主要考虑2个功能点，considerate 2 function:

1. 乘客司机上报实时地址 drivers and passengers upload real-time location
2. 平台给离乘客最近司机派单 system dispatch order to the driver closest to passenger

**思路 thinking**

1. 客户端间隔0.5秒发送含地理位置信息的心跳请求到服务端，服务端更新位置信息

App client send heart-beat request which contains location information to server, server update location information

1. 由于涉及到距离的计算，可以使用REDIS GEO，通过GEORADIUS获取附近的司机

Considerate use REDIS GEO command to calculate location, get drivers closest to the passengers

1. 考虑用REDIS存储用户定位信息，城市作为key，司机和乘客的用户id作为member

According to 2, use REDIS to store user location information, use city as key, drivers and passengers’ as member parameter

1. 由近及远的搜索，逐渐扩大GEORADIUS的最大距离参数

Expand REDIS GEORADIUS distance parameter to get drivers far from passengers

**架构图 architecture diagram**



1. 乘客下单后会立刻返回，APP进入等待轮询服务端返回订单司机信息 After passenger create order, client will get response soon, and start to loop for querying back end to get driver information
2. 乘客通过心跳请求上报地理位置 Passengers’APP upload location information through heart-beat request
3. 司机获取订单 drivers’APP query back end to get order
4. 司机通过心跳请求上报地理位置 Drivers’APP upload location information through heart-beat reques
5. Order system notice Event Engine system to calculate the driver closest to the passenger
6. Event Engine system won’t block request from Order system, it will figure out the driver and call Order system later
7. Event Engine system call Location system to calculate the driver closest to the passenger
8. Location system query REDIS and get result return to Event Engine system
9. Event Engine system call Order system and update order

**Key “must have” design goals for Dada’s system architecture**

1. Scalability

Dada hopes to scale to multiple cities, so we need to focus on scalability to make sure that our scale won’t be hard cause of system limit.

1. High Availability

If we want to serve more people include drivers and passengers, we need to improve system availability. If we can guarantee our system can work well during 7\*24, that we will be easy to win users’ favor, and make sure that our company won’t get loss cause of bug or system crash.

**How to implement those goals?**

1. We can Distributed deploy our system, so that when we face large requests, each instance’s pressure is small. If we make sure that our dependency is high available, for example, we can enable REDIS Sentinel mode, that can help us stay stable.