  System Design Essentials | JUL 2021

**Assignment 2 | 18th July 2021**

**For any doubts regarding the assignment, ask questions in the** [**System Design**](https://community.letsupgrade.in/group/lsdpejul121) **Group in the Community.**

**Submit assignments at** [**https://forms.gle/bmo4q768B3tmDy6u5**](https://forms.gle/bmo4q768B3tmDy6u5)**.**

Name: Manish Panda

**Question 1**

1. Define the following cache strategies:

a) Cache Aside

b) Read Through

c) Write Through

d) Write behind

**Ans:**

a) Cache Aside Caching:

This is perhaps the most commonly used caching approach, at least in the projects that I worked on. The cache sits on the side and the application directly talks to both the cache and the database.

When cache-aside is used, the most common write strategy is to write data to the database directly. When this happens, cache may become inconsistent with the database. To deal with this, developers generally use time to live (TTL) and continue serving stale data until TTL expires. If data freshness must be guaranteed, developers either **invalidate the cache entry** or use an appropriate write strategy, as we’ll explore later.

b) Read-Through Caching:

When an application asks the cache for an entry, for example the key X, and X is not already in the cache, Coherence will automatically delegate to the CacheStore and ask it to load X from the underlying data source. If X exists in the data source, the CacheStore will load it, return it to Coherence, then Coherence will place it in the cache for future use and finally will return X to the application code that requested it. This is called Read-Through caching. Refresh-Ahead Cache functionality may further improve read performance (by reducing perceived latency).

c) Write-Through Caching:

Coherence can handle updates to the data source in two distinct ways, the first being Write-Through. In this case, when the application updates a piece of data in the cache (that is, calls put(...) to change a cache entry,) the operation will not complete (that is, the put will not return) until Coherence has gone through the CacheStore and successfully stored the data to the underlying data source. This does not improve write performance at all, since you are still dealing with the latency of the write to the data source. Improving the write performance is the purpose for the *Write-Behind Cache* functionality.

d) Write-Behind Caching:

In the Write-Behind scenario, modified cache entries are asynchronously written to the data source after a configured delay, whether after 10 seconds, 20 minutes, a day, a week or even longer. Note that this only applies to cache inserts and updates - cache entries are removed synchronously from the data source. For Write-Behind caching, Coherence maintains a write-behind queue of the data that must be updated in the data source. When the application updates X in the cache, X is added to the write-behind queue (if it isn't there already; otherwise, it is replaced), and after the specified write-behind delay Coherence will call the CacheStore to update the underlying data source with the latest state of X. Note that the write-behind delay is relative to the first of a series of modifications—in other words, the data in the data source will never lag behind the cache by more than the write-behind delay.

**Question 2**

Explain Message Queues and PUB- SUB Queues. Mention where they are used.

**Ans:**

Message Queues:

A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures. Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer. Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.

Message queues are used in RabbitMQ.

PUB- SUB Queues:

Publish/subscribe messaging, or pub/sub messaging, is a form of asynchronous service-to-service communication used in serverless and microservices architectures. In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic. Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability.