spark Streaming的saveAsHadoopFiles等方法落地时会不会有小文件问题？

背景简述：对海量日志(每小时TB级)进行汇集与校验。日志会存在大量小文件（部分低于100M一个数量级），Streaming接收kafka发送过来的文件名后进行读取校验，接着用saveAsHadoopFiles等方法落地到hdfs。

问题1：如果Streaming的源为sc.textFile(file, 2)得到的，那最终saveAsHadoopFiles(prefix, [suffix])等方法内部是如何做到的？是对每个输入文件保存为Receiver的n个hdfs文件吗？

问题2：如果Streaming的源是类似于flume的动态源，那saveAsHadoopFiles等方法保存文件到hdfs中时内部是如何实现的？会不会存在小文件的问题？

问题3：如果小文件问题还是存在的话，Streaming一般如何解决这些小文件的问题呢？

使用 Spark Streaming 时，如果实时计算结果要写入到 HDFS，那么不可避免的会遇到一个问题，那就是在默认情况下会产生非常多的小文件，这是由 Spark Streaming 的微批处理模式和 DStream(RDD) 的分布式(partition)特性导致的，Spark Streaming 为每个 Partition 启动一个独立的线程来处理数据，一旦文件输出到 HDFS，那么这个文件流就关闭了，再来一个 batch 的 parttition 任务，就再使用一个新的文件流。

那么假设，一个 batch 为10s，每个输出的 DStream 有32个 partition，那么一个小时产生的文件数将会达到(3600/10)\*32=11520个之多。

众多小文件带来的结果是有大量的文件元信息，比如文件的 location、文件大小、block number 等需要 NameNode 来维护，NameNode 会因此鸭梨山大。不管是什么格式的文件，parquet、text、JSON 或者 Avro，都会遇到这种小文件问题，这里讨论几种处理 Spark Streaming 小文件的典型方法。

1. 增加 batch 大小:

这种方法很容易理解，batch 越大，从外部接收的 event 就越多，内存积累的数据也就越多，那么输出的文件数也就回变少，比如上边的时间从10s增加为100s，那么一个小时的文件数量就会减少到1152个。但别高兴太早，实时业务能等那么久吗，本来人家10s看到结果更新一次，现在要等快两分钟，是人都会骂娘。所以这种方法适用的场景是消息实时到达，但不想挤压在一起处理，因为挤压在一起处理的话，批处理任务在干等，这时就可以采用这种方法。

2. Coalesce大法好:

文章开头讲了，小文件的基数是 batch\_number \* partition\_number，而第一种方法是减少 batch\_number，那么这种方法就是减少 partition\_number 了，这个 api 不细说，就是减少初始的分区个数。看过 spark 源码的童鞋都知道，对于窄依赖，一个子 RDD 的 partition 规则继承父 RDD，对于宽依赖(就是那些个叉叉叉ByKey操作)，如果没有特殊指定分区个数，也继承自父 rdd。那么初始的 SourceDstream 是几个 partiion，最终的输出就是几个 partition。所以 Coalesce 大法的好处就是，可以在最终要输出的时候，来减少一把 partition 个数。但是这个方法的缺点也很明显，本来是32个线程在写256M数据，现在可能变成了4个线程在写256M数据，而没有写完成这256M数据，这个 batch 是不算做结束的。那么一个 batch 的处理时延必定增长，batch 挤压会逐渐增大。

3. SparkStreaming外部来处理: 另外启动MR任务合并小文件【遍历合并】

我们既然把数据输出到 hdfs，那么说明肯定是要用 Hive 或者 Spark Sql 这样的“sql on hadoop”系统类进一步进行数据分析，而这些表一般都是按照半小时或者一小时、一天，这样来分区的(注意不要和 Spark Streaming 的分区混淆，这里的分区，是用来做分区裁剪优化的)，那么我们可以考虑在 Spark Streaming 外再启动定时的批处理任务来合并 Spark Streaming 产生的小文件。这种方法不是很直接，但是却比较有用，“性价比”较高，唯一要注意的是，批处理的合并任务在时间切割上要把握好，搞不好就可能回去合并一个还在写入的 Spark Streaming 小文件。

4. 自己调用 foreach 去 append:

Spark Streaming 提供的 foreach 这个 outout 类 api，可以让我们自定义输出计算结果的方法。那么我们其实也可以利用这个特性，那就是每个 batch 在要写文件时，并不是去生成一个新的文件流，而是把之前的文件打开。考虑这种方法的可行性，首先，HDFS 上的文件不支持修改，但是很多都支持追加，那么每个 batch 的每个 partition 就对应一个输出文件，每次都去追加这个 partition 对应的输出文件，这样也可以实现减少文件数量的目的。这种方法要注意的就是不能无限制的追加，当判断一个文件已经达到某一个阈值时，就要产生一个新的文件进行追加了。