BIG DATA

6.MapReduce编程

共享计算的几种方式

单CPU多核

单机多CPU

“摩尔定律”的失效

- “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

为什么Google还需要MapReduce?

- 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

MapReduce模型简介

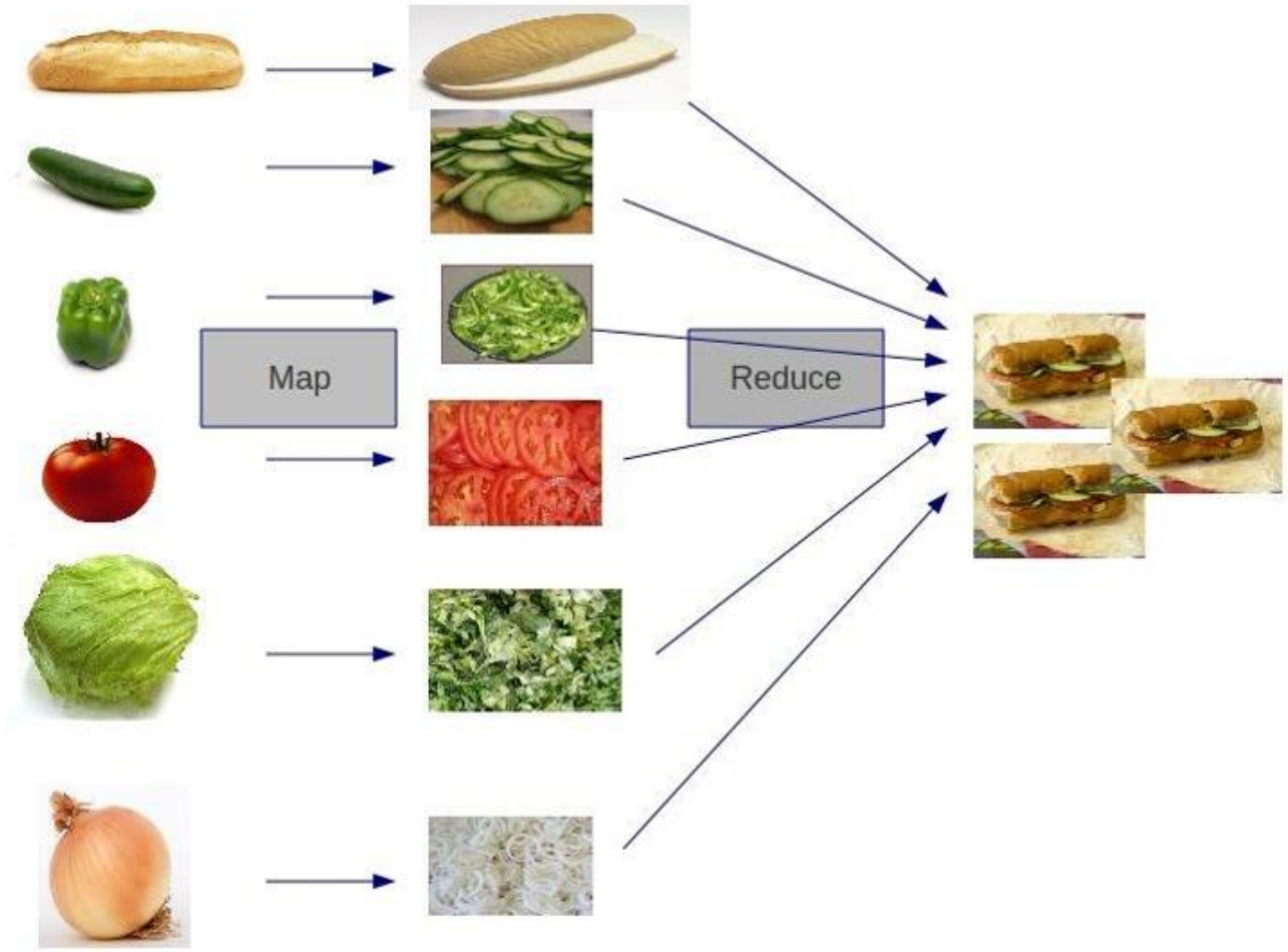
- MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce
- 编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量数据的计算
- MapReduce采用“**分而治之**”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理
- MapReduce设计的一个理念就是“**计算向数据靠拢**”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销
- MapReduce框架采用了Master/Slave架构，包括一个Master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

以一阶线性函数计算为例

- $y=k_1x_1+k_2x_2+k_3x_3$
- 该计算分为两步：
  1. 各个参与计算的x乘以参数k
  2. 将上一步计算的所有结果加到一起

- 数据计算往往分为两步：
  1. 将格式不同的数据进行格式转换变为能够直接使用的数据
  2. 将已经转换好格式的数据代入公式进行计算

以三明治为例



以图书馆管理员为例

- 几名图书馆管理员要数图书馆中的所有书。
- A数1号书架，B数2号书架，...
  - **Map**
  - 人越多，数书就更快
- 现在AB...集中到一起，把所有人的统计数加在一起。
  - **Reduce**

分而治之

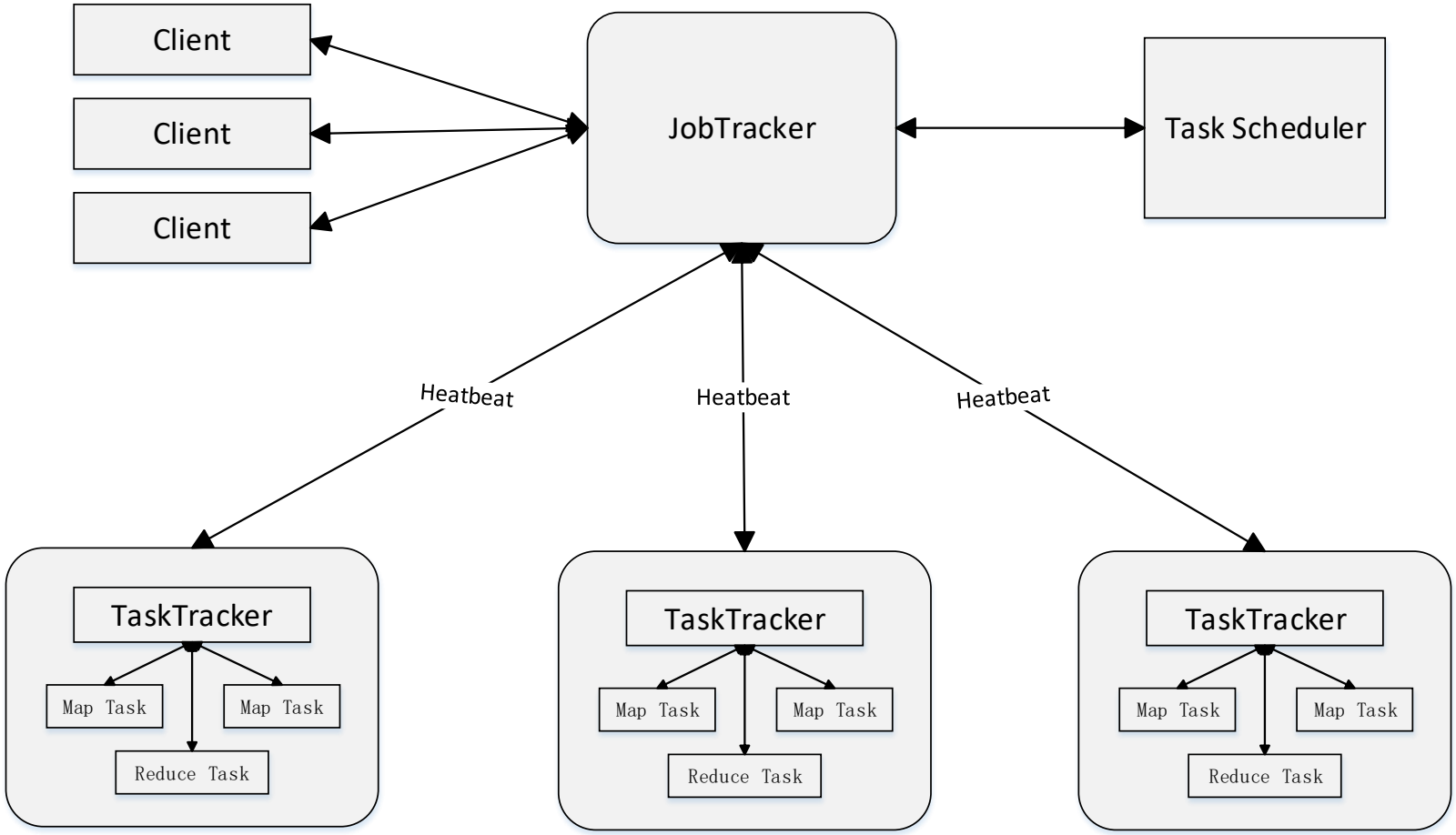
- **Mapper负责“分”**，即把复杂的任务分解为若干个“简单的任务”来处理。“简单的任务”包含三层含义：
  - 一是数据或计算的规模相对原任务要大大缩小；
  - 二是就近计算原则，即任务会分配到存放着所需数据的节点上进行计算；
  - 三是这些小任务可以并行计算，彼此间几乎没有依赖关系。
- **Reducer**负责对map阶段的**结果进行汇总**。至于需要多少个Reducer，用户可以根据具体问题，通过在mapred-site.xml配置文件里设置参数mapred.reduce.tasks的值，缺省值为1。

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	$<k_2,\text{List}(v_2)>$ 如： <“a”,<1,1,1>>	$<k_3,v_3>$ <“a”,3>	输入的中间结果$<k_2,\text{List}(v_2)>$中的List(v_2)表示是一批属于同一个k_2的value

Hadoop下MapReduce计算的体系结构

- MapReduce体系结构主要由四个部分组成，分别是：Client、JobTracker、TaskTracker以及Task



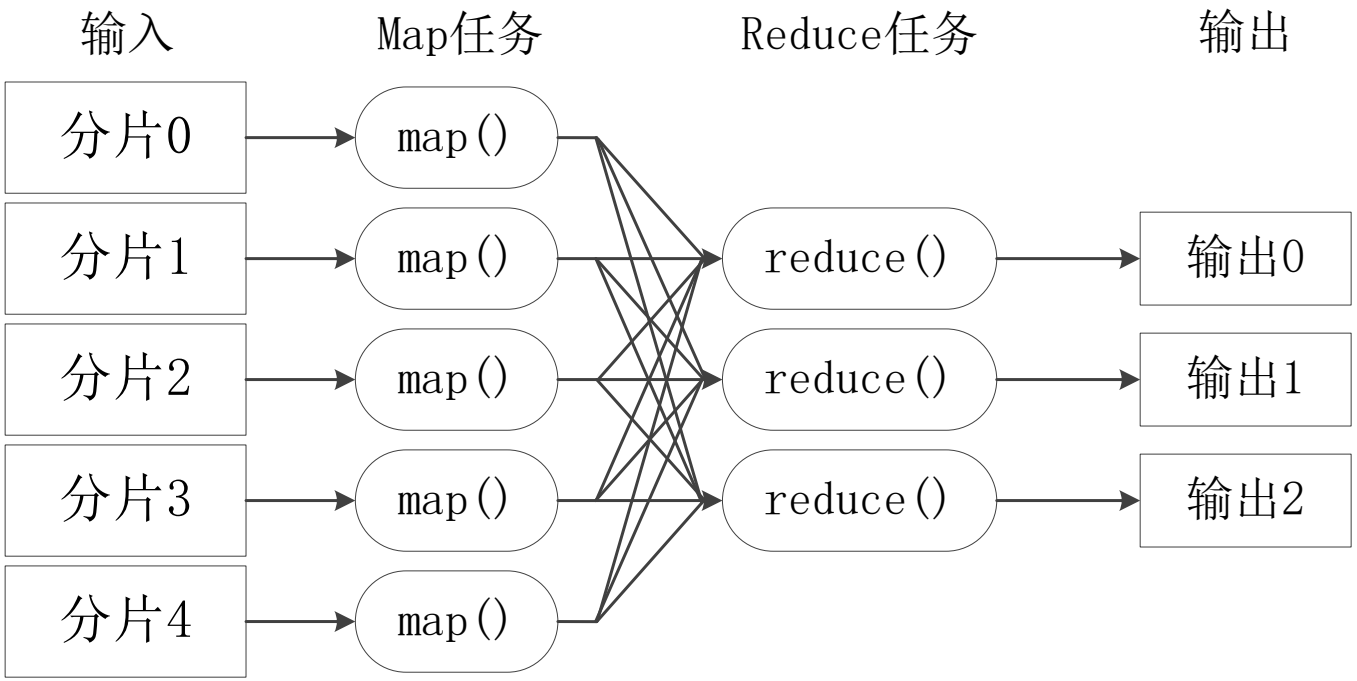
Hadoop下MapReduce计算的体系结构

- MapReduce主要有以下4个部分组成：
- **1）Client**
  - 用户编写的MapReduce程序通过Client提交到JobTracker端
  - 用户可通过Client提供的一些接口查看作业运行状态
- **2）JobTracker**
  - JobTracker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
  - 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（TaskScheduler），而调度器会在资源出现空闲时，选择合适的任务去使用这些资源

Hadoop下MapReduce计算的体系结构

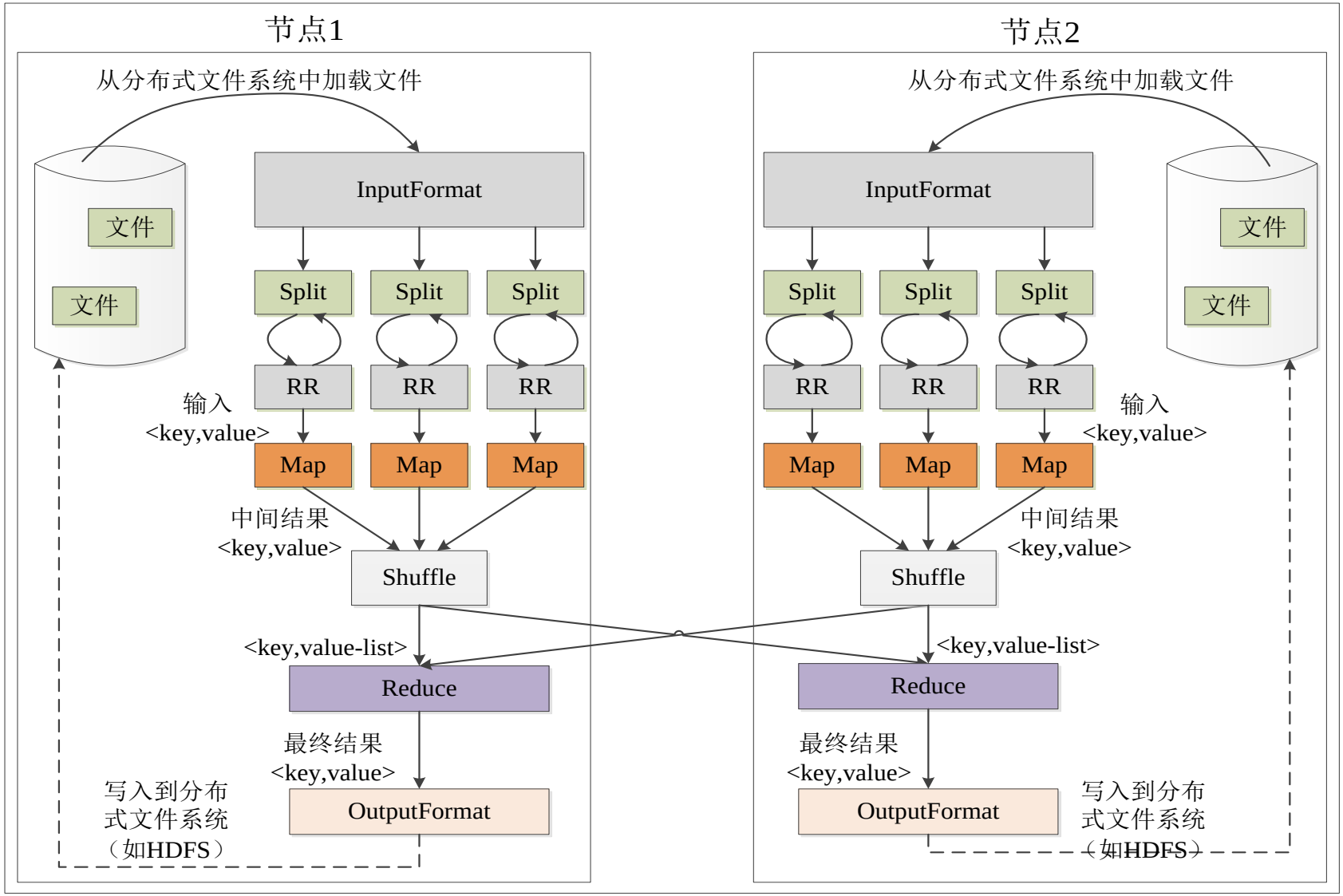
- **3）TaskTracker**
  - TaskTracker 会周期性地通过“心跳”将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）
  - TaskTracker 使用“slot”等量划分本节点上的资源量（CPU、内存等）。一个Task 获取到一个slot 后才有机会运行，而Hadoop调度器的作用就是将各个TaskTracker上的空闲slot分配给Task使用。slot 分为Map slot 和Reduce slot 两种，分别供MapTask 和Reduce Task 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

工作流程概述



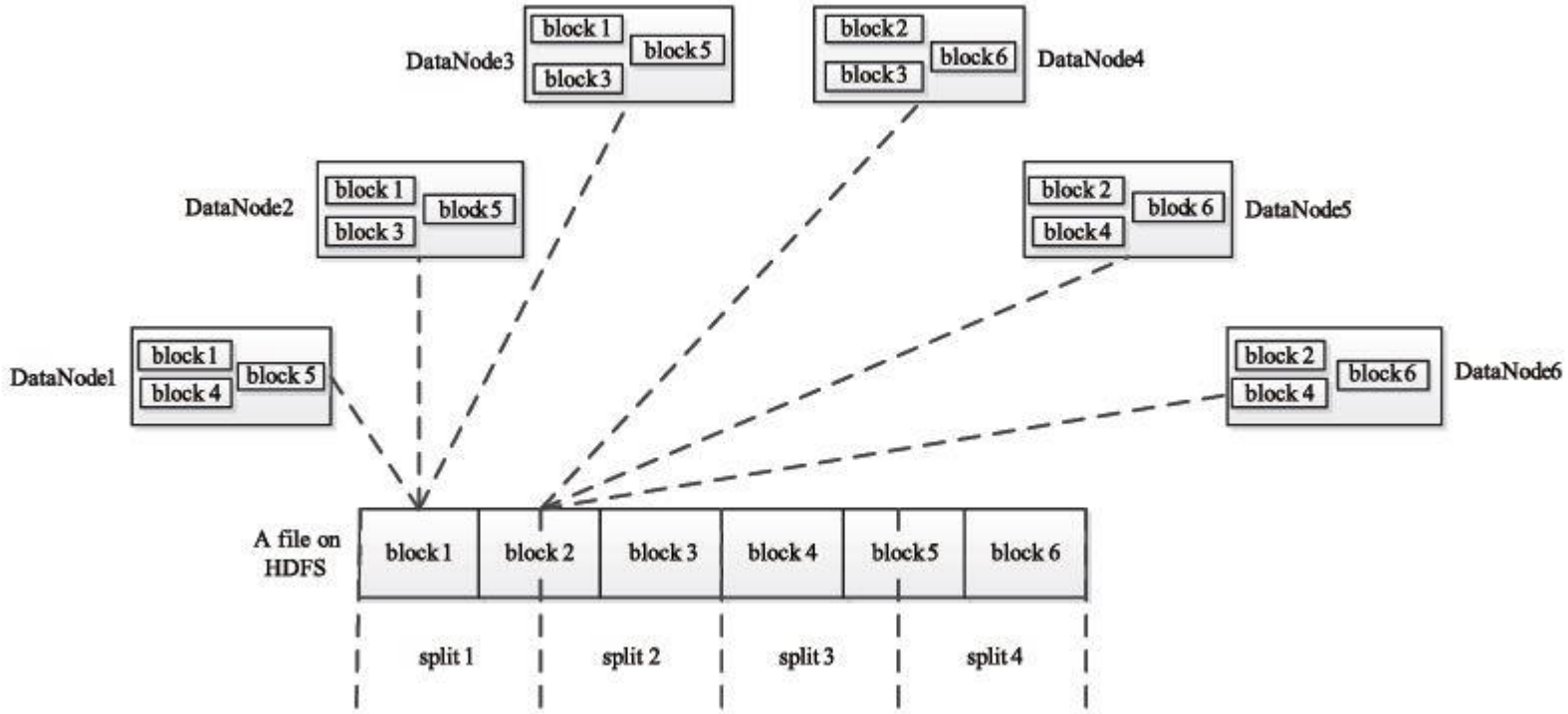
- 不同的Map任务之间不会进行通信
- 不同的Reduce任务之间也不会发生任何信息交换
- 用户不能显式地从一台机器向另一台机器发送消息
- 所有的数据交换都是通过MapReduce框架自身去实现的

MapReduce各个执行阶段



MapReduce各个执行阶段

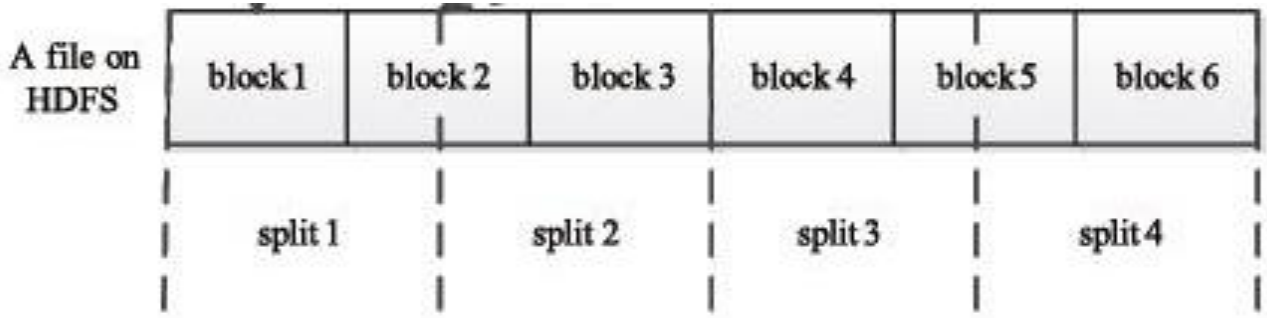
- 关于Split（分片）



- 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。

MapReduce各个执行阶段

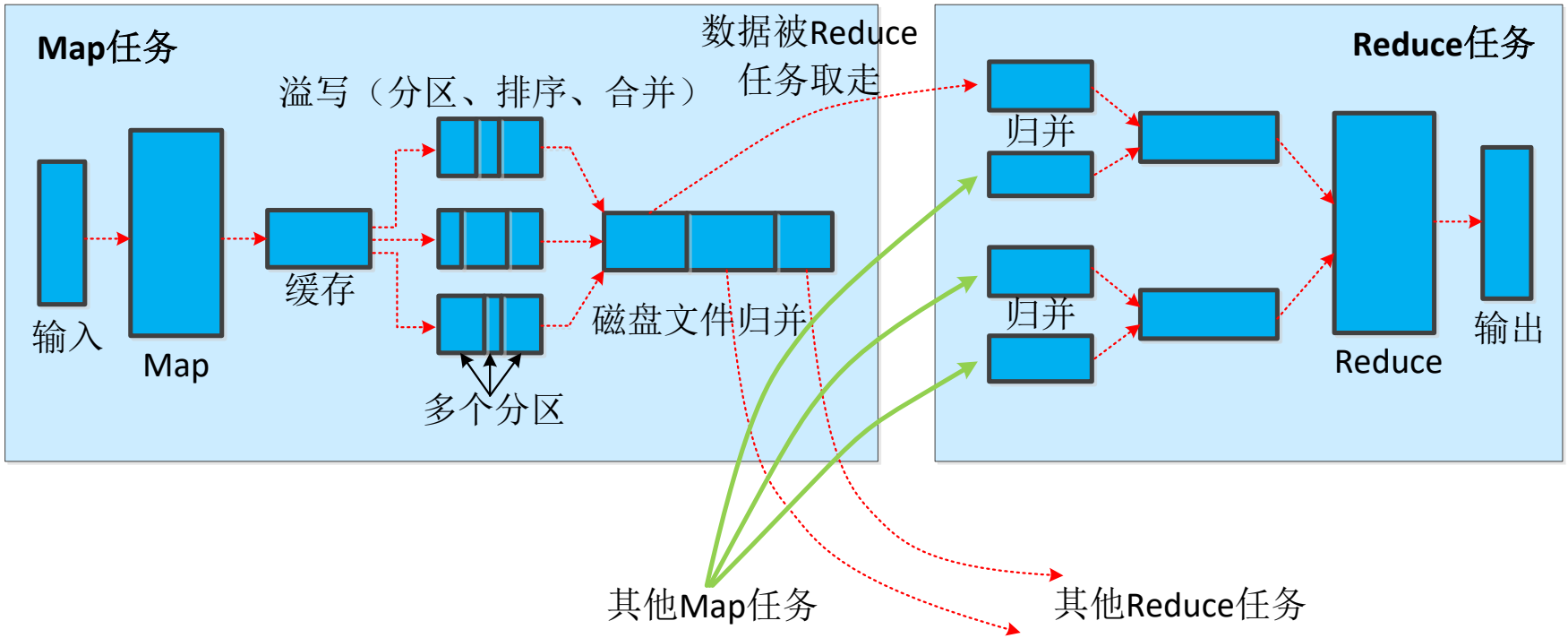
- **Map任务的数量**
  - Hadoop为每个split创建一个Map任务，split 的多少决定了Map任务的数目。大多数情况下，理想的分片大小是一个HDFS块



- **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）

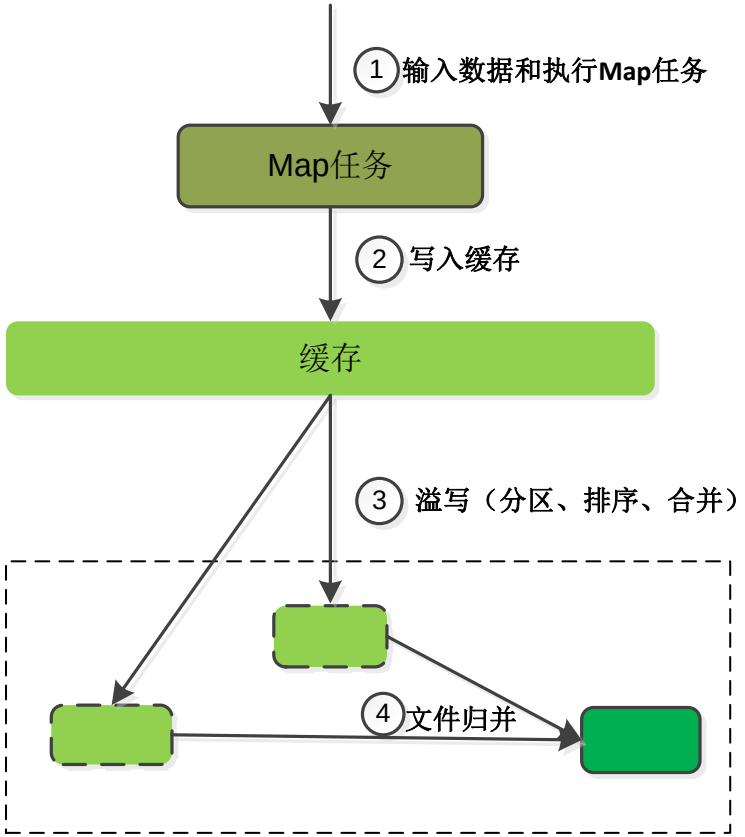
Shuffle过程详解

- 1. Shuffle过程简介



Shuffle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预定值（默认是3）则可以再次启动Combiner，少于3不需要
- JobTracker会一直监测Map任务的执行，并通知Reduce任务来领取数据

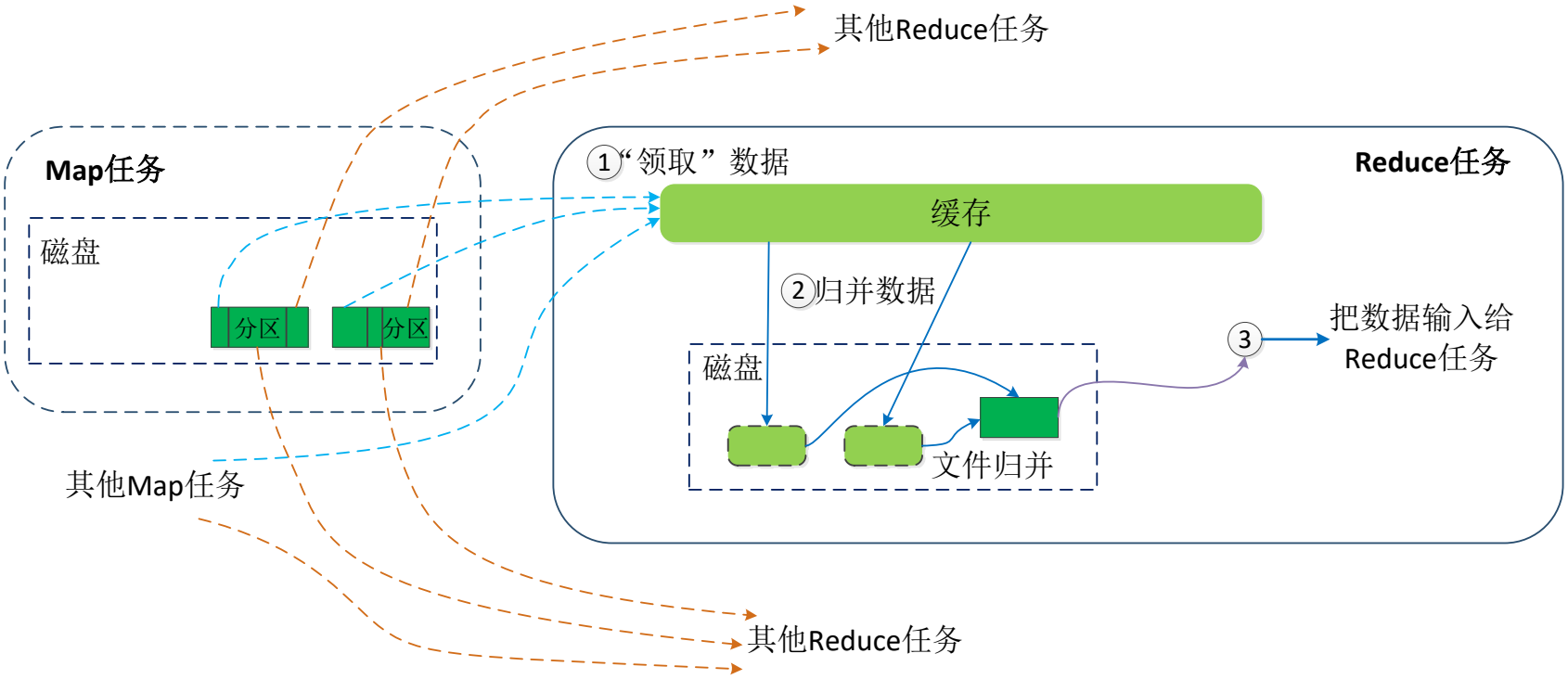
• 合并（Combine）和归并（Merge）的区别：

• 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>>

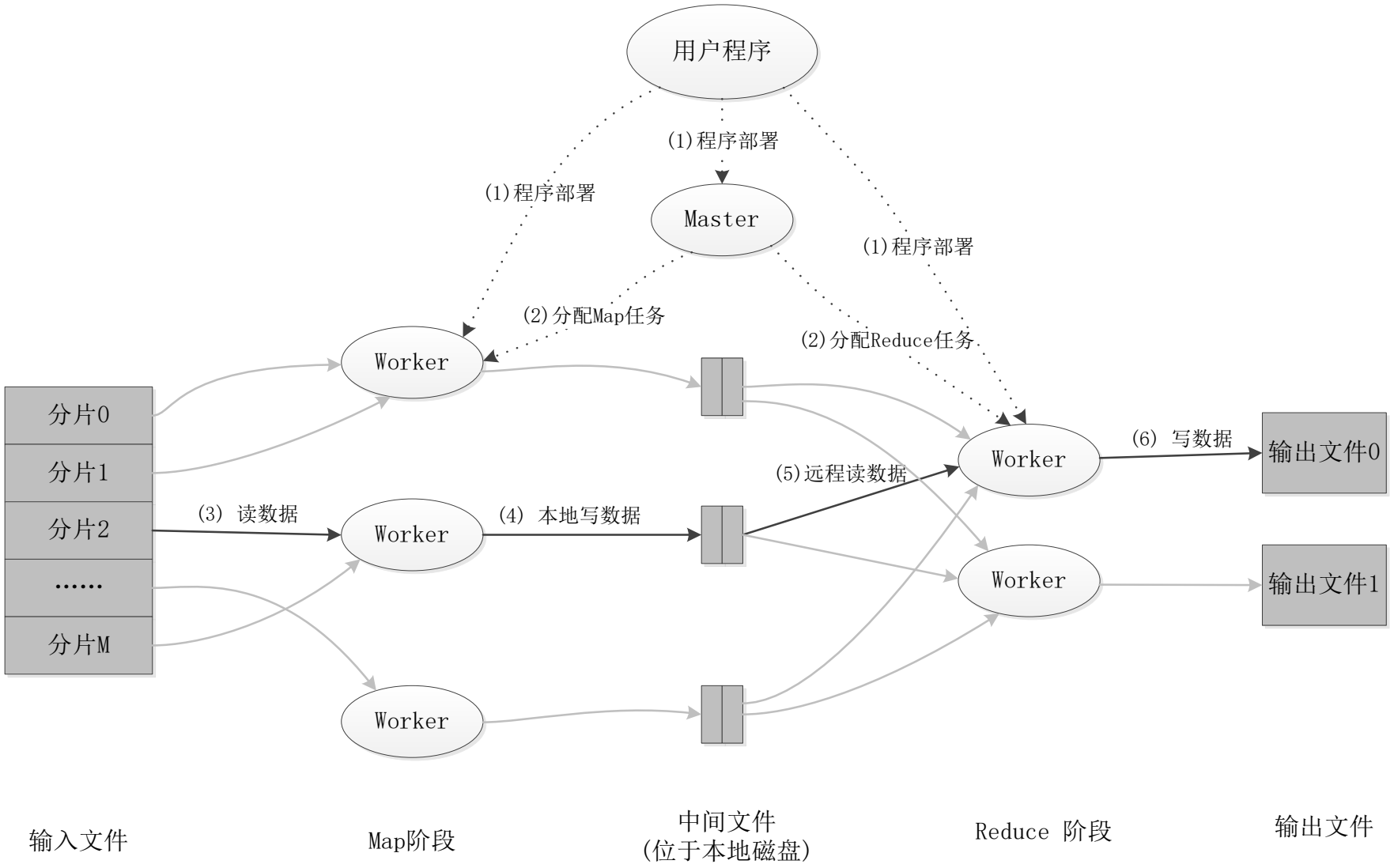
Shuffle过程详解

• 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
- 当数据很少时，不需要溢写到磁盘，直接在缓存中归并，然后输出给Reduce



MapReduce应用程序执行过程



实例：WordCount

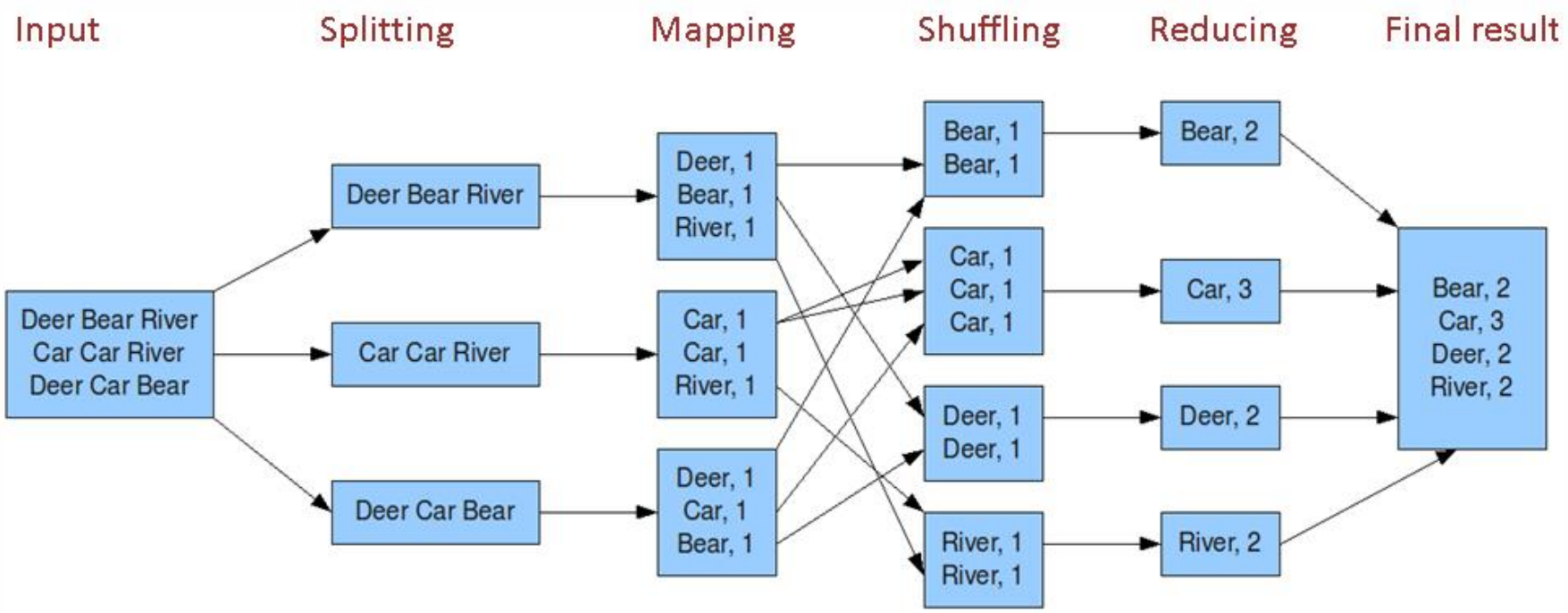
程序	WordCount
输入	一个包含大量单词的文本文件
输出	文件中每个单词及其出现次数（频数），并按照单词字母顺序排序，每个单词和其频数占一行，单词和频数之间有间隔

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

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

MapReduce设计



```
 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> [<in>...] <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

```
public static class TokenizerMapper
     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 IntSumReducer
     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);
  }
}
```

MapReduce简化模板

```
Mapper{
        setup();      // Called once at the beginning of the task.
        map();        // Called once for each key/value pair in the input split
        cleanup();    // Called once at the end of the task.
}

Reducer{
        setup();      // Called once at the beginning of the task.
        reduce();     // This method is called once for each key.
        cleanup();    // Called once at the end of the task.
}
```

例：MapReduce统计分析

- 数据源

200101	陈白露	女	1980-02-10	2001	12312341234	北京市海淀区黄庄	优秀
200102	刘云飞	男	1980-10-01	2001	13612341234	北京市西城区56号	良好
200103	张小强	男	1979-08-02	2001	19999228822	武汉市洪山区88号	良好
200104	张一夫	男	1979-08-09	2001	13511111111	郑州市金水区28号	优秀
200201	黄微	女	1981-01-03	2002	13522222222	郑州市中原区99号	良好
200202	杨勇	男	1981-08-27	2002	13433333333	郑州市二七区11号	一般
200203	朱慧娟	女	1980-12-20	2002	13344444444	武汉市江夏区45号	一般
200204	高峰	男	1981-11-08	2002	13255555555	武汉市汉江区66号	良好
200301	李菲	女	1981-05-28	2003	13666666666	北京市朝阳区15号	良好
200302	向东明	男	1981-03-02	2003	13577777777	武汉市桥口区75号	优秀

- 任务1：统计属于郑州的学生人数
- 任务2：查找并输出“杨勇”同学的手机号
- 任务3：查找所有10月出生的女生姓名和手机号
- 任务4：输出所有张姓学生在不同城市的数量分布

任务1：统计属于郑州的学生人数

```
Mapper {      map() {
        String[] toks = value.toString()
                                .trim().split("\t");
        if("郑州".equals(toks[6].substring(0,2)){
            context.write(new Text("郑州"),
                            new Text(1));
          }
     }
}
Reducer {reduce() {
        int count=0;
        for (Text value : values) {
             count++;
        }
          context.write(key, count);
    }
}
```

任务2：查找并输出“杨勇”同学的手机号

```
Mapper {map() {
        String[] toks = value.toString().trim()
                              .split("\t");
        if("杨勇".equals(toks[1])){
             context.write(new Text(toks[5]),
             new Text(1));
       }
    }
}
Reducer {reduce() {
       context.write("杨勇", key);
    }
}
```

任务3：查找所有10月出生的女生姓名和手机号

```
Mapper {map() {
        String[] toks = value.toString().trim()
                              .split("\t");
        if("10".equals(toks[3].substring(5,7)
                    && "女".equals(toks[2])){
            context.write(new Text(toks[1]),
                              new Text(toks[5]));
      }
    }
}
Reducer { reduce() {
      for (Text value : values) {
          context.write(key, value);
        }
    }
}
```

任务4：输出所有张姓学生在不同城市的数量分布

```
Mapper { map() {
        String[] toks = value.toString().trim().split("\t");
        if("张".equals(toks[1].substring(0,1)){
            context.write(
             new Text(toks[6].substring(0,3)),
              new Text(1));
          }
    }
}
Reducer { reduce() {
        int count=0;
        for (Text value : values) {
            count++;
        }
        context.write(key,count);
    }
}
```