SparkSQL大数据实战：揭开Join的神秘面纱

2017-04-05 10:00:49

Join操作是数据库和大数据计算中的高级特性，大多数场景都需要进行复杂的Join操作，本文从原理层面介绍了SparkSQL支持的常见Join算法及其适用场景。

Join背景介绍

Join是数据库查询永远绕不开的话题，传统查询SQL技术总体可以分为简单操作（过滤操作-where、排序操作-limit等），聚合操作-groupby以及Join操作等。其中Join操作是最复杂、代价最大的操作类型，也是OLAP场景中使用相对较多的操作。因此很有必要对其进行深入研究。

另外，从业务层面来讲，用户在数仓建设的时候也会涉及Join使用的问题。通常情况下，数据仓库中的表一般会分为“低层次表”和“高层次表”。

所谓“低层次表”，就是数据源导入数仓之后直接生成的表，单表列值较少，一般可以明显归为维度表或事实表，表和表之间大多存在外健依赖，所以查询起来会遇到大量Join运算，查询效率很差。而“高层次表”是在“低层次表”的基础上加工转换而来，通常做法是使用SQL语句将需要Join的表预先进行合并形成“宽表”，在宽表上的查询不需要执行大量Join，效率很高。但宽表缺点是数据会有大量冗余，且相对生成较滞后，查询结果可能并不及时。

为了获得时效性更高的查询结果，大多数场景都需要进行复杂的Join操作。Join操作之所以复杂，主要是通常情况下其时间空间复杂度高，且有很多算法，在不同场景下需要选择特定算法才能获得最好的优化效果。本文将介绍SparkSQL所支持的几种常见的Join算法及其适用场景。

Join常见分类以及基本实现机制

当前SparkSQL支持三种Join算法：shuffle hash join、broadcast hash join以及sort merge join。其中前两者归根到底都属于hash join，只不过在hash join之前需要先shuffle还是先broadcast。其实，hash join算法来自于传统数据库，而shuffle和broadcast是大数据的皮（分布式），两者一结合就成了大数据的算法了。因此可以说，大数据的根就是传统数据库。既然hash join是“内核”，那就刨出来看看，看完把“皮”再分析一下。

hash join

先来看看这样一条SQL语句：select * from order,item where item.id = order.i_id，很简单一个Join节点，参与join的两张表是item和order，join key分别是item.id以及order.i_id。现在假设这个Join采用的是hash join算法，整个过程会经历三步：

1. 确定Build Table以及Probe Table：这个概念比较重要，Build Table使用join key构建Hash Table，而Probe Table使用join key进行探测，探测成功就可以join在一起。通常情况下，小表会作为Build Table，大表作为Probe Table。此事例中item为Build Table，order为Probe Table。
2. 构建Hash Table：依次读取Build Table（item）的数据，对于每一行数据根据join key（item.id）进行hash，hash到对应的Bucket，生成hash table中的一条记录。数据缓存在内存中，如果内存放不下需要dump到外存。
3. 探测：再依次扫描Probe Table（order）的数据，使用相同的hash函数映射Hash Table中的记录，映射成功之后再检查join条件（item.id = order.i_id），如果匹配成功就可以将两者join在一起。

热门标签

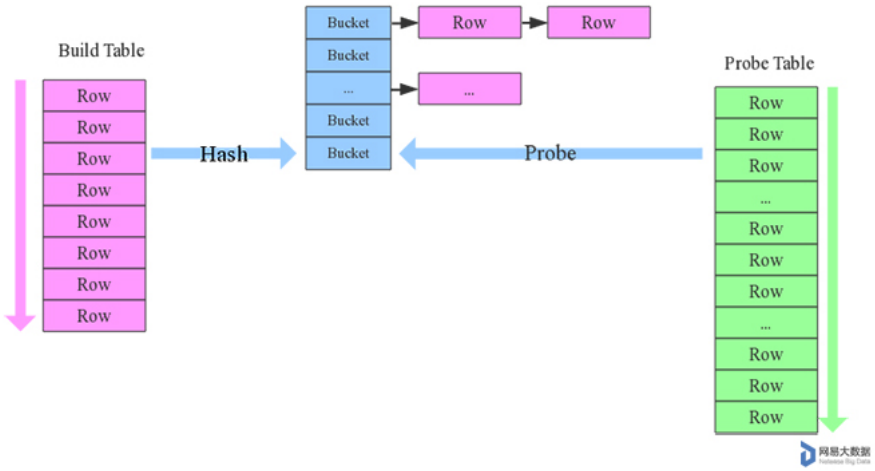
网易云信 即时通讯

版本更新 PaaS

云客服 网易易盾

热门推荐

1 致力解决客户问题，网易云轻
2 挖财、宝宝树、美食杰等十大
3 十年•杭研技术秀 | 构建高可用
4 十年•杭研技术秀 | 网易蜂巢的
5 十年·杭研技术秀 | Hadoop数
6 十年•杭研大咖说 | 尧飘海：
7 十年•杭研大咖说 | 陈谔：为



基本流程可以参考上图，这里有两个小问题需要关注：

1. hash join性能如何？很显然，hash join基本都只扫描两表一次，可以认为o(a+b)，较之最极端的笛卡尔集运算a*b，不知甩了多少条街。
2. 为什么Build Table选择小表？道理很简单，因为构建的Hash Table最好能全部加载在内存，效率最高；这也决定了hash join算法只适合至少一个小表的join场景，对于两个大表的join场景并不适用。

上文说过，hash join是传统数据库中的单机join算法，在分布式环境下需要经过一定的分布式改造，就是尽可能利用分布式计算资源进行并行化计算，提高总体效率。hash join分布式改造一般有两种经典方案：

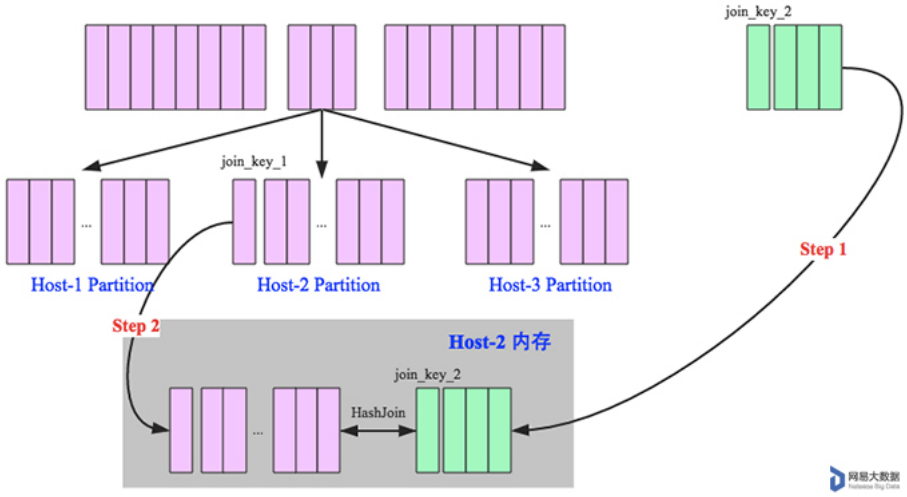
1. broadcast hash join：将其中一张小表广播分发到另一张大表所在的分区节点上，分别并发地与其上的分区记录进行hash join。broadcast适用于小表很小，可以直接广播的场景。
2. shuffler hash join：一旦小表数据量较大，此时就不再适合进行广播分发。这种情况下，可以根据join key相同必然分区相同的原理，将两张表分别按照join key进行重新组织分区，这样就可以将join分而治之，划分为很多小join，充分利用集群资源并行化。

下面分别进行详细讲解。

broadcast hash join

如下图所示，broadcast hash join可以分为两步：

1. 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；要不就是基于BitTorrent的TorrentBroadcast。
2. hash join阶段：在每个executor上执行单机版hash join，小表映射，大表试探。

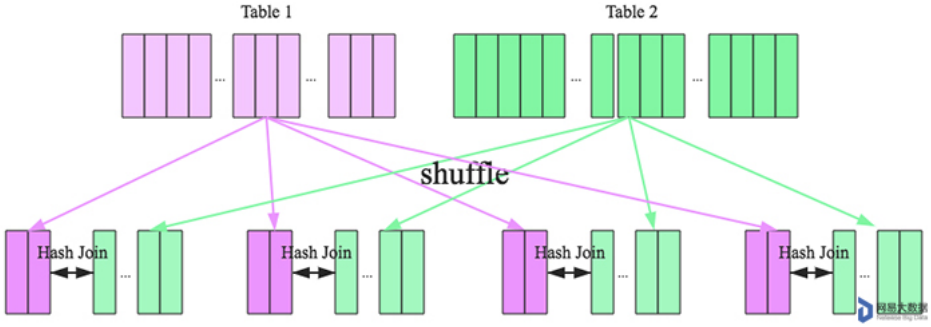


3. SparkSQL规定broadcast hash join执行的基本条件为被广播小表必须小于参数spark.sql.autoBroadcastJoinThreshold，默认为10M。

shuffle hash join

在大数据条件下如果一张表很小，执行join操作最优的选择无疑是broadcast hash join，效率最高。但是一旦小表数据量增大，广播所需内存、带宽等资源必然就会太大，broadcast hash join就不再是最优方案。此时可以按照join key进行分区，根据key相同必然分区相

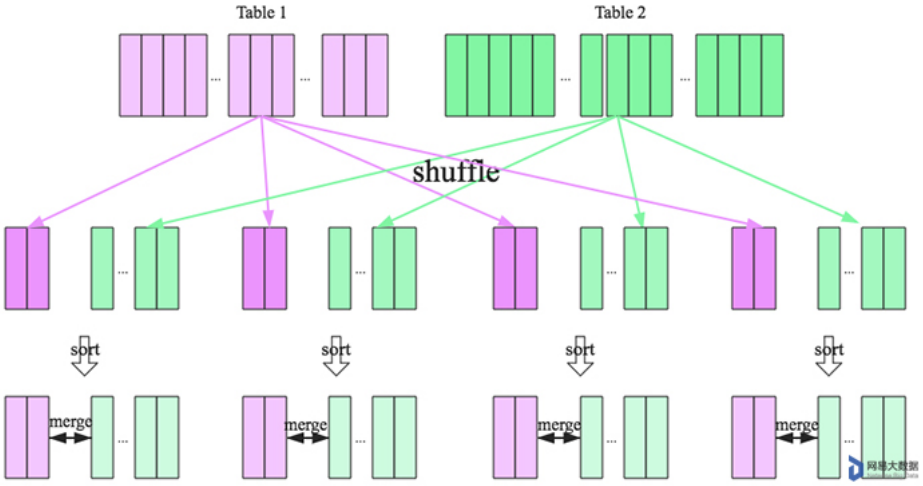
1. shuffle阶段：分别将两个表按照join key进行分区，将相同join key的记录重分布到同一节点，两张表的数据会被重分布到集群中所有节点。这个过程称为shuffle。
2. hash join阶段：每个分区节点上的数据单独执行单机hash join算法。



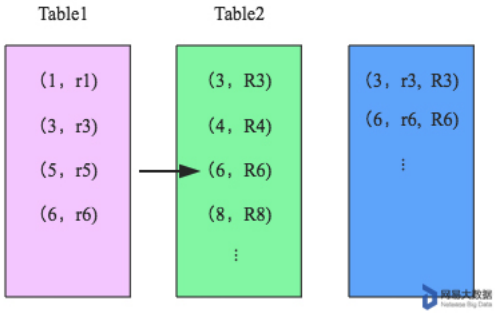
看到这里，可以初步总结出来如果两张小表join可以直接使用单机版hash join；如果一张大表join一张极小表，可以选择broadcast hash join算法；而如果是一张大表join一张小表，则可以选择shuffle hash join算法；那如果是两张大表进行join呢？

sort merge join

SparkSQL对两张大表join采用了全新的算法 - sort-merge join，如下图所示，整个过程分为三个步骤：



1. 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。
2. sort阶段：对单个分区节点的两表数据，分别进行排序。
3. merge阶段：对排好序的两张分区表数据执行join操作。join操作很简单，分别遍历两个有序序列，碰到相同join key就merge输出，否则取更小一边。如下图所示：



经过上文的分析，很明显可以得出来这几种Join的代价关系：cost(broadcast hash join) < cost(shuffle hash join) < cost(sort merge join)，数据仓库设计时最好避免大表与大表的join查询，SparkSQL也可以根据内存资源、带宽资源适量将参数spark.sql.autoBroadcastJoinThreshold调大，让更多join实际执行为broadcast hash join。

总结

Join操作是数据库和大数据计算中的高级特性，因为其独特的复杂性，很少有同学能够讲清楚其中的原理。本文试图带大家真正走进Join的世界，了解常用的几种Join算法以及各自的适用场景。后面两篇文章将会在此基础上不断深入Join内部，一点一点地揭开它的面纱，敬请关注！

分享到：

上一篇：Cloud Native实践从0到1

下一篇：45天架构变迁实战，网易美学平滑微服务化靠什么？

评论 (0)

评论文章前您需要先 登录

发表评论