Database Structure:



Data Contract from Client-Side:

{

"articleId": "00000000-0000-0000-0000-000000000000",

"userId": "00000000-0000-0000-0000-000000000000",

"currentState": false

}

Whenever a like record inserted/updated in the table, we will write a trigger to insert the data into activity Table

DataContract of Activity Table

{

"activityId": "00000000-0000-0000-0000-000000000000",

"activityType": null,

"articleId": "00000000-0000-0000-0000-000000000000",

"count": 0

}

Post Insertion we will update the Redis synchronously.

From the UI, we need to call the API from JS for every 1 sec to get the latest Likes Data (Just like the way the news feed in Instagram will get refresh and update us with new posts, similarly, we need the same mechanism to Hit our GetLikes API and POST Likes API to Fetch and update the Data in the Backend and REDIS)

* How can you improve the feature to make it more resilient against abuse/exploitation?

**First case: User keeps on hitting the like button and the count keeps on Increasing**.

**Second Case: Do we need to hit the API whenever the user hits like button Continuously without gap**

To address this, we will use Client-side UI. we will need a toggle (e.g., Current State – Liked when the user clicks the button once and Dislike when the user hits the same button again) here We will store the likes Count in the local storage in the front end and will use this data to toggle between the likes count when the user keeps on hitting the like button.

we will call the API, once in every 100ms so that the current state (at the 100th millisecond) will be the input to that particular API and the API will update the record based on that current state.

Also, while configuring the load balancer we will limit the number of requests per user for a time (e.g., User can hit the API for 100 times in 3 sec so after 3rd sec even though the user hits the button API would not get hit which makes it more resilient).

* How can you improve the feature to make it scale to millions of users and perform without issues?

**FirstCase: DB**

Since we have designed the Like table in such a way that no two users will be updating a single row which makes there is no LOCK in terms of updating a Row since each row updating is an individual entry. Here autoscaling our DB and provisioning our DB with High i/o throughput and making the DB available in multi-AZ.

For the Database we will use MASTER-SLAVE Architecture where there will be one master and the slaves will be the read replicas for our system. The data synchronization happens continuously from master to slave with the help of polling from the slave to the master. The slave will be in the separate availability zone from that of the master. At most one slave can be in the same AZ as master. If the master is down then one of the slaves become the master making the system Highly available.

**SecondCase: API**

We shall use RateLimiting i.e we will limit the number of requests calling the API which makes the performance better.

In this case when a million request hits the API, the server chokes, in order to solve this problem, we will configure an autoscaling group and attach it to our webserver which serves our application logic with a load balancer which follows Consistent Hashing Algorithm.

For the GetLikes call, I will use REDIS Cache and fetch the data from it and create Indexes whenever necessary to fetch the Data if the call went to the DB. More than that, we will use Sharding so that retrievals become even more faster.

Finally, using an autoscaling group and a Load Balancer along with REDIS and asynchronous API calls will resolve this problem.

* A million concurrent users clicking the button at the same time

Horizontal Scaling will solve this problem, we will create an autoscaling group and configure it in such a way that if the CPU utilization is more than 80% then create another instance of the same configuration. Also, when inserting the database, we will use sharding based on the region so that the retrieval and the Db operation will become much faster

* A million concurrent users requesting the article's like count at the same time

In this case, the DB level scaling will help. Like I mentioned in the above questions, using the master slave architecture and having quite a good amount of read replicas will solve these kinds of problems. One million users hitting the Same article count at the same time will obviously make the DB so, Caching is one of the best options here which can help in improving the performance. We need a smart load balancer which has the server configured with Rate limits and also in memory Cache. Therefore when a request is cached in the server, It need not go to the DB and fetch the Data.