SparkSQL join探究

上海直真君智科技有限公司

徐浩

2017.12

目录

1.引言

直真君智
ZHI ZHEN JUN ZHI

- 性能是客户重点关注的内容
  - 在于客户的沟通中发现他们对性能特别的重视
  - 在项博的指导下，总分析流的时间由30s左右，降到了10s，单个分支降到了4s
  - 在诸多业务场景中，涉及到shuffle操作最多的是join和groupby，也是优化的重点
  - 我搜了一些关于join资料，结合实践，进行了整理和总结，在此分享给大家

1.引言

直真君智
ZHI ZHEN JUN ZHI

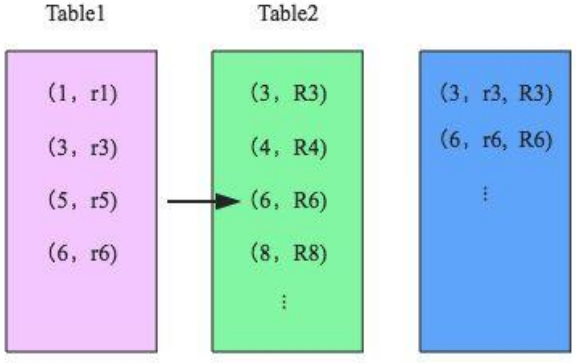
抛砖

- 遗憾的是我在公司集群上尝试复现一些场景，尝试了很久没有成功，所以以下有部分理论还没有经过实践验证。
- 希望后面有同事碰到类似情况可以一起交流和分享；或者等后续，有时间实践一下，再做一次更系统的分享

目录

2.SQL join算法

- 什么是join操作



这里特指inner join

4种常用jion操作

1, null	(r1， null)
3, 3	(r3， R3)
null, 4	(null， R4)
5, null	(r5， null)
6, 6	(r6， R6)
null, 8	(null， R8)

1, null	(r1， null)
3, 3	(r3， R3)
5, null	(r5， null)
6, 6	(r6， R6)

3, 3	(r3， R3)
null, 4	(null， R4)
6, 6	(r6， R6)
null, 8	(null， R8)

2.SQL join算法

- 一般SQL的join算法

方法1：nest loop

Step1，遍历 B中所有的记录，对每条记录遍历A寻找key匹配的记录

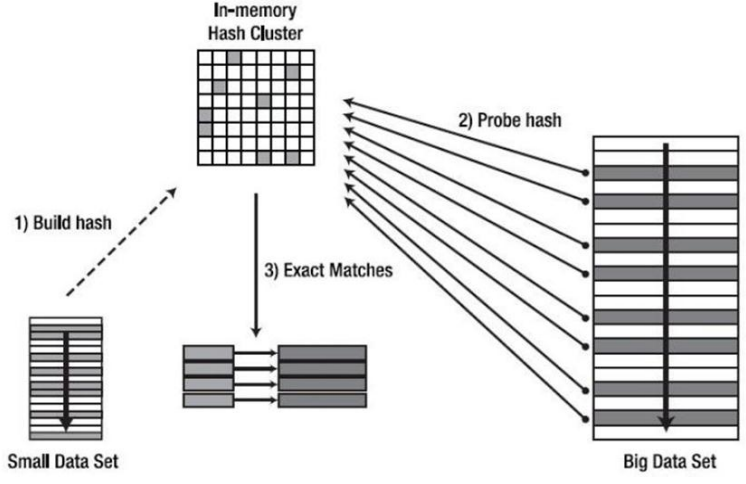
时间复杂度：O(m*n)

方法2：hash join

Step1，找到较小的表A，将其key进行hash

Step2，遍历B，在hash表中找到与B.key对应的记录

时间复杂度：O(m) + n*O(1) = O(m + n)



2.SQL join算法

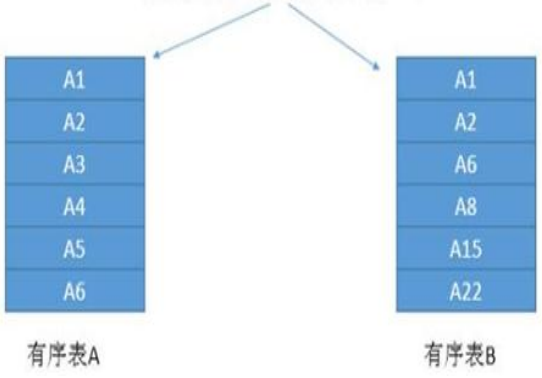
- 一般SQL的join算法

方法3：merge join

Step1，A排序，B排序

Step2，两个指针分别从两表开头比对对应值得大小，如果小，对应指针下移。如果相等输出该条记录。

时间复杂度：O(m*log(m) + n*log(n))



2.SQL join算法

- 上述三种join算法的比较

Nest loop

优点：　join规则可以任意指定；
　　表非常小，且join的key建有索引时较快

缺点：　不适用于大数据
　　不适用于key无索引的join类型。

Hash join

优点：　三种算法中时间复杂度最低
　　适用于大数据集

缺点：　只能做等值连接
　　hash过程不适用于大表的情况，内存会溢出，产生磁盘IO操作

Merge join

优点：　适用于大数据集
　　可以做非等值的连接

缺点：　理论上要比hash join慢
　　最后返回数据

目录

3.SparkSQL join算法

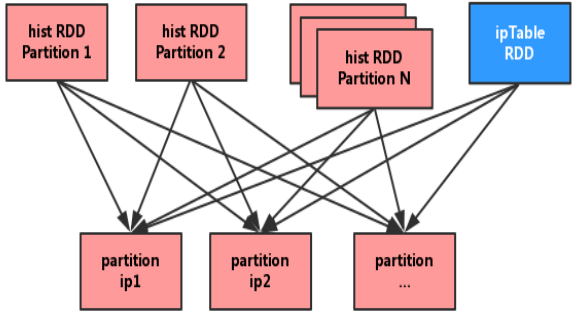
- SparkSQL的join算法

方法1：shuffle hash join

Step1，shuffle阶段：分别将两个表按照join key进行分区，将相同join key的记录重分布到同一节点，两张表的数据会被重分布到集群中所有节点。这个过程称为shuffle。

Step2，hash join阶段：每个分区节点上的数据单独执行hash join算法。

条件：build端的小表的平均分区小于spark.sql.autoBroadcastJoinThreshold。



shuffle join

3.SparkSQL join算法

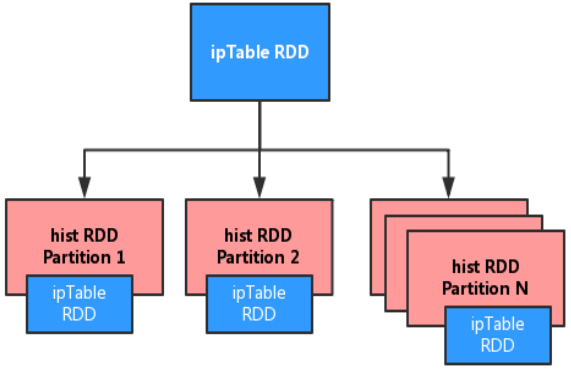
- SparkSQL的join算法

方法2：broadcast hash join

Step1，broadcast阶段：将小表广播分发到大表所在的所有主机。广播算法可以有很多，最简单的是先发给driver，driver再统一分发给所有executor。

Step2，hash join阶段：在每个executor上执行单机版hash join，小表hash，大表probe。

条件：广播小表必须小于spark.sql.autoBroadcastJoinThreshold。



broadcast join

3.SparkSQL join算法

- SparkSQL的join算法

  方法3：sort merge join

  Step1，shuffle阶段：将两张大表根据join key进行重新分区，两张表数据会分布到整个集群，以便分布式并行处理。

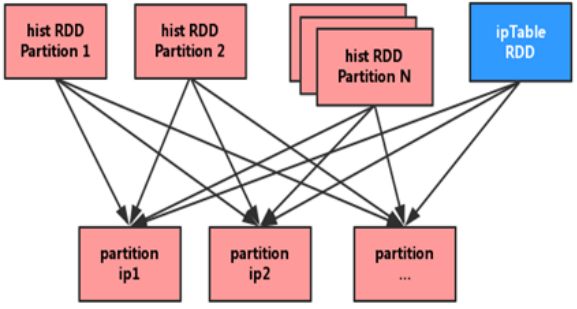
  Step2，在每个分区节点上进行sort merge join

  sort阶段：对单个分区节点的两表数据，分别进行排序。

  merge阶段：对单个分区的两张表单独执行merge join操作。

  特点：

  无论分区有多大，Sort Merge Join都不用把某一侧的数据全部加载到内存中，而是即用即取即丢，从而大大提升了大数据量下sql join的稳定性。



目录

4.实践和总结

直真君智 ZHI ZHEN JUN ZHI

- 1. 由上述内容可以看到，对于同时适用于上述三种join的情形，broadcast join > shuffle hash join > sort merge join，所以在可能的情况下我们优先选择broadcast join。

bigTable.join(smallTable)或bigTable.join(broadcast(smallTable))

- 2. spark内置了一些优化规则(RBO)。

当两个小表进行join时可以不涉及shuffle直接通过hash join实现，当一个大表join一个小表时可以判断小表是否小于spark.sql.autoBroadcastJoinThreshold来将其广播。当小表超过阈值时，spark会进行shuffle hash join，前提是小表的平均分区小于参数spark.sql.autoBroadcastJoinThreshold。当不满足上述条件或者是两个大表join时，采用sort merge join。

- 3. 我们可以通过调节spark.sql.autoBroadcastJoinThreshold来实现小表（最好200M以内）的broadcast join。

4.实践和总结

- 4.在进行inner join时尽量使用大表join小表，而不是小表join大表。
- 5.有时候避开shuffle（join）也许是个更好的选择。

1）如一个大表join极小表（数千条记录以内）的时候可以尝试map side join，即将join转化为map+filter：

```
// 方案1
val newDataFrame = bigDF.join(broadcast(smallDF))

// 方案2
val smallArr = smallDF.collect().map(r=>r(0).asInstanceOf[Int])
val src_index = bigDF.schema.fieldIndex("srcaddr")
val newDataFrame = sqlc.createDataFrame(
    bigDF.rdd.
    filter(r => smallArr.contains(r(src_index))),bigDF.schema)
```

这里smallArr稍大时也可以通过sc.Broadcast将其广播。

4.实践和总结

- 2）有时候优化一下算法流程也能避免join操作

key	value
1	aa
1	bb
1	aa
2	bb

key	count	value
1	3	aa
1	3	bb
1	3	aa
2	1	bb

```
// 方案1
val countTable = table.grouBy("key").count
val newTable = table.join(countTable, "left")

// 方案2
val table_rdd = table.rdd.map(row =>
  {
    key = row.getInt(0)
    value = row.getString(1)
    (key, (Array(value), 1))
  })
val new_rdd = table_rdd.reduceByKey{
      case ((arr1, count1), (arr2, count2)) =>
      (arr1 ++ arr2, count1 + count2)
    }.flatMapValues{
    case (arr, count) => arr.map((count, _))
    }
```

4.实践和总结

直真君智 ZHI ZHEN JUN ZHI

- 6.有时，通过pair rdd替代join有时也是一个不错的选择。即：

```
val tableA_rdd = tableA.rdd.map(row => (row.getString(0), 1))
val key_index = tableB.schema.fieldIndex("srcaddr")
val tableB_rdd = tableB.rdd.map(x => (x.getString(key_index), x))
val new_rdd = tableB_rdd.join(tableA_rdd).mapValues(_._1)
val newDataFrame = sqlc.createDataFrame(new_rdd.values, ipsession.schema)
```

- 7.数据倾斜也是join的一大障碍，当数据倾斜发生时可以尝试：

  1）filter大的key单独处理；

  2）拆分大表中大的key，同时小表对应膨胀，再join；

  3）调大spark.sql.shuffle.partitions。

4.实践和总结

直真君智 ZHI ZHEN JUN ZHI

- 8.当两个表join时，事先的分区数会影响join的效率，也会影响broadcast过程。

以在数据源读入的过程中慎用repartion，采用的是均匀分区，shuffle时又会重新按key分区，增加了时耗。除非后面绝大多数应用是基于map类型的，尽量不要repartition。

- 9.SparkSQL中的非等值join可以利用。

情景：tableB表有key，有记录时间。tableA表有key，有起止时间。规定匹配规则，tableB的key和tableA的key对应，且tableB的记录时间在tableA对应起止时间内，认为两条记录对应。两表一大一小，大表13G，小表5G。

```
// 方案1
tableB.join(tableA,tableA("key") === tableB("key"))
    .filter("tableB_time >= tableA_startTime and tableB_time <= tableA_startTime")
// 方案2
tableB.join(tableA,tableB("key") === tableA("key")
    && signal("tableB_time")
    .between(tableA("tableA_startTime"),tableA("tableA_startTime"))
```

方案1对应shuffle hash join，方案2对应sort merge join。对于大的数据，在实践上感觉方案2更稳健一些。

4.实践和总结

- 10.兵无常势，水无常形，能因敌变化而制胜者，谓之神也。

数据按key的分布不同、分区不同、集群配置不同，往往策略不同。最好的方式是：察看数据结构，选定几种方案，都尝试一下，观察哪种最好。

谢 谢！

2017年