> 本文已经加入「大数据成神之路PDF版」中提供下载。
> 你可以关注公众号，后台回复： **「PDF」** 即可获取。

更多PDF下载可以参考： **《重磅,大数据成神之路PDF可以分类下载啦!》**

Spark重点难点系列：

- **《【Spark重点难点01】你从未深入理解的RDD和关键角色》**
- **《【Spark重点难点02】你以为的Shuffle和真正的Shuffle》**
- **《【Spark重点难点03】你的数据存在哪了?》**
- **《【Spark重点难点04】你的代码跑起来谁说了算?(内存管理)》**
- **《【Spark重点难点05】SparkSQL YYDS(上)！》**
- **《【Spark重点难点06】SparkSQL YYDS(中)！》**
- **《【Spark重点难点07】SparkSQL YYDS(加餐)！》**

前言

Spark3.0版本的发布已经很长时间了，3.0版本增加了很多令人兴奋的新特性。

包括动态分区剪裁(Dynamic Partition Pruning)、自适应查询执行(Adaptive Query Execution)、加速器感知调度(Accelerator-aware Scheduling)、支持 Catalog 的数据源API（Data Source API with Catalog Supports）、SparkR 中的向量化（Vectorization in SparkR）、支持 Hadoop 3/JDK 11/Scala 2.12 等等。

这里面最重要的两个特性分别是：

- AQE(Adaptive Query Execution,自适应查询执行)
- DPP(Dynamic Partition Pruning,动态分区剪裁)

我们分别就分别就这两个特性进行一下讲解。

AQE(Adaptive Query Execution,自适应查询执行)

AQE是Spark SQL的一种动态优化机制，是对查询执行计划的优化。

我们可以设置参数 `spark.sql.adaptive.enabled` 为true来开启AQE，在Spark 3.0中默认是false。

在运行时，AQE会结合Shuffle Map阶段执行完毕后的统计信息，基于既定的规则动态地调整、修正尚未执行的逻辑计划和物理计划，来完成对原始查询语句的运行时优化。

在介绍AQE之前我们先讲解两个优化策略：

- RBO(Rule Based Optimization，基于规则的优化)。它往往基于一些规则和策略实现，如谓词下推、列剪枝，这些规则和策略来源于数据库领域已有的应用经验。也就是说，启发式的优化实际上算是一种「经验主义」。

- CBO(Cost Based Optimization，基于成本的优化)。CBO是一种基于数据统计信息例如数据量、数据分布来选择代价最小的优化策略的方式。

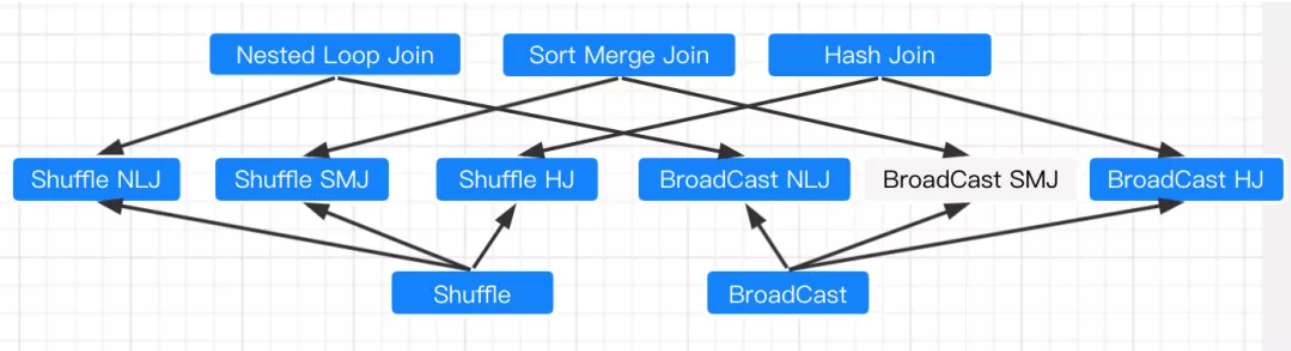
RBO相对于CBO而言要成熟得多，常用的规则都基于经验制定，可以覆盖大部分查询场景，并且方便扩展。其缺点则是不够灵活，对待相似的问题和场景都使用同一类解决方案，忽略了数据本身的信息。

Spark在2.2版本中推出了CBO，主要就是为了解决RBO 「经验主义」 的弊端。

AQE的三大特性包括： Join策略调整 、 分区自动合并 、 自动倾斜处理 。

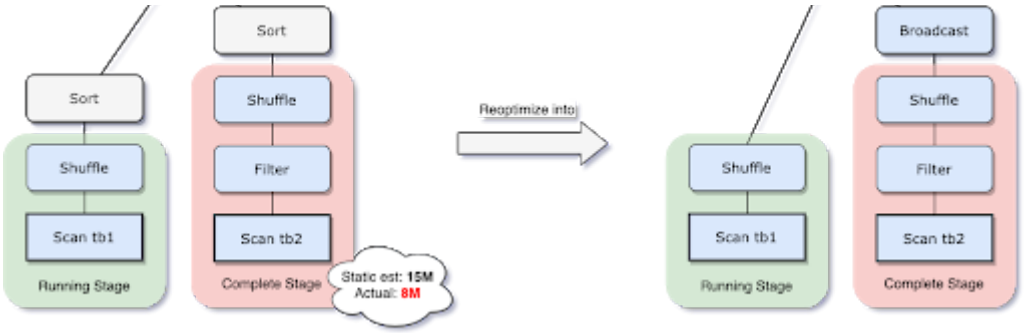
Join策略调整

关于Spark支持的Join策略，我们在之前的文章中做过详细介绍了：



Spark 支持的许多 Join 策略中，Broadcast Hash Join通常是性能最好的，前提是参加 join 的一张表的数据能够装入内存。由于这个原因，当 Spark 估计参加 join 的表数据量小于广播大小的阈值时，其会将 Join 策略调整为 Broadcast Hash Join。但是，很多情况都可能导致这种大小估计出错——例如存在一个非常有选择性的过滤器。

由于AQE可以精确的统计上游数据，因此可以解决该问题。比如下面这个例子，右表的实际大小为15M，而在该场景下，经过filter过滤后，实际参与join的数据大小为8M，小于了默认broadcast阈值10M，应该被广播。



在我们执行过程中转化为BHJ的同时，我们甚至可以将传统shuffle优化为本地shuffle(例如shuffle读在mapper而不是基于reducer)来减小网络开销。

分区自动合并

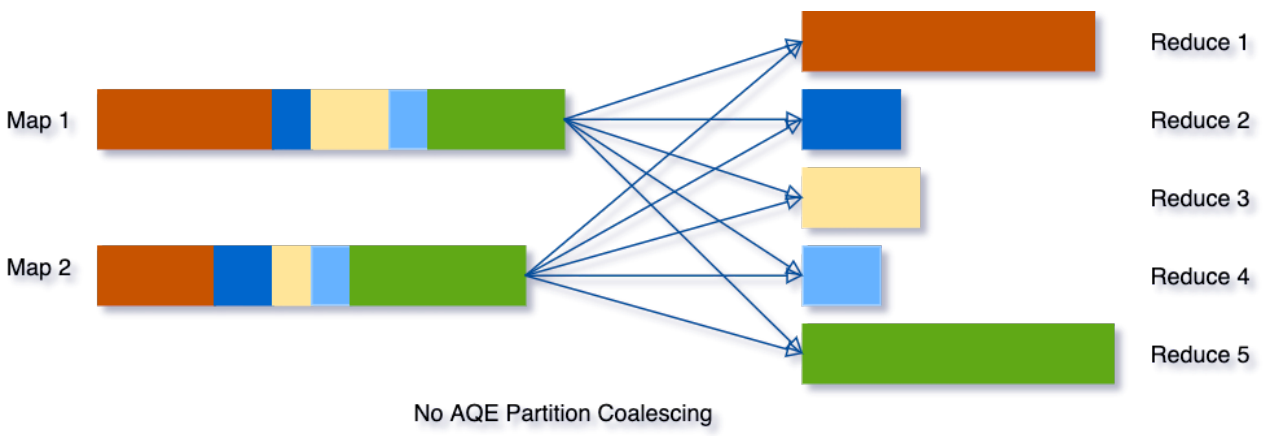
在我们处理的数据量级非常大时，shuffle通常来说是最影响性能的。因为shuffle是一个非常耗时的算子，它需要通过网络移动数据，分发给下游算子。

在shuffle中，partition的数量十分关键。partition的最佳数量取决于数据，而数据大小在不同的query不同stage都会有很大的差异，所以很难去确定一个具体的数目。

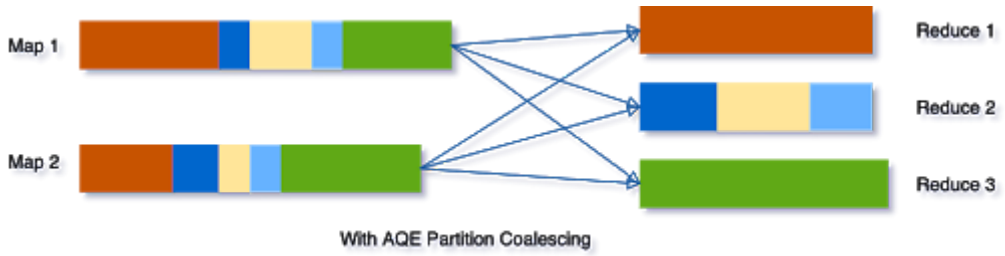
在这部分，有两个非常重要的参数用来控制目标分区的大小：

- `spark.sql.adaptive.advisoryPartitionSizeInBytes`，分区合并后的推荐尺寸
- `spark.sql.adaptive.coalescePartitions.minPartitionNum`，分区合并后最小分区数

为了解决该问题，我们在最开始设置相对较大的shuffle partition个数，通过执行过程中shuffle文件的数据来合并相邻的小partitions。 例如，假设我们执行 `SELECT max(i) FROM table GROUP BY j`，表table只有2个partition并且数据量非常小。我们将初始shuffle partition设为5，因此在分组后会出现5个partitions。若不进行AQE优化，会产生5个tasks来做聚合结果，事实上有3个partitions数据量是非常小的。



这种情况下，AQE生效后只会生成3个reduce task。

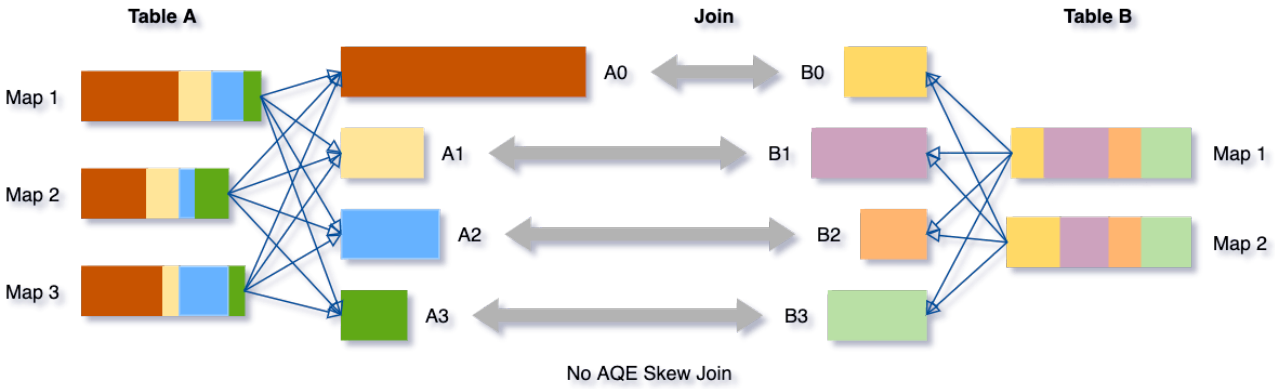


自动倾斜处理

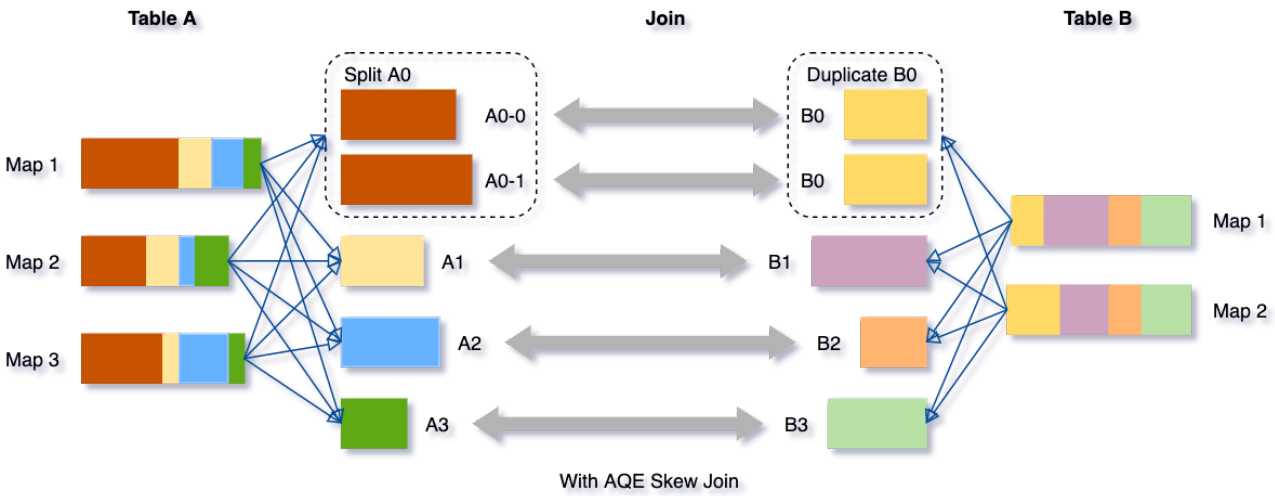
Spark Join操作如果出现某个key的数据倾斜问题，那么基本上就是这个任务的性能杀手了。在AQE之前，用户没法自动处理Join中遇到的这个棘手问题，需要借助外部手动收集数据统计信息，并做额外的加盐，分批处理数据等相对繁琐的方法来应对数据倾斜问题。

AQE根据shuffle文件统计数据自动检测倾斜数据，将那些倾斜的分区打散成小的子分区，然后各自进行join。

我们可以看下这个场景，Table A join Table B，其中Table A的partition A0数据远大于其他分区。



AQE会将partition A0切分成2个子分区，并且让他们独自和Table B的partition B0进行join。



如果不做这个优化，SMJ将会产生4个tasks并且其中一个执行时间远大于其他。经优化，这个join将会有5个tasks，但每个task执行耗时差不多相同，因此个整个查询带来了更好的性能。

关于如何定位这些倾斜的分区，主要靠下面三个参数：

- `spark.sql.adaptive.skewJoin.skewedPartitionFactor` ，判定倾斜的倾斜因子
- `spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes` ，判定倾斜的最低阈值
- `spark.sql.adaptive.advisoryPartitionSizeInBytes` ，倾斜数据分区拆分,小数据分区合并优化时,建议的分区大小(以字节为单位)

DPP(Dynamic Partition Pruning，动态分区剪裁)

所谓的动态分区裁剪就是基于运行时（run time）推断出来的信息来进一步进行分区裁剪，从而减少事实表中数据的扫描量、降低 I/O 开销，提升执行性能。

我们在进行事实表和维度表的Join过程中，把事实表中的无效数据进行过滤，例如：

```
SELECT * FROM dim
JOIN fact
ON (dim.col = fact.col)
WHERE dim.col = 'dummy'
```

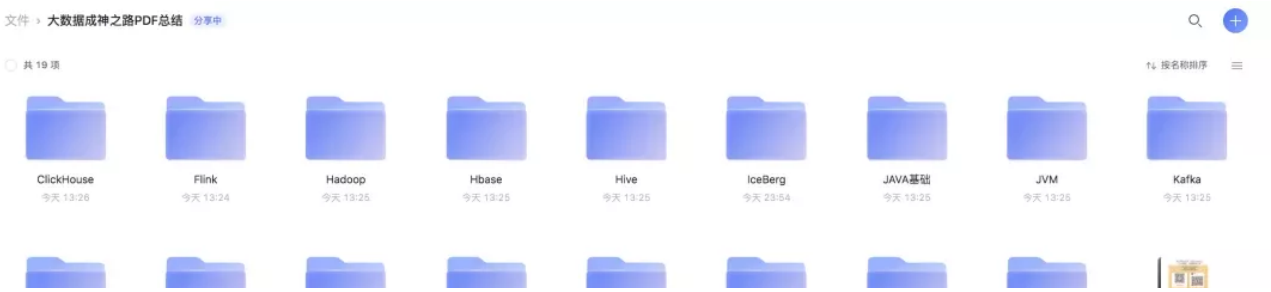
当SQL满足DPP的要求后，会根据关联关系 `dim.col = fact.col` ,通过维度表的列传导到事实表的col字段，只扫描事实表中满足条件的部分数据，就可以做到减少数据扫描量，提升I/O效率。

但是使用DPP的前提条件比较苛刻，需要满足以下条件：

1. 事实表必须是分区表
2. 只支持等值Join
3. 维度表过滤之后的数据必须小于广播阈值： `spark.sql.autoBroadcastJoinThreshold`

以上就是Spark3.0中最重要的两个特性 `AQE` 和 `DPP` 了。

电子版把他们分类做成了下面这个样子，并且放在了阿里云盘提供下载。



我们点开一个文件夹后:

> Hi，我是王知无，一个大数据领域的原创作者。
> 放心关注我，获取更多行业的一手消息。