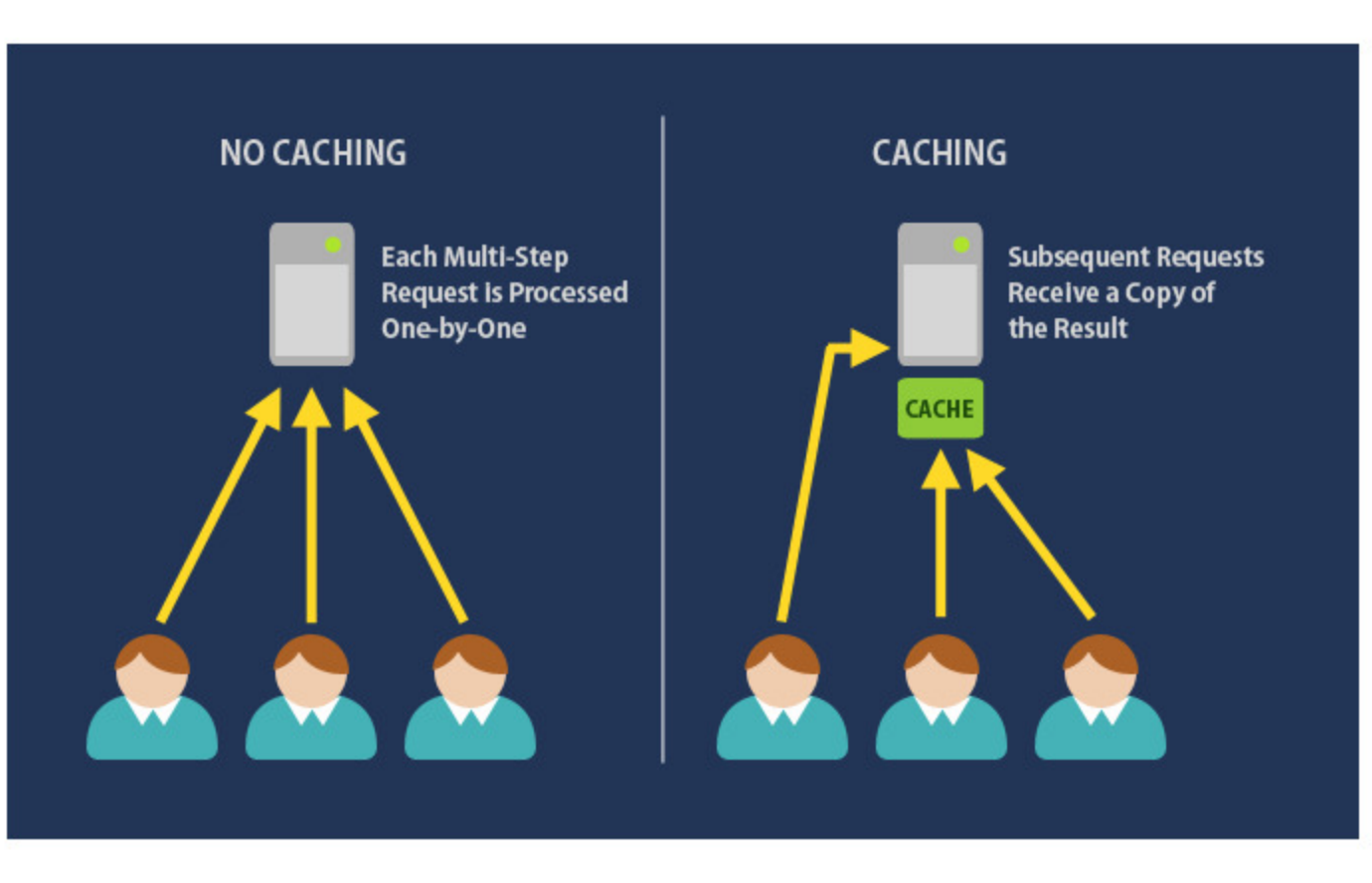
System Design — Caching

Peng Yang Follow
Apr 6 · 6 min read



References

- System design cache
- Wiki cache
- Introduction to architecting systems

Table of Contents

1. Concepts
2. Caches in different layers
3. Cache invalidation
4. Eviction Policy
5. Other

1. Concepts

- Take advantage of the locality of reference principle: recently requested data is likely to be requested again.
- Exist at all levels in architecture, but often found at the level nearest to the front end.
- A cache is like short-term memory which has a limited amount of space. It is typically faster than the original data source.
- Caching consists of
  1. **precalculating results** (e.g. the number of visits from each referring domain for the previous day)
  2. **pre-generating expensive indexes** (e.g. suggested stories based on a user's click history)
  3. **storing copies** of frequently accessed data in a faster backend (e.g. Memcache instead of PostgreSQL.

2. Caches in different layers

2.1 Client-side

- Use case: Accelerate retrieval of web content from websites (browser or device)
- Tech: HTTP Cache Headers, Browsers
- Solutions: Browser Specific

2.2 DNS

- Use case: Domain to IP Resolution
- Tech: DNS Servers
- Solutions: Amazon Route 53

2.3 Web Server

- Use case: Accelerate retrieval of web content from web/app servers. Manage Web Sessions (server side)
- Tech: HTTP Cache Headers, CDNs, Reverse Proxies, Web Accelerators, Key/Value Stores
- Solutions: Amazon CloudFront, ElastiCache for Redis, ElastiCache for Memcached, Partner Solutions

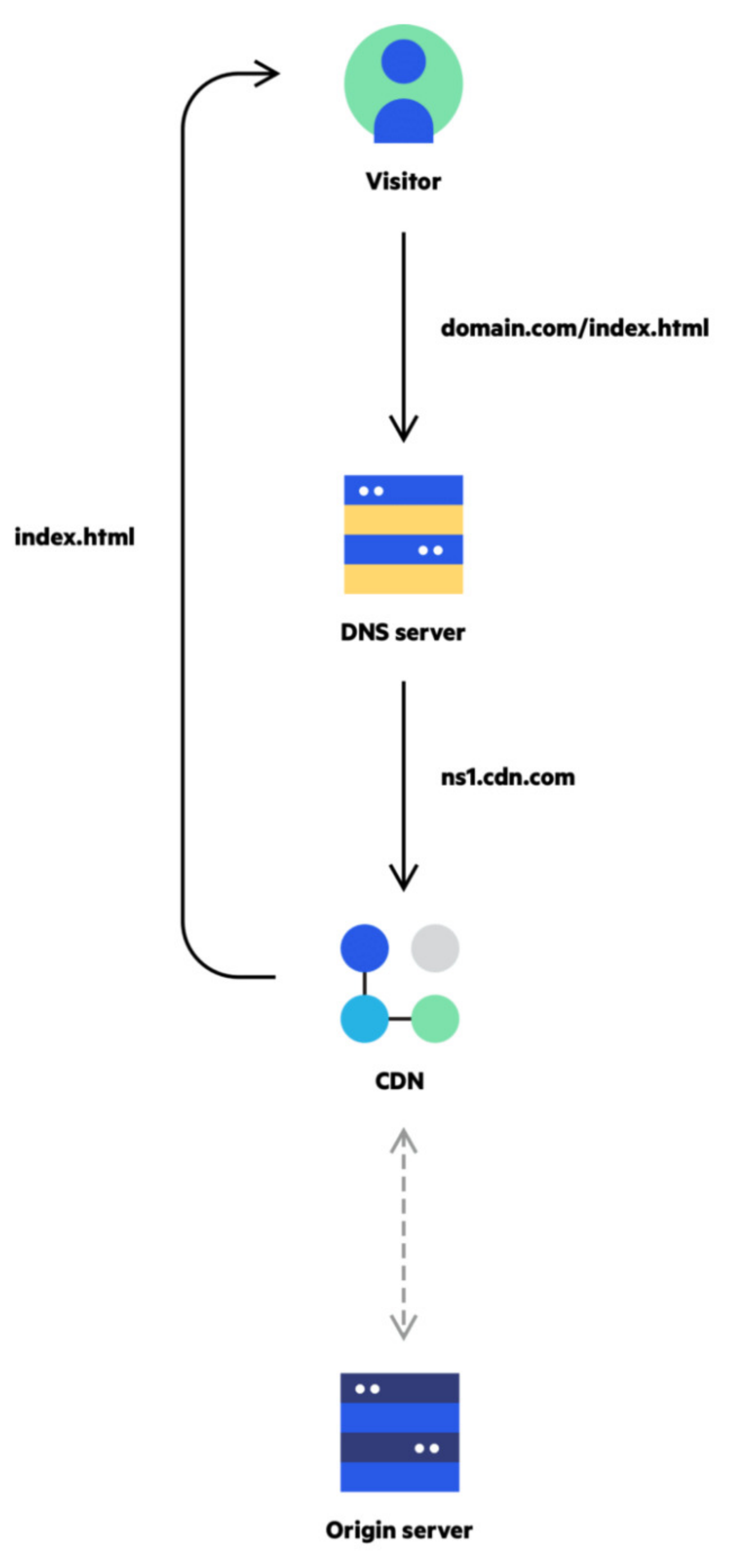
2.4 Application

- Use case: Accelerate application performance and data access
- Tech: Key/Value data stores, Local caches
- Solutions: Redis, Memcached
- Note: Basically it **keeps a cache directly on the Application server**. Each time a request is made to the service, the node will quickly return local, cached data if it exists. If not, the requesting node will query the data by going to network storage such as a database. When the application server is expanded to many nodes, we may face the following issues:
  1. The load balancer randomly distributes requests across the nodes.
  2. The same request can go to different nodes, increase cache misses.
  3. Extra storage since the same data will be stored in two or more different
  nodes. Solutions for the issues:
  1. Global caches
  2. Distributed caches

2.5 Database

- Use case: Reduce latency associated with database query requests
- Tech: Database buffers, Key/Value data stores
- Solutions: The database usually includes some level of caching in a default configuration, optimized for a generic use case. Tweaking these settings for specific usage patterns can further boost performance, can also use Redis, Memcached

2.6 Content Distribution Network (CDN)



- Use case: Take the burden of serving static media off of your application servers and provide a geographic distribution.
- Solutions: Amazon CloudFront, Akamai
- Note: **If the system is not large enough for CDN, it can be built like this:**
  1. Serving static media off a separate subdomain using a lightweight HTTP server (e.g. Nginx, Apache).
  2. Cutover the DNS from this subdomain to a CDN layer.

2.7 Other Cache

- CPU Cache: Small memories on or close to the CPU can operate faster than the much larger main memory. Most CPUs since the 1980s have used one or more caches, sometimes in cascaded levels; modern high-end embedded, desktop and server microprocessors may have as many as six types of cache (between levels and functions)
- GPU Cache
- Disk Cache: While CPU caches are generally managed entirely by hardware, a variety of software manages other caches. The page cache in main memory, which is an example of disk cache, is managed by the operating system kernel

3. Cache Invalidation

If the data is modified in the database, it should be invalidated in the cache, if not, this can cause inconsistent application behavior. There are majorly three kinds of caching systems: **Write-through cache, Write-around cache, Write-back cache.**

1. **Write through cache:** Where writes go through the cache and write is confirmed as success only if writes to DB and the cache BOTH succeed.
   Pro: Fast retrieval, complete data consistency, robust to system disruptions.
   Con: Higher latency for write operations.
2. **Write around cache:** Where write directly goes to the DB, bypassing the cache.
   Pro: This may reduce latency.
   Con: However, it increases cache misses because the cache system reads the information from DB in case of a cache miss. As a result of it, this can lead to higher read latency in case of applications that write and re-read the information quickly. Read must happen from slower back-end storage and experience higher latency.
3. **Write back cache:** Where the write is directly done to the caching layer and the write is confirmed as soon as the write to the cache completes. The cache then asynchronously syncs this write to the DB.
   Pro: This would lead to a really quick write latency and high write throughput for the write-intensive applications.
   Con: However, there is a risk of losing the data in case the caching layer dies because the only single copy of the written data is in the cache. We can improve this by having more than one replica acknowledging the write in the cache.

4. Cache eviction policies

Following are some of the most common cache eviction policies:

1. **First In First Out (FIFO):** The cache evicts the first block accessed first without any regard to how often or how many times it was accessed before.
2. **Last In First Out (LIFO):** The cache evicts the block accessed most recently first without any regard to how often or how many times it was accessed before.
3. **Least Recently Used (LRU):** Discards the least recently used items first.
4. **Most Recently Used (MRU):** Discards, in contrast to LRU, the most recently used items first.
5. **Least Frequently Used (LFU):** Counts how often an item is needed. Those that are used least often are discarded first.
6. **Random Replacement (RR):** Randomly selects a candidate item and discards it to make space when necessary.

5. Other

5.1 Global caches

All the nodes use the same single cache space (a server or file store). Each of the application nodes queries the cache in the same way it would a local one. However, it is very easy to overwhelm a single global cache system as the number of clients and requests increase but is very effective in some architectures.

Two forms of handling cache miss:

- Cache server handles cache miss, which is used by most applications.
- Request nodes handle cache miss:
  1. Have a large percentage of the hot data set in the cache.
  2. An architecture where the files stored in the cache are static and shouldn't be evicted.
  3. The application logic understands the eviction strategy or hot spots better than the cache.

5.2 Distributed caches

The cache is divided up using a consistent hashing function and each of its nodes owns part of the cached data. If a requesting node is looking for a certain piece of data, it can quickly use the hashing function to locate the information within the distributed cache to determine if the data is available.

- Pros: Cache space can be increased easily by adding more nodes to the request pool.
- Cons: A missing node can lead to cache lost. We may get around this issue by storing multiple copies of the data on different nodes.

5.3 Design a Cache System

- A single machine is going to handle 1M QPS
- Map and LinkedList should be used as the data structures. We may get better performance on the double-pointer linked-list on the remove operation.
- Master-slave technique

Thank you for reading! if you liked this blog, please also check my other blogs of System Design series.

- System Design — Load Balancing
- System Design — Caching
- System Design — Sharding / Data Partitioning
- System Design — Indexes
- System Design — Proxies
- System Design — Message Queues
- System Design — Redundancy and Replication
- System Design — SQL vs. NoSQL
- System Design — CAP Problem
- System Design — Consistent Hashing
- System Design — Client-Server Communication
- System Design — Storage
- System Design — Other Topics
- Object-Oriented Programming — Basic Design Patterns in C++

System Design Interview | Caching | Cache

5 claps

WRITTEN BY
Peng Yang Follow
Software/Infrastructure Engineer. Aim to become an engineer who understands computer science very well.
https://www.linkedin.com/in/peng-larry-yang-9a794561/

Computer Science Fundamentals Follow
Computer Science Fundamentals

Write the first response