Spark Streaming性能调优详解

Spark Streaming提供了高效便捷的流式处理模式，但是在有些场景下，使用默认的配置达不到最优，甚至无法实时处理来自外部的数据，这时候我们就需要对默认的配置进行相关的修改。由于现实中场景和数据量不一样，所以我们无法设置一些通用的配置（要不然Spark Streaming开发者就不会弄那么多参数，直接写死不得了），我们需要根据数据量，场景的不同设置不一样的配置，这里只是给出建议，这些调优不一定试用于你的程序，一个好的配置是需要慢慢地尝试。

1、设置合理的批处理时间(batchDuration)。

在构建StreamingContext的时候，需要我们传进一个参数，用于设置Spark Streaming批处理的时间间隔。Spark会每隔batchDuration时间去提交一次Job，如果你的Job处理的时间超过了batchDuration的设置，那么会导致后面的作业无法按时提交，随着时间的推移，越来越多的作业被拖延，最后导致整个Streaming作业被阻塞，这就间接地导致无法实时处理数据，这肯定不是我们想要的。

另外，虽然batchDuration的单位可以达到毫秒级别的，但是经验告诉我们，如果这个值过小将会导致因频繁提交作业从而给整个Streaming带来负担，所以请尽量不要将这个值设置为小于500ms。在很多情况下，设置为500ms性能就很不错了。

那么，如何设置一个好的值呢？我们可以先将这个值位置为比较大的值（比如10S），如果我们发现作业很快被提交完成，我们可以进一步减小这个值，知道Streaming作业刚好能够及时处理完上一个批处理的数据，那么这个值就是我们要的最优值。

2、增加Job并行度

我们需要充分地利用集群的资源，尽可能的将Task分配到不同的节点，一方面可以充分利用集群资源；另一方面还可以及时的处理数据。比如我们使用Streaming接收来自Kafka的数据，我们可以对每个Kafka分区设置一个接收器，这样可以达到负载均衡，及时处理数据（关于如何使用Streaming读取Kafka中的数据，可以参见《Spark Streaming和Kafka整合开发指南(一)》和《Spark Streaming和Kafka整合开发指南(二)》）。

再如类似reduceByKey()和Join函数都可以设置并行度参数。

3、使用Kryo系列化。

Spark默认的是使用Java内置的系列化类，虽然可以处理所有自继承java.io.Serializable的类系列化的类，但是其性能不佳，如果这个成为性能瓶颈，可以使用Kryo系列化类，关于如何在Spark中使用Kroy，请参见《在Spark中自定义Kryo序列化输入输出API》
。使用系列化数据可以很好地改善GC行为。

4、缓存需要经常使用的数据

对一些经常使用到的数据，我们可以显式地调用rdd.cache()来缓存数据，这样也可以加快数据的处理，但是我们需要更多的内存资源。

5、清除不需要的数据

随着时间的推移，有一些数据是不需要的，但是这些数据是缓存在内存中，会消耗我们宝贵的内存资源，我们可以通过配置spark.cleaner.ttl为一个合理的值；但是这个值不能过小，因为如果后面计算需要用的数据被清除会带来不必要的麻烦。而且，我们还可以配置选项spark.streaming.unpersist为true(默认就是true)来更智能地去持久化（unpersist）RDD。这个配置使系统找出那些不需要经常保有的RDD，然后去持久化它们。这可以减少Spark
RDD的内存使用，也可能改善垃圾回收的行为。

6、设置合理的GC

GC是程序中最难调的一块，不合理的GC行为会给程序带来很大的影响。在集群环境下，我们可以使用并行Mark-
Sweep垃圾回收机制，虽然这个消耗更多的资源，但是我们还是建议开启。可以如下配置：

```
spark.executor.extraJavaOptions=-XX:+UseConcMarkSweepGC
```

更多的关于GC行为的配置，请参考Java垃圾回收相关文章。这里就不详细介绍了。

7、设置合理的CPU资源数

很多情况下Streaming程序需要的内存不是很多，但是需要的CPU要很多。在Streaming程序中，CPU资源的使用可以分为两大类：（1）、用于接收数据；（2）、用于处理数据。我们需要

设置足够的CPU资源，使得有足够的CPU资源用于接收和处理数据，这样才能及时高效地处理数据。