# Spark 优化

## 数据分区

在分布式集群里，网络通信的代价很大，减少网络传输可以极大提升性能。

回想下，Mapreduce框架的性能开支主要在哪里？

IO：大量读写文件

网络传输：压缩（大文件变小文件从而减少网络传输，但是增加CPU计算负载）

网络传输主要在shuffle阶段，shuffle的原因是**相同的key存在不同的节点上**，按key进行聚合的时候不得不进行shuffle。

Spark把RDD进行分片(分区)，放在集群上并行计算。

同一个RDD，分片100个，集群有10个节点，平均一个节点10个分区。

对于sum型的计算：先进行每个分区的sum，然后把sum值shuffle传输到主程序进行全局sum，所以此时shuffle过程中只需要传输分区sum，网络开销很小。

但对于Join类的计算，需要把数据本身进行shuffle，网络开销很大。

Spark是如何优化这个问题的？

Spark把key-value RDD通过key的hashcode进行分区，且保证相同的key存储在同一个节点上。

key的分布不均衡决定了有的分区大，有的分区小。

这样对该RDD进行key聚合时，不需要shuffle过程。

Join操作时，通常把用的**频繁**的大表事先进行分区，如：

val userData = sc.sequenceFile[UserID, UserInfo]("hdfs://...")

.partitionBy(new HashPartitioner(100)) // 构造100个分区

.persist()

进行join时，仅需要对另一个小数据量的表进行shuffle过程。

## 从分区中获益的操作

能够从数据分区中**获益**的操作有cogroup()、

groupWith()、**join()**、leftOuterJoin()、rightOuterJoin()、groupByKey()、reduceByKey()、

combineByKey() 以及lookup()。基于key的操作都会获益。

而对于诸如join() 这样的二元操作，预先进行数据分区会让其中至少一个RDD（使用已知分区器的那个RDD）不发生数据shuffle。如果两个RDD 使用同样的分区方式，并且它们还缓存在同样的机器上（比如一个RDD 是通过mapValues() 从另一个RDD 中创建出来的，这两个RDD 就会拥有相同的键和分区方式），跨节点的数据shuffle就不会发生了。

## 参数优化

进行spark-submit时，会给每个作业分配资源。处理的数据量越大，需要分配的资源越多。