大数据管理
Big Data Management

张海腾

htzhang@ecust.edu.cn

第7章 MapReduce

- 7.1 概述
- 7.2 MapReduce工作流程
- 7.3 实例分析：WordCount
- 7.4 MapReduce的具体应用
- 7.5 MapReduce编程实践
- 7.6 本章小结

7.1 概述

- 7.1.1 分布式并行编程
- 7.1.2 MapReduce模型简介
- 7.1.3 Map和Reduce函数

7.1.1 分布式并行编程

- “摩尔定律”， CPU性能大约每隔18个月翻一番
- 从2005年开始摩尔定律逐渐失效，需要处理的数据量快速增加，人们开始借助于分布式并行编程来提高程序性能
- 分布式程序运行在大规模计算机集群上，可以并行执行大规模数据处理任务，从而获得海量的计算能力
- 谷歌公司最先提出了分布式并行编程模型MapReduce，Hadoop MapReduce是它的开源实现，后者比前者使用门槛低很多

7.1.1 分布式并行编程

- **问题：**在MapReduce出现之前，已经有像MPI这样非常成熟的并行计算框架了，那么为什么Google还需要MapReduce？MapReduce相较于传统的并行计算框架有什么优势？

	传统并行计算框架	MapReduce
集群架构/容错性	共享式(共享内存/共享存储)，容错性差	非共享式，容错性好
硬件/价格/扩展性	刀片服务器、高速网、SAN，价格贵，扩展性差	普通PC机，便宜，扩展性好
编程/学习难度	what-how，难	what，简单
适用场景	实时、细粒度计算、计算密集型	批处理、非实时、数据密集型

7.1.2 MapReduce模型简介

- MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce
- 编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量数据的计算
- MapReduce采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理

7.1.2 MapReduce模型简介

- MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销
- MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave。Master上运行JobTracker，Slave上运行TaskTracker
- Hadoop框架是用Java实现的，但是，MapReduce应用程序则不一定要用Java来写

7.1.3 Map和Reduce函数

- 在MapReduce中，一个存储在分布式文件系统中的大规模数据集会被切分成许多独立的小数据块，这些小数据块可以被多个Map任务并行处理。
- MapReduce框架会为每个Map任务输入一个数据子集，Map任务生成的结果会继续作为Reduce任务的输入，最终由Reduce任务输出最后结果，并写入分布式文件系统。
- 特别需要注意的是，适合用MapReduce来处理的数据集需要满足一个前提条件：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。

7.1.3 Map和Reduce函数

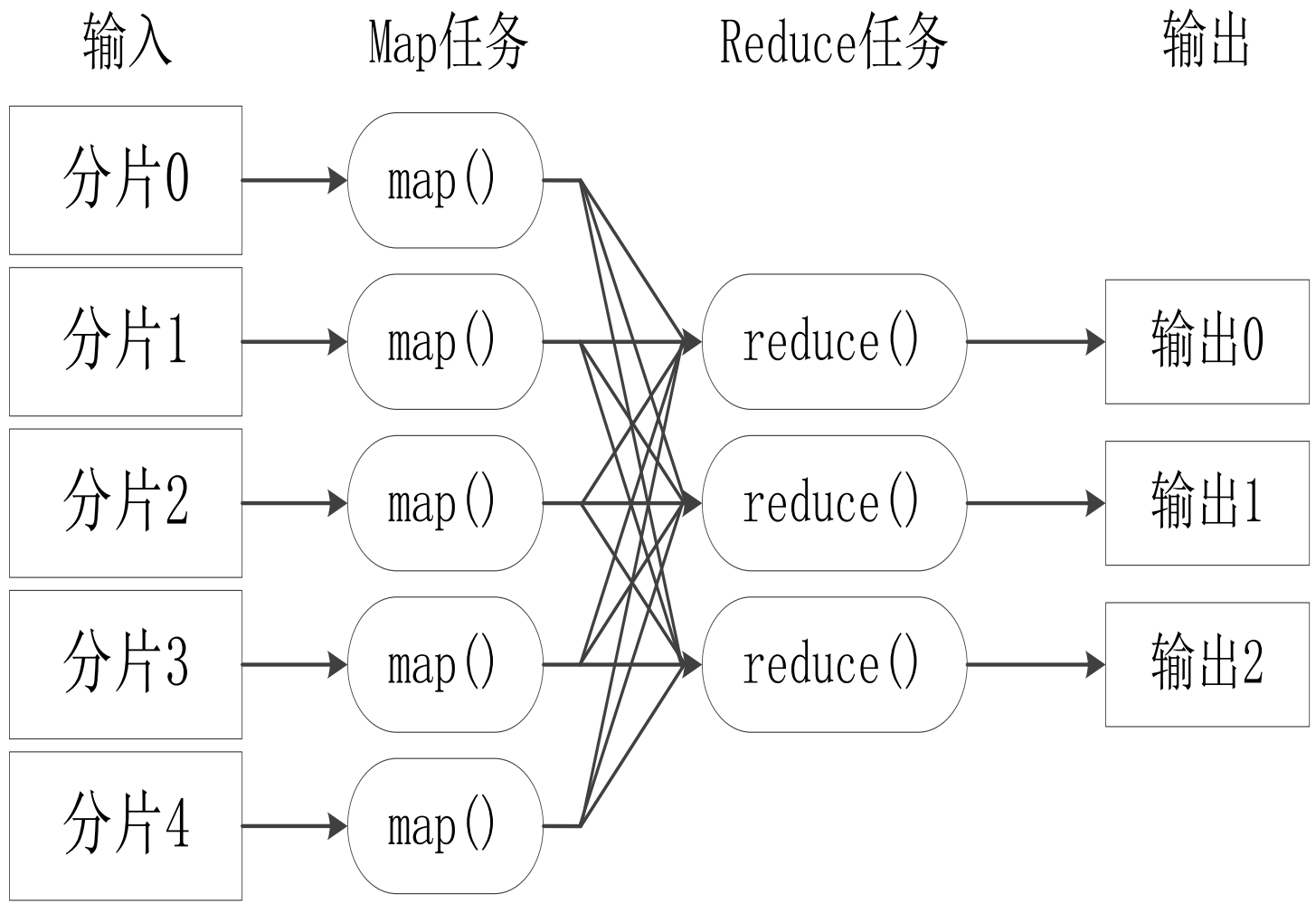
函数	输入	输出	说明
Map	<k_1，v_1> 如： <行号，“a b c”>	List(<k_2，v_2>) 如： <“a”，1> <“b”，1> <“c”，1>	1.将小数据集进一步解析成一批<key，value>对，输入Map函数中进行处理 2.每一个输入的<k_1，v_1>会输出一批<k_2，v_2>。<k_2，v_2>是计算的中间结果
Reduce	<k_2，List(v_2)> 如：<“a”，<1，1，1>>	<k_3，v_3> <“a”，3>	输入的中间结果<k_2，List(v_2)>中的List(v_2)表示是一批属于同一个k_2的value

7.2 MapReduce的体系结构

- 7.2.1 工作流程概述
- 7.2.2 MapReduce各个执行阶段
- 7.2.3 Shuffle过程详解

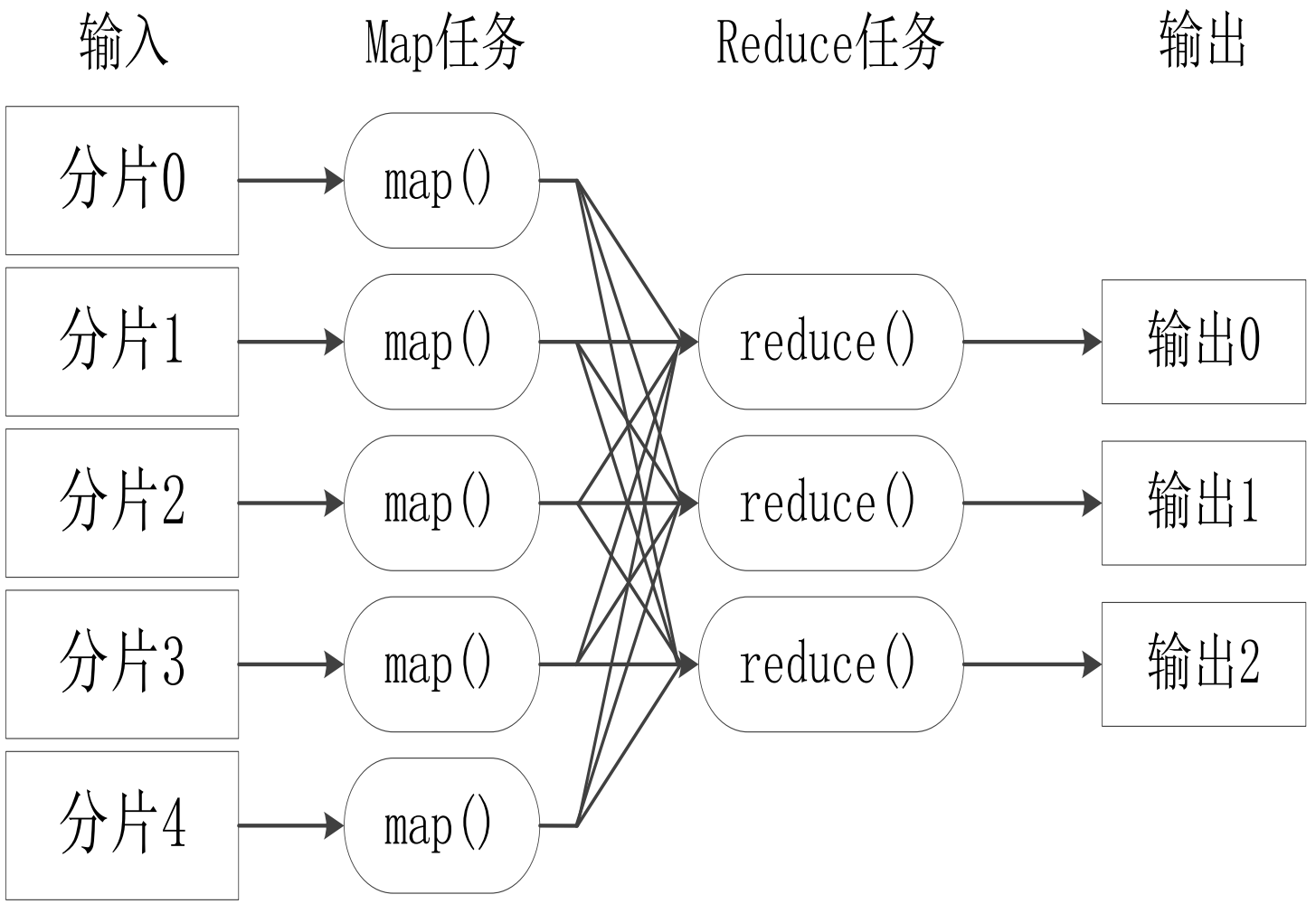
7.2.1 工作流程概述

- 一个大的MapReduce作业，首先会被拆分成许多个Map任务在多台机器上并行执行。
- 每个Map任务通常运行在数据存储的节点上，这样，计算和数据就可以放在一起运行，不需要额外的数据传输开销。



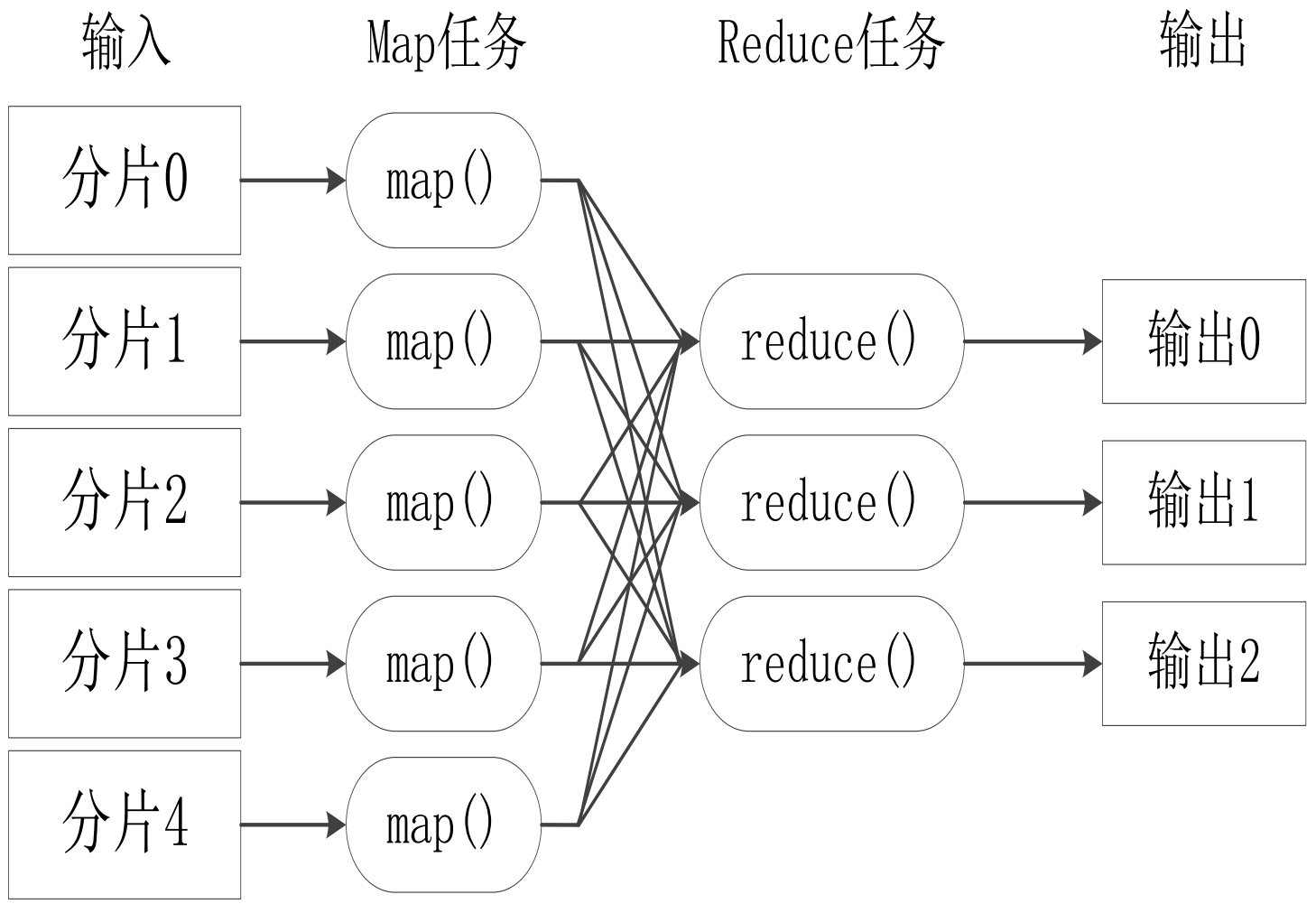
7.2.1 工作流程概述

- 当Map任务结束后，会生成以<key value>形式表示的许多中间结果。然后，这些中间结果会被分发到多个Reduce任务在多台机器上并行执行。
- 具有相同key的会被发送到同一个Reduce任务那里，Reduce任务会对中间结果进行汇总计算得到最后结果，并输出到分布式文件系统中。



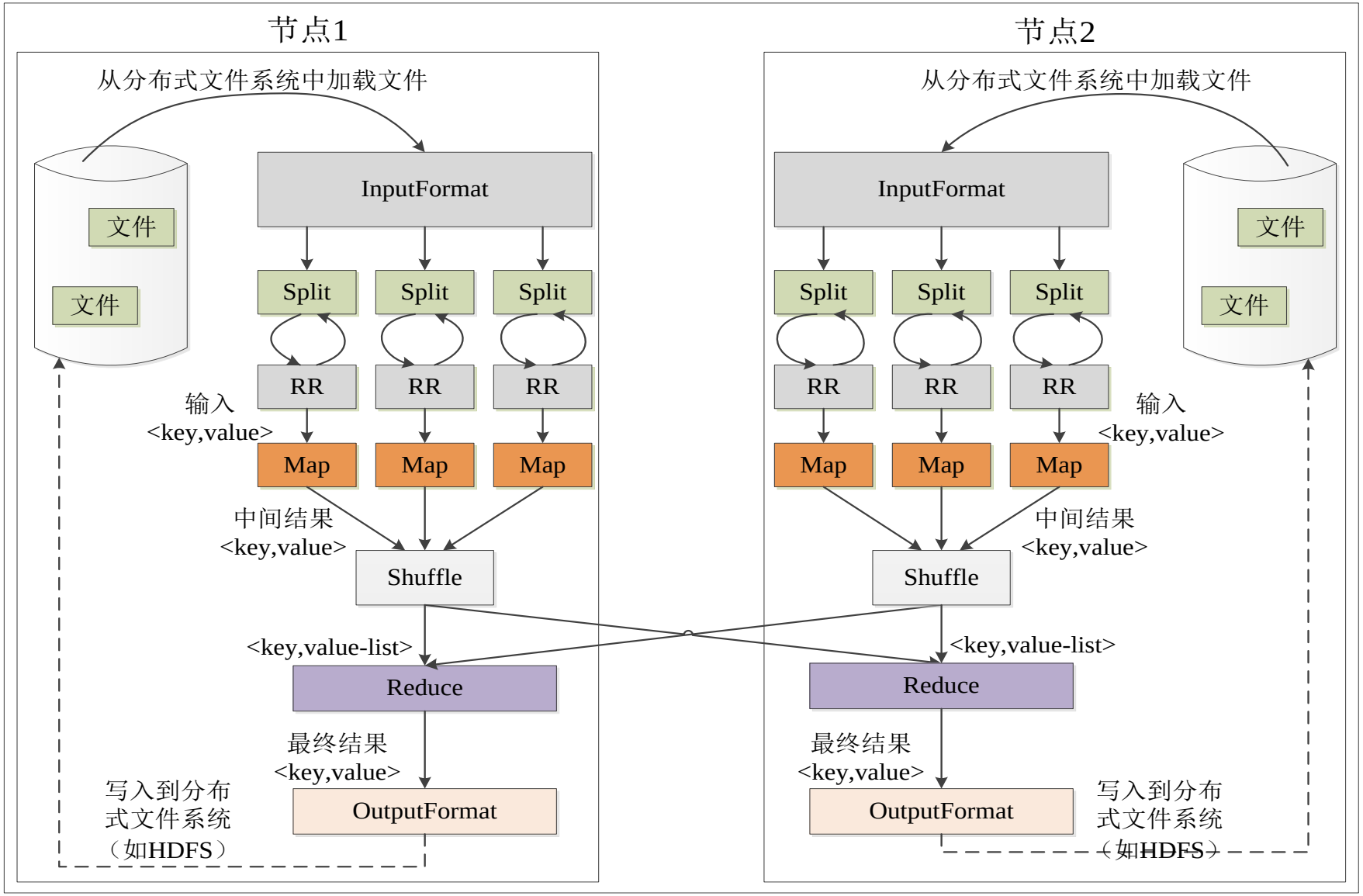
7.2.1 工作流程概述

- 不同的Map任务之间不会进行通信
- 不同的Reduce任务之间也不会发生任何信息交换
- 用户不能显式地从一台机器向另一台机器发送消息
- 所有的数据交换都是通过MapReduce框架自身去实现的



7.2.2 MapReduce各个执行阶段

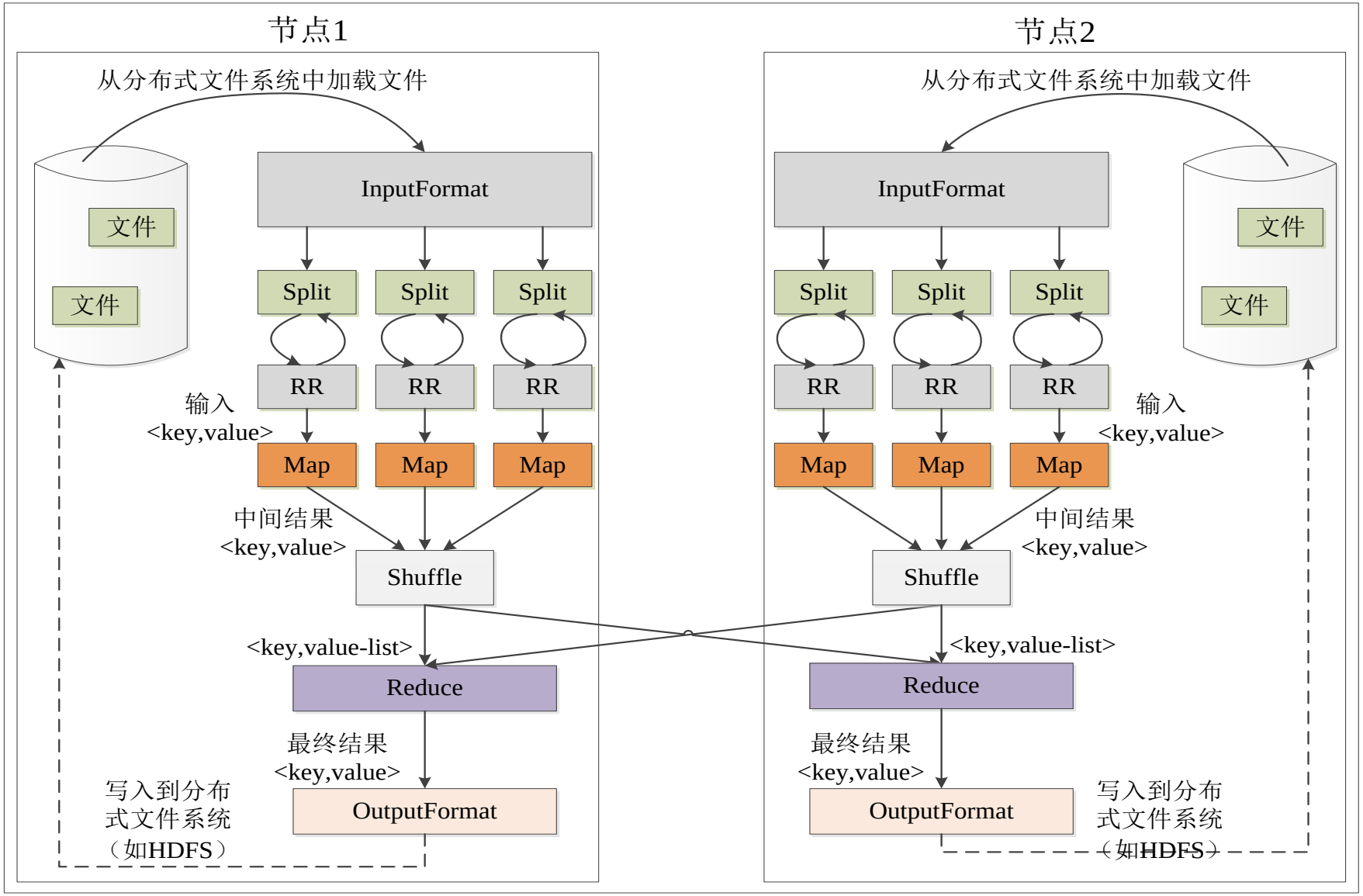
（1）MapReduce 框架使用 InputFormat 模块做 Map 前的预处理，比如验证输入的格式是否符合输入定义；然后，将输入文件切分为逻辑上的多个 InputSplit（逻辑概念，并没有实际切割）



7.2.2 MapReduce各个执行阶段

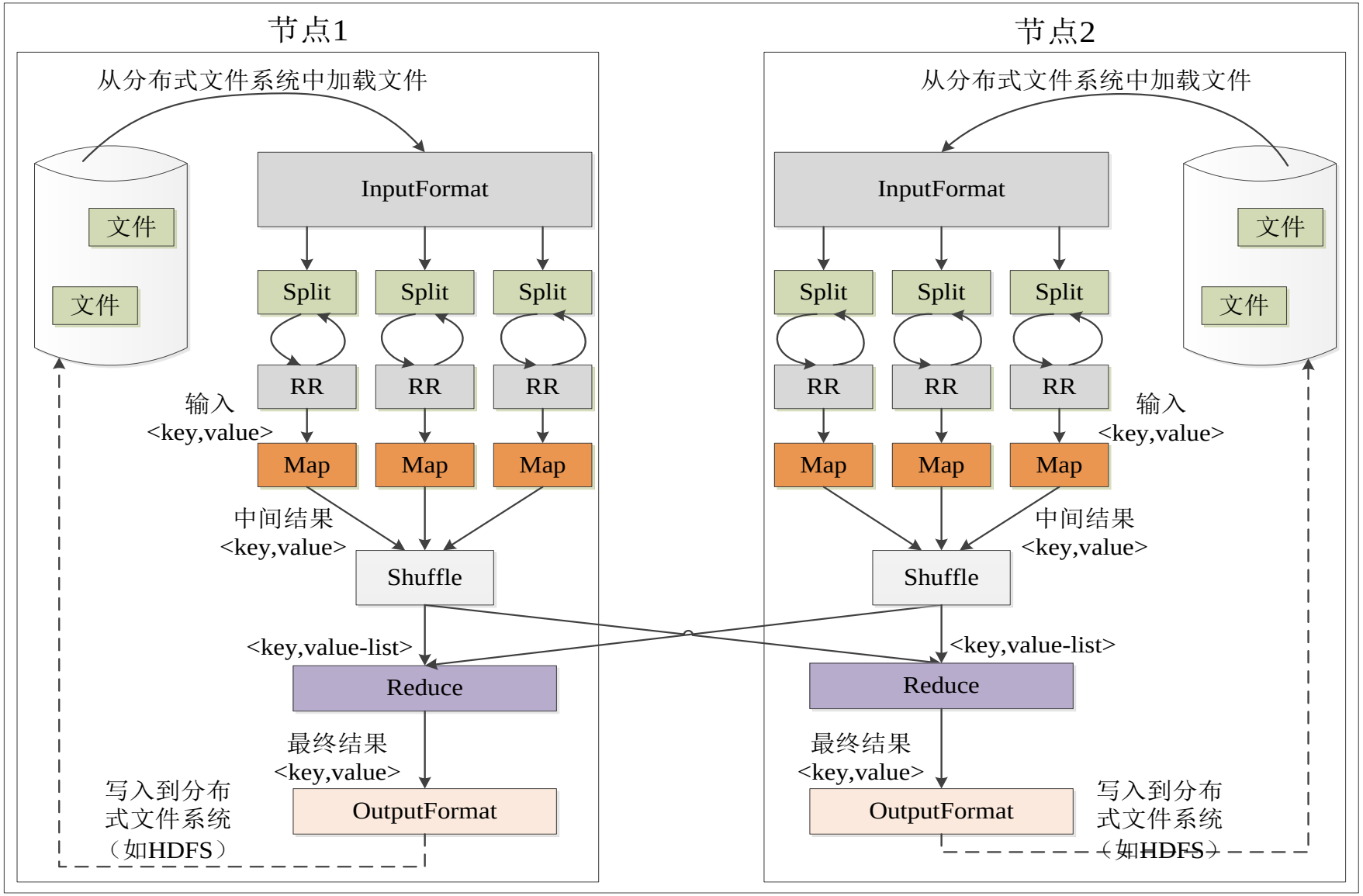
（2）RecorderReader (RR) 处理 InputSplit 中的具体记录，加载数据并转换为合适的键值对，输入给 Map 任务

（3）Map 任务根据用户自定义映射规则，输出一系列的 <key，value> 作为中间结果



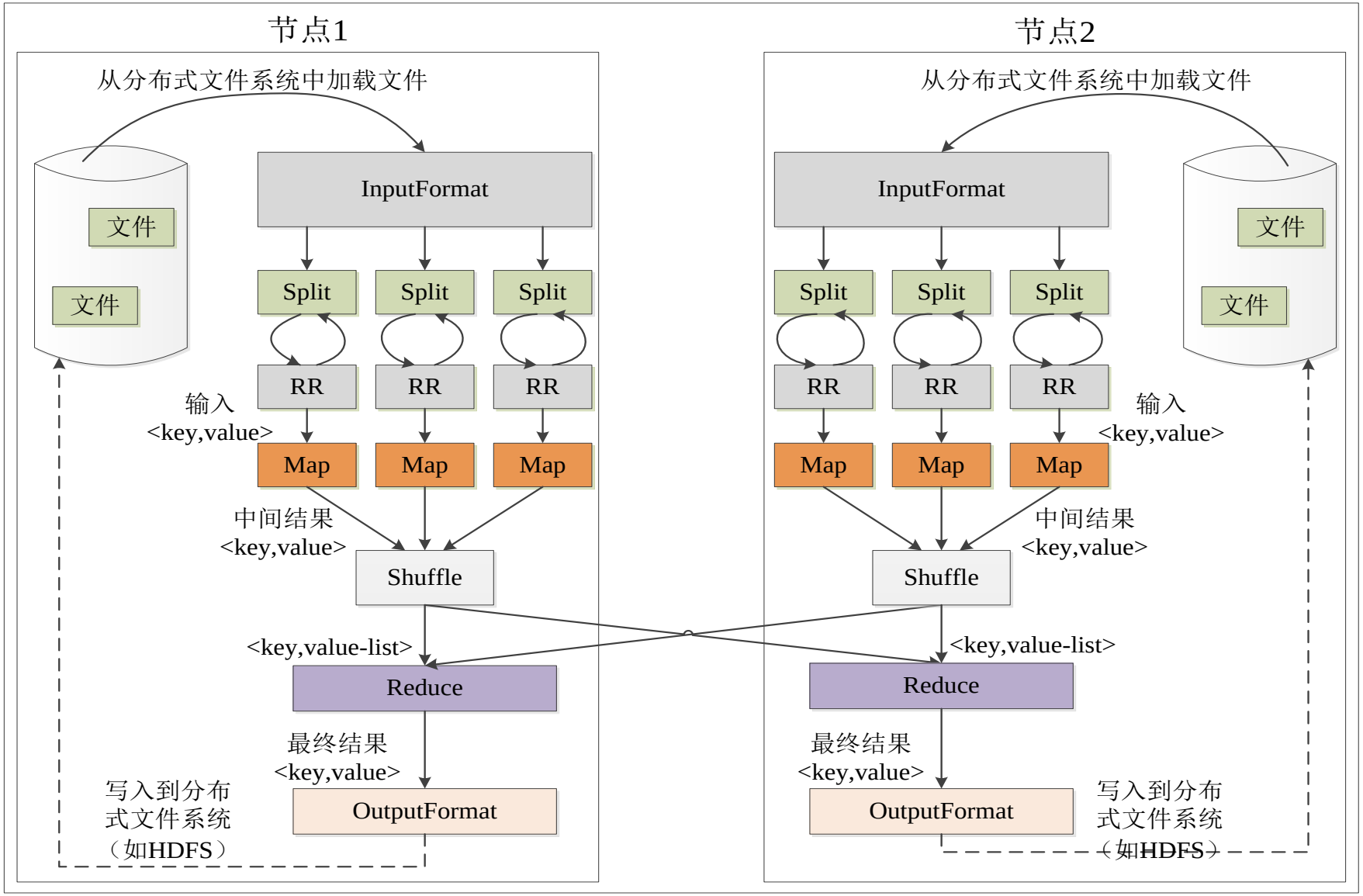
7.2.2 MapReduce各个执行阶段

（4）**Shuffle（洗牌）**，通过排序（Sort）、合并（Combine）、归并（Merge）等操作，**将无序的 <key，value> 转换为有序的 < key，value-list>**



7.2.2 MapReduce各个执行阶段

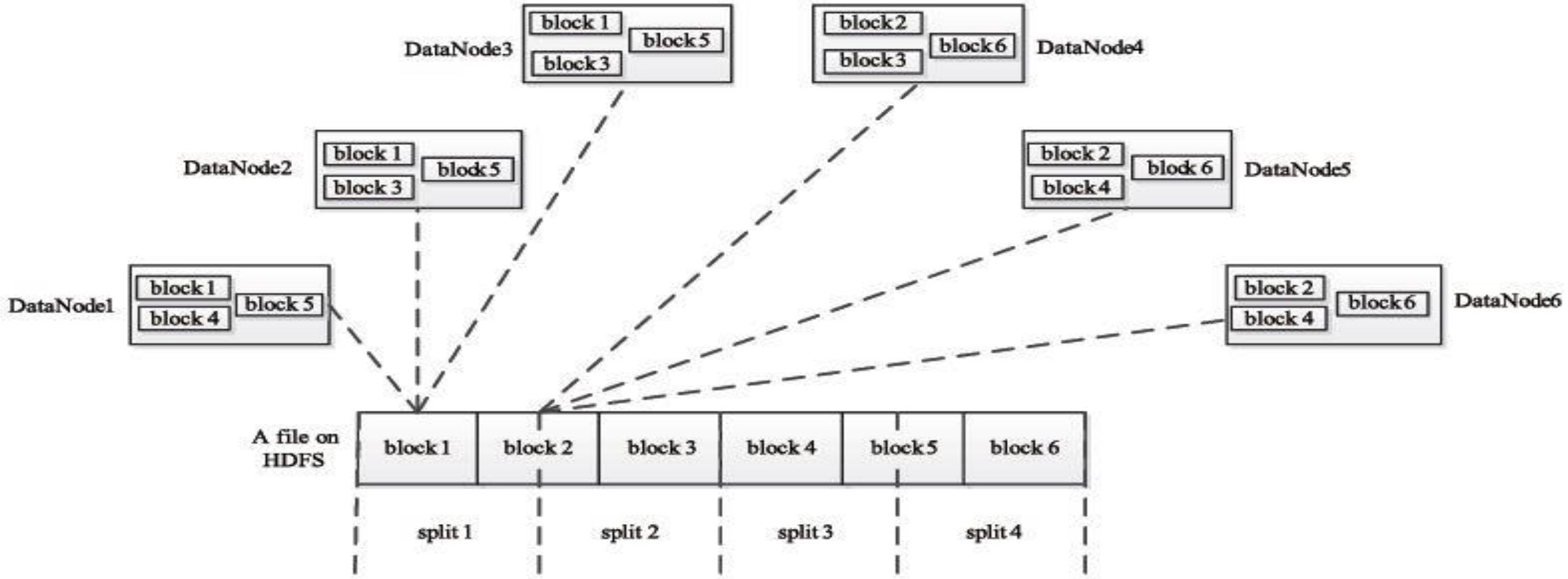
（5）Reduce 以一系列 <key，value-list> 中间结果作为输入，执行用户定义的逻辑，输出结果给 OutputFormat 模块



7.2.2 MapReduce各个执行阶段

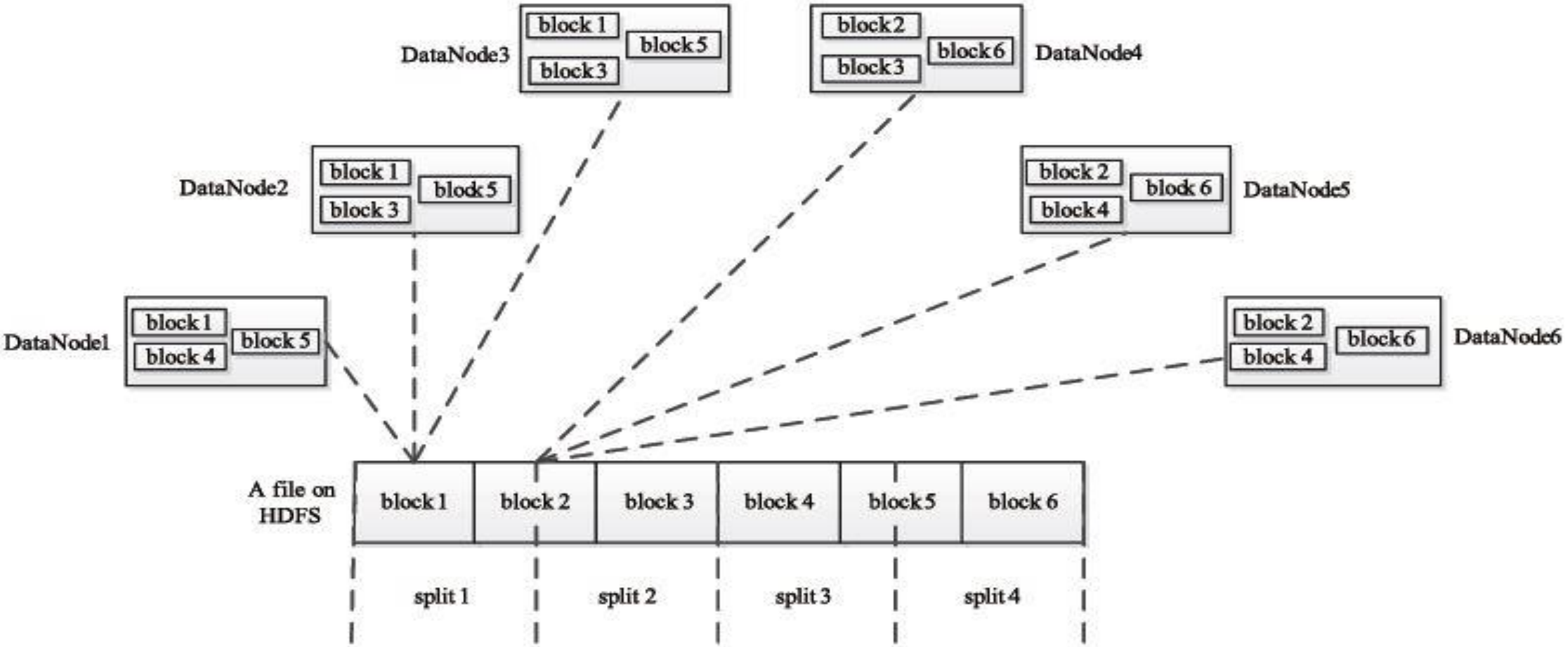
关于Split（分片）

- 模块HDFS 以固定大小的block 为基本单位存储数据，而对于MapReduce而言，其**处理单位是split**。



7.2.2 MapReduce各个执行阶段

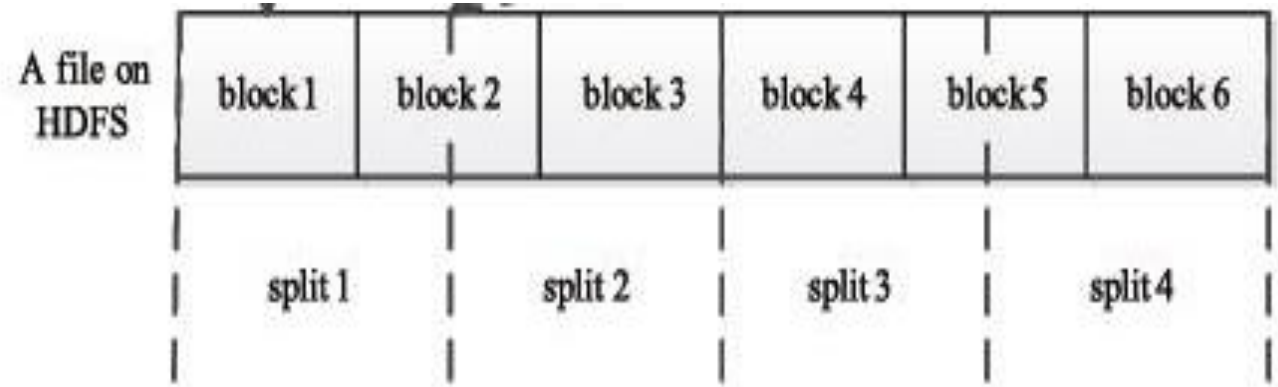
- split 是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在节点等。它的划分方法完全由用户自己决定



7.2.2 MapReduce各个执行阶段

Map任务的数量

- split创建一个Map任务，split 的多少决定了Map任务的数目。大多数情况下，理想的分片大小是一个HDFS块

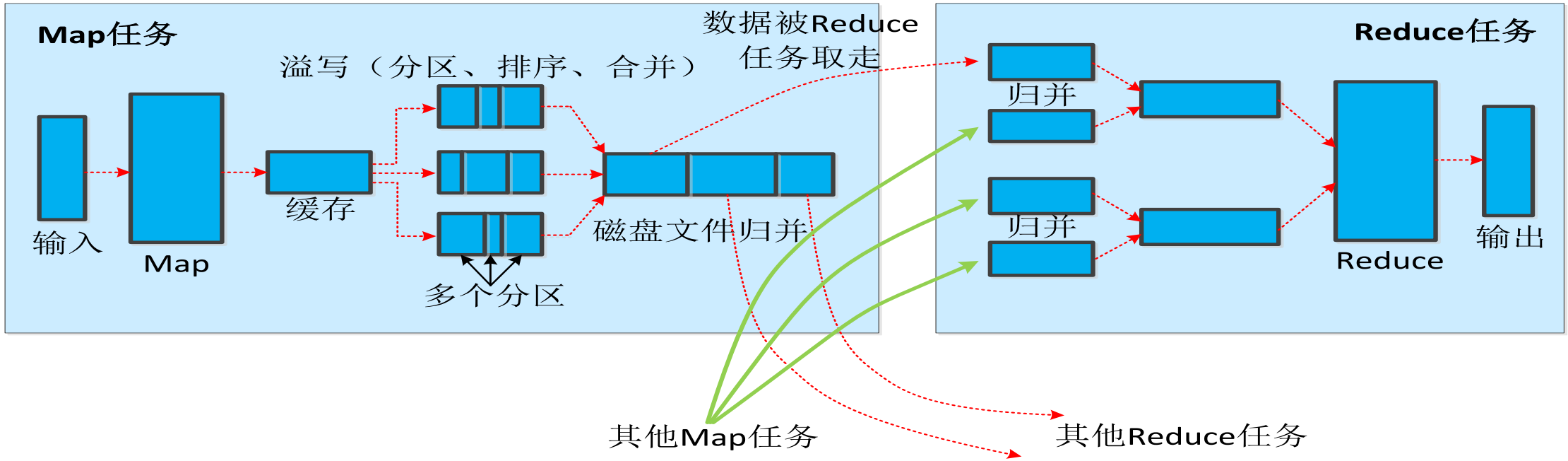


Reduce任务的数量

- 最优的Reduce任务个数取决于集群中可用的reduce任务槽(slot)的数目
- 通常设置比reduce任务槽数目稍微小一些的Reduce任务个数（这样可以预留一些系统资源处理可能发生的错误）

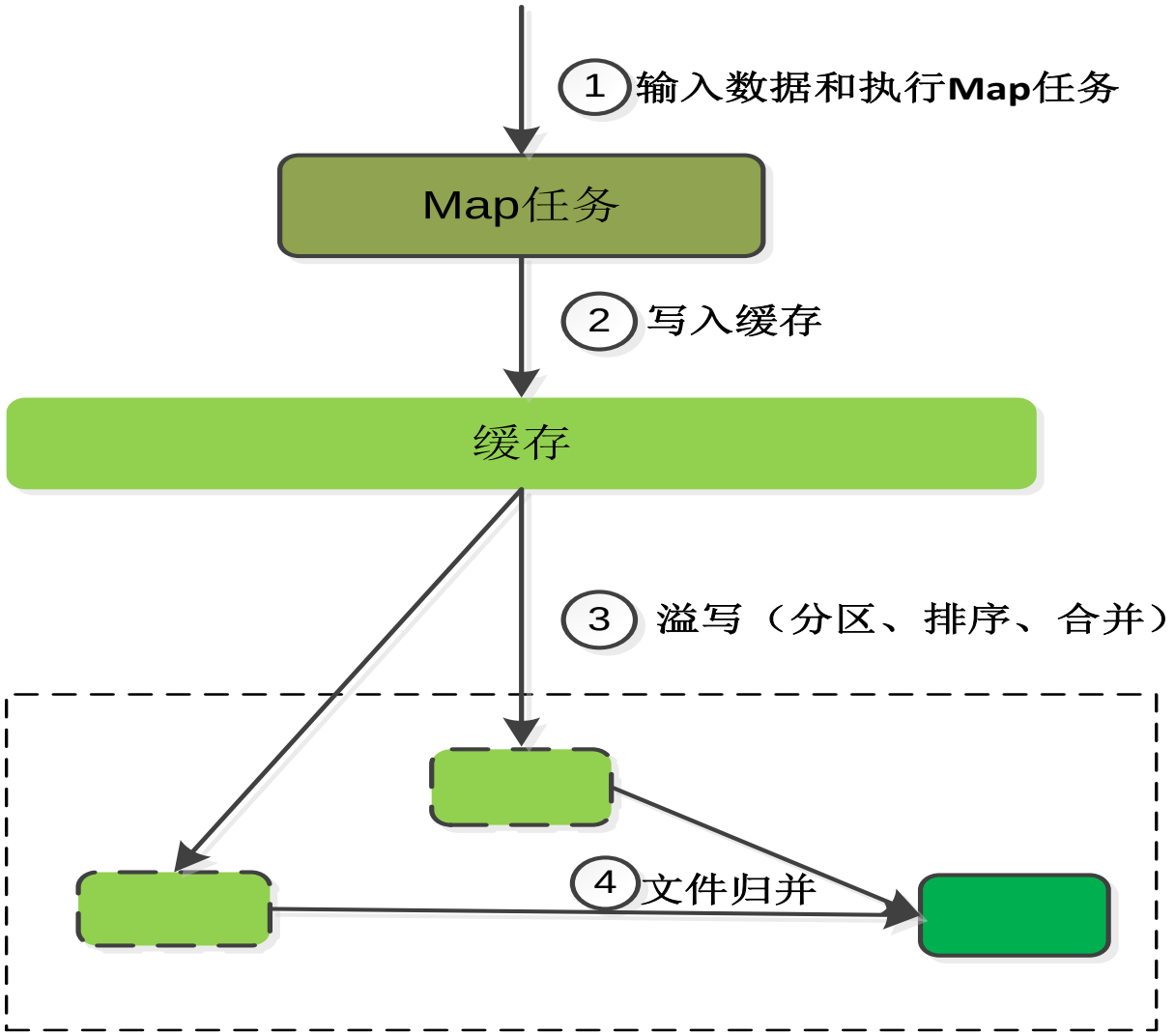
7.2.3 Shuffle过程详解

- 为了让Reduce可以并行处理Map的结果，需要对Map的输出进行一定的分区（Portition）、排序（Sort）、合并（Combine）、归并（Merge）等操作，得到形式的中间结果，再交给对应的 Reduce 进行处理，**这个过程称为 Shuffle。**
- Shuffle过程分为**Map端的操作和Reduce端的操作**



7.2.3 Shuffle过程详解

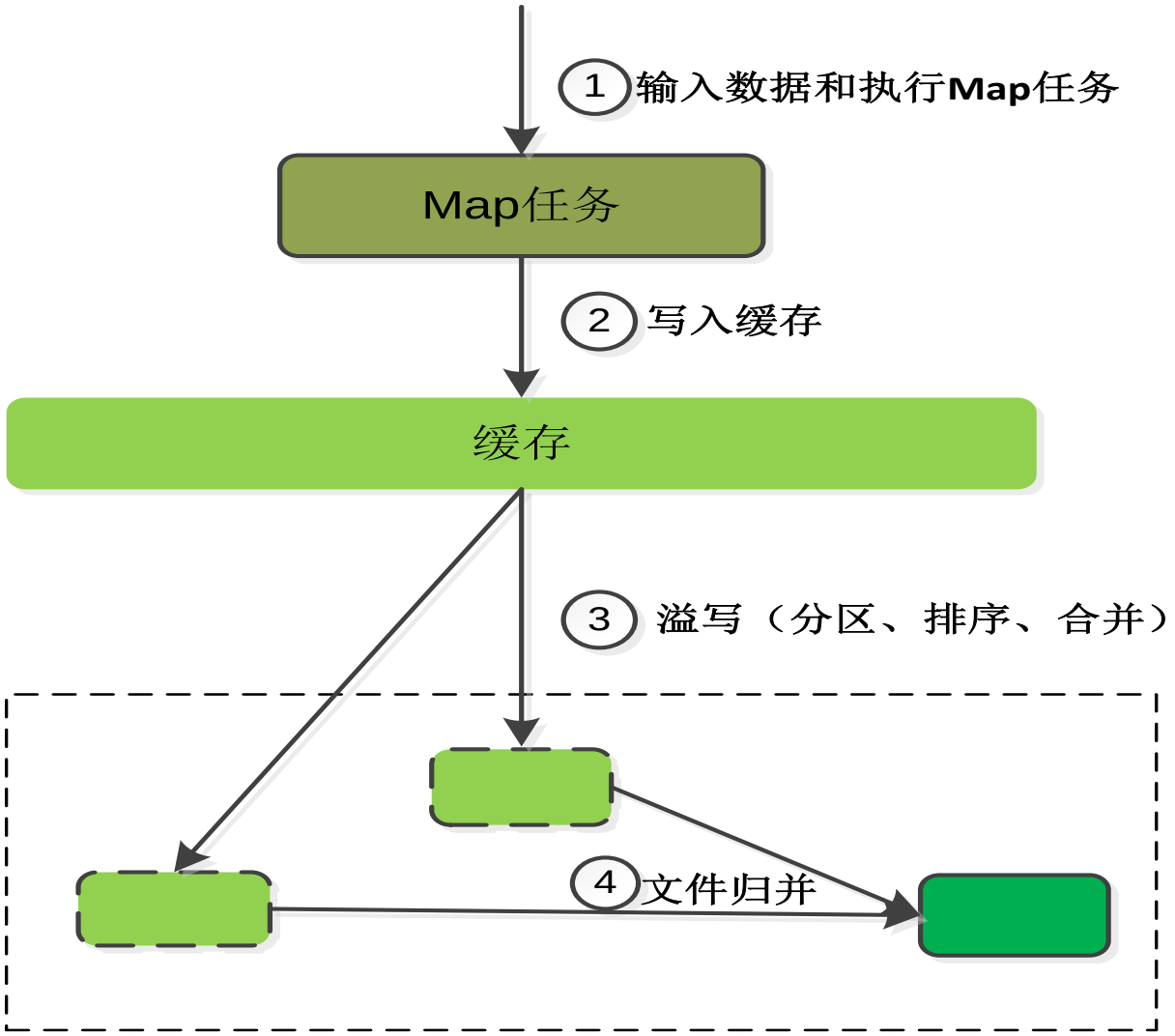
2. Map端的Shuffle过程



- Map的输出结果首先被写入缓存，当缓存满时，就启动溢写操作，把缓存中的数据写入磁盘文件，并清空缓存。
- 当启动溢写操作时，首先需要把缓存中的数据进行分区，然后对每个分区的数据进行排序（Sort）和合并（Combine），之后再写入磁盘文件。

7.2.3 Shuffle过程详解

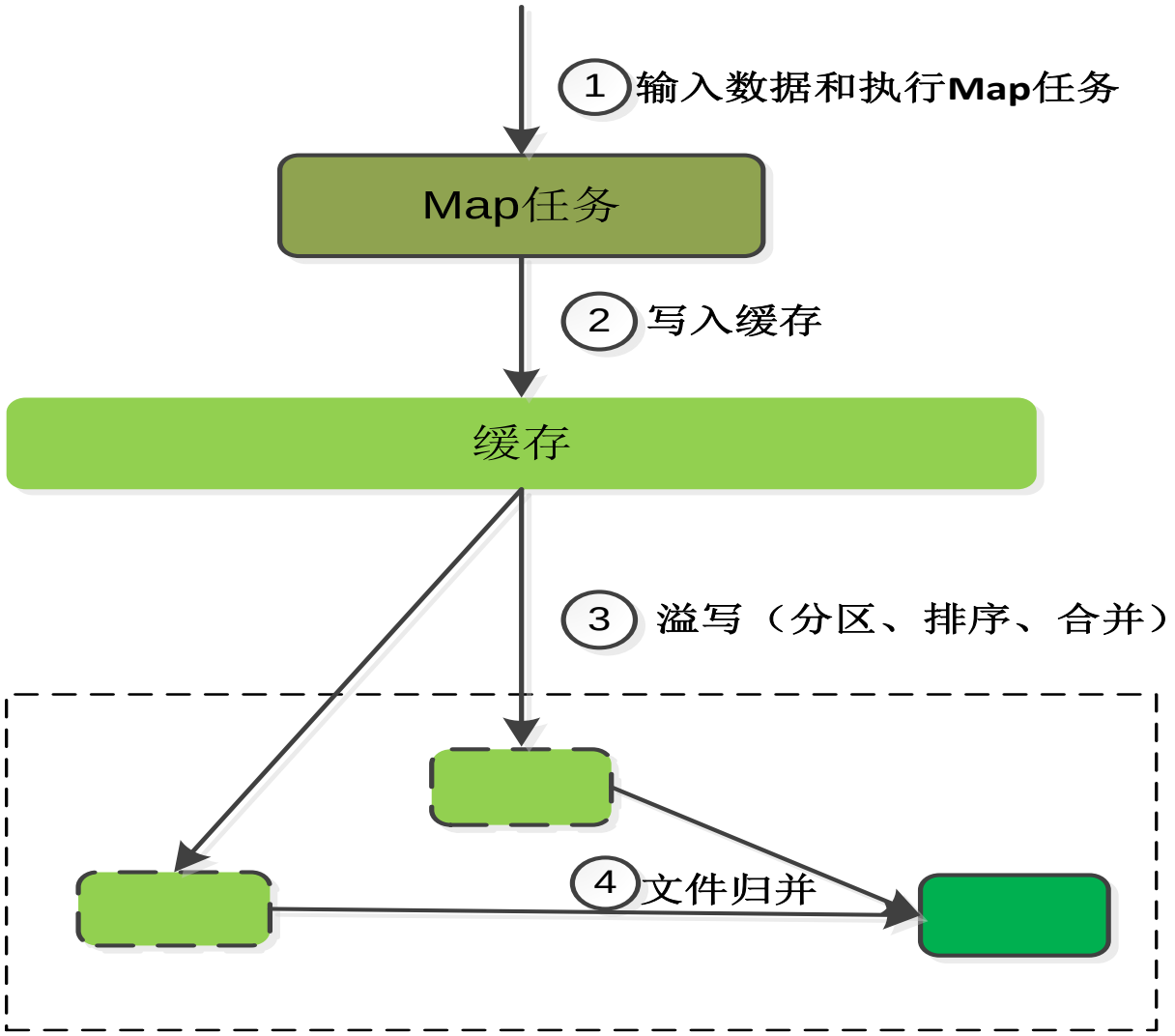
2. Map端的Shuffle过程



- 每次溢写操作会生成一个新的磁盘文件，随着Map任务的执行，磁盘中就会生成多个溢写文件。写入溢写文件中的所有键值对都是经过分区和排序的。

7.2.3 Shuffle过程详解

□ 2. Map端的Shuffle过程



- 在Map任务全部结束之前，这些溢写文件会被归并（Merge）成一个大的磁盘文件，这个大的溢写文件中的所有键值对也是经过分区和排序的。然后通知相应的Reduce任务来领取属于自己处理的数据。

7.2.3 Shuffle过程详解

□ 说明

1、并非所有场合都可以使用Combiner。

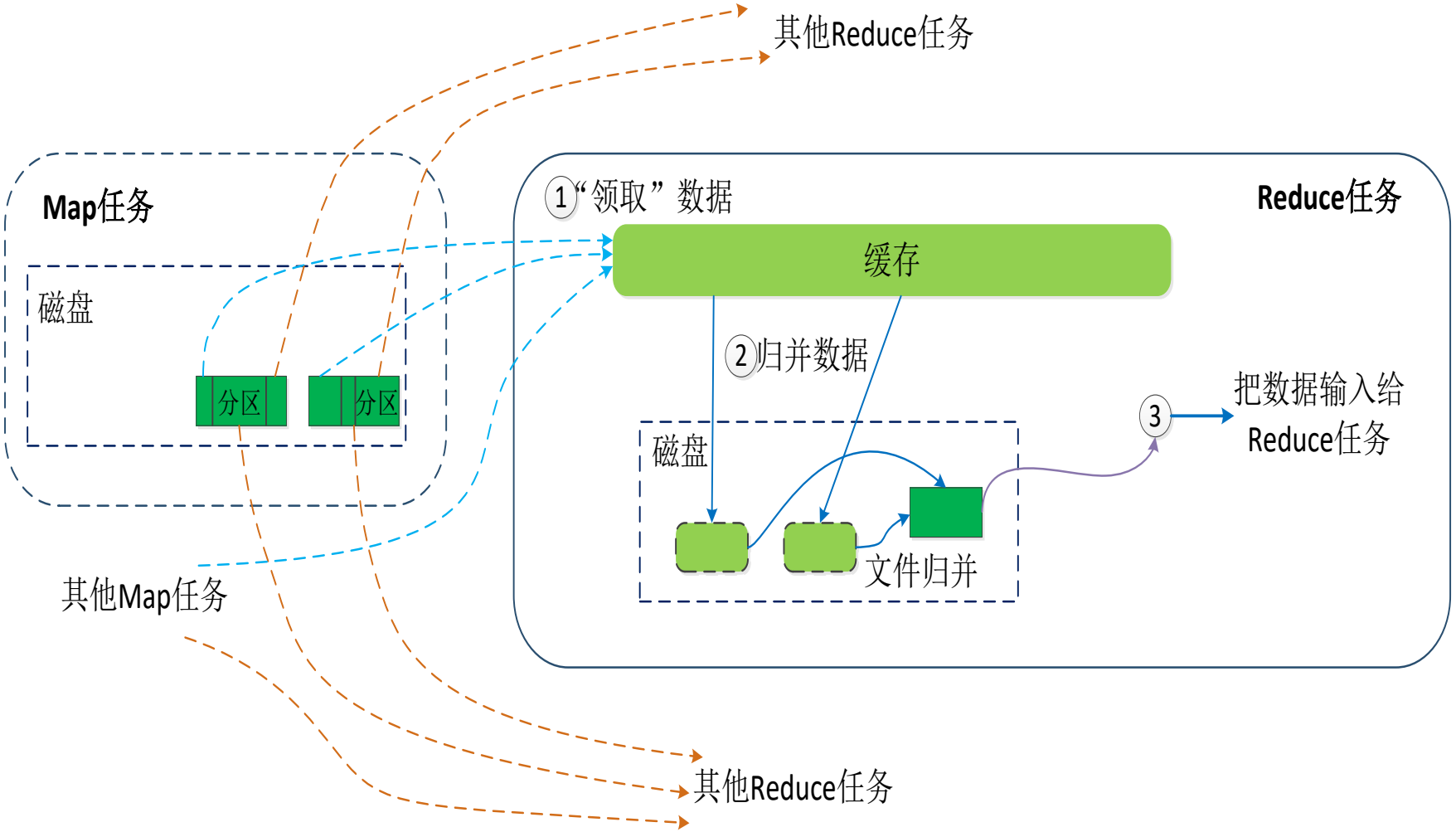
因为Combiner的输出是Reduce任务的输入，Combiner绝不能改变Reduce任务最终的计算结果，一般而言，累加、最大值等场景可以使用合并操作。

2、合并（Combine）和归并（Merge）的区别：

两个键值对<“a”，1>和<“a”，1>，如果合并，会得到<“a”，2>，如果归并，会得到<“a”，<1，1>>

7.2.3 Shuffle过程详解

□ Reduce端的Shuffle过程



包括3个步骤：

（1）“领取”数据。

（2）归并数据

（3）把数据输入给Reduce任务

7.2.3 Shuffle过程详解

（1）“领取”数据

- Map端的Shuffle过程结束后，所有Map输出结果都保存在Map机器的本地磁盘上，Reduce任务需要把这些数据**“领取”（Fetch）回来存放到自己所在机器的本地磁盘上。**
- 每个Reduce任务会不断地通过**RPC向JobTracker询问**Map任务是否已经完成；JobTracker监测到一个Map任务完成后，就会通知相关的Reduce任务来“领取”数据；一旦一个Reduce任务收到JobTracker的通知，它就会到该Map任务所在机器上把属于自己处理的分区数据领取到本地磁盘中。一般系统中会存在多个Map机器，因此Reduce任务会**使用多个线程同时从多个Map机器领回数据**。

7.2.3 Shuffle过程详解

（2）归并数据

- 从Map端领回的数据会首先被存放在Reduce任务所在机器的缓存中，如果缓存被占满，就会像Map端一样被溢写到磁盘中。当溢写过程启动时，具有相同key的键值对会被归并（Merge），如果用户定义了Combiner，则归并后的数据还可以执行合并操作，减少写入磁盘的数据量
- 最终，当所有的 Map 端数据都已经被领回时，和 Map端类似，多个溢写文件会被归并成一个大文件，归并的时候还会对键值对进行排序，从而使得最终大文件中的键值对都是有序的。假设磁盘中生成了50个溢写文件，每轮可以归并10个溢写文件，则需要经过5轮归并，得到5个归并后的大文件。

7.2.3 Shuffle过程详解

（3）把数据输入给Reduce任务

- 磁盘中经过多轮归并后得到的若干个大文件，不会继续归并成一个新的大文件，而是**直接输入给Reduce任务**，这样可以减少磁盘读写开销。由此，整个Shuffle过程顺利结束。
- 接下来，Reduce任务会执行 Reduce 函数中定义的各种映射，输出最终结果，并保存到分布式文件系统中（比如GFS或HDFS）。

7.3 实例分析：WordCount

- 7.3.1 WordCount程序任务
- 7.3.2 WordCount设计思路
- 7.3.3 WordCount的具体执行过程
- 7.3.4 一个WordCount执行过程的实例

7.3.1 WordCount程序任务

程序	WordCount
输入	一个包含大量单词的文本文件
输出	文件中每个单词及其出现次数（频数），并按照单词字母顺序排序，每个单词和其频数占一行，单词和频数之间有间隔

口一个WordCount的输入和输出实例

输入	输出
Hello World Hello Hadoop Hello MapReduce	Hadoop 1 Hello 3 MapReduce 1 World 1

7.3.2 WordCount设计思路

- 首先，需要检查WordCount程序任务是否可以采用MapReduce来实现。
  - 适合用MapReduce来处理的数据集需要满足一个前提条件：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。
  - 在WordCount程序任务中，不同单词之间的频数不存在相关性，彼此独立，可以把不同的单词分发给不同的机器进行并行处理，因此可以采用MapReduce来实现词频统计任务。

7.3.2 WordCount设计思路

- 其次，确定MapReduce程序的设计思路。
  - 把文件内容解析成许多个单词，然后把所有相同的单词聚集到一起，最后计算出每个单词出现的次数进行输出。
- 最后，确定MapReduce程序的执行过程。
  - 把一个大文件切分成许多个分片，每个分片输入给不同机器上的 Map任务，并行执行完成“从文件中解析出所有单词”任务。
  - Map的输入采用Hadoop默认的<k，v>输入，即文件行号作为key，该行号对应的文件的一行内容作为value；

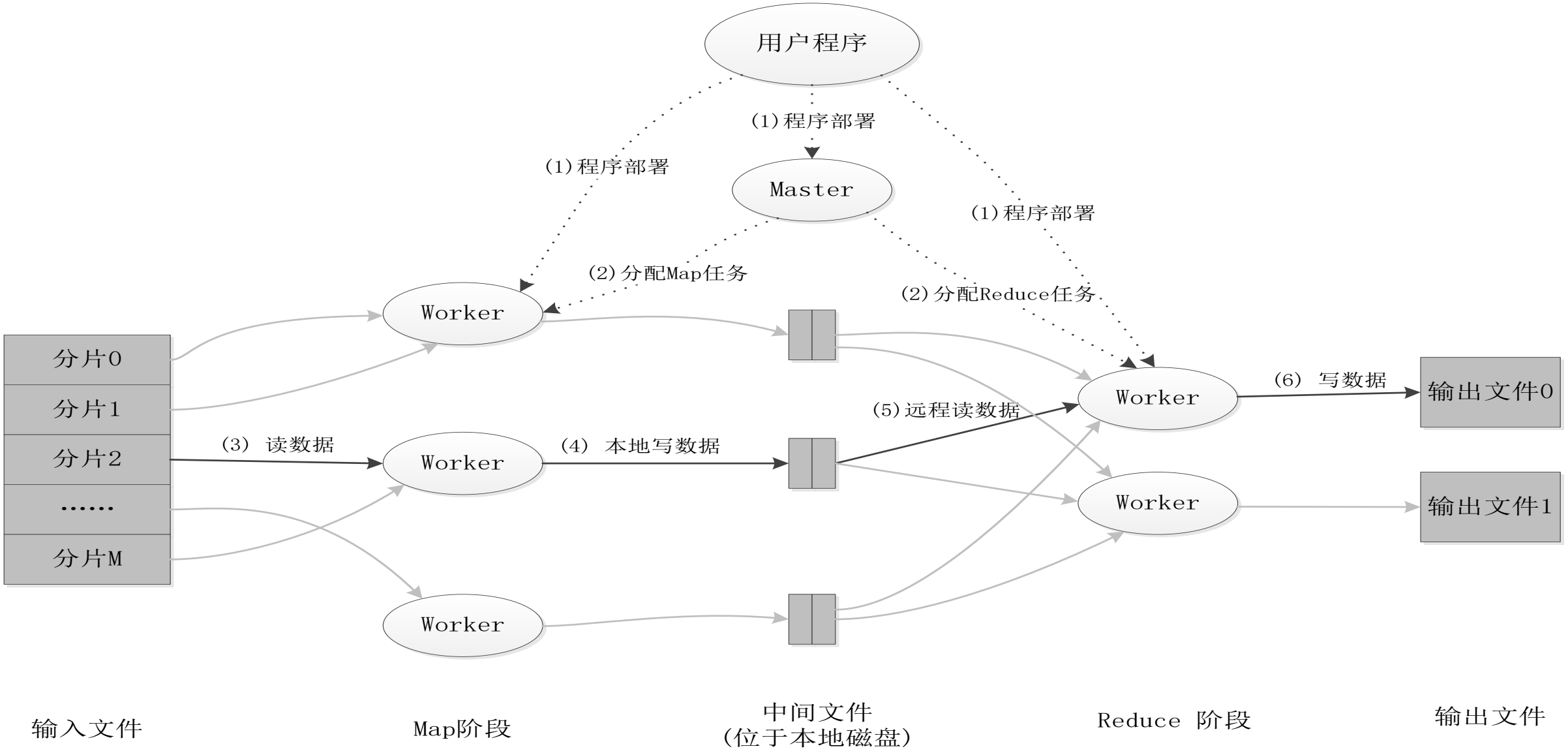
7.3.2 WordCount设计思路

- Map的输出以单词作为key，1作为value，即 <单词，1> 表示该单词出现了 1 次。
- Map 阶段结束以后，会输出许多 <单词，1> 形式的中间结果，然后 Sort 会把这些中间结果进行排序并把并把同一单词的出现次数合并成一个列表，得到 <key，List(value)>形式。例如，<Hello，<1，1，1，1，1>> 就表明 Hello 单词在 5 个地方出现过。

7.3.2 WordCount设计思路

- 如果使用 Combine，那么 Combine 会把每个单词的 List(value) 值进行合并，得到 <key，value> 形式，例如，<Hello，5> 表明 Hello 单词出现过 5 次。
- 在 Partition 阶段，会把 Combine 的结果分发给不同的 Reduce 任务。Reduce 任务接收到所有分配给自己的中间结果以后，就开始执行汇总计算工作，计算得到每个单词出现的次数并把结果输出到 HDFS 中。

7.3.3 WordCount的具体执行过程



7.3.3 WordCount的具体执行过程

- 对于WordCount程序任务，整个MapReduce过程实际的执行顺序如下。

①执行WordCount的用户程序（采用MapReduce编写），会被系统分发部署到集群中的多台机器上，其中一个机器作为Master，负责协调调度作业的执行，其余机器作为Worker，可以执行Map或Reduce任务。

②系统分配一部分Worker执行Map任务，一部分Worker执行Reduce任务；MapReduce将输入文件切分成M个分片，Master将M个分片分给处于空闲状态的N个Worker来处理；

7.3.3 WordCount的具体执行过程

③执行Map任务的Worker读取输入数据，执行Map操作，生成一系列<key，value>形式的中间结果，并将中间结果保存在内存的缓冲区中。

④缓冲区中的中间结果会被定期刷写到本地磁盘上，并被划分为R个分区，这R个分区会被分发给R个执行Reduce任务的Worker进行处理：Master会记录这R个分区在磁盘上的存储位置，并通知R个执行Reduce任务的Worker来“领取”属于自己处理的那些分区的数据。

7.3.3 WordCount的具体执行过程

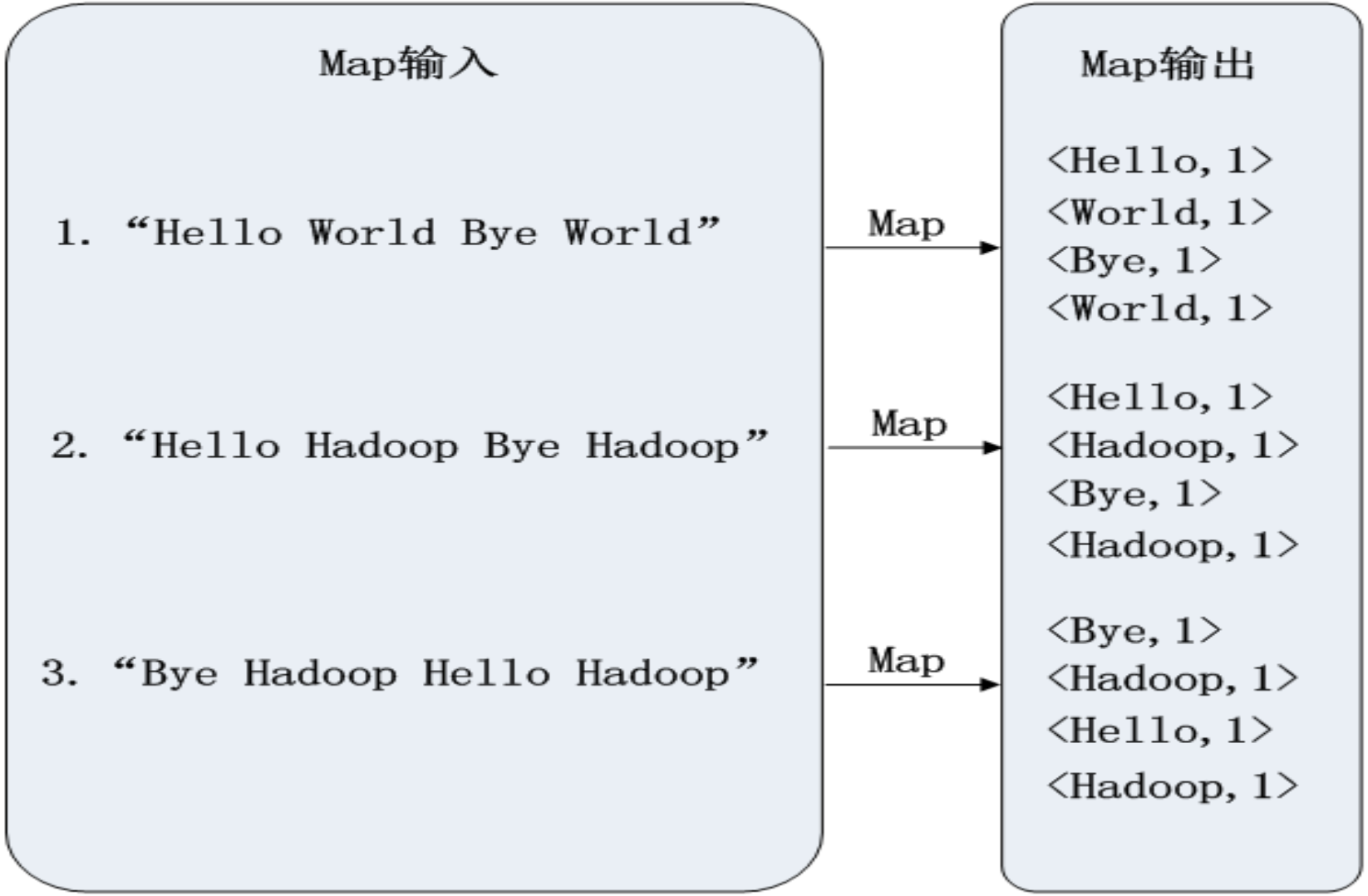
⑤执行Reduce任务的Worker收到Master的通知后，就到相应的Map机器上“领回”属于自己处理的分区，需要注意的是，正如之前在Shuffle过程阐述的那样，可能会有多个Map机器通知某个Reduce机器来领取数据，因此，一个执行Reduce任务的Worker，可能会从多个Map机器上领取数据。当位于所有Map机器上的、属于自己处理的数据，都已经领取回来以后，这个执行Reduce任务的Worker，会对领取到的键值对进行排序（如果内存中放不下需要用到外部排序），使得具有相同key的键值对聚集在一起，然后就可以开始执行具体的Reduce操作了。

7.3.3 WordCount的具体执行过程

⑥执行Reduce任务的Worker遍历中间数据，对每一个唯一key，执行Reduce函数，结果写入到输出文件中；执行完毕后，唤醒用户程序，返回结果。

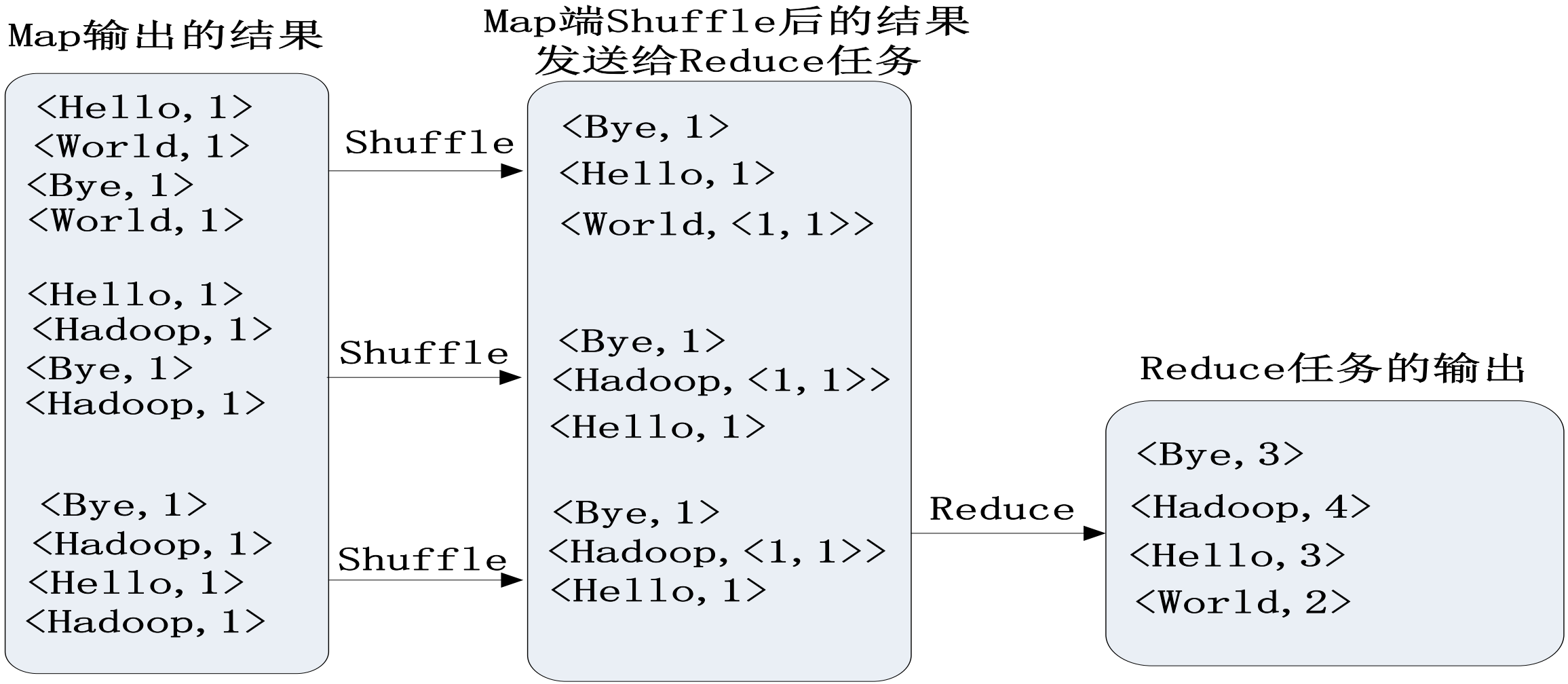
7.3.4 一个WordCount执行过程的实例

假设执行单词统计任务的MapReduce作业中，有3个执行Map任务的Worker和一个执行Reduce任务的Worker。一个文档包含3行内容，每行分配给一个Map任务来处理。



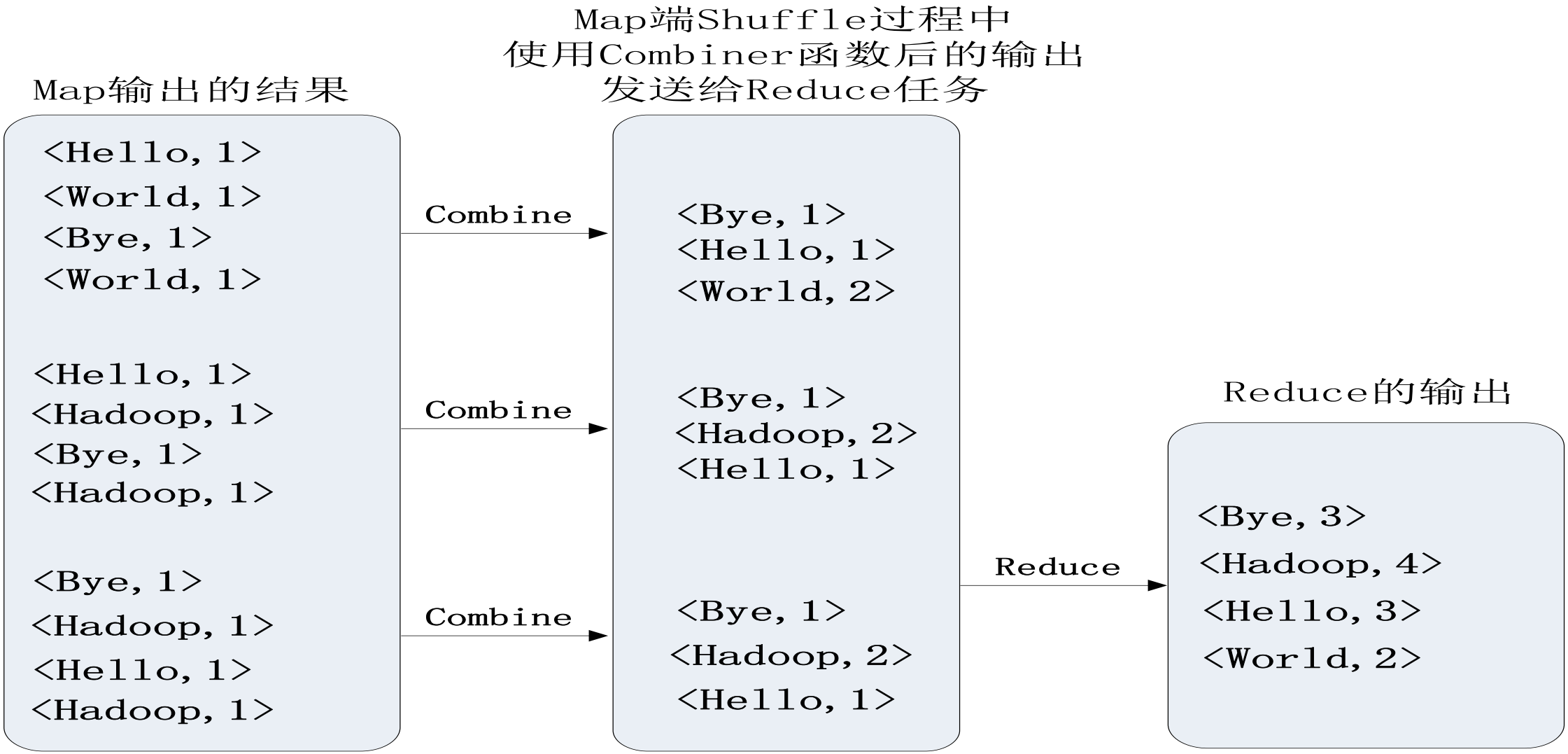
Map过程示意图

7.3.4 一个WordCount执行过程的实例



用户没有定义Combiner时的Reduce过程示意图

7.3.4 一个WordCount执行过程的实例



用户有定义Combiner时的Reduce过程示意图

7.5 MapReduce的具体应用

- **MapReduce可以很好地应用于各种计算问题**
  - 关系代数运算（选择、投影、并、交、差、连接）
  - 分组与聚合运算
  - 矩阵-向量乘法
  - 矩阵乘法

7.5 MapReduce编程实践

7.6.1 任务要求

7.6.2 编写Map处理逻辑

7.6.3 编写Reduce处理逻辑

7.6.4 编写main方法

7.6.5 编译打包代码以及运行程序

https://dblab.xmu.edu.cn/blog/2481/

7.6 本章小结

- 本章介绍了MapReduce编程模型的相关知识。MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce，并极大地方便了分布式编程工作，编程人员在不会分布式并行编程的情况下，也可以很容易将自己的程序运行在分布式系统上，完成海量数据集的计算。

- MapReduce执行的全过程包括以下几个主要阶段：从分布式文件系统读入数据、执行Map任务输出中间结果、通过 Shuffle阶段把中间结果分区排序整理后发送给Reduce任务、执行Reduce任务得到最终结果并写入分布式文件系统。在这几个阶段中，Shuffle阶段非常关键，必须深刻理解这个阶段的详细执行过程。

7.6 本章小结

- MapReduce具有广泛的应用，比如关系代数运算、分组与聚合运算、矩阵-向量乘法、矩阵乘法等
- 本章最后以一个单词统计程序为实例，详细演示了如何编写MapReduce程序代码以及如何运行程序