> 思考意志力是如何损耗的

Spark之JVM调优

> 对于JVM调优，首先应该明确，full gc/minor gc，都会导致JVM的工作线程停止工作，即stop the world。

调优一：降低cache操作的内存占比

1.静态内存管理机制

根据Spark静态内存管理机制，堆内存被划分为了两块，**Storage和Execution**。Storage主要用于缓存RDD数据和broadcast数据，Execution主要用于缓存在shuffle过程中产生的中间数据，Storage占系统内存的60%，Execution占系统内存的20%，并且两者完全独立。

在一般情况下，**Storage的内存都提供给了cache操作**，但是如果在某些情况下cache操作内存不是很紧张，而task的算子中创建的对象很多，Execution内存又相对较小，这回**导致频繁的minor gc，甚至于频繁的full gc，**进而导致Spark频繁的停止工作，性能影响会很大。

在Spark UI中可以查看每个stage的运行情况，包括每个task的运行时间、gc时间等等，如果发现**gc太频繁**，时间太长，就可以**考虑调节Storage的内存占比**，让task执行算子函数式，有更多的内存可以使用。

Storage内存区域可以通过**spark.storage.memoryFraction**参数进行指定，默认为0.6，即60%，可以逐级向下递减，如代码清单2-6所示：

Storage内存占比设置

```
val conf = new SparkConf()
 .set("spark.storage.memoryFraction", "0.4")
```

2.统一内存管理机制

根据Spark统一内存管理机制，堆内存被划分为了两块，Storage和Execution。Storage主要用于缓存数据，Execution主要用于缓存在shuffle过程中产生的中间数据，两者所组成的内存部分称为统一内存，Storage和Execution各占统一内存的50%，**由于动态占用机制的实现，shuffle过程需要的内存过大时，会自动占用Storage的内存区域，因此无需手动进行调节。**

调优二：调节Executor堆外内存

Executor的堆外内存主要用于**程序的共享库、Perm Space、 线程Stack和一些Memory mapping**等, 或者类C方式allocate object。

有时，如果你的Spark作业处理的数据量非常大，**达到几亿的数据量**，此时运行Spark作业会时不时地报错，例如shuffle output file cannot find，executor lost，task lost，out of memory等，这可能是Executor的堆外内存不太够用，导致Executor在运行的过程中内存溢出。

stage的task在运行的时候，可能要从一些Executor中去拉取shuffle map output文件，但是Executor可能已经由于内存溢出挂掉了，**其关联的BlockManager也没有了**，这就可能会报出shuffle output file cannot find，executor lost，task lost，out of memory等错误，此时，就可以**考虑调节一下Executor的堆外内存**，也就可以避免报错，与此同时，堆外内存调节的比较大的时候，对于性能来讲，也会带来一定的提升。

默认情况下，**Executor堆外内存上限大概为300多MB，在实际的生产环境下，对海量数据进行处理的时候，这里都会出现问题**，导致Spark作业反复崩溃，无法运行，此时就会去调节这个参数，**到至少1G，甚至于2G、4G。**

Executor堆外内存的配置需要在spark-submit脚本里配置

```
--conf spark.yarn.executor.memoryOverhead=2048
```

以上参数配置完成后，会避免掉某些JVM OOM的异常问题，同时，可以提升整体Spark作业的性能。

调优三：调节连接等待时长

在Spark作业运行过程中，Executor优先从自己本地关联的BlockManager中获取某份数据，如果本地BlockManager没有的话，**会通过TransferService远程连接其他节点上Executor的BlockManager来获取数据**

如果task在运行过程中创建大量对象或者创建的对象较大，会占用大量的内存，这回**导致频繁的垃圾回收**，但是垃圾回收会导致工作现场全部停止，也就是说，**垃圾回收一旦执行，Spark的Executor进程就会停止工作**，无法提供相应，此时，由于没有响应，无法建立网络连接，会导致网络连接超时。

在生产环境下，有时会遇到file not found、file lost这类错误，在这种情况下，**很有可能是Executor的BlockManager在拉取数据的时候，无法建立连接**，然后超过默认的连接等待时长60s后，宣告数据拉取失败，如果反复尝试都拉取不到数据，可能会导致Spark作业的崩溃。这种情况也可能会导致DAGScheduler反复提交几次stage，TaskScheduler返回提交几次task，大大延长了我们的Spark作业的运行时间。

此时，可以考虑调节连接的超时时长，连接等待时长需要在spark-submit脚本中进行设置

```
--conf spark.core.connection.ack.wait.timeout=300
```

调节连接等待时长后，通常可以避免部分的XX文件拉取失败、XX文件lost等报错。