5 ARCHITECTURAL PROBLEMS:

NAME: M. AALYAN MUGHAL

REG #: FA22-BSE-094

SECTION: 5B

DATE: 31 DECEMBER, 2024

Twitter's Scalability Crisis (2010)

The Issue:

In 2010, Twitter faced big problems because its system couldn't handle the increasing number of users and tweets. It would often crash during high-traffic events, like breaking news.

The Solution in Detail:

Twitter fixed this issue by changing how its system was built. They moved from one large, all-in-one system to smaller, separate parts that worked together. Here's how they did it:

1. **Breaking Down the System:**

   Twitter split its system into smaller parts, like a timeline service for tweets and a user service for accounts. Each part worked on its own, so if one had a problem, it wouldn't affect the others.

2. **Improving Databases:**

   Instead of using one big database, Twitter spread the data across multiple smaller databases. This made it faster and more reliable, especially during busy times.

3. **Using Caches:**

   Twitter stored frequently accessed data in temporary storage areas (caches) like Redis and Memcached. This reduced the need to keep asking the main database for the same information.

4. **Real-Time Updates:**

   They used tools like Apache Kafka to handle the large amount of tweets and interactions happening at the same time. This made updates to user timelines faster and smoother.

5. **Adding More Servers:**

   Instead of upgrading to bigger, more expensive servers, Twitter added more servers to share the workload. This made it easier and cheaper to grow.

6. **Keeping an Eye on Performance:**

   Twitter set up tools to monitor how well the system was working. These tools sent alerts if there were any issues, so they could be fixed quickly.

Outcome:

By making these changes, Twitter became much better at handling more users and tweets without crashing. The platform became more reliable and could easily grow as more people joined. These updates were part of a new version of their system, showing the improvements they made.

CODE EXAMPLE IN JAVA:

```
import java.util.*;
import java.util.concurrent.*;

// Service for handling user data
class UserService {
   private Map<Integer, String> userDatabase = new
ConcurrentHashMap<>();

   public void addUser(int userId, String username) {
      userDatabase.put(userId, username);
   }

   public String getUser(int userId) {
```

```
        return userDatabase.get(userId);
    }
}

// Service for handling tweets
class TweetService {
    private Map<Integer, List<String>> tweetDatabase = new
ConcurrentHashMap<>();

    public void addTweet(int userId, String tweet) {
        tweetDatabase.computeIfAbsent(userId, k -> new
ArrayList<>()).add(tweet);
    }

    public List<String> getTweets(int userId) {
        return tweetDatabase.getOrDefault(userId,
Collections.emptyList());
    }
}

// Caching layer to reduce database load
class CacheService<K, V> {
    private Map<K, V> cache = new ConcurrentHashMap<>();

    public void put(K key, V value) {
```

```
        cache.put(key, value);
    }

    public V get(K key) {
        return cache.get(key);
    }
}

// Message queue for real-time updates
class MessageQueue {
    private BlockingQueue<String> queue = new
LinkedBlockingQueue<>();

    public void produce(String message) {
        try {
            queue.put(message);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    public String consume() {
        try {
            return queue.take();
```

```
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return null;
        }
    }
}

// Main application to demonstrate the system
public class TwitterScalabilityExample {
    public static void main(String[] args) {
        UserService userService = new UserService();
        TweetService tweetService = new TweetService();
        CacheService<Integer, List<String>> cacheService = new
CacheService<>();
        MessageQueue messageQueue = new MessageQueue();

        // Adding users
        userService.addUser(1, "Alice");
        userService.addUser(2, "Bob");

        // Adding tweets
        tweetService.addTweet(1, "Hello, World!");
        tweetService.addTweet(1, "Learning scalability concepts!");
        tweetService.addTweet(2, "Hi there!");
```

```
        // Simulating caching for user tweets
        cacheService.put(1, tweetService.getTweets(1));
        cacheService.put(2, tweetService.getTweets(2));

        // Message queue for real-time updates
        messageQueue.produce("User 1 posted a new tweet!");
        messageQueue.produce("User 2 liked a tweet!");

        // Consuming messages from the queue
        System.out.println("Real-time Updates:");
        System.out.println(messageQueue.consume());
        System.out.println(messageQueue.consume());

        // Accessing cached data
        System.out.println("\nCached Tweets for User 1:");
        System.out.println(cacheService.get(1));

        System.out.println("\nCached Tweets for User 2:");
        System.out.println(cacheService.get(2));
    }
}
```