



下载APP



19 | 配置项详解：哪些参数会影响应用程序执行性能？

2021-10-22 吴磊

《零基础入门Spark》

课程介绍 >



讲述：吴磊

时长 19:47 大小 18.13M



你好，我是吴磊。

在上一讲，我们学习了 Broadcast Join 这种执行高效的 Join 策略。要想触发 Spark SQL 选择这类 Join 策略，可以利用 SQL Functions 中的 broadcast 函数来强制广播基表。在这种情况下，Spark SQL 会完全“尊重”开发者的意愿，只要基表小于 8GB，它就会竭尽全力地去尝试进行广播并采用 Broadcast Join 策略。

除了这种比较“强势”的做法，我们还可以用另一种比较温和方式，来把选择权“下放”给 Spark SQL，让它自己来决定什么时候选择 Broadcast Join，什么时候回退到 Shuffle Join。这种温和的方式，就是配置项设置。在第 12 讲，我们掌握了 Spark 常规配置项，今天这一讲，咱们来说一说与 Spark SQL 有关的那些配置项。



不过，打开 Spark 官网的 [Configuration 页面](#)，你会发现，这里有上百个配置项，与 Spark SQL 相关的有好几十个，看得人眼花缭乱、头晕目眩。实际上，绝大多数配置项仅需采用默认值即可，并不需要我们过多关注。因此，我们把目光和注意力聚集到 Join 策略选择和 AQE 上。

Join 策略的重要性不必多说，AQE (Adaptive Query Execution) 是 Spark 3.0 推出的新特性，它帮助 Spark SQL 在运行时动态地调整执行计划，更加灵活地优化作业的执行性能。

Broadcast Join

接下来，我们先来说说，如何使用配置项来“温和”地让 Spark SQL 选择 Broadcast Join。对于参与 Join 的两张表来说，我们把其中尺寸较小的表称作基表。

如果基表的存储尺寸小于广播阈值，那么无需开发者显示调用 broadcast 函数，Spark SQL 同样会选择 Broadcast Join 的策略，在基表之上创建广播变量，来完成两张表的数据关联。

那么问题来了，广播阈值是什么，它是怎么定义的呢？广播阈值实际上就是一个标记存储尺寸的数值，它可以是 10MB、也可是 1GB，等等。**广播阈值由如下配置项设定，只要基表小于该配置项的设定值，Spark SQL 就会自动选择 Broadcast Join 策略。**


配置项	含义	默认值
spark.sql.autoBroadcastJoinThreshold	基表尺寸必须小于该值，才能触发 Broadcast Join	10MB



广播阈值配置项

如上表所示，广播阈值的默认值为 10MB。一般来说，在工业级应用中，我们往往把它设置到 2GB 左右，即可有效触发 Broadcast Join。广播阈值有了，要比较它与基表存储尺寸谁大谁小，Spark SQL 还要还得事先计算好基表的存储尺寸才行。那问题来了，Spark SQL 依据什么来计算这个数值呢？

这个问题要分两种情况讨论：如果基表数据来自文件系统，那么 Spark SQL 用来与广播阈值对比的基准，就是基表在磁盘中的存储大小。如果基表数据来自 DAG 计算的中间环节，那么 Spark SQL 将参考 DataFrame 执行计划中的统计值，跟广播阈值做对比，如下所示。

 复制代码

```
1 val df: DataFrame = _
2 // 先对分布式数据集加Cache
3 df.cache.count
4
5 // 获取执行计划
6 val plan = df.queryExecution.logical
7
8 // 获取执行计划对于数据集大小的精确预估
9 val estimated: BigInt = spark
10 .sessionState
11 .executePlan(plan)
12 .optimizedPlan
13 .stats
14 .sizeInBytes
```

讲到这里，你也许会有点不耐烦：“何必这么麻烦，又要设置配置项，又要提前预估基表大小，真是麻烦！还不如用上一讲提到的 broadcast 函数来得干脆！”

从开发者的角度看来，确实 broadcast 函数用起来更方便一些。不过，广播阈值加基表预估的方式，除了为开发者提供一条额外的调优途径外，还为 Spark SQL 的动态优化奠定了基础。

所谓动态优化，自然是相对静态优化来说的。在 3.0 版本之前，对于执行计划的优化，Spark SQL 仰仗的主要是编译时（运行时之前）的统计信息，如数据表在磁盘中的存储大小，等等。

因此，在 3.0 版本之前，Spark SQL 所有的优化机制（如 Join 策略的选择）都是静态的，它没有办法在运行时动态地调整执行计划，从而顺应数据集在运行时此消彼长的变化。

举例来说，在 Spark SQL 的逻辑优化阶段，两张大表的尺寸都超过了广播阈值，因此 Spark SQL 在物理优化阶段，就不得不选择 Shuffle Join 这种次优的策略。

但实际上，在运行时期间，其中一张表在 Filter 过后，剩余的数据量远小于广播阈值，完全可以放进广播变量。可惜此时“木已成舟”，静态优化机制没有办法再将 Shuffle Join 调整为 Broadcast Join。

AQE

为了弥补静态优化的缺陷、同时让 Spark SQL 变得更加智能，Spark 社区在 3.0 版本中推出了 AQE 机制。

AQE 的全称是 Adaptive Query Execution，翻译过来是“自适应查询执行”。它包含了 3 个动态优化特性，分别是 Join 策略调整、自动分区合并和自动倾斜处理。

或许是 Spark 社区对于新的优化机制偏向于保守，AQE 机制默认是未开启的，要想充分利用上述的 3 个特性，我们得先把 spark.sql.adaptive.enabled 修改为 true 才行。

配置项	含义	默认值
spark.sql.adaptive.enabled	是否启用AQE，设置值的类型是布尔值	FALSE



是否启用AQE

好啦，成功开启了 AQE 机制之后，接下来，我们就结合相关的配置项，来聊一聊这些特性都解决了哪些问题，以及它们是如何工作的。

Join 策略调整

我们先来说说 Join 策略调整，如果用一句话来概括，**Join 策略调整指的就是 Spark SQL 在运行时动态地将原本的 Shuffle Join 策略，调整为执行更加高效的 Broadcast Join。**

具体来说，每当 DAG 中的 Map 阶段执行完毕，Spark SQL 就会结合 Shuffle 中间文件的统计信息，重新计算 Reduce 阶段数据表的存储大小。如果发现基表尺寸小于广播阈值，那么 Spark SQL 就把下一阶段的 Shuffle Join 调整为 Broadcast Join。

不难发现，这里的关键是 Shuffle，以及 Shuffle 的中间文件。**事实上，不光是 Join 策略调整这个特性，整个 AQE 机制的运行，都依赖于 DAG 中的 Shuffle 环节。**

所谓巧妇难为无米之炊，要做到动态优化，Spark SQL 必须要仰仗运行时的执行状态，而 Shuffle 中间文件，则是这些状态的唯一来源。

举例来说，通过 Shuffle 中间文件，Spark SQL 可以获得诸如文件尺寸、Map Task 数据分片大小、Reduce Task 分片大小、空文件占比之类的统计信息。正是利用这些统计信息，Spark SQL 才能在作业执行的过程中，动态地调整执行计划。

我们结合例子进一步来理解，以 Join 策略调整为例，给定如下查询语句，假设 salaries 表和 employees 表的存储大小都超过了广播阈值，在这种情况下，对于两张表的关联计算，Spark SQL 只能选择 Shuffle Join 策略。

不过实际上，employees 按照年龄过滤之后，剩余的数据量是小于广播阈值的。这个时候，得益于 AQE 机制的 Join 策略调整，Spark SQL 能够把最初制定的 Shuffle Join 策略，调整为 Broadcast Join 策略，从而在运行时加速执行性能。

[📄 复制代码](#)

```
1 select * from salaries inner join employees
2   on salaries.id = employees.id
3   where employees.age >= 30 and employees.age < 45
```

你看，在这种情况下，广播阈值的设置、以及基表过滤之后数据量的预估，就变得非常重要。原因在于，这两个要素决定了 Spark SQL 能否成功地在运行时充分利用 AQE 的 Join 策略调整特性，进而在整体上优化执行性能。因此，我们必须掌握广播阈值的设置方法，以及数据集尺寸预估的方法。

介绍完 Join 策略调整，接下来，我们再来说说 AQE 机制的另外两个特性：自动分区合并与自动倾斜处理，它们都是对于 Shuffle 本身的优化策略。

我们先来说说，自动分区合并与自动倾斜处理都在尝试解决什么问题。我们知道，Shuffle 的计算过程分为两个阶段：Map 阶段和 Reduce 阶段。Map 阶段的数据分布，往往由分布式文件系统中的源数据决定，因此数据集在这个阶段的分布，是相对均匀的。

Reduce 阶段的数据分布则不同，它是由 Distribution Key 和 Reduce 阶段并行度决定的。并行度也就是分区数目，这个概念咱们在之前的几讲反复强调，想必你并不陌生。

而 Distribution Key 则定义了 Shuffle 分发数据的依据，对于 reduceByKey 算子来说，Distribution Key 就是 Paired RDD 的 Key；而对于 repartition 算子来说，Distribution Key 就是传递给 repartition 算子的形参，如 repartition(\$ "Column Name")。

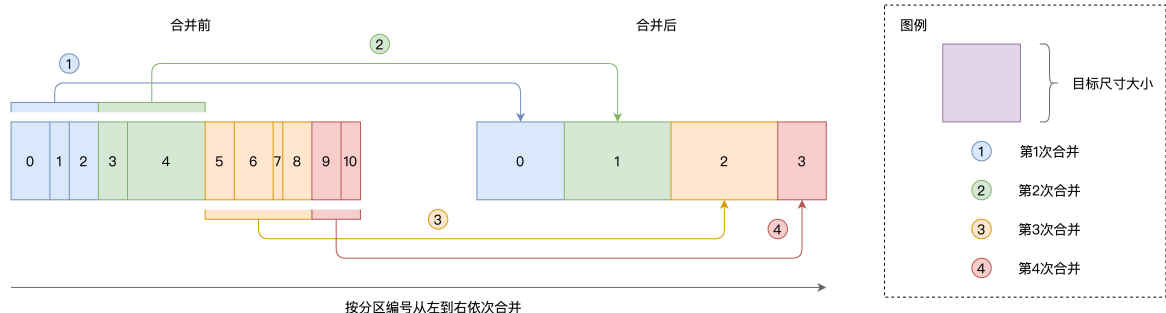
在业务上，Distribution Key 往往是 user_id、item_id 这一类容易产生倾斜的字段，相应地，数据集在 Reduce 阶段的分布往往也是不均衡的。

数据的不均衡往往体现在两个方面，一方面是一部分数据分区的体量过小，而另一方面，则是少数分区的体量极其庞大。AQE 机制的自动分区合并与自动倾斜处理，正是用来应对数据不均衡的这两个方面。

自动分区合并

了解了自动分区合并的用武之地，接下来，我们来说说，Spark SQL 具体如何做到把 Reduce 阶段过小的分区合并到一起。要弄清楚分区合并的工作原理，我们首先得搞明白：“分区合并从哪里开始？又到哪里结束呢？”

具体来说，Spark SQL 怎么判断一个数据分区是不是足够小、它到底需不需要被合并？再者，既然是对多个分区做合并，那么自然就存在一个收敛条件。原因很简单，如果一直不停地合并下去，那么整个数据集就被捏合成了一个超级大的分区，并行度也会下降至 1，显然，这不是我们想要的结果。



分区合并示意图

事实上，Spark SQL 采用了一种相对朴素的方法，来实现分区合并。具体来说，**Spark SQL 事先并不去判断哪些分区是不是足够小，而是按照分区的编号依次进行扫描，当扫描过的数据体量超过了“目标尺寸”时，就进行一次合并。**而这个目标尺寸，由以下两个配置项来决定。

配置项	含义
spark.sql.adaptive.advisoryPartitionSizeInBytes	合并后的目标分区大小
spark.sql.adaptive.coalescePartitions.minPartitionNum	分区合并之后，并行度不能低于该设置值



分区合并相关配置项

其中，开发者可以通过第一个配置项 `spark.sql.adaptive.advisoryPartitionSizeInBytes` 来直接指定目标尺寸。第二个参数用于限制 Reduce 阶段在合并之后的并行度，避免因为合并导致并行度过低，造成 CPU 资源利用不充分。

结合数据集大小与最低并行度，我们可以反推出来每个分区的平均大小，假设我们把这个平均大小记作是 `#partitionSize`。那么，实际的目标尺寸，取 `advisoryPartitionSizeInBytes` 设定值与 `#partitionSize` 之间较小的那个数值。

确定了目标尺寸之后，Spark SQL 就会依序扫描数据分区，当相邻分区的尺寸之和大于目标尺寸的时候，Spark SQL 就把扫描过的分区做一次合并。然后，继续使用这种方式，依次合并剩余的分区，直到所有分区都处理完毕。

自动倾斜处理

没有对比就没有鉴别，分析完自动分区合并如何搞定数据分区过小、过于分散的问题之后，接下来，我们再来说一说，自动倾斜处理如何应对那些倾斜严重的大分区。

经过上面的分析，我们不难发现，自动分区合并实际上包含两个关键环节，一个是确定合并的目标尺寸，一个是依次扫描合并。与之相对应，自动倾斜处理也分为两步，**第一步是检测并判定体量较大的倾斜分区，第二步是把这些大分区拆分为小分区。**要做到这两步，Spark SQL 需要依赖如下 3 个配置项。

作用	配置项	含义
判定大分区	spark.sql.adaptive.skewJoin.skewedPartitionFactor	倾斜分区的比例系数
	spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes	倾斜分区的最低阈值
拆分大分区	spark.sql.adaptive.advisoryPartitionSizeInBytes	倾斜分区的拆分单位



自动倾斜处理的配置项

其中，前两个配置项用于判定倾斜分区，第 3 个配置项 `advisoryPartitionSizeInBytes` 我们刚刚学过，这个参数除了用于合并小分区外，同时还用于拆分倾斜分区，可以说是“一菜两吃”。

下面我们重点来讲一讲，Spark SQL 如何利用前两个参数来判定大分区的过程。

首先，Spark SQL 对所有数据分区按照存储大小做排序，取中位数作为基数。然后，将中位数乘以 `skewedPartitionFactor` 指定的比例系数，得到判定阈值。凡是存储尺寸大于判定阈值的数据分区，都有可能被判定为倾斜分区。

为什么说“有可能”，而不是“一定”呢？原因是，倾斜分区的判定，还要受到 `skewedPartitionThresholdInBytes` 参数的限制，它是判定倾斜分区的最低阈值。也就是说，只有那些尺寸大于 `skewedPartitionThresholdInBytes` 设定值的“候选分区”，才会最终判定为倾斜分区。

为了更好地理解这个判定的过程，我们来举个例子。假设数据表 `salaries` 有 3 个分区，大小分别是 90MB、100MB 和 512MB。显然，这 3 个分区的中位数是 100MB，那么拿它乘以比例系数 `skewedPartitionFactor`（默认值为 5），得到判定阈值为 $100\text{MB} * 5 = 500\text{MB}$ 。因此，在咱们的例子中，只有最后一个尺寸为 512MB 的数据分区会被列为“候选分区”。

接下来，Spark SQL 还要拿 512MB 与 `skewedPartitionThresholdInBytes` 作对比，这个参数的默认值是 256MB。

显然，512MB 比 256MB 要大得多，这个时候，Spark SQL 才会最终把最后一个分区，判定为倾斜分区。相反，假设我们把 `skewedPartitionThresholdInBytes` 这个参数调大，设置为 1GB，那么最后一个分区就不满足最低阈值，因此也就不会被判定为倾斜分区。

倾斜分区判定完毕之后，下一步，就是根据 `advisoryPartitionSizeInBytes` 参数指定的目标尺寸，对大分区进行拆分。假设我们把这个参数的值设置为 256MB，那么刚刚 512MB 的大分区就会被拆成两个小分区（ $512\text{MB} / 2 = 256\text{MB}$ ）。拆分之后，`salaries` 表就由 3 个分区变成了 4 个分区，每个数据分区的尺寸，都不超过 256MB。


重点回顾

好啦，到此为止，与 Spark SQL 相关的重要配置项，我们就讲到这里。今天的内容很多，我们一起来总结一下。

首先，我们介绍了广播阈值这一概念，它的作用在于，当基表尺寸小于广播阈值时，Spark SQL 将自动选择 Broadcast Join 策略来完成关联计算。

然后，我们分别介绍了 **AQE (Adaptive Query Execution) 机制的 3 个特性，分别是 Join 策略调整、自动分区合并、以及自动倾斜处理**。与 Spark SQL 的静态优化机制不同，AQE 结合 Shuffle 中间文件提供的统计信息，在运行时动态地调整执行计划，从而达到优化作业执行性能的目的。

所谓 Join 策略调整，它指的是，结合过滤之后的基表尺寸与广播阈值，Spark SQL 在运行时动态地将原本的 Shuffle Join 策略，调整为 Broadcast Join 策略的过程。基表尺寸的预估，可以使用如下方法来获得。

 复制代码

```
1 val df: DataFrame = _
2 // 先对分布式数据集加Cache
3 df.cache.count
4
5 // 获取执行计划
6 val plan = df.queryExecution.logical
7
8 // 获取执行计划对于数据集大小的精确预估
9 val estimated: BigInt = spark
10 .sessionState
11 .executePlan(plan)
12 .optimizedPlan
```

自动分区合并与自动倾斜处理，实际上都是用来解决 Shuffle 过后，数据分布不均匀的问题。自动分区合并的作用，在于合并过小的数据分区，从而避免 Task 粒度过细、任务调度开销过高的问题。与之相对，自动倾斜处理，它的用途在于拆分过大的数据分区，从而避免个别 Task 负载过高而拖累整个作业的执行性能。

不论是广播阈值，还是 AQE 的诸多特性，我们都可以通过调节相关的配置项，来影响 Spark SQL 的优化行为。为了方便你回顾、查找这些配置项，我整理了如下表格，供你随时参考。

作用	配置项	含义
AQE	spark.sql.adaptive.enabled	是否启用AQE，设置值的类型是布尔值
Join策略调整	spark.sql.autoBroadcastJoinThreshold	基表要小于该值才能触发Broadcast Join
自动分区合并	spark.sql.adaptive.advisoryPartitionSizeInBytes	合并后的目标分区大小
	spark.sql.adaptive.coalescePartitions.minPartitionNum	分区合并之后，并行度不能低于该设置值
自动倾斜处理	spark.sql.adaptive.skewJoin.skewedPartitionFactor	倾斜分区的比例系数
	spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes	倾斜分区的最低阈值



每课一练

结合 AQE 必须要依赖 Shuffle 中间文件这一特点，你能说一说，AQE 有什么不尽如人意之处吗？（提示：从 Shuffle 的两个计算阶段出发，去思考这个问题）

欢迎你在留言区跟我交流讨论，也推荐你把这一讲分享给更多的同事、朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | 数据关联优化：都有哪些Join策略，开发者该如何取舍？

下一篇 20 | Hive + Spark强强联合：分布式数仓的不二之选

1024 活动特惠

VIP 年卡直降 ¥2000

新课上线即解锁，享 365 天畅看全场

超值拿下 ¥999



精选留言 (2)

💬 写留言



海阔天空

2021-10-23

自动倾斜处理后，如何保证同样key的数据在同一个reduce里执行



👍 3



Geek_1e4b29

2021-10-26

同问，相同的key如何解决,加问一个，你们的生产上开了这个参数吗，作业有多大，有什么风险🙄



