**一：Spark性能优化核心基石**

1. Spark是采用Master-Slaves的模式进行资源管理和任务执行的管理：
   1. 资源管理：Master-Workers，在一台机器上可以有多个Workers，可以通过配置Spark-env.sh让每台host上有若干的Worker；
   2. 任务执行：Driver-Executors，每个应用程序在的Driver只有一个，当在一台机器上分配多个Workers的时候那么默认情况下每个Worker都会为当前运行的应用程序分配一个Executor，但是我们可以修改配置来让每个Worker为我们当前的应用程序分配若干个Executors；程序运行的时候会被划分成为若干个Stages（Stage内部没有Shuffle，遇到Shuffle的时候会划分出新的Stage），每个Stage里面包含若干个处理逻辑完全一样只是处理的数据不一样的Tasks，这些Tasks会被分配到Executor上去并行执行，Stage内部的计算都以Pipeline的方式进行，不同的Stage之间是产生Shuffle的唯一方式。
2. 在Spark中可以考虑在Worker节点上使用固态硬盘以及把Worker的Shuffle结果保存到RAM Disk的方式来极大的提高性能。

**二：OOM：内存溢出**

1. 当你经常发现机器频繁的OOM的时候，可以考虑的一种方式就是减少并行度，这样同样的内存空间并行运算的任务少，那么对内存的占用就更少，也就减少了OOM的可能性；
2. 默认情况下Spark的Executor会尽可能占用当前机器上尽量多的Core，这样带来的一个好处就是可以最大化的提高计算的并行度，减少一个Job中任务运行的批次，但带来的一个风险就是如果每个Task占用内存比较大，就需要频繁的spill over或者有更多的OOM的风险。

**注：**因此如果集群中即运行了Spark框架，又运行了其他框架，比如，Hadoop中的MapReduce,这时候就需要使用资源管理器比如yarn, mesos等来分配资源。否则Spark会占用绝大多数资源。所以Spark给人的印象就是耗资源，但是这种耗资源是为了换取快速运行。

**尽快运算包含两方面：**

占用尽量多的CPU。

占用尽量多的内存。

1. 处理Spark Job的时候如果发现比较容易内存溢出，另外一个比较有效的办法是减少并行的Executor数量，这样每个Executor就可以分配到更多的内存，进而增加每个Task使用的内存数量，降低OOM的风险。
2. 处理Spark Job时候如果发现比较容易内存溢出，一个比较有效的办法就是增加Task的并行度，这样每个Task处理的Partition的数据量就变少了，减少了OOM的可能性。
3. 适当设置Partition分片数是非常重要的，过少的Partition分片数可能会因为每个Partition数据量太大而导致OOM以及频繁的GC(JVM的垃圾回收机制)，而过多的Partition分片数可能会因为每个Partition数据量太小而导致执行效率低下。
4. 最简单的一个方法就是增大分片的数量。每个分片的数据规模就减少了。

**三：小文件特别多的情况**

1. 处理Spark Job的过程中如果出现特别多的小文件，这时候就可以通过coalesce来减少Partition的数量，进而减少并行运算的Task的数量来减少过多任务的开辟，从而提升硬件的使用效率。

**注**：

**1**.coalesce函数是用来对RDD进行重分区，并且可以通过传入参数的方式对分区数手动进行设置。Shuffle默认是false，如果你设置的分区数大于原来的分区数则必须将shuffle设置为true。

**def** coalesce(numPartitions: Int, shuffle: Boolean = **false**)(**implicit** ord: Ordering[T] = **null**)

**2.**提升硬件的使用效率，怎么理解？因为处理的总数据量没变，并行度减少了，Task数量就减少了，因此任务间的切换就减少了，所以处理效率就增高了。

2，如果有大量的小文件，这个时候你就必须要减少文件的分片数。

**四：Spark中的慢任务**

1. 处理Spark Job时候如果发现某些Task运行的特别慢，这个时候应该考虑增加任务的并行度，减少一个任务具体处理的Partition的数据量来提高执行效率。

2，处理Spark Job时候如果发现某些Task运行的特别慢，另外一个处理办法是增加并行的Executor的个数，这样每个Executor分配的计算资源就变少了，可以提升硬件的整体使用效率。

**注：**增加Executor的个数，这样每个Executor分配的计算资源就变少了，因为Task运行太慢的话，其占用的资源，其他的Task就无法使用，因此，要增加Executor个数。

**五：提升Spark硬件尤其是CPU使用率**

1. 提升Spark硬件尤其是CPU使用率的一个方式就是增加Executor的并行度，但是如果Executor过多的话，直接分配在每个Executor的内存就大大减少，在内存中的操作就减少，基于磁盘的操作就越来越多，导致性能越来越差。因为Spark是粗粒度的，资源一旦被分配就一直被占用，因此，增加Executor的并行度就可以增加CPU和硬件的使用。
2. 如果Spark中CPU的使用率不够高，可以考虑为当前的程序分配更多的Executor，或者增加更多的Worker实例来充分的使用多核的潜能，其目的就是要为当前程序多使用cores.

**六：问题**

1. Executor占用绝大多数的cores，但是CPU的使用率却不高？

可以通过设置让Executor占用少一点的cores，或者说增加更多的Worker，或者增加一个Worker下更多的Executor，来增加并行度，从而提高CPU的利用率。另一方面，Executor在处理数据的时候，我们必须严格的考虑内存的使用，因为很容易出现OOM，Executor中的Task，每个Task处理一个Partition，有多少Partition就是有多少并行度。如果Parition比较小，那分片的数据量就比较小。导致执行效率低下，因为时间都耗在切换上了，但是如果Partition的值太大的话，导致每个分片的数据量太大。这时候内存的压力就会非常大。内存压力大导致的后果是：OOM，分片变少了，总并行度变低了，这时候就不能很好的利用硬件计算的潜力。

1. 在实际情况下，如何设置Partition大小？

默认情况下是根据parent RDD中的最大的并行度决定的，因为并行度会继承，spark.default.conf配置。

1. 如何设置cores？

CPU有两种模式（共享，独占），影响CPU最直接的就是并行度。在实际运行的时候一般都会根据输入和Executor内存大小，来确定使用多少cores。

为什么？因为如果输入数据很大的话，你不考虑你使用的cores的个数的话，那肯定会产生OOM。实际上一个简单事实是每个Core可以考虑分配2~3个Task，而钨丝计划的核心是：解决GC(是JVM的垃圾回收机制)。

七：总结

性能调优的有效性是暂时的！！！例如为当前的应用程序增加Executor可能在一开始可以提高性能（例如CPU使用率提高等），但是随着Executor越来越多，性能可能会下降！！！因为Executor越来越多的时候，为每个Executor分配的内存就越来越少，Task执行过程中可用的内存就越来越少，这个时候就要频繁Spill over到磁盘，此时自然而然的导致性能变差。

**总结如下：**

