车流量项目问题

A1 关于数据倾斜

使用Hive ETL预处理数据

方案适用场景：

如果导致数据倾斜的是Hive表。如果该Hive表中的数据本身很不均匀（比如某个key对应了100万数据，其他key才对应了10条数据），而且业务场景需要频繁使用Spark对Hive表执行某个分析操作，那么比较适合使用这种技术方案。

方案实现思路：

此时可以评估一下，是否可以通过Hive来进行数据预处理（即通过Hive ETL预先对数据按照key进行聚合，或者是预先和其他表进行join），然后在Spark作业中针对的数据源就不是原来的Hive表了，而是预处理后的Hive表。此时由于数据已经预先进行过聚合或join操作了，那么在Spark作业中也就不需要使用原先的shuffle类算子执行这类操作了。

方案实现原理：

这种方案从根源上解决了数据倾斜，因为彻底避免了在Spark中执行shuffle类算子，那么肯定就不会有数据倾斜的问题了。但是这里也要提醒一下大家，这种方式属于治标不治本。因为毕竟数据本身就存在分布不均匀的问题，所以Hive ETL中进行group by或者join等shuffle操作时，还是会出现数据倾斜，导致Hive ETL的速度很慢。我们只是把数据倾斜的发生提前到了Hive ETL中，避免Spark程序发生数据倾斜而已。

过滤少数导致倾斜的key

方案适用场景：

如果发现导致倾斜的key就少数几个，而且对计算本身的影响并不大的话，那么很适合使用这种方案。比如99%的key就对应10条数据，但是只有一个key对应了100万数据，从而导致了数据倾斜。

方案实现思路：

如果我们判断那少数几个数据量特别多的key，对作业的执行和计算结果不是特别重要的话，那么干脆就直接过滤掉那少数几个key。比如，在Spark SQL中可以使用where子句过滤掉这些key或者在Spark Core中对RDD执行filter算子过滤掉这些key。如果需要每次作业执行时，动态判定哪些key的数据量最多然后再进行过滤，那么可以使用sample算子对RDD进行采样，然后计算出每个key的数量，取数据量最多的key过滤掉即可。

方案实现原理：

将导致数据倾斜的key给过滤掉之后，这些key就不会参与计算了，自然不可能产生数据倾斜。

提高shuffle操作的并行度

方案实现思路：

在对RDD执行shuffle算子时，给shuffle算子传入一个参数，比如reduceByKey(1000)，该参数就设置了这个shuffle算子执行时shuffle read task的数量。对于Spark SQL中的shuffle类语句，比如group by、join等，需要设置一个参数，即spark.sql.shuffle.partitions，该参数代表了shuffle read task的并行度，该值默认是200，对于很多场景来说都有点过小。

方案实现原理：

增加shuffle read task的数量，可以让原本分配给一个task的多个key分配给多个task，从而让每个task处理比原来更少的数据。举例来说，如果原本有5个不同的key，每个key对应10条数据，这5个key都是分配给一个task的，那么这个task就要处理50条数据。而增加了shuffle read task以后，每个task就分配到一个key，即每个task就处理10条数据，那么自然每个task的执行时间都会变短了。

缺点：不是万能的，没有从根本上改变数据的格式所以数据倾斜问题还是可能存在。

例如极端情况，有一个key对应数据就是100w，别的key对应1w，你再怎么增加task，相同的可以还是会分配到一个task中去计算。

不能根本解决问题，可以缓解一波。

双重聚合

方案适用场景：

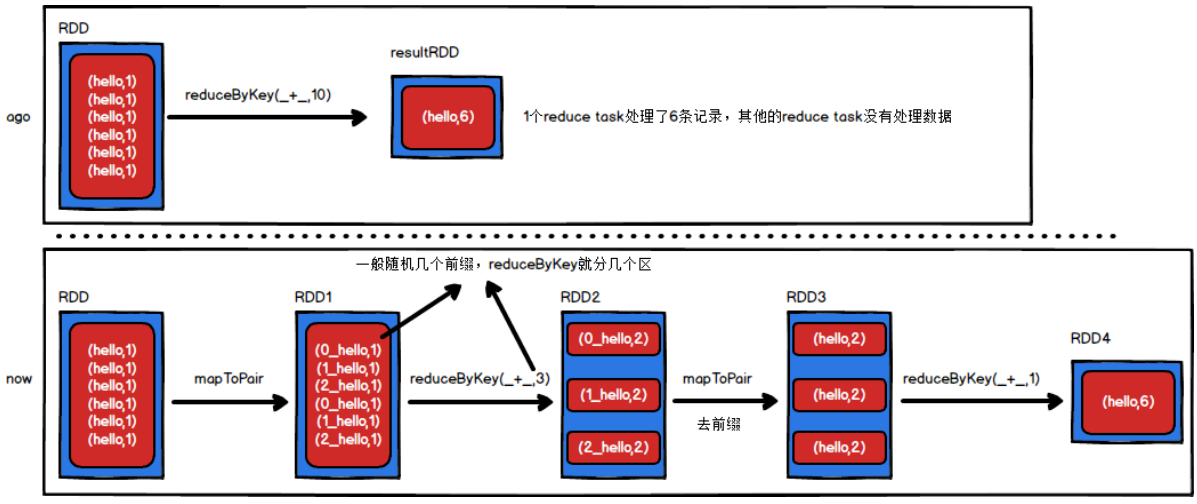
对RDD执行reduceByKey、groupByKey等聚合类shuffle算子或者在Spark SQL中使用group by语句进行分组聚合时，比较适用这种方案。

方案实现思路：

这个方案的核心实现思路就是进行两阶段聚合。第一次是局部聚合，先给每个key都打上一个随机数，比如10以内的随机数，此时原先一样的key就变成不一样的了，比如(hello, 1) (hello, 1) (hello, 1) (hello, 1)，就会变成(1_hello, 1) (1_hello, 1) (2_hello, 1) (2_hello, 1)。接着对打上随机数后的数据，执行reduceByKey等聚合操作，进行局部聚合，那么局部聚合结果，就会变成了(1_hello, 2) (2_hello, 2)。然后将各个key的前缀给去掉，就会变成(hello,2)(hello,2)，再次进行全局聚合操作，就可以得到最终结果了，比如(hello, 4)。

方案实现原理：

将原本相同的key通过附加随机前缀的方式，变成多个不同的key，就可以让原本被一个task处理的数据分散到多个task上去做局部聚合，进而解决单个task处理数据量过多的问题。接着去除掉随机前缀，再次进行全局聚合，就可以得到最终的结果。



如果一个RDD中有一个key导致数据倾斜，同时还有其他的key，那么一般先对数据集进行抽样，然后找出倾斜的key,再使用filter对原始的RDD进行分离为两个RDD，一个是由倾斜的key组成的RDD1，一个是由其他的key组成的RDD2，那么对于RDD1可以使用加随机前缀进行多分区多task计算，对于另一个RDD2正常聚合计算，最后将结果再合并起来。

只适用于聚合类的shuffle操作，join这类不太适合。

将reduce join转为map join

BroadCast+filter(或者map)

方案适用场景：

在对RDD使用join类操作，或者是在Spark SQL中使用join语句时，而且join操作中的一个RDD或表的数据量比较小（比如几百M或者一两G），比较适用此方案。

方案实现思路：

不使用join算子进行连接操作，而使用Broadcast变量与map类算子实现join操作，进而完全规避掉shuffle类的操作，彻底避免数据倾斜的发生和出现。将较小RDD中的数据直接通过collect算子拉取到Driver端的内存中来，然后对其创建一个Broadcast变量；接着对另外一个RDD执行map类算子，在算子函数内，从Broadcast变量中获取较小RDD的全量数据，与当前RDD的每一条数据按照连接key进行比对，如果连接key相同的话，那么就将两个RDD的数据用你需要的方式连接起来。

方案实现原理：

普通的join是会走shuffle过程的，而一旦shuffle，就相当于会将相同key的数据拉取到一个shuffle read task中再进行join，此时就是reduce join。但是如果一个RDD是比较小的，则可以采用广播小RDD全量数据+map算子来实现与join同样的效果，也就是map join，此时就不会发生shuffle操作，也就不会发生数据倾斜。

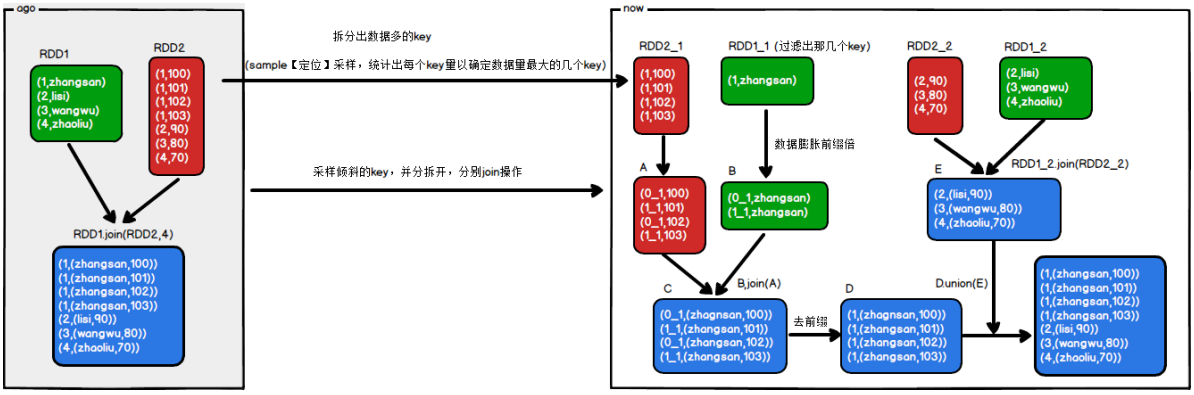
采样倾斜key并分拆join操作

方案适用场景：

两个RDD/Hive表进行join的时候，如果数据量都比较大，无法采用“解决方案五”，那么此时可以看一下两个RDD/Hive表中的key分布情况。如果出现数据倾斜，是因为其中某一个RDD/Hive表中的少数几个key的数据量过大，而另一个RDD/Hive表中的所有key都分布比较均匀，那么采用这个解决方案是比较合适的。

方案实现思路：

对包含少数几个数据量过大的key的那个RDD，通过sample算子采样出一份样本来，然后统计一下每个key的数量，计算出来数据量最大的是哪几个key。然后将这几个key对应的数据从原来的RDD中拆分出来，形成一个单独的RDD，并给每个key都打上n以内的随机数作为前缀，而不会导致倾斜的大部分key形成另外一个RDD。接着将需要join的另一个RDD，也过滤出来那几个倾斜key对应的数据并形成一个单独的RDD，将每条数据膨胀成n条数据，这n条数据都按顺序附加一个0~n的前缀，不会导致倾斜的大部分key也形成另外一个RDD。再将附加了随机前缀的独立RDD与另一个膨胀n倍的独立RDD进行join，此时就可以将原先相同的key打散成n份，分散到多个task中去进行join了。而另外两个普通的RDD就照常join即可。最后将两次join的结果使用union算子合并起来即可，就是最终的join结果 。



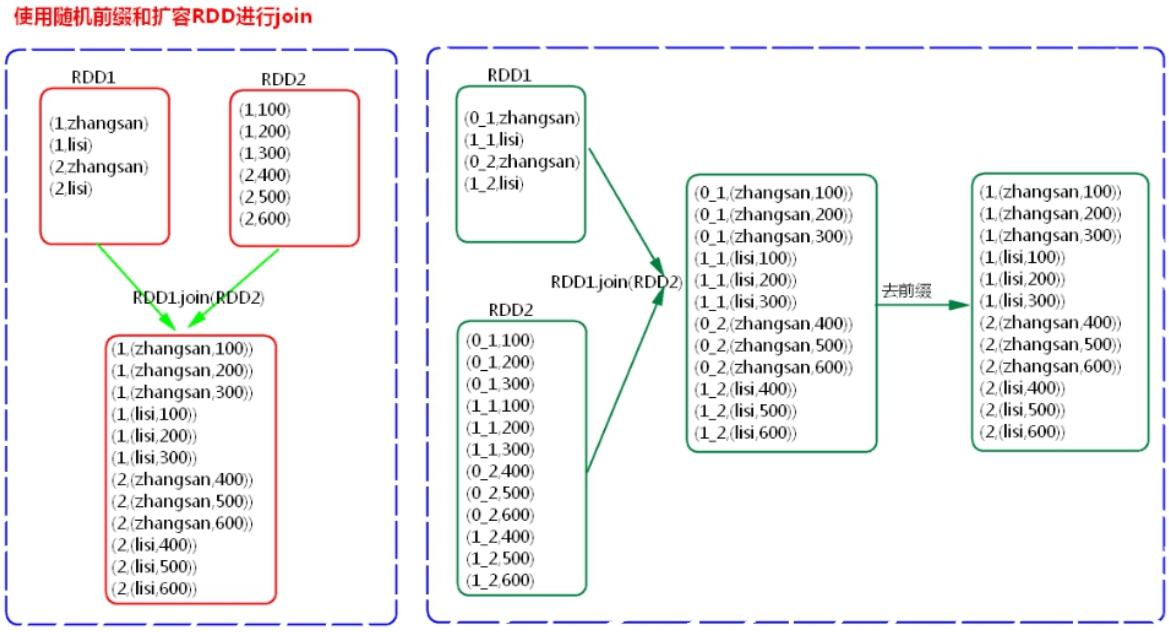
使用随机前缀和扩容RDD进行join

方案适用场景：

如果在进行join操作时，RDD中有大量的key导致数据倾斜，那么进行分拆key也没什么意义，此时就只能使用最后一种方案来解决问题了。

方案实现思路：

该方案的实现思路基本和“解决方案六”类似，首先查看RDD/Hive表中的数据分布情况，找到那个造成数据倾斜的RDD/Hive表，比如有多个key都对应了超过1万条数据。然后将该RDD的每条数据都打上一个n以内的随机前缀。同时对另外一个正常的RDD进行扩容，将每条数据都扩容成n条数据，扩容出来的每条数据都依次打上一个0~n的前缀。最后将两个处理后的RDD进行join即可。



前面的join,可以并行处理数据的task最多为2，后面优化后，join可以并行处理数据的task最多为3

rdd中数据倾斜，sample抽样取出样本(key)，统计key对应的数量，以及最大的数据量的key。

把这些key从rdd中抽取出去，形成单独的rdd，给key打上随机数作为前缀，另一部分没有抽出去的数据在形成另外一个rdd。

另一个rdd（join），也将上次抽取出来的key过滤出一个单独的rdd，但是需要做扩容（数据量的膨胀）。在这些数据上追加上随机数，和之前的rdd加随机数的规则一致，没有抽离的数据也还是形成一个新的rdd。

两两join，有随机数的记得去掉前缀，然后最终结果union得出。

A2 StringBuilder

```
StringBuilder tmpInfos = new StringBuilder();//同一个monitorId下，对应的所有的不同的camearId信息

int count = 0;//统计车辆数的count                    就创建一个对象，如果是string+东西会再创建一个对象
String areaId = "";
/**
 * 这个while循环  代表的是当前的这个卡扣一共经过了多少辆车，     一辆车的信息就是一个row
```

A3 自定义累加器

6步骤

isZero

copy

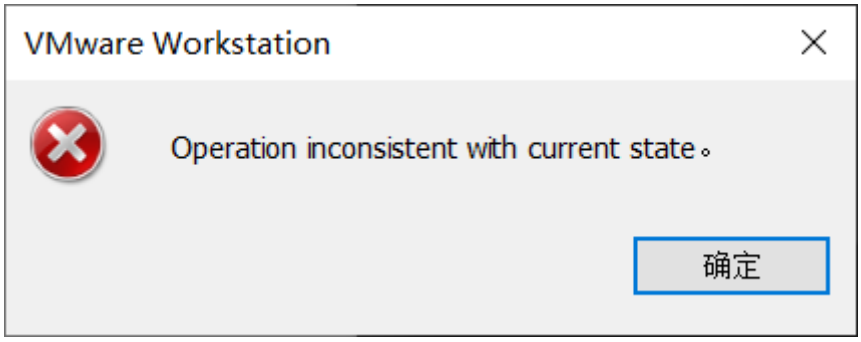
reset

add

merge

value

A4 虚拟机打开出错



解决：关掉重开，重启

A5 解决mysql中文乱码

```
show variables like 'char%';
```

```
mysql> show variables like 'char%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | latin1                     |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | latin1                     |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)
```

```
mysql> show variables like 'char%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | utf8                       |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | utf8                       |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)
```

这么改不一定生效，需要改配置文件：

`vim /etc/my.cnf`

```
[mysqld]
#
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
```

```
character-set-server=utf8
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
[mysql]
no-auto-rehash
default-character-set=utf8

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

A6 遇到问题

```
java.lang.NumberFormatException Create breakpoint : multiple points
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1890)
    at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
    at java.lang.Double.parseDouble(Double.java:538)
```

百度

蓝的信息（自己写的）

操作

A7 kafka参数和机制总结

https://blog.csdn.net/weixin_34205076/article/details/92488763

A8 集群分配

NN自己一个人玩

7亿条数据，日500GB

A9 项目总梳理

9.1 数据

> 数据从哪里来，到哪里去
>
> 源，落地

描述项目架构：lambda架构，离线实时数据同源

任务触发流程：

9.2 业务流程

1. 卡口状态监控：自定义累加器//卡口流量topN//高速卡口topN，并求出卡口对应车速前十的车辆信息

- 自定义累加器（6步骤）
- 高速卡口：广播变量+过滤

2. 指定卡口车辆的行车轨迹
3. 随机抽取车辆，求行车轨迹（Map<Date,Map<Hour,Llist(Index)>>）
4. 道路转换率
5. 区域道路流量top

- sparkSql相关业务
    - UDF：随机数+双重聚合解决数据倾斜问题
    - UDAF

6. 道路流量实时情况

- 平均速度

7. 布控（动态改变广播变量）