> 思考学习的深度

Spark之Shuffle调优

调优一：调节map端缓冲区大小

在Spark任务运行过程中，如果shuffle的map端处理的数据量比较大，但是map端缓冲的大小是固定的，可能会出现map端缓冲数据频繁spill溢写到磁盘文件中的情况，使得性能非常低下，通过调节map端缓冲的大小，可以**避免频繁的磁盘IO操作**，进而提升Spark任务的整体性能。

map端缓冲的默认配置是32KB，如果每个task处理640KB的数据，那么会发生640/32 = 20次溢写，如果每个task处理64000KB的数据，机会发生64000/32=2000此溢写，这对于性能的影响是非常严重的。

map端缓冲的配置方法如代码

```
val conf = new SparkConf()
  .set("spark.shuffle.file.buffer", "64")
```

调优二：调节reduce端拉取数据缓冲区大小

Spark Shuffle过程中，shuffle reduce task的buffer缓冲区大小决定了**reduce task每次能够缓冲的数据量**，也就是每次能够拉取的数据量，如果内存资源较为充足，适当增加拉取数据缓冲区的大小，可以减少拉取数据的次数，也就可以**减少网络传输的次数**，进而提升性能。

reduce端数据拉取缓冲区的大小可以通过spark.reducer.maxSizeInFlight参数进行设置，默认为48MB，该参数的设置方法如代码

```
val conf = new SparkConf()
  .set("spark.reducer.maxSizeInFlight", "96")
```

调优三：调节reduce端拉取数据重试次数

Spark Shuffle过程中，reduce task拉取属于自己的数据时，如果因为**网络异常等原因导致失败会自动进行重试。**对于那些包含了特别耗时的shuffle操作的作业，建议**增加重试最大次数**（比如60次），以避免由于JVM的full gc或者网络不稳定等因素导致的数据拉取失败。在实践中发现，对于针对超大数据量（数十亿~上百亿）的shuffle过程，调节该参数可以大幅度提升稳定性。

reduce端拉取数据重试次数可以通过spark.shuffle.io.maxRetries参数进行设置，该参数就代表了可以重试的最大次数。如果在指定次数之内拉取还是没有成功，就可能会导致作业执行失败，**默认为3**

reduce端拉取数据重试次数配置

```
val conf = new SparkConf()
 .set("spark.shuffle.io.maxRetries", "6")
```

调优四：调节reduce端拉取数据等待间隔

Spark Shuffle过程中，reduce task拉取属于自己的数据时，如果因为网络异常等原因导致失败会自动进行重试，在一次失败后，会等待一定的**时间间隔**再进行重试，可以通过加大间隔时长（比如60s），以增加shuffle操作的稳定性。

reduce端拉取数据等待间隔可以通过spark.shuffle.io.retryWait参数进行设置，**默认值为5s**

reduce端拉取数据等待间隔配置

```
val conf = new SparkConf()
 .set("spark.shuffle.io.retryWait", "60s")
```

调优五：调节SortShuffle排序操作阈值

对于SortShuffleManager，如果shuffle reduce task的数量小于某一阈值则shuffle write过程中不会进行排序操作，而是直接按照**未经优化的HashShuffleManager的方式去写数据**，但是最后会将每个task产生的所有临时磁盘文件都合并成一个文件，并会创建单独的索引文件。

当你使用SortShuffleManager时，如果的确不需要排序操作，那么建议将这个参数调大一些，大于shuffle read task的数量，那么此时map-side就不会进行排序了，**减少了排序的性能开销**，但是这种方式下，依然会产生大量的磁盘文件，因此shuffle write性能有待提高。

SortShuffleManager排序操作阈值的设置可以通过spark.shuffle.sort. bypassMergeThreshold这一参数进行设置，默认值为200

参数的设置方法

```
val conf = new SparkConf()
  .set("spark.shuffle.sort.bypassMergeThreshold", "400")
```