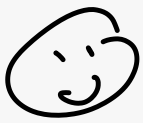
UBER SYSTEM DESIGN

OLA SYSTEM DESIGN

RAPIDO SYSTEM DESIGN

NammaYatri system Design



Functional Requirements:

1) Seeing Cabs
2) Estimated Time of Arrival between arrival and destination
3) Approximate Pricing
4) Booking Service
5) Location Service

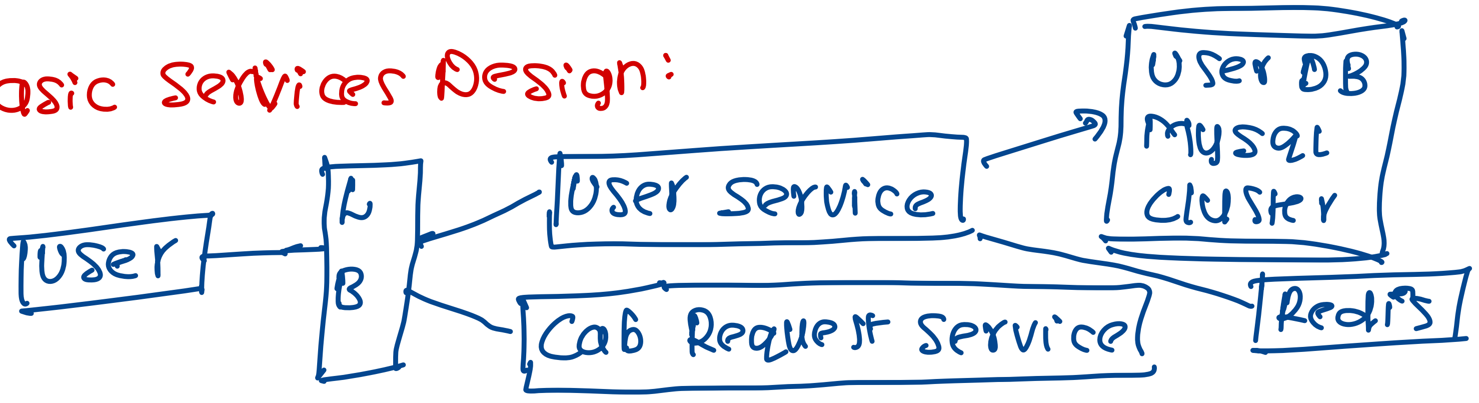
Non-Functional Requirements:

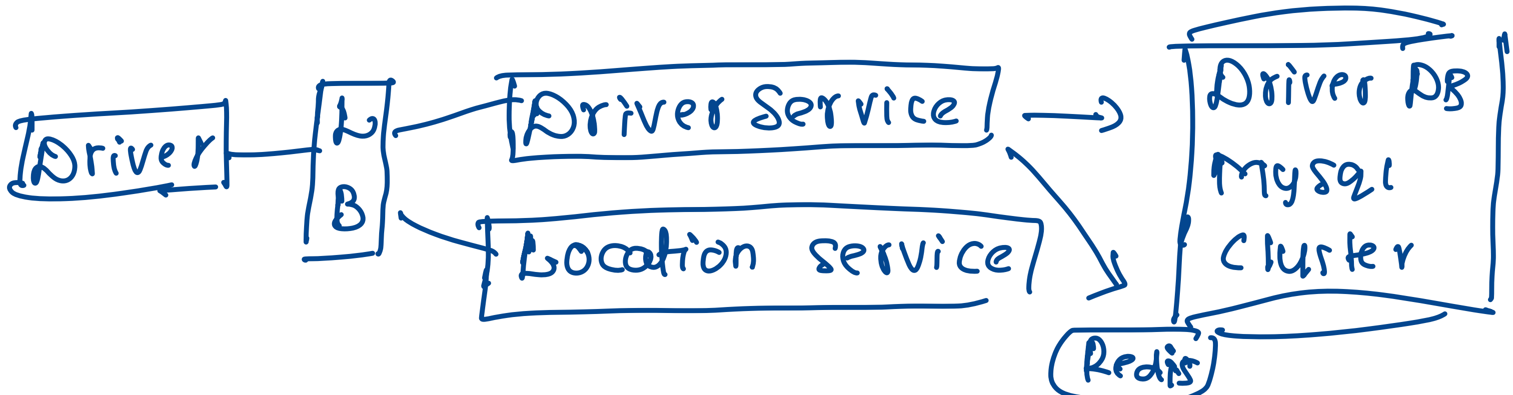
1) Global
2) Highly Available
3) Highly consistent

As per CAP Theorem, Only one of Availability and Consistency can be acheived but in our case certain services need to be consistent while other should be highly available

Architecture: Client - Server Architecture

Basic Services Design:





USER SERVICE:

User service helps the user to update his details and retrive.

Example: /post/User-Profile /get/UserDetails
/post/user_name ...

All this information is stored in Mysql Data cluster. Since most of the data is structured - Mysql is opted

Driver SERVICE:

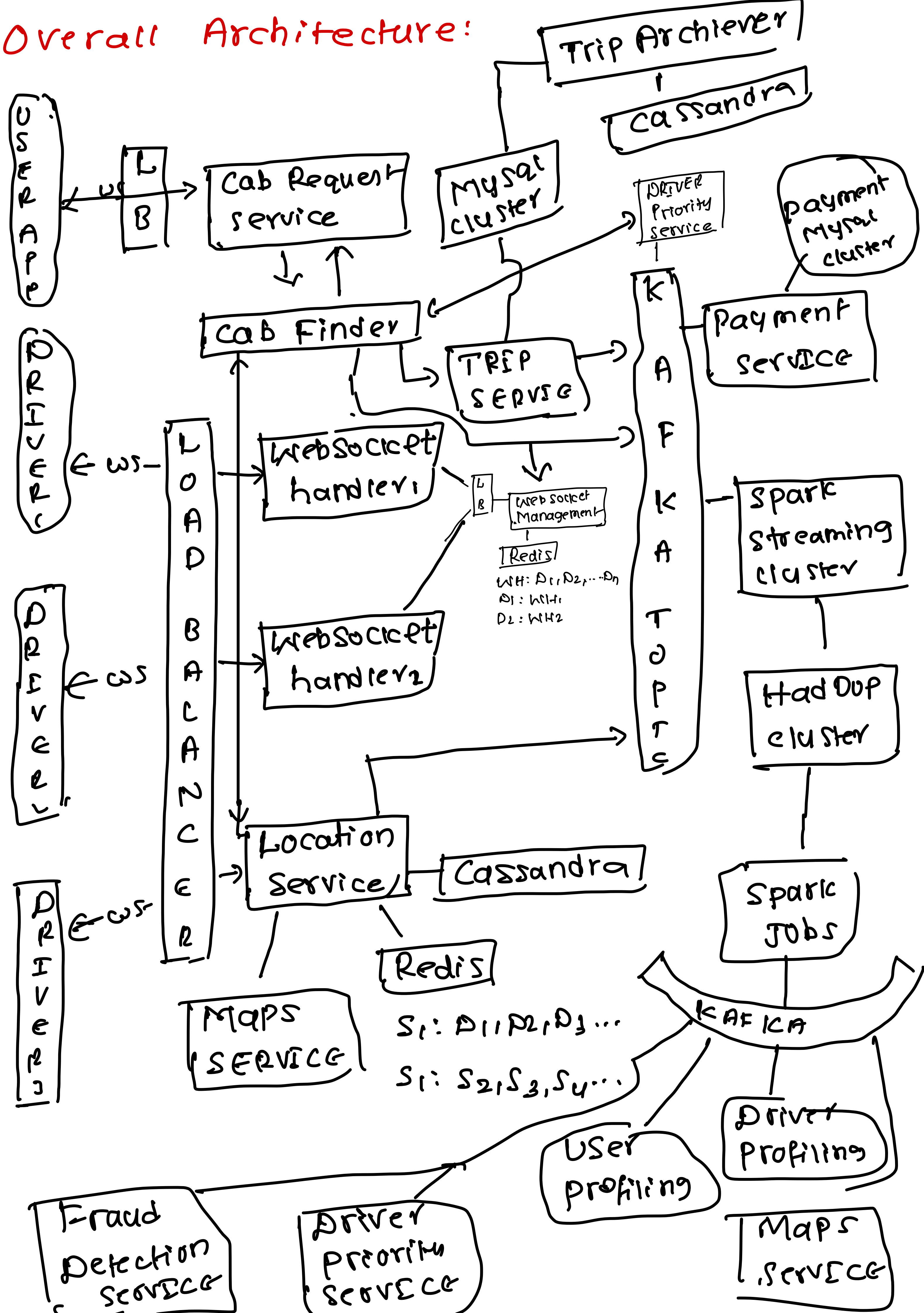
· Driver service mimics user service. It helps drivers' to update their details.

Redis: Redis is in-mem cache that helps in retriving data Fastly. Since low latency is required ... We opted for Read/Write through cache

Communication channel between the client and server:

Websocket can be the better option since it is consistent and bi-directional in communication. unlike polling & long polling.

Overall Architecture:



Describing services:

1) Cab Request Service:

cab request service maintains a websocket connection with user app.

It revert backs with either driver details or No cab found after interacting with cab finder service.

2) CAB FINDER SERVICE:

Cab finder Service plays a crucial role in Uber system Design.

Role 1: It takes the responsibility of responding back to users request with driver details or No driver found pop-up

Role 2: It shares lats & longs of the user with the location service to identify the segment to which the user belongs to and the cabs in that segment and neighbouring segment

Role 3: It triggers the trip service when a cab is booked with details like user-id, driver-id, ETA, Approx price, status etc..

Role 4: It listens to the driver priority service to identify the best possible driver for the user

Role 5: once the driver is alloted, It informs to the WebSocket Management to trigger event to the driver

3) WebSocket Handler:

WebSocket Handler establishes an active websocket connection with drivers

Role 1: When the Driver is online, It informs the driver presence to the WebSocket Management

Role 2: When WebSocket Management has a message to the driver, It informs the WebSocket Handler with the message

4) WebSocket Management Service:

WebSocket Management service is responsible for communicating messages with WebSocket Handlers.

Role 1: It communicates when an event is posted from Cab finder service

⇒ Identifies to which WebSocket Handler the driver is connected to and passes the message

Role 2: It keeps track of all websocket Handlers and the corresponding drivers list connected to each websocket Handler and vice-versa

WSH1: $D_1, D_2, D_3 \cdots D_n$

WSH2: $D_{n+1}, D_{n+2}, D_{n+3} \cdots D_m$

D_1 : WSH1 $\quad$ D_{n+2} : WSH2

D_2 : WSH1 $\quad$ ...

5) Location Service:

Location service is responsible for keeping track of locations of the drivers'.

Role 1: Drivers constantly communicate their location status with location service say for each 5 secs. Now, The location service updates the location of the drivers to cassandra.

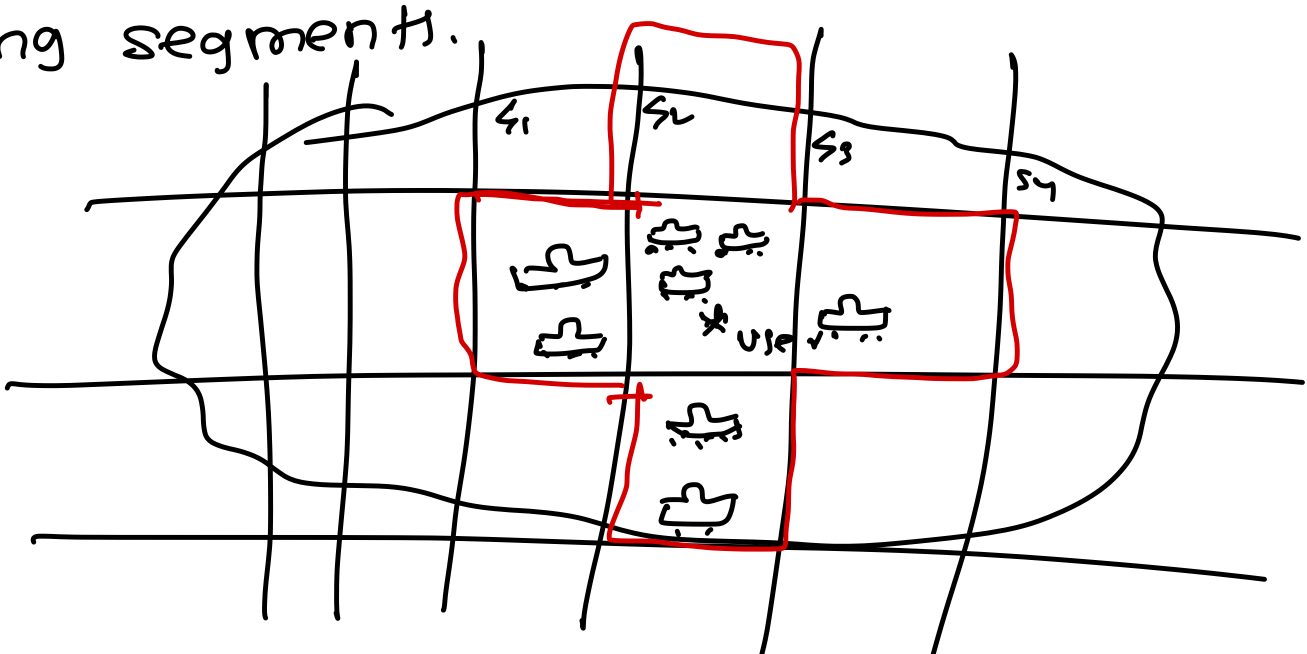
We need to do this to identify to which segment the driver belongs to also, to audit if the driver is following the map shared or not

Role 2: Location service gets the details pertaining to which segment the driver belongs to by communicating with Maps service. Also, the location of the user.

Role 3: It pushes segments, drivers info to the kafka topic so that Drivers priority service can use it in identifying the best driver.

6) Maps service:

Maps service is responsible for identifying the segment to which user belongs to and Drivers in that segment and in surronding segments.



Here, we divided the NYC into segments. Based on the lats & longs of the user ...

We identify to which segment he belongs to and drivers list is collected from the segment and neighbouring segments.

7) TRIP SERVICE:

Whenever a trip is booked in - the Cab finder service triggers Trip service.

Role: Trip service updates the MySQL DB cluster for that trip

8) Trip Archieval Service:

When trip is completed .- Trip A service will push the record in to cassandra for future audit purpose

9) Payment Service:

Once Trip is completed, the details pertaining to the payment record is pushed to payment cluster

10) Spark Streaming cluster:

This is a distributed data processing cluster. It receives various information like driver details & segment info from location service, No cab found message from cab finder, Trip details from Trip service. These details are processing in multiple mini channels and Stored in Hadoop cluster for analytics

(1) Spark Jobs) ML:

On the Hadoop cluster, we can perform multiple analytics to update user profiling like rating, Premium customer or not, regular or occasional etc...

Similar on Driver Data -- Best driver or not rating

Fraud Service: If same customer & driver are following for several times must be friends ha..

Map service : Traffic

Heat Map: We can identify segments with low cabs availability and can push notification to drivers using heat map or high demand segments etc...