系统设计-设计Uber

2018-01-21 | 2018-12-17 | 系统设计

大纲

- RingPop
    - https://github.com/uber/ringpop-node
    - 一个分布式架构
    - 扩展阅读
      http://ubr.to/1S47b8g [Hard]
    - http://bit.ly/1Yg2dnd [Hard]
- TChannel
  https://github.com/uber/tchannel
- 一个高效的RPC协议
  RPC: Remote Procedure Call
- Google S2
  https://github.com/google/s2-geometry-library-java
  一个地理位置信息存储与查询的算法
- Riak
  Dynamo DB 的开源实现

1. Scenario 场景

需要设计哪些功能？设计到什么地步？

- 第一阶段
    - Driver report locations 司机报告自己的位置——heart beat模式
    - Rider request Uber, match a driver with rider 乘客叫车，匹配一辆车
- 第二阶段
    - Driver deny/accept a request 司机取消/启动 接单
    - Driver cancel a match request 司机取消订单
    - Ride cancel a request 乘客取消请求

- Driver pick up a ride/ start a trip 司机接人
    - Driver drop off a rider/end a trip 司机送到人
- 第三阶段
    - Uber Pool
    - Uber Eat

QPS / Stroage

猜一猜：

2011年的QPS?——大约1k.

2015年的QPS?

- 假设20w司机同时在线：
    - Driver QPS = 200k/4 = 50k，每次汇报200k个请求，每4秒汇报一次；【这个占大头】
    - Peek Driver QPS = 50k x 3 = 150k
    - Rider QPS可以忽略：不用随时汇报位置，一定远小于Driver QPS

2. Service 服务

Uber主要干的事情就两件：

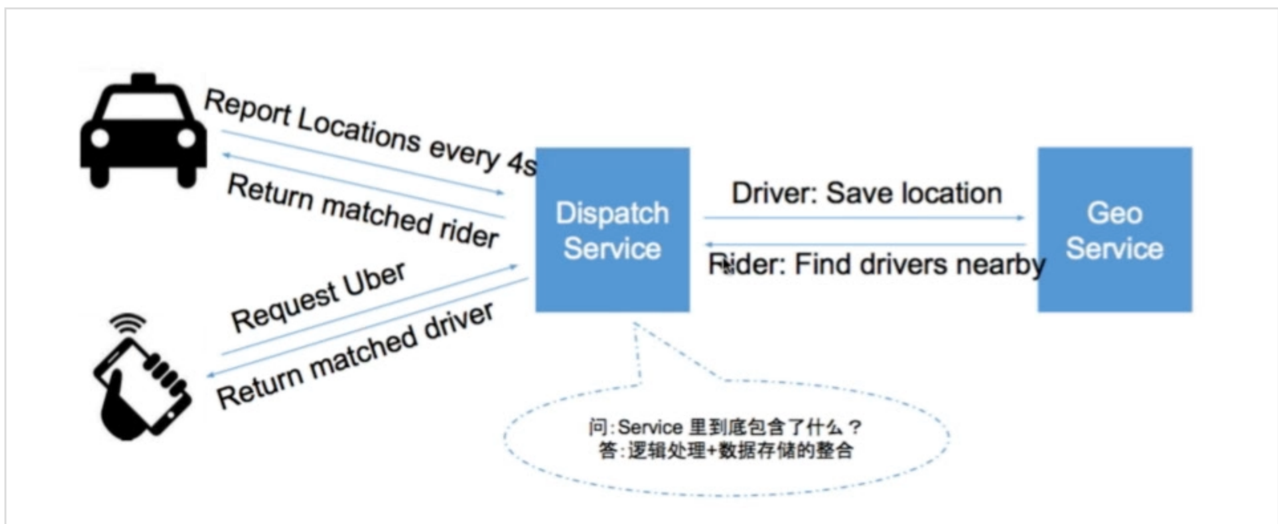
- 记录车的位置：GeoService
- 匹配打车请求：DispatchService

大概如下图：

问题：这张图漏掉了什么？—— 司机怎么知道乘客在哪？如果直接让车从GeoService拿用户定位？——这样不太好，GeoService负担太重了

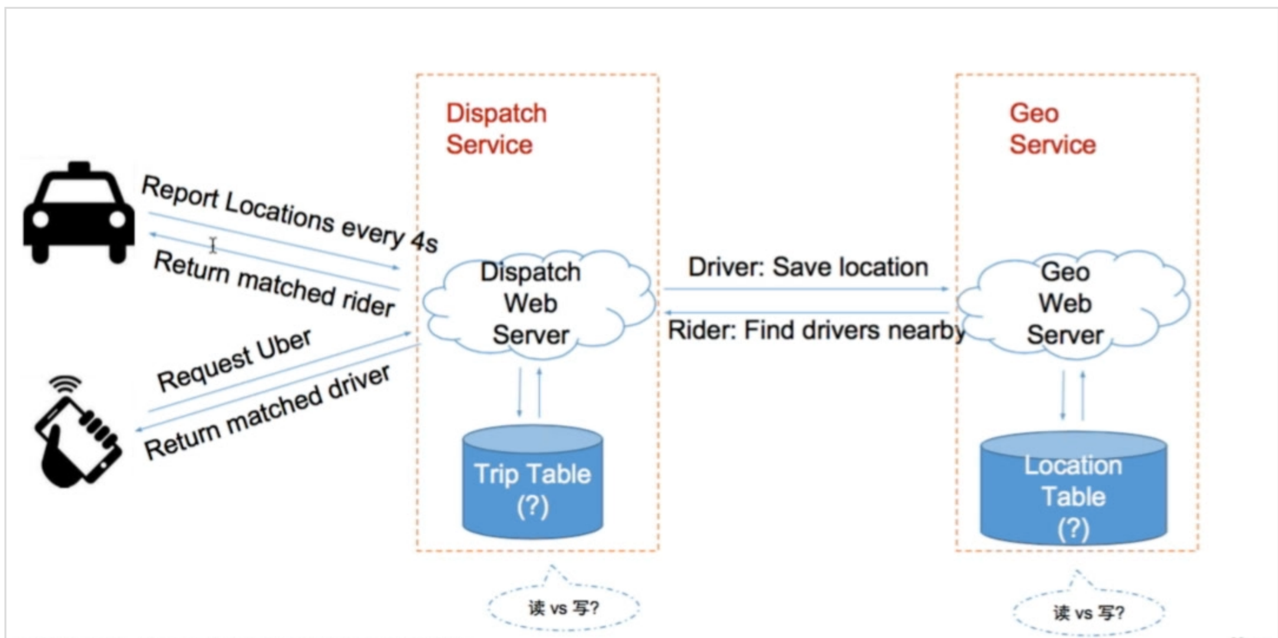
修改一下框架：

Driver如何获得打车请求？—— Report location的同时，服务器顺便返回匹配上的打车请求



- 司机向dispatch service发送位置信息，同时返回匹配的乘客信息
- 乘客向dispatch service发送打车请求，同时返回请求

具体地来看，就是：



问题：

Location Table——读多还是写多？——写多

Trip Table—— 读多还是写多？——读多（司机每4秒requst时会去读一下有没有匹配的）

存储——细化表单

```
1  class Trip {
2      public Integer tripId;
3      public Integer driverId, riderId;
4      public Double Latitude, Longitude;
5      public Integer status;
6      public Datetime createdAt;
7  }
8
9  class Location {
```

```
10        public Integer driverId;
11        public Double Latitude, Longitude;
12        public Datetime updatedAt;
13    }
```

从数据库角度来看，就是：

Trip Table	type	comments
id	pk	primary key
rider_id	fk	User id
driver_id	fk	User id
lat	float	Latitude 纬度
lng	float	Longitude 经度
status	int	New request / waiting for driver / on the way to pick up / in trip / cancelled / ended
created_at	timestamp	

Location Table	type	comments
driver_id	fk	Primary key
lat	fk	纬度
lng	fk	经度
updated_at	timestamp	存最后更新的时间，这样知道司机是不是掉线了

问题——如何通过Location Table查询某个乘客周围5公里以内的司机？

SQL:

```
1  SELECT * FROM Location WHERE lat < myLat + delta
2                           AND   lat > myLat - delta
3                           AND   lng < myLng + delta
4                           AND   lng > myLng - delta
```

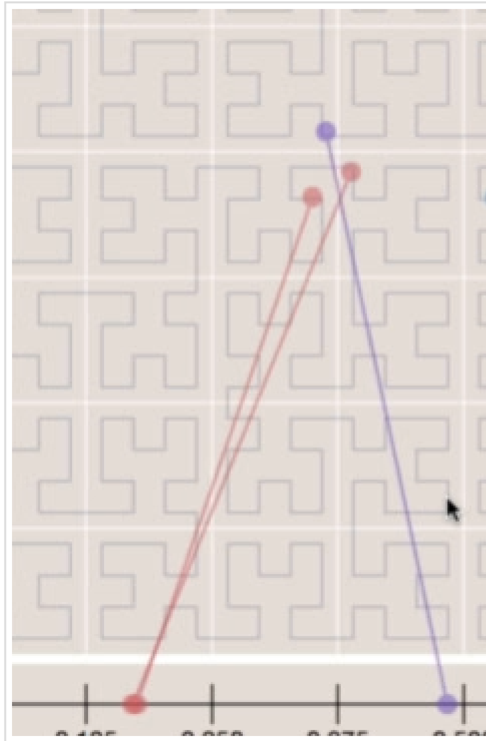
这个也太复杂了，基本上等同于扫描了一遍所有的数据。

优化方法

Google S2

- 基本思想：把二维空间变成一维空间，将地址空间映射到2^64的整数。
- 方式：希尔伯特曲线：曲线一笔能勾勒遍这个图，然后图上的每个点编个号，这个编号就是所需要的整数。
- 特性：**如果空间上比较接近的两个点，对应的整数也比较接近**
- Example: `(-30.043800, -51.140220) → 10743750136202470315`

但并不是100%保证两个点如果比较近，整数就近，比如：



- 两个红点在一维和二维上都上比较近
- 但红点和紫点虽然在二维上近，但其实在一维上不太近（属于误差）

Read more: http://bit.ly/1WgMpSJ

Hilbert Curve: http://bit.ly/1V16HRa

Geohash

- Peano Curve：将经纬度转化为字符串！
- Base32：0-9, a-z 去掉 (a,i,l,o)——为什么用 base32？因为刚好 25 可以用 5 位二进制表示
- 核心思路二分法
- 特性：**公共前缀越长，两个点越接近**
- Example: `(-30.043800, -51.140220) → 6feth68y4tb0`

Read more: http://bit.ly/1S0Qzeo

主要思想如下所示：

- 把地图分成格子；
- 每个格子由0-9, a-z 去掉 (a,i,l,o)的32个字符组成
- 在一个盒子内部，再划分32份；然后继续分分分
- 如果我们把linkedIn和google经纬度分别弄成这种方式，我们发现前缀有4个一样；然后facebook远一些。
- 那么到底有多远呢？主要参考表：如果两个公共前缀有geohash length个，那么就是对应的

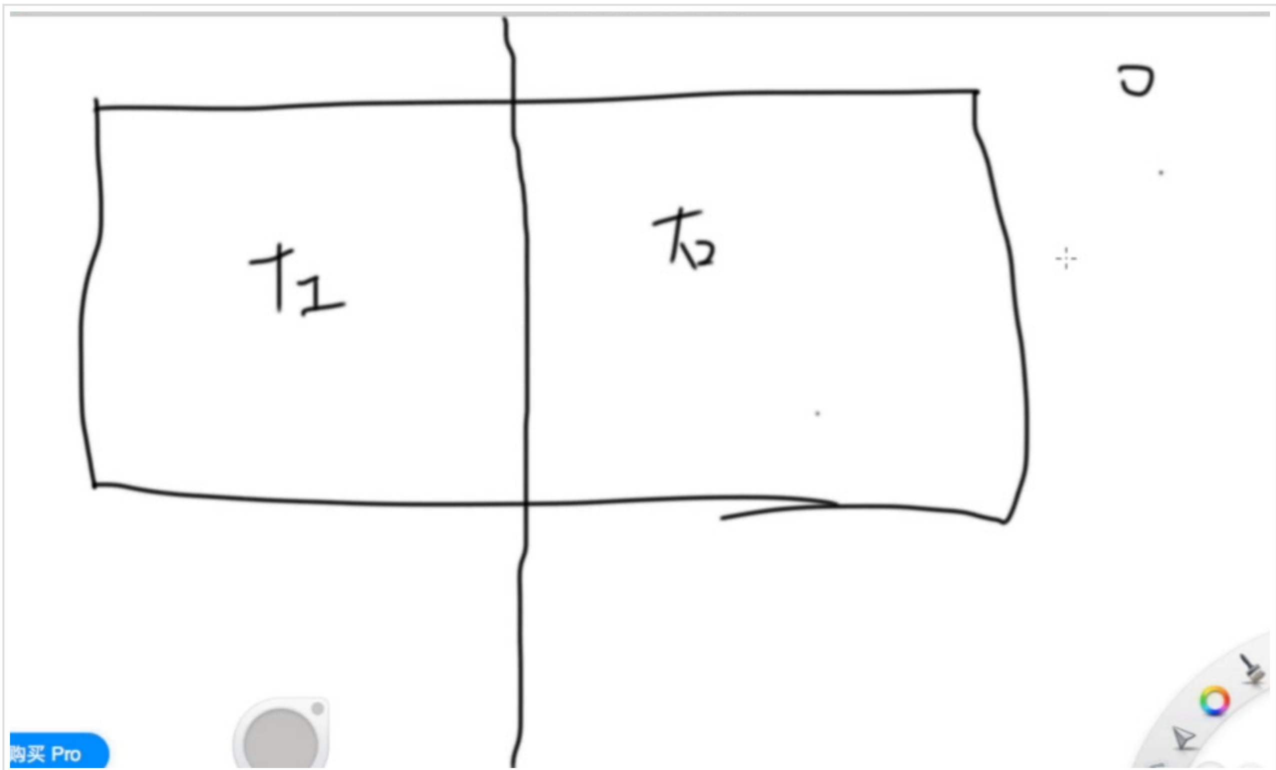
- Examples:
- LinkedIn HQ: 9q9hu3hhsjxx
- Google HQ: 9q9hvu7wbq2s
- Facebook HQ: 9q9j45zvr0se

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	±630
3	7	8	± 0.70	± 0.7	±78
4	10	10	± 0.087	± 0.18	±20
5	12	13	± 0.022	± 0.022	±2.4
6	15	15	± 0.0027	± 0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

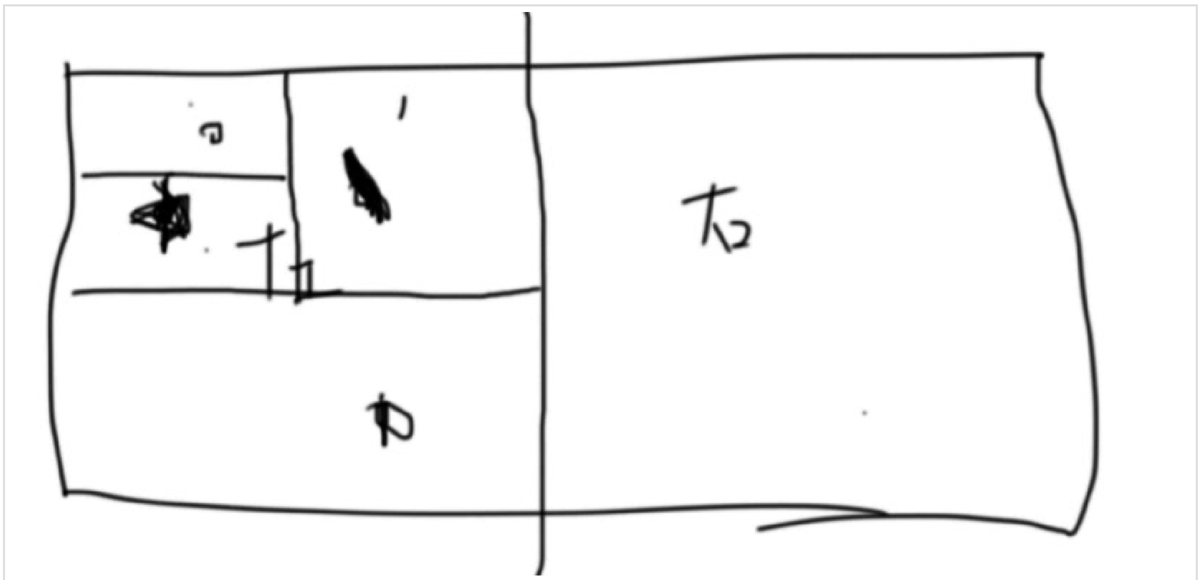
Geohash的二分优化

事实上，它还有优化：

- 先把地球分成两份，如果这个点在左边，就记为0，右边记为1：



- 再横向切一刀，如果在上边就记为0，下边记为1



- 继续切切切，如果切5次，就是一个5位的二进制数字；这个5位的二进制数字可以表达为1位的刚才32位的字符，比如11100表示为W
- 总的来说，其实就是比如将经度(-180,180)切一次，然后纬度切切切，如下所示：

- 北海公园：lat=39.928167, lng=116.389550
- 二分(-180,180) 逼近经度，左半部记0，右半部记1
  - (-180, 180)里116.389550在右半部 → 1
  - (0, 180)里116.389550在右半部 → 1
  - (90, 180)里116.389550在左半部 → 0
  - (90, 135)里116.389550在右半部 → 1
  - (112.5, 135)里116.389550在左半部 → 0
- 二分(-90,90) 逼近纬度，下半部记0，上半部记1
  - (-90,90) 里 39.928167 在上半部 → 1
  - (0,90) 里 39.928167 在下半部 → 0
  - (0,45) 里 39.928167 在上半部 → 1
  - (22.5,45) 里 39.928167 在上半部 → 1
  - (33.75,45) 里 39.928167 在上半部 → 1
  - … (还可以继续二分求获得更多的精度)

课后作业：
http://www.lintcode.com/problem/geohash/

先经后纬，经纬交替

1110011101

W X

查询半径2公里内的车辆

- 根据下表，看出来大概geohash length即前缀5位相同的，差不多在2km内
- 而Google的位置为9q9hvu7wbq2s
- 找到位置以9qqhv开头的车辆即可！！！！——问题：在数据库怎么实现？

查询Google半径2公里内的车辆

- 找到精度误差 > 2公里的最长长度

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	±630
3	7	8	± 0.70	± 0.7	±78
4	10	10	± 0.087	± 0.18	±20
5	12	13	± 0.022	± 0.022	±2.4
6	15	15	± 0.0027	± 0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

- Google HQ: 9q9hvu7wbq2s
- 找到位置以9q9hv以开头的所有车辆

怎样在数据库中实现该功能？

3. Stroage 数据

如何找到位置开头以9q9hv开头的车辆？数据库怎么存？

- SQL 数据库
  - 首先需要对 geohash 建索引

    ```
    CREATE INDEX on geohash;
    ```

  - 使用 Like Query:

    ```
    SELECT * FROM location WHERE geohash LIKE "9q9hv%";
    ```

- NoSQL - Cassandra
  - 将 geohash 设为 column key
  - 使用 `range query (9q9hv0, 9q9hvz)`
- NoSQL - Redis / Memcached
  - Driver 的位置分级存储（小技巧）
    - 如 Driver 的位置如果是 9q9hvt，则存储在 9q9hvt， 9q9hv， 9q9h 这 3 个 key 中，而这三个类似于不同的经度；检索时，就依次减小经度进行查询看看有没有车
    - 6位 geohash 的精度已经在一公里以内，对于 Uber 这类应用足够了
    - 4位 geohash 的精度在20公里以上了，再大就没意义了，你不会打20公里以外的车
  - key = 9q9hvt, value = set of drivers in this location

不同数据库比较

能够熟悉每种数据存储结构的特性，对于面试十分加分！

- SQL 可以，但相对较慢
  - 原因1：Like query 很慢，应该尽量避免；即便有index，也很慢

- 原因2：Uber 的应用中，Driver 需要实时 Update 自己的地理位置
  被index的column并不适合经常被修改
  B+树不停变动，效率低
- NoSQL – Cassandra 可以，但相对较慢
  - 原因： Driver 的地理位置信息更新频次很高
    Column Key 是有 index 的，被 index 的 column 不适合经常被“修改”
- NoSQL – Memcached 并不合适
  - 原因1：Memcached 没有持久化存储，一旦挂了，数据就丢失
  - 原因2：Memcached 并不原生支持 set 结构
    需要读出整个 set，添加一个新元素，然后再把整个set 赋回去

因此这个题，可以考虑用Redis

Redis

- 数据可持久化
- 原生支持list，set等结构
- 读写速度接近内存访问速度 >100k QPS

用Redis怎么做这个呢?

用户打车角度

用户发出打车请求，查询给定位置周围的司机

- (lat,lng) → geohash → [driver1, driver2, …]
  - 先查6位的 geohash 找0.6公里以内的
  - 如果没有 再查5位的 geohash 找2.4公里以内的
  - 如果没有 再查4位的 geohash 找20公里以内的
- 那么上述的过程的数据库长这样：

Location Table	
key	geohash
value	{driver1_id, driver2_id, driver3_id …}

匹配司机成功后

匹配司机成功，用户查询司机所在位置

- driver1 → (lat, lng)

表如下所示：

Driver Table	
key	driver_id
value	(lat, lng, status, updated_at, trip_id)

司机角度

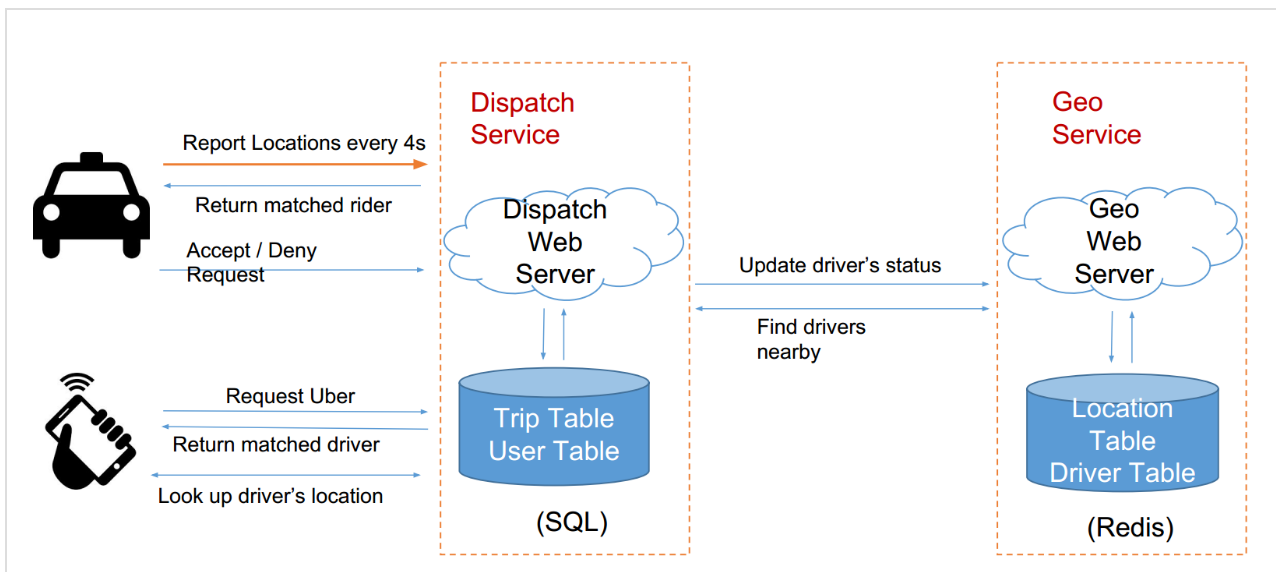
- 司机汇报自己的位置
  - 计算当前位置 lat, lng的geohash
    - geohash4, geohash5, geohash6
  - 查询自己原来所在的位置
    - geohash4',geohash5', geohash6'
  - 在Driver Table中更新自己的最后活跃时间

对比是否发生变化
并将变化的部分在 Redis 中进行修改

- 司机接受打车请求
  - 修改 Trip 状态
    - 用户发出请求时就已经在 Trip Table 中创建一次旅程，并Match上最近的司机
  - 在Driver Table 中标记自己的状态进入不可用状态
- 司机完成接送，结束一次Trip
  - 在 Trip Table 中修改旅程状态
  - 在Driver Table 中标记自己的状态进入可用状态

4. 可行解总结

1. 乘客发出打车请求，服务器创建一次Trip
   1. 将 trip_id 返回给用户
   2. 乘客每隔几秒询问一次服务器是否匹配成功
2. 服务器找到匹配的司机，写入Trip，状态为等待司机回应
   1. 同时修改 Driver Table 中的司机状态为不可用，并存入对应的 trip_id
3. 司机汇报自己的位置
   1. 顺便在 Driver Table 中发现有分配给自己的 trip_id
   2. 去 Trip Table 查询对应的 Trip，返回给司机
4. 司机接受打车请求
   1. 修改 Driver Table, Trip 中的状态信息
   2. 乘客发现自己匹配成功，获得司机信息
5. 司机拒绝打车请求
   1. 修改 Driver Table，Trip 中的状态信息，标记该司机已经拒绝了该trip
   2. 重新匹配一个司机，重复第2步



5. Scale 拓展

看看有哪些问题没有解决，需要优化；出现故障怎么办

有什么隐患?

需求是150k QPS。Redis 的读写效率 > 100 QPS。那么是不是1-2台就可以了?

万一Redis挂了就gg了，分分钟损失几百万!

解决方式——**DB Sharding**

- 目的1：分摊流量
- 目的2：防止单点失败 Avoid Single Point Failure

按照城市Sharding

- 难点1：如何定义城市?
- 难点2：如何根据位置信息知道用户在哪个城市? ——用多边形代表城市的范围，问题本质变为：求一个点是否在多边形内，属于计算几何问题。
  城市数目：400个
- 万一乘客在两个城市边界怎么办?
  - 找到乘客周围的2-3个城市
  - 这些城市不能隔太远以至于车太远
  - 汇总多个城市的查询结果
  - 这种情况下司机的记录在存哪个城市关系不大

如何判断一个乘客是否在机场内?

同样可以用Geo Fence。类似机场这样的区域有上万个，直接O(N)查询太慢

分为两级Fence查询，先找到城市，再在城市中查询Airport Fence

Read More: http://ubr.to/20qK4F4

如何减小一个db挂了的损失?

- 方法1：Replica by Redis —— Master-Slave

- 方法2：Replica by yourself
  - 底层存储的接口将每份数据写3分
  - sharding key 从 123(city_id) 扩展为：
    - 123-0
    - 123-1
    - 123-2
  - 读取的时候，从任意一份replica读取数据；读不到的时候，就从其它replica读
  - 三份replica极有可能存在三个不同机器上，同时挂掉的概率很小
    当然也有可能不巧存在一个机器上，这个问题如何解决请参考Dynamo DB的论文
- 方法3：让更强大的NoSQL帮你处理——Riak / Cassandra
  既然一定需要用多台机器了，那么每台的流量也就没有150k QPS这么高了
  用 Riak / Cassandra 等NoSQL数据库，能够帮助你更好的处理 Replica 以及机器挂掉之后恢复的问题

6. 总结

- 分析出 Uber 是一个写密集的应用
  - 与大部分应用都是读密集不一样
- 解决 LBS 类应用的核心问题 – Geohash / Google S2
  - 如何存储司机的地理位置
  - 如何查询乘客周围的车
- 分析出一个 Work Solution，说明整个打车的流程

WEAK

- 分析出按照城市进行 Sharding
- 知道如何做 Geo Fence 的查询
- 通过分析选择合适的数据库
- 考虑到单点失效（多机）与数据故障（备份）如何解决

HIRED

- 深入理解 NoSQL DB 的实现以及了解 Ring Pop的实现 *
  - 只能靠大家自己读论文了 http://bit.ly/1mDs0Yh
- 设计 Uber Pool / Uber Eat *

HIRED

禁止录像与传播录像，否则将追究法律责任和经济赔偿

7. Bit tigger

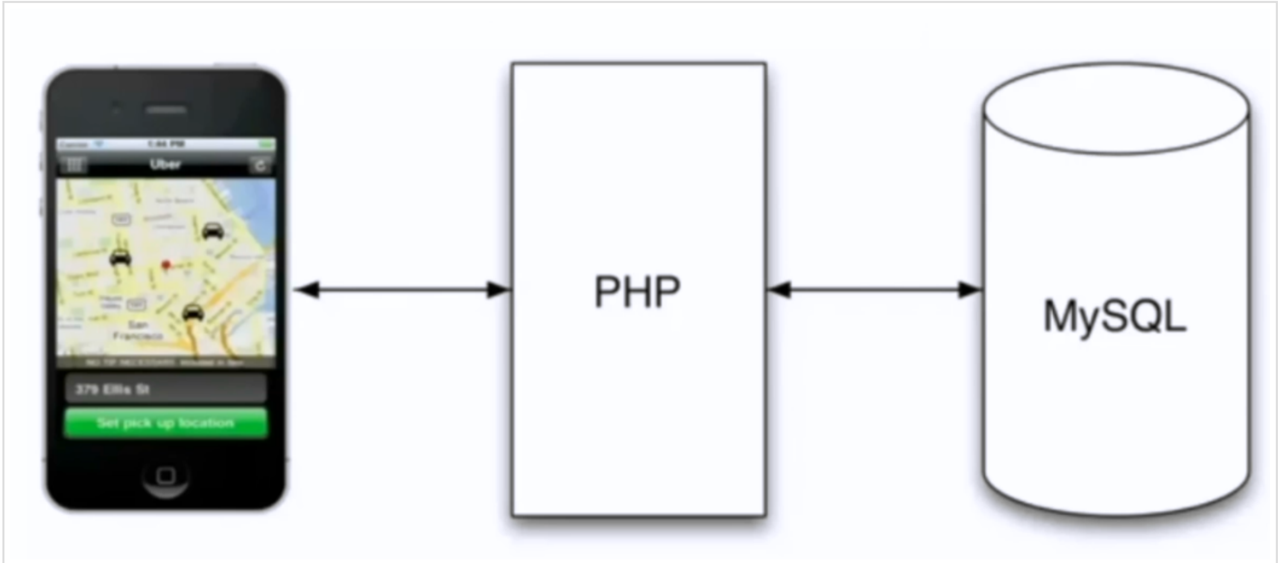
大纲

- Uber 从 0 到 1
- Uber 从 1 到 1万
- S2： 地理信息库
- Ringpop

什么是Uber?

一键打车

很久以前的Uber



问题：QPS(每秒请求)是多少呢?

分析问题：请求有哪些?

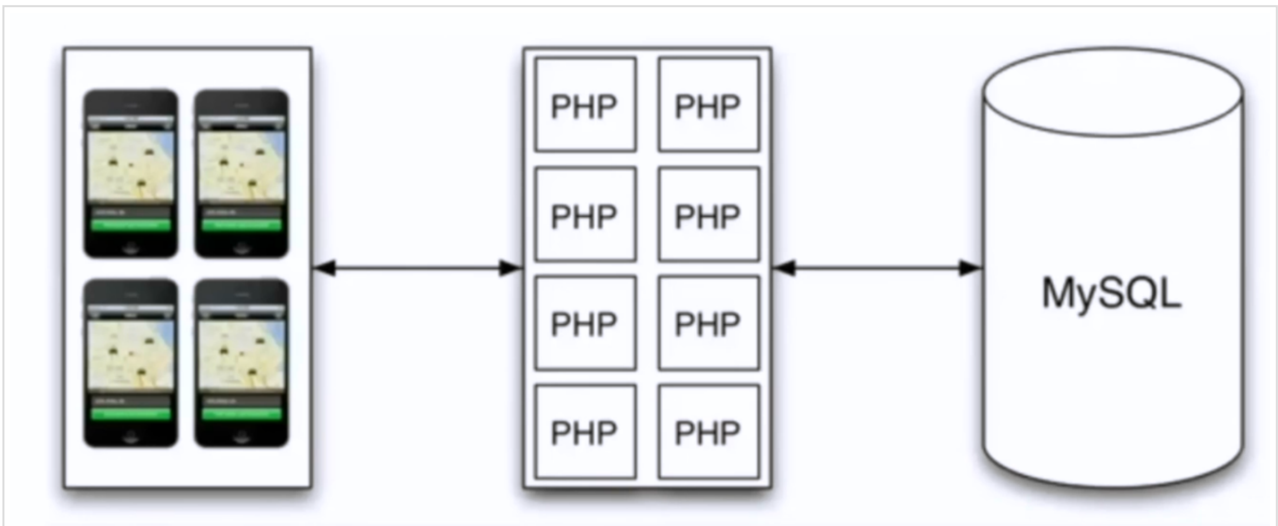
1. driver的写服务 : 100个车， 4秒一次 = 25
2. user的读服务 : 会读一个范围内的

问题：为什么要纠结于QPS?

QPS直接决定了后台服务器的选择!

人太多怎么办?

暴力横向扩展PHP



能不能扩展MySQL?

如果将MySQL分成好几个服务器呢？—— 读的时候可能会向各个服务器请求，然后再merge回来。很麻烦哇。

知识点:

服务器的负载均衡有两种：

1. 连接的负载均衡——可以缓存下来需要写入的请求

其它挑战

1. 一个司机能不能带两个乘客？（这个架构不能解决）
2. 有没有可能两个司机一个车？
3. 一个账号两个车呢？

8. 设计

第一个解决的问题应该是什么？

分发问题！

数据库设计：

1. driver messge
2. passager message
3. driver - passager 的当前状态： OD 、出发时间等

数据库设计步骤：

1. 找到所有实体
2. 设计实体之间的关系

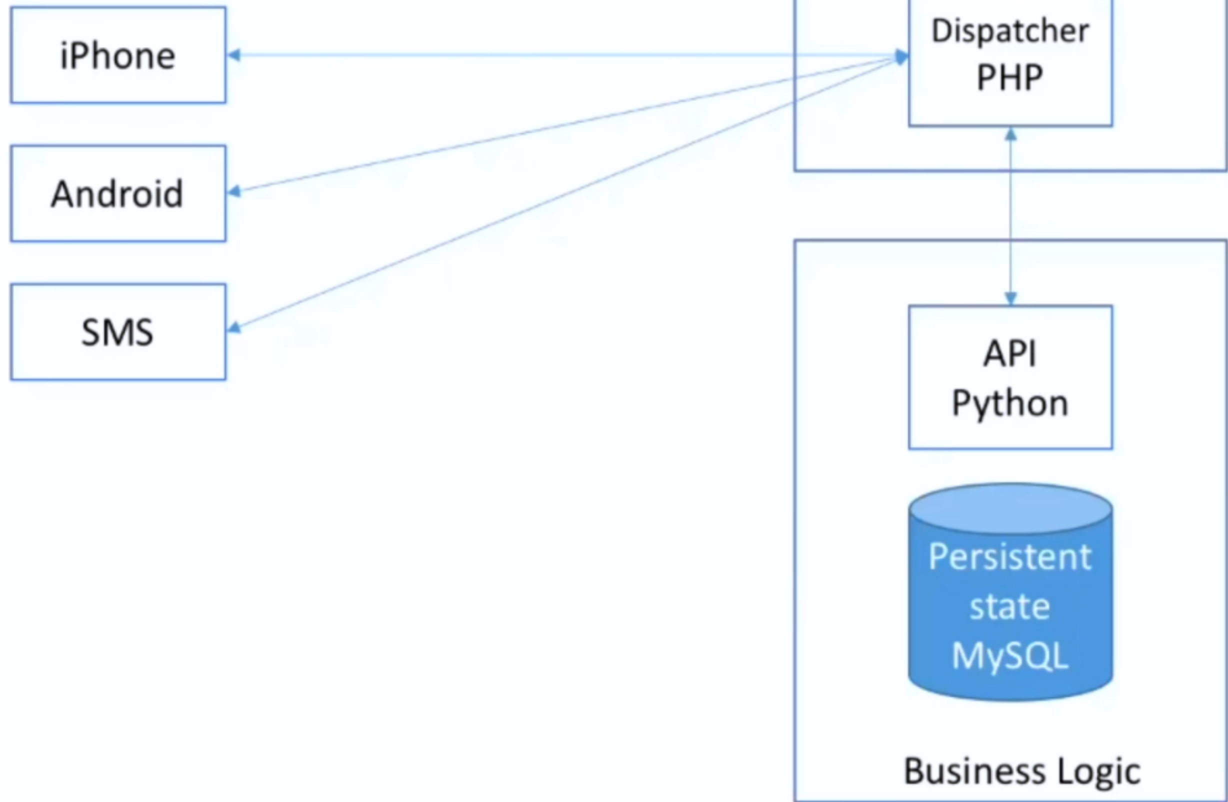
为什么用MySQL？

1. 一般来说NoSQL与SQL性能都差不多
2. MySQL相对更简单些

还缺个啥？

持久化的支付平台！（Business Logic）

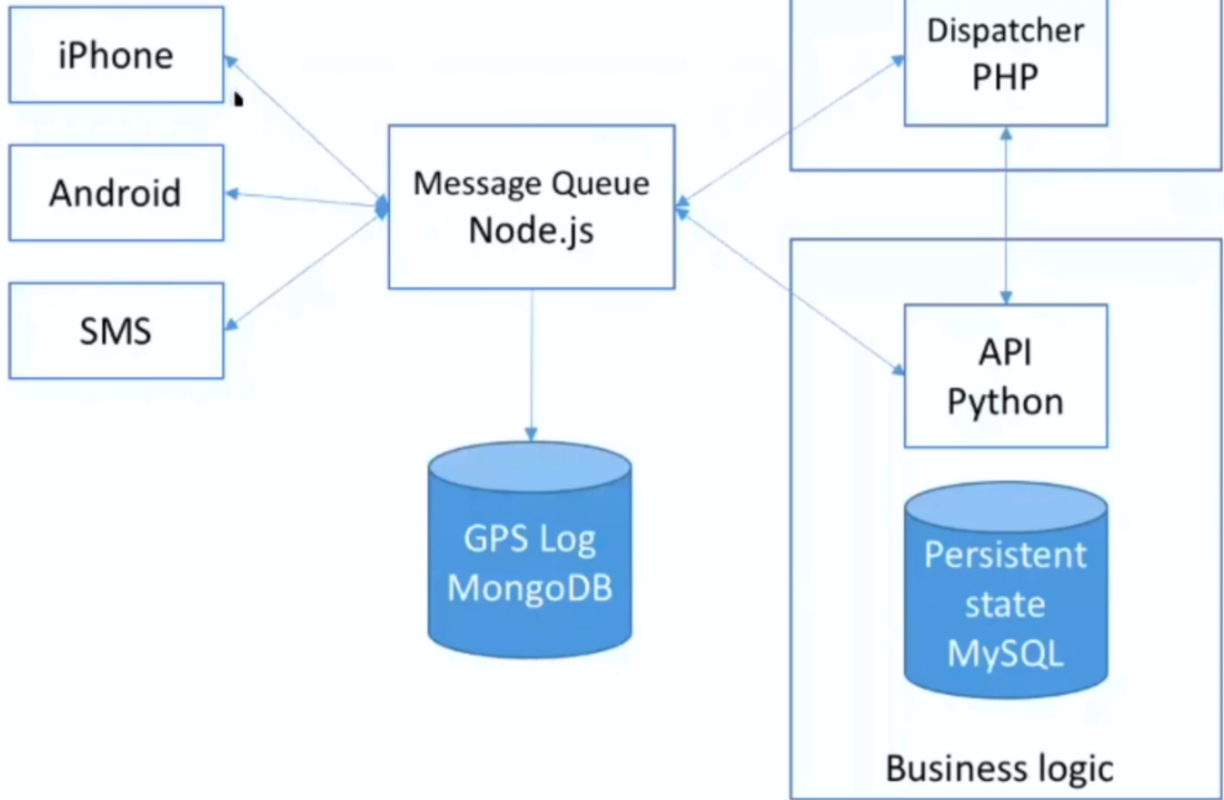
为啥用python？因为后台处理数据更方便



升级

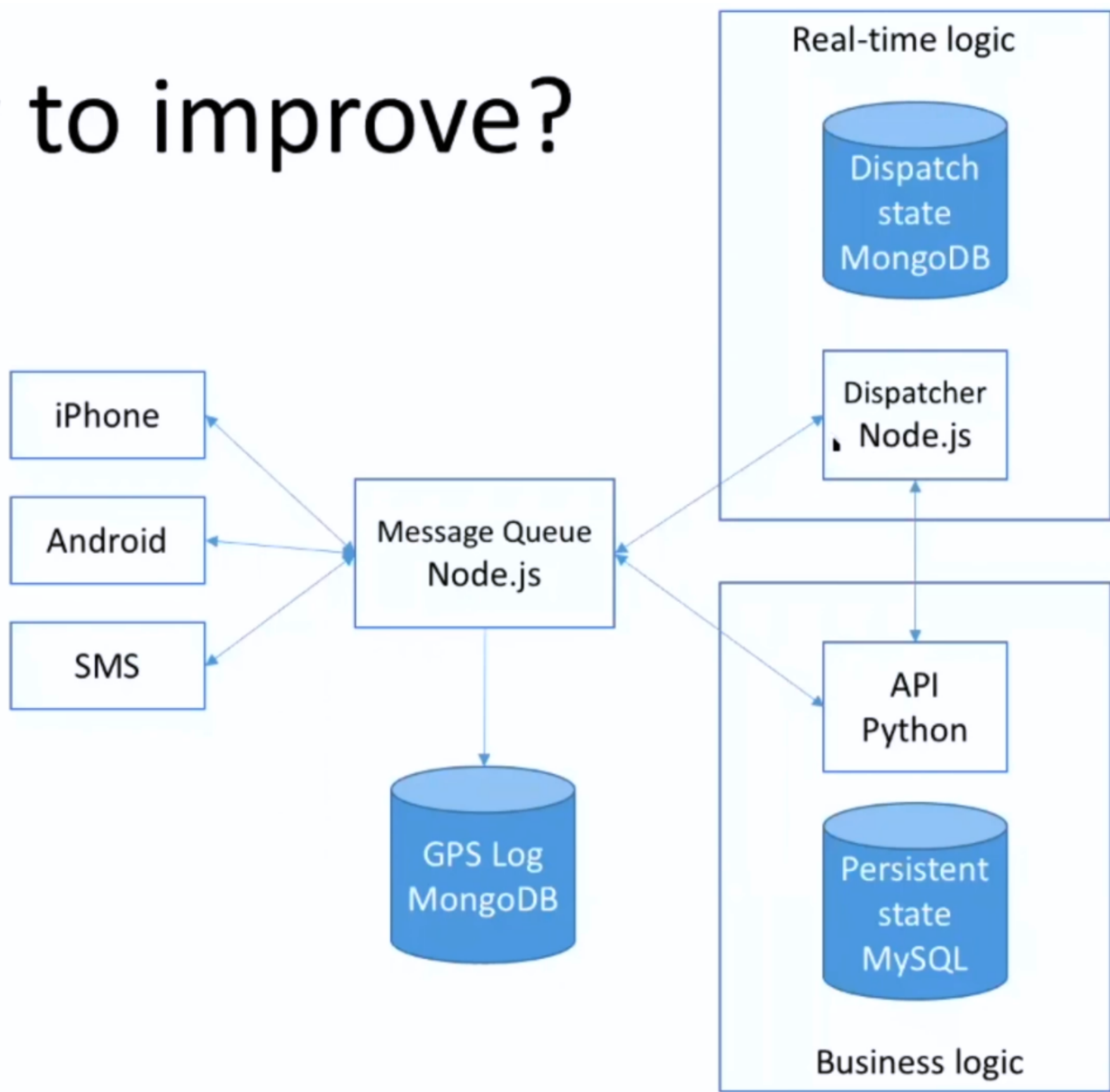
当请求越来越多的时候，就需要一个消息队列

最大的请求：司机地理位置的更新！



mongoDB是文档型的，对接这种log是非常好的

再升级？

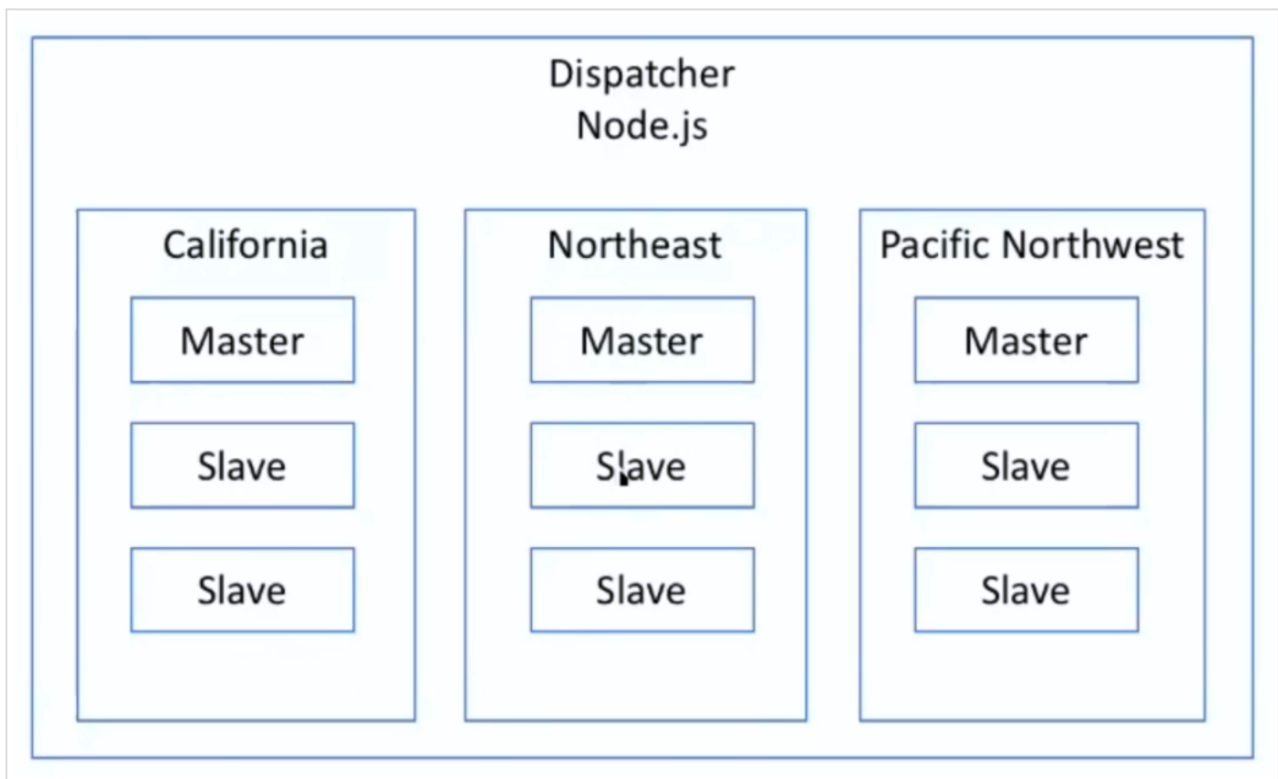


js - 服务逻辑

py - 数据逻辑

如何避免单点失败?

master - slave架构



1. 热备份：当master挂掉后用slave
2. 如果资源有限，可以对外开放slave的读的权限

各区信息需要同步吗？

1. 一般不需要
2. 但是！！！！用户数据！！（万一有的人跑去国外呢）

这样分区，天然缺点是什么？

万一有用户刚好在分割线上。。。。

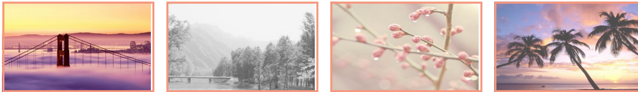
‹ 算法-线段树

算法-位操作 ›

登录

欢迎参与讨论

还没有评论，快来抢沙发吧！



热评话题

leetcode刷题总结 | jiayi797

机器学习实践-O2O优惠券预测-对第一名的思路源码分析（二） | jiayi797

算法-卡兰特数 | jiayi797

python实现某网站自动打卡 | jiayi797

ms实习-指数平滑 | jiayi797

BUPT_ACM_2017_F_Simple_recursion | jiayi797

搜索引擎-ES引擎架构和原理 | jiayi797

© 2018 ♥ jiayi797

感谢hexo.Next | 博客全站共309.4k字

访问人次 11439 | 总访问量 30350 次