**Caching**

Some of the things that we'll cover in this module are:

1. What is Caching
2. How is it implemented
3. What factors should you consider to implement a good cache

How many times have we experienced slow or partial loading of data on certain websites, even when our internet connection seems alright? In such scenarios, cache certainly seems to be the answer to our woes.

What is a Cache?

*A cache is high-speed data access and storage layer that helps us fetch data that we had previously retrieved or computed.*

If we need frequent or repeated access to certain information that we have already queried from a service or a database, then instead of repeatedly querying the service, it's better to cache that information for subsequent use, for it to be readily available.

For Example, for Twitter users, we can cache all the information that is needed to load the homepage of a user (like followers, tweets, etc.). Caching will help avoid repeatedly querying the same service, caused by the refreshing of the browser by the user or other similar use cases. In this case, caching helps to reduce latency and improve resource utilization of servers.

When to Cache?

A cache is primarily used in the following scenarios:

* When the data involves a lot of computation, then to reduce latency and CPU utilization, it's good to cache the pre-calculated information for fast retrieval later on.
* To reduce frequent database or network (API) calls, it's beneficial to cache the previously fetched data for fast retrieval. This helps reduce latency as well as bandwidth requirements.

How Does Caching Work?

A cache is primarily used to store the most recently or frequently used data, in the hope that it will soon be fetched again. Caches are typically faster than databases and services when it comes to re-accessing this stored information. What makes them so fast is the fact that caches store data in SSDs and mostly RAMs which reduces the lookup time. However, this does not mean that we should cache everything.

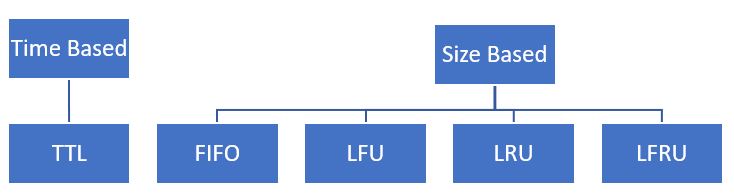
When Not to Cache?

There are some scenarios where the negative aspects of caching outweigh its benefits:

* **High Consistency requirements**: When we fetch the previously stored data from the cache, there is a possibility of stale data being displayed to the user. For example, for a social media app, then some stale data is probably fine. However, for a stock price display app, then the cache must be in sync with the primary data source.
* **Write heavy / Read Once**: When write operations (updates to data) are more frequent than read operations (data retrieval). For example, caching the data of an analytics system would only increase the hardware maintenance cost.
* **Low repetition**: When the action of retrieval of the same information is not frequently repeated by the user. For example, the cost calculated by the trip cost estimation module of a cab booking app between the exact two points need not be cached.

Cache Eviction Strategies

We need to regularly expel data from the cache to limit the size of the cache and to maintain its speed while ensuring that the entries are up to date.



Time Based

We keep an entry in the cache for some amount of time. In this strategy, we set a TTL (Time To Live). We will evict the entry from the cache after a certain pre-determined time has elapsed. The time which an entry stays in the cache before being evicted is called TTL.

Size Based

We keep at most some number of entries in the cache.

* **FIFO (First In First Out)**: We harness the FIFO property of the queue data structure to evict old entries.
* **LFU (Least Frequently Used)**: When we must evict an entry, we will evict the least frequently used entry.
* **LRU (Least Recently Used)**: When we must evict an entry, we will evict the least recently used entry.
* **LFRU (Least Frequently and Recently Used)**: We evict the least valuable entry, the one that's neither used frequently nor recently. This strategy gives the best results.

Metrics

We can use the following metrics to evaluate the performance of the cache:

* **Size**: Increasing the size of the cache usually increases the response time and therefore reduces its performance.
* **Latency**: The introduction of cache into the system must reduce latency from what it was earlier.
* **Cache Hit Rate**: It is the ratio of results found in the cache to the requests made to the cache. If the cache hit rate is low i.e., requests to cache are not returning the desired data, then it essentially slows down the system.

Out of the Box Solutions

It's usually not recommended to write your own cache implementation, because there are a lot of really good cache implementations out there that you can use. Some of the most popular caching products that are available in the market are:

* Ehcache
* Hazelcast
* Memcached
* Redis