Published in Nerd For Tech

Kasun Dissanayake Follow

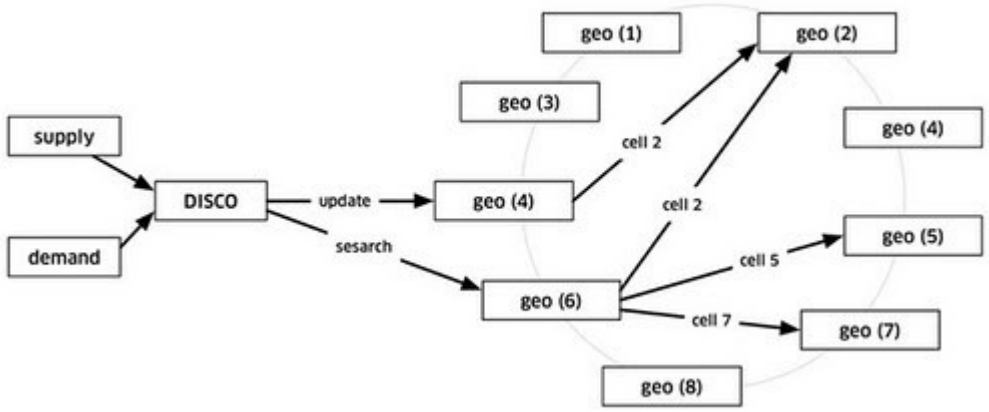
Mar 18, 2021 · 13 min read · Listen

Uber Architecture and System Design

In this article, we're learning about the Architecture and the system design of Taxi Application services like Uber. In the Uber Application when the rider(The person who wants a CAB) requests a driver on the App, the Driver goes to the place to pick that User. Behind the scene, there are 1000 servers which support the trip, and terabytes of the data have been used for the trip. When the beginning of the Uber company they had simple **monolithic architecture**. They had a backend service, frontend service, and database. Initially, they used **python** for application servers. They used a Python-based framework for asynchronous tasks.

After 2014 and now Uber's architecture has evolved to something called **service-oriented architecture**. Now Uber owns taxis as well as food and cargo. Everything has built into one system. Now the challenging thing for Uber or any taxi-related platform is to meet the supply to the demand or **demand to the supply**. the main task of the backend of Uber is to serve the mobile traffic. Because without the mobile phone it is pretty hard to run these services. Because everything works on **GPS**. The next thing is the Uber system is like a real-time marketplace to match riders to the cabs. So that means we need 2 different services in the architecture,

- **Supply Service**
- **Demand Service**



DISCO (Dispatch Optimization)

Let's talk about how this Dispatch system works. The dispatch system completely works on Map and Location data. That means we had to model our location data and map properly. It is pretty much hard to summarize and approximate locations using latitude and longitude data. To solve that Uber used the Google S2 library.

What is Google S2 Geometry Library?

A unique feature of the S2 library is that unlike traditional geographic information systems, which represent data as flat two-dimensional projections (similar to an atlas), the S2 library represents all data on a three-dimensional sphere (similar to a globe). This makes it possible to build a worldwide geographic database with no seams or singularities, using a single

coordinate system, and with low distortion everywhere compared to the true shape of the Earth. While the Earth is not quite spherical, it is much closer to being a sphere than it is to be flat!

S2 Features

Notable features of the library include:

- Flexible support for spatial indexing, including the ability to approximate arbitrary regions as collections of discrete S2 cells. This feature makes it easy to build large distributed spatial indexes.
- Fast in-memory spatial indexing of collections of points, polylines, and polygons.
- Robust constructive operations (such as intersection, union, and simplification) and boolean predicates (such as testing for containment).
- Efficient query operations for finding nearby objects, measuring distances, computing centroids, etc.
- A flexible and robust implementation of snap rounding (a geometric technique that allows operations to be implemented 100% robustly while using small and fast coordinate representations).
- A collection of efficient yet exact mathematical predicates for testing relationships among geometric primitives.

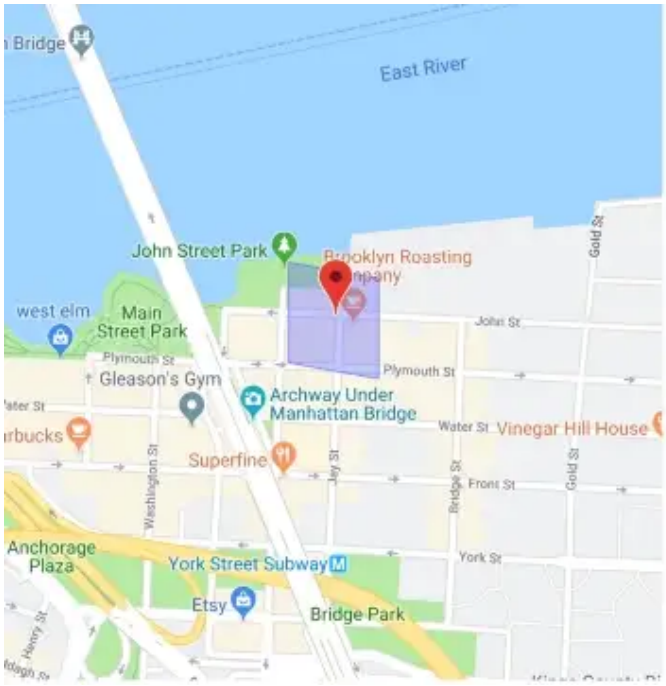
- Extensive testing on Google's vast collection of geographic data.
- Flexible Apache 2.0 license.

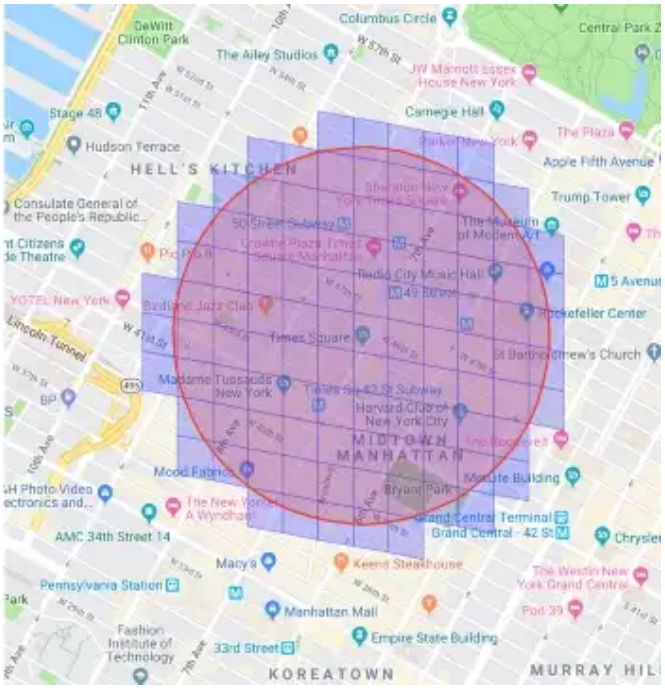
Overview

S2 is a library for spherical geometry that aims to have the same robustness, flexibility, and performance as the very...

s2geometry.io

This library takes spherical map data and it decides this data into tiny cells (**Ex: 1km x 1 km cell**). Once we join these cells we can get the complete map. Each cell has a unique ID. This is the easiest way to store data in a distributed manner. You just need the ID of the cell to get the location data from the servers (**We can use consistent hashing to store data**).

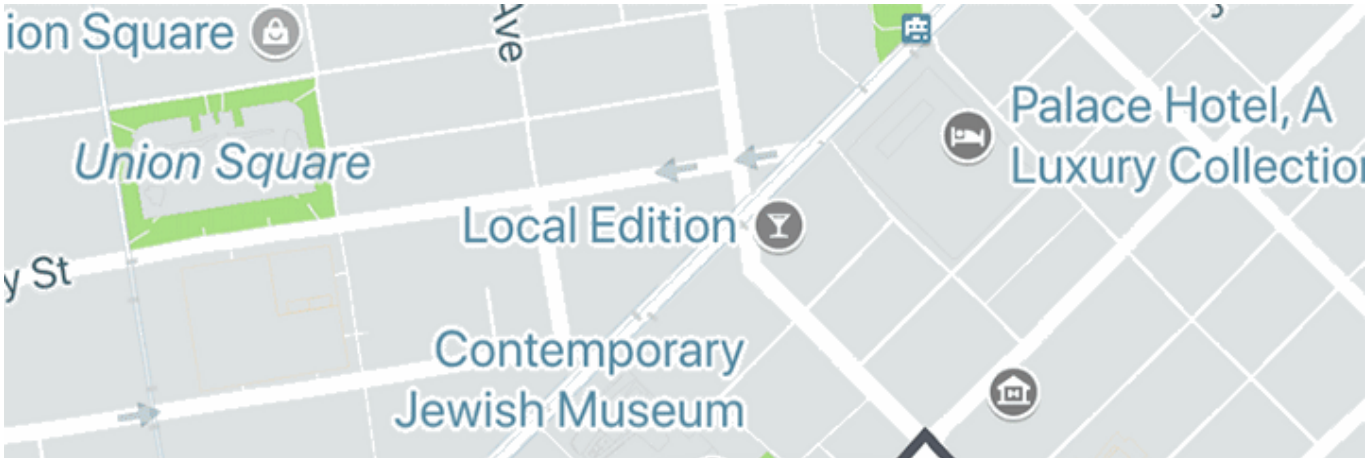


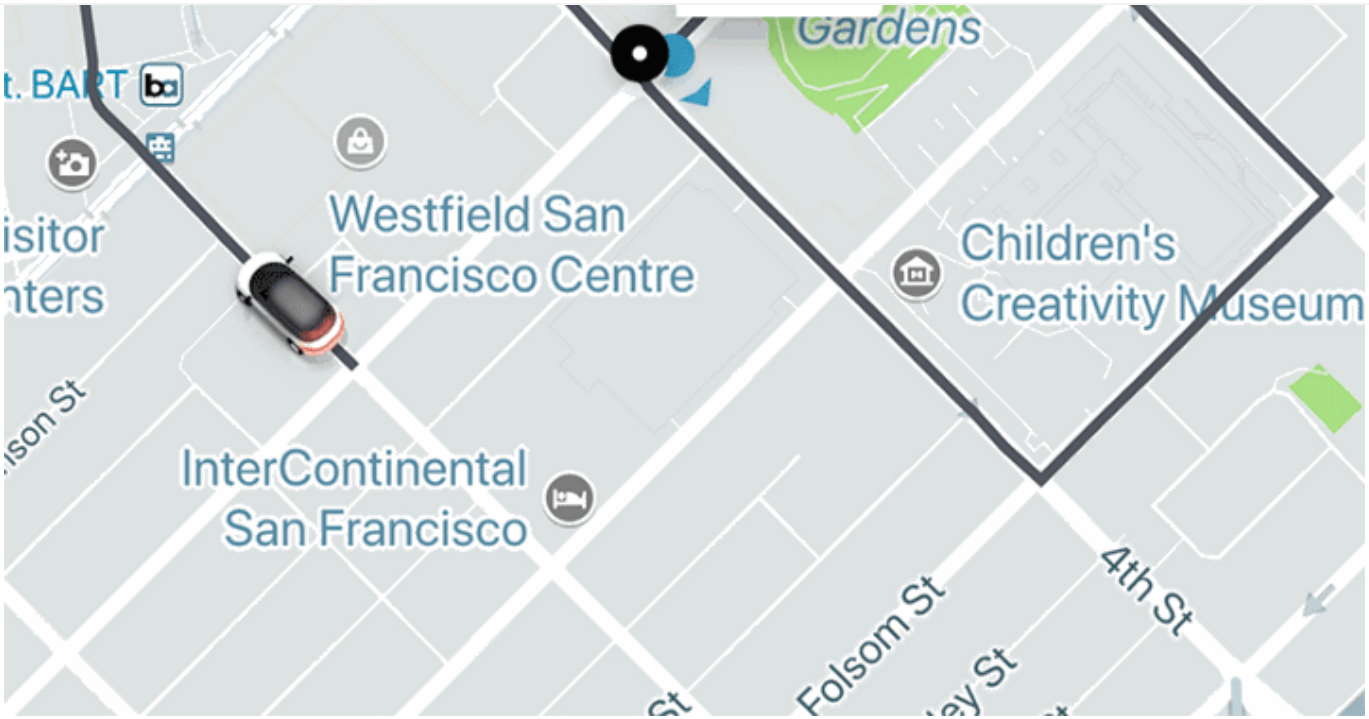


Also, S2 Library gives you the coverage of any given cell. Say for example we want a drawn circle on the map and we want to figure out all the suppliers available inside the circle. What we need to do is use the S2 library and give a radius to it. So it will automatically filter out all the cells which contribute to that particular circle. That way we know all the cell IDs. So now we can easily filter the data which we need and also which belongs to that cell. That way we had a list of supplies available in all the cells and we can calculate ETA.

When do we want to match Rider to Driver?

Draw a circle 2–3 km radius from the Rider's location. And list out all the drivers available using the S2 library. Now we need to check for **ETA**. Here we can use the shortest distance(**Euclidean Distance**). But this won't accurately give you the ETA. You can't just draw a line A to B directly. The path can be longer than we expected. You had to go with the connected roads.





Now we know the maximum distance for all filtered drivers. And the same time we can send a notification for the Drivers. If the driver accepts we can match a rider to the driver.

System Design

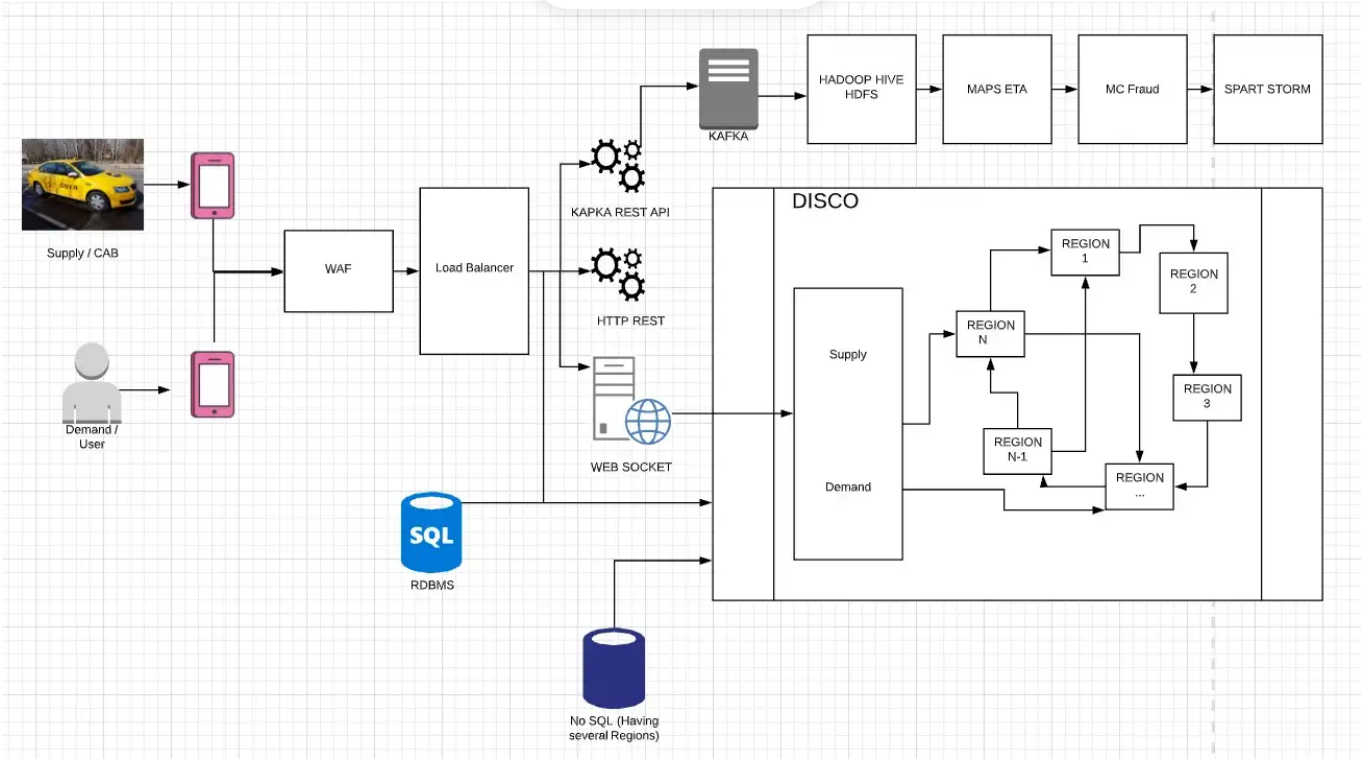


diagram 1

Above diagram, CAB is the **supply** which means the CABs and User is the **demand** where the User request the Driver. **Every 4-sec** once the Cabs will be

sending location data to the **KAFKA REST API**. Every call happens through the **Firewall**. Then it gets to the Load Balancer and it goes to **KAFKA** and it is going for different servers. And also a copy of location data sends to the **Database and also Dispatch Optimizatio**n to keep the latest location of the Cab.

Web Application Firewall (WAF) — Use for Security purposes. Here we can block the requests from the blocked IPs, Bots, and regions which is not supported by Uber.

Load Balancer — Here we can use different layers of Load balancers like Layer 3, Layer4, and Layer 7. Layer 3 works based on IP based Load Balancer(All the IPs for traffic go Layer3 Load balancer. In the Layer4 we can use DNS based Load Balancing. In Layer7 works based on Application-level Load Balancing)

KAFKA REST API — This will provide an endpoint to consume all the location data for every Cab. Example: We have 1000 Cabs running for a City and every 4 sec we are sending a location that means every 4 sec we have 1000 locations been sending for KAFKA REST API. Those locations will be sent to DISCO to keep the states alive.

WEB SOCKET — Unless normal HTTP requests web sockets are really helpful for these kinds of Applications. Because we need synchronize way to sending messages from Client to the Server and Server to the Client at any given point of the time. We should have a connection established between the Cab Application to the Server or The User to the Server. Web Socket keeps the connection opens for all of the Uber Application and based on the changes that happen in the DISCO or any component in the server the data will be exchanged between the Application and the Server. Mainly written in NodeJS(**Asynchornize and event-driven framework**).

DISCO Component

Dispatch System is mainly written in **NodeJS**. So that server can send/push the messages to the Application whenever it wants.

How do we scale these DISCO servers?

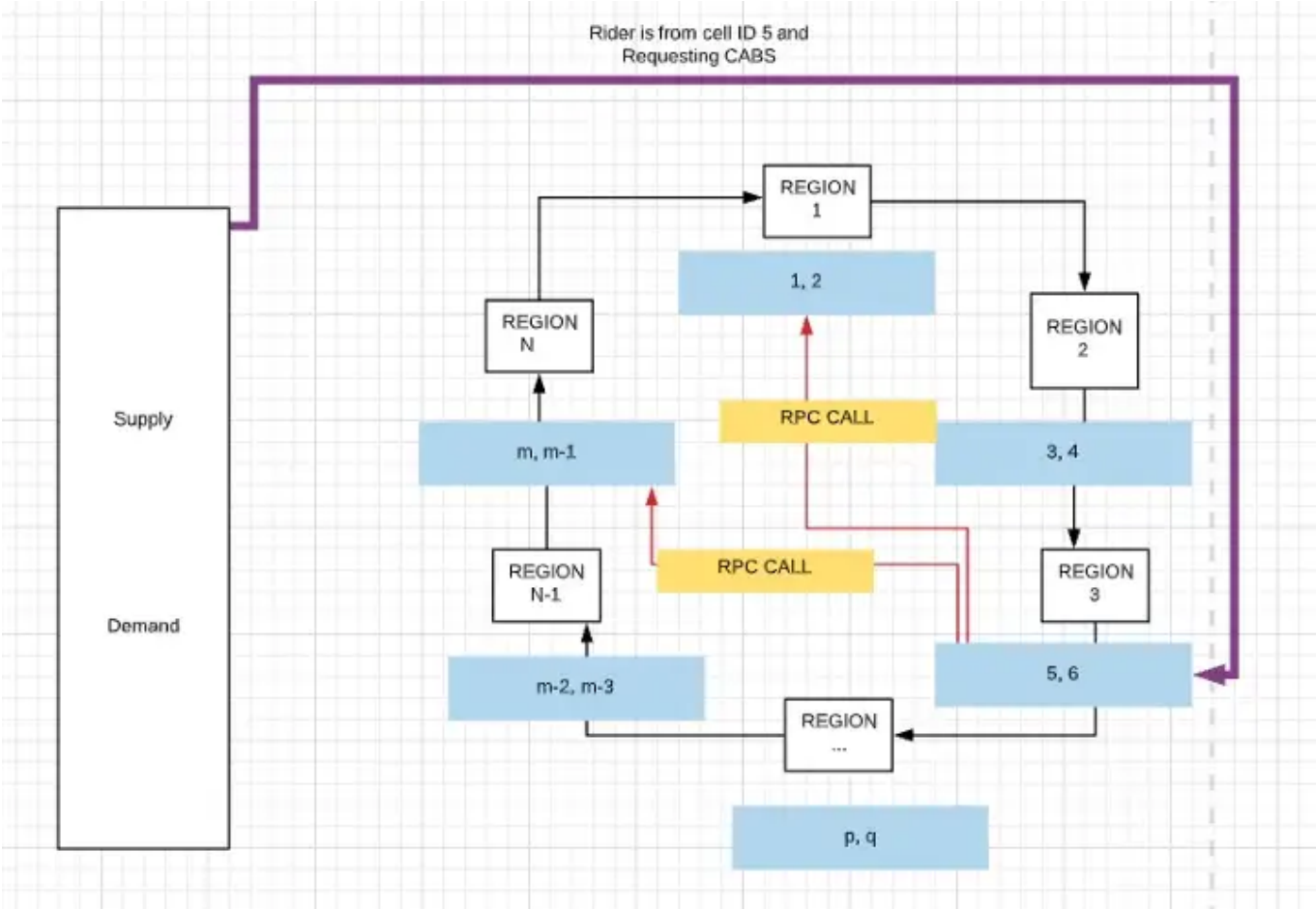
To scale this Uber used something called **REPOP**. It has 2 functionalities.

- **Consistent Hashing** — To distribute the work between the servers
- **RPC Call** — To make calls from the server to another server

It also uses a **SWIM protocol/Gossip Protocol** which helps servers to knows the other servers' responsibilities. Using this protocol w can easily add or remove one server in the ring. When we add a server the responsibilities are distributed to the newly added server and if we remove responsibilities add for other servers.

When the User request a Driver How it works?

When the user requests a ride then the request lands on the Web Socket. Web Socket hands over the request to the Demand Service. Then the Demand Service knows the requirement of CAB or Ride. Then the Demand Service request for Supply Service with the information of the ride(what type of ride, how many rides needed, what's the location). The supply service now knows the location(cell ID) of the User(Rider) and requests one of the servers on the server ring. In Consistent Hashing, we have equally distributed the responsibility. Like that all the location data equally distributed among the servers like the below image(blue color). The supply service is now trying to figure out the CABS which are near to the Rider from the servers by calculating the ETA values. And after calculating the ETA values supply server notify CABS through WEB SOCKET like "this Rider wants to go to this place could you please accept that?". If the Driver accepts that request trip assigned to that particular Rider and Driver.



Let's assume there are N servers and all the m number of location cells distributed among these servers(Here we assumed that 2 location data or each server). The rider is requesting from location cell "5". Then Server Region 3 is looking for available CABS and get the ETA value for available cabs and pass those data to the Supply Server. If there are not available CABS supply server gets the next location data which is near to the Rider using drawing a circle with a particular radius. Assume then we got location cell IDs as "2" and "m". Then server region 3 call region 1 and N which is RPC call to check for available CABS. After calculating the ETA value all the data sent to the Supply Server.

These are the steps that Uber follows to make a Uber request successful.

- Demand server asks supply server by location ID
- Supply server finds the cabs
- A request placed to cabs by order
- After a certain timeout, it tries the next cab
- After the matching user is notified

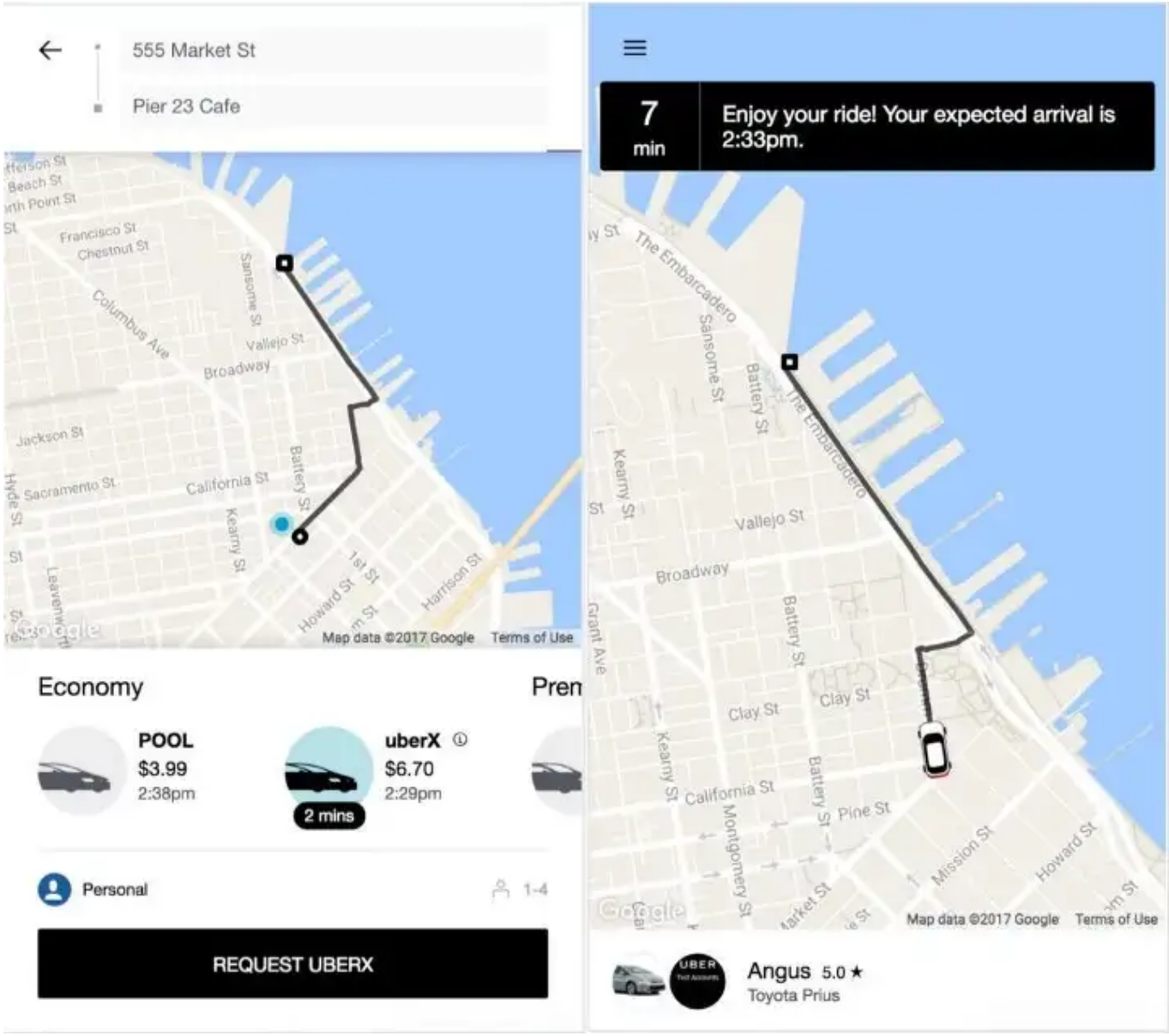
How DISCO handles more traffic?

Assume Uber adds a new city to the system and now How can we handle the traffic for the newly added city? We need to add new servers to **DISCO ring pop**. Now the newly added server's responsibilities are unknown. **MAPS ETA** component(diagram 1) knows all the newly added servers sell IDs and based on that newly added cells distributed mong newly added servers.

What happens when we remove a server from DISCO?

When we remove a server from the ring it reshuffles all the cell IDs and distributes them among existing servers.

What is Uber Geo-Spatial Design?



As you know how Uber uses the **Google S2 library** to breakdown the google maps into particular cells and that is used to identify the nearest CABS location to Riders location.

The user uses **Google Maps** to show maps. Google earlier used Map Box because of the Google pricing strategy. But now Uber is back to Google APIs and Maps. Now they are using the Google Maps framework to show maps on App, they are using it for calculating the ETA between 2 points and also they are using it for build paths and get the speed real-time location of the Cabs and Riders.

As you know when you are in a large building or are which Uber now allowed to access. Uber shows the **Preferred Access Points** like the entrance and the exit points. It learns repeatedly from Drivers Cabs used to stop near the entrance and exit gates. From that, it automatically shows to the customers they can only pick up from the entrance and the exit points. Uber uses several algorithms and Machine Learning to figure out the preferred access points.

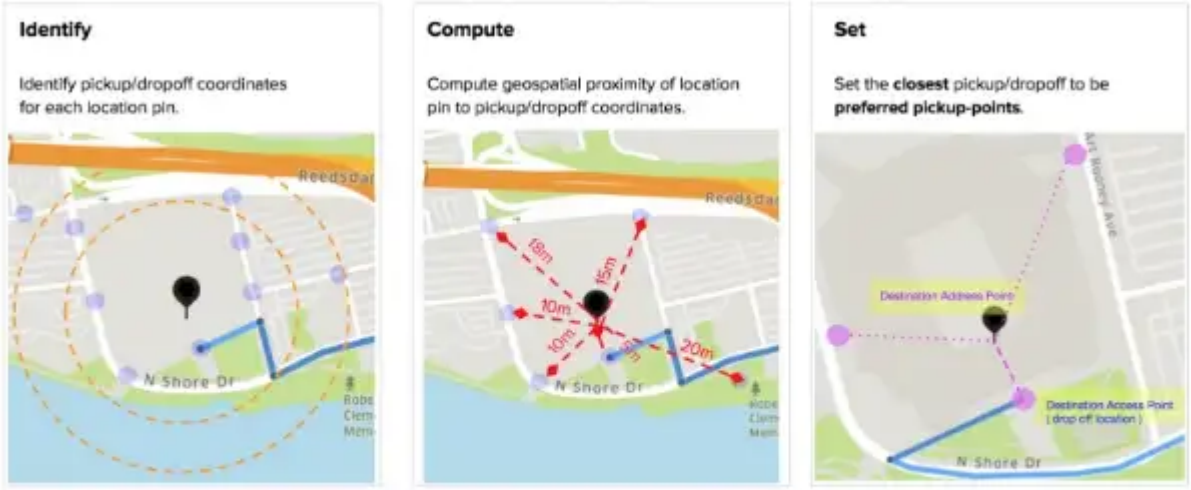


Figure 4: Refining access points is composed of three main steps: identify actual pick-up and drop-off locations used by drivers for a place or address (left), compute the distances from those locations to the place or address (middle), and then set the preferred access point based on the shortest distance (right).

How ETA(estimated time of arrival) is Calculated?

Before a trip starts, your app provides an ETA for when your driver should arrive at your pickup location.

After your trip starts, your app provides an ETA for when you should arrive at your destination.

Please note that ETA times are estimates and not guaranteed. A variety of external factors like heavy traffic or road construction can impact travel time.

Before you request a ride, your app displays a time in the black SET PICKUP LOCATION bar. This time estimates how long nearby drivers should take to arrive at your pickup location.

Using the slider at the bottom of your screen, you can view the ETA for each vehicle option available in your city.

After a trip starts, your app will continually update the ETA for your destination.

Read More :

Networks

https://marketplace.uber.com/matching Matching riders and drivers is a core component of what makes Uber so successful...

blogs.cornell.edu

What Databases Uber used to use?

Earlier they have used RDBMS to save profile-related data and GPS points and everything. But they identified they couldn't scale when they got more and more users as well as cities. Then they moved to NoSQL databases that

are built on top of MYSQL something called schemaless. They considered about

- Horizontally scalable — You can add multiple nodes in different regions and altogether acts as one database
- Write and the Read availability — every 4-sec cab will be sending the GPS location to the database. So that there are tons of reading and write happens to the system.
- No downtime — The system will be always available and what uber adds or remove from the system or while they are doing some maintenance for the system then the system should be up and there should be no downtime.
- Nearest datacenters — When they add new cities to the system they try to add new data centers or else they store data on the nearest datacenters to the newly added city to give the seamless service.

More details :

Why Uber Engineering Switched from Postgres to MySQL

The early architecture of Uber consisted of a monolithic backend application written in Python that used for data...

eng.uber.com

Analytics

Analytics is making sense of the data that we have. Uber needs to understand the customers, behaviors of the CAB drivers. That's the way to optimized Uber's system and cost of operations and also make customer satisfaction better. Uber uses different tools and frameworks for analytics. All the location data of Drivers and the data from Riders store in a **NoSQL** or **RDBMS** or **HDFS**. For some data analytics, you may need data from realtime. You can consume data from the **KAFKA**. HADOOP platform consists of a lot of analytics-related tolls which we can use for analytics. We can get dumb data from the NoSQL database using **HDFS.** And we can use Query tools like HIVE to get the data from HDFS.

You can consume the Historical data from the database and compare it with real-time data which can get from **KAFKA** and we can build new Maps that can improve the Map data which we have. And also from the real-time data, we can identify new traffic situations and drivers' speed, and a lot of things. That will helps to identify ETA values more accurately.

For **machine Learning and FRAUD detection**, Uber deal with multiple types of fraud, such as **payment fraud, incentive abuse, and compromised accounts.** Fraud has a direct impact on Uber as well as user experiences on the platform. To combat bad actors, Uber has a dedicated team of anti-fraud analysts, data scientists, and UX experts who work collaboratively on this

issue. As part of this effort, we build internal services that help us continually monitor and respond to changes in the ever-evolving fraud landscape.

These services look for errant behaviors, actions that would not have been taken by legitimate users. Using our fraud-fighting technologies, we can, for instance, differentiate between actual trips and those created by GPS spoofing, or analyze how our apps are being used to reveal fraudsters.

Payment fraud

Payment fraud happens when bad actors use stolen credit cards to pay for Uber trips. Typically, when credit card owners discover unauthorized transactions on their accounts, they call the bank or credit card company to dispute it, and Uber refunds the charge. In order to maximize the profit from stolen credit cards, fraudsters don't take these trips themselves. Instead, working as an agent service, they advertise discounted trip services on websites and chat forums to other people.

Incentive abuse

Uber frequently offers new users credit for signing up or referring friends, as well as bonuses for drivers who complete a certain amount of trips within a given time period. Fraudsters try to take advantage of these incentives by creating fake accounts to earn new user and referral credits or simulating fake trips to earn a driver bonus.

Compromised accounts

Fraudsters also use phishing techniques to access rider and driver accounts. With a rider account, a fraudster can offer agent services, selling rides to other people. Access to a driver account might let a fraudster withdraw money. Phishing techniques usually include emails, text messages, or phone calls to trick users into giving up their passwords and two-factor authentication codes.

More Details:

Advanced Technologies for Detecting and Preventing Fraud at Uber

we highlighted some presentations delivered during our second annual Uber Technology Day. In this article, engineering...

eng.uber.com

How to handle total datacenter failures?

Datacenter failure doesn't happen very often but Uber still maintains a backup data center to run the trip smoothly. This data center includes all the components but Uber never copies the existing data into the backup datacenter.

*Then how Uber tackles the **datacenter failure**??*

It actually uses **driver phones** as a source of trip data to tackle the problem of data center failure.
When The driver's phone app communicates with the **dispatch system** or the API call is happening between them, the dispatch system sends the encrypted **state digest** (to keep track of the latest information/data) to the driver's phone app. Every time this state digest will be received by the driver's phone app. In case of a data center failure, the backup data center (backup **DISCO**) doesn't know anything about the trip so it will ask for the

state digest from the driver's phone app and it will update itself with the state digest information received by the driver's phone app.

I think you have learned all the components in **Uber Architecture and System Design.** See you again in another tutorial.

Thank You!

Uber How Uber Works Disco Uber Architecture Uber System Design