黑客派

MapReduce 概述

作者：ibywind

原文链接：https://hacpai.com/article/1580301635611

来源网站：黑客派

<h2 id="一-MapReduce-概述">一、MapReduce 概述</h2>

Hadoop MapReduce 是一个分布式计算框架，用于编写批处理应用程序。编写好的程序可以提到 Hadoop 集群上用于并行处理大规模的数据集。
MapReduce 作业通过将输入的数据集拆分为独立的块，这些块由 <code>map</code> 以并的方式处理，框架对 <code>map</code> 的输出进行排序，然后输入到 <code>reduce</code> 中。MapReduce 框架专门用于 <code><key，value></code> 键值对处理，它将作业的输视为一组 <code><key，value></code> 对，并生成一组 <code><key，value></cod> 对作为输出。输出和输出的 <code>key</code> 和 <code>value</code> 都必须实现 <a href "https://link.hacpai.com/forward?goto=http%3A%2F%2Fhadoop.apache.org%2Fdocs%2Fstabe%2Fapi%2Forg%2Fapache%2Fhadoop%2Fio%2FWritable.html" target="_blank" rel="nofollw ugc">Writable 接口。
<pre><code class="highlight-chroma">(input) <k1, v1> -> map -> <k2, v2> > combine -> <k2, v2> -> reduce -> <k3, v3> (output)
</code></pre>
<h2 id="二-MapReduce-编程模型简述">二、MapReduce 编程模型简述</h2>
这里以词频统计为例进行说明，MapReduce 处理的流程如下：

 input：读取文本文件；
 splitting：将文件按照行进行拆分，此时得到的 <code>K1</code> 行，<code>V1</code> 表示对应行的文本内容；
 mapping：并行将每一行按照空格进行拆分，拆分得到的 <code>List(K2,2)</code>，其中 <code>K2</code> 代表每一个单词，由于是做词频统计，所以 <code>V2</code> 的值为 1，代表出现 1 次；
 shuffling：由于 <code>Mapping</code> 操作可能是在不同的机器上行处理的，所以需要通过 <code>shuffling</code> 将相同 <code>key</code> 值的数据分发到一个节点上去合并，这样才能统计出最终的结果，此时得到 <code>K2</code> 为每一个单词，<code>List(V2)</code> 为可迭代集合，<code>V2</code> 就是 Mapping 中的 V2；
 Reducing：这里的案例是统计单词出现的总次数，所以 <code>Reducin</code> 对 <code>List(V2)</code> 进行归约求和操作，最终输出。

MapReduce 编程模型中 <code>splitting</code> 和 <code>shuffing</code> 操作都是由架实现的，需要我们自己编程实现的只有 <code>mapping</code> 和 <code>reducing</code>，这也就是 MapReduce 这个称呼的来源。
<h2 id="三-combiner---partitioner">三、combiner & partitioner</h2>

img src="https://static.hacpai.com/image
/img-loading.svg" alt="" data-src="https://github.com/heibaiying/BigData-Notes/raw/master
pictures/Detailed-Hadoop-MapReduce-Data-Flow-14.png">
<h3 id="-">[</h3>

](https://github.com/heibaying/BigData-Notes/blob/master/notes/Hadoop-MapReduce.md#31-inputformat--recordreaders)3.1 InputFormat & RecordReaders
<code>InputFormat</code> 将输出文件拆分为多个 <code>InputSplit</code>，并由 <code>RecordReaders</code> 将 <code>InputSplit</code> 转换为标准的 <key，value> 键对，作为 map 的输出。这一步的意义在于只有先进行逻辑拆分并转为标准的键值对格式后，才能为个 <code>map</code> 提供输入，以便进行并行处理。
<h3 id="3-2-Combiner">3.2 Combiner</h3>
<code>combiner</code> 是 <code>map</code> 运算后的可选操作，它实际上是一个本化的 <code>reduce</code> 操作，它主要是在 <code>map</code> 计算出中间文件后做一个单的合并重复 <code>key</code> 值的操作。这里以词频统计为例：
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<code>map</code> 在遇到一个 hadoop 的单词时就会记录为 1，但是这篇文章里 hadoop 能会出现 n 多次，那么 <code>map</code> 输出文件冗余就会很多，因此在 <code>reduce</code> 计算前对相同的 key 做一个合并操作，那么需要传输的数据量就会减少，传输效率就可以得到提。
但并非所有场景都适合使用 <code>combiner</code>，使用它的原则是 <code>combiner</code> 的输出不会影响到 <code>reduce</code> 计算的最终输入，例如：求总数，最大值，最小时都可以使用 <code>combiner</code>，但是做平均值计算则不能使用 <code>combiner</code>。
不使用 combiner 的情况：

使用 combiner 的情况：

可以看到使用 combiner 的时候，需要传输到 reducer 中的数据由 12keys，降低到 10keys。低的幅度取决于你 keys 的重复率，下文词频统计案例会演示用 combiner 降低数百倍的传输量。</>
<h3 id="3-3-Partitioner">3.3 Partitioner</h3>
<code>partitioner</code> 可以理解成分类器，将 <code>map</code> 的输出按照 key 的不同分别分给对应的 <code>reducer</code>，支持自定义实现，下文案例会给出演示。
<h2 id="四-MapReduce-词频统计案例">四、MapReduce 词频统计案例</h2>
<h3 id="4-1-项目简介">4.1 项目简介</h3>
这里给出一个经典的词频统计的案例：统计如下样本数据中每个单词出现的次数。
<pre>Spark	HBase
Hive	Flink	Storm	Hadoop	HBase	Spark

Flink
HBase Storm
HBase Hadoop Hive Flink
HBase Flink Hive Storm
Hive Flink Hadoop
HBase Hive
Hadoop Spark HBase Storm
HBase Hadoop Hive Flink
HBase Flink Hive Storm
Hive Flink Hadoop
HBase Hive</pre>
为方便大家开发，我在项目源码中放置了一个工具类 <code>WordCountDataUtils</code> 用于模拟产生词频统计的样本，生成的文件支持输出到本地或者直接写到 HDFS 上。
<h3 id="4-2-项目依赖">4.2 项目依赖</h3>
想要进行 MapReduce 编程，需要导入 <code>hadoop-client</code> 依赖：
<pre><dependency>
<groupId>org.apache.hadoopgroupId>
<artifactId>hadoop-clientartifactId>
<version>${hadoop.version}version>
dependency></pre>
<h3 id="4-3-WordCountMapper">4.3 WordCountMapper</h3>
将每行数据按照指定分隔符进行拆分。这里需要注意在 MapReduce 中必须使用 Hadoop 定义类型，因为 Hadoop 预定义的类型都是可序列化，可比较的，所有类型均实现了 <code>WritableCmparable</code> 接口。
<pre>public class WordCountMapper <spa>extends Mapper<LongWritable, Text, <pan>Text, IntWritable> {

}</pre>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr pt>

<!-- 黑客派PC帖子内嵌-展示 -->

<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in >
<script>
(adsbygoogle = window.adsbygoogle || []).push({});
</script>
<code>WordCountMapper</code> 对应下图的 Mapping 操作：
<img src="https://static.hacpai.com/images/img-loading.svg" al ="" data-src="https://github.com/heibaiying/BigData-Notes/raw/master/pictures/hadoop-co e-mapping.png">
<code>WordCountMapper</code> 继承自 <code>Mappe</code> 类，这是一个泛型类定义如下：
<pre>WordCountMapper extends Mapper<LongWritable< span>, Text, Text, IntWritable>

public class Mapper {
.....
}</pre>

 KEYIN：<code>mapping</code> 输入 key 的类型，即每行的偏移量 (行第一个字符在整个文本中的位置)，<code>Long</code> 类型，对应 Hadoop 中的 <code>LonWritable</code> 类型；
 VALUEIN：<code>mapping</code> 输入 value 的类型，即每行数据 <code>String</code> 类型，对应 Hadoop 中 <code>Text</code> 类型；
 KEYOUT： <code>mapping</code> 输出的 key 的类型，即每个单词 <code>String</code> 类型，对应 Hadoop 中 <code>Text</code> 类型；
 VALUEOUT： <code>mapping</code> 输出 value 的类型，即每个单出现的次数；这里用 <code>int</code> 类型，对应 <code>IntWritable</code> 类型。

<h3 id="4-4-WordCountReducer">4.4 WordCountReducer</h3>
在 Reduce 中进行单词出现次数的统计：
<pre>public class WordCountReducer <spn>extends Reducer<Text, IntWritable, <san>Text, IntWritable> {

}</pre>

如下图，<code>shuffling</code> 的输出是 reduce 的输入。这里的 key 是每个单词，values 是一个可迭代的数据类型，类似 <code>(1,1,1,...)</code>。
<img src="https://static.hacpai.com/images/img-loading.svg" al="" data-src="https://github.com/heibaiying/BigData-Notes/raw/master/pictures/hadoop-coe-reducer.png">
<h3 id="4-4-WordCountApp">4.4 WordCountApp</h3>
组装 MapReduce 作业，并提交到服务器运行，代码如下：
<pre>/**
 * 组装作业 并提交到集群运行
 */
public class WordCountApp {

}</pre>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>

<!-- 黑客派PC帖子内嵌-展示 -->

<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>

需要注意的是：如果不设置 <code>Mapper</code> 操作的输出类型，则程序默认它和 <cod
>Reducer</code> 操作输出的类型相同。
<h3 id="4-5-提交到服务器运行">4.5 提交到服务器运行</h3>
在实际开发中，可以在本机配置 hadoop 开发环境，直接在 IDE 中启动进行测试。这里主要介
一下打包提交到服务器运行。由于本项目没有使用除 Hadoop 外的第三方依赖，直接打包即可：

<pre># mvn clean package</pre>
使用以下命令提交作业：
<pre>hadoop jar /usr/appjar/hadoop-word-count-1.0.jar \
com.heibaiying.WordCountApp \
/wordcount/input.txt /wordcount/output/WordCountApp</pre>
作业完成后查看 HDFS 上生成目录：
<pre># 查看目录
hadoop fs -ls /wordcount/output/WordCountApp

查看统计结果
hadoop fs -cat /wordcount/output/WordCountApp/part-r-00000</pre>

<a href="https://link.hacpai.com/forward?goto=https%3A%2F%2Fgithub.com%2Fheibaiy
ng%2FBigData-Notes%2Fblob%2Fmaster%2Fpictures%2Fhadoop-wordcountapp.png" target
"_blank" rel="nofollow ugc"><img src="https://static.hacpai.com/images/img-loading.svg" al
="" data-src="https://github.com/heibaiying/BigData-Notes/raw/master/pictures/hadoop-wo
dcountapp.png">
<h2 id="五-词频统计案例进阶之-Combiner">五、词频统计案例进阶之 Combiner</h2>
<h3 id="5-1-代码实现">5.1 代码实现</h3>
想要使用 <code>combiner</code> 功能只要在组装作业时，添加下面一行代码即可：
<pre>// 设置 Combiner
job.setCombinerClass(WordCountReducer.c
ass);</pre>
<h3 id="5-2-执行结果">5.2 执行结果</h3>
加入 <code>combiner</code> 后统计结果是不会有变化的，但是可以从打印的日志看出 <co
e>combiner</code> 的效果：
没有加入 <code>combiner</code> 的打印日志：
<a href="https://link.hacpai.com/forward?goto=https%3A%2F%2Fgithub.com%2Fheibaiy
ng%2FBigData-Notes%2Fblob%2Fmaster%2Fpictures%2Fhadoop-no-combiner.png" target="
blank" rel="nofollow ugc"><img src="https://static.hacpai.com/images/img-loading.svg" alt
"" data-src="https://github.com/heibaiying/BigData-Notes/raw/master/pictures/hadoop-no-
ombiner.png">
加入 <code>combiner</code> 后的打印日志如下：
<a href="https://link.hacpai.com/forward?goto=https%3A%2F%2Fgithub.com%2Fheibaiy
ng%2FBigData-Notes%2Fblob%2Fmaster%2Fpictures%2Fhadoop-combiner.png" target="_bl
nk" rel="nofollow ugc"><img src="https://static.hacpai.com/images/img-loading.svg" alt=""
ata-src="https://github.com/heibaiying/BigData-Notes/raw/master/pictures/hadoop-combin
r.png">
这里我们只有一个输入文件并且小于 128M，所以只有一个 Map 进行处理。可以看到经过 comb
ner 后，records 由 <code>3519</code> 降低为 <code>6</code>(样本中单词种类就只有 6 种
，在这个用例中 combiner 就能极大地降低需要传输的数据量。
<h2 id="六-词频统计案例进阶之-Partitioner">六、词频统计案例进阶之 Partitioner</h2>
<h3 id="6-1-默认的-Partitioner">6.1 默认的 Partitioner</h3>
这里假设有个需求：将不同单词的统计结果输出到不同文件。这种需求实际上比较常见，比如统
产品的销量时，需要将结果按照产品种类进行拆分。要实现这个功能，就需要用到自定义 <code>Part
tioner</code>。
这里先介绍下 MapReduce 默认的分类规则：在构建 job 时候，如果不指定，默认的使用的是 <

ode>HashPartitioner</code>：对 key 值进行哈希散列并对 <code>numReduceTasks</code> 取余。其实现如下：
<pre>public class HashPartitioner extends Partitioner<K, V> {

public int getPartition(K key, V value,
int numReduceTasks) {
return (key.hashCode() not found render function for node [type=NodeHTMLEntity, Tokens=&]not found render function for node [type=NodeHTMLEntity, Tokens=&] Integer
/span>MAX_VALUE) % numReduceTasks;
}

}</pre>

<h3 id="6-2-自定义-Partitioner">6.2 自定义 Partitioner</h3>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
这里我们继承 <code>Partitioner</code> 自定义分类规则，这里按照单词进行分类：
<pre>public class CustomPartitioner extends Partitioner<Text, IntWritable> {

}</pre>

在构建 <code>job</code> 时候指定使用我们自己的分类规则，并设置 <code>reduce</code> 的个数：
<pre>// 设置自定义分区规则
job.setPartitionerClass(CustomPartitioner.cls);
// 设置 reduce 个数
job.setNumReduceTasks(WordCountDataUtils.WORD_LIST.size());</pre>
<h3 id="6-3-执行结果">6.3 执行结果</h3>
执行结果如下，分别生成 6 个文件，每个文件中为对应单词的统计结果：

