Evaluation of Uber's Design

Let's evaluate our design for the non-functional requirements.

We'll cover the following

- Fulfill non-functional requirements
  - Availability
  - Scalability
  - Reliability
  - Consistency
  - Fraud detection
- Conclusion

Fulfill non-functional requirements

Let's evaluate how our system fulfills the non-functional requirements.

Availability

Our system is highly available. We used WebSocket servers. If a user gets disconnected, the session is recreated via a load balancer with a different server. We've used multiple replicas of our databases with a primary-secondary replication model. We have the Cassandra database, which provides highly available services and no single point of failure. We used a CDN, cache, and load balancers, which increase the availability of our system.

Scalability

Our system is highly scalable. We used many independent services so that we can scale these services horizontally, independent of each other as per our needs. We used QuadTrees for searching by dividing the map into smaller segments, which shortens our search space. We used a CDN, which increases the capacity to handle more users. We also used a NoSQL database, Cassandra, which is horizontally scalable. Additionally, we used load balancers, which improve speed by distributing read workload among different servers.

Reliability

Our system is highly reliable. The trip can continue even if the rider's or driver's connection is broken. This is achieved by using their phones as local storage. The use of multiple WebSocket servers ensures smooth, nearly real-time operations. If any of the servers fail, the user is able to reconnect with another server. We also used redundant copies of the servers and databases to ensure that there's no single point of failure. Our services are decoupled and isolated, which eventually increases the reliability. Load balancers help move the requests away from any failed servers to healthy ones.

Consistency

We used storage like MySQL to keep our data consistent globally. Moreover, our system does synchronous replication to achieve strong consistency. Because of a limited number of data writers and viewers for a trip (rider, driver, some internal services), the usage of traditional databases doesn't become a bottleneck. Also, data sharding is easier in this scenario.

Fraud detection

Our system is able to detect any fraudulent activity related to payment. We used the RADAR system to detect any suspicious activity. RADAR recognizes the beginning of a fraud attempt and creates a rule to prevent it.

Meeting Non-functional Requirements

Requirements	Techniques
Availability	• Using server replicas • Using database replicas with Cassandra database • Load balancers hide server failures from end users
Scalability	• Horizontal sharding of the database • The Cassandra NoSQL database
Reliability	• No single point of failure • Redundant components
Consistency	• Strong consistency using synchronous replications
Fraud detection	• Using RADAR to recognize and prevent any fraud related to payment

Conclusion

This chapter taught us how to design a ride-hailing service like Uber. We discussed its functional and non-functional requirements. We learned how to efficiently locate drivers on the map using QuadTrees. We also discussed how to efficiently calculate the estimated time of arrival using routing algorithms and machine learning. Additionally, we learned that guarding our service against fraudulent activities is important for the business's success.

← Back
Payment Service and Fraud Detection in Ube...

Next →
Quiz on Uber's De

✓ Mark as Complete