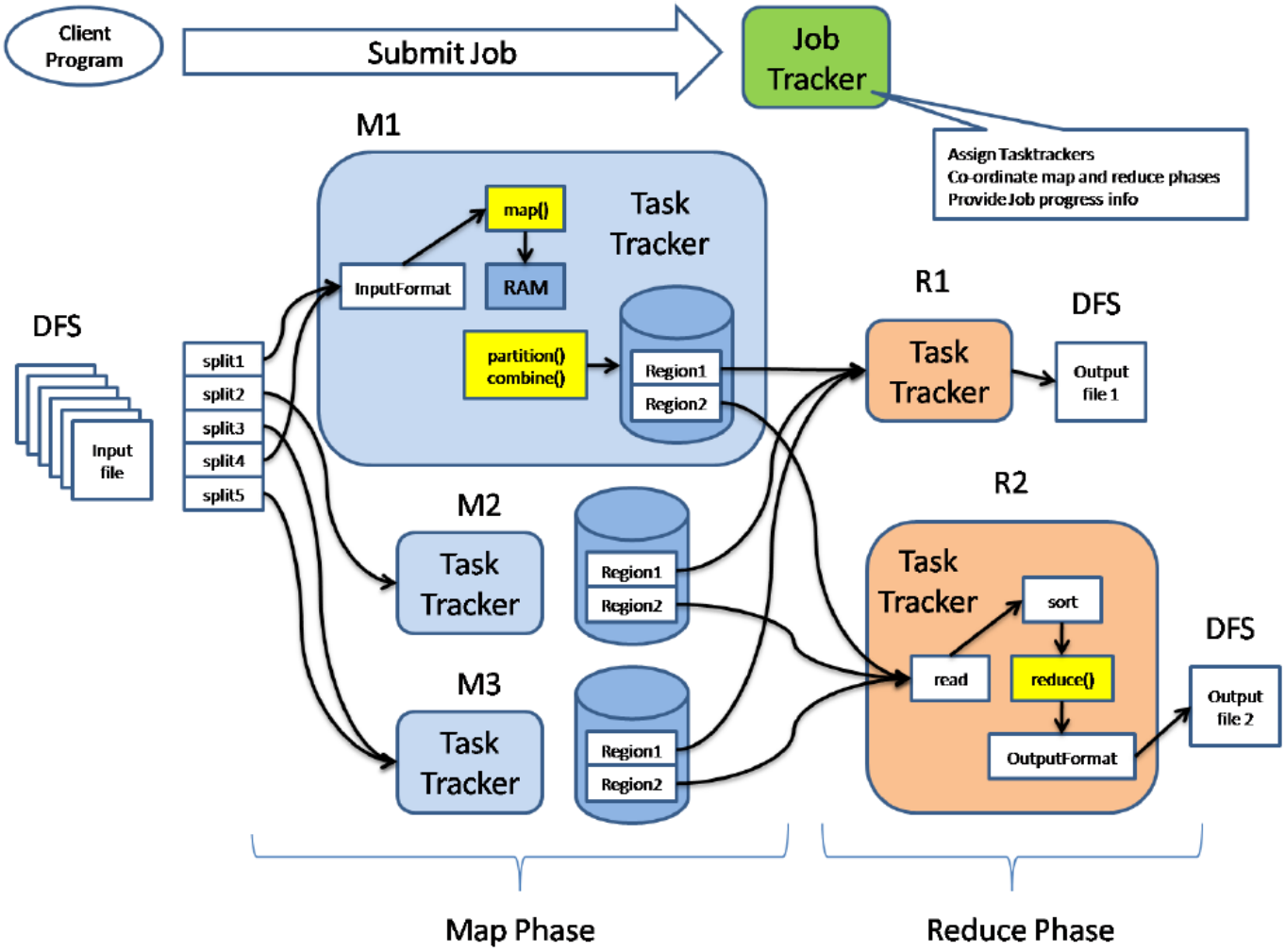
机器学习算法系列（32）：MapReduce执行流程详解

一、简述

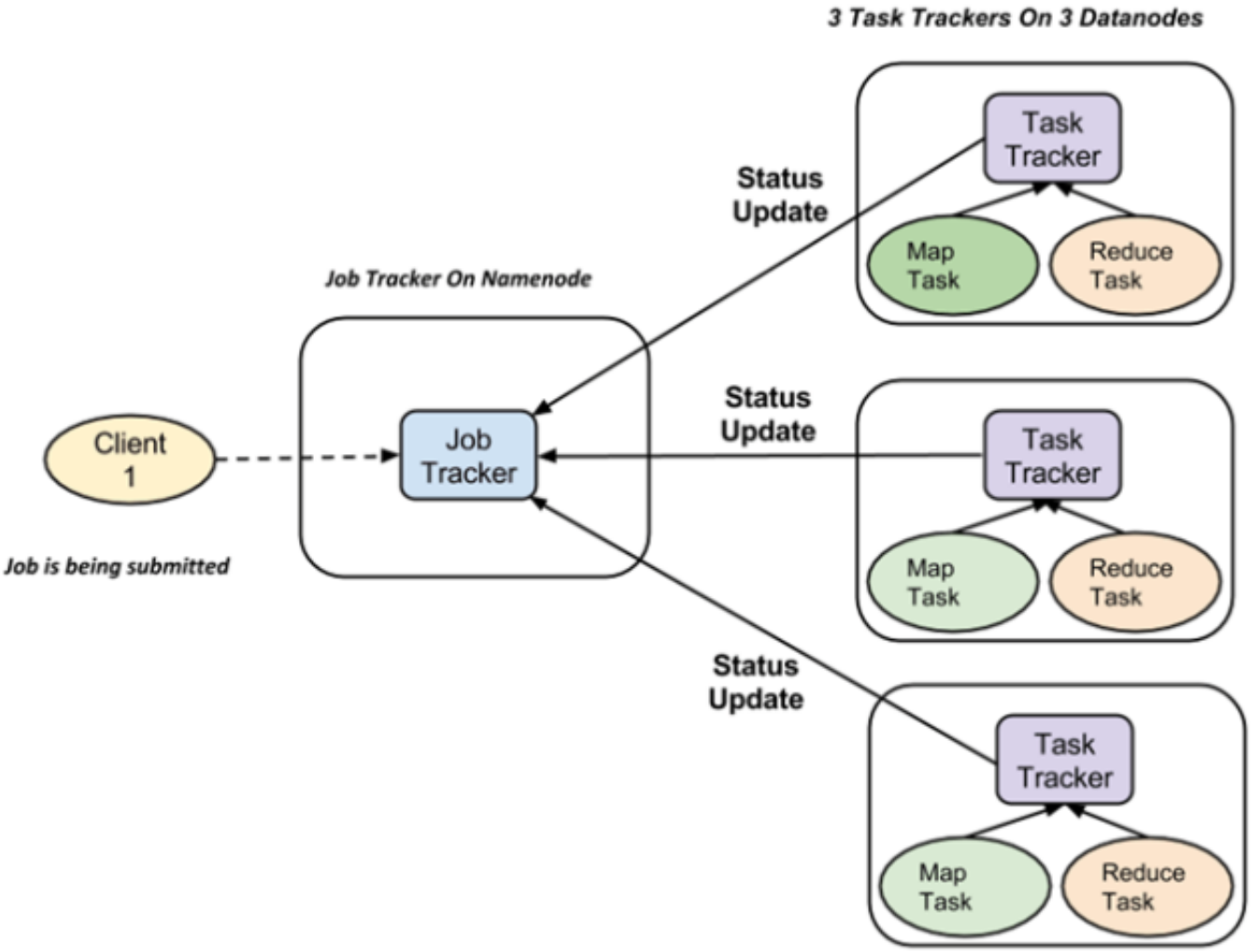
MapReduce最早是由Google提出的分布式数据处理模型，随后受到了业内的广泛关注，并被大量应用到各种商业场景中。比如：

- 搜索：网页爬取、倒排索引、PageRank。
- Web访问日志分析：分析和挖掘用户在web上的访问、购物行为特征，实现个性化推荐；分析用户访问行为。
- 文本统计分析：比如莫言小说的WordCount、词频TFIDF分析；学术论文、专利文献的引用分析和统计；维基百科数据分析等。
- 海量数据挖掘：非结构化数据、时空数据、图像数据的挖掘。
- 机器学习：监督学习、无监督学习、分类算法如决策树、SVM等。
- 自然语言处理：基于大数据的训练和预测；基于语料库构建单词同现矩阵，频繁项集数据挖掘、重复文档检测等。
- 广告推荐：用户点击（CTR)和购买行为（CVR）预测。

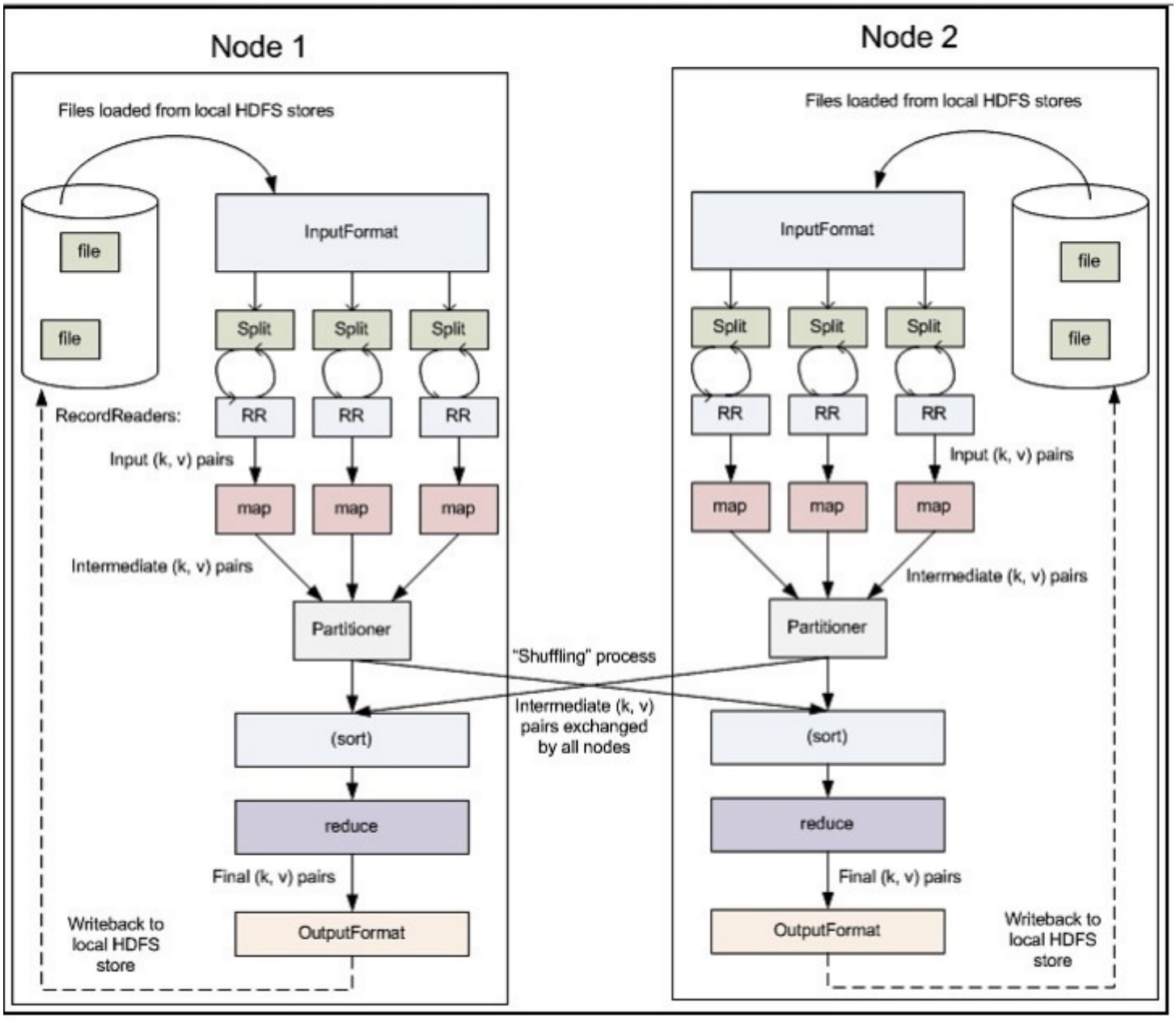
一个Map/Reduce作业（job）通常会把输入的数据（input file）且分为若干个独立的数据块（splits），然后由map任务（task）以完全并行的方式处理它们。Map/Reduce框架会对map的输出做一个Shuffle操作，Suffle操作后的结果会输入给reduce任务。整个Map/Reduce框架负责任务的调度和监控，以及重新执行已经失败的任务。



Map/Reduce计算集群由一个单独的JobTracker（master）和每个集群节点一个TaskTracker（slave）共同组成。JobTracker负责调度构成一个作业的所有任务，这些任务会被分派到不同的TaskTracker上去执行，JobTRacker会去监控它们的执行、重新执行已经失败的任务。而TaskTracker仅负责执行由JobTracker指派的任务。

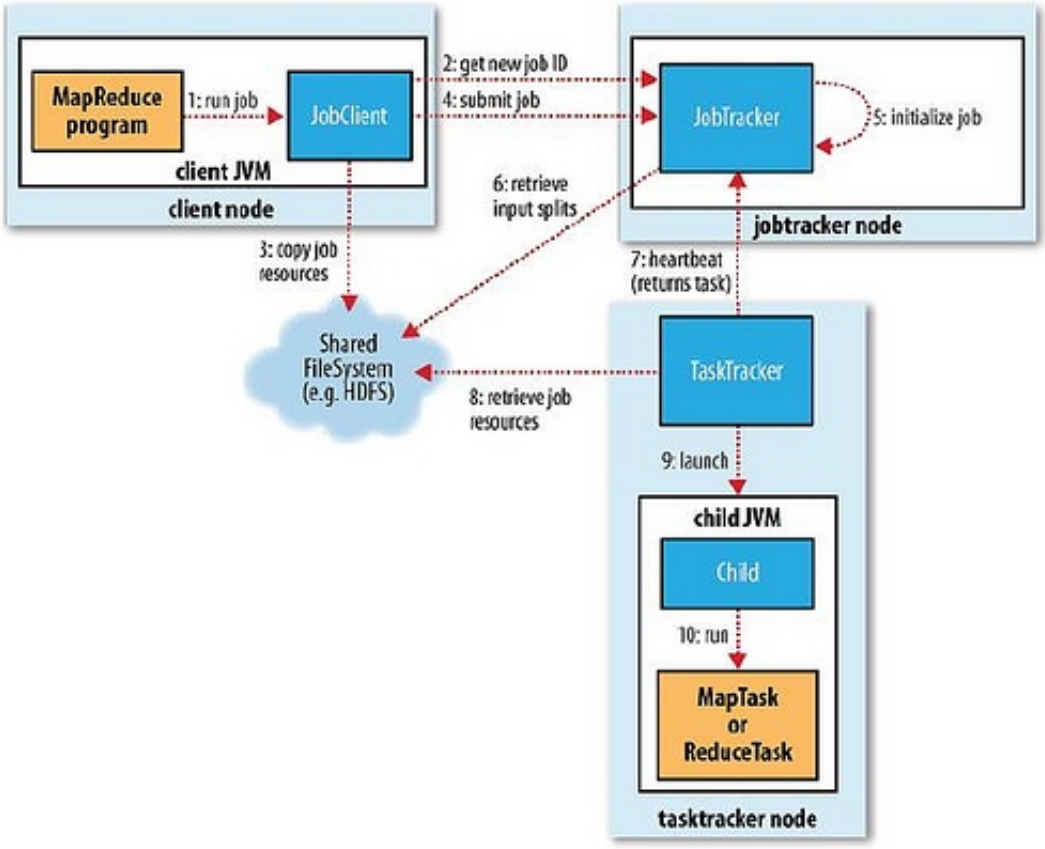


本文将按照map/reduce执行流程中各个任务的时间顺序详细叙述map/reduce的各个任务模块，包括：输入分片（input split）、map阶段、combiner阶段、shuffle阶段和reduce阶段。下图是一个不错的执行流程图：



二、作业的提交与监控

JobClient是用户提交的作业与JobTracker交互的主要接口。



JobClient提交作业的过程如下：

1. map/reduce程序通过runJob()方法新建一个JobClient实例;
2. 向JobTracker请求一个新jobID，通过JobTracker的getNewJobId()获取；
3. 检查作业输入输出说明。如果没有指定输出目录或者输出目录已经存在，作业将不会被提交，map/reduce程序； 输入作业划分split，如果划分无法计算（如：输入路径不存在），作业将不会被提交，错误返回给map/reduce程序。
4. 将运行作业所需要的资源（作业的jar文件、配置文件、计算所得的输入划分）复制到一个以作业ID命名的目录中；
5. 通过调用JobTracker的submitJob()方法，告诉JobTracker作业准备提交；
6. JobTracker将提交的作业放到一个内部队列中，交由作业调度器进行调度，并对其进行初始化。
7. 创建Map任务、Reduce任务：一个split对应一个map，有多少split就有多少map; Reduce任务的数量由JobConf的mapred.reduce.tasks属性决定
8. TaskTracker执行一个简单的循环，定期发送心跳（heartbeat）给JobTracker

三、执行流程

3.1 Input file

Input file是map/reduce任务的原始数据，一般存储在HDFS上。应用程序至少应该指明输入/输出

的位置（路径），并通过实现合适的接口或抽象类提供map和reduce函数。再加上其他作业的参数，就构成了作业配置（job configuration）。然后，Hadoop的 job client提交作业（jar包/可执行程序等）和配置信息给JobTracker，后者负责分发这些软件和配置信息给slave、调度任务并监控它们的执行，同时提供状态和诊断信息给job-client。

3.1.1 InputFormat

InputFormat为Map/Reduce作业输入的细节规范。Map/Reduce框架根据作业的InputFormat来：

1. 检查作业输入的正确性，如格式等。
2. 把输入文件切分成多个逻辑InputSplit实例， 一个InputSplit将会被分配给一个独立的Map任务。
3. 提供RecordReader实现，这个RecordReader从逻辑InputSplit中获得输入记录（”K-V对”），这些记录将由Map任务处理。

InputFormat有如下几种:

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

InputFormats provided by MapReduce

> TextInputFormat:

TextInputFormat是默认的INputFormat，输入文件中的每一行就是一个记录，Key是这一行的byte offset，而value是这一行的内容。如果一个作业的Inputformat是TextInputFormat，并且框架检测到输入文件的后缀是 .gz 和 .lzo，就会使用对应的CompressionCodec自动解压缩这些文件。但是需要注意，上述带后缀的压缩文件不会被切分，并且整个压缩文件会分给一个mapper来处理。

> KeyValueTextInputFormat

输入文件中每一行就是一个记录，第一个分隔符字符切分每行。在分隔符字符之前的内容为Key，在之后的为Value。分隔符变量通过key.value.separator.in.input.line变量设置，默认为(\t)字符。

> NLineInputFormat

与TextInputFormat一样，但每个数据块必须保证有且只有N行，mapred.line.input.format.linespermap属性，默认为1。

> SequenceFileInputFormat

一个用来读取字符流数据的InputFormat，为用户自定义的。字符流数据是Hadoop自定义的压缩的二进制数据格式。它用来优化从一个MapReduce任务的输出到另一个MapReduce任务的输入之间的数据传输过程。

3.2 输入分片（Input files）

InputSplit是一个单独的Map任务需要处理的数据块。一般的InputSplit是字节样式输入，然后由RecordReader处理并转化成记录样式。通常一个split就是一个block，这样做的好处是使得Map任务可以在存储有当前数据的节点上运行本地的任务，而不需要通过网络进行跨节点的任务调度。

可以通过设置mapred.min.split.size， mapred.max.split.size, block.size来控制拆分的大小。如果mapred.min.split.size大于block size，则会将两个block合成到一个split，这样有部分block数据需要通过网络读取；如果mapred.max.split.size小于block size，则会将一个block拆成多个split，增加了Map任务数。

在进行map计算之前，mapreduce会根据输入文件计算输入分片（input split），每个输入分片（input split）针对一个map任务，输入分片（input split）存储的并非数据本身，而是一个分片长度和一个记录数据的位置的数组，输入分片（input split）往往和hdfs的block（块）关系很密切，假如我们设定hdfs的块的大小是64mb，如果我们输入有三个文件，大小分别是3mb、65mb和127mb，那么mapreduce会把3mb文件分为一个输入分片（input split），65mb则是两个输入分片（input split）而127mb也是两个输入分片（input split），换句话说我们如果在map计算前做输入分片调整，例如合并小文件，那么就会有5个map任务将执行，而且每个map执行的数据大小不均，这个也是mapreduce优化计算的一个关键点。

```
输入文件大小                  10M     65M     127M
分割后的InputSplit大小        10M     64M,1M      64M, 63M
```

在Map任务开始前，会先获取文件在HDFS上的路径和block信息，然后根据splitSize对文件进行切分（splitSize = computeSplitSize(blockSize, minSize, maxSize)），默认splitSize 就等于blockSize的默认值（64m）。

假设现在我们有两个文本文件，作为我们例子的输入：

```
File 1 内容:
My name is Tony
My company is pivotal

File 2 内容:
My name is Lisa
My company is EMC
```

3.2 Map阶段

Map是一类将输入记录集转换为中间格式记录集的独立任务，主要是读取InputSplit的每一个Key,Value对并进行处理。

首先我们的输入就是两个文件， 默认情况下就是两个split, 对应前面图中的split 0, split 1

两个split 默认会分给两个Mapper来处理， WordCount例子相当地暴力， 这一步里面就是直接把文件内容分解为单词和 1 （注意， 不是具体数量， 就是数字1）其中的单词就是我们的主健，也称为Key, 后面的数字就是对应的值，也称为value.

那么对应两个Mapper的输出就是：

> split 0

```
My       1
name     1
is         1
Tony     1
My          1
company     1
is       1
Pivotal   1
```

> split 1

```
My       1
name     1
is       1
Lisa     1
My       1
company  1
is       1
EMC       1
```

3.3 Shuffle阶段

将map的输出作为reduce的输入的过程就是shuffle了，这个是mapreduce优化的重点地方。这里我不讲怎么优化shuffle阶段，讲讲shuffle阶段的原理，因为大部分的书籍里都没讲清楚shuffle阶段。Shuffle一开始就是map阶段做输出操作，一般mapreduce计算的都是海量数据，map输出时

候不可能把所有文件都放到内存操作，因此map写入磁盘的过程十分的复杂，更何况map输出时候要对结果进行排序，内存开销是很大的，map在做输出时候会在内存里开启一个环形内存缓冲区，这个缓冲区专门用来输出的，默认大小是100mb，并且在配置文件里为这个缓冲区设定了一个阀值，默认是0.80（这个大小和阀值都是可以在配置文件里进行配置的），同时map还会为输出操作启动一个守护线程，如果缓冲区的内存达到了阀值的80%时候，这个守护线程就会把内容写到磁盘上，这个过程叫spill，另外的20%内存可以继续写入要写进磁盘的数据，写入磁盘和写入内存操作是互不干扰的，如果缓存区被撑满了，那么map就会阻塞写入内存的操作，让写入磁盘操作完成后再继续执行写入内存操作，前面我讲到写入磁盘前会有个排序操作，这个是在写入磁盘操作时候进行，不是在写入内存时候进行的，如果我们定义了combiner函数，那么排序前还会执行combiner操作。每次spill操作也就是写入磁盘操作时候就会写一个溢出文件，也就是说在做map输出有几次spill就会产生多少个溢出文件，等map输出全部做完后，map会合并这些输出文件。这个过程里还会有一个Partitioner操作，对于这个操作很多人都很迷糊，其实Partitioner操作和map阶段的输入分片（Input split）很像，一个Partitioner对应一个reduce作业，如果我们mapreduce操作只有一个reduce操作，那么Partitioner就只有一个，如果我们有多个reduce操作，那么Partitioner对应的就会有多个，Partitioner因此就是reduce的输入分片，这个程序员可以编程控制，主要是根据实际key和value的值，根据实际业务类型或者为了更好的reduce负载均衡要求进行，这是提高reduce效率的一个关键所在。到了reduce阶段就是合并map输出文件了，Partitioner会找到对应的map输出文件，然后进行复制操作，复制操作时reduce会开启几个复制线程，这些线程默认个数是5个，程序员也可以在配置文件更改复制线程的个数，这个复制过程和map写入磁盘过程类似，也有阀值和内存大小，阀值一样可以在配置文件里配置，而内存大小是直接使用reduce的tasktracker的内存大小，复制时候reduce还会进行排序操作和合并文件操作，这些操作完了就会进行reduce计算了。

3.3.1 Partition

Partition 是什么？ Partition 就是分区。

为什么要分区？ 因为有时候会有多个Reducer, Partition就是提前对输入进行处理， 根据将来的Reducer进行分区. 到时候Reducer处理的时候， 只需要处理分给自己的数据就可以了。

如何分区？ 主要的分区方法就是按照Key 的不同，把数据分开，其中很重要的一点就是要保证Key的唯一性， 因为将来做Reduce的时候有可能是在不同的节点上做的， 如果一个Key同时存在于两个节点上， Reduce的结果就会出问题， 所以很常见的Partition方法就是哈希。

结合我们的例子， 我们这里假设有两个Reducer, 前面两个split 做完Partition的结果就会如下：

> split 0

```
Partition 1:
company  1
is       1
```

```
is       1

Partition 2:
My       1
My       1
name     1
Pivotal    1
Tony       1
```

> split 1

```
Partition 1:
company 1
is       1
is       1
EMC      1

Partition 2:
My       1
My       1
name     1
Lisa     1
```

其中Partition 1 将来是准备给Reducer 1 处理的， Partition 2 是给Reducer 2 的

这里我们可以看到， Partition 只是把所有的条目按照Key 分了一下区， 没有其他任何处理， 每个区里面的Key 都不会出现在另外一个区里面。

3.3.2 Sort

Sort 就是排序喽， 其实这个过程在我来看并不是必须的， 完全可以交给客户自己的程序来处理。 那为什么还要排序呢？ 可能是写MapReduce的大牛们想，“大部分reduce 程序应该都希望输入的是已经按Key排序好的数据， 如果是这样， 那我们就干脆顺手帮你做掉啦， 请叫我雷锋！” .....好吧， 你是雷锋.

那么我们假设对前面的数据再进行排序， 结果如下：

> split 0

```
Partition 1:
company  1
is       1
is       1
```

```
Partition 2:
My       1
My        1
name    1
Pivotal    1
Tony      1
```

> split 1

```
Partition 1:
company 1
EMC     1
is       1
is      1

Partition 2:
Lisa     1
My       1
My        1
name     1
```

这里可以看到， 每个partition里面的条目都按照Key的顺序做了排序

3.3.3 Combine

什么是Combine呢？ Combine 其实可以理解为一个mini Reduce 过程， 它发生在前面Map的输出结果之后， 目的就是在结果送到Reducer之前先对其进行一次计算， 以减少文件的大小， 方便后面的传输。 但这步也不是必须的。

按照前面的输出， 执行Combine:

> split 0

```
Partition 1:
company  1
is       2

Partition 2:
My       2
name    1
Pivotal    1
Tony      1
```

> split 1

```
Partition 1:
company 1
EMC     1
is       2

Partition 2:
Lisa     1
My       2
name     1
```

我们可以看到， 针对前面的输出结果， 我们已经局部地统计了is 和My的出现频率， 减少了输出文件的大小。

3.3.4 copy

下面就要准备把输出结果传送给Reducer了。 这个阶段被称为Copy, 但事实上雷子认为叫他Download更为合适， 因为实现的时候， 是通过http的方式， 由Reducer节点向各个mapper节点下载属于自己分区的数据。

那么根据前面的Partition, 下载完的结果如下：

Reducer 节点 1 共包含两个文件:

```
Partition 1:
company  1
is       2
```

```
Partition 1:
company    1
EMC        1
is        2
```

Reducer 节点 2 也是两个文件:

```
 Partition 2:
My       2
name    1
Pivotal   1
Tony      1
```

```
Partition 2:
Lisa     1
My       2
name     1
```

这里可以看到， 通过Copy, 相同Partition 的数据落到了同一个节点上。

3.3.5 merge

如上一步所示， 此时Reducer得到的文件是从不同Mapper那里下载到的， 需要对他们进行合并为一个文件， 所以下面这一步就是Merge, 结果如下：

> Reducer 节点 1

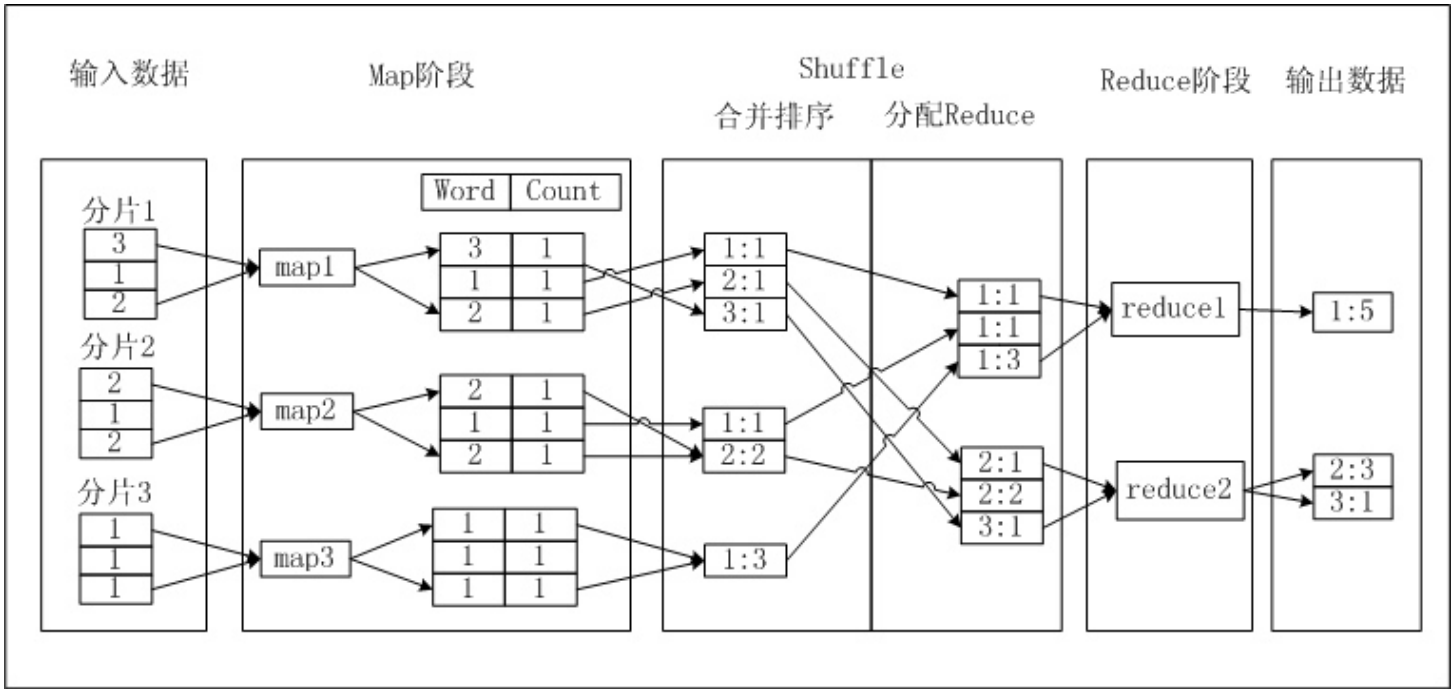
```
company  1
company  1
EMC      1
is       2
is        2
```

> Reducer 节点 2

```
Lisa    1
My       2
My        2
name    1
name    1
Pivotal   1
Tony      1
```

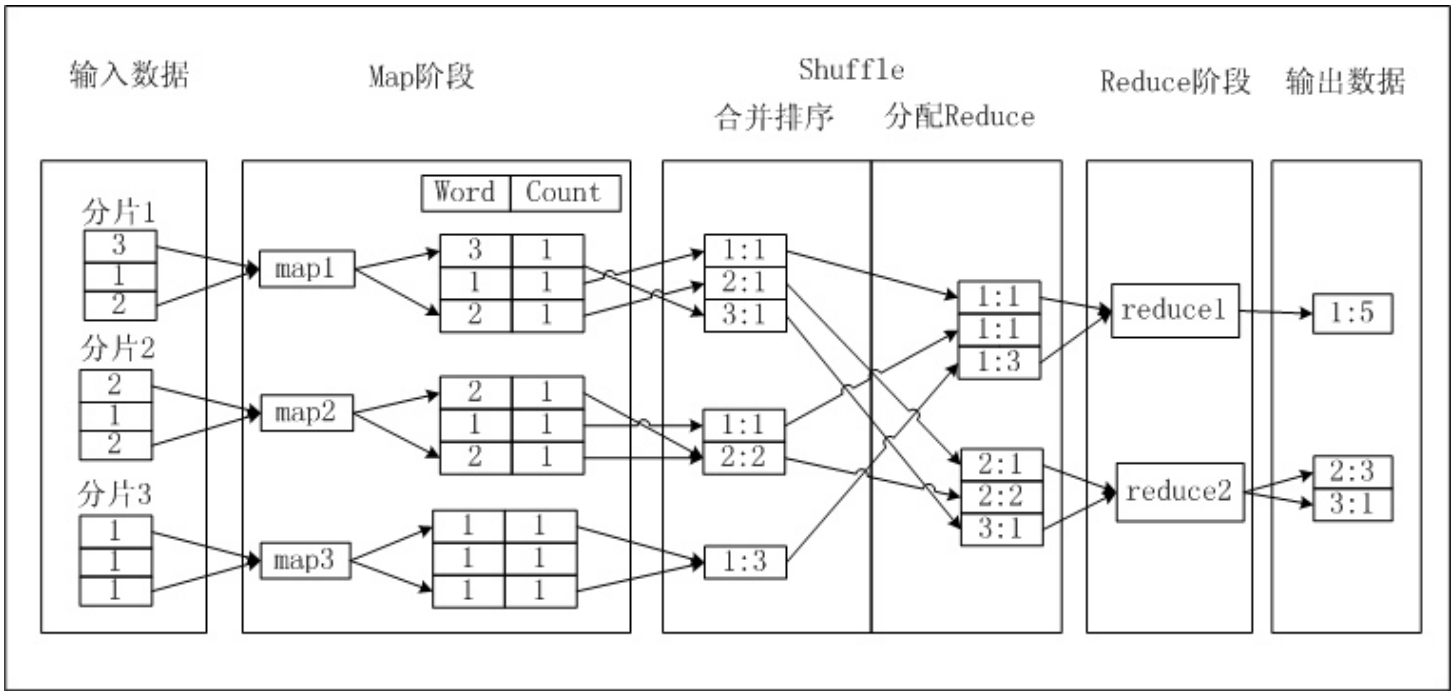
3.4 Redeuce阶段

reduce阶段对数据进行归约处理。键相等的键值对会调用一次reduce方法。经过这一阶段，数据量会减少。归约后的数据输出到本地文件中。本阶段默认是没有的，需要用户自己增加这一阶段的代码。



四、实例说明

下面将以WordCount为例，解释MapReduce各个阶段的概念。 假设存在一个文本a.txt，文本内每行是一个数字，我们要统计每个数字出现的次数。文本内的数字称为Word，数字出现的次数称为Count。如果MaxCompute Mapreduce完成这一功能，需要经历下图描述的几个步骤：



1. 首先对文本进行分片，将每片内的数据作为单个Map Worker的输入；
2. Map处理输入，每获取一个数字，将数字的Count设置为1，并将此对输出，此时以Word作为输出数据的Key； 在Shuffle阶段前期，首先对每个Map Worker的输出，按照Key值，即Word值排序。排序后进行Combine操作，即将Key值(Word值)相同的Count累加，构成一个新的对。此过程被称为合并排序；

3. 在Shuffle阶段后期，数据被发送到Reduce端。Reduce Worker收到数据后依赖Key值再次对数据排序；
4. 每个Reduce Worker对数据进行处理时，采用与Combiner相同的逻辑，将Key值(Word值)相同的Count累加，得到输出结果；

分布式相关
mr 方案解决矩阵相乘的代码；
hadoop原理，shuffle如何排序，map如何切割数据，如何处理数据倾斜，join的mr代码

MR的shuffle过程？内存不够时涉及大文件排序如何处理？ 答：先hash到不同文件中，每个文件排序，然后每个文件读取行，类似归并排序的思路？

Hadoop,Spark,storm下面的产品，原理，适用场景

spark跟hadoop的区别答：spark有RDD机制，写内存，相对hadoop适合迭代运算

如何用hadoop实现k-means
简单介绍 MapReduce 原理，有没有看过源码，说说 Map 阶段怎么实现的，
MapReduce 实现统计出现次数最多的前 100 个访问 IP.
MapReduce 实现统计不重复用户 ID,MapReduce 实现两个数据集求交集。
HBase 行健怎么设计,spark 性能一般优化方法,spark streaming 和 storm 区别.给了一张笔试题， 10 道选择，一道大题。选择题是 java 基础知识，大题一个有三问：根据场景写出 Hive 建表语句； Hsql 从表中查询；
用MapReduce写好友推荐，在一堆单词里面找出现次数最多的k个
用分布式的方法做采样怎么保证采样结果完全符合预期?
后面又问了Hadoop,Spark,storm下面的产品，原理，适用场景，
写一个 Hadoop 版本的 wordcount。
K-means能否分布式实现？ 答：因为本身是迭代式算法，所以只能半分布式实现，即在计算类的均值、每个样本点属于哪个类的时候
还有怎么解决mapreduce数据倾斜