学习 Hadoop 有一段时间了，前几天完成老师的实验室用 Hadoop 实现后觉得很简单，可是真到集群上运行的时候各种问题都一下子出来了，比如中间结果太大，没有使用 ChainMapper 而去一下子开多个 Job。所以说确定一个 Map 和 Reduce 函数不难的，难的地方在于数据的组织和处理。下面我把在网上看到的一些关于 hadoop 编程优化的东西。

1. 自定义 Writable

Hadoop 自带的 Text 很好用，同时字符串转换的时候会消耗大量资源，所以还是根据自己的需要自定义合适的 Writable，注意作为 Key 时要实现 WritableCompareable

不要用 output.collect(new Text( ),new Text())

而采用 key.set( ) value.set( ) output.collect(key,value)

前者会产生大量的 Text 对象，使用完后 Java 垃圾回收器会花费大量的时间去收集这些对象

2. 使用 StringBuilder

不要使用 Formatter StringBuffer（这个是线程安全）

不要介意使用+ 在使用 StringBuffer 的时候尽量少使用多个 append 方法，而是在一个 append 内使用+连接各个字符串

3. 使用 DistributedCache 加载文件，比如配置文件，词典，共享文件，而不要使用 static 变量

4. 充分使用 Combiner Parttitioner Comparator。

Combiner：可以减少 Map 端和 Reduce 端数据传输量，一般和 reduce 相同

Parttitioner ： 设计合适的 Parttitioner 可避免将 map 结果按 key 只分配到一个或者少数 Reduce 中，造成负载不平衡

Comparator ： 二次排序，先按 key 排序，再按 value 排序

比如 Hadoop 权威指南中的求每天的最大气温，map 结果为日期：气温列表

如果气温列表是降序的，那么就不需要遍历整个气温列表，直接取第一个值即可

5. 根据自己的需要灵活实现自己的 InputFormat 和 OutputFormat

6. MapReduce 编程中应该避免的事情

静态变量：不能用于计数，应该使用 Counter，有系统自带的和自定义的，也可分为静态和

动态。平时调试 MR 程序时也要很好的结合 Counter

大对象：比如 Map List 尽量做为 Map 和 Reduce 的成员变量

大循环：这和上面的是对应的，我都中枪了

递归：避免递归的深度过大

超长正则表达式：消耗性能，要在 map 或 reduce 函数外编译正则表达式

不要创建本地文件，其实是一种变向的把 HDFS 里面的数据转移到 TaskTracker，占用网络带宽

不要大量创建目录和文件，write create mkdir

不要大量使用 System.out.println，而使用 Logger

不要自定义过多的 Counter，最好不要超过 100 个

不要配置过大内存，mapred.child.java.opts -Xmx2000m 是用来设置 mapreduce 任务使用的最大 heap 量，关于具体的参数调优下面也有会讲。

7.关于 map 的数目

map 数目过大，一般是由大量小文件造成的，或者 dfs.block.size 设置的太小，如果一个 Job 只有 map 任务，那么会产生大量的小文件，这些小文件可以 Archive 文件或者 Hadoop fs -merge.此外如果太多的 map 任务数目，而且每个 map 的生命周期又很短，那么创建和初始化 map 的开销相对于执行来说就有点大了。那么如果 map 的任务数又太少呢？？这样会造成单个 hadoop 的时间过长，频繁推测执行，而且还容易内存不足，Hadoop 的并行性优势也不能提现出来。这样的话就要根据具体情况决定合适的 map 数目，dfs.block.size 一般为 256M-512M，具体设置多少可参考 http://genie13.iteye.com/admin/blogs/1504114

对于压缩的 Text 文件是不能被分割的，所以尽量使用 Sequence File，可以切分，map 数目只和文件大小有关，而和文件数目无关。

关于小文件的处理方案可参考：http://genie13.iteye.com/admin/blogs/1504130

8.关于 reduce 的数目

reduce 数目过大，会产生大量的小文件，消耗大量不必要的资源，reduce 数目过低呢，和 map 有着同样的问题，而且会造成数据倾斜问题，通常不能通过修改参数改变。最佳方案是：mapred.reduce.tasks 设为-1 以变成 AutoReduce。

Key 的分布，也在某种程度上决定了 Reduce 数目，所以要根据 Key 的特点设计相对应的

Parttitioner 避免数据倾斜。

关于最大 map 和 reduce 数目可参考：http://genie13.iteye.com/admin/blogs/1535474

9.Map-side 相关参数优化

io.sort.mb（100MB）：通常 m 诶个 map 任务都会对应一个内存 buffer，内存 buffer 主要用来缓存 map 部分计算结果，并做一些预排序工作来提高 map 性能，如果 map 产生的结果比较大，那么可以调高这个参数，减少 map 任务进行 spill 任务的个数，从而降低对 DISK I/O 的操作。如果 map 任务的瓶颈在 I/O 的话，那么将会大大提高 map 性能。那么如何判断 map 任务的瓶颈呢？？具体可参考：http://blog.csdn.net/ae86_fc/article/details/5957793

io.sort.spill.percent(0.8)：那么什么叫 spill 操作呢？当内存 buffer 超过一定阈值（这里通常是百分比）的时候，会将 buffer 中得数据写到 Disk 中，这个过程就叫 spill。而不是等 buffer 满后在 spill，这样的话会造成 map 的计算任务等待 buffer 空间的释放。通常情况下是调整 io.sort.mb 而不是这个参数。

io.sort.factor（10）：从上面可知 map 任务会产生很多的 spill 文件，而 map 任务在正常退出之前会将这些 spill 合并成一个文件，即 map 结束之前的 merger 过程，默认情况下一次是合并 10 个参数，调大 io.sort.factor，会减少 merge 的次数，减少 Disk I/O 操作，提高 map 性能。

min.num.spill.for.combine：通常为了减少 map 和 reduce 数据传输量，我们会制定一个 Combiner，将 map 结果进行本地聚集。这里运行 Combiner 可能在 merger 之前，也可能在其之后。那么什么时候在其之前呢？？当 spill 个数至少为 min.num.spill.for.combine 指定的数目时同时程序指定了 Combiner，Combiner 会在其之前运行，以此来进行本地聚集，减少写入到 Disk 的数据量，减少读写频率。

10.压缩

我们知道 MR 中的数据无论是中间数据还是最后结果通常都是巨大的，如果不使用压缩的话不仅浪费磁盘空间而且会大量消耗网络带宽。无论是在 map 的中间结果，spill 结果，merge 结果（reduce 也对有一个 merge 操作），还是 reduce 结果我们都可以使用压缩。

我们可以通过 mapred.compress.map.output 决定是否使用压缩

通过 mapred.compress.output.compression.codec 指定一个压缩方式，如果想在 cpu 时间和压缩比之间寻找一个平衡，LzoCodec 比较适合。

这里需要注意的是使用压缩会相应的增加 cpu 时间用来解压和压缩，不过通常 MR 任务的瓶颈不在 CPU 而在于 I/O，所以大部分的 MR 任务都适合使用压缩。

相关可参考：http://blog.csdn.net/AE86_FC/article/details/5284230

11. reduce-side 相关参数优化

reduce 的过程 copy->sort->reduce，也称为 shuffle 过程。

mapred.reduce.parellel.copies（5）：因为每一个 map 任务可能包含一个或者多个 reduce 所需要的数据，所以一旦一个 map 任务完成后，相应的 reduce 就会下载自己所需要的数据。我们可以通过这个参数来指定并行下载的数量，调大这个参数比较适合 map 任务比较多且完成时间比较短的 Job。

mapred.reduce.copy.backoff：reduce 端从 map 端下载数据也有可能由于网络故障，map 端机器故障而失败。那么 reduce 下载线程肯定不会无限等待，当等待时间超过 mapred.reduce.copy.backoff 时，reduce 会放弃从这个下载，而尝试从其他地方进行下载。这里需要注意的是，在网络情况比较差的环境，我们需要调大这个参数，防止避免 reduce 下载线程被误判为失败的情况。

io.sort.factor：recude 将 map 结果下载到本地是，也是需要 merge，这里的情况类似于 map 端，可参考 9，如果 reduce 的瓶颈在于 I/O，可尝试调高增加 merge 的并发吞吐，增强 reduce 性能。、

mapred.job.shuffle.input.buffer.percent（0.7）：Reduce 从 map 下载的数据不会立刻就写到 Disk 中，而是先缓存在内存中，mapred.job.shuffle.input.buffer.percent 指定内存的多少比例用于缓存数据，其中的内存大小可通过 mapred.child.java.opts 来设置。举个例子，mapred.job.shuffle.input.buffer.percent 是 0.7，mapred.child.java.opts 是 1G，也就是说大概 700M 用户缓存数据。这 700M 数据和 map 端类似，也不是等到写满才往磁盘中写，也是到达一种阈值就往 Disk 写，这个阈值可由 mapred.job,shuffle.merge.percent 来制定。如果 Reduce 下载速度很快的话，容易内存溢出，那么适当增大这个参数对增加 reduce 性能有些帮助。

mapred.job.reduce.input.buffer.percent (0)

当 Reduce 下载 map 数据完成之后，就会开始真正的 reduce 的计算过程，reduce 的计算必然也是要消耗内存的，那么在读物 reduce 所需要的数据时，同样需要内存作为 buffer，这个参数是决定多少的内存百分比作为 buffer。默认为 0，也就是说 reduce 全部从磁盘读数据。如果 redcue 计算任务消耗内存很小，那么可以设置这个参数大于 0，使一部分内存用来缓存数据。

另外附上两个网址，也是讲 Hadoop 相关优化的

1.http://dongxicheng.org/mapreduce/hadoop-optimization-0/

2.http://www.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/