Handling Concurrent Requests With Message Queues

In this lesson, we will have an insight into how concurrent requests are handled with a message queue.

We'll cover the following

Using A Message Queue To Handle the Traffic Surge

In the distributed *NoSQL* databases lesson, we learned about Eventual Consistency & Strong Consistency. We discussed how both the consistency models come into effect when incrementing the value of a "*Like*" counter.

Here is a quick insight into a way where we can use a message queue to manage a high number of concurrent requests to update an entity.

When millions of users around the world update an entity concurrently, we can queue all the update requests in a high throughput message queue. Then we can process them one by one in a *FIFO First in First Out* approach sequentially.

This would enable the system to be highly available, open to updation & still being consistent at the same time.

Though the implementation of this approach is not as simple as it sounds, implementing anything in a distributed real-time environment is not so trivial. I thought I will just bring this approach up so that you guys can meditate upon this.

How Facebook Handles Concurrent Requests On Its Live Video Streaming Service With a Message

Its Live Video Streaming Service With a Message Queue?

Facebook's approach of handling concurrent user requests on its LIVE video streaming service is another good example of how queues can be used to efficiently handle the traffic surge.

On the platform, when a popular person goes LIVE there is a surge of user requests on the LIVE streaming server. To avert the incoming load on the server Facebook uses cache to intercept the traffic.

But, since the data is streamed LIVE often the cache is not populated with real-time data before the requests arrive. Now, this would naturally result in a *cache-miss* & the requests would move on to hit the streaming server.

To avert this, Facebook queues all the user requests, requesting for the same data. It fetches the data from the streaming server, populates the cache & then serves the queued requests from the cache.

This is a recommended read on Facebook's Live Streaming architecture.

Alright, moving on!! In the next chapter we will take a deep dive into stream processing.