Uber System Design

Uber is a ride-sharing service that uses a mobile application to connect passengers with drivers. The company is committed to improving its operations and services by introducing new features and technologies. The Uber system has several key goals, which include:

- Providing efficient and reliable service to meet market demand.
- Helping drivers find the most efficient routes.
- Detecting and preventing fraud to ensure safety for all users.
- Updating data in real-time to provide efficient service.

In this blog, we will explore various components to design Uber system.

Key Requirements

- To use the ride-sharing service, riders simply need to request a trip from their starting point to their destination. Once the ride request is made, nearby drivers will be notified and one of them will confirm the ride.
- To ensure efficient service, drivers are required to update their current position and availability frequently, so the service can match them with nearby riders.
- Once a ride is confirmed, both the driver and rider can track each other's current positions until the ride is completed. After the ride, the rider will receive a receipt that includes trip information such as the route map and pricing.
- To ensure riders are informed about their pickup, the service displays a pickup ETA for the rider when the ride is dispatched.

Capacity Estimation

- Let's assume we have 100M customers and 1M drivers, with 1M daily active customers and 100K daily active drivers.
- Let's also assume that there are 1 million daily ride requests, and all

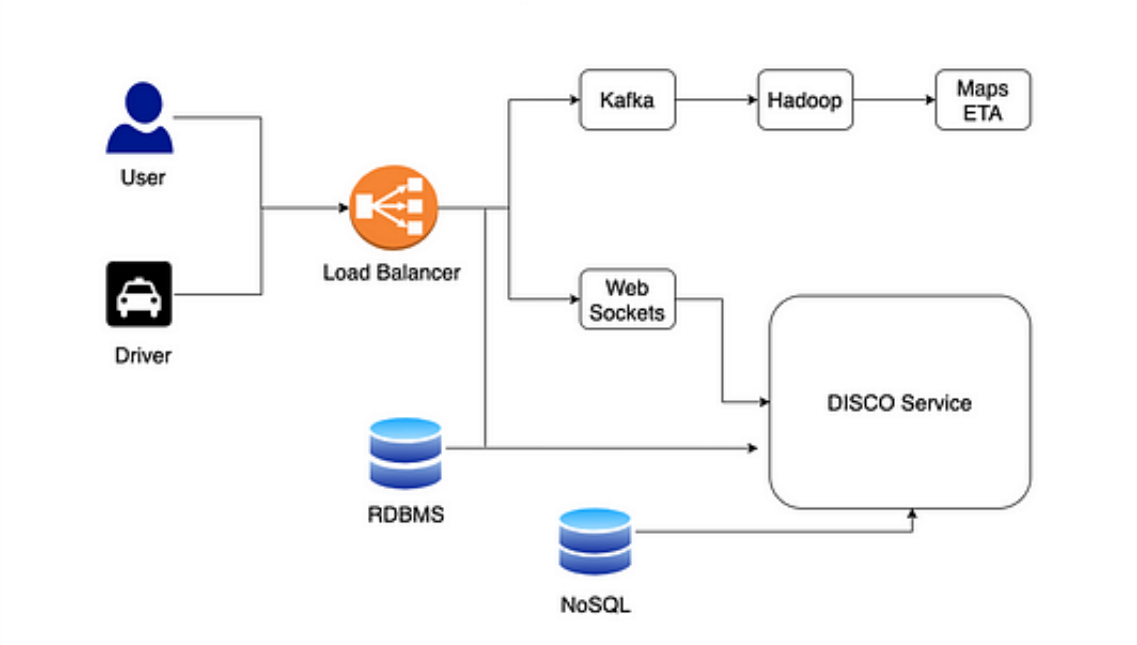
active drivers update their current location every 4 seconds.
- To ensure a smooth experience, the system must be able to instantly contact nearby drivers when a customer requests a ride.

High Level Design

The Uber app uses mobile devices to connect riders with drivers. Once a rider requests a ride through the app, the driver is directed to their location for pick-up. Behind the scenes, the service is supported by a large number of servers that handle and process significant amounts of data during each ride.

Over time, Uber's system design has evolved from a simpler monolithic architecture to a more complex, service-oriented architecture. The architecture consists of two main components: the Supply Service, which manages the availability of cars and drivers, and the Demand Service, which handles rider requests.

So the Real-time matching of riders and drivers is a crucial aspect of the Uber service. This is achieved through the Dispatch system, which uses data from the Supply and Demand Services to match riders with the nearest and most appropriate driver.



Data Storage

In its early days, Uber used a traditional relational database management system (RDBMS) to store data such as user profiles, GPS locations, and other information. However, as the number of users increased, the system struggled to process the vast amount of data in real time. To address this issue, Uber switched to using NoSQL databases, which provided better scalability and performance.

By switching to NoSQL, Uber's databases became horizontally scalable i.e. they could handle an increasing amount of data and users without compromising on performance. Additionally, NoSQL databases offer better write and read availability, allowing for faster and more efficient data processing. This technology change allowed Uber to continue to grow and serve an increasing number of users.

Detailed Component Design

In this section, we will dive deep into the design of components involved in Uber system.

Demand Service

The Demand Service in the Uber system is responsible for managing the requests made by riders through the app. When a rider requests a ride, the demand service receives the request through a web socket and begins to monitor the rider's GPS location. It also processes other types of requests, such as the number of seats required, preferred car type, and whether the rider is requesting a shared ride.

Using this information, the Demand Service matches the rider with the appropriate driver and vehicle, taking into account the rider's location and other preferences. It also tracks the supply and demand for rides in a specific geographic area to optimize the matching process.

Supply Service

The Uber system uses a combination of technologies to ensure an efficient supply of services to its customers. When a driver is on the road,

their cab's location is constantly updated and communicated to Uber's servers every 4 seconds. This information is sent through a web application firewall and load balancer, which helps to ensure that the data is secure and properly distributed.

Once the location data reaches the server, it is passed through Kafka's Rest APIs. This technology allows for the efficient transfer of large amounts of data and ensures that the server always has the most up-to-date location information for each cab.

Now location data is stored in the main memory of the relevant worker nodes. This allows for quick and easy access to information, which is essential for providing accurate and timely service to customers. Additionally, a copy of the location data is also transmitted to a database, which is used for dispatch optimization. This helps to ensure that customers are always matched with the closest and most available driver.

Dispatch System (DISCO)

The dispatch mechanism of the Uber system is based on map and location data, which requires correct modelling and mapping. Using latitude and longitude data to summarise and estimate places is tricky. However, summarizing and estimating places using latitude and longitude data can be challenging. To solve this issue, Uber use the Google S2 library. **Note:** DISCO had to meet several objectives like reducing extra driving, Minimising waiting time and Minimising overall ETA.

Let’s have a look at how a dispatch service works:

- The dispatch system begins by dividing the map data into small cells and assigning a unique ID to each cell using the S2 library. This is a simple technique that allows for efficient storage and distribution of data in a distributed system.
- The S2 library also allows for the creation of specific areas or "coverage" based on a given location. For example, if you want to find all the drivers within a 5 kilometers radius of a city, you can

create a 5km radius circle using the S2 libraries and it will filter out any cells with IDs that fall inside that radius.
- Next, the list of available cabs is given to the ETA (estimated time of arrival) algorithm, which uses the road system to compute the distance between the cab and the rider. The sorted ETA is then transmitted back to the supply system and offered to the driver.
- This process helps to minimize additional driving, reduce waiting time, and accurately predict the total ETA for the customer.

How is ETA Calculated?

The Estimated Time of Arrival (ETA) is a feature in the Uber app that allows riders to see approximately how long it will take for a driver to reach their pickup location before a trip begins. The app also displays an estimated time of arrival at the destination after the trip starts.

To calculate the ETA, the location data of nearby drivers are used to estimate the time it will take for them to arrive at the rider's pickup location. Once the trip begins, the app regularly updates the ETA for the destination. This provides the rider with a better idea of when they can expect to arrive at their destination. The ETA calculation also considers real-time traffic, weather, and other factors that may affect the trip duration.

Overall Summary: How Uber system works?

- When a user requests a ride, the request is sent through a web socket to the Demand Service. The Demand Service becomes aware of the need for a ride and submits a request to the Supply Service using the information from the ride. The critical question to think: Why Uber system uses web sockets?
- The Supply Service uses the information provided by the Demand Service, such as the rider's location, to determine which drivers are closest to the rider. It then sends a request to one of the servers in the server ring. These servers use ETA values to find out which cabs are closest to the rider.

- After the Supply Service computes the ETA values, it notifies the drivers through Web Sockets. If the driver accepts the request, the ride is allocated to the rider and driver. The driver will then pick up the rider and take them to their destination.

Analytics

Uber uses analytics to gain insight into the needs and behaviours of its customers and drivers, optimize its operations costs, and improve customer satisfaction. To achieve this, Uber employs a range of technologies and frameworks.

- Driver and rider location data are stored in databases such as NoSQL, RDBMS, or HDFS to enable analytics. For real-time analytics, the Hadoop platform provides several analytics-related tools.
- With HDFS, Uber can extract raw data from a NoSQL database and access HDFS data using query tools like HIVE. This enables Uber to analyze and interpret data to enhance its services and operations.

Surge Pricing

Surge pricing is a dynamic pricing strategy employed by Uber and other ride-hailing companies to increase the cost of rides during times of high demand. The aim of surge pricing is to incentivize more drivers to hit the road and provide more rides to meet the increased demand.

The algorithm used by Uber to determine surge pricing is based on the principle of supply and demand. The system employs real-time data on the number of drivers and riders in a particular area, as well as other factors such as traffic and weather, to determine the level of demand for rides.

When demand surges and the supply of drivers is low, the algorithm increases the price of rides to encourage more drivers to hit the road. The price may also change depending on the time of day, day of the week, and other factors.

Here are some general steps and insights behind this algorithm:

1. **Defining the base price:** Uber set the base price for the ride based on distance, time, and any other relevant factors.
2. **Determining the level of demand:** The system estimates the demand for rides in a particular area by analyzing historical data or real-time data.
3. **Determining the level of supply:** Now system estimates the supply of available drivers in the same area at that time.
4. **Calculating the surge multiplier:** Now system calculates the surge multiplier by dividing the demand by the supply. For example, if the demand is twice the supply, the surge multiplier might be 2.0x.
5. **Adjusting the price:** Finally, system adjusts the price of the ride by multiplying the base price by the surge multiplier.

System Optimization

Fast Searching

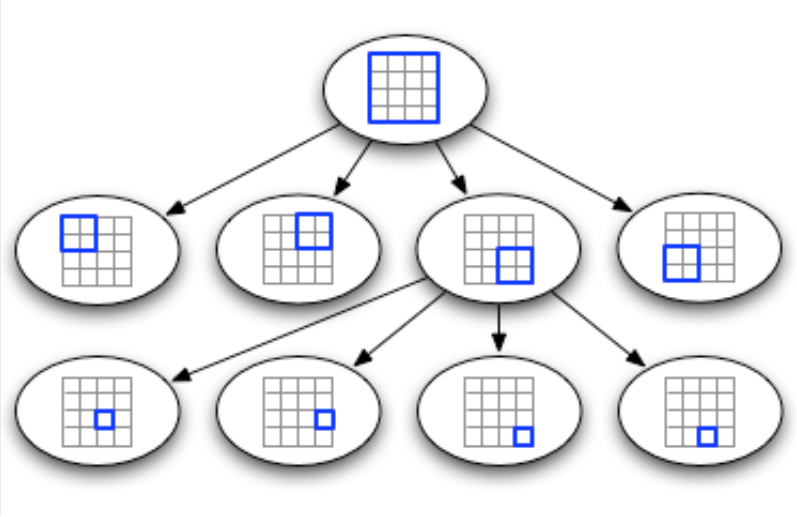
To improve the efficiency of the Uber system, it is necessary to reduce the number of locations it searches in the background. To accomplish this, we can implement a grid system where each grid has a maximum limit of 1000 locations (Or some other number). When a grid exceeds this limit, it is divided into four smaller grids of similar size using the **QuadTree** data structure.

The QuadTree is a tree-based structure that stores data about all the locations within its corresponding grid. When a node in the QuadTree reaches its 1000-location limit, it is divided into four child nodes, each storing a subset of the locations. This process is repeated until we reach the leaf nodes, which represent grids that cannot be divided further. In this way, we can efficiently search through the locations while keeping the number of locations in each grid to a manageable limit.

- To ensure that the positions of all current drivers are accurately reflected in our data structures, we need to update them in real time

as the drivers move. However, this can be time-consuming and resource-intensive. To achieve this, we must first locate the correct grid based on the driver's previous position and then update it with their new location. If the driver's new position is not in the same grid, we must remove them from the old grid and add them to the new one. If the new grid reaches its maximum driver limit after this transfer, we must repartition it to maintain the grid's size limit.

- Our system needs a fast and efficient way to communicate the current positions of all surrounding drivers to any active client in a specific region. Additionally, while a ride is in progress, the system must keep both the driver and the passenger informed about the vehicle's current location.
- To ensure that the information is up-to-date, we must continuously update QuadTree with the most recent driver positions. As all drivers report their positions every 4 seconds, we will receive many updates to our tree. So as soon as the server receives an update on a driver's position, it will notify the appropriate QuadTree server and all interested clients.



To efficiently share a driver's location with customers, we use a Push Model, where the server pushes the location updates to all relevant users. When a customer opens the Uber app on their phone and requests nearby drivers, we subscribe to all updates from those drivers on the server-side before sending the list of available drivers to the client. This way, we can keep track of all clients interested in knowing a driver's location at any

given time.

Whenever there is an update to a driver's position in the QuadTree, we broadcast their current location to all subscribing clients. This ensures that customers always see the most up-to-date information on a driver's location and helps make the search for nearby drivers more efficient and quick.

Fault Tolerance and Replication

To ensure high availability, we would need duplicates of our servers so that if the primary server fails, the backup server can take over. Additionally, we can store this critical data in permanent storage, such as SSDs with rapid I/O speeds, so that if both the primary and secondary servers fail, we can still recover the data from the persistent storage.

While data center failures are rare, Uber maintains a second data center to ensure that trips run smoothly. However, Uber does not replicate existing data into the backup data center, even though it has all the necessary components. To mitigate the effects of a data center failure, Uber uses driver phones as a source of trip data. When the driver's phone app connects with the dispatch system or an API call is made between them, the dispatch system transmits the encrypted data to the driver's phone app, which receives the data every time.

In the event of a data center failure, the backup data center will not have information on ongoing trips, so it will request the data from the driver's phone app. The data obtained from the driver's phone app will then be used to update the backup data center.

Conclusion

In this blog, we discussed how to design an Uber-like system. However, we have just covered only a few components in this blog. In the coming future, we will add more insights.

Additional blogs to explore:

- Building Uber's Alerting Ecosystem
- Scaling the Uber Engineering Codebase As We Grow
- The Uber Engineering Tech Stack, Part I
- The Uber Engineering Tech Stack, Part II

Thanks to Suyash for his contribution in creating the first version of this content. If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!