# Hadoop分布式计算框架MapReduce

## Hadoop离线计算— 杜黎明

# 目标



- 了解Mapreduce的应用场景和由来。
- 熟悉Mapreduce的编程模型。
- 开发环境搭建，使用eclipse调试MapReduce。
- Mapreduce的典型实例介绍。

# 目录



1 为什么需要MapReduce?

2 如何对付大数据处理-分而治之

3 MapReduce简介

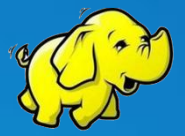
4 MapReduce示例

5 WordCountMapReduce编程模型

6 MapReduce调优

7 MapReduce案例

# 数据洪流时代



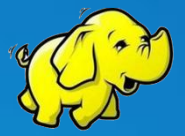
Hadoop离线计算

我们生活在这个数据爆炸的时代，很难估算全球电子设备中存储的数据总共有多少。但是据IDC估计2006年“数字全球”项目(digital universe)的数据总量为0.18 ZB，并且预测到2011年这个数字将达到1.8 ZB。

- 纽约证交所每天产生的交易数据多达1TB。
- 脸谱网存储的照片约100亿张，存储容量为1PB。
- 互联网档案馆的存储数据约为2PB，并以每月20PB的速度持续增长。
- 瑞士日内瓦附近的大型强子对撞机每年产生约15 PB的数据。

**我们有一个好消息和一个坏消息。好消息是有海量数据！坏消息是我们正在为存储和分析这些数据而奋斗不息。**

# 这么多数据我们怎么处理？



# Hadoop离线计算

例如：Intel近20年的CPU性能的发展

| 时间 | 型号 | 主频MHz | CPU/core | Flop/cycle | 性能Mflops | 性能提升倍数 |
|---|---|---|---|---|---|---|
| 1993 | Pentium | 60 | 1 | 1 | 60 | 1.0 |
| 1997 | Pentium II Xeon | 300 | 1 | 1 | 300 | 5.0 |
| 1999 | Pentium III Xeon | 500 | 1 | 1 | 500 | 8.3 |
| 2001 | Pentium 4 Xeon | 1700 | 1 | 2 | 3400 | 56.7 |
| 2005 | Pentium 4 Xeon | 3600 | 1 | 2 | 7200 | 120.0 |
| 2005 | Xeon 5000 | 3730 | 2 | 2 | 14920 | 248.7 |
| 2006 | Xeon 5100 | 3000 | 2 | 4 | 24000 | 400.0 |
| 2006 | Xeon 5300 | 2660 | 4 | 4 | 42560 | 709.3 |
| 2007 | Xeon 5400 | 3200 | 4 | 4 | 51200 | 853.3 |
| 2008 | Nehalem-EP5500 | 2930 | 4/8wSMT | 4 | 46880 | 实测5400的1.7倍 |
| 2009 | Nehalem-EX | 2270 | 8/12 | 4 | 72640 | 1210.7 |
| 2010 | Westmere-EP 5600 | 3330 | 6/12wSMT | 4 | 79920 | 1332.0 |
| 2010 | Nehalem-EX 7500 | 2270 | 8/16wSMT | 4 | 72640 | 1210.7 |
| 2011 | Sandy Bridge | 2930 | 8/16wSMT | 8 | 187520 | 3125.3 |
| 2011 | Westmere-EX | 2400 | 10/20wSMT | 4 | 96000 | 1600.0 |

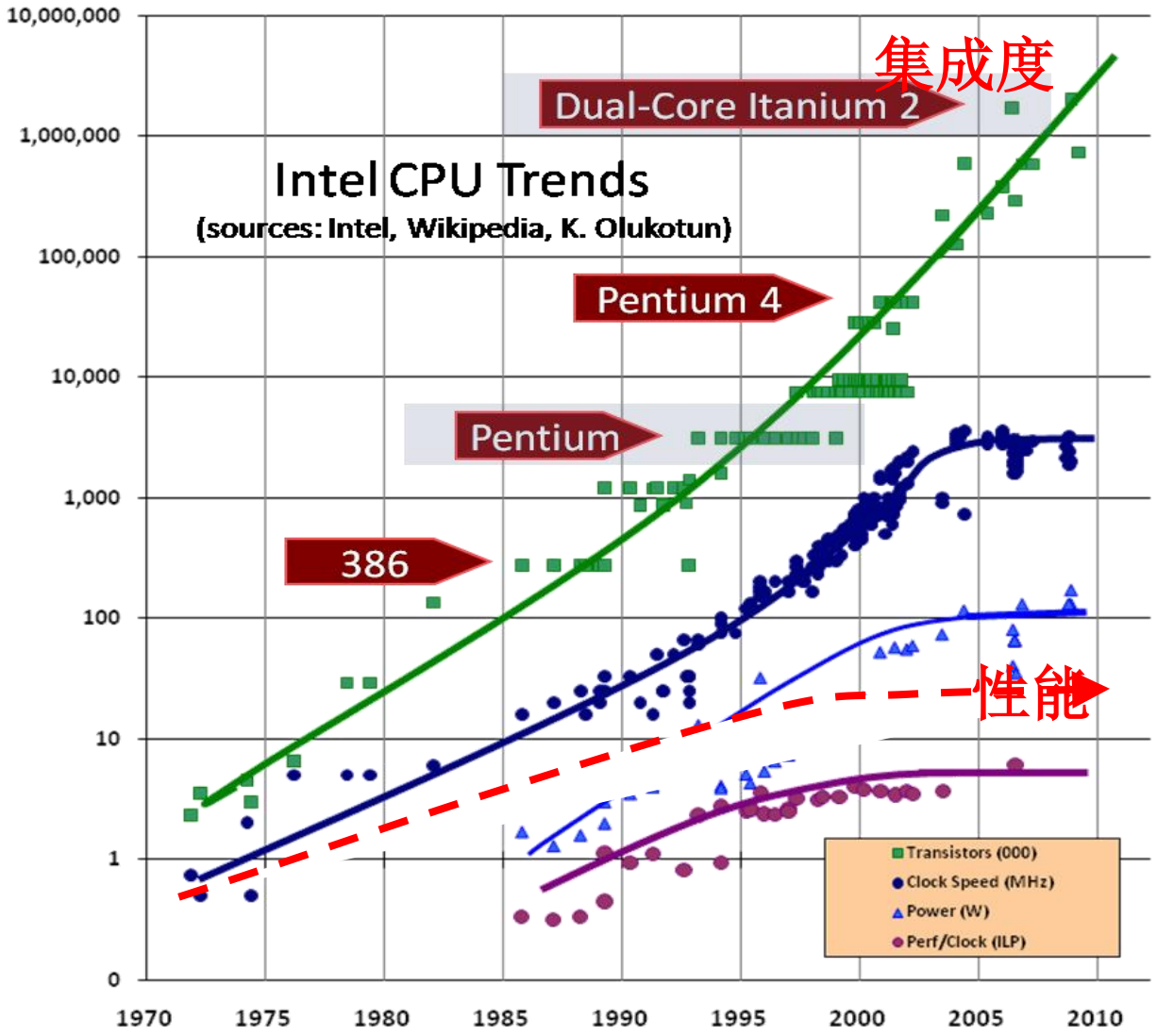
通过芯片制程工艺+处理器微架构设计+服务器平台技术，使CPU性能大幅提升

Intel
微处理器
每秒
1千8百亿次
浮点运算！

近20年性能
提高3千多倍

# 这么多数据我们怎么处理？

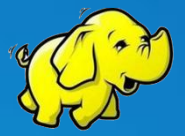
# Hadoop离线计算




所有这些技术极大地提高了微处理器的计算性能，但2004后处理器的性能不再像人们预期的那样提高。

单核处理器性能
提升接近极限！

# 这么多数据我们怎么处理？

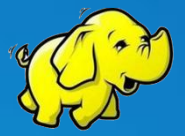


Hadoop离线计算

在古时候，人们用牛来拉重物。当一头牛拉不动一根圆木时，人们从来没有考虑过要培育更强壮的牛。同理，我们也不该想方设法的打造超级计算机，而应该千方百计综合利用更多计算机来解决问题。

——格蕾丝·霍博(Grace Hopper)

**解决方案：并行计算。**





## MapReduce是目前面向海量数据处理最为成功的技术

MapReduce是目前业界和学界公认的最为有效和最易于使用的海量数据并行处理技术，之前尚无其它更有效的技术

Google, Yahoo, IBM, Amazon, 百度等国内外公司普遍使用

Google:超过7千个程序基于MapReduce开发！

## 问题与需求：

如何对巨量的Web文档建立索引、根据网页链接计算网页排名，从上百万文档中训练垃圾邮件过滤器，运行气象模拟，数十亿字符串的排序？

## 解决方案：

如果你想学习如果编写程序完成这些巨量数据的处理问题，MapReduce将为你提供一个强大的分布式计算环境和构架，让你仅需关注你的应用问题本身，编写很少的程序代码即可完成看似难以完成的任务！

## 医疗行业

- Seton Healthcare是采用IBM最新沃森技术医疗保健内容分析预测的首个客户。该技术允许企业找到大量病人相关的临床医疗信息，通过大数据处理，更好地分析病人的信息。
- 在加拿大多伦多的一家医院，针对早产婴儿，每秒钟有超过3000次的数据读取。通过这些数据分析，医院能够提前知道哪些早产儿出现问题并且有针对性地采取措施，避免早产婴儿夭折。
- 通过社交网络来收集数据的健康类App。也许未来数年后，它们搜集的数据能让医生给你的诊断变得更为精确，比方说不是通用的成人每日三次一次一片，而是检测到你的血液中药剂已经代谢完成会自动提醒你再次服药。
- Express s是一家处方药管理服务公司，目前它正在通过一些复杂模型来**检测虚假药品**，这些模型还能及时提醒人们何时应该停止用药。Express s能够解决该问题的原因在于所有有关数据。因为它每年管理着1.4亿处方，覆盖了一亿美国人和65,000家药店。他们还着眼于一些事情，如所开处方的药物种类，甚至如果一个医生的行为被标记为红色的旗帜，那么他在网络上是个好人的形象，更是你所需要的医生。

## 能源行业

- 故障预警：伴随风电行业的快速发展，风电机组的运行故障问题日益突出。由于风电机组安装在复杂环境下，使得风电机组故障频发。基于风电场、集控指挥中心的长期数据做深度学习，可以有效地预测事故隐患并实现快速准确的系统维护，减少部件损坏带来经济损失，提高企业的经济效益。
- 机组性能分析：通过性能损失与停机损失量化的、直观的分析风电场对应风资源下的发电效能，同时提供方便的为发现问题而设置的功率曲线分析、损失电量分解、故障分析等工具。可以用来评估各省、各机型的性能及故障率水平，从而实现运维能力评估、机型评估，可用于前期规划、机型选购、运维能力考核等。
- 智能电网现在欧洲已经做到了终端，也就是所谓的智能电表。在德国，为了鼓励利用太阳能，会在家庭安装太阳能，除了卖电给你，当你的太阳能有多余电的时候还可以买回来。通过电网收集每隔五分钟或十分钟收集一次数据，收集来的这些数据可以用来预测客户的用电习惯等，从而推断出在未来2~3个月时间里，整个电网大概需要多少电。有了这个预测后，就可以向发电或者供电企业购买一定数量的电。**因为电有点像期货一样，如果提前买就会比较便宜，买现货就比较贵。**通过这个预测后，可以降低采购成本。

一种面向大规模海量数据处理的高性能并行计算平台和软件编程框架。

源自于Google的Mapreduce论文

- 发表于2014年2月。
- Hadoop Mpareduce是Google Mapreduce的克隆。

Mapreduce的特点

- 良好的扩展性。
- 高容错性。
- 适合PB级及海量数据的离线处理。

## ◆ 基于集群的高性能并行计算平台(Cluster Infrastructure)

允许用市场上现成的普通PC或性能较高的刀架或机架式服务器，构成一个包含数千个节点的分布式并行计算集群。

## ◆ 并行程序开发与运行框架(Software Framework)

提供了一个庞大但设计精良的并行计算软件构架，能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行子任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算中的很多复杂细节交由系统负责处理，大大减少了软件开发人员的负担。

## ◆ 并行程序设计模型与方法(Programming Model & Methodology)

借助于函数式语言中的设计思想，提供了一种简便的并行程序设计方法，用Map和Reduce两个函数编程实现基本的并行计算任务，提供了完整的并行编程接口，完成大规模数据处理

## 大规模数据处理时，MapReduce在三个层面上的基本构思。

### 如何对付大数据处理：分而治之

对相互间不具有计算依赖关系的大数据，实现并行最自然的办法就是采取分而治之的策略。

### 上升到抽象模型：Mapper与Reducer

MPI等并行计算方法缺少高层并行编程模型，为了克服这一缺陷，MapReduce借鉴了Lisp函数式语言中的思想，用Map和Reduce两个函数提供了高层的并行编程抽象模型。

### 上升到构架：统一构架，为程序员隐藏系统层细节

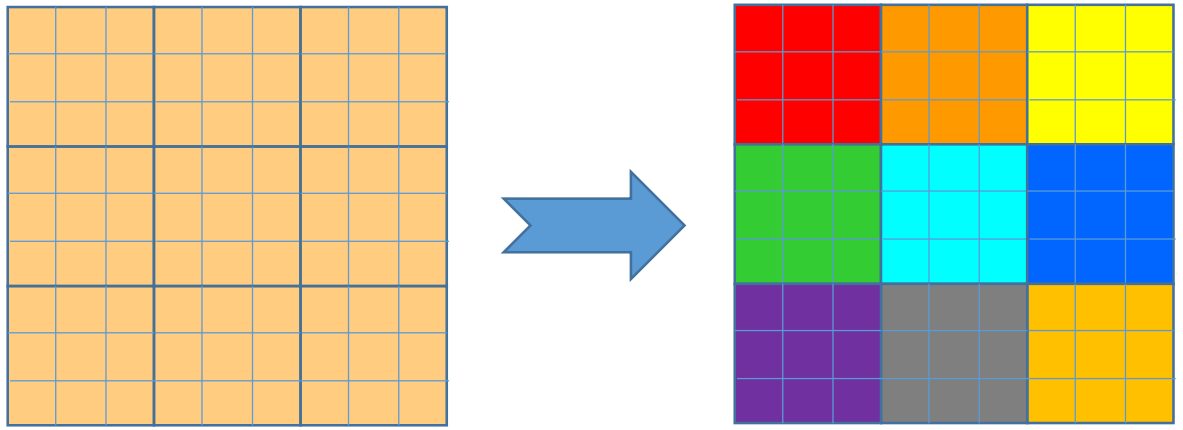
MPI等并行计算方法缺少统一的计算框架支持，程序员需要考虑数据存储、划分、分发、结果收集、错误恢复等诸多细节；为此，MapReduce设计并提供了统一的计算框架，为程序员隐藏了绝大多数系统层面的处理细节。

# 如何对付大数据处理-分而治之



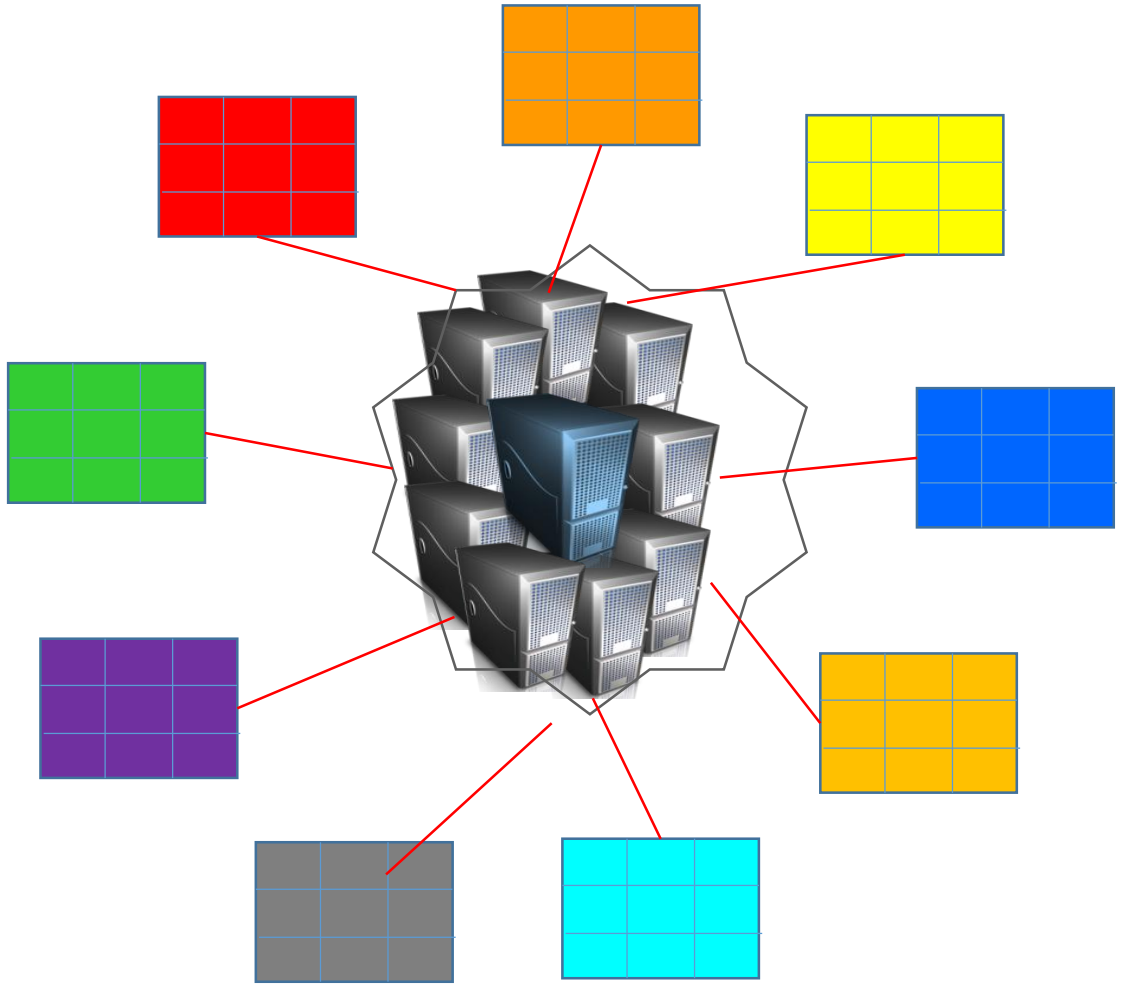
一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是并行计算。

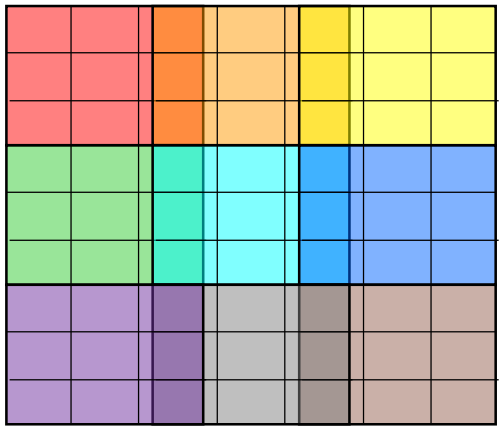
例如：假设有一个巨大的2维数据需要处理(比如求每个元素的开立方)，其中对每个元素的处理是相同的，并且数据元素间不存在数据依赖关系，可以考虑不同的划分方法将其划分为子数组，由一组处理器并行处理。



# 如何对付大数据处理-分而治之



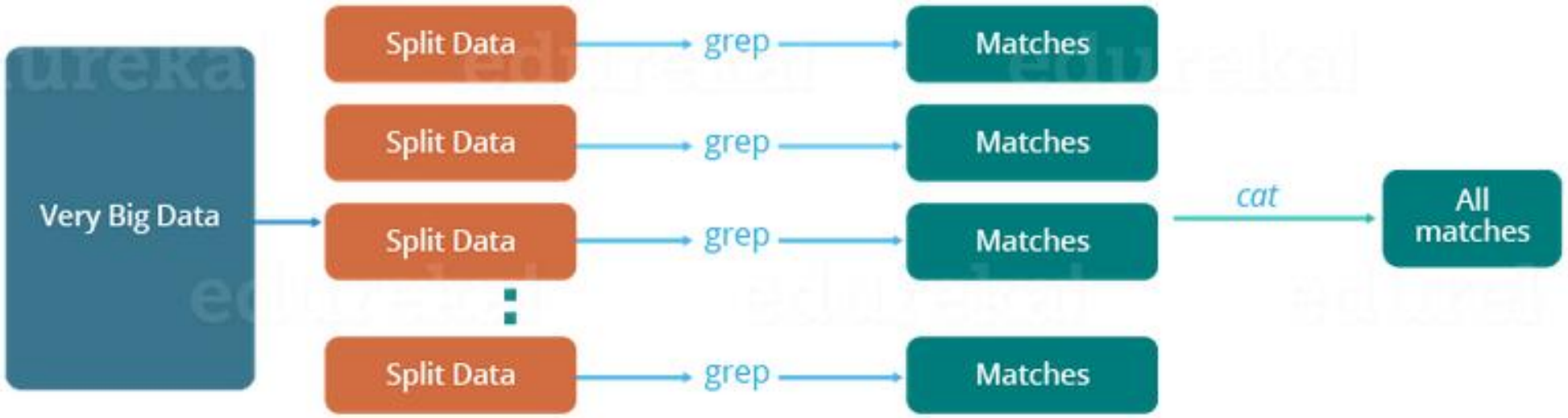


Master：负责划分和分配任务

Workder：负责数据块计算

# 上升到构架：统一构架，为程序员隐藏系统层细节 Hadoop离线计算




假设Mapreduce框架还没有出现，那么我们要如何并行的处理一块很大的数据呢？
假设现在有2000年到2015年的每日的天气数据，每日天气数据记录了一天的平均气温。那么要如何计算出2000到2017年的最高气温呢？

如果用传统的方式，分一下几步：

- 将会把天气数据首先分为多块存储到不同的机器上。
- 计算出每块数据中的最高温度。
- 汇总各台机器上的所有数据的最高气温。
- 汇总所有机器上最高气温。

# 上升到构架：统一构架，为程序员隐藏系统层细节 Hadoop离线计算

## 问题：

- 如何管理和存储数据？如何划分数据？
- 如何调度计算任务？
- 如果节点间需要共享或交换数据怎么办？
- 如何考虑数据通信和同步？
- 如何掌控节点的执行完成情况？如何收集中间和最终的结果数据？
- 节点失效如何处理？如何恢复数据？如何恢复计算任务？
- 节点扩充后如何保证原有程序仍能正常运行并保证系统性能提升？

我们能把这些细节和复杂性交给系统去负责处理吗？

# 上升到构架：统一构架，为程序员隐藏系统层细节

## MapReduce提供一个统一的计算框架，可完成：

- 计算任务的划分和调度。
- 数据的分布存储和划分。
- 处理数据与计算任务的同步。
- 结果数据的收集整理(sorting, combining, partitioning,...)。
- 系统通信、负载平衡、计算性能优化处理。
- 处理系统节点出错检测和失效恢复。

# 什么样的任务可以进行并行计算？



并行计算的第一个重要问题是如何划分计算任务或者计算数据以便对划分的子任务或数据块同时进行计算。但一些计算问题恰恰无法进行这样的划分！

**Nine women cannot have a baby in one month!**

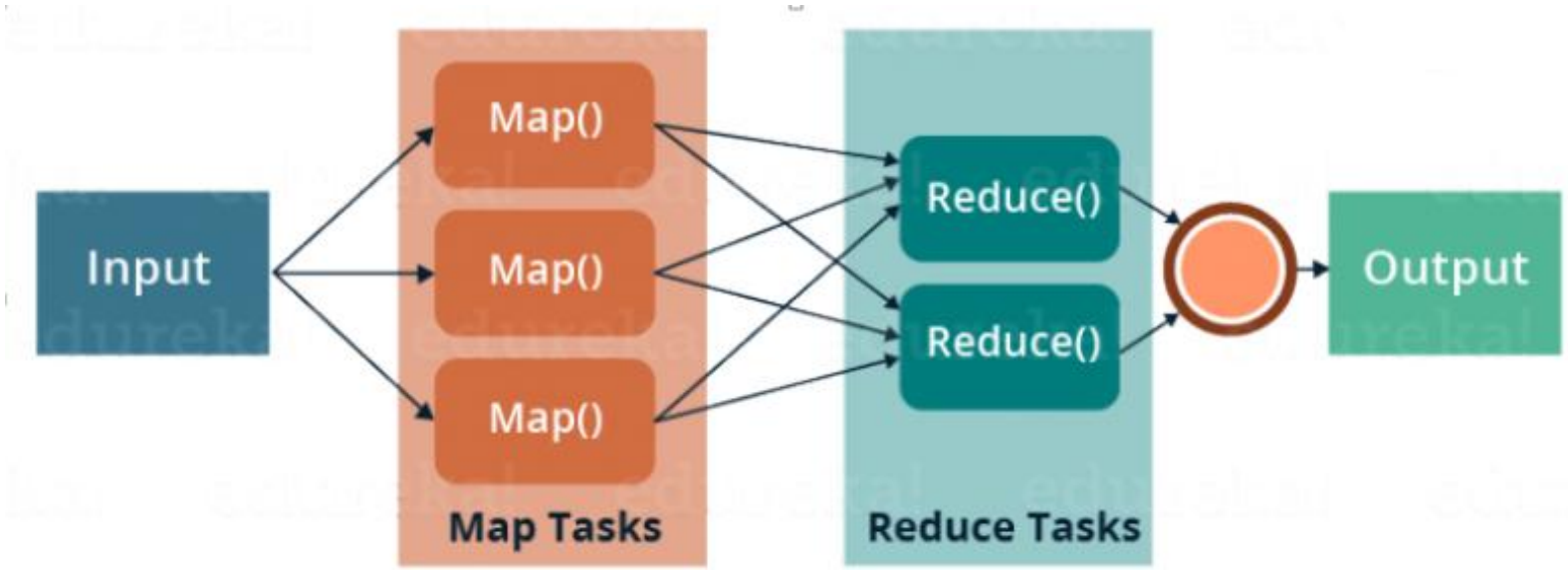
例如：Fibonacci函数： Fk+2 = Fk + Fk+1

前后数据项之间存在很强的依赖关系！只能串行计算！

**结论：不可分拆的计算任务或相互间有依赖关系的数据无法进行并行计算！**

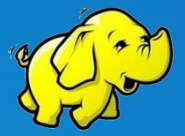
# MapReduce计算模型






- 主要由Map和Reduce两部分组成。
- Reduce阶段在Map阶段执行结束之后执行。
- Map阶段的输出结果作为Reduce阶段的输入结果。
- Reduce阶段的输入结果对应于多个Map的输出结果。
- Reduce阶段计算最终结果并将结果输出。

# MapReduce示例-文档词频统计WordCount



# Hadoop离线计算

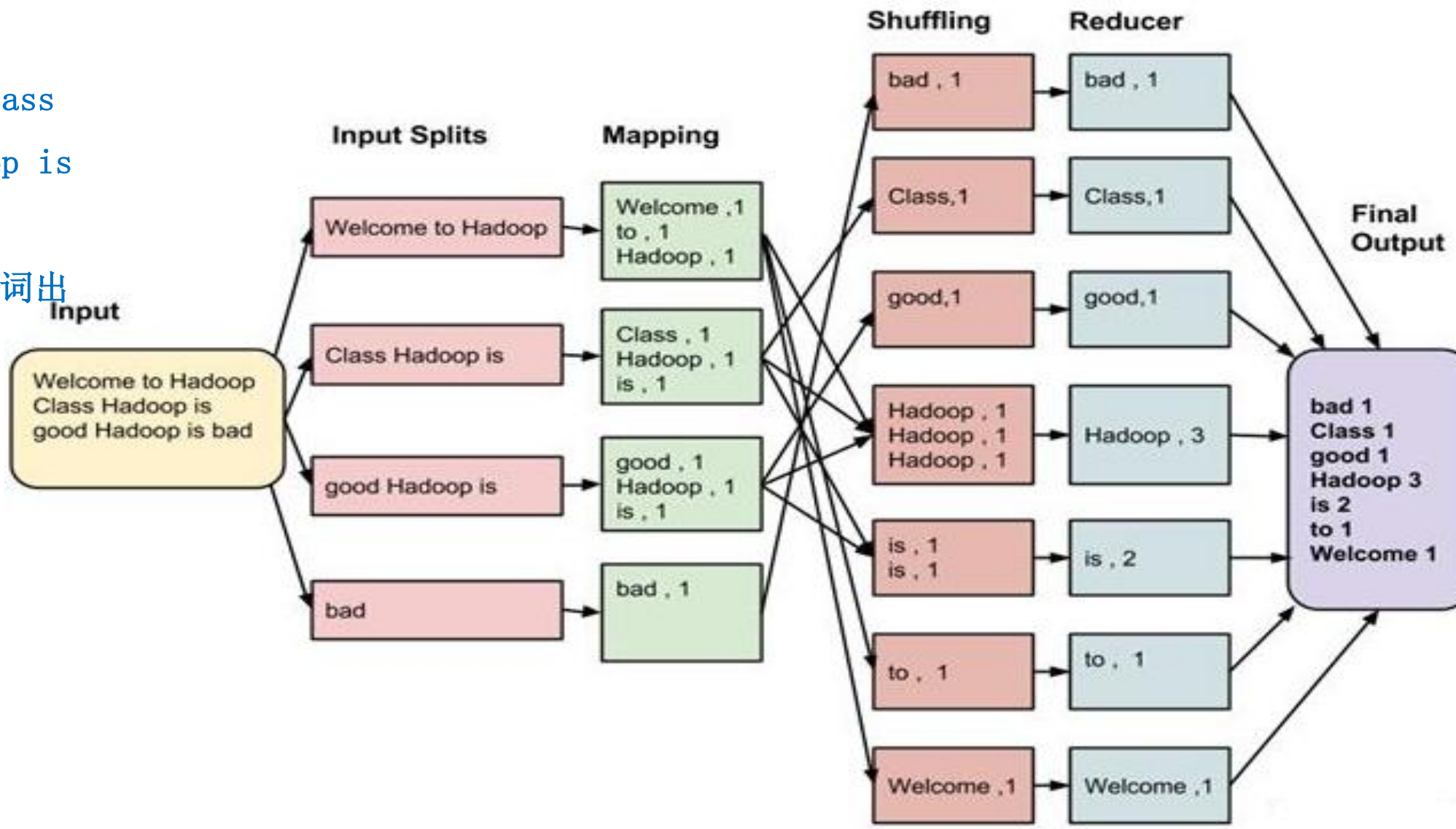
假设有一组文字：

welcome to Hadoop class

Hadoop is good Hadoop is

bad

统计这组文字中每个单词出现的次数。



# MapReduce编程模型-wordCount

- 输入数据：一系列key/value对。
- 用户实现两个函数，map和reduce。
- ✓ Map(k,v)→list(k1,v1)
- ✓ Reduce(k,list(v1)) →v2
- (k1,v1)是中间key/value的结果对。
- Output:一系列的（k2,v2）对。

# MapReduce编程模型-wordCount



```java
public static class Map extends Mapper<Object,Text,Text,IntWritable>{
    //one表示单词出现一次
    private static IntWritable one = new IntWritable(1);
    //word存储切下的单词
    private Text word = new Text();
    public void map(Object key,Text value,Context context)
            throws IOException,InterruptedException{
        //对输入的行切词
        StringTokenizer st = new StringTokenizer(value.toString());
        while(st.hasMoreTokens()){
            word.set(st.nextToken());//切下的单词存入word
            context.write(word, one);
        }
    }
}
```

```
<terminated> WordCoutJava [Java Application] C:\Java\jdk1.7
to:1
is:2
Hadoop:3
class:1
good:1
bad:1
welcome:1
```

```java
public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable>{
    //result记录单词的频数
    private static IntWritable result = new IntWritable();
    public void reduce(Text key,Iterable<IntWritable> values,Context context)
            throws IOException,InterruptedException{
        int sum = 0;
        //对获取的<key,value-list>计算value的和
        for(IntWritable val:values){
            sum += val.get();
        }
        //将频数设置到result
        result.set(sum);
        //收集结果
        context.write(key, result);
    }
}
```

# 序列化&反序列化

- 序列化是指将结构化的数据转化为字节流以便在网络上传输或写入到磁盘进行永久存储的过程，反序列化是指将字节流转换为结构化对象的逆过程。
- 序列化常见应用场景：进程间通信和永久存储。
- Hadoop中，序列化要满足：
  - ✓ 紧凑
  - ✓ 快速
  - ✓ 可扩展
  - ✓ 支持互相操作

Hadoop中使用了自己的序列化格式Writable。它绝对紧凑、速度快、但不容易扩展。

# MapReduce常用数据类型

*Table 4-7. Writable wrapper classes for Java primitives*

| Java primitive | Writable implementation | Serialized size (bytes) |
|---|---|---|
| boolean | BooleanWritable | 1 |
| byte | ByteWritable | 1 |
| short | ShortWritable | 2 |
| int | IntWritable | 4 |
| | VIntWritable | 1–5 |
| float | FloatWritable | 4 |
| long | LongWritable | 8 |
| | VLongWritable | 1–9 |
| double | DoubleWritable | 8 |



这些数据类型都实现了WritableComparable接口，以便进行网络传输和文件存储，以及进行大小比较

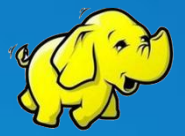
```
IntWritable iw = new IntWritable(1);
System.out.println(  iw.get() );  // 1

BooleanWritable bw = new BooleanWritable(true);
System.out.println(  bw.get() );  // true
```

## 自定义数据类型

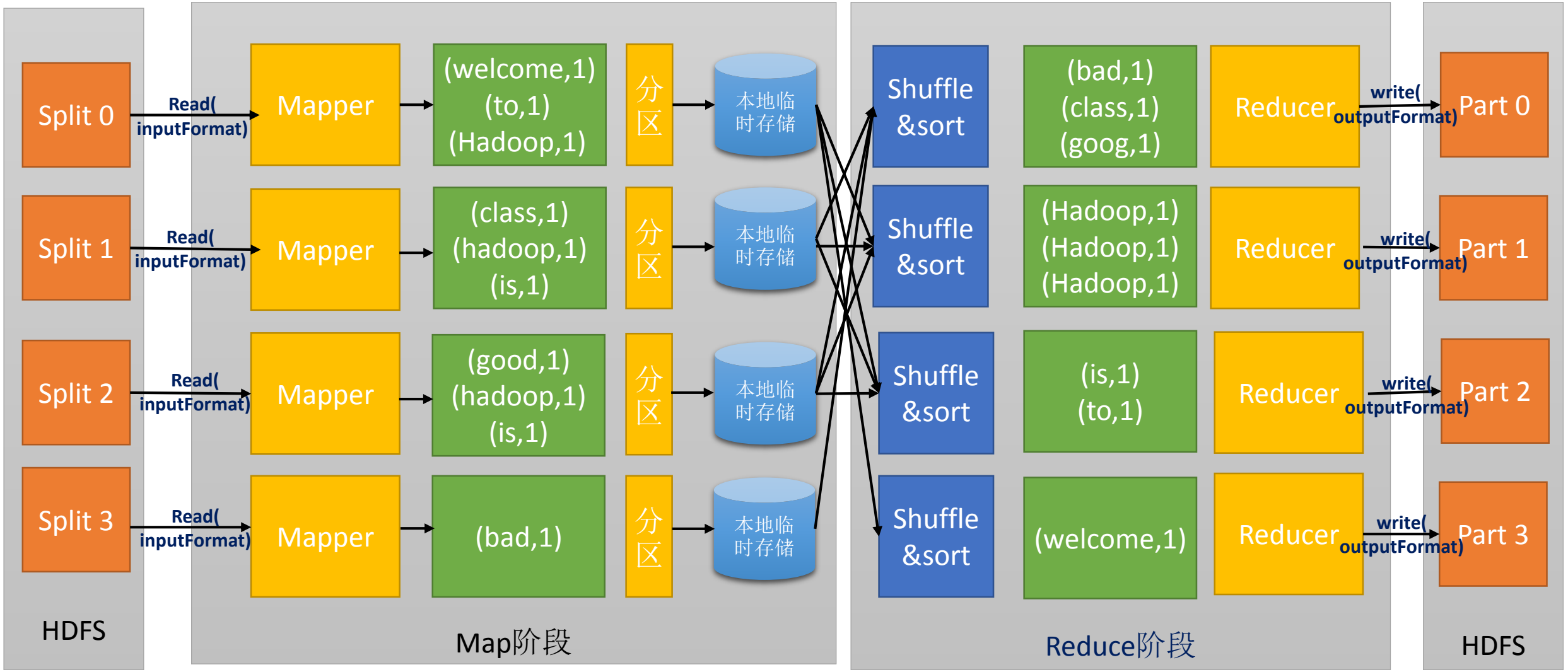
- 实现Writable接口，以便该数据能被序列化后完成网络传输或文件输入/输出。
- 如果该数据需要作为主键key使用，或需要比较数值大小时，则需要实现WritableComparable接口。

- MapReduce将作业分成两个阶段：**Map阶段**和**Reduce阶段**。
- Map阶段由一定数量的MapTask组成。
  - ✓ 输入数据格式解析：**InputFormat**
  - ✓ 输入数据处理：**Mapper**
  - ✓ 数据分组：**Partitioner**
- Reduce阶段由一定数量的ReduceTask组成。
  - ✓ 数据远程拷贝
  - ✓ 数据按照key进行排序。
  - ✓ 数据处理：**Reducer**
  - ✓ 数据输出格式：**OutputFormat**

# MapReduce编程模型-wordCount

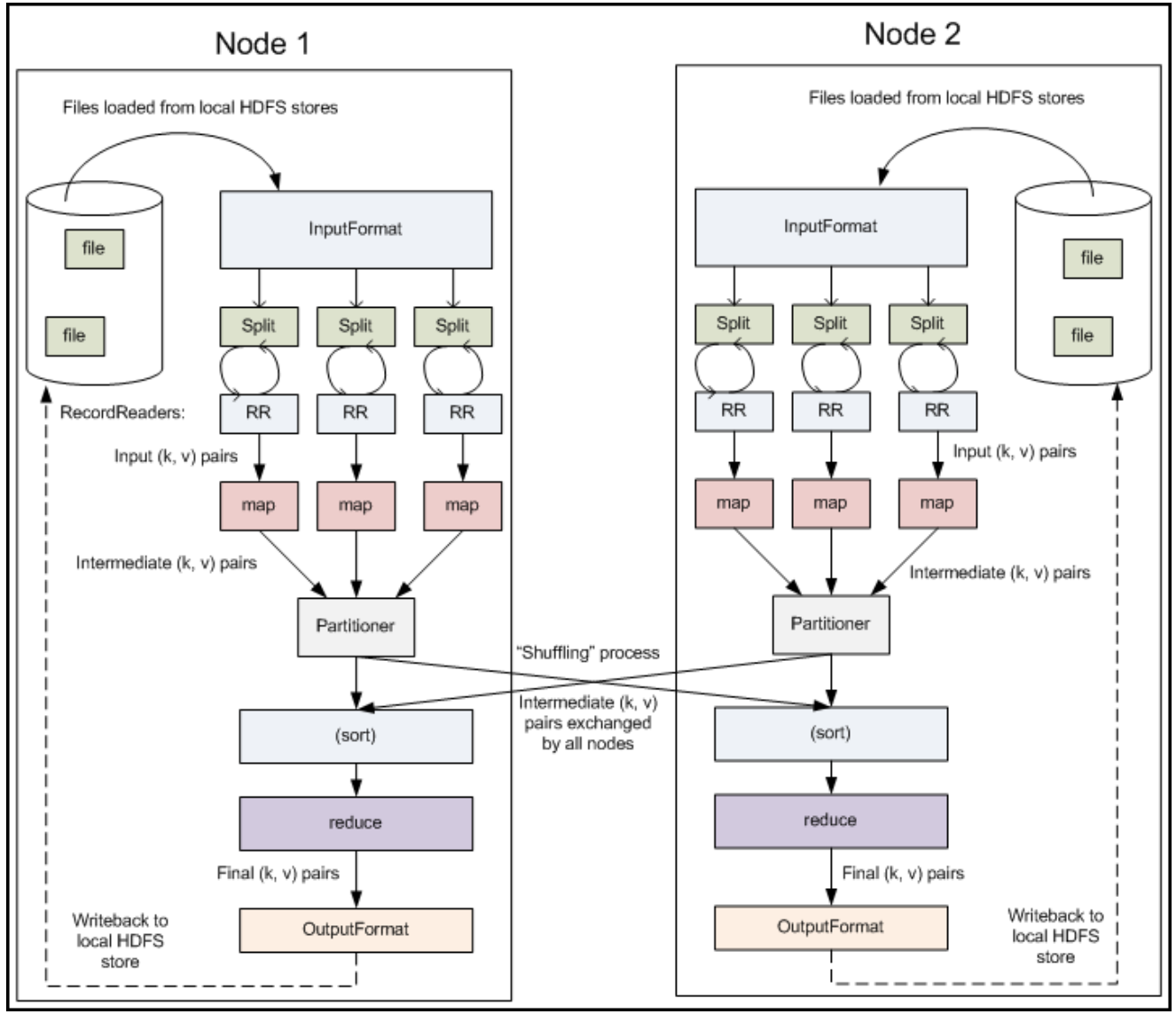


Hadoop离线计算



# MapReduce数据流






**数据分片：**等长的数据块，每个分片对应一个map任务。分片的大小很大程度影响了map执行的效率。建议分片大小和hdfs数据块大小相同。“输入分片”并非数据本身，而是逻辑上的分片长度和数据位置数组。

**Map任务结果并非保存在hdfs：**临时数据。Map失败后会在其他节点上启动map。

**Reduce的任务数量并非由输入数据大小指定，而事实上是独立指定的。Map:reduce=n:m**

**Partition:**map输出结果会根据reduce个数进行分区，默认是hash函数来分区，也可以自定义(partition函数)。

**Shuffle:**map和reduce之间的数据流。每个reduce的输入来自于多个map任务的输出。在这个阶段，参数调优对job的运行时间有很大的影响。**是奇迹发生的地方。**

# MapReduce编程模型-inputFormat



InputFormat接口定义的方法就是如何读取文件和分割文件以提供分片给mapper。

TextInputFormat是InputFormat的默认实现类。

通过使用InputFormat，MapReduce可以做到：

- ✓验证作业的输入的正确性
- ✓把输入文件切分成多个逻辑块（InputSplit），并把它们分发给一个单独的MapperTask。
- ✓实例化一个能再每个InputSplit类上工作的RecordReader的实现，这个RecordReader从指定的InputSplit中正确读出一条一条的K－V对，这些K－V对将由我们写的Mapper方法处理。

# MapReduce编程模型-split和block

- Block（块）
  - ✓ HDFS中存储数据的最小单元。
  - ✓ 默认大小是128MB。
- Split（分片）
  - ✓ Mapreduce中最小计算单元。
  - ✓ 分片并非数据本身，而是记录一个分片的长度和分片数据位置信息的数组。
  - ✓ 每个Split对应一个MapTask。
  - ✓ 默认split数量和Block一一对应。
  - ✓ Block和split的关系是任意的，可由用户控制。

# MapReduce编程模型-split和block

- **文件大小<=块容量(128M)：** 每个文件对应一个InputSplit。
- **文件大小>块容量(存在多个块时)：** 使用更为复杂的公式来计算InputSplit大小。

InputSplitSize = Max(minSplitSize, Min(blockSize, maxSplitSize))

minSplitSize = mapreduce.input.fileinputformat.split.minsize

blockSize = dfs.blocksize

maxSplitSize=mapreduce.input.fileinputformat.split.maxsize

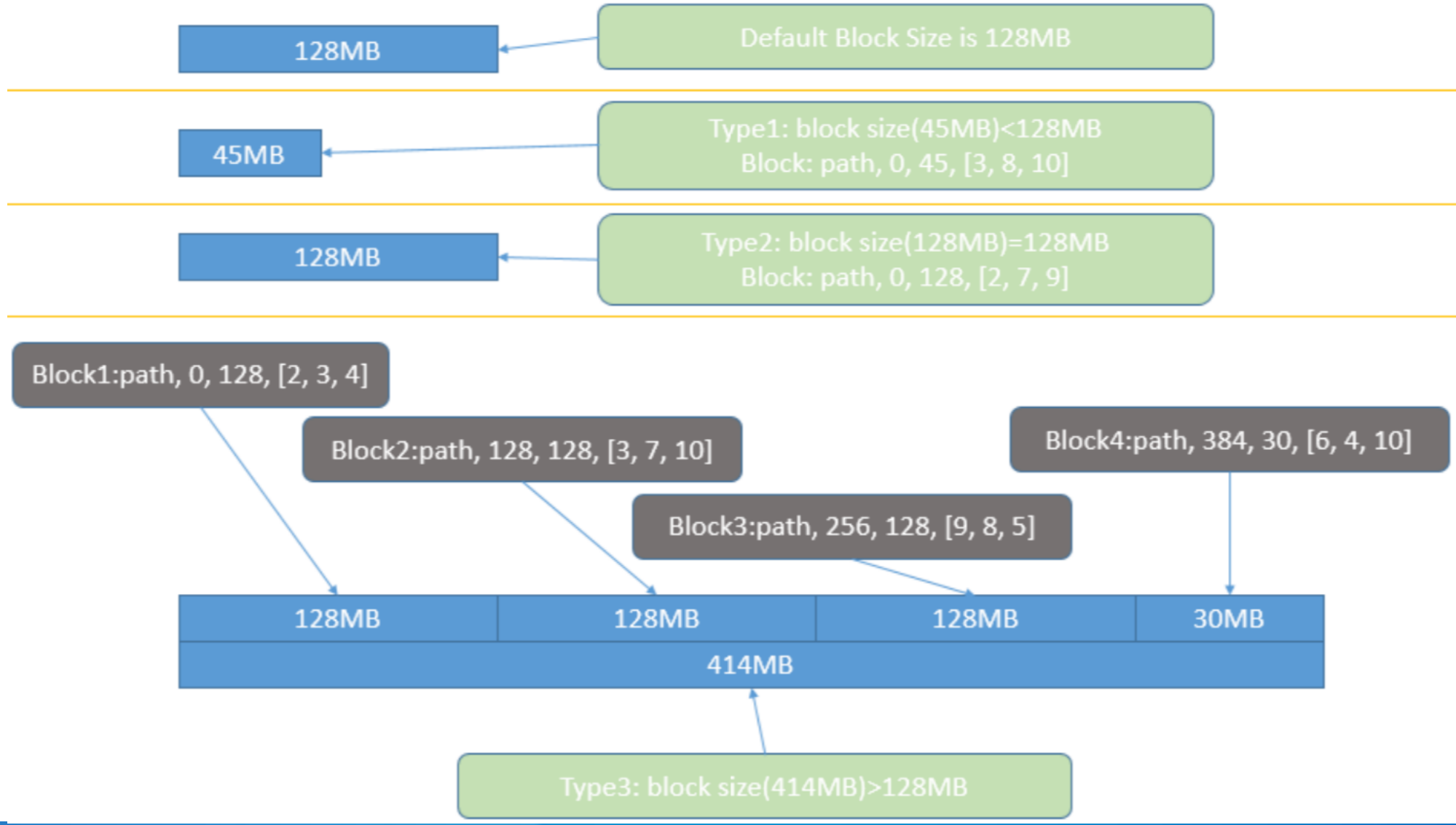
minSplitSize默认值=1； maxSplitSize默认值=Long.MAX_VALUE;

- **分片数量=文件大小/分片大小(splitNum = totalsize / inputSplitSize)**
- **得出以下结论：**

如果想增加map数量： 需要调小maxSplitSize的值。

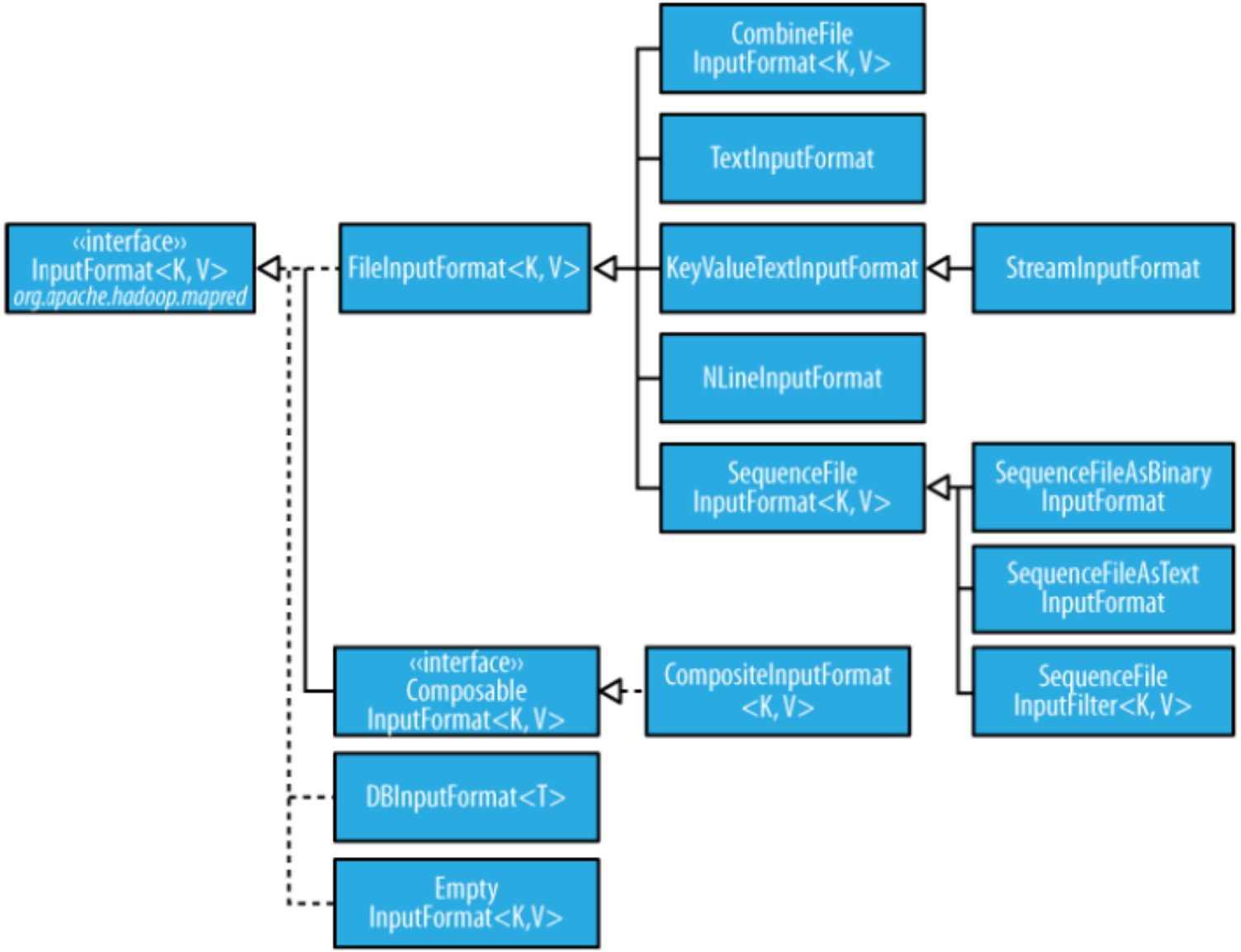
如果想减少map数量： 需要调大minSplitSize的值。

# MapReduce编程模型-split和block



# MapReduce编程模型-inputFormat





# MapReduce编程模型-partitioner

- Partitioner决定Map的输出结果交给哪个Reducer Task来处理。
- 默认实现： hash partitioner

Key的hash值%reduce Task数量。

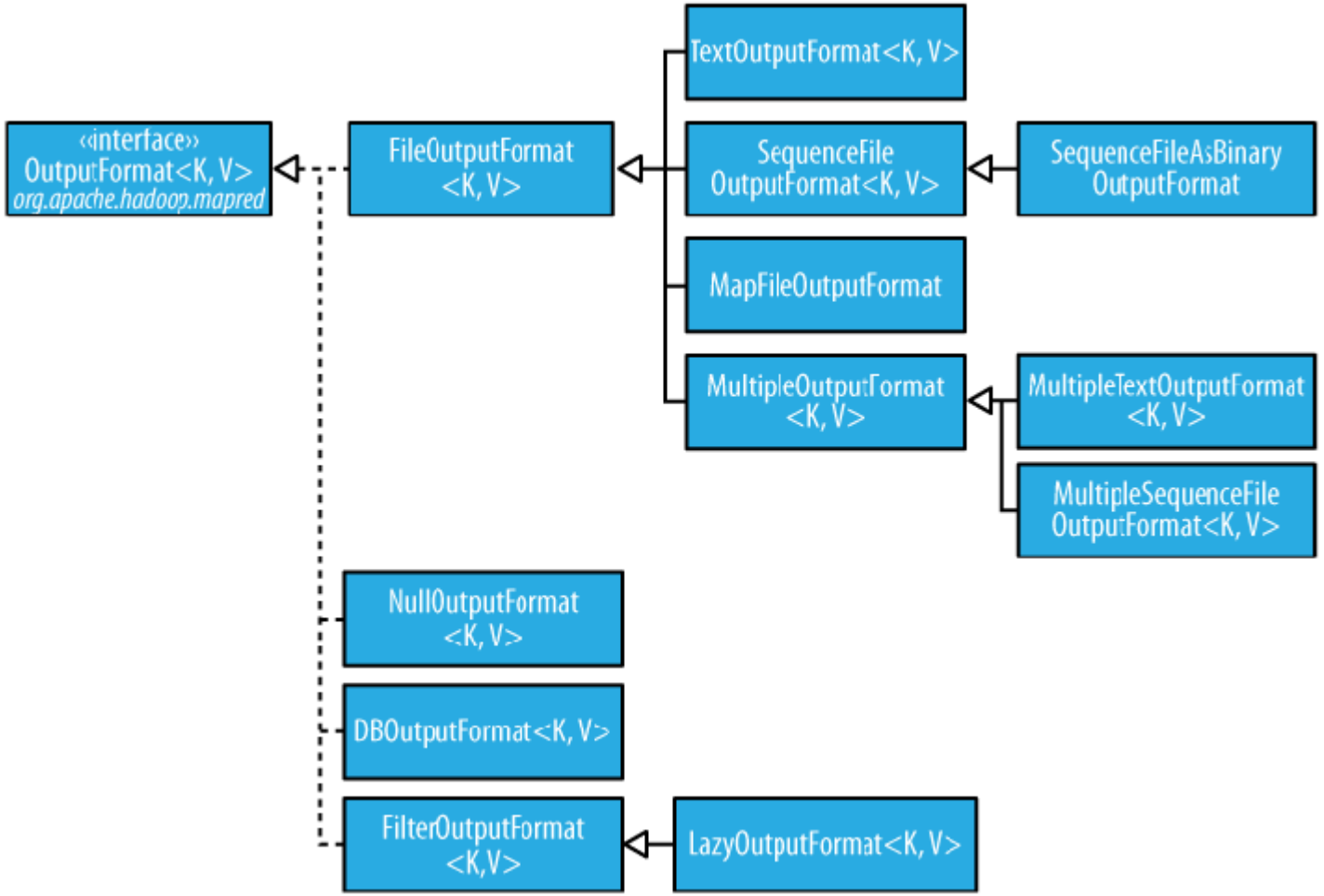
- 自定义Partitioner。

控制相同的类的数据作为一个单独的文件输出。

```
public static class CountPartition extends HashPartitioner<Text, IntWritable> {

    @Override
    public int getPartition(Text key, IntWritable value, int numReduceTasks) {
        if(key.equals(new Text("count"))) {
            return 1;
        }else {
            //其余使用默认的分区方式,此时传递的分区数应该-1
            return super.getPartition(key, value, numReduceTasks - 1);
        }
    }
}
```

```
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setPartitionerClass(CountPartition.class);
job.setNumReduceTasks(2);
```

# MapReduce编程模型-OutputFormat



# MapReduce编程模型

- Map阶段
  - ✓ InputFormat(默认为TextInputFormat)
  - ✓ Mapper
  - ✓ Combiner(local Reducer)
  - ✓ Partitioner
- Reduce阶段
  - ✓ Reducer
  - ✓ OutputFormat

# MapReduce Shuffle和排序



MapReduce确保每个reduce的输入都是按键排序的。系统指定排序的过程(即map输出作为reduce的输入)称为shuffle。理解shuffle的过程有助于我们理解MapReduce的工作机制和优化MapReduce。

Mapreduce.task.io.sort.mb
mapreduce.map.sort.spill.percent

**Map端：**

- Map输出并非简单输出到磁盘，而是缓冲的方式写入内存并做预排序。当缓冲区满了则刷入磁盘。
- Map缓冲区每刷一次磁盘，产生一个溢出文件。Map输出结束后，所有溢出文件会被合并为一个已分区且已排序的输出文件。
- 将压缩后的map结果输出可以减少磁盘写入量和网络传输量。

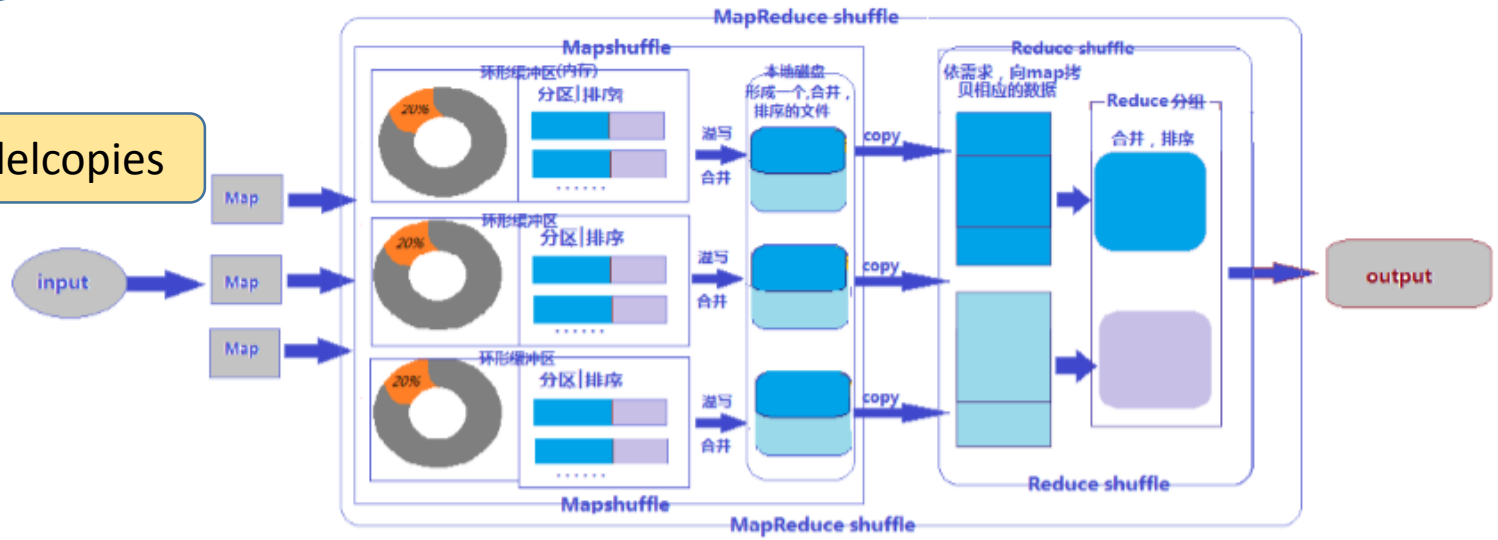
Mapreduce.map.output.compress

mapreduce.task.io.sort.factor:控制一次合并多少流。

**Reduce端：**

- 只要有一个map任务完成，reduce便开始复制其输出，reduce有少量的线程做复制。
- 复制完所有map输出后，reduce便会进入合并阶段。合并后的数据作为reduce输入。

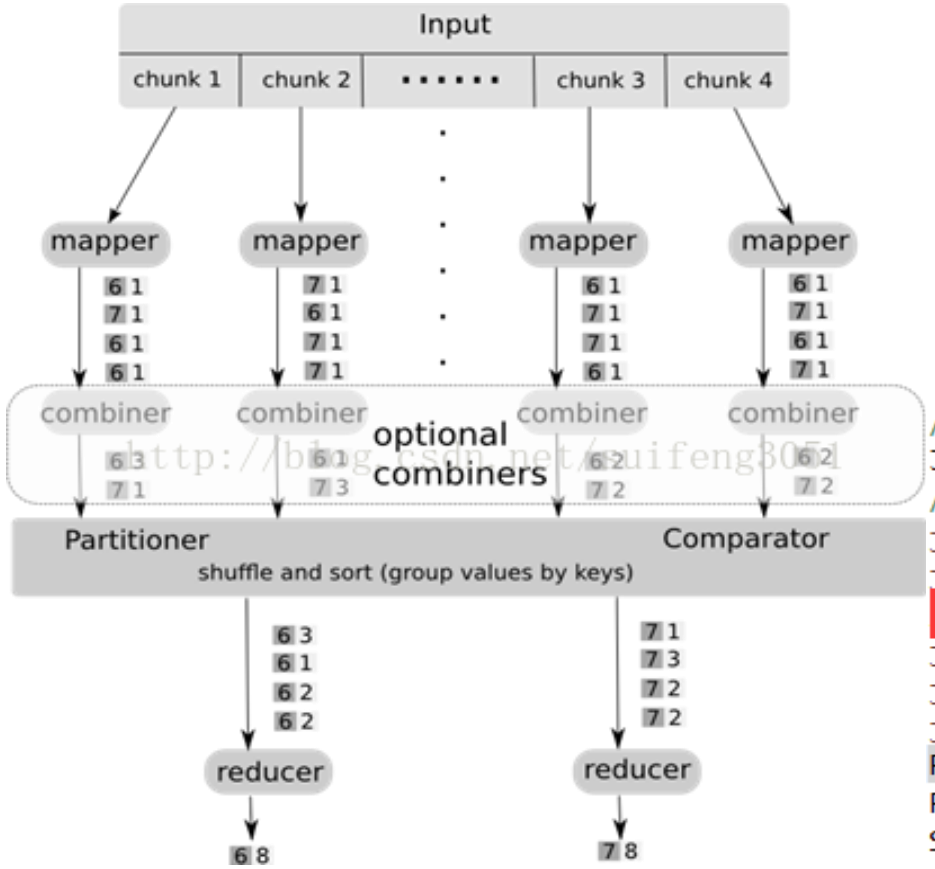
Mapreduce.reduce.shuffle.parallelcopies



# MapReduce Combiner函数



集群上最紧俏的资源便是网络带宽，因此尽量减少map和reduce阶段的网络传输对MapReduce的性能提升是很重要的。Hadoop为map任务的输出指定了一个合并函数(combiner)，合并函数的输出作为reduce的输入。Combiner是的map的输出结果更加紧凑，同时减少了写磁盘和网络传输的数据量。**Combiner又称为Local Reducer。**




例如：map输出结果{(6,1),(7,1),(6,1),(6,1)}，统计每个数字出现的次数。

没有combiner函数时，到reduce的输入数据为： {(6,1),(7,1),(6,1),(6,1)}

增加了Combiner函数后，到reduce的输入数据为{(6,3)(7,1)}

依次，如果在大量数据的情况下，这样可以减少很大的网络传输。

※注意：并不是所有的场合都适合做combiner,例如AVG。

```
//配置作业名
Job job = new Job(conf,"word count");
//配置作业各个类
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setCombinerClass(Reduce.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

# MapReduce压缩



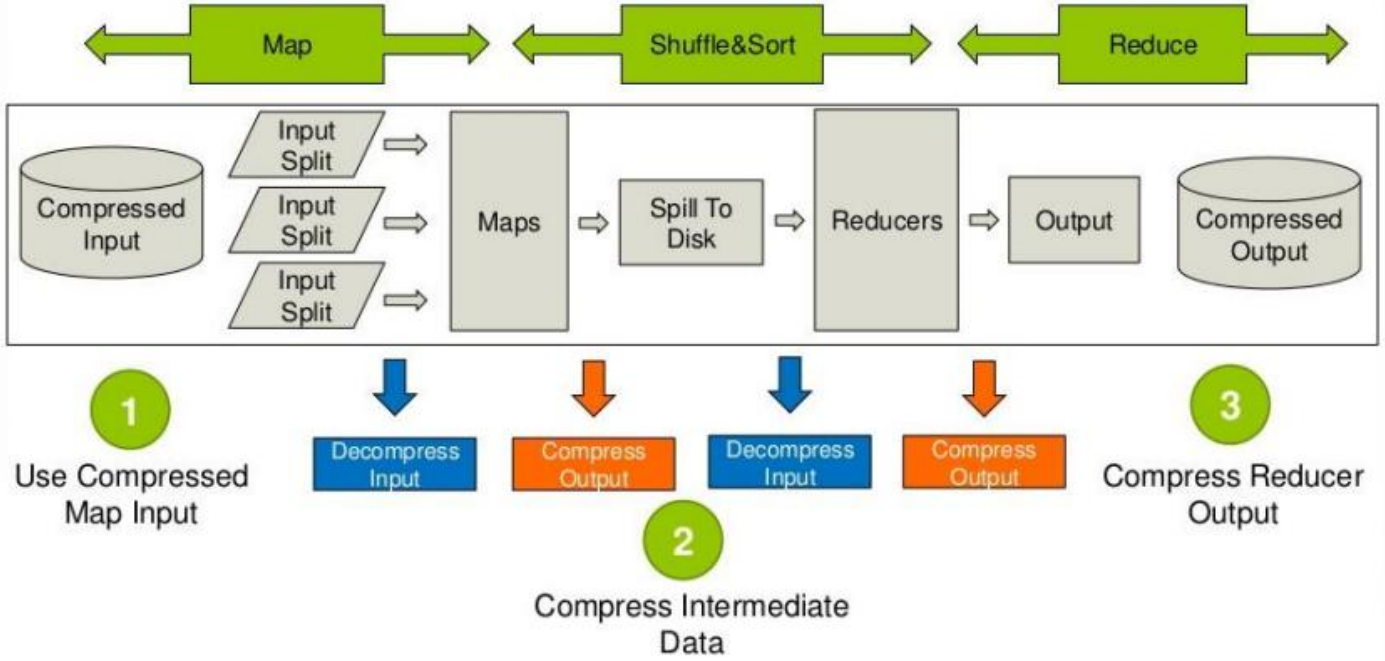
- 压缩可以发生在三个地方：

输入数据压缩、map输出压缩、reduce输出压缩。

- 压缩的优点：减少网络I/O，减少磁盘占用。
- 压缩的缺点：增加CPU计算量。
- 选择压缩格式

速度 or 空间。

支持分割机制的压缩。~~Gzip~~->bzip->lzo。




- MapReduce中如何配置压缩：

MapRed会1.输入文件的压缩。
根据文件后缀自动识别使用哪种压缩解码器。
2.输出压缩：
mapreduce.output.fileoutputformat.compress，mapreduce.output.fileoutputformat.compress.type(block/Row)，mapreduce.output.fileoutputformat.compress.codec
3.Map输出压缩：
mapreduce.map.output.compress，mapreduce.map.output.compress.codec

Hadoop支持的压缩格式对比

| 压缩格式 | split | native | 压缩率 | 速度 | 是否hadoop自带 | linux命令 | 换成压缩格式后，原来的应用程序是否要修改 |
|---|---|---|---|---|---|---|---|
| gzip | 否 | 是 | 很高 | 比较快 | 是，直接使用 | 有 | 和文本处理一样，不需要修改 |
| lzo | 是 | 是 | 比较高 | 很快 | 否，需要安装 | 有 | 需要建索引，还需要指定输入格式 |
| snappy | 否 | 是 | 比较高 | 很快 | 否，需要安装 | 没有 | 和文本处理一样，不需要修改 |
| bzip2 | 是 | 否 | 最高 | 慢 | 是，直接使用 | 有 | 和文本处理一样，不需要修改 |

- Gzip:当每个文件压缩之后在130M以内的（1个块大小内），都可以考虑用gzip压缩格式。
- Lzo:一个很大的文本文件，压缩之后还大于200M以上的可以考虑，而且单个文件越大，lzo优点越越明显。
- Snappy:当mapreduce作业的map输出的数据比较大的时候，作为map到reduce的中间数据的压缩格式；或者作为一个mapreduce作业的输出和另外一个mapreduce作业的输入。
- Bzip2:适合对速度要求不高，但需要较高的压缩率的时候，可以作为mapreduce作业的输出格式；或者输出之后的数据比较大，处理之后的数据 需要压缩存档减少磁盘空间并且以后数据用得比较少的情况；或者对单个很大的文本文件想压缩减少存储空间，同时又需要支持split，而且兼容之前的应用程 序（即应用程序不需要修改）的情况。

- Reducer个数。
- 输入：大文件优于小文件。
- 减少网络传输：压缩、combiner
- 优化Map数量。

# MapReduce应用案例

- 排序实现。
- 数据去重实现。
- Mapreduce实战Top K算法。
- Mapreduce实现条件查询。
- Mapreduce实战多表关联。
- Mapreduce实现行数统计。
- Mapreduce实现计算平均值。

从给定的文件中的找到最大的100个值。

```
1  pi10mr,WT02287,2015-09-01 00:00:00,1,3.54,0,0,214.89,0,40.76,40.64,20.5,26.31,47.05,0,0,0,0,0,0,0,-0.52,0.32,9.46539e+06,0,0,0,27
2  pi10mr,WT02287,2015-09-01 00:10:00,1,3.22,0,0,224.85,0,40.61,40.53,20.43,26.2,46.75,0,0,0,0,0,0,0,-0.3,0.48,9.46539e+06,0,0,0,20
3  pi10mr,WT02287,2015-09-01 00:20:00,1,2.81,0,0,217.79,0,40.57,40.3,20.37,26.07,46.44,0,0,0,0,0,0,0,-0.5,0.31,9.46539e+06,0,0,0,24
4  pi10mr,WT02287,2015-09-01 00:30:00,1,3.67,0,0,3[illegible]53,0,40.36,40.16,20.32,25.88,46.13,0,0,0,0,0,0,0,-0.47,0.39,9.46539e+06,0,0,0,15
5  pi10mr,WT02287,2015-09-01 00:40:00,1,3.69,0,0,332.9,0,40.16,40,20.17,25.61,45.77,0,0,0,0,0,0,0,-0.05,0.38,9.46539e+06,0,0,0,21
6  pi10mr,WT02287,2015-09-01 00:50:00,1,4.04,0,0,112.91,0,39.57,39.47,20.21,25.86,45.43,0,0,0,0,0,0,0,-0.22,0.05,9.46539e+06,0,0,0,23
7  pi10mr,WT02287,2015-09-01 01:00:00,1,4.42,8.92,0,37.96,0,39.25,39.11,20.33,25.88,45.15,0,0,0,0,0,0,0,-0.08,0.01,9.46539e+06,0,0,0,26
8  pi10mr,WT02287,2015-09-01 01:10:00,1,3.57,1092.65,14.87,217.13,0,39.65,40.27,20.03,28.05,45.39,0,0,0,0,0,0,0,-0.3,0.33,9.46539e+06,0,0,0,23
9  pi10mr,WT02287,2015-09-01 01:20:00,1,2.84,1320.28,18.24,332.9,0,39.88,41.87,19.79,28.1,44.83,0,0,0,0,0,0,0,-0.31,0.37,9.46539e+06,0,0,0,29
```

思路：数据分片后，每个分片计算出一个topK。然后所有的map结果进入一个Reducer计算出最终的topK。计算topK的时候借助java的TreeMap来实现自动排序。

运行结果：

```
21.72
21.73
22.15
22.27
22.41
22.62
22.82
23.1
23.2
23.27
```

**作业：给出不同风机2015年的10min数据。计算出每台风机每月份的风速TOP10。**

# MapReduce实例-条件查询

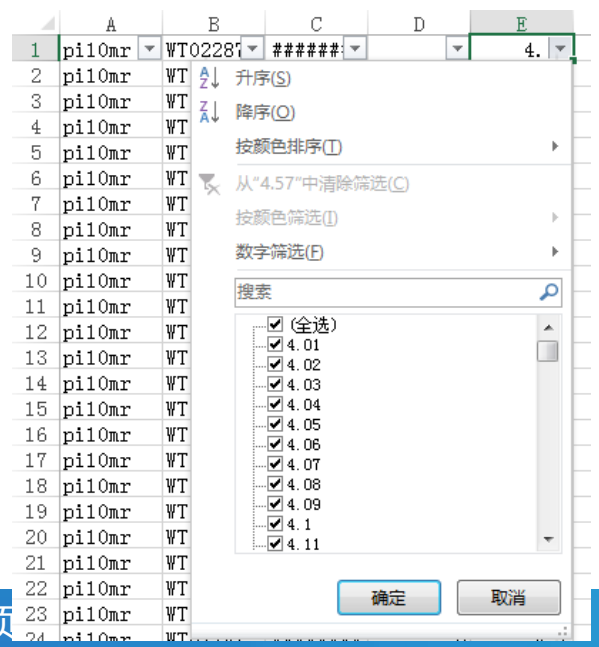


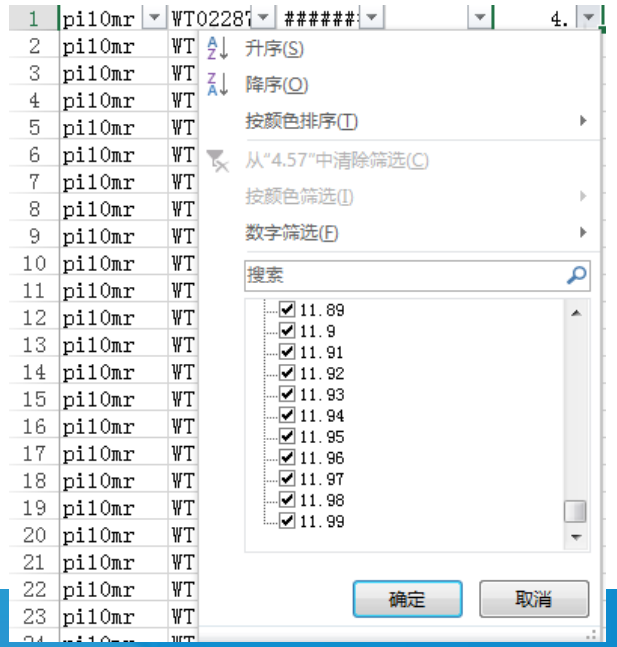
给出某风机2015年的10min数据，实现SQL：select * from a where a.windspeed >4 and a.windspeed < 12的功能。

```
pi10mr,WT02287,2015-09-01 00:00:00,1,3.54,0,0,214.89,0,40.76,40.64,20.5,26.31,47.05,0,0,0,0,0,0,0,0,-0.52,0.32,9.46539e+06,0,0,0,27
pi10mr,WT02287,2015-09-01 00:10:00,1,3.22,0,0,224.85,0,40.61,40.53,20.43,26.2,46.75,0,0,0,0,0,0,0,0,-0.3,0.48,9.46539e+06,0,0,0,20
pi10mr,WT02287,2015-09-01 00:20:00,1,2.81,0,0,217.79,0,40.57,40.3,20.37,26.07,46.44,0,0,0,0,0,0,0,0,-0.5,0.31,9.46539e+06,0,0,0,24
pi10mr,WT02287,2015-09-01 00:30:00,1,3.67,0,0,3[风速]53,0,40.36,40.16,20.32,25.88,46.13,0,0,0,0,0,0,0,0,-0.47,0.39,9.46539e+06,0,0,0,15
pi10mr,WT02287,2015-09-01 00:40:00,1,3.69,0,0,332.9,0,40.16,40,20.17,25.61,45.77,0,0,0,0,0,0,0,0,-0.05,0.38,9.46539e+06,0,0,0,21
pi10mr,WT02287,2015-09-01 00:50:00,1,4.04,0,0,112.91,0,39.57,39.47,20.21,25.86,45.43,0,0,0,0,0,0,0,0,-0.22,0.05,9.46539e+06,0,0,0,23
pi10mr,WT02287,2015-09-01 01:00:00,1,4.42,8.92,0,37.96,0,39.25,39.11,20.33,25.88,45.15,0,0,0,0,0,0,0,0,-0.08,0.01,9.46539e+06,0,0,0,26
pi10mr,WT02287,2015-09-01 01:10:00,1,3.57,1092.65,14.87,217.13,0,39.65,40.27,20.03,28.05,45.39,0,0,0,0,0,0,0,0,-0.3,0.33,9.46539e+06,0,0,0,23
pi10mr,WT02287,2015-09-01 01:20:00,1,2.84,1320.28,18.24,332.9,0,39.88,41.87,19.79,28.1,44.83,0,0,0,0,0,0,0,0,-0.31,0.37,9.46539e+06,0,0,0,29
```

思路：在map阶段取出风速后，直接对风速做比较，如果满足条件则map输出，不满足条件map不输出。Reduce阶段不做任何处理，直接输出。

运行结果：



# MapReduce实例-条件查询

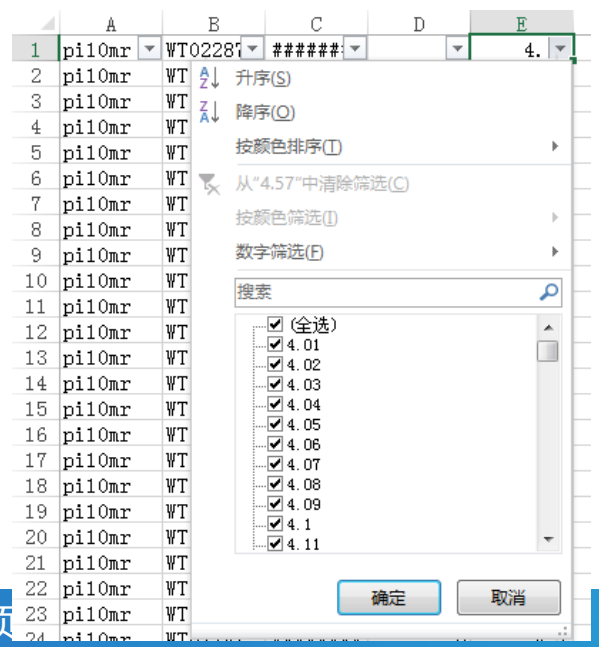


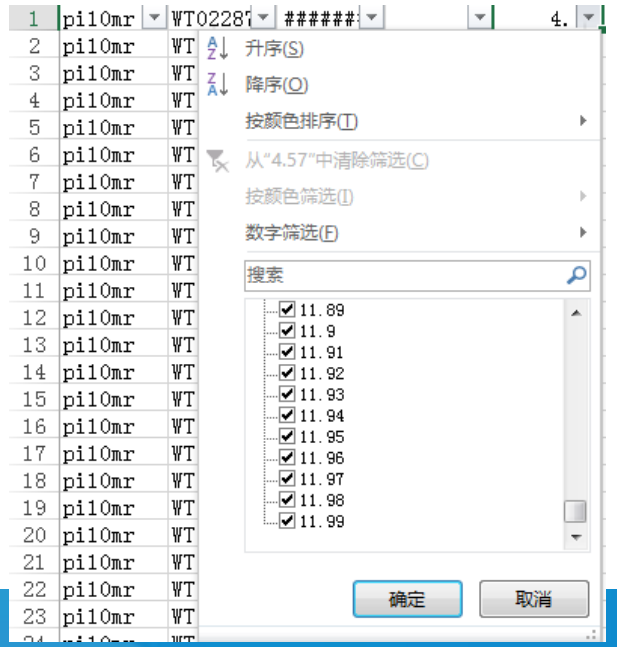
给出某风机2015年的10min数据，实现SQL：select * from a where a.windspeed >4 and a.windspeed < 12的功能。

```
pi10mr,WT02287,2015-09-01 00:00:00,1,3.54,0,0,214.89,0,40.76,40.64,20.5,26.31,47.05,0,0,0,0,0,0,0,0,-0.52,0.32,9.46539e+06,0,0,0,27
pi10mr,WT02287,2015-09-01 00:10:00,1,3.22,0,0,224.85,0,40.61,40.53,20.43,26.2,46.75,0,0,0,0,0,0,0,0,-0.3,0.48,9.46539e+06,0,0,0,20
pi10mr,WT02287,2015-09-01 00:20:00,1,2.81,0,0,217.79,0,40.57,40.3,20.37,26.07,46.44,0,0,0,0,0,0,0,0,-0.5,0.31,9.46539e+06,0,0,0,24
pi10mr,WT02287,2015-09-01 00:30:00,1,3.67,0,0,3[风速]53,0,40.36,40.16,20.32,25.88,46.13,0,0,0,0,0,0,0,0,-0.47,0.39,9.46539e+06,0,0,0,15
pi10mr,WT02287,2015-09-01 00:40:00,1,3.69,0,0,332.9,0,40.16,40,20.17,25.61,45.77,0,0,0,0,0,0,0,0,-0.05,0.38,9.46539e+06,0,0,0,21
pi10mr,WT02287,2015-09-01 00:50:00,1,4.04,0,0,112.91,0,39.57,39.47,20.21,25.86,45.43,0,0,0,0,0,0,0,0,-0.22,0.05,9.46539e+06,0,0,0,23
pi10mr,WT02287,2015-09-01 01:00:00,1,4.42,8.92,0,37.96,0,39.25,39.11,20.33,25.88,45.15,0,0,0,0,0,0,0,0,-0.08,0.01,9.46539e+06,0,0,0,26
pi10mr,WT02287,2015-09-01 01:10:00,1,3.57,1092.65,14.87,217.13,0,39.65,40.27,20.03,28.05,45.39,0,0,0,0,0,0,0,0,-0.3,0.33,9.46539e+06,0,0,0,23
pi10mr,WT02287,2015-09-01 01:20:00,1,2.84,1320.28,18.24,332.9,0,39.88,41.87,19.79,28.1,44.83,0,0,0,0,0,0,0,0,-0.31,0.37,9.46539e+06,0,0,0,29
```

思路：在map阶段取出风速后，直接对风速做比较，如果满足条件则map输出，不满足条件map不输出。Reduce阶段不做任何处理，直接输出。

运行结果：



# MapReduce实例-多表关联

- 提供风机台账数据taizhang.csv,提供风机数据fanData.csv，数据具体格式参见excel文件。

要求实现以下SQL语句的功能：

SELECT b.*,a.风场名称,a.所属区域,a.项目公司, a.风机类型 FROM taizhang a, fanData b where a.fanNo=b.FanNo;

- 思路：map阶段获取输入文件名，map阶段输出的KEY=风机编号，但是value值根据文件名做下区分。
- Reduce阶段的输入数据是相同风机编号的数据，然后分别找出fanData数据，找出台账数据。想fanData数据中添加台账的字段。

运行结果：

```
1  pi10mr,WT02287,2015-11-30 23:50:00,0,4.57,0,0,267,0,11,12.5,-4.32,-1.1,5.6,0,0,0,0,0,0,0,0,-0.75,0.23,9.75146e+06,0,0,0,4_fandata,富饶山风电场,辽宁,沈阳龙源雄亚风
2  pi10mr,WT02287,2015-11-30 23:40:00,0,4.8,0,0,359,0,11.1,13.1,-4.47,-1.3,5.95,0,0,0,0,0,0,0,0,-0.25,0.22,9.75146e+06,0,0,0,4_fandata,富饶山风电场,辽宁,沈阳龙源雄亚
3  pi10mr,WT02287,2015-11-30 23:30:00,0,4.07,0,0,355,0,11.27,13.63,-4.67,-1.23,6.37,0,0,0,0,0,0,0,0,0,0.13,9.75146e+06,0,0,0,3_fandata,富饶山风电场,辽宁,沈阳龙源雄亚
4  pi10mr,WT02287,2015-11-30 23:20:00,0,4.26,0,0,347.2,0,11.34,14.52,-5.02,-1.1,6.92,0,0,0,0,0,0,0,0,0,0.6,0.1,9.75146e+06,0,0,0,5_fandata,富饶山风电场,辽宁,沈阳龙源雄
5  pi10mr,WT02287,2015-11-30 23:10:00,0,3.86,0,0,330.14,0,11.4,15.27,-5.16,-1.09,7.58,0,0,0,0,0,0,0,0,0,-0.79,0.12,9.75146e+06,0,0,0,14_fandata,富饶山风电场,辽宁,沈阳龙
6  pi10mr,WT02287,2015-11-30 23:00:00,0,4.37,0,0,348.04,0,11.42,16.4,-5.23,-1,8.58,0,0,0,0,0,0,0,0,0,-0.38,0.1,9.75146e+06,0,0,0,24_fandata,富饶山风电场,辽宁,沈阳龙源雄
7  pi10mr,WT02287,2015-11-30 22:50:00,0,4.12,1053.67,14.48,346.41,0,12.53,18.63,-5,-0.72,8.68,0,0,0,0,0,0,0,0,0,-6,0.05,9.75146e+06,0,0,0,46_fandata,富饶山风电场,辽宁,
8  pi10mr,WT02287,2015-11-30 22:40:00,0,4.4,762.89,9.93,230.28,0,10.93,12.86,-5.16,-0.92,5.52,0,0,0,0,0,0,0,0,0,-3.37,0.12,9.75147e+06,0,0,0,81_fandata,富饶山风电场,辽
9  pi10mr,WT02287,2015-11-30 22:30:00,0,4.25,200.14,1.39,47.75,0,10.19,10.28,-5.33,-1.32,4.94,0,0,0,0,0,0,0,0,0,-3.8,0.16,9.75146e+06,0,0,0,76_fandata,富饶山风电场,辽宁
10 pi10mr,WT02287,2015-11-30 22:20:00,0,5.41,32.05,0.2,98.38,0,9.12,8.4,-5.4,-2.27,4.53,0,0,0,0,0,0,0,0,0,-1.75,0.28,9.75146e+06,0,0,0,76_fandata,富饶山风电场,辽宁,沈阳
11 pi10mr,WT02287,2015-11-30 22:10:00,0,4.42,0,0,56.52,0,9.16,8.51,-5.51,-2.45,4.54,0,0,0,0,0,0,0,0,0,-1.43,0.27,9.75147e+06,0,0,0,83_fandata,富饶山风电场,辽宁,沈阳龙源
12 pi10mr,WT02287,2015-11-30 22:00:00,0,2.69,0,0,180.13,0,9.3,8.72,-5.77,-2.6,4.57,0,0,0,0,0,0,0,0,0,-1.52,0.22,9.75147e+06,0,0,0,84_fandata,富饶山风电场,辽宁,沈阳龙源
13 pi10mr,WT02287,2015-11-30 21:50:00,0,2.63,0,0,131,0,9.48,8.9,-5.75,-2.46,4.59,0,0,0,0,0,0,0,0,0,-1.34,0.31,9.75147e+06,0,0,0,77_fandata,富饶山风电场,辽宁,沈阳龙源雄
```

# MapReduce实例-count()实现

**作业：**

提供风机数据fanData.csv，数据具体格式参见excel文件。
要求实现以下SQL语句的功能：

```
SELECT count(b.fanNo) FROM  fanData b;
SELECT COUNT(b.fanNo) From fanData b where b.ws > 4 and b.ws < 12;
```

实现思路：
Map阶段根据风速过滤数据。每个map统计出各自数据的数据条数。
Reduce将各个map的统计结果汇总。

**作业：**

提供风机数据fanData.csv，数据具体格式参见excel文件。

要求实现以下SQL语句的功能：

SELECT avg(b.ws) FROM  fanData b;

SELECT b.月份,b.fanNo,avg(b.ws) FROM  fanData b group by b.月份,b.fanNo。

实现思路：

Map阶段计算一个sum和count。如果有group by的话，按照月份分组时，map的输出key=月份_fanNo。

Reduce阶段对相同的月份_fanNo数据计算AVG，并输出。

# 复习&预习

Hadoop技术内幕 深入理解MapReduce架构设计与实现原理，第2,3,9章节。
hadoop权威指南（第二版）第6,7,8章节。

# 总结

- Mapreduce简介
  - ✓ 分而治之
  - ✓ 抽象出Mapper和Reducer
  - ✓ 隐藏实现细节
- Mapreduce的编程模型
  - ✓ Map阶段
  - ✓ Reduce阶段
- Mapreduce的优化
  - ✓ Combine
  - ✓ 压缩
  - ✓ Shuffle

# THANKS

致敬每一个看似微渺的小梦想！

致敬每一位正在路上的追梦人！