《大数据技术原理与应用》

http://dblab.xmu.edu.cn/post/bigdata

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第七章 MapReduce

（PPT版本号：2016年1月28日版本）

林子雨

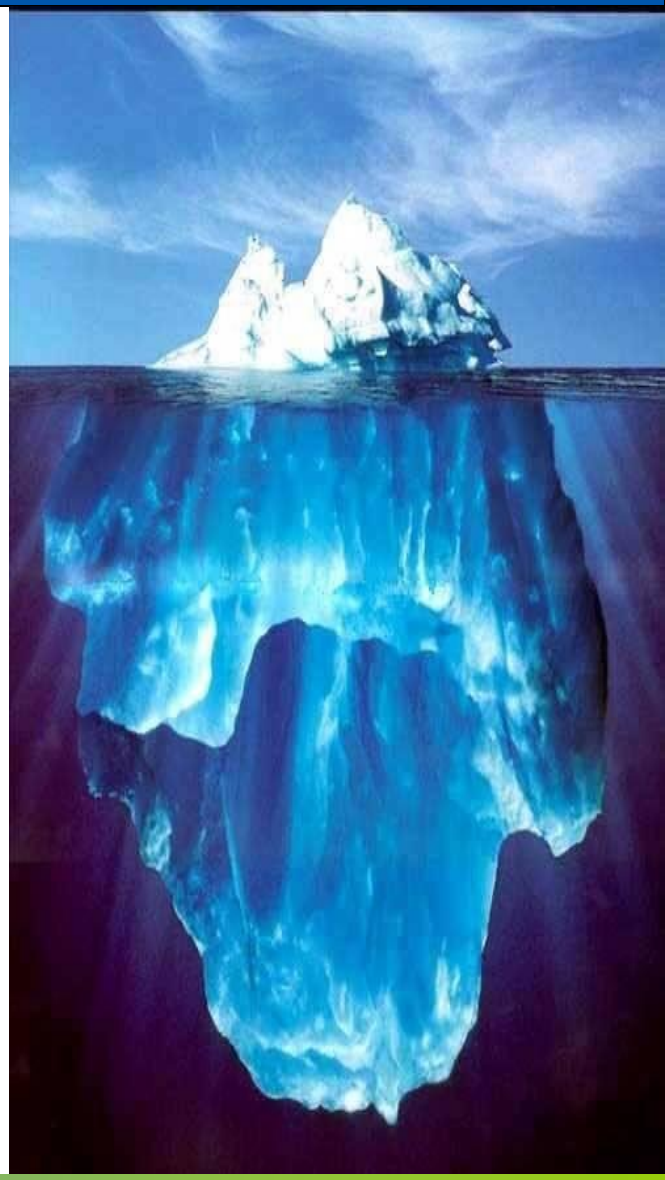
厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：**http://www.cs.xmu.edu.cn/linziyu**

提纲

- 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



本PPT是如下教材的配套讲义：
21世纪高等教育计算机规划教材
《大数据技术原理与应用
——概念、存储、处理、分析与应用》
（2015年6月第1版）
厦门大学 林子雨 编著，人民邮电出版社
ISBN:978-7-115-39287-9

欢迎访问《大数据技术原理与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

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”，大约每隔18个月性能翻一番
- 从2005年开始摩尔定律逐渐失效，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，集群中包括大量廉价服务器，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

7.1.2MapReduce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
- 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，并写入到分布式文件系统中
- 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

7.1.3Map和Reduce函数

表7-1 Map和Reduce

函数	输入	输出	说明
Map	<k_1,v_1>	List(<k_2,v_2>)	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)>	<k_3,v_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工作流程

- 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

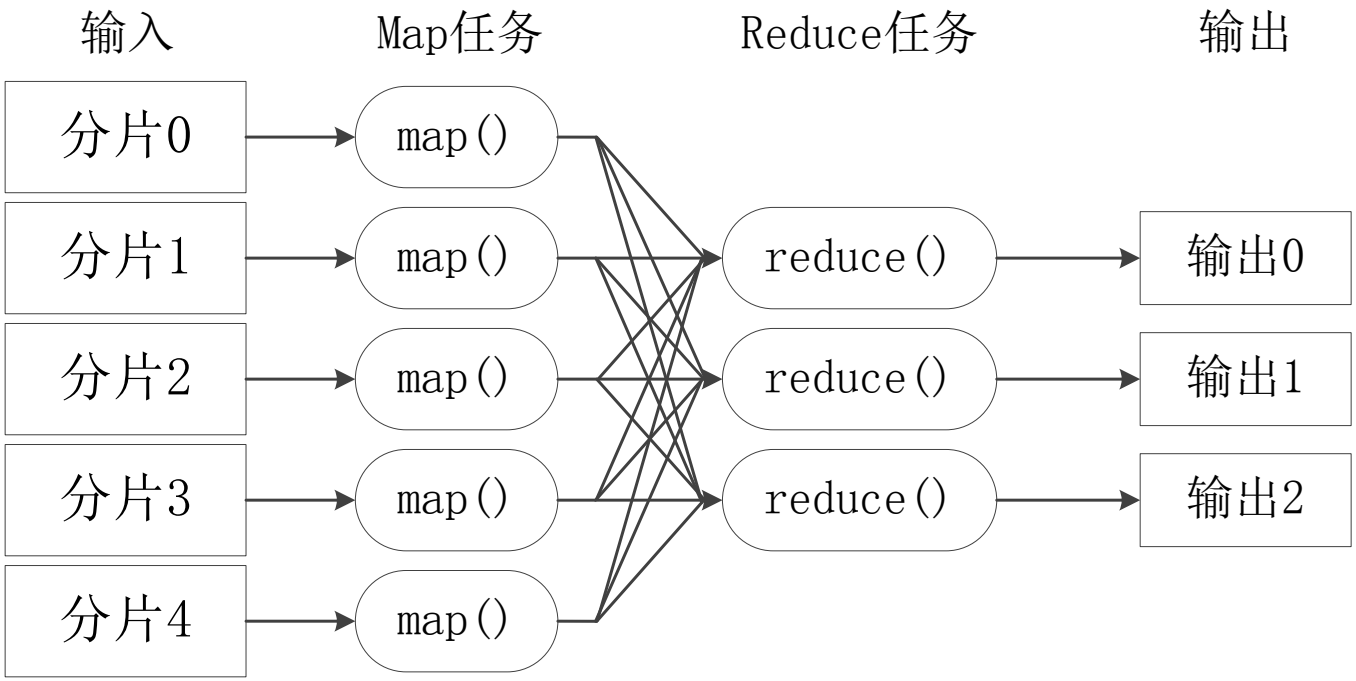


图7-1 MapReduce工作流程

7.2.2MapReduce各个执行阶段

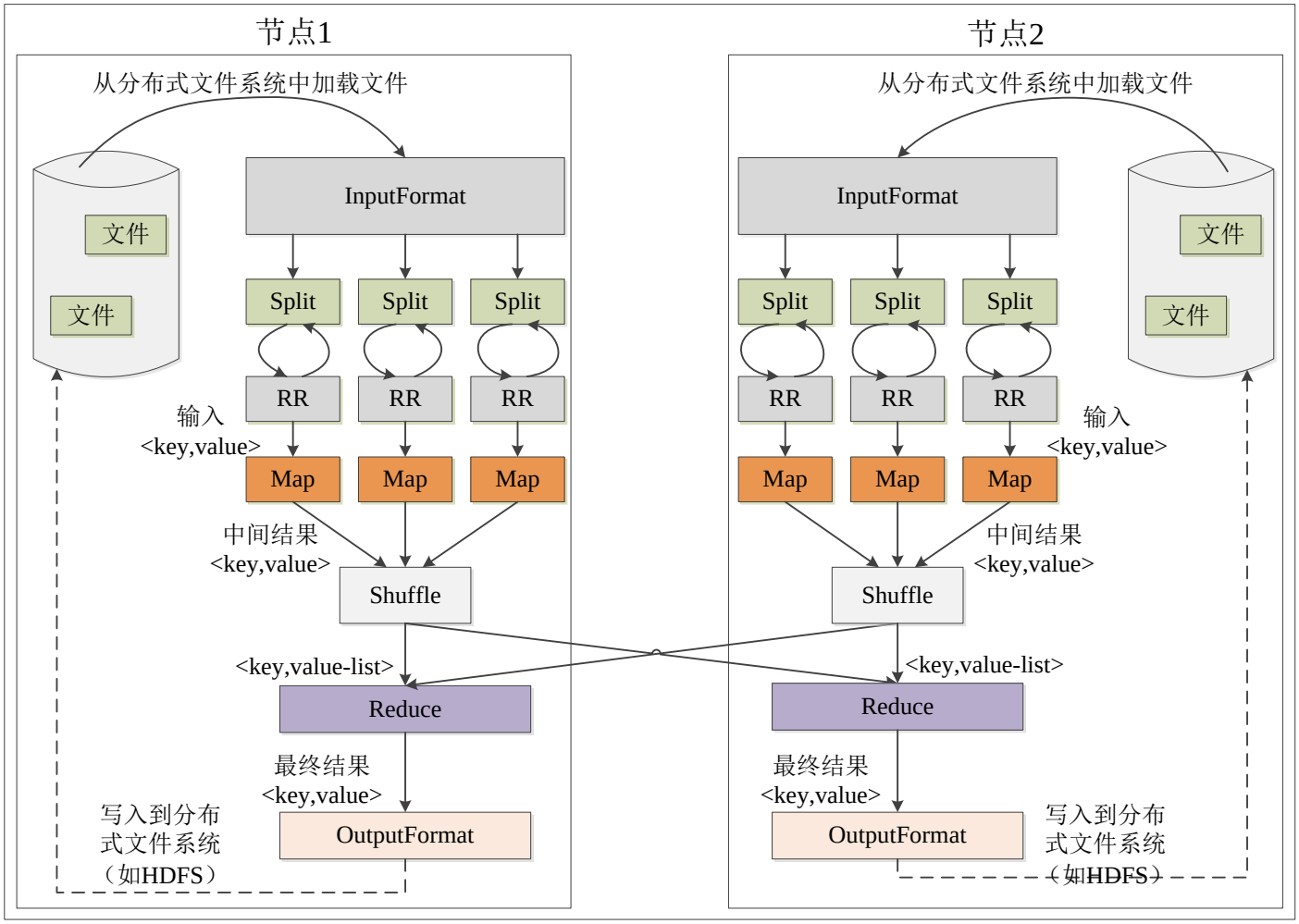
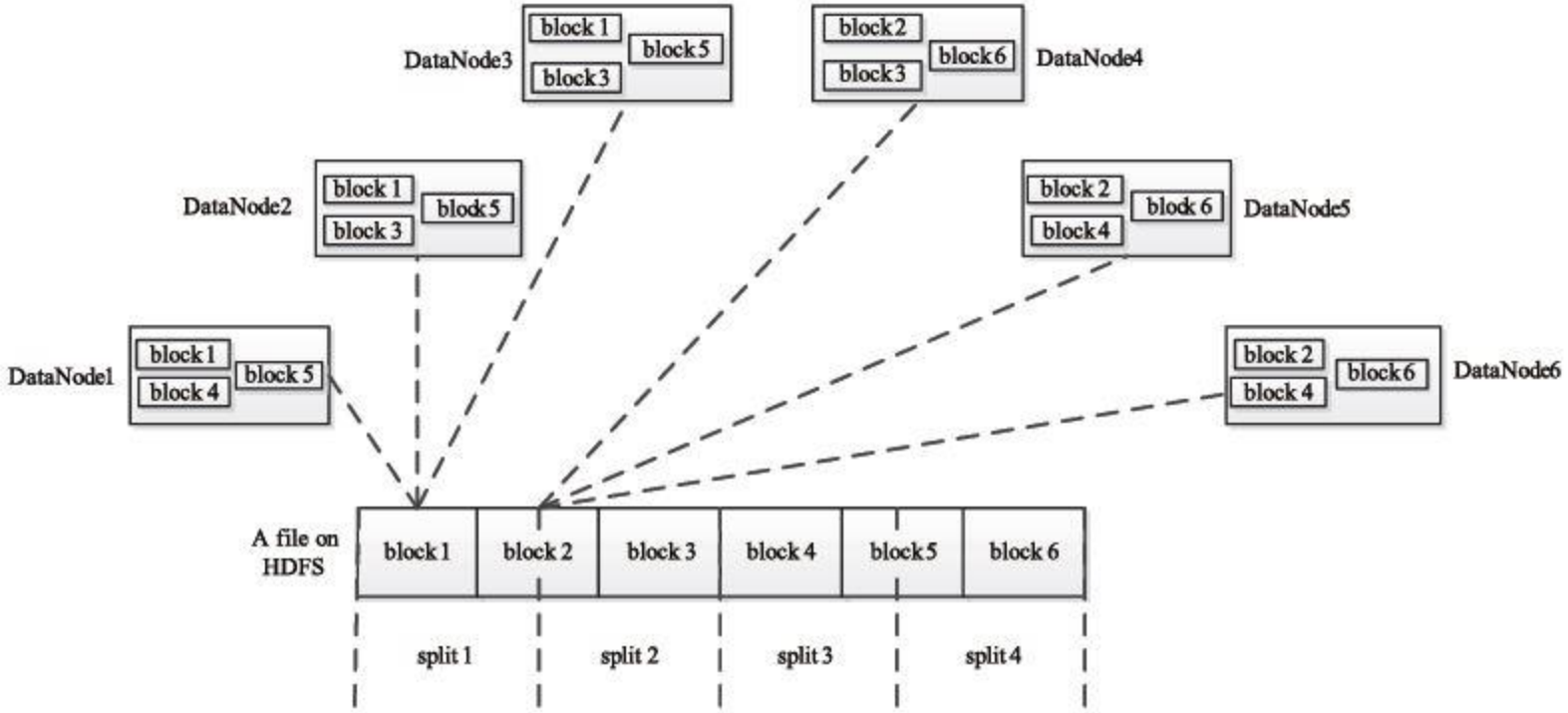


图7-2 MapReduce工作流程中的各个执行阶段

7.2.2MapReduce各个执行阶段



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。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。但需要注意的是，split 的多少决定了Map Task 的数目，因为每个split 只会交给一个Map Task 处理。

7.2.3Shuffle过程详解

1. Shuffle过程简介

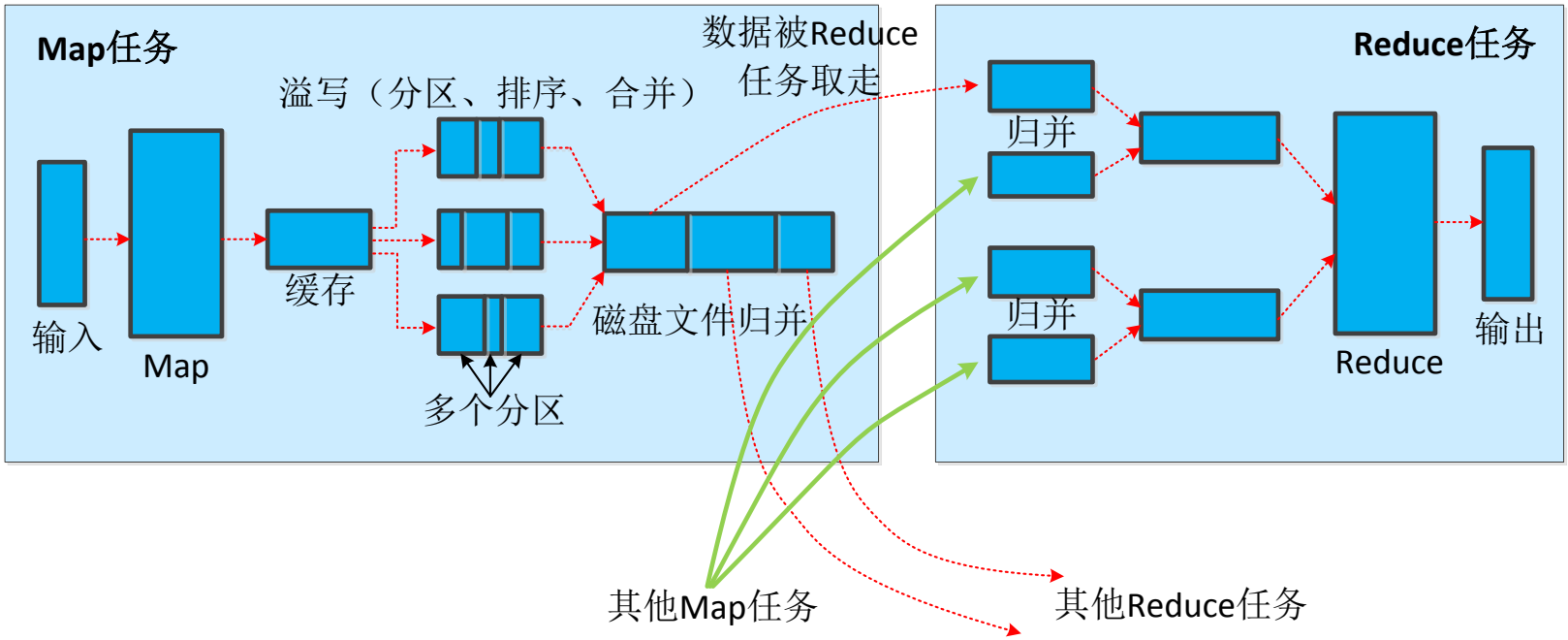
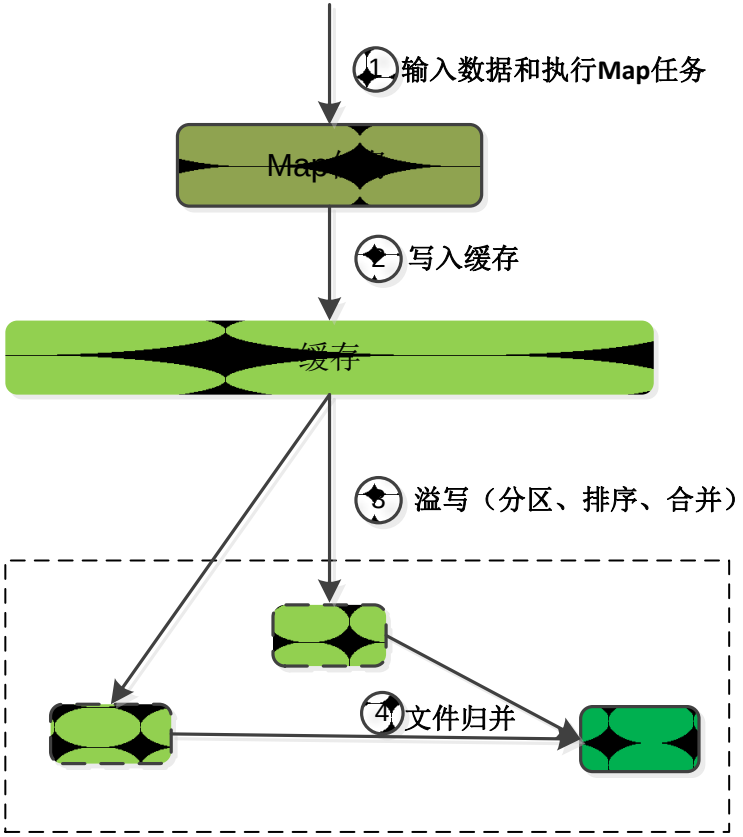


图7-3 Shuffle过程

7.2.3Shuffle过程详解

2. Map端的Shuffle过程



- 每个Map任务分配一个缓存
- MapReduce默认100MB缓存
- 设置溢写比例0.8
- 分区默认采用哈希函数
- 排序是默认的操作
- 排序后可以合并（Combine）
- 合并不能改变最终结果
- 在Map任务全部结束之前进行归并
- 归并得到一个大的文件，放在本地磁盘
- 文件归并时，如果溢写文件数量大于3则可以再次启动Combiner，少于3不需要

图7-4 Map端的Shuffle过程

7.2.3Shuffle过程详解

3. Reduce端的Shuffle过程

- Reduce任务通过RPC向JobTracker询问Map任务是否已经完成，若完成，则领取数据
- Reduce领取数据先放入缓存，来自不同Map机器，先归并，再合并，写入磁盘
- 多个溢写文件归并成一个或多个大文件，文件中的键值对是排序的

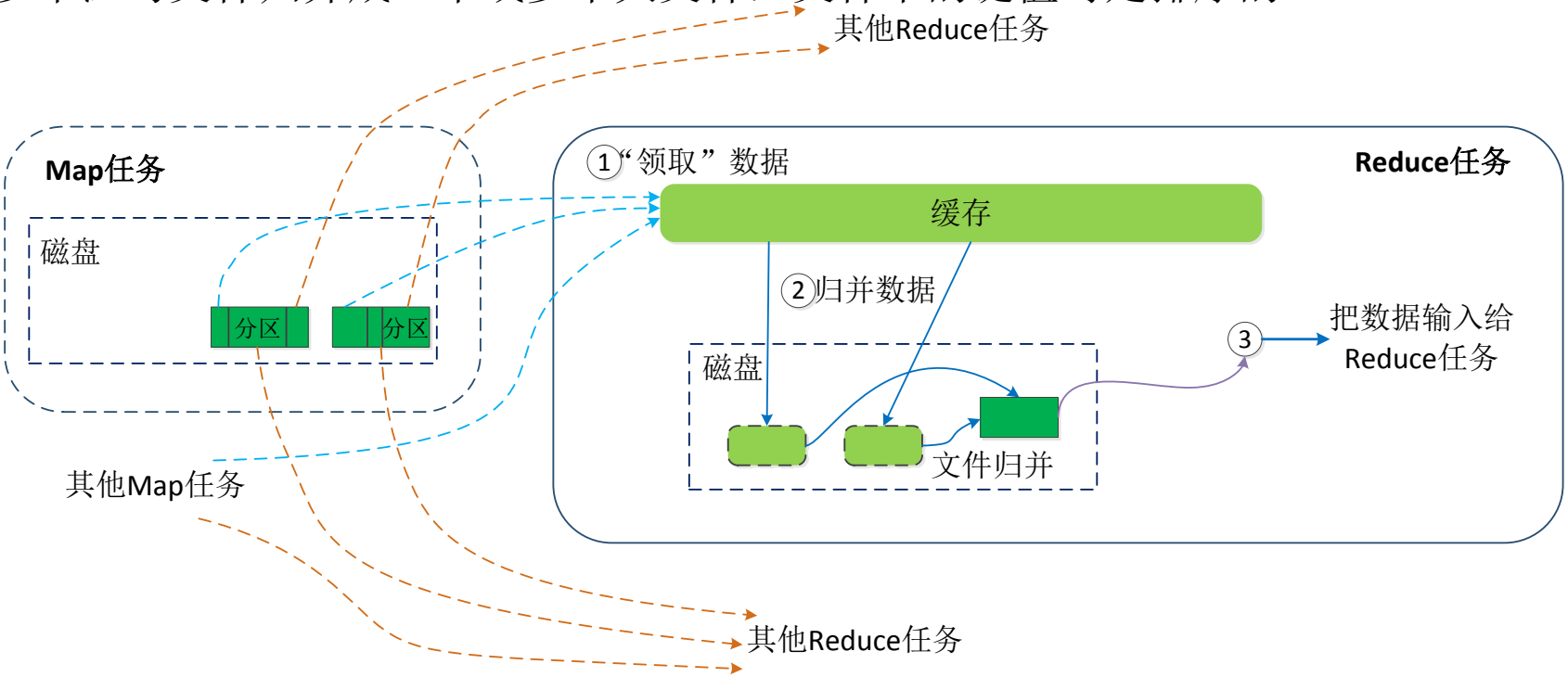


图7-5 Reduce端的Shuffle过程

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 MapReduce具体执行过程
- 7.3.4 一个WordCount执行过程的实例

7.3.1WordCount程序任务

表7-2 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

表7-3 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

7.3.2WordCount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
- 其次，确定MapReduce程序的设计思路
- 最后，确定MapReduce程序的执行过程

7.3.3MapReduce具体执行过程

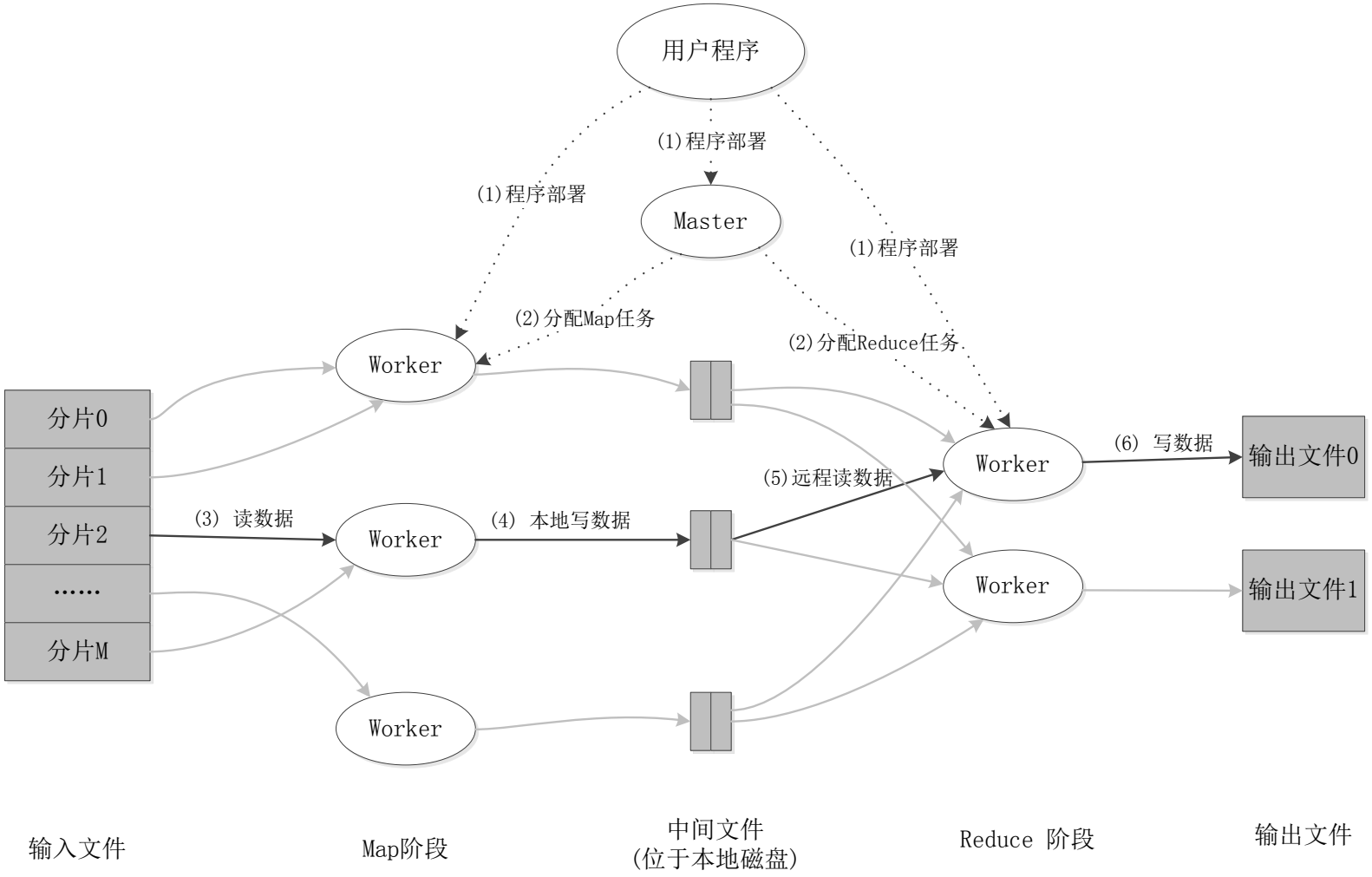


图7-6 WordCount执行过程

7.3.4一个WordCount执行过程的实例

图7-7 Map过程示意图

7.3.4一个WordCount执行过程的实例

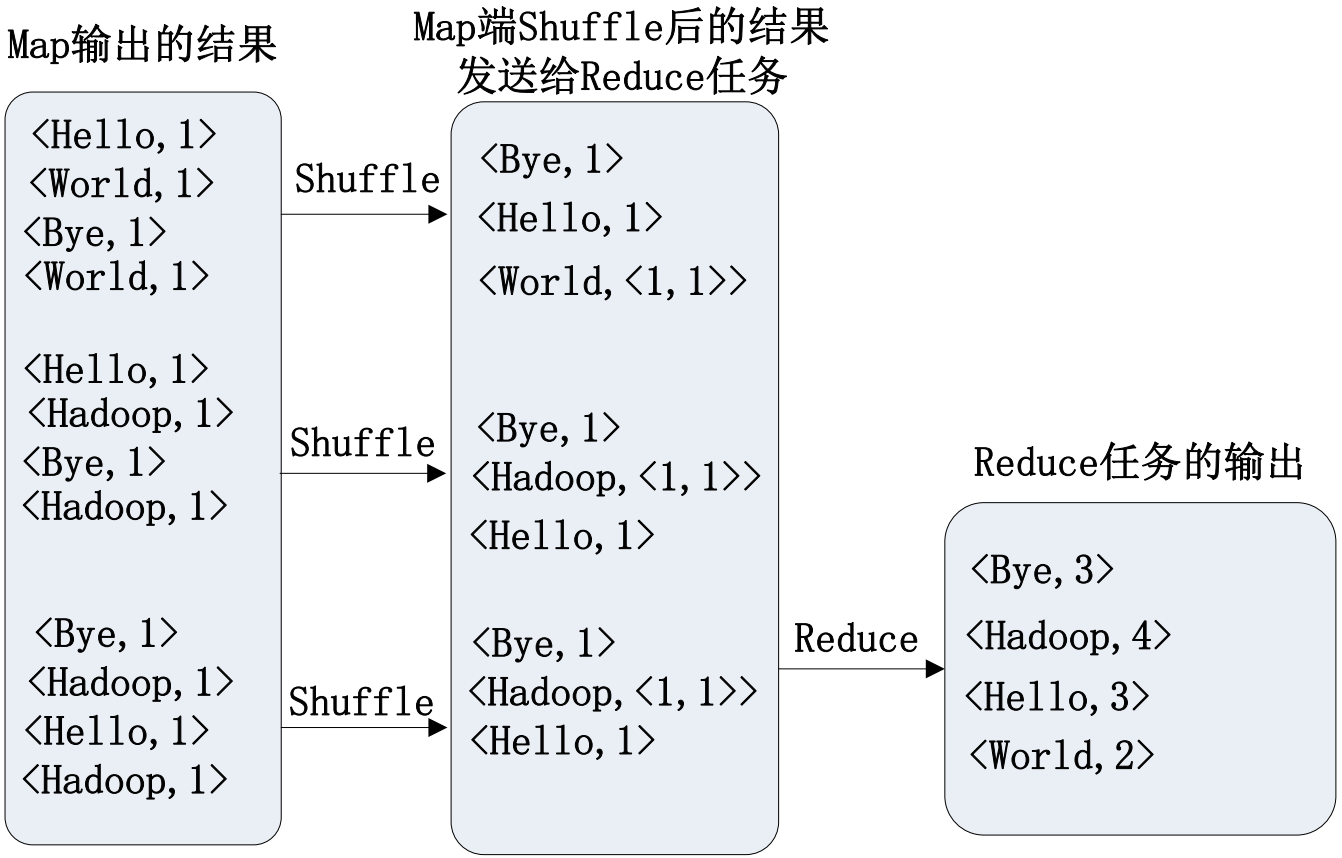


图7-8 用户没有定义Combiner时的Reduce过程示意图

7.3.4一个WordCount执行过程的实例

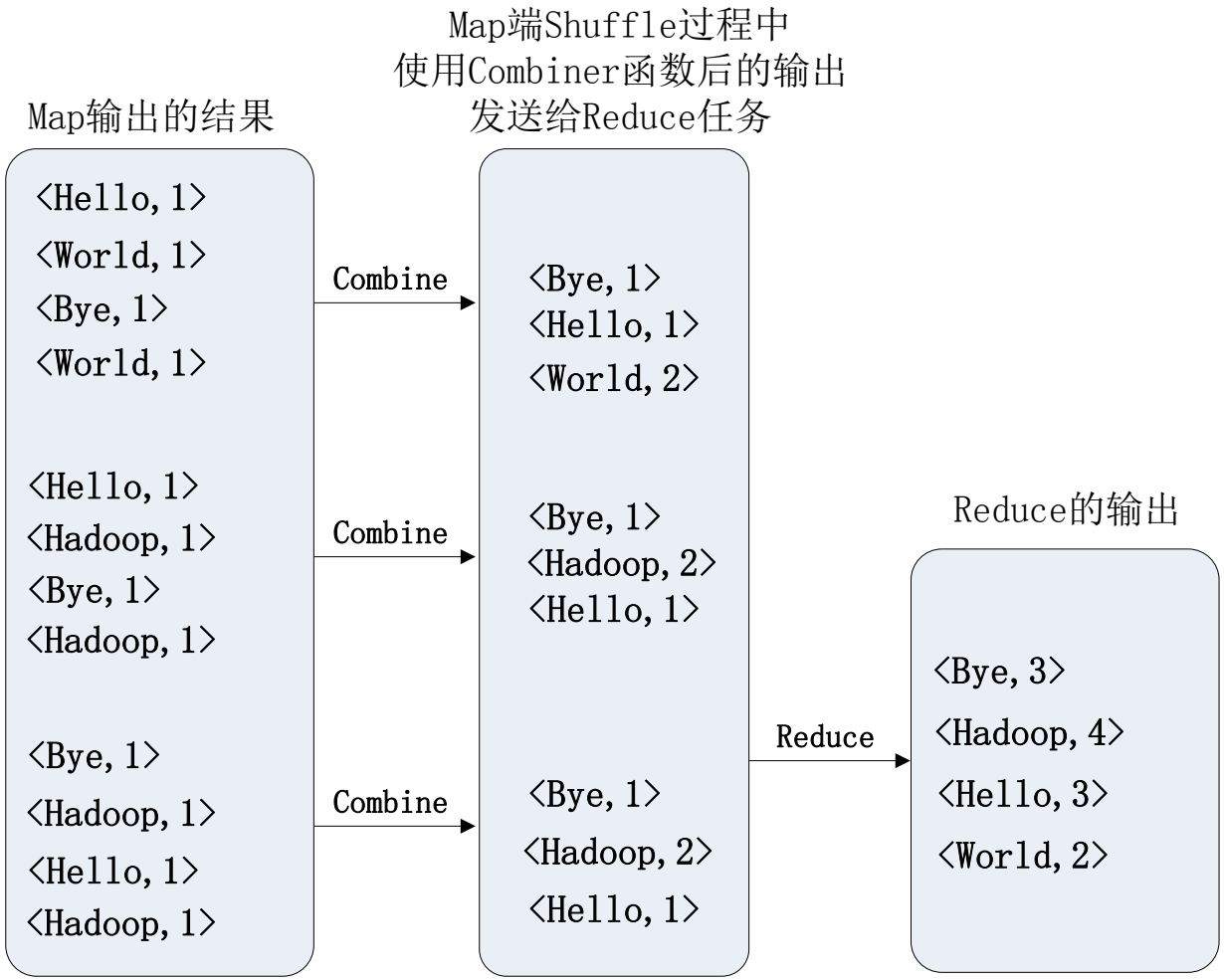


图7-9 用户有定义Combiner时的Reduce过程示意图

7.4MapReduce的具体应用

- MapReduce可以很好地应用于各种计算问题，这里以关系代数运算、分组与聚合运算、矩阵-向量乘法、矩阵乘法为例，介绍如何采用MapReduce计算模型来实现各种运算
- 7.4.1MapReduce在关系代数运算中的应用
- 7.4.2分组与聚合运算
- 7.4.3矩阵-向量乘法
- 7.4.4矩阵乘法

7.4.1MapReduce在关系代数运算中的应用

- 针对数据的很多运算可以很容易地采用数据库查询语言来表示，即使这些查询本身并不在数据库管理系统中执行
- 关系数据库中的关系(relation)可以看做是由一系列属性值组成的表，关系中的行称为元组(tuple)，属性的集合称为关系的模式
- 下面介绍基于MapReduce模型的关系上的标准运算，包括选择、投影、并、交、差以及自然连接

7.4.1MapReduce在关系代数运算中的应用

1. 关系的选择运算

- 对于关系的选择运算，只需要Map过程就能实现，对于关系R中的每个元组t，检测是否是满足条件的元组，如果满足条件，则输出键值对<t,t>，也就是说，键和值都是t。这时的Reduce函数就只是一个恒等式，对输入不做任何变换就直接输出

7.4.1MapReduce在关系代数运算中的应用

2. 关系的投影运算

- 假设对关系R投影后的属性集为S，在Map函数中，对于R中的每个元组t，剔除t中不属于S的字段得到元组t'，输出键值对<t'，t'>。对于Map任务产生的每个键，可能存在一个或多个键值对<t'，t'>，因此，需要通过Reduce函数来剔除冗余，把属性值完全相同的元组合并起来得到<t',<t',t',...>>，剔除冗余后只输出一个<t',t'>。

7.4.1MapReduce在关系代数运算中的应用

3. 关系的并、交、差运算

- 对两个关系求并集时，Map任务将两个关系的元组转换成键值对<t,t>，Reduce任务则相当于一个剔除冗余数据的过程(合并到一个文件中)
- 对两个关系求交集时，使用与求并集相同的Map过程，在Reduce过程中，如果键t有两个相同值与它关联，则输出一个元组<t,t>，如果与键关联的只有一个值，则输出空值（NULL）
- 对两个关系求差时，Map过程产生的键值对，不仅要记录元组的信息，还要记录该元组来自于哪个关系（R或S），Reduce过程中按键值相同的t合并后，与键t相关联的值如果只有R（说明该元组只属于R，不属于S），就输出元组，其他情况均输出空值

7.4.1MapReduce在关系代数运算中的应用

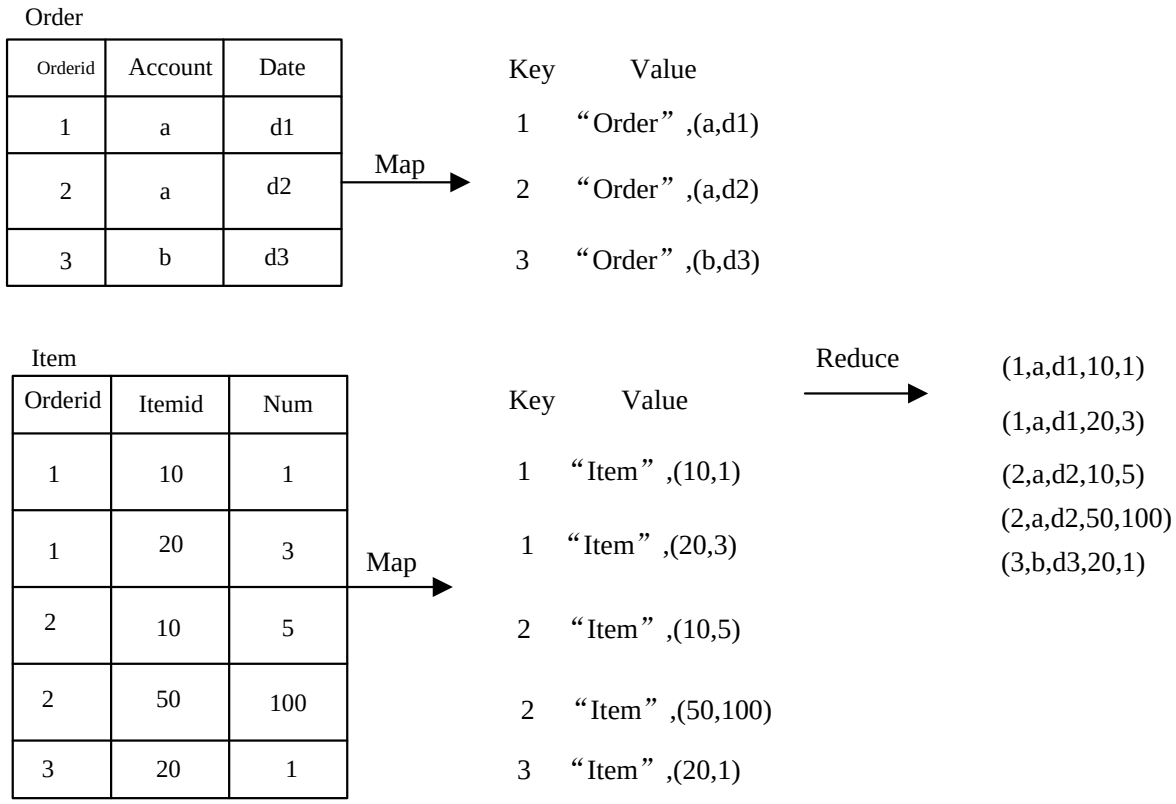
4. 关系的自然连接

- 有关系R(A，B)和S(B,C)，将属性B的值分别作为它们的键和元组中其他属性值关联起来
- 使用Map过程，把来自R的每个元组<a,b>转换成一个键值对<b,<R,a>>，其中的键就是属性B的值。注意，这里把关系R包含到值中，这样做使得我们可以在Reduce阶段，只把那些来自R的元组和来自S的元组进行匹配。类似地，使用Map过程，把来自S的每个元组<b,c>，转换成一个键值对<b,<S,c>>
- 所有具有相同B值的元组被发送到同一个Reduce进程中，Reduce进程负责对这些元组进行合并。假设使用k个Reduce进程，这里选择一个哈希函数h，它可以把属性B的值映射到k个哈希桶，每个哈希值对应一个Reduce进程。每个Map进程把键是b 的键值对都发送到与哈希值h(b)对应的Reduce进程
- Reduce进程的输出则是连接后的元组<a,b,c>，输出被写到一个单独的输出文件中

7.4.1MapReduce在关系代数运算中的应用

4. 关系的自然连接

- 以某工厂接到的订单与仓库货存为例，演示关系自然连接运算的MapReduce过程



7.4.2分组与聚合运算

- 词频计算就是典型的分组聚合运算
- 在Map过程中，选择关系的某一字段（也可以是某些属性构成的属性表）的值作为键，其他字段的值作为与键相关联的值。
- 在Reduce过程中，对键值相同的键值对的值施加某种聚合运算，如SUM(求和)、COUNT(计数)、AVG(求平均值)、MIN和MAX(求最小最大值)等，输出则为<键，聚合运算结果>

7.4.3矩阵-向量乘法

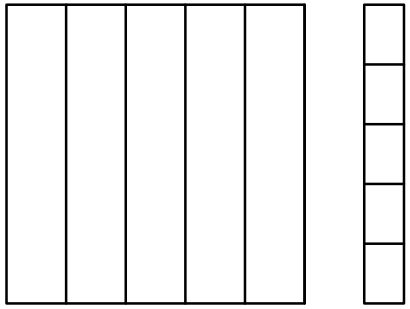
- 假定一个n维向量V，其第j个元素记为v_j，和一个$n \times n$的矩阵M，其第i行第j列元素记为m_{ij}，矩阵M和向量V的乘积是一个n维向量X，其第i个元素$x_i = \sum_{j=1}^{n} m_{ij} v_j$。
- 矩阵M和向量V各自会在分布式文件系统（比如HDFS）中存成一个文件。假定我们可以获得矩阵元素的行列下标，例如从矩阵元素在文件中的位置来获得，或者从元素显式存储的三元组<i,j,m_{ij}>中来获得。

7.4.3矩阵-向量乘法

- 每个Map任务将整个向量V和矩阵M的一个文件块作为输入。对每个矩阵元素，Map任务会产生键值对<i, $m_{ij}v_j$>。
- 计算所得的n个求和项的键都相同，即都是i。
- Reduce任务将所有与给定键i关联的值相加即可得到<i, x_i >。

7.4.3矩阵-向量乘法

- 如果n的值过大，使向量V无法完全放入内存中，那么，在计算过程中就需要多次将向量的一部分导入内存，这就会导致大量的磁盘访问
- 一种替代方案是，将矩阵分割成多个宽度相等的垂直条，同时，将向量分割成同样数目的水平条，每个水平条的高度等于矩阵垂直条的宽度



- 矩阵第i个垂直条只和第i个水平条相乘。因此，可以将矩阵的每个条存成一个文件，同样，将向量的每个条存成一个文件。矩阵某个条的一个文件块及对应的完整向量条输送到每个Map任务。然后，Map任务和Reduce任务可以按照前述过程进行

7.4.4矩阵乘法

- 矩阵M第i行第j列的元素记为m_{ij}，矩阵N中的第j行第k列的元素记为n_{jk}，矩阵$P=M\times N$，第i行第k列元素为$p_{ik}=\sum_j m_{ij}n_{jk}$。
- 把矩阵看作一个带有三个属性的关系：行下标、列下标和值。因此，矩阵M可以看作关系M(I,J,V)，元组为<i，j，m_{ij}>，矩阵N可以看作关系N(J,K,W)，元组为<j,k,n_{jk} >。
- 矩阵乘法可以看作是一个自然连接运算再加上分组聚合运算。关系M和N根据公共属性J将每个元组连接得到元组<i,j,k,v,w>，这个五字段元组代表了两个矩阵的元素对<m_{ij} , n_{jk} >，对矩阵元素进行求积运算后可以得到四字段元组<i,j,k, $v\times w$ >，然后可以进行分组聚合运算，其中，I、K是分组属性，$V\times W$的和是聚合结果。综上所述，矩阵乘法可以通过两个MapReduce运算的串联来实现。

7.4.4矩阵乘法

1. 自然连接阶段

- Map函数：对每个矩阵元素产生一个键值对<j,<M,i, m_{ij} >>，对每个矩阵元素产生一个键值对<j,<N,k, n_{jk} >>
- Reduce函数：对每个相同键j，输出所有满足形式<j,<i,k,$m_{ij}n_{jk}$ >>的元组。

7.4.4矩阵乘法

2.分组聚合阶段

- Map函数：对自然连接阶段产生的键值对<j,<<i_1,k_1,v_1>,<i_2,k_2,v_2>,...<i_p,k_p,v_p>>>（其中每个v_q是对应的m_{ij}和n_{jq}的乘积），Map任务会产生p个键值对<<<i_1,k_1>,v_1>,<<i_2,k_2>,v_2>...,<<i_p,k_p>,v_p>>。
- Reduce函数：对每个键<i,k>，计算与此键关联的所有值的和，结果记为<<i,k>,v>，其中，v就是矩阵P的第i行第k列的值。

7.5 MapReduce编程实践

- 7.5.1任务要求
- 7.5.2编写Map处理逻辑
- 7.5.3编写Reduce处理逻辑
- 7.5.4 编写main方法
- 7.5.5 编译打包代码以及运行程序
- 7.5.6 Hadoop中执行MapReduce任务的几种方式

7.5.1 任务要求

文件A的内容如下：

China is my motherland

I love China

文件B的内容如下：

I am from China

期望结果如右侧所示：

I	2
is	1
China	3
my	1
love	1
am	1
from	1
motherland	1

7.5.2 编写Map处理逻辑

- Map输入类型为<key,value>
- 期望的Map输出类型为<单词，出现次数>
- Map输入类型最终确定为<Object,Text>
- Map输出类型最终确定为<Text,IntWritable>

```
public static class MyMapper extends
Mapper<Object,Text,Text,IntWritable>{
                private final static IntWritable one = new
IntWritable(1);
                private Text word = new Text();
                public void map(Object key, Text value, Context
context) throws IOException,InterruptedException{
                        StringTokenizer itr = new
StringTokenizer(value.toString());
                        while (itr.hasMoreTokens())
                        {
                                word.set(itr.nextToken());
                                context.write(word,one);
                        }
                }
        }
```

7.5.3 编写Reduce处理逻辑

- 在Reduce处理数据之前，Map的结果首先通过Shuffle阶段进行整理
- Reduce阶段的任务：对输入数字序列进行求和
- Reduce的输入数据为<key,Iterable容器>

Reduce任务的输入数据：
<”I”,<1,1>>
<”is”,1>
……
<”from”,1>
<”China”,<1,1,1>>

7.5.3 编写Reduce处理逻辑

```
public static class MyReducer extends
Reducer<Text,IntWritable,Text,IntWritable>{
                    private IntWritable result = new IntWritable();
                    public void reduce(Text key,
Iterable<IntWritable> values, Context context) throws
IOException,InterruptedException{
                        int sum = 0;
                        for (IntWritable val : values)
                        {
                            sum += val.get();
                        }
                        result.set(sum);
                        context.write(key,result);
                    }
    }
```

7.5.4 编写main方法

```
public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();  //程序运行时参数
        String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
        if (otherArgs.length != 2)
        {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf,"word count");  //设置环境参数
        job.setJarByClass(WordCount.class);  //设置整个程序的类名
        job.setMapperClass(MyMapper.class);  //添加MyMapper类
        job.setReducerClass(MyReducer.class);  //添加MyReducer类
        job.setOutputKeyClass(Text.class);  //设置输出类型
        job.setOutputValueClass(IntWritable.class);  //设置输出类型
        FileInputFormat.addInputPath(job,new Path(otherArgs[0]));  //设置输入文件
        FileOutputFormat.setOutputPath(job,new Path(otherArgs[1])); //设置输出文件
        System.exit(job.waitForCompletion(true)?0:1);
    }
```

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount{
    public static class MyMapper extends Mapper<Object,Text,Text,IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws IOException,InterruptedException{
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()){
                word.set(itr.nextToken());
                context.write(word,one);
            }
        }
    }

    public static class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable>{
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException{
            int sum = 0;
            for (IntWritable val : values)
            {
                sum += val.get();
            }
            result.set(sum);
            context.write(key,result);
        }
    }
    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
        if (otherArgs.length != 2)
        {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf,"word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        job.setOutputKeyClass(Text.class);
```

7.5.5 编译打包代码以及运行程序

实验步骤：

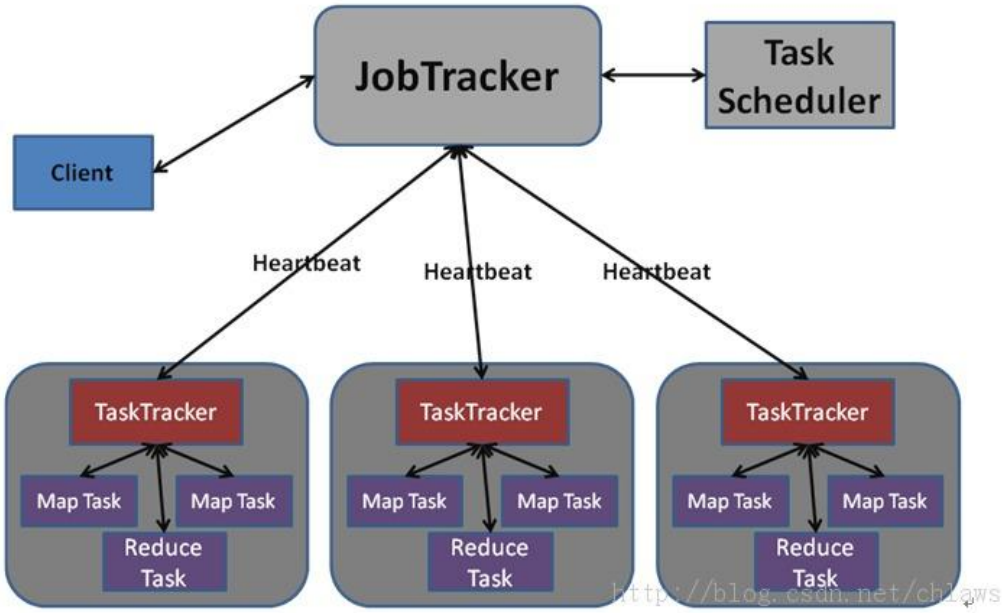
- 使用java编译程序，生成.class文件
- 将.class文件打包为jar包
- 运行jar包
- 查看结果

7.5.6 Hadoop中执行MapReduce任务的几种方式

- Hadoop jar
- Pig
- Hive
- Python
- shell脚本

在解决问题的过程中，开发效率、执行效率都是要考虑的因素，不要太局限于某一种方法

7.6 MapReduce架构



和HDFS一样，MapReduce也是采用Master/Slave的架构

7.6 MapReduce架构

MapReduce主要有以下4个部分组成：

1）Client

2）JobTracker

JobTracke负责资源监控和作业调度。JobTracker 监控所有TaskTracker 与job的健康状况，一旦发现失败，就将相应的任务转移到其他节点；同时，JobTracker 会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器，而调度器会在资源出现空闲时，选择合适的任务使用这些资源。在Hadoop 中，任务调度器是一个可插拔的模块，用户可以根据自己的需要设计相应的调度器。

3）TaskTracker

TaskTracker 会周期性地通过Heartbeat 将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）。TaskTracker 使用“slot”等量划分本节点上的资源量。“slot”代表计算资源（CPU、内存等）。一个Task 获取到一个slot 后才有机会运行，而Hadoop 调度器的作用就是将各个TaskTracker 上的空闲slot 分配给Task 使用。slot 分为Map slot 和Reduce slot 两种，分别供MapTask 和Reduce Task 使用。TaskTracker 通过slot 数目（可配置参数）限定Task 的并发度。

4）Task

Task 分为Map Task 和Reduce Task 两种，均由TaskTracker 启动。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。

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
- 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

附录：主讲教师

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度厦门大学奖教金获得者。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，编著出版中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》并成为畅销书籍，编著并免费网络发布40余万字中国高校第一本闪存数据库研究专著《闪存数据库概念与技术》；主讲厦门大学计算机系本科生课程《数据库系统原理》和研究生课程《分布式数据库》《大数据技术基础》。具有丰富的政府和企业信息化培训经验，曾先后给中国移动通信集团公司、福州马尾区政府、福建省物联网科学研究院、石狮市物流协会、厦门市物流协会、福建龙岩卷烟厂等多家单位和企业开展信息化培训，累计培训人数达2000人以上。

附录：大数据学习教材推荐

扫一扫访问教材官网

《大数据技术原理与应用——概念、存储、处理、分析与应用》，由厦门大学计算机科学系林子雨博士编著，是中国高校第一本系统介绍大数据知识的专业教材。

全书共有13章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

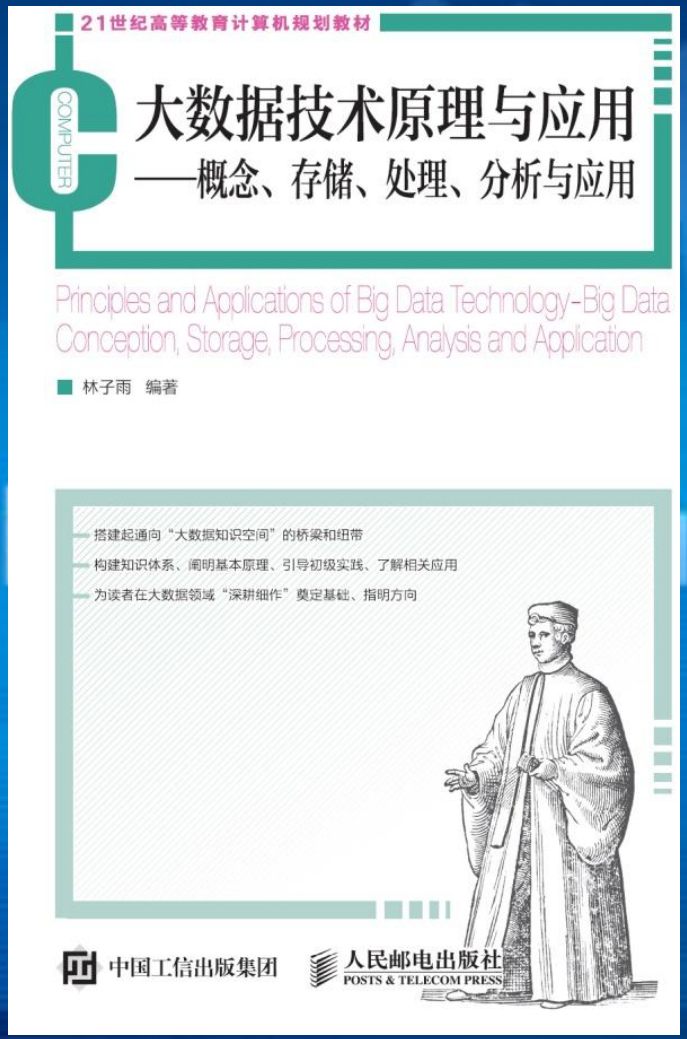
附录：中国高校大数据课程公共服务平台

中国高校大数据课程
公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片



Department of Computer Science, Xiamen University, 2016