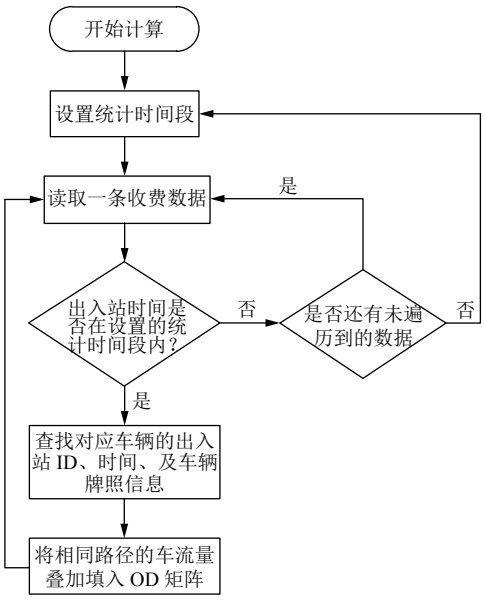

图 4 统计车流量 OD 矩阵的计算流程

第 1 步. 设置统计数据的时间间隔.

第 2 步. 读取一条数据, 根据统计的时间间隔判断该数据的出入站时间是否在该时间间隔内.

第 3 步. 如果是在时间间隔内, 查找该数据对应的车辆的进出站 ID 和时间、车辆牌照信息. 否则判断是否有未遍历到的数据, 若仍有未遍历到的数据, 则读取另一条数据, 否则重新设置统计数据的时间段.

第 4 步. 取进出站时间满足统计时间间隔的完整数据, 将相同路径的车流量叠加, 最终车流量放入流量 OD 矩阵中.

3 基于 HBase 的 OD 数据存储模型

HBase 是 Hadoop 家族中的 NoSQL 数据库, 是一个高可靠性、高性能、列存储、可伸缩、支持实时读写、面向联机事物处理、分布式的大型数据存储系统. 底层的 Hadoop 分布式文件系统 HDFS 具有高容错性且可以被部署在低价的硬件设备之上[8,9].

HBase 表主要由以下几部分构成:

表 (table): HBASE 在表中组织数据, 表名是字符串和字符的组合.

行 (row): 表中数据按水平划分为多行, 每行由唯一的 rowkey 确定.

行健 (rowkey): 是行的唯一标识数据, 没有数据类型, 一般是一个字节数组.

列族 (Column Family, CF): 数据在竖直方向划分成多个列族, 列族要预先定义, 且不容易修改, 每行数据都拥有相同的列族, 但是可能有些行的数据为空, 列族是字符串和字符的组合.

列限定符 (Column Qualifier, CQ, 也叫列标识): 数据在列族中的位置是通过列标识指定的, 每行的列标识可以不同, 列标识没有数据类型, 一般是一个字节数组.

单元 (cell): 单元是行健 (rowkey)、列族 (column family)、列限定符 (column qualifier) 的组合, 这些存储在单元中的数据被称为单元数据.

时间戳 (timestamp): 单元数据是有版本的, 每一个单元数据对应一个版本, 由时间戳来指定.

统计高速公路车辆旅行时间的 OD 矩阵在 HBASE 中存储模型如表 1 所示. 将设置的时间段、进站 ID、出站 ID 作为 RowKey, 将车辆的平均旅行时间作为 value.

表 1 车辆旅行时间 OD 矩阵物理模型

参数	类型
时间段	row, key
进站 ID	row, key
出站 ID	row, key
CF	key
CQ	key
平均旅行时间	value

统计高速公路车流量的 OD 矩阵物理模型如表 2 所示. 将设置的时间段、进站 ID、出站 ID 作为 rowkey, 将车流量作为 value.

表 2 车流量 OD 矩阵物理模型

参数	类型
时间段	row, key
进站 ID	row, key
出站 ID	row, key
CF	key
CQ	key
车流量	value

4 OD 矩阵计算在大数据环境下的实现

4.1 旅行时间 OD 矩阵计算方法在 MapReduce 下的实现

MapReduce 计算框架由 Input, Map, Shuffle, Reduce, Output 5 个部分组成.

Input 阶段: 读入数据并设置统计时间段.

Map 阶段: 读取所有输入数据, 对每行数据进行解析. 以“出站时间+进站点+出站点”作为 key, 计算旅行时间作为 value.

Shuffle 阶段: key 值相同的 value 会存入一组, 传送到 Reduce.

Reduce 阶段: 针对每个 key, 统计 value 的个数 count, 并计算 value 和 sum, 计算 value 均值 sum/count 作为新输出 value.

Output 阶段: 以 key-value 的方式输出计算结果, 最终的 key 为“出站时间+进站点+出站点”, value 为“平均旅行时间”.

该算法实现的流程如图 5 所示, 伪代码如算法 1 所示.

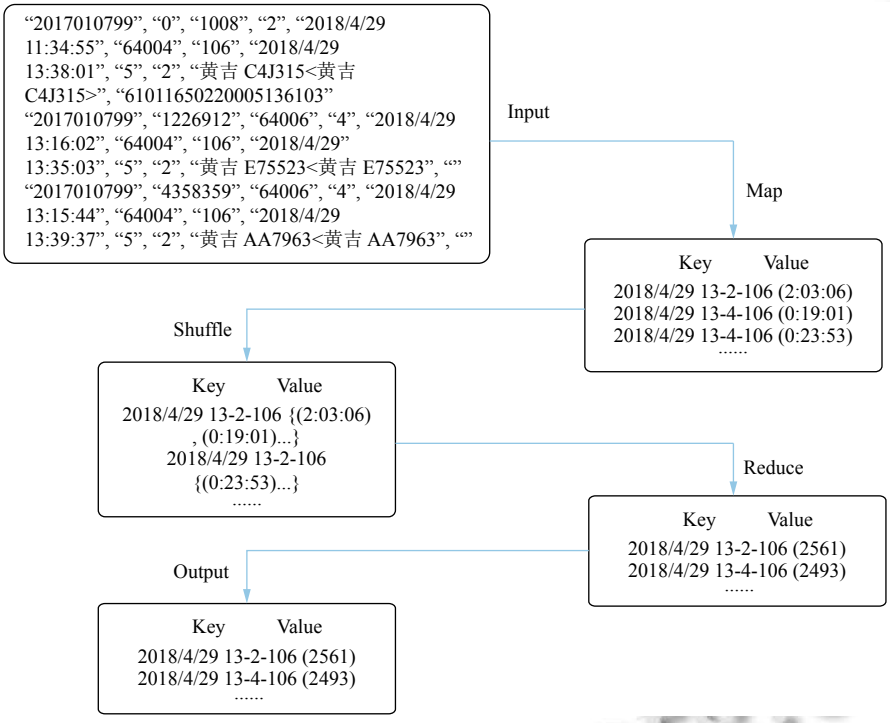

图 5 旅行时间 OD 矩阵实现流程

算法 1. 旅行时间 OD 矩阵计算方法

输入: 高速流量数据
输出: 旅行时间 OD 统计信息

```
1. function MAP(key, value, context)
2.    items = value.split("r");
3.    timeIn = items[4];
4.    timeOut = items[7];
5.    key = items[7].split(":")[0] + items[3] + items[6];
6.    odTime = timeOut – timeIn;
7.    context.write(key, odTime);
8. end function
9. function REDUCE(key, Iterable<ODTime>, context)
10.    count = 0;
11.    totaltime = 0;
12.    avg = 0;
13.    for ODTime TIME: L do
14.       totaltime = totaltime + TIME.getTOTALTIME();
15.       count + +;
16.    end for
17.    avg = totaltime/count;
18.    context.write(key, avg);
19. end function
```

4.2 车流量 OD 矩阵计算方法在 MapReduce 下的实现

车流量 OD 矩阵计算方法的实现与旅行时间类似. 主要区别在 Reduce 阶段, 需要对进出站的时间进行过滤.

Input 阶段: 读入数据并设置统计时间段.

Map 阶段: 读取所有输入数据, 对每行数据进行解析. 以“出站时间+进站点+出站点”作为 key, 旅行时间

作为 value.

Shuffle 阶段: key 值相同的 value 会存入一组, 传送到 Reduce.

Reduce 阶段: 针对每个 key, 统计 value 的个数 count, count 作为新输出 value.

Output 阶段: 以 key-value 的方式输出计算结果, 最终的 key 为“出站时间+进站点+出站点”, value 为“车流量数”.

该计算方法的流程如图 6 所示, 伪代码如算法 2 所示.

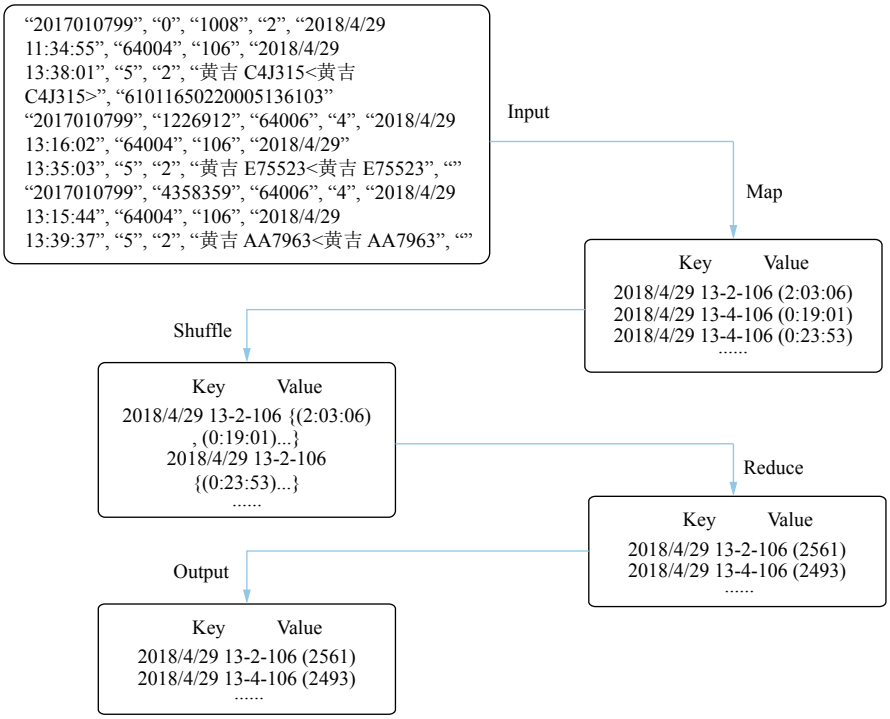

图 6 旅行时间 OD 矩阵实现流程伪代码

算法 2. 车流量 OD 矩阵计算方法

输入: 高速流量数据

输出: 车流量 OD 统计信息

```
1. function MAP(key, value, context)
2.    items = value.split(",");
3.    timeIn = items[4];
4.    timeOut = items[7];
5.    key = items[7].split(",")[0] + items[3]+ items[6];
6.    odTime = timeOut – timeIn;
7.    context.write(key, odTime);
8. end function
9. function REDUCE(key, Iterable<ODTime>, context)
10.    count = 0;
11.    totaltime = 0;
12.    for ODTimo TIME : L do
13.       totaltime = totaltime + TIME .getTOTALTIME();
14.       count + +;
15.    end for
16.    context.write(key, count);
17. end function
```

5 实验评价

5.1 实验环境

本文中生成 OD 矩阵所需的高速公路的收费数据数据量庞大, 但是对实时性要求不高, 所以选用在 Hadoop 集群环境下使用 MapReduce 分布式离线计算框架进行实验. 集群包含 3 个节点, 其中 1 个主节点, 2 个从节点, 每个节点的 CPU 都为 4 核, 运行内存 16 GB, 主机硬盘 2 TB.

5.2 实验数据

实验数据为河南省高速 2018 年的真实收费数据. 该数据记录了 300 余个高速公路出入口, 全年的车辆出入情况, 共计 10 743 万余条记录. 每条数据的记录形式如表 3 所示, 包含车牌号、驶入站点、驶出站点、驶入时间、驶出时间等 12 个属性.

为了方便进行实验和性能评估, 需要对数据进行过滤和清洗. 数据中最核心的属性就是车牌号, 所以按照标准的车辆牌照字符要求: 正规的车辆牌照应该为

7 位有效字符, 第一位为省份汉字简称, 第二位为大写英文字母形式的省内区域代号, 后五位应该是大写英文字母和数字的组合. 根据上述要求, 可能出现的不规范数据问题有如下几项:

(1) 车辆牌照第一个字符不是省份汉字简称.

(2) 省份汉字简称多余 1 个.

(3) 车辆牌照中出现非法字符, 如: 小写字母, 特殊符号等.

(4) 车辆牌照位数不足.

(5) 无牌照信息.

包含上述任何一条的数据将被认为是不符合要求的数据, 将这些数据过滤掉. 本文中提到的实验只需要各种类车辆在时间和空间上的信息, 所以将过滤后的数据进行进一步的压缩, 只保留车牌号、驶入站点、驶出站点、驶入时间、驶出时间和车辆类型这六个属性, 其他属性对于本次实验来说并非必要数据, 所以将其余属性去除以保证在程序运行和计算过程没有额外的不必要消耗, 保证程序运行性能.

表 3 数据属性

属性名称	数据类型	含义
CARDID	VARCHAR	MTC 卡编号
CARDNETWORK	VARCHAR	员工编号
ENTRYLANE	VARCHAR	进站栏位
ENTRYSTATION	VARCHAR	进站站点 ID
ENTRYTIME	VARCHAR	进站时间
ETCCPUID	VARCHAR	ETC 卡编号
EXITLANE	VARCHAR	出站栏位
EXITSTATION	VARCHAR	出站站点 ID
EXITTIME	VARCHAR	出站时间
VEHICLECLASS	VARCHAR	车辆种类
VEHICLELICENSE	VARCHAR	车牌号
VEHICLETYPE	VARCHAR	车辆类型

5.3 计算模型评价

实验对生成旅行时间 OD 矩阵和车流量 OD 矩阵所用时间进行了评价. 在相同的硬件条件下, 控制输入时数据的规模, 统计计算所用时间.

5.3.1 旅行时间 OD 矩阵

实验测试不同的统计天数统计车辆平均旅行时间的 OD 矩阵计算效率如表 4 所示, 实验证明, 数据量越大的情况下, 计算效率越高.

5.3.2 车流量 OD 矩阵

实验测试不同的统计天数统计车流量的 OD 矩阵计算效率如表 5 所示, 实验证明, 当数据量增大时, 计算效率增高.

表 4 不同数据量的旅行时间 OD 矩阵的计算效率

统计天数 (天)	输入数据条数 (千条)	输出数据条数 (千条)	计算时间 (ms)
7	1658	55	65 754
15	4149	124	70 318
30	8743	249	78 345
90	26 065	562	163 660

表 5 不同数据量的车流量 OD 矩阵的计算效率

统计天数 (天)	输入数据条数 (千条)	输出数据条数 (千条)	计算时间 (ms)
7	1658	56	63 004
15	4149	124	76 405
30	8743	249	79 512
90	26 065	734	174 828

5.4 存储模型评价

对于存储模型的设计, 主要考虑设计的模型占用的物理存储空间大小. 传统的存储方式主要是基于 MySQL 的关系型数据存储模式,这种存储方式不适合存储海量的数据, 需要多个表进行存储才便于对数据的处理, 而本文采用基于 HBase 的列式存储模式, 数据量相对较小时, 两种存储方式差别并不大, 而数据量较大时, 本文设计的存储方式节省物理存储空间. 如表 6 所示, 当存储时间长度为 7 天的数据时, 本文设计的存储方式只比传统存储方法节省 3% 的物理存储空间, 当存储时间长度为一年的数据时, 本文设计的存储方式比传统存储方式节省近 10% 的物理存储空间.

表 6 不同数据量的存储模型占用空间 (单位: MB)

方法	7 天	15 天	30 天	90 天	365 天
传统方法	15.93	40.47	83.09	250.76	1209.84
本文	15.3	39.15	78.58	230.81	1095.79

两种存储方式的对比情况如图 7 所示.

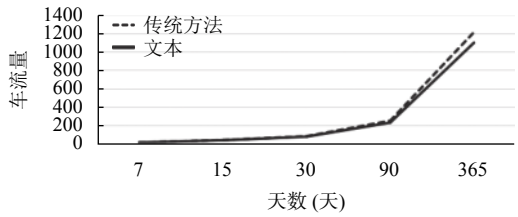

图 7 车流量 OD 矩阵实现流程

6 总结

高速公路经过长时间的发展, 已经形成了比较健全的路网结构, 信息化的建设已经逐步成型, 并覆盖到

高速公路的许多业务领域, 现有的高速公路运营系统对于海量数据的处理效率低下, 本文从大数据分析的角度实现了生成车辆 OD 矩阵的过程, 将生成的矩阵存入 HBase 中, 并对存储的效率和存储空间进行了分析, 以便于后续对生成的 OD 矩阵进行聚类等分析.

参考文献

1 赵明, 王寒凝. 基于车牌照识别技术的 OD 调查系统分析与设计. 公路交通科技: 应用技术版, 2008, (12): 188–190.

2 宋广辉, 刘淑珍, 耿英杰, 等. OD 调查简介. 东北公路, 1996, (2): 77–82.

3 陈满堂. 关于车牌号 OD 调查方法的探讨. 公路, 2004, (9): 86–88. [doi: 10.3969/j.issn.0451-0712.2004.09.020]

4 林勇, 蔡远利, 黄永宣. 高速公路动态 OD 矩阵估计. 长安大学学报 (自然科学版), 2003, 23(6): 83–86.

5 多雪松, 张晶, 高强. 基于 Hadoop 的海量数据管理系统. 微计算机信息, 2010, 26(5-1): 202–204.

6 龙伟, 陈志, 万亚平. 基于 Hadoop 的高速公路联网收费稽查系统. 西部交通科技, 2014, (8): 73–78.

7 Newman A, Li YF, Hunter J. Scalable semantics—The silver lining of cloud computing. Proceedings of the 2008 IEEE 4th International Conference on eScience. Indianapolis, IN, USA. 2008. 111–118.

8 王惟一. 基于存储模型的 HBase 查询优化技术研究[硕士学位论文]. 南京: 南京大学, 2018

9 徐德智, 刘扬, Ahmed S. 基于 Hadoop 的 RDF 数据存储及查询优化. 计算机应用研究, 2017, 34(2): 477–480, 486. [doi: 10.3969/j.issn.1001-3695.2017.02.035]