There are two types of users in our system:

1. Drivers
2. Customers.

Functional Requirements

1. Drivers should be notified about their current location and their availability to pick up passengers.
2. The passengers when selecting a ride should be able to see all the available drivers in a given radius.
3. As and when the passengers make a request for the ride, the drivers within the radius must be notified about the request being made.
4. Once the drive and passenger confirm on the ride details, they should be able to monitor the location about where they are heading until the passenger gets off.
5. When the passenger reaches the destination, the driver should mark the journey complete and should make himself available for the next ride.

Non-Functional Requirements

- The system should be available in real time. We can't affrod any delay in latency. This might affect the performance of our system.
- The system should be highly consistent

Capacity and Estimation

Storage Estimation
Total users = 50 million
Total Numbers of Drivers = 3 million
Total Rides per day. = 10 million

Total Rides in Years = 10 * 10^6 * 30 * 12 = 3.6 billion

Avg. Size of 1 Ride Information = 200. kb
Avg. size of 1 User Information = 100 KB
Avg, size of 1 Driver information = 100KB

Total Storage for 1 Year = 50 * 50 * 10 ^ 6 + 3 * 10 ^6 * 100 + 3.6 * 10 ^ 9 * 200
= (2.5 + 6 + 72) = 80.5 GB per year
Total Storage for 5 years = 80.5 * 5 = 402.5 GB

Also, for live ride, we need to maintain information in Quadtree .These quadtree would be updated frequently to accommodate changing location of ride.

For each Place, if we cache only Location and Lat/Long, we would need 20GB to store all places.

24 * 600M => 12.4 GB

Since each grid can have a maximum of 500 places, and we have 600M locations, how many total grids we will have?

600M / 500 => 800K grids

Which means we will have 800K leaf nodes and they will be holding 12.4 GB of location data. A Quadtree with 800K leaf nodes will have approximately 1/3rd internal nodes, and each internal node will have 4 pointers (for its children). If each pointer is 8 bytes, then the memory we need to store all internal nodes would be:

800K * 1/3 * 4 * 8 = 10 MB

So, total memory required to hold the whole Quadtree would be 12.01GB

Basic System Desgin

Since all active drivers are reporting their locations every three seconds, we need to update our data structures to reflect that. If we have to update the Quadtree for every change in the driver’s position, it will take a lot of time and resources. To update a driver to its new location, we must find the right grid based on the driver’s previous location. If the new position does not belong to the current grid, we have to remove the driver from the current grid and move/reinsert the user to the correct grid. After this move, if the new grid reaches the maximum limit of drivers, we have to repartition it.

Although our Quadtree helps us find nearby drivers quickly, a fast update in the tree is not guaranteed .

For frequent and fast updates in quadtree, we will utilize caching sever to store driver's current location to propagate the information to nearby users. Similarly, user/drive current location will be stored in caching server to provide real-time update to both driver and users for rides.

Do we need to modify our Quadtree every time a driver reports their location?

No, we don't need to update our Quadtree .We keep the latest position reported by all drivers in a hash table and update our Quadtree less frequently.

Let's assume we guarantee that a driver's current location will be reflected in the Quadtree within 15 seconds. Meanwhile, we will maintain a hash table that will store the current location reported by drivers; let's call this DriverLocationHT.

Distribution of DriverLocationHT onto multiple servers

Although our memory and bandwidth requirements don't require this, since all this information can easily be stored on one server, but, for scalability, performance, and fault tolerance, we should distribute DriverLocationHT onto multiple servers. We will distribute based on the DriverID to make the distribution completely random. Let's call the machines holding DriverLocationHT the Driver Location server. Other than storing the driver's location, each of these servers will do two things:

1. As soon as the server receives an update for a driver's location, they will broadcast that information to all the interested customers.
2. The server needs to notify the respective Quadtree server to refresh the driver's location. As discussed above, this can happen every 10 seconds.

Efficient broadcasting of driver's location to customers

We will have a **Push Model** where the server will push the positions to all the relevant users. We will have a dedicated Notification Service that can broadcast the current location of drivers to all the interested customers.

We propose build our Notification service on a publisher/subscriber model. When a customer opens the Uber app on their cell phone, they query the server to find nearby drivers. On the server side, before returning the list of drivers to the customer, we will subscribe the customer for all the updates from those drivers. We can maintain a list of customers (subscribers) interested in knowing the location of a driver and whenever we have an update in DriverLocationHT for that driver, we will broadcast the current location of the driver to all subscribed customers. This way, our system makes sure that we always show the driver's current position to the customer.

Implement Notification service

We will use HTTP long polling or push notifications.

How about if clients pull information about nearby drivers from the server? Clients can send their current location, and the server will find all the nearby drivers from the Quadtree to return them to the client. Upon receiving this information, the client can update their screen to reflect the current positions of the drivers. Clients can query every five seconds to limit the number of round trips to the server. This solution looks simpler compared to the push model described above. However, such information needs to be cached for maximum performance.

Repartition a grid as soon as it reaches the maximum limit? We can have a cushion to let each grid grow a little bigger beyond the limit before we decide to partition it. Let's say our grids can grow/shrink an extra 10% before we partition/merge them. This should decrease the load for a grid partition or merge on high traffic grids.

Database Design

Driver

‘id’: unique user identifier (VARCHAR),
‘name’: first name, last initial, like ‘Matt J.’ (VARCHAR),
‘review count’: review count (INT),
‘useful votes’: count of useful votes across all reviews (INT),
‘email’: (VARCHAR),
‘password’: (VARCHAR)
Last login: DATETIME
Driver license VARCHAR
CAR MODEL VARCHAR
CAR LICENCE PLATE VARCHAR

Users

‘id’: unique user identifier (VARCHAR),
‘name’: first name, last initial, like ‘Matt J.’ (VARCHAR),
‘review count’: review count (INT),
‘useful votes’: count of useful votes across all reviews (INT),
‘email’: (VARCHAR),
‘password’: (VARCHAR)
Last login: DATETIME

Reviews

driver_id’: the identifier of the reviewed business (VARCHAR),
‘user_id’: the identifier of the authoring user (VARCHAR),
‘stars’: star rating, integer 1-5 (INT),
‘text’: review text (TEXT),
‘useful votes’: count of useful votes (INT),

User Rides

Column	**Type**
Ride id	Integer

User id	Integer
Driver id	Integer

Driver Rides

Column	**Type**
Ride id	Integer
User id	Integer
Driver id	Integer

Rides

Column	**Integer**
Ride id	Integer
Ride start time	Datetime
Ride end time	Datetime
Source longitude	Integer
Source latitude	integer
Destination. longitude	Integer
Destination latitude	integer

Keeping in above design, we purpose to utilized NoSQL for system for faster scability and larger storage for storing coordinates in quad server, we need Geospatial Database .

For this purpose, we propose to utilize Cassandra/Google Bigtable for our system. Cassandra will be used to store user/driver/ride information. Google Bigtable will be used for storing quad tree information and geospatial information related to ride

Fault Tolerance and Replication

What if a Driver Location server or Notification server dies? We would need replicas of these servers, so that if the primary dies the secondary can take control. Also, we can store this data in some persistent storage like SSDs that can provide fast IOs; this will ensure that if both primary and secondary servers die, we can recover the data from the persistent storage.

Cache

For providing live updates of ride ,we need to frequently update the location of quad tree. To achieve this, we need to store user/driver present location/quad tree/grid information in cache.
In cache ,user/driver id and grid/quad tree will be stored as key/value pair .

We propose to utilize Redis for Caching purpose. It acts as in-memory distributed cache for storing huge amount of information.