2018 年 12 月

哈爾濱工業大學

大数据计算基础

题　　目：52:路线骑行时间估计

专　　业：计算机科学与技术

学　　号：1160300621

姓　　名：黄友勤

课程类别：必修

1. 问题描述

题号：52

题目名称：骑行路线时间估计

a. 问题的背景/意义

根据自行车的轨迹数据，实时地估计全城任意路线的骑行时间，可以有效地推荐骑行路线。

b. 问题的技术难点

(1)轨迹数据的稀疏性：在过去的一段时间里，很多道路上并没有轨迹数据。
(2)不同轨迹的组合问题:对于有数据的路段，有很多种子轨迹组合的方式来估计时间,寻找最优解很困难。
(3)效率、准确性和扩展性的权衡:城市范围很大，轨迹快速变化，子轨迹组合方式很多，但时间估计的实时性要求很强。

c. 数据来源

https://www.microsoft.com/en-us/research/publication/detecting-urban-black-holes-based-on-human-mobility-data/

d. 设计要求

利用所提供的自行车轨迹数据，设计(1)骑行时间预测算法；(2)基于自行车轨迹的城市路线骑行时间评估系统。

e. 实验要求

算法要求至少与 2 个相关工作进行对比；系统要求采用分布式实时计算系统 Storm，实现基于 Storm 的城市路线骑行时间评估系统。

2. 基于 Storm 的系统

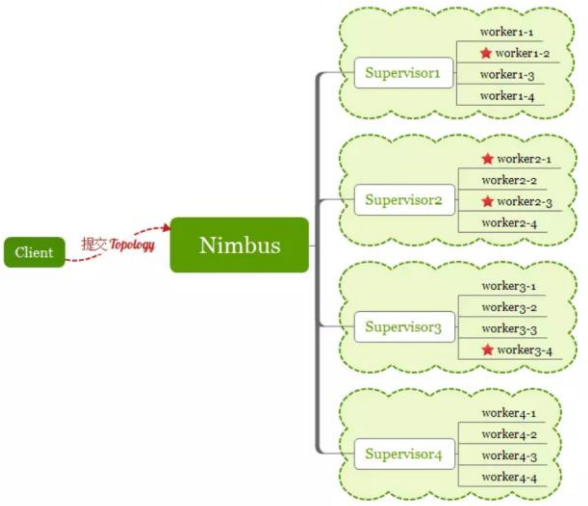
介绍你使用的主要的系统或者算法库

a. 系统架构

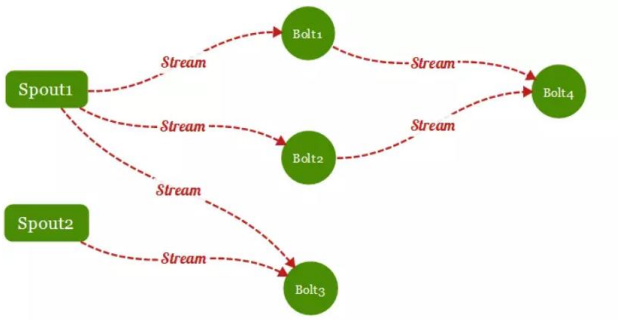
Storm 是一个分布式计算框架，在 被 Twitter 取得后开源。它使用用户定义的 spout 和 bolt 来定义信息源和操作，以处理批量、分布式的流数据。

Storm 集群主要包括：
Nimbus：主节点，是一个调度中心，负责分发任务。

Supervisor：从节点，负责任务的执行
Worker：任务工作进程，一个 Supervisor 中可以有多个 Worker。
Executor：Worker 进程在执行任务时，会启动多个 Executor 线程。



Storm 的任务拓扑结构：
Topology：任务的抽象概念。由于 storm 是流式计算的框架，它的数据流和拓扑图很像，所以它的任务就叫 topology。
Spout：从数据源获取数据并进行分发。
Bolt：得到 Spout 或者上一个 Bolt 的数据,然后进行处理后交给下一个 Bolt 处理。
Tuple：在 storm 中，一条数据可以理解为是一个 Tuple。



b. 你为什么选择使用这个系统

Storm 系统适合处理流式数据。即当数据流通过 spout 进入系统以后，bolt 会将数据进行处理，处理结束后，**没有等待的时间**，立刻将处理好的数据传输进入到下一个 bolt 进行新的处理。因此，Storm 系统对于数据的处理时间是极快的。

Storm 系统由于对于数据的处理极快，因此适合流式数据，而实验要求的是根据已有信息，对于骑行路线进行时间的估计，这个时间估计有很高的时效性，因此适合使用 Storm 来处理。

在这里需要注意一点，Storm 和 Map-Reduce 这样的框架不同，Map-Reduce 是等数据 Map 全部处理结束以后，再进行 Reduce（即中间有一个等待的过程），将所有 key 值相同的数据作为同一个 Reduce 的任务来处理。而 Storm 强调的是时效性，当这一部分数据处理完成后，马上发送给下一个部分进行数据处理过程，中间是没有等待的过程。即：没有办法依据其它 bolt 的数据对自己的 bolt 的数据进行处理，这是使用 Storm 系统处理问题的难点。

c. 你是如何使用这个系统

1.使用的是 Windows 10，Eclipse，Java 8-Maven 来使用系统，在 pom.xml 定义 Storm 的 Maven 依赖后在 Eclipse 编写程序。
2.编写系统的 Spout 和 Bolt 完成实验的功能。
3.将 Storm 和 Redis 集成——输出的结果成功存放到 Redis 数据库中。
4.等程序编写完成后，配置伪集群，将任务打包成 jar 包提交成功。

3. 系统设计/算法设计

a. 系统架构/算法原理的描述，详细描述系统每个模块的功能和实现

Spout 部分：
dataSpout:
①读取数据集的数据，读取的数据来自同一目录下的 csv 文件（即实验要求使用的自行车轨迹数据集）。
②读取数据以后，将数据搭建成为一个张量（即一个三维的矩阵），可以将其看作有 x、y、z 三个维度，x 和 y 维度代表的是地点（代表从地点 x 出发，到达地点 y），z 代表的是时间范围（将一天分为 12 份，每一份是 2 个小时）。该张量记为 A。
每一个点的值代表的是在时间范围 z 中，从 x 出发，到达 y，所花费的时间是多少。
③读取 query 的数据，即问询路线 a->b->c->d->e 在时间 t 所花费需要的时间。
④将问询的数据分为多份，分发到 ConstructNearTimeBolt 中进行处理。传入的 tuple 包括被分割的问询数据 QL（querylist）、构建的张量 A，所有地点的属性特征 places。
Bolt 部分：
1、 ConstructNearTimeBolt:
①已经所有地点的信息，根据传入的地点特征中的经纬度，计算出任意两个地点所有的距离，使用的公式如下图:

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!*

②假设问询的数据是：在 t 时刻（例如凌晨 5 点）从地点 a 经过地点 b、c、d 最后到达地点 e 所花的时间是多少（以秒为单位）。
将被 dataSpout 分割后的一小部分问询数据根据问询的时刻进行排序
③由于我们将数据中的时间分为了 12 份（即一天的 24 小时中，我们将 2 小时作为一个时间范围），那么上面过程的排序结果也应该是对于我们问询的数据有 12 份的结果。
对于每一份结果，都有一个对应的时刻 t，在张量中找到 z 维度的三个与时间 t 相邻的平面，即时间 t，时间 t-1，时间 t+1 的所有数据（这么找的依据是，我们要问询从时刻 t 出发的路程时间，那么在其相邻时刻的数据的参考价值是比较大的）。
④将被排序的 12 份中的每一份，与相邻时刻的数据，分发到 PathSpiltBolt 进行下一个处理。这样，这个子任务被分发了 12 份下去。

2、 PathSpiltBolt：
这一个 bolt 得到的数据是问询相同时刻的问询列表 query-list，以及其相邻时刻的数据。
①根据相邻时刻的数据，做一个线性回归，其中，参数 X 是 ab 两点之间的距离，参数 Y 是从 a 走到 b 所需要花费的时间，因此可以得到一个参数 theta，由于数据量并不是特别大，这里使用的是正规矩阵的方法求解。
②只是根据简单的回归得到的结果是不够准确的，还需要一些处理才能得到一个更好的结果。在这里，我们将相邻时刻的数据的每一份，都作为一个点，构建出一颗 KD 树，这个 KD 数是四维的。每一个点代表的物理意义是：从地点 a 到地点 b 所花费的时间（关于时刻的问题，在之前的 ConstructNearTimeBolt 我们已经选取了相邻时刻来分发任务，因此这里不需要再考虑），因此，将 a 的经度、纬度与 b 的经度、纬度作为 KD 树的四个维度，以此得到了一棵 KD 树。
③由于我们现在得到的问询数据是一个问询列表，包含了多个问询，将其拆分成一个个单独的问询分发到 SpiltIntoOneRoadBolt，其中一起分发的数据还有：构建好的 KD 树以及线性回归得到的 theta。

3、 SpiltIntoOneRoadBolt：
由于每一个问询的形式都是：a->b->c->d->e 的格式，那么可以将问询再进行依次拆分，例如针对问询路径的时间：a->b->c->d->e，将其拆分成为 a->b 的时间，b->c 的时间，c->d 的时间，d->e 的时间。分别将这些时间计算任务分发到 TimeExtimateBolt。

4、 TimeExtimateBolt：
这一部分，我们得到的是每一段小路径的时间估计，可用的信息有，KD 树和线性回归得到的参数 theta。假设我们要计算的是路径 a->b 的时间：
①首先，根据 theta 和计算 ab 之间的距离，我们可以得到一个时间的估计。
②另一方面，利用 KD 树，找出和 a 和 b 地理位置最相似的几个点（利用经纬度），然后可以得到另一个距离与时间的关系，根据它也可以估计出从 a 到 b 之间的时间。
③综合这两部分之间，可以得到一个更为准确的时间估计。
计算结束后，将得到的时间分发到 TimeSummaryBolt 去取和，得到每一个问询的时间。在这里采用的分发策略是：fieldsGrouping。即，属于相同问询，但是被拆分的路径，所估计的时间必须要被分到同一个 Bolt 上处理（这有些类似于 Map-Reduce 的思想，key 值是问询的序号，value 是路径处理后得到的时间）。

5、 TimeSummaryBolt：

做一个 HashMap，key 值是问询的序号，value 是目前为止该问询累加的时间之和。
每当一个路径计算的结果到达该 Bolt 时，将其累加存储到 Hashmap 中，并将结果分发到 RedisStoreBolt 去。

6、 RedisStoreBolt:
Redis 和 Storm 之间程序有良好的接口，在 pom.xml 声明依赖关系以后，直接调用接口，可以将数据直接存储到 Redis 数据库中。

b. 实验中所用的算法介绍

1.正规方程法求线性回归
我们已知的是如下图矩阵:

$$X=\begin{bmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(\mathrm{m})} & \cdots & x_n^{(\mathrm{m})} \end{bmatrix}$$

要求得到的结果是:

$$Y=[y^{(1)},y^{(2)},\cdots,y^{(m)}]^T$$

找出:

$$\theta=[\theta_0,\theta_1,\theta_2\ldots\theta_n]^T$$

使得:

$$j(\theta_0,\theta_1,\theta_2\ldots\theta_n)=\frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)})-y^i)^2$$ 最小。

经过数学的推导，方法是:

$$\theta=(X^TX)^{-1}X^TY$$

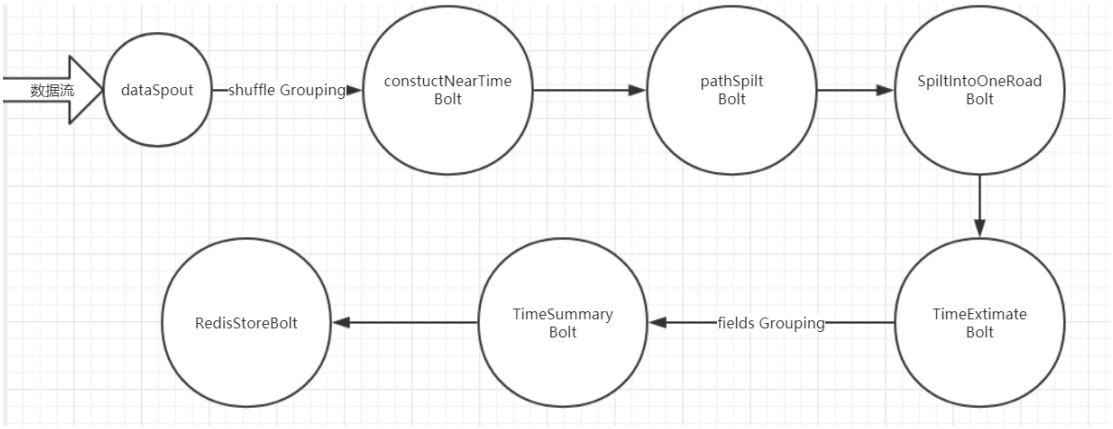
这属于机器学习的一个简单的应用。在本实验中，X 是两点之间的距离，Y 则是估计的时间。利用上图公式可以计算出参数 theta。

2.KD 树:
k-d 树是每个节点都为 k 维点的二叉树。所有非叶子节点可以视作用一个超平面把空间分割成两个半空间。节点左边的子树代表在超平面左边的点，节点右边的子树代表在超平面右边的点。选择超平面的方法如下：每个节点都与 k 维中垂直于超平面的那一维有关。

实验中使用的是四维的 KD 树,
构建过程:
将 KD 树的节点根据第 k'维的大小排序，选取中间节点为父亲，k 维较小的数据为中间节点的左儿子，k 维较大的数据为中间节点的右儿子。递归创建，其中，递归深度每增加一层，就使排序时候的维度加 1 模 4。

搜索过程:
使用的是欧几里得距离来定义距离,
首先通过二叉树搜索（比较待查询节点和当前节点的当前维度的值，如果小于，就进入左子树分支，大于就进入右子树分支，直到叶子结点结束），顺着“搜索路径”很快能找到最近邻的近似点，也就是与待查询点处于同一个子空间的叶子结点。
然后再回溯搜索路径,并判断搜索路径上的结点的其他子结点空间中是否可能有距离查询点更近的数据点，如果有可能，则需要跳到其他子结点空间中去搜索（将其他子结点加入到搜索路径）。重复这个过程直到搜索路径为空。

c. 系统架构框图



4. 实验流程

详细介绍你完成实验的主要流程

a. 主要过程描述，如系统搭建，论文阅读，算法设计实现等

1．论文阅读
论文是：Yilun Wang, Yu Zheng, Yexiang Xue. Travel Time Estimation of a Path using Sparse

Trajectories. In Proceedings of the 20th SIGKDD conference on Knowledge Discovery and Data Mining (KDD 2014).

这篇论文对我的设计有一定的参考价值，但是由于论文所解决的问题与我要解决的问题是有比较大的差别的，而且论文中的方法事实上在 Storm 架构上很难做到（具体原因后文提到），因此，我只是参考了论文的思路，并没有全盘使用论文的算法。

接下来简单的介绍一下论文的思想：

①论文使用的数据集的每一条记录是一辆汽车在某个时间段一条轨迹的总行驶时间，如下图：

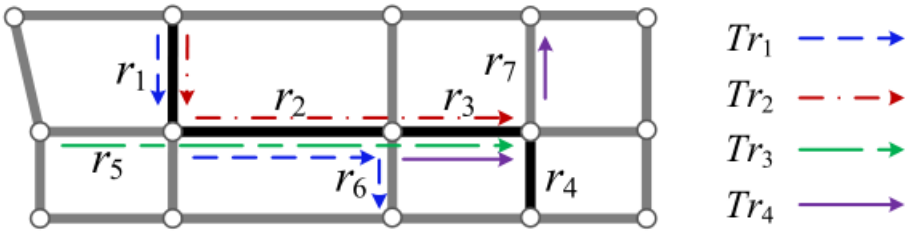


Figure 1. Problem demonstration

例如，给定了车辆 a 在 1 点时候出发，走完 Tr1（经过了 r1,r2,r6），所花费的所有时间。

当然，这样的路径有很多，组织成为了一个城市的交通网络。

②可以将城市的交通网络构建成为一个三维的张量（Tensor），其中三个维度分别代表路径 r，所行走的车辆，以及时间范围。而张量点的值代表花费时间。

③然而，由于交通的特点，有很多地方事实上是没有数据的（显然，车辆不可能每时每刻都在路上行走，并且很多道路由于比较偏远，大部分时候是没有车辆行走的），这样造成了数据的缺失，因此得到的张量是一个稀疏的张量，很大一部分的数据是没有的。

因此，将以前的轨迹数据也整合到这一张量中，减少张量的稀疏性。

④将刚刚得到的张量标记为 A，在定义两个矩阵（二维）X 和 Y，其中，Y 存储着路径的地理特征（例如这条路径的方向，距离，具体位置等等），X 存储着不同时间的路径花费的时间的关系。

⑤定义损失函数：

$$\mathcal{L}(S,R,U,T,F,G) = \frac{1}{2}\|\mathcal{A}_r - S \times_R R \times_U U \times_T T\|^2 + \frac{\lambda_1}{2}\|X - TG\|^2 + \frac{\lambda_2}{2}\|Y - RF\|^2 + \frac{\lambda_3}{2}(\|S\|^2 + \|R\|^2 + \|U\|^2 + \|T\|^2 + \|F\|^2 + \|G\|^2), \quad (3)$$

其中，S 是一个小的三维张量，R、U、T 是二维矩阵。

⑥使用梯度下降法，找出最适合的 S,R,U,T，使得 A 与 S×R×U×T 得到的三维张量，它们的损失函数最小，具体的算法如下：

Algorithm 1: Tensor Decomposition

Input: tensor $\mathcal{A}$, matrix X, and matrix Y, an error threshold ε
Output: R, U, T, S
1. Initialize $S \in \mathbb{R}^{d_R \times d_U \times d_T}$, $R \in \mathbb{R}^{N \times d_R}$, $U \in \mathbb{R}^{M \times d_U}$, $T \in \mathbb{R}^{2L \times d_T}$, $G \in \mathbb{R}^{d_T \times P}$, $F \in \mathbb{R}^{d_R \times Q}$ with small random values
2. Set η as step size
3. **While** $L_t - L_{t+1} > \varepsilon$
4. **Foreach** $\mathcal{A}_{ijk} \neq 0$
5. $Y_{ijk} = S \times_R R_{i*} \times_U U_{j*} \times_T T_{k*}$;
6. $R_{i*} \leftarrow R_{i*} - \eta\lambda_3 R_{i*} - \eta\left(Y_{ijk} - \mathcal{A}_{ijk}\right) \times S \times_U U_{j*} \times_T T_{k*} - \eta\lambda_2(R_{i*} \times F - Y_{i*}) \times F$;
7. $U_{j*} \leftarrow U_{j*} - \eta\lambda_3 U_{j*} - \eta\left(Y_{ijk} - \mathcal{A}_{ijk}\right) \times S \times_R R_{i*} \times_T T_{k*}$;
8. $T_{k*} \leftarrow T_{k*} - \eta\lambda_3 T_{k*} - \eta\left(Y_{ijk} - \mathcal{A}_{ijk}\right) \times S \times_R R_{i*} \times_U U_{j*} - \eta\lambda_1(T_{k*} \times G - X_{k*}) \times G$;
9. $S \leftarrow S - \eta\lambda_3 S - \eta\left(Y_{ijk} - \mathcal{A}_{ijk}\right) \times R_{i*} \otimes U_{j*} \otimes T_{k*}$;
10. $G \leftarrow G - \eta\lambda_3 G - \eta\lambda_1(T_{k*} \times G - X_{k*}) \times T_{k*}$;
11. $F \leftarrow F - \eta\lambda_3 F - \eta\lambda_2(R_{i*} \times F - Y_{i*}) \times R_{i*}$;
12. **Return** R, U, T, S

Figure 5. Algorithm for decomposing a tensor

⑦经过前面的过程，可以得到一个密集的张量了，可以根据这一张量求解最优的拼接（即估计路径的时间）。论文中，若是求解的路径时间在给定的数据中有，可以直接使用，否则就使用拟合后的张量中的值代替。

然而，论文的最重要方法就是利用张量的分解去构造一个与原张量相似的张量，以填补稀疏数据，然而，它所使用的梯度下降法，并不适合用于 storm 上面。因为 storm 要求把任务分发出去完成，意味着分发后，各部分被分发到不同 bolt 的数据是没有办法共享的，而梯度下架的任务并没有办法分发（分发出去以后，矩阵的值会修改，但是过程的下一步，又需要对于在不同 bolt 使用被修改的矩阵的值），因此，这一任务在我看来，是不适合 storm 来完成的。因此，我体会了论文设计构造的思路，自己设计的框架的运行过程与时间估计的方法。

2.使用系统框架编写程序
如果是单纯的编写程序，配置 storm 是一件非常简单的事情。
由于我所使用的是 eclipse IDE 的 Maven 项目，因此只要在：pom.xml 声明：

```
<dependency>
    <groupId>org.apache.storm</groupId>
    <artifactId>storm-core</artifactId>
    <version>1.2.2</version>
    <scope>provided</scope>
</dependency>
```

就可以进行程序的编写了。
如果需要配置伪集群，那么会有很多的坑。具体配置伪集群的部分在 3 中阐述，这一部分阐述的是设计的过程
①如果将问题用平白的语言所说，那么就是给定一堆自行车的轨迹数据，让你估计其它轨迹的行驶时间。而问题的难点在于，给定的数据是稀疏的，有很多数据是缺失的。因此需要填补缺失的部分。
②观察数据集：

tripduratic	starttime	stoptime	start statio	start statio	start statio	start statio	end statio	end statio	end statio	end statio	bikeid	usertype	birth year	gender
326	######	######	239	Willoughb	40.69197	-73.9813	366	Clinton Av	40.69326	-73.9689	16052	Subscribe	1982	1
729	######	######	322	Clinton St	40.69619	-73.9912	398	Atlantic Av	40.69165	-74	19412	Customer	\N	0
520	######	######	174	E 25 St &	40.73818	-73.9774	403	E 2 St & 2	40.72503	-73.9907	19645	Subscribe	1984	1
281	######	######	430	York St &	40.70149	-73.9866	323	Lawrence	40.69236	-73.9863	16992	Subscribe	1985	1
196	######	######	403	E 2 St & 2	40.72503	-73.9907	401	Allen St &	40.7202	-73.99	15690	Subscribe	1986	1
1948	######	######	369	Washingtc	40.73224	-74.0003	307	Canal St &	40.71427	-73.9899	19846	Subscribe	1977	1
1327	######	######	254	W 11 St &	40.73532	-73.998	539	Metropoli	40.71535	-73.9602	14563	Subscribe	1986	2

从左到右分别是，这一条路径的行驶时间，出发时间，到达时间，出发的站点序号，出发的站点名称，出发地的经度，出发地的纬度，目的地的站点序号，目的地的站点名称，目的地的经度，目的地的纬度，自行车序号，骑车人的身份，骑车人的出生年份，骑车人的性别。可以这么猜想：行驶的时间与路径的距离是有很大的关系的。同时，由于早上和晚上的光线视野，或者下班、上班的高峰期，同一段路径的行驶时间，会根据出发时间不同而有巨大的差异。因此，将一天的 24 小时分成 12 份，每一份是 2 个小时，估计时间时，将这两个因素着重考虑。
③一开始我打算使用的是论文的方法，因此调用了如下的一个矩阵运算库：

```
<dependency>
    <groupId>colt</groupId>
    <artifactId>colt</artifactId>
    <version>1.2.0</version>
</dependency>
```

并且自己实现了很多论文中需要用到的方法，（该库只支持二维的矩阵运算，以及二维和三维的矩阵声明），例如生成随机小数的二维和三维的矩阵，三维张量和二维矩阵相乘，取出二维矩阵的某一行或某一列，以及三维矩阵投影都二维矩阵，还有矩阵做克鲁尔乘积运算（Korn）等等，接下来便开始编写论文的方法。并且做了修改（由于距离与时间关系很大，我同时拟合一个距离和时间的关系矩阵，将其加入到损失函数）。但是想将其放到 Storm 框架中发现其并不适合，因此放弃。

④利用论文的思想，同样是构建了一个张量，不同的是，三维分别代表地点 x，地点 y，出发的时间 z，如（1,2,3）的值是 4，代表从地点 1 出发到达地点 2，在时间 3 的时候发现，花费的时间是 4。

接下来进行尝试：

query	time	number								
path0	13:05:42	3	W 15 St &	Lawrence	W 38 St & 8 Ave					
path1	8:34:28	2	5 Ave & E	W 18 St & 6 Ave						
path2	12:14:29	8	W 45 St &	Greenwich	Pearl St &	E 20 St &	8 Ave & W	Grand St &	Clinton Av	Pershing Square N
path3	7:02:41	8	E 59 St &	W 47 St &	Broadway	Lefferts Pl	9 Ave & W	Macon St	Broadway	St James Pl & Pearl S
path4	20:45:16	6	W 11 St &	Columbia	Fulton St	Columbia	E 20 St &	York St & Jay St		

先介绍一下问询的数据：

这是自己生成的，time 代表出发时间，number 列代表经过的地点个数，后面的字母代表地点的名称。

例如询问 path0，从 13:05:42 出发，经过

W 15 St & 7 Ave	Lawrence St & Willoughby St	W 38 St & 8 Ave

三个地点，所花费的时间是多少。

<1.第一次尝试：这里我借用了简单的线性回归的思想，比如我要询问，在时刻 t 出发，x->y 所需要花费的时间是多长，那么寻找与 t 时刻相邻的时刻的数据，根据这些数据中所包含的路径距离与花费时间关系，可以做一个线性回归，然后再根据 x->y 的距离关系，套用刚刚的回归结果，就可以得到一个结果。当然，这样的结果并不够好。

<2.第二次尝试：可以使用 KNN 的思想，加入问询的还是：在时刻 t 出发，x->y 所需要花费的时间是多长，那么，根据数据中的时刻 t，x 的位置信息，y 的位置信息，设计出了一个距离函数，找到与问询最相近的几组数据，根据这几组数据的花费时间，取均值去估计问询的花费时间。这一个思想很简单，但是却能极好的利用 storm 框架的特点——将每一次问询都分发出去分别估计。

<3.第三次尝试：构建 KD 树，对于相同时刻的问询，将其相邻时刻的数据拿过来构造一颗 KD 树，包含的属性是：出发点的经度，出发点的纬度，目的地的经度，目的地的纬度。

接下来将问询拆分成小的路径估计时间以后，利用 KD 树，找出与问询出发地点和目的地点最相似的几个点，利用它们的距离时间关系，去估计问询的距离时间关系。

这一个部分的优先在于，充分考虑了出发地点与目的地点的地理特征。

<4.第四次尝试：综合第二次尝试与第三次的优点，将它们得到的结果进行综合，得到最终的问询路径时间估计。

⑤编写程序

主要的原理在之前的第二部分系统架构写过了，编写的主要是如下的部分：

DataSpout

ConstructNearTimeBolt

PathSplitBolt
SplitIntoOneRoadBolt
TimeExtimateBolt
TimeSummaryBolt
RedisStoreBolt
具体的各部分输入输出在该部分的 b 和 c 中阐述。
⑥生成测试的问询数据
就是将读入的数据集作为基础，记录其出现过的地点信息，然后设置随机数，随机生成问询的时间和地点。
⑦尝试运行（在没有集成 Redis 的情况）
这里出现了一个小的问题，程序报错：告诉我传递的东西不是 Serialize 的，我去网上找了原因，发现问题在于：模拟集群运行的时候，数据结构在机器间传递的要求是，这个数据必须是实现了 Serializeble 接口的。

因此在自己设计的结构加上：`implements Serializable{` 即可。
⑧集成 Redis
在 https://github.com/MSOpenTech/redis/releases 下载。
解压以后：使用 cmd，切换到 redis 目录运行：

```
redis-server.exe redis.windows.conf
```

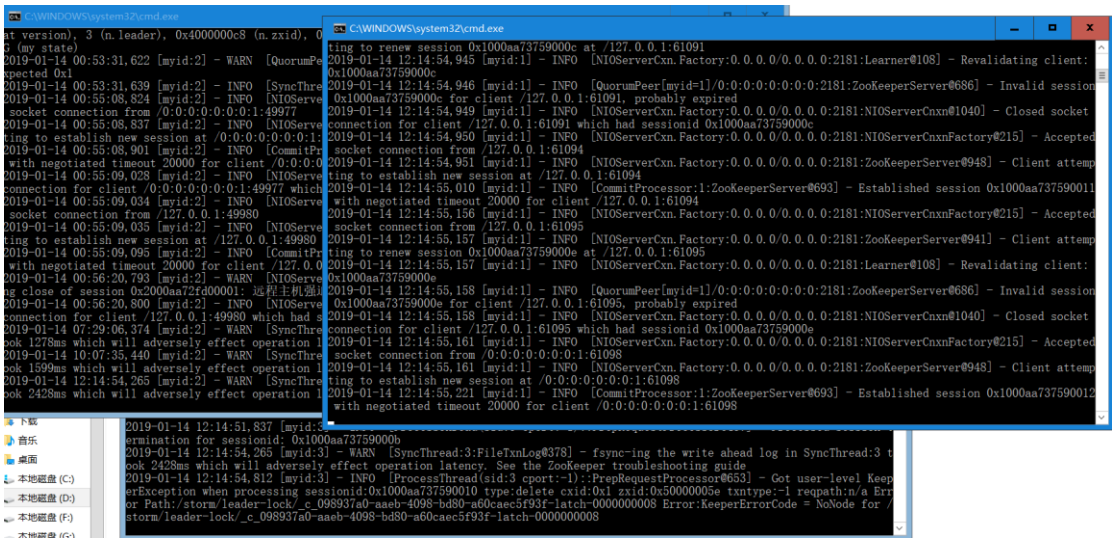
在程序中调用接口 storm 与 Redis 的接口。
使用 Redis-desktop 可视化界面观察结果。
遇到的坑：Redis 在挂着代理的时候，会报错。当时电脑挂着 SS，结果程序一直运行失败，找了很久的原因才发现是这个问题，需要将代理关系才能成功运行。

3.配置伪集群
①首先，程序方面原来是使用的 storm 所提供的接口，模拟集群方式运行，现在修改为：StormSubmitter.submitTopology。
②从官网上下载 zookeeper，将其解压以后，复制粘贴 3 份，开始配置 zookeeper 伪集群。将这 3 份 zookeeper 文件夹分别命名为 zookeeper，zookeeper2，zookeeper3。依次进入其 conf 文件夹下的 zoo.cfg。

```
tickTime=2000

initLimit=10

syncLimit=5

dataDir=D://zookeeper//data
clientPort=2181

server.1=localhost:2887:3887
server.2=localhost:2888:3888
server.3=localhost:2889:3889
```

其中 zookeeper 中的 clientport 设置为 2181，zookeeper2 中的设置为 2182，zookeeper3 的设置为 2183。
踩坑：不可以先将 zookeeper 的 zoo.cfg 修改以后，直接把整个 zookeeper 目录复制粘贴，会出现奇怪的错误，具体运行的时候集群会抛出异常，是一个非常奇怪的错误。必须先复制

粘贴以后分别修改。

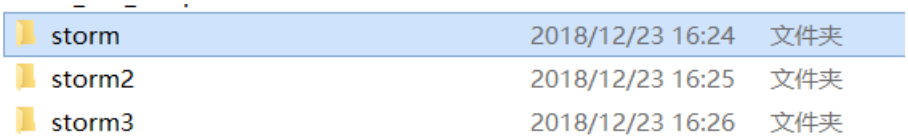


分别进入三个 zookeeper 的文件夹，进入 bin 目录，打开 zkServer.exe。发现三个命令行如上图所示即配置成功 zookeeper 集群。

③配置 storm 伪集群

同样是去 storm 官网下载压缩包，然后解压。

与 zookeeper 一样，复制粘贴拷贝 3 份，分别命名为 storm，storm2，storm3.



分别修改其目录下的 conf 文件夹中的 storm.yaml

```
########## These MUST be filled in for a storm configuration
storm.zookeeper.servers:
    - "localhost"
storm.zookeeper.port: 2181
nimbus.host: "localhost"
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703

storm.local.dir: "D:\\storm\\localdata"
storm.messaging.transport: "backtype.storm.messaging.netty.Context"
storm.messaging.netty.server_worker_threads: 1
storm.messaging.netty.client_worker_threads: 1
storm.messaging.netty.buffer_size: 5242880
storm.messaging.netty.max_retries: 100
storm.messaging.netty.max_wait_ms: 1000
storm.messaging.netty.min_wait_ms: 100
```

```
########## These MUST be filled in for a storm configuration
storm.zookeeper.servers:
    - "localhost"
storm.zookeeper.port: 2182
nimbus.host: "localhost"
supervisor.slots.ports:
    - 7700
    - 7701
    - 7702
    - 7703

storm.local.dir: "D:\\storm2\\localdata"
storm.messaging.transport: "backtype.storm.messaging.netty.Context"
storm.messaging.netty.server_worker_threads: 1
storm.messaging.netty.client_worker_threads: 1
storm.messaging.netty.buffer_size: 5242880
storm.messaging.netty.max_retries: 100
storm.messaging.netty.max_wait_ms: 1000
storm.messaging.netty.min_wait_ms: 100
```

```
########### These MUST be filled in for a storm configuration
storm.zookeeper.servers:
    - "localhost"
storm.zookeeper.port: 2183
nimbus.host: "localhost"
supervisor.slots.ports:
    - 8700
    - 8701
    - 8702
    - 8703

storm.local.dir: "D:\\storm3\\localdata"
storm.messaging.transport: "backtype.storm.messaging.netty.Context"
storm.messaging.netty.server_worker_threads: 1
storm.messaging.netty.client_worker_threads: 1
storm.messaging.netty.buffer_size: 5242880
storm.messaging.netty.max_retries: 100
storm.messaging.netty.max_wait_ms: 1000
storm.messaging.netty.min_wait_ms: 100
```

三个 storm.yaml 分别修改为如上图所示。
接下来，打开 cmd，切换目录到 storm 的 bin 文件夹，输入 storm.py nimbus,。
打开第二个 cmd，一样切换到 storm 的 bin 文件夹，输入 storm.py supervisor。
打开第三个 cmd，一样切换到 storm 的 bin 文件夹，输入 storm.py ui。
打开第四个 cmd，一样切换到 storm2 的 bin 文件夹，输入 storm.py supervisor。
打开第五个 cmd，一样切换到 storm3 的 bin 文件夹，输入 storm.py supervisor。
将打包的 storm 的 jar 包提交到拓扑上即可。
提交过程：打开 cmd，切换目录到 storm 的 bin 文件夹，输入 storm.py ……jar org.apache.storm.Storm_Experiment.Storm(即程序执行的主类) xx(自己取的任意名字)，可以成功提交，即完成了。

踩坑：这里的坑是非常多的。
首先，和刚刚一样的问题，storm 集群也必须先拷贝再修改，否则会无法运行。
其次，网上的资料其实非常少，而且大部分都是非集群上搭建的 storm 系统，伪集群的资料凤毛麟角，例如 storm.yaml 文件：

```
storm.zookeeper.servers:
    - "localhost"
storm.zookeeper.port: 2183
```

网上的资料大多是真正的几台机器搭建，那么再 storm.zookeeper.servers 的地址写的是：各个机器的 IP 地址。然而，我这是伪集群的搭建，只有一个 IP 地址。
其次，storm.zookeeper.port 只能设置一个端口，不能设置多个。这个本意是说，我有多台机器，所有的机器都运行在一个端口上。然而，伪集群必须设置成为多个端口（我试过设置成一个端口的情况，会提示端口发生冲突，zookeeper 直接无法运行）。
我曾一度以为单机器是无法配置成为多节点的伪集群的，然而我最终还是配置成功了，将三个 storm.yaml 改成与 zookeeper 对应的端口以后，是可以成功将它们设置为主从节点的。
第二个坑主要是说，如果我的机器提交过一次拓扑以后，修改程序再次提交，可能会报错：org.apache.storm.utils.NimbusLeaderNotFoundException: Could not find leader nimbus from seed hosts [localhost]. Did you specify a valid list of nimbus hosts for config nimbus.seeds?
这样的问题只要将 storm 的 data 目录中的所有文件直接删除就解决了，似乎是因为之前数据缓存的问题。我当时一直认为是我的配置出现问题了，结果总是失败，后来在 Stack Overflow 才找到解决办法。
第三个坑是说，在启动了 storm.py supervisor 以后，这个 supervisor 有时候没有任务会自动断掉，如果需要必须重新启动。

```
D:\storm2\bin>storm.py supervisor
Running: C:\Program Files\Java\jdk1.8.0_161\bin\java.exe -server -Ddaemon.name=supervisor -Dstorm.options= -Dstorm.h
D:\storm2 -Dstorm.log.dir=D:\storm2\logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.fil
cp D:\storm2\*;D:\storm2/lib\*;D:\storm2/extlib\*;D:\storm2/extlib-daemon\*;D:\storm2\conf -Xmx256m -Dlogfile.name=s
visor.log -Dlog4j.configurationFile=D:\storm2\log4j2\cluster.xml org.apache.storm.daemon.supervisor.Supervisor
```

启动时如上图所示：

```
D:\storm\bin>storm.py supervisor
Running: C:\Program Files\Java\jdk1.8.0_161\bi
D:\storm -Dstorm.log.dir=D:\storm\logs -Djava.
 D:\storm\*;D:\storm/lib\*;D:\storm/extlib\*;D
og -Dlog4j.configurationFile=D:\storm\log4j2\c


D:\storm\bin>_
```

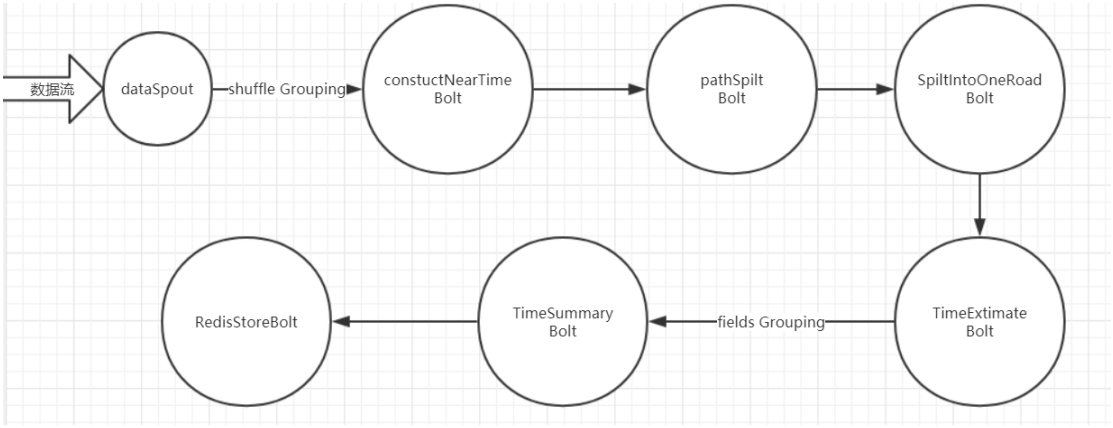
忽然停止时如上图所示：

如果想要重新执行，必须重新输入 storm.py supervisor。

④redis 数据库

踩坑：redis 数据库最大的坑就是，我挂着代理（SS）的时候它拒绝让我访问……，我把代理关掉以后访问成功。

b. 每个算法/系统模块输入与输出数据示例截图，并作详细描述



1. DataSpout:

《1》输入部分：

读取数据文件：

tripduratic	starttime	stoptime	start static	start static	start static	start static	end statio	end statio	end statio	end statio	bikeid	usertype	birth year	gender
326	######	######	239	Willoughb	40.69197	-73.9813	366	Clinton Av	40.69326	-73.9689	16052	Subscriber	1982	1
729	######	######	322	Clinton St	40.69619	-73.9912	398	Atlantic Av	40.69165	-74	19412	Customer	\N	0
520	######	######	174	E 25 St &	40.73818	-73.9774	403	E 2 St & 2	40.72503	-73.9907	19645	Subscriber	1984	1
281	######	######	430	York St &	40.70149	-73.9866	323	Lawrence	40.69236	-73.9863	16992	Subscriber	1985	1
196	######	######	403	E 2 St & 2	40.72503	-73.9907	401	Allen St &	40.7202	-73.99	15690	Subscriber	1986	1
1948	######	######	369	Washingto	40.73224	-74.0003	307	Canal St &	40.71427	-73.9899	19846	Subscriber	1977	1
1327	######	######	254	W 11 St &	40.73532	-73.998	539	Metropoli	40.71535	-73.9602	14563	Subscriber	1986	2

从左到右每一列分别代表：

行驶时间 出发时间 到达时间 出发站点序号 出发站点名称 出发站点经度 出发站点经度 到达站点序号 到达站点名称 到达站点经度 到达站点维度 自行车的 id 自行车手的类型 自行车手的出生年份 自行车手的性别

读取的询问如下图：

query	time	number										
path0	13:05:42	3	W 15 St &	Lawrence	W 38 St & 8 Ave							
path1	8:34:28	2	5 Ave & E	W 18 St & 6 Ave								
path2	12:14:29	8	W 45 St &	Greenwich	Pearl St &	E 20 St &	8 Ave & W	Grand St &	Clinton Av	Pershing Square N		
path3	7:02:41	8	E 59 St &	W 47 St &	Broadway	Lefferts Pl	9 Ave & W	Macon St	Broadway	St James Pl & Pearl St		
path4	20:45:16	6	W 11 St &	Columbia	Fulton St &	Columbia	E 20 St &	York St & Jay St				
path5	4:58:41	2	W 29 St &	E 51 St & Lexington Ave								
path6	15:13:38	8	E 20 St &	Adelphi St	Montague	E 37 St &	Clinton St	Cumberla	W 15 St &	Greenwich St & N Moore St		
path7	11:34:57	5	1 Ave & E	E 16 St &	Madison S	Sands St &	1 Ave & E 15 St					
path8	6:26:53	4	3 Ave & St	Clark St &	Lispenard	Mercer St & Bleecker St						
path9	17:45:28	5	E 10 St &	6 Ave & B	W 45 St &	DeKalb Av	Broadway & E 14 St					
path10	4:49:14	2	E 3 St & 1	Little West St & 1 Pl								
path11	5:03:32	5	William St	E 10 St &	Madison S	William St	Vesey Pl & River Terrace					
path12	8:44:57	2	Dean St &	Washington Pl & Broadway								
path13	5:03:56	5	Canal St &	Cumberla	W 39 St &	6 Ave & W	W 31 St & 7 Ave					
path14	7:48:30	8	Ashland P	Lafayette	E 14 St &	DeKalb Av	Centre St	South St &	W 20 St &	Hancock St & Bedford Ave		
path15	6:22:28	3	Henry St &	Montague	Hicks St &	Montague St						
path16	1:14:37	5	W 52 St &	Fulton St &	Hudson St	10 Ave & W	W 29 St & 9 Ave					
path17	0:52:26	2	10 Ave & W	W 43 St & 6 Ave								

从左到右每一列分别代表：
问询的路　出发的时间　经过的地点数量　经过的各个地点名称
要计算的是自行车行驶的时间。

《2》输出部分：

```
declarer.declare(new Fields("QL","tensor","places"));
```

问询的列表，张量，各个地点的位置信息。
其中，将每一万个循环分成一份，然后张量是根据数据集构造的张量 A，

```
declarer.declare(new Fields("qlst","points","dts","places"));
```

各个地点位置信息（存放着它们的经纬度）

2.ConstructNearTimeBolt:
《1》输入部分：
即 dataSpout 的输出部分，与前面一致。
《2》输出部分：

```
declarer.declare(new Fields("qlst","points","dts","places"));
```

分别是刚刚传入的问询列表，张量中与问询相邻的时间的节点（用于构造后面的 KD 树）
距离-时间的关系列表　以及之前传入的各个地点位置信息。

3.PathSplitBolt:
《1》输入部分：
即 ConstructNearTimeBolt 的输出部分，与前面一致。
《2》输出部分：

```
declarer.declare(new Fields("seq","q","queryslot","theta","kdtree","places"));
```

分别是：
问询的序号（如问询文件中第一列对应的是 path3312，对应的序号是 3312）
经过的路径序号（如 a->b->c，a 对应序号是 1，b 对应是 2，c 对应是 3），那么这个传入的就是（1,2,3）

Theta：回归算出的参数
kdtree：根据刚刚传入的点构造的 kd 树
places：依旧是各个地点的地理位置信息

4.SplitIntoOneRoadBolt:
《1》输入部分：
即 PathSplitBolt 的输出部分，与前面一致。
《2》输出部分：

```
declarer.declare(new Fields("seq","x","y","theta","kdtree","places"));
```

每个问询的可能包括（a->b->c）的时间，可以拆成 a 到 b 的时间+b 到 c 的时间
因此输出是分别是：
问询的序号
出发地点序号 x
目的地序号 y
回归的参数 theta
构造的 kd 树
各个地点的地理位置信息

5.TimeExtimateBolt:
《1》输入部分：
即 SplitIntoOneRoadBolt 的输出部分，与前面一致。
《2》输出部分：

```
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("seq","extimate"));
}
```

分别是：
问询的序号
估计的小段路径的时间

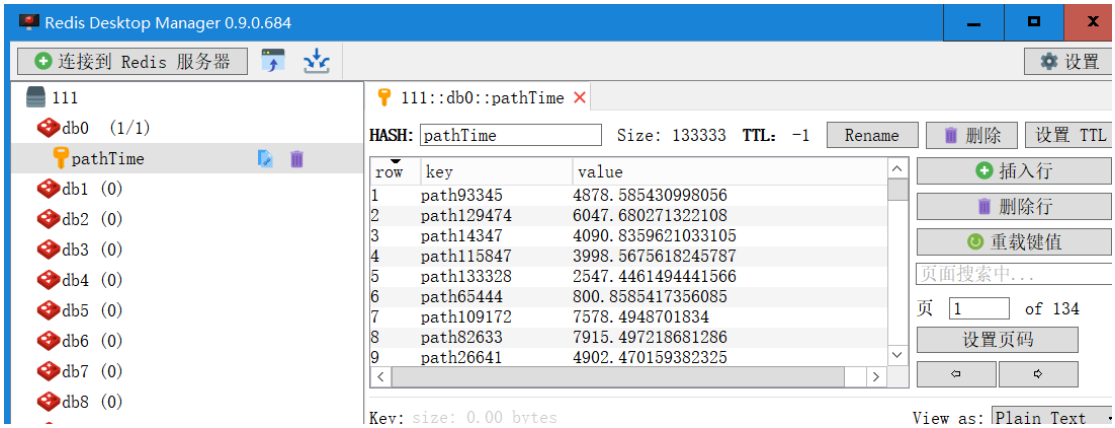
6.TimeSummaryBolt:
《1》输入部分：
即 TimeExtimateBolt 的输出部分，与前面一致。
《2》输出部分：

```
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("seq","time"));
}
```

问询的序号，以及将划分的小路径求和的总时间

7.RedisStoreBolt:
《1》输入部分：
即 TimeSummaryBolt 的输出部分，与前面一致。
《2》输出部分：

输出到 redis 数据库中：如下图



row	key	value
1	path93345	4878.585430998056
2	path129474	6047.680271322108
3	path14347	4090.8359621033105
4	path115847	3998.5675618245787
5	path133328	2547.4461494441566
6	path65444	800.8585417356085
7	path109172	7578.4948701834
8	path82633	7915.497218681286
9	path26641	4902.470159382325

Key 值就是问询的具体序号，value 值是求得的总时间。

5. 实验

测试你设计实现的系统/算法的效果

a. 测试环境：CPU、内存、磁盘

CPU：i5-6700HQ

内存：8G

磁盘：128SSD+1T 机械（storm 框架并不需要使用硬盘，只需要使用内存）

b. 数据集：数据集的来源（是自己生成的还是下载的，生成程序是如何编写的，下载的数据是选取了一部分还是全部使用了）、数据文件或数据库占用磁盘空间大小、包含的记录个数（行数）、包含的属性及其意义

数据集来源：从以下网站下载，一共 190+MB，全部使用

https://www.microsoft.com/en-us/research/publication/detecting-urban-black-holes-based-on-human-mobility-data/

tripdurati	starttime	stoptime	start stati	start stati	start stati	start stati	end statio	end statio	end statio	end statio	bikeid	usertype	birth year	gender
326	######	######	239	Willoughb	40.69197	-73.9813	366	Clinton Av	40.69326	-73.9689	16052	Subscribe	1982	1
729	######	######	322	Clinton St	40.69619	-73.9912	398	Atlantic Av	40.69165	-74	19412	Customer	\N	0
520	######	######	174	E 25 St &	40.73818	-73.9774	403	E 2 St & 2	40.72503	-73.9907	19645	Subscribe	1984	1
281	######	######	430	York St &	40.70149	-73.9866	323	Lawrence	40.69236	-73.9863	16992	Subscribe	1985	1
196	######	######	403	E 2 St & 2	40.72503	-73.9907	401	Allen St &	40.7202	-73.99	15690	Subscribe	1986	1
1948	######	######	369	Washingto	40.73224	-74.0003	307	Canal St &	40.71427	-73.9899	19846	Subscribe	1977	1
1327	######	######	254	W 11 St &	40.73532	-73.998	539	Metropoli	40.71535	-73.9602	14563	Subscribe	1986	2
1146	######	######	490	8 Ave & W	40.75155	-73.9939	438	St Marks P	40.72779	-73.9856	16793	Subscribe	1959	1
380	######	######	468	Broadway	40.76527	-73.9819	385	E 55 St &	40.75797	-73.966	16600	Customer	\N	0
682	######	######	300	Shevchenl	40.72815	-73.9902	519	Pershing S	40.75188	-73.9777	15204	Subscribe	1992	1
863	######	######	322	Clinton St	40.69619	-73.9912	398	Atlantic Av	40.69165	-74	19494	Customer	\N	0
664	######	######	482	W 15 St &	40.73936	-73.9993	505	6 Ave & W	40.74901	-73.9885	17241	Subscribe	1973	1
324	######	######	463	9 Ave & W	40.74207	-74.0044	254	W 11 St &	40.73532	-73.998	19330	Subscribe	1991	1
537	######	######	285	Broadway	40.73455	-73.9907	462	W 22 St &	40.74692	-74.0045	16872	Subscribe	1974	1
400	######	######	499	Broadway	40.76916	-73.9819	352	W 56 St &	40.76341	-73.9772	19690	Subscribe	1974	1
777	######	######	410	Suffolk St	40.72066	-73.9852	532	S 5 Pl & S	40.71045	-73.9609	16626	Subscribe	1985	1
653	######	######	504	1 Ave & E	40.73222	-73.9817	531	Forsyth St	40.71894	-73.9927	20304	Subscribe	1987	2
410	######	######	307	Canal St &	40.71427	-73.9899	291	Madison S	40.71313	-73.9848	18353	Subscribe	1988	1
626	######	######	262	Washingto	40.69178	-73.9737	143	Clinton St	40.6924	-73.9934	17118	Subscribe	1970	1
900	######	######	441	E 52 St &	40.75601	-73.9674	379	W 31 St &	40.74916	-73.9916	17079	Subscribe	1964	1

CSV 文件，共有 1037713 行

从左到右每一列分别代表：

行驶时间 出发时间 到达时间 出发站点序号 出发站点名称 出发站点经度 出发站点经度 到达站点序号 到达站点名称 到达站点经度 到达站点维度 自行车的 id 自行车手的类型 自行车手的出生年份 自行车手的性别

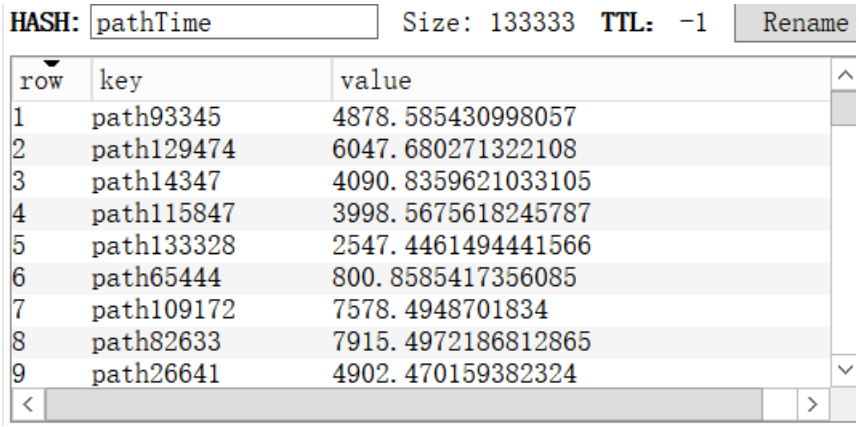
问询数据：自己生成
即根据下载的数据集所包含的地点，使用随机数生成了出发时间，以及经过的地点。

query	time	number									
path0	13:05:42	3	W 15 St &	Lawrence	W 38 St & 8 Ave						
path1	8:34:28	2	5 Ave & E	W 18 St & 6 Ave							
path2	12:14:29	8	W 45 St &	Greenwich	Pearl St &	E 20 St &	8 Ave & W	Grand St &	Clinton Av	Pershing Square N	
path3	7:02:41	8	E 59 St &	W 47 St &	Broadway	Lefferts Pl	9 Ave & W	Macon St	Broadway	St James Pl & Pearl St	
path4	20:45:16	6	W 11 St &	Columbia	Fulton St	Columbia	E 20 St &	York St & Jay St			
path5	4:58:41	2	W 29 St &	E 51 St & Lexington Ave							
path6	15:13:38	8	E 20 St &	Adelphi St	Montague	E 37 St &	Clinton St	Cumberla	W 15 St &	Greenwich St & N Moore St	
path7	11:34:57	5	1 Ave & E	E 16 St &	Madison S	Sands St &	1 Ave & E 15 St				
path8	6:26:53	4	3 Ave & St	Clark St &	Lispenard	Mercer St & Bleecker St					
path9	17:45:28	5	E 10 St &	6 Ave & B	W 45 St &	DeKalb Av	Broadway & E 14 St				

从左到右每一列分别代表：
问询的路　出发的时间　经过的地点数量　经过的各个地点名称
这是 CSV 文件，共有 133334 行，自己生成的数据。

b. 测试算法/系统的性能
系统处理 193MB 的数据集与 133334 行的问询数据时，所花费的时间一共是 67s（包括了系统模拟集群的时间，实际运行时间会更低一些）。
观测运行时候内存，其峰值最高约占用了 1500MB。

最终由于共有 133333 个问询，在 redis 数据库有 133333 条记录。



6. 结论

在我看来，我的实验完成得还是比较成功的。事实上给定一些数据，让你去估计另一段路径的时间这样的事情已经是被做过很多的了。
相关工作对比：
<1.例如给定的论文：Yilun Wang, Yu Zheng, Yexiang Xue. Travel Time Estimation of a Path using Sparse Trajectories. In Proceedings of the 20th SIGKDD conference on Knowledge Discovery and Data Mining (KDD 2014)。这一个主要就是使用的张量的 tucker 分解的方法，然后根据实际的问题去定义损失函数，来达到一个好的拟合效果，以填充稀疏的数据。而具体的方法则是使用梯度下降法，将各个矩阵的乘积逐步变得与原先的张量很相似则停止。

这样的方法适用于我已经有了一部分数据了，接下来我根据这部分数据做一个模型，再用这个模型去估计具体的问询的时间，由于使用的是梯度下降法，速度还是比较快的（唯一的速度花费就在于矩阵的乘法运算），但如果矩阵维度过高，事实上，速度会减慢很多。
<2. Traffic Estimation and Prediction Based on Real Time Floating Car Data
这是我自己在网络上找的一篇相关工作的论文，这一篇论文主要是根据我现在已有的车辆行驶速度，我去判断它未来的速度是怎么样的，依据这一速度信息来估计时间。其中使用了很多的数学公式与方法定义，简单地说，就是使用了很多数学公式与当前的汽车数据，来得到一个“speed pattern”，据此估计时间。

而我的方法在原理上，与第一个方法是比较相近的，事实上就是填充一个稀疏的三维张量。我使用的方法是 KD 树加上回归模型，从程序处理的速度上来说，假如问询数据是在几十万级别的来说，我的处理方法应该是速度更快的，因为我套用了 storm 框架，很好的运用了分布式计算的原理，而论文的方法需要先计算一个好的模型，才能开始估计，计算模型时间耗时巨大。但是如果问询到达千万级别，那么论文的方法会更好一些，因为它训练模型以后，后面的运算全部就是简单的加减了，而我的方法依旧是一个个去逐个计算问询。

从准确性来说，由于论文使用的数据集和我使用的数据集不同，针对我所使用的数据集，我认为我的准确性应该稍高一点。因为事实上，自行车行走一段路程的时间是和这一段路程强相关的，我使用的方法充分考虑了这一性质。