**What is Caching**

Caching is used to serve the user requested resources/information more quickly. It is used to cache the data which is most often requested by the user. This will help us to improve the application performance.

Generally, when the user sent a request to the webserver the request will be served back with a response. The response will have the requested information this information is taken from the database.

So, this will increase the number of hits to the database server as a result the database server may crash. To avoid this when the user requested for an information for the first time, we will get the data from database and store it in a cache for a certain time period. When the user again requested for the same information within the cache time period instead of getting it from database, we will get it from the cache.

Let’s say we have a large complex computation in our application and the result of this computation is requested more often. So, instead of running the computation for each request we will run the computation on first request and cache this result. From the next request onwards, we will serve the result from the cache.

**\***Cache will use our RAM. We all know that we have two types of memory.

1. RAM (Random access memory)

2. ROM (Read only memory)

We all know that read/write operations are faster and more efficient in RAM over the ROM (Hard disk, SSD, etc.). So, accessing data from RAM is faster than accessing data from ROM/Hard drive.

Our application data is stored in database but this database is presented in ROM. So, reading data from database is slower than the reading data from ROM. But RAM has limited memory capacity and expensive, this capacity is not sufficient to store our application data. And also, RAM is volatile memory. This means what ever the data presented in RAM will persist until the termination of the application or shutdown the system. Once we close the application or shutdown the system the data stored in the RAM is erased.

So, we will store the data which is need or requested more often in the cache for a particular time period.

**Where can Cache added**

Cache can be implemented almost in all layers like Hardware, OS, web applications, web browsers etc.

**Types of Cache**

1. Application server cache
2. Distributed cache
3. Global cache
4. CDN

**Application server cache**

Application server cache is nothing but server-side caching. When user sent a request from client web browser to web server/application server (apache, nginx) for the first time it will get the data from database and servers back to the user then stores it into server machine cache.

When the user again requested for the same information it will check in cache first if it’s present in cache it will server to the user. But there is a problem with this approach when we have load balancers.

Now a days most of the applications are using load balancers to handle the load (request traffic) on the application to keep the application 100% availability.

Let’s say we have used application server cache and load balancers in our application, when the user requests a resource for the first time it will get data from database and servers to user and store it in cache for certain time period let say 5 min.

After 3 min we have sent request to the same resource/endpoint this time we have no guaranty that the request is routed to the same instance, it may route to the different instance because of load balancer. At this time moment this instance doesn’t have the data in cache. So, we have a cache miss. It again hit the database for the information.

So, to avoid this disadvantage we use Distributed cache or Global cache.

**Distributed cache**

In distribute cache each node/server will have its own in-memory cache. In distributed environment multiple nodes/servers are connected by a network. So, distributed means multiple servers are connected with each other through a network.

In the same way all individual caches of each server are connected / pooled together. In this way they know about which node/server cache contains what data.

distributed cache can grow beyond the memory limits of a single computer by linking together multiple nodes/servers–referred to as a distributed architecture or a distributed cluster.

Distributed caches are especially useful in environments with high data volume and load.

When we are using distributed cache, we need to think about 3 things.

Scalability

Availability

Performance

**Global/Centralized cache**

In global/centralized cache all nodes/servers are using same cache server. All these nodes are connected to a single cache and whatever the data requested is served from the global cache only.

In global cache we have 2 types of approaches while requested data not present.

1. Cache itself responsible for getting data which is not there, from database/disk etc.
2. Node/server is responsible for getting data which is not found in cache, from the database or disk etc.

When the user requested for a data 1st time it goes to cache, let assume it’s not found the data in cache.

In the 1st approach cache is responsible for cache miss.so, cache will get the data from database or disk, etc.

In the second approach node/web server is responsible for cache miss.so, server get the data from database, disk, etc. and served to client then cache the data.

**CDN (Content delivery network)**

A Content Delivery Network (CDN) is a critical component of nearly any modern web application. CDN merely improved the delivery of content by replicating commonly requested files (static content) across a globally distributed set of caching servers, and these are also called as edge locations.

CDN will reduce the load on an application origin and improve the experience of the requestor by delivering a local copy of the content from a nearby cache edge, or Point of Presence (PoP).

CDN mostly used for caching/delivering static content like images, videos, web pages etc. but now a days CDN can deliver dynamic content also.

Think of a CDN as being like a chain of grocery stores: Instead of going all the way to the farms where food is grown, which could be hundreds of miles away, shoppers go to their local grocery store, which still requires some travel but is much closer. Because grocery stores stock food from faraway farms, grocery shopping takes minutes instead of days. Similarly, CDN caches ‘’ the content that appears on the Internet so that webpages load much more quickly.

When the user requested a web application for his profile information for the 1st time, it will hit the application origin and servers back to user. This information is saved in the CDN edge server location which is closed to the requested user location. From the next time user requested same information the request goes through the CDN edge server if the data present then it will servers back to the user faster. If not found the data then it will forward the request to application origin and server the data back to user.

**Note:**

1. CDN cut down on bandwidth cost. Delivering content from CDN cache proxies removes the burden from the origin (backend) server, significantly reducing bandwidth costs associated with serving content to numerous visitors.

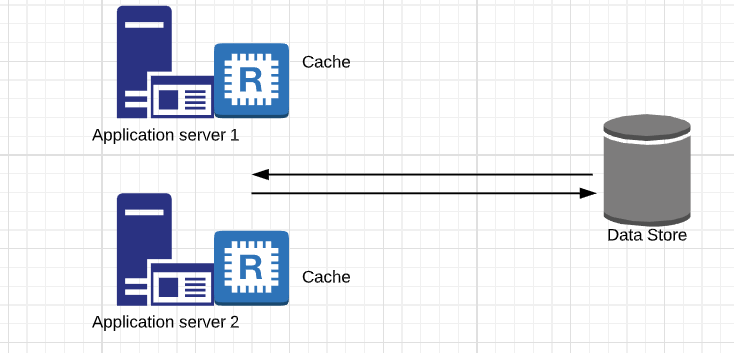
2. Improves user experience. A globally distributed network of cache proxy servers, CDNs bring your website’s content closer to all visitors, no matter where they are. Having this content delivered from a local server significantly improves access speed and user experience.

**Where to place the cache in our application architecture**

We can place cache close to the servers or close to the database.

Let’s say we have decided to **place cache close to the server**. How close we can place cache to the server. We can place cache in-memory of server. it will be faster if we use an in-memory cache with the server.

But there will be some problems with in-memory cache. Take this example



1. The first problem is the **Cache failure**.

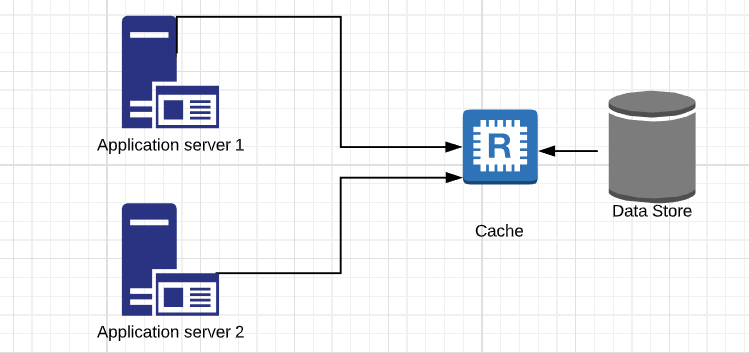
Assuming Application Server 1 failed. Now the Cache also failed. We will lose that data in Application Server1.

2. The second thing is the **consistency.**

Application server 1 data and Application server 2 data are not the same. They are not in sync. If there are some critical data you can’t keep this (Ex: Updated Password or any other credentials).

If we **place cache close to the database**

If we **place cache close to the database** using a **Global cache** the benefit is that all servers are hitting this global cache. If there is a miss it will query the database otherwise it will return data to the servers.



Now assume Application server 1 is failed. Then there is no problem with all other servers asking data from the Cache and the cache is still alive. And also, it is **more accurate**.

**Cache Invalidation**

We cannot cache the data for long period of time. We want the data to be cached upto certain time period after that the data to be invalid.

And we want the data to be invalid upon changing/updating the data in database.

To invalid cache data after certain time period we need to set TTL (Time to live) on the cached data. This means while caching the data we need to specify the TTL.

To invalid cache data upon changing/updating data in the database. We have 3 methods.

1. Write through cache.

2. Write around cache.

3. Write back cache.

**Write through cache**

Write through cache is when ever the write request or post/put request made the data is write into the database through cache. This means the data is write into cache and database at the same time. The acknowledgement is sent back only after successfully saving the data into cache and database.

**Adv:**

Write through cache minimizes the risk of data loss.

**Dis-adv:**

Write through cache will have higher latency because of writing data into cache and database at a time.

user writes to cache write to DB acknowledgement acknowledgement

cache

database

**Write around cache**

Write around the cache is when the write/post/put request made the data is write into the database directly. Acknowledgement will be sent back when data written into database successfully.

When the read/get request made for the 1st time we will have cache miss and data is pulled from database and cached it then servers the user.

cache

database

Write request

Read/get request

Write success acknowledgment

**Adv:**

This will reduce the latency of write operation we have seen in write through cache approach.

**Dis-adv:**

This will cause the cache miss for the 1st time get request made. The data need to pull from slower backend (i.e. database) for the first time.

**Write back cache**

Write back cache is when the write request is made the data is written first into cache and acknowledgment will be sent back immediately. After that the data which is written into cache will be written to permanent storage database asynchronously.

Write request

cache

New Data is written into DB async

database

Acknowledgment

**Dis-adv:**

This approach having risk of data loss in case of cache crash while writing data from cache to database.

**Cache eviction**

Cache eviction is nothing but cleaning cache. Let’s say we have a cache of size 1gb and we store some data in it. We keep on adding data into cache, at some point cache will be full there is no space to add new data. So, we need to clear some data from cache. This is called as cache eviction.

To perform cache eviction, we have different strategies.

1. FIFO

2. LIFO

3. LRU (Least recently used)

4. MRU (Most recently used)

5. LFU (Least frequently used)

6. Random replacement

Among all these approaches LRU is best and most commonly used.

**LRU (Least recently used)**

LRU is implemented based on the data structures hash table and doubly linked list.

To understand how LRU works please refer this video

<https://www.youtube.com/watch?v=DUbEgNw-F9c&list=PL1ny_dBrANDBTIkdjxk45XuFwPeQrPdcY&index=3>

**Internals of cache**

We know that our cache is present in RAM. RAM is a volatile memory where the data stored in RAM is not persistent. If something gone wrong and our system is restarted then we lose all the cached data. So, we need to make our cached data persistent and fault tolerance.

We have 2 approaches for this.

1. Regular interval snapshot

2. Log reconstruction

**Regular interval snapshot**

This is nothing but take a snapshot of our cache in regular time intervals and these are stored in hard disk. This is what Redis do to make cache fault tolerance.

**Log reconstruction**

This is nothing but logging all the cache requests. Let say we made write request to cache with key and value. After that we made get request with key to get value.

We log all these requests and store this log file in hard disk.