Real time location sharing system:

1. Design a feature that the riders can share the trip with the followers
2. 设计一个app,用户打车的时候可以分享一个链接到朋友圈，朋友打开链接可以看到该用户的实时位置。 其他时间都在聊天聊经历聊人生和bq。
3. 设计一个next stop bus 的app，能够支持查找离用户最近的5个车站各趟线路的ETA。先讨论了如何获得各个ETA的方案，然后在此基础上增加限制条件减少数据的写入，最后要求设计出一套完整的data path。
4. bq+设计分享uber即时旅途

Service:

Core in this system design?

Android system has Google Fused Location API (send either location data through cell tower data or WIFI signal and GPS), another function is for API live update.

Key points in this system design?

1. low battery/power consumption
2. set up listener with location services with a minimum distance (10 meters for example)
3. update location asap
4. use fuse listener for every 5 second as a heartbeat
5. send new locations to server only if new location is delta different
6. A socket can be set up between device sharing location and server, continuous push and device receiving location can do it via HTTP GET call

Background:

How to search a location?

Naïve search by SQL is hard, because range query search is very costly.

Google S2 or Geohash is preferred. Map the location to a long int.

Geohash: map a location (longitude, latitude) to a 32-based string,

Storage:

Database search, create a Geohash table:

1. Index on SQL database, like query. (slow and costly, not real-time, discarded)
2. NoSQL-Cassandra: column key is geohash table, range query (column key is indexed, cannot be changed frequently)
3. NoSQL-Redis/Memcached: drive location storage based on different levels of accuracy of locations. (Memcached not persisted). The reason why choose Redis is because this app involves a lot of quick write to DB, which Cassandra is not good at.
4. Everyone in this “position sharing group” has frequently update its position and check with 3 layers of storage and see if they need to change.

Scale:

How to support this service among different cities, countries?

DB sharing: based on city

How to avoid Redis breakdown?

Master-slave model. This can also reduce the network traffic. 3 copies of DB replica and they are stored on different machines, reducing single point failure. Write: master assign which machine to write, select a captain then write 3 copies to all 3 machines. Read, master choose one machine for reading, if not working, choose another one.

Database: a thread table, based on whether it is many-many or many-one or one-one. Use one-many for example, one rider updates his position, followers can be acknowledged. A thread table (thread id, participants id, owner id, created time, update time). Indexed by owner id and updated time. (SQL because NoSQL is not good at secondary index).

How to achieve real-time communication?

By socket, push service. Owner push his locations to the web server, once the socket is set up for the connection. Other users will poll the data from the web server, owner sends location to web server, web server send the information to push server. On the other side, other users, send a request to web server, web server sends a socket back to the participants, participants connect with push server by socket. Others get the notification from the push server.

Channel service, participants send request to web server, web servers tell which channels this user is in. When offer, push server will tell channel server he is off. Certain channel owner sends his location, channel finds current online user and sends the location out.

Think about how to shar the channel server and push server. It is in the slides but not very easy to understand.