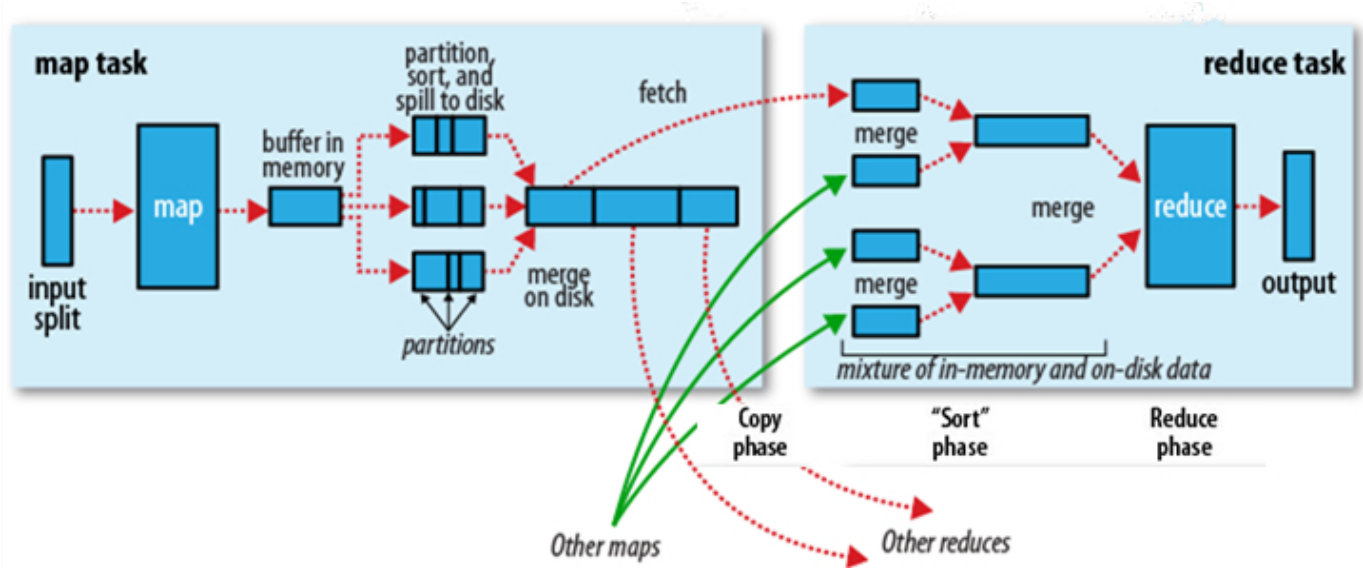
MapReduce：详细介绍Shuffle的执行过程

Shuffle过程是MapReduce的核心，也被称为奇迹发生的地方。要想理解MapReduce， Shuffle是必须要了解的。我看过很多相关的资料，但每次看完都云里雾里的绕着，很难理清大致的逻辑，反而越搅越混。前段时间在做MapReduce job 性能调优的工作，需要深入代码研究MapReduce的运行机制，这才对Shuffle探了个究竟。考虑到之前我在看相关资料而看不懂时很恼火，所以在这里我尽最大的可能试着把Shuffle说清楚，让每一位想了解它原理的朋友都能有所收获。如果你对这篇文章有任何疑问或建议请留言到后面，谢谢！

Shuffle的正常意思是洗牌或弄乱，可能大家更熟悉的是Java API里的Collections.shuffle(List)方法，它会随机地打乱参数list里的元素顺序。如果你不知道MapReduce里Shuffle是什么，那么请看这张图：



这张是官方对Shuffle过程的描述。但我可以肯定的是，单从这张图你基本不可能明白Shuffle的过程，因为它与事实相差挺多，细节也是错乱的。后面我会具体描述Shuffle的事实情况，所以这里你只要清楚Shuffle的大致范围就成 - 怎样把map task的输出结果有效地传送到reduce端。也可以这样理解， Shuffle描述着数据从map task输出到reduce task输入的这段过程。

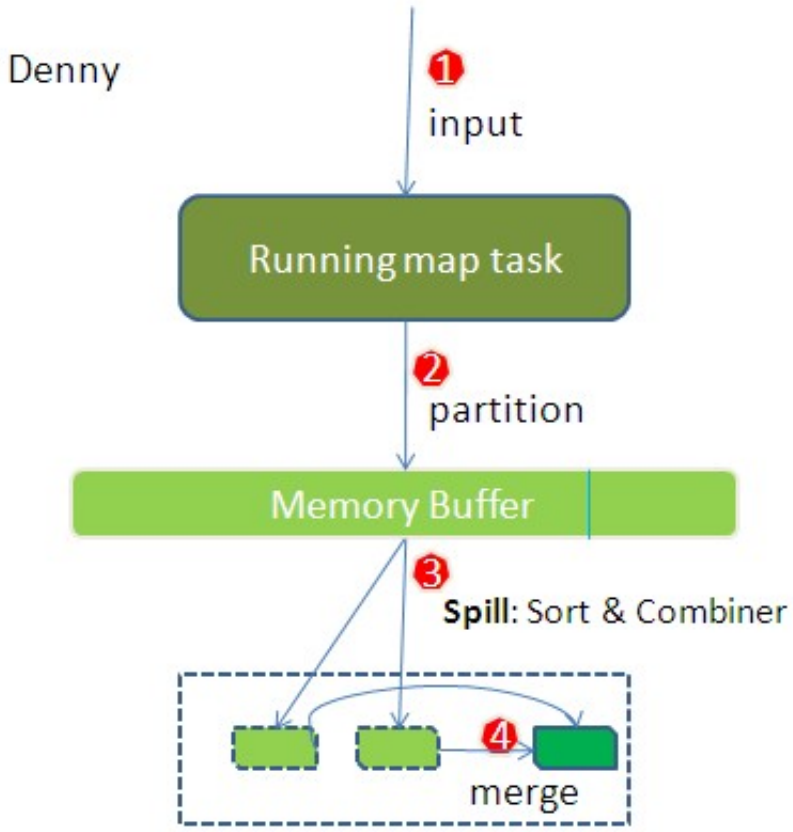
在Hadoop这样的集群环境中，大部分map task与reduce task的执行是在不同的节点上。当然很多情况下Reduce执行时需要跨节点去拉取其它节点上的map task结果。如果集群正在运行的job有很多，那么task的正常执行对集群内部的网络资源消耗会很严重。这种网络消耗是正常的，我们不能限制，能做的就是最大化地减少不必要的消耗。还有在节点内，相比于内存，磁盘IO对job完成时间的影响也是可观的。从最基本的要求来说，我们对Shuffle过程的期望可以有：

完整地从map task端拉取数据到reduce 端。
在跨节点拉取数据时，尽可能地减少对带宽的不必要消耗。
减少磁盘IO对task执行的影响。

OK，看到这里时，大家可以先停下来想想，如果是自己来设计这段Shuffle过程，那么你的设计目标是什么。我想能优化的地方主要在于减少拉取数据的量及尽量使用内存而不是磁盘。

我的分析是基于Hadoop0.21.0的源码，如果与你所认识的Shuffle过程有差别，不吝指出。我会以WordCount为例，并假设它有8个map task和3个reduce task。从上图看出，Shuffle过程横跨map与reduce两端，所以下面我也会分两部分来展开。

先看看map端的情况，如下图：



上图可能是某个map task的运行情况。拿它与官方图的左半边比较，会发现很多不一致。官方图没有清楚地说明partition， sort与combiner到底作用在哪个阶段。我画了这张图，希望让大家清晰地了解从map数据输入到map端所有数据准备好的全过程。

整个流程我分了四步。简单些可以这样说，每个map task都有一个内存缓冲区，存储着map的输出结果，当缓冲区快满的时候需要将缓冲区的数据以一个临时文件的方式存放到磁盘，当整个map task结束后再对磁盘中这个map task产生的所有临时文件做合并，生成最终的正式输出文件，然后等待reduce task来拉数据。

当然这里的每一步都可能包含着多个步骤与细节，下面我对细节来一一说明：

1、在map task执行时，它的输入数据来源于HDFS的block，当然在MapReduce概念中，map task只读取split。Split与block的对应关系可能是多对一，默认是一对一。在WordCount例子里，假设map的输入数据都是像“aaa”这样的字符串。

2、在经过mapper的运行后，我们得知mapper的输出是这样一个key/value对：key是“aaa”， value是数值1。因为当前map端只做加1的操作，在reduce task里才去合并结果集。前面我们知道这个job有3个reduce

task，到底当前的“aaa”应该交由哪个reduce去做呢，是需要现在决定的。

MapReduce提供Partitioner接口，它的作用就是根据key或value及reduce的数量来决定当前的这对输出数据最终应该交由哪个reduce task处理。默认对key hash后再以reduce task数量取模。默认的取模方式只是为了平均reduce的处理能力，如果用户自己对Partitioner有需求，可以订制并设置到job上。

在我们的例子中，“aaa”经过Partitioner后返回0，也就是这对值应当交由第一个reducer来处理。接下来，需要将数据写入内存缓冲区中，缓冲区的作用是批量收集map结果，减少磁盘IO的影响。我们的key/value对以及Partition的结果都会被写入缓冲区。当然写入之前，key与value值都会被序列化成字节数组。

整个内存缓冲区就是一个字节数组，它的字节索引及key/value存储结构我没有研究过。如果有朋友对它有研究，那么请大致描述下它的细节吧。

3、这个内存缓冲区是有大小限制的，默认是100MB。当map task的输出结果很多时，就可能会撑爆内存，所以需要在一定条件下将缓冲区中的数据临时写入磁盘，然后重新利用这块缓冲区。这个从内存往磁盘写数据的过程被称为Spill，中文可译为溢写，字面意思很直观。这个溢写是由单独线程来完成，不影响往缓冲区写map结果的线程。溢写线程启动时不应该阻止map的结果输出，所以整个缓冲区有个溢写的比例spill.percent。这个比例默认是0.8，也就是当缓冲区的数据已经达到阈值（buffer size * spill percent = 100MB * 0.8 = 80MB），溢写线程启动，锁定这80MB的内存，执行溢写过程。Map task的输出结果还可以往剩下的20MB内存中写，互不影响。

当溢写线程启动后，需要对这80MB空间内的key做排序(Sort)。排序是MapReduce模型默认的行为，这里的排序也是对序列化的字节做的排序。

在这里我们可以想想，因为map task的输出是需要发送到不同的reduce端去，而内存缓冲区没有对将发送到相同reduce端的数据做合并，那么这种合并应该是体现是磁盘文件中的。从官方图上也可以看到写到磁盘中的溢写文件是对不同的reduce端的数值做过合并。所以溢写过程一个很重要的细节在于，如果有很多个key/value对需要发送到某个reduce端去，那么需要将这些key/value值拼接到一块，减少与partition相关的索引记录。

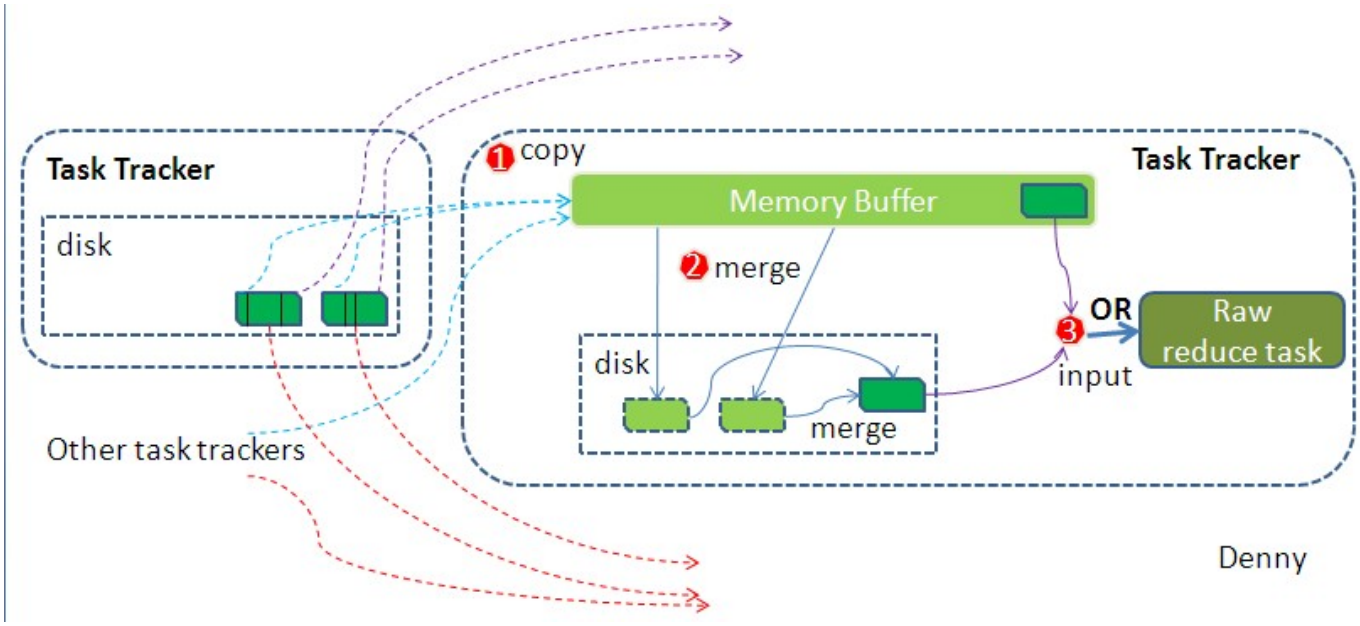
在针对每个reduce端而合并数据时，有些数据可能像这样：“aaa”/1，“aaa”/1。对于WordCount例子，就是简单地统计单词出现的次数，如果在同一个map task的结果中有很多个像“aaa”一样出现多次的key，我们就应该把它们的值合并到一块，这个过程叫reduce也叫combine。但MapReduce的术语中，reduce只指reduce端执行从多个map task取数据做计算的过程。除reduce外，非正式地合并数据只能算做combine了。其实大家知道的，MapReduce中将Combiner等同于Reducer。

如果client设置过Combiner，那么现在就是使用Combiner的时候了。将有相同key的key/value对的value加起来，减少溢写到磁盘的数据量。Combiner会优化MapReduce的中间结果，所以它在整个模型中会多次使用。那哪些场景才能使用Combiner呢？从这里分析，Combiner的输出是Reducer的输入，Combiner绝不能改变最终的计算结果。所以从我的想法来看，Combiner只应该用于那种Reduce的输入key/value与输出key/value类型完全一致，且不影响最终结果的场景。比如累加，最大值等。Combiner的使用一定得慎重，如果用好，它对job执行效率有帮助，反之会影响reduce的最终结果。

4、每次溢写会在磁盘上生成一个溢写文件，如果map的输出结果真的很大，有多次这样的溢写发生，磁盘上相应的就会有多个溢写文件存在。当map task真正完成时，内存缓冲区中的数据也全部溢写到磁盘中形成一个溢写文件。最终磁盘中会至少有一个这样的溢写文件存在(如果map的输出结果很少，当map执行完成时，只会产生一个溢写文件)，因为最终的文件只有一个，所以需要将这些溢写文件归并到一起，这个过程就叫做Merge。Merge是怎样的？如前面的例子，“aaa”从某个map task读取过来时值是5，从另外一个map 读取时值是8，因为它们有相同的key，所以得merge成group。什么是group。对于“aaa”就是像这样的：{“aaa”, [5, 8, 2, ...]}，数组中的值就是从不同溢写文件中读取出来的，然后再把这些值加起来。请注意，因为merge是将多个溢写文件合并到一个文件，所以可能也有相同的key存在，在这个过程中如果client设置过Combiner，也会使用Combiner来合并相同的key。

至此，map端的所有工作都已结束，最终生成的这个文件也存放在TaskTracker够得着的某个本地目录内。每个reduce task不断地通过RPC从JobTracker那里获取map task是否完成的信息，如果reduce task得到通知，获知某台TaskTracker上的map task执行完成，Shuffle的后半段过程开始启动。

简单地说，reduce task在执行之前的工作就是不断地拉取当前job里每个map task的最终结果，然后对从不同地方拉取过来的数据不断地做merge，也最终形成一个文件作为reduce task的输入文件。见下图：



如map 端的细节图，Shuffle在reduce端的过程也能用图上标明的三点来概括。当前reduce copy数据的前提是它要从JobTracker获得有哪些map task已执行结束，这段过程不表，有兴趣的朋友可以关注下。Reducer真正运行之前，所有的时间都是在拉取数据，做merge，且不断重复地在做。如前面的方式一样，下面我也分段地描述reduce 端的Shuffle细节：

1、Copy过程，简单地拉取数据。Reduce进程启动一些数据copy线程(Fetcher)，通过HTTP方式请求map task所在的TaskTracker获取map task的输出文件。因为map task早已结束，这些文件就归TaskTracker管理在本地磁盘中。

2、Merge阶段。这里的merge如map端的merge动作，只是数组中存放的是不同map端copy来的数值。Copy过来的数据会先放入内存缓冲区中，这里的缓冲区大小要比map端的更为灵活，它基于JVM的heap size设置，因为Shuffle阶段Reducer不运行，所以应该把绝大部分的内存都给Shuffle用。这里需要强调的是，merge有三种形式：1)内存到内存 2)内存到磁盘 3)磁盘到磁盘。默认情况下第一种形式不启用，让人比较困惑，是吧。当内存中的数据量到达一定阈值，就启动内存到磁盘的merge。与map 端类似，这也是溢写的过程，这个过程中如果你设置有Combiner，也是会启用的，然后在磁盘中生成了众多的溢写文件。第二种merge方式一直在运行，直到没有map端的数据时才结束，然后启动第三种磁盘到磁盘的merge方式生成最终的那个文件。

3、Reducer的输入文件。不断地merge后，最后会生成一个“最终文件”。为什么加引号？因为这个文件可能存在于磁盘上，也可能存在于内存中。对我们来说，当然希望它存放于内存中，直接作为Reducer的输入，但默认情况下，这个文件是存放于磁盘中的。至于怎样才能让这个文件出现在内存中，之后的性能优化篇我再说。当Reducer的输入文件已定，整个Shuffle才最终结束。然后就是Reducer执行，把结果放到HDFS上。

本文转载自：http://langyu.iteye.com/blog/992916