Caching Strategies

Caching patterns are the design patterns for integrating a cache into a system known as caching strategies. Cache-aside or gradual loading (a reactive strategy) and write-through are two standard methods (a proactive approach).

a). Write-Around Cache

In the write around the cache, data is directly written to the permanent storage, not the cache, so the write operations on the cache are reduced. Therefore, the cache line(the memory transferred to the cache) is invalidated without writing back because the permanent storage is already up to date. When the data is not found in the cache, the situation is called a cache miss.

Cache Miss: A cache miss happens when a data entry is requested from the cache, but the entry does not exist for the supplied key (or simply, a miss).

Advantages of write-around cache:

1. Faster write operations

Disadvantages of write-around cache:

1. Slow read operations due to a cache miss

b). Write-Through Cache

The order in which the cache is populated is reversed in a write-through cache.

1. The primary database is updated by the application, batch, or backend process.
2. The data in the cache is also updated immediately afterwards.

Advantages of write-through cache:

1. No Stale Data
2. High Consistency

Disadvantages of write-through cache:

1. Slow write operations

c). Write-Back Cache

Write back is when data is written in cache only when the data changes and writing is done only to the cache.

Advantages of write-back cache:

1. Write operation is very fast

Disadvantages of write-back cache:
1. Loss of data in case the cache fails

Cache Eviction Strategies
The sequence in which entries are removed from a full cache is determined by the eviction policy of the cache (used for removing or replacing the old data with the new data). It is used to make space so that new entries can be added.

1. LRU (Least Recently Used):
LRU (Least Recently Used) is a popular eviction policy that prioritises removing entries that haven't been used in a long time. To keep track of the relative utilisation of entries, LRU requires additional RAM.

2. MRU(Most Recently Used):
The entry which is most recently used is evicted. We keep track of the data most used and remove it and leave the one which is less used.

3. LFU (Least Frequently Used):
LFU (Least Frequently Used) prioritises the removal of the least frequently used entries. LFU, once again, necessitates additional memory to keep track of the utilisation of entries.

4. FIFO(First In First Out):
FIFO removes items from the cache in the order in which they were added. It evicts the first data that was added.

5. LIFO(Last In First Out):
LIFO evicts the last added data.

6. RR(Random Replacement):
This makes the cache random. It is used when all the elements have an equal probability of getting evicted.

Benefits of Caching:

1. Improve Application Performance

2. Reduce Database Cost
3. Reduce the Load on the Backend
4. Predictable Performance
5. Eliminate Database Hotspots
6. Increase Read Throughput (IOPS)

Disadvantages of Caching

1. Cache memory is expensive and has a limited capacity.
2. As information is stored in the cache, it makes the page heavy.
3. Sometimes the updated information does not show as the cache is not updated.

Applications of Caching:

Caches can be used in various technology levels, including operating systems, networking layers such as Content Delivery Networks (CDN) and DNS, web applications, and databases.

1. Many read-intensive application workloads, such as Q&A portals, gaming, media sharing and social networking could benefit from caching to improve latency and IOPS.
2. Database queries, computationally complex calculations, API requests/responses, and web artefacts such as HTML, JavaScript, and picture files are all examples of cached data.
3. An In-Memory data layer operating as a cache benefits compute-intensive applications that manipulate data sets, like recommendation engines and high-performance computing simulations.
4. Massive data sets must be accessed in real-time across clusters of servers that can span hundreds of nodes in these applications. Manipulation of this data in a disk-based store is a significant constraint for many applications due to the speed of the underlying hardware.