evicted from the cache. This method is suitable for specific business scenarios with known hot keys, where popular data items can be identified in advance and preloaded into the cache.

Cache invalidation

Cache invalidation is a crucial process in maintaining data consistency between storage and cache. When data is modified, updating both storage and cache simultaneously can be challenging. Cached data may become stale, leading to incorrect results and potential performance issues. To solve this problem, cache invalidation techniques are used to remove or refresh stale data in the cache.



The most common cache invalidation strategies are:

1. Invalidation when modifying: This approach involves actively invalidating the data in cache when the application modifies the data in storage. It is a widely used technique.
2. Invalidation when reading: With this method, the application checks the validity of the data when it retrieves it from the cache. If the data is stale, it is read from the storage and written to the cache. While this approach is flexible, it can add more complexity to the application.
3. Time-to-live (TTL): The application sets a lifetime for the cached data, and the data is removed or marked as invalid once the TTL expires. This is a popular mechanism for managing cache invalidation

Cache invalidation is an essential part of cache management. The choice of cache invalidation strategy depends on the specific application needs and performance requirements. By using appropriate cache invalidation techniques, we ensure data consistency between storage and cache and improve the overall system performance.

To be continued.

381 Likes · 11 Restacks

12 Comments

Write a comment...

Julia Mar 16, 2023 · *edited Mar 16, 2023*

"There are only two hard things in Computer Science: cache invalidation and naming things."

Thanks for the good work. Read it through. Can I ask a few questions?

1. Cache invalidation. The 2nd approach invalidate on read is done through checking the cached content validity. But it doesn't say how to check validity. Like query both DB and compare the data with the cached data? Would that be extra cost to query both the data storage and cache, and increase latency too? Would be nice to hear the common approaches on validity check .

2.. Would it be possible to evaluate the pros and cons among the 3 ways of cache deployment, and use some examples to derive some guidelines of what typical use case requires which deployment?

3. Is it also possible to talk a little more on what are the common cache invalidation techniques in some often seen caches in engineering world?

For instance, for cache in user side, CDN, and remote cache, what are the good and typical practice to invalidate them?

LIKE (16) REPLY SHARE ...

3 replies

Yashasvi Girdhar Oct 29, 2023

part 2: https://blog.bytebytego.com/p/a-crash-course-in-caching-part-2

part 3: https://blog.bytebytego.com/p/a-crash-course-in-caching-final-part

LIKE (6) REPLY SHARE ...

10 more comments...