中科院计算培训中心

Part 2 MapReduce/Spark计算模式

基于分布式文件的并行计算

- 杨文川

1）分布式文件系统DFS
2）MapReduce计算模型介绍
3）使用MR进行算法设计
4）DAG及其算法设计

大数据快速处理

- 互联网应用引发了对大数据进行快速处理的需求。
  - 在很多互联网应用中，数据量相当大且规整，这给应用并行化处理技术提供了大量机会。
- 下面是一些例子：
  - 1)Web网页按重要性排序，包括迭代的矩阵-向量乘法计算，其矩阵、向量的维度达到百亿维
  - 2)在社交网站上的朋友关系网络中进行搜索，该网络图结构，包括上亿个节点和几十亿条边。

为啥采用 DFS+MapReduce

- 大数据环境下，数据挖掘技术的关键点是，
- **在大数据基础上的简单算法，比小数据基础上的复杂算法更加有效。**
  - 在少量数据情况下运行得最好的算法，在大量数据条件下运行得最不好的。
  - 当数据只有500万的时候，某些一种简单的算法表现得很差
  - 但当数据达10亿的时候，它变成了表现最好的，准确率从原来的75%提高到了95%以上。

传统算法的问题

- 小数据时代--采用数据库技术，追求挖掘结果的精确性
  - 传统的样本分析师，需要采用复杂的随机采样方法，确保每个数据的精确性，才不会导致分析结果的偏差。
- 随机采样取是统计分析的主心骨
  - 它的成功依赖于采样的绝对随机性，但是实现采样的随机性非常困难。
  - 一旦采样过程中存在任何偏见，分析结果就会相去甚远。

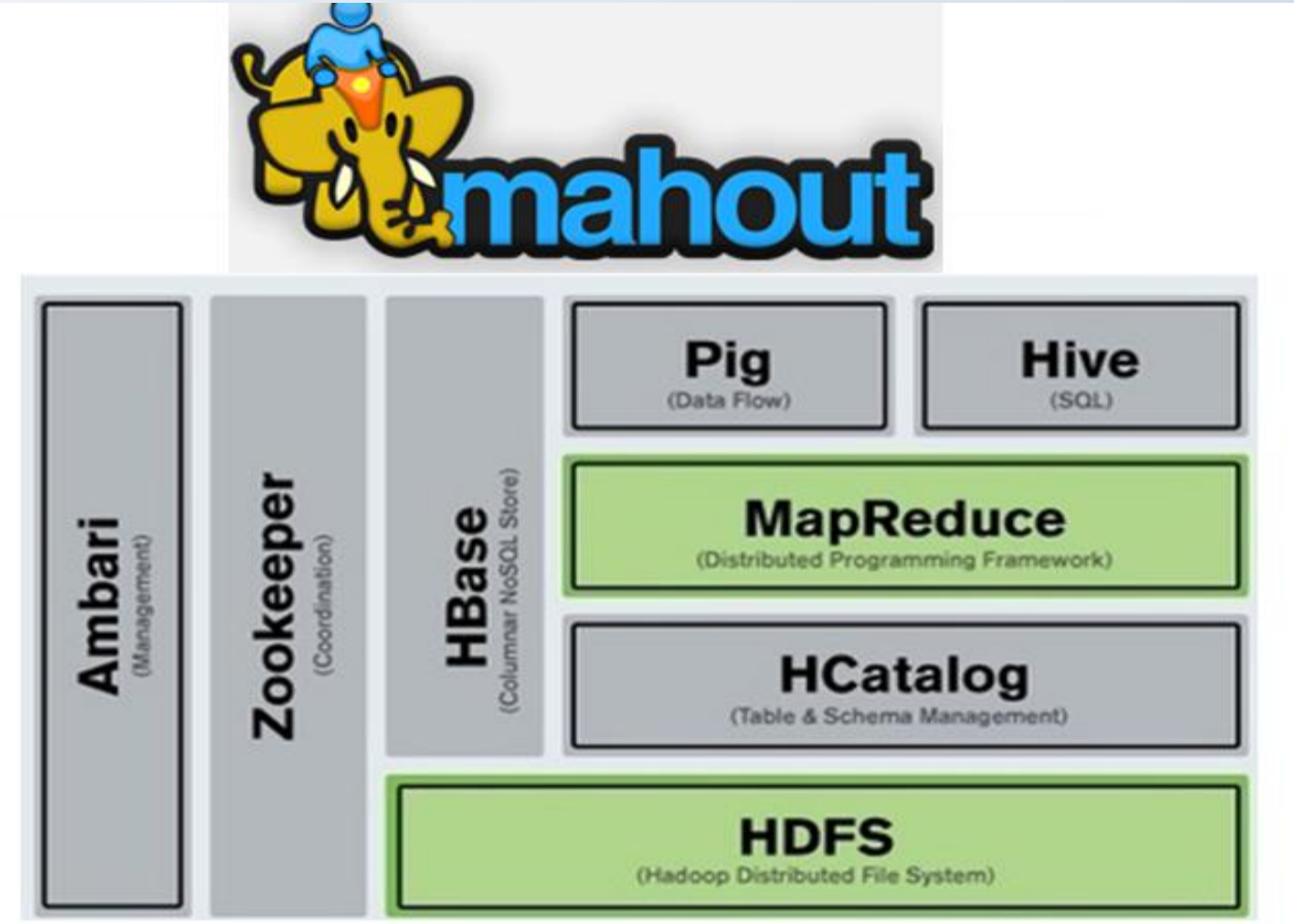
大数据时代--并行实时处理
追求分析数据的相关性

- 大数据时代， 我们要接受纷繁的数据并从中受益，而不是以高昂的代价消除所有不确定性。
- **全数据模式-样本就是总体**

知道是什么就够了

- 之前专业的分析人员需要深入了解，是什么让客户做出了选择，要把握客户做决定背后的真正原因
  - 专业技能和多年的经验受到高度重视
- 大数据挖掘中，知道是什么就够了，没必要知道为什么。
  - Amazon的推荐系统显示，通过大数据梳理出的相关关系，更有用。

Mahout与Hadoop成员关系



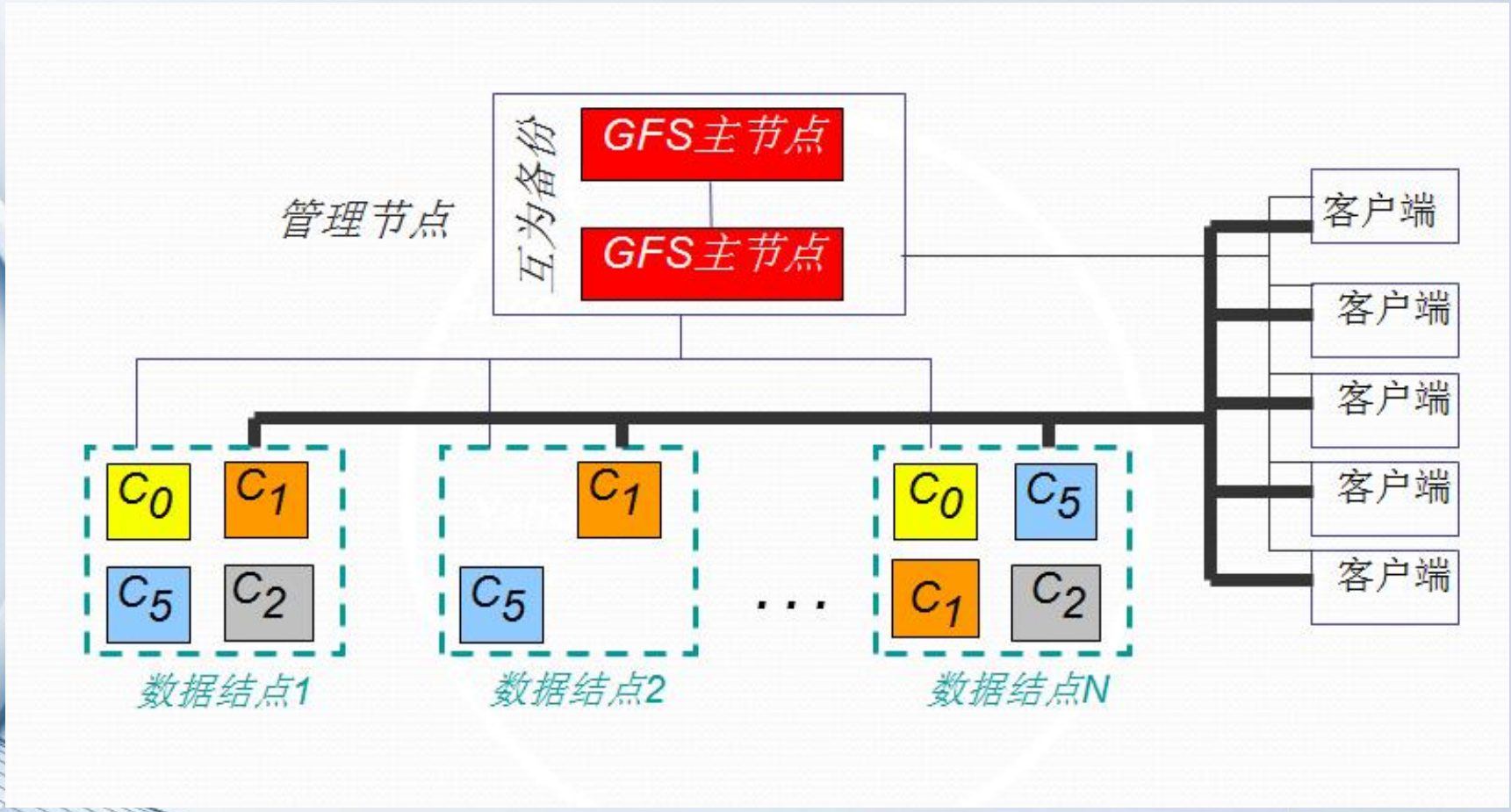
采用DFS和MapReduce

- 一个新的软件栈用于处理上述应用。
- 下层是分布式文件系统DFS，主要特征是：
  - 存储单位比传统操作系统中的磁盘块大很多
  - 提供数据冗余机制，来防止数据分布在上千块磁盘上时频发媒介故障。
- 在DFS上，采用MapReduce编程系统。
  - MapReduce能够在大规模计算机集群上，高效实现并行计算
  - 而且它能够支持计算过程的硬件容错性

分布式文件系统

- 大规模Web服务的流行，使得越来越多的计算，是在拥有几千个计算节点的计算装置上完成的
  - 这些节点之间或多或少相互独立，计算节点都由商用硬件构成，与采用专用硬件的并行计算机相比，大大降低了硬件开销
  - 这些系统能够发挥并行化的优势，同时可以避免可靠性问题。其中，GFS就是一个分布式文件系统的典型

GFS分布式文件系统

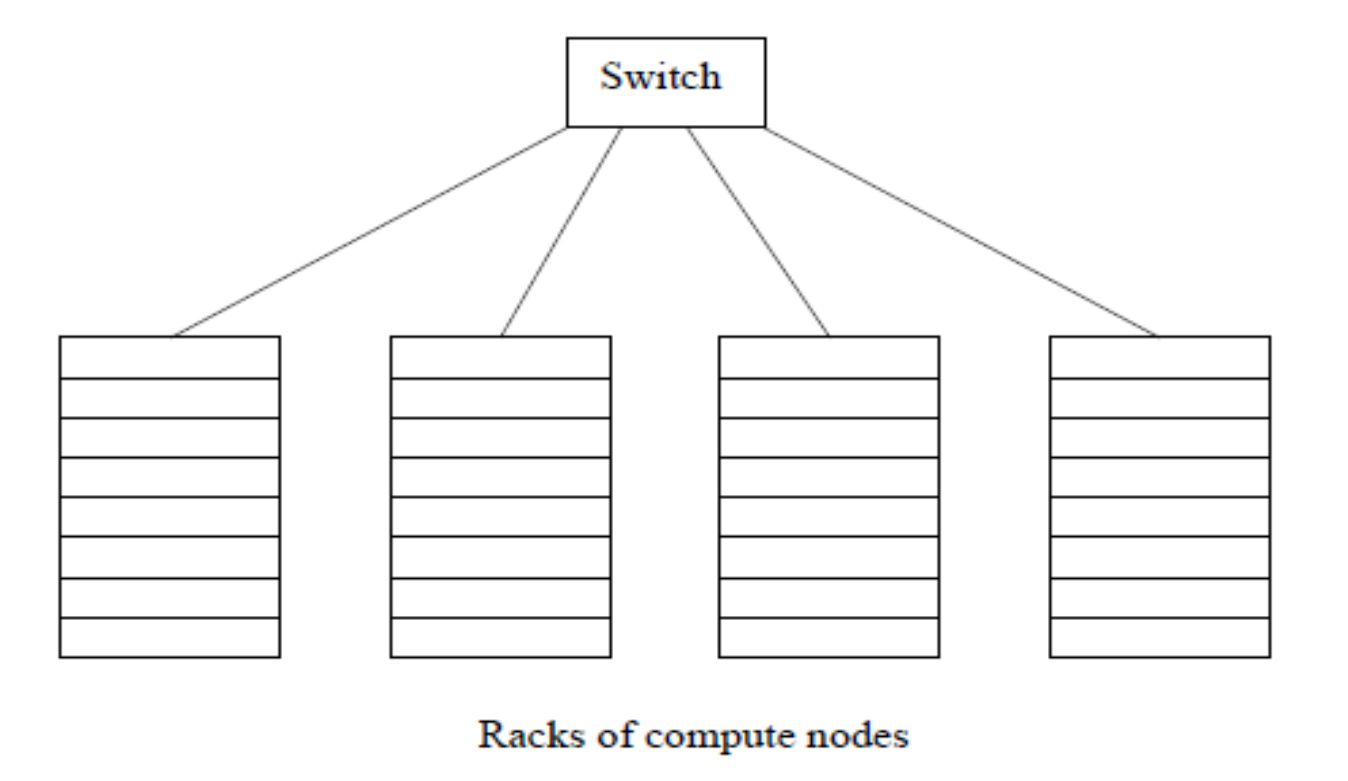

管理节点
互为备份
GFS主节点
GFS主节点
客户端
客户端
客户端
客户端
客户端
C0
C1
C5
C2
C1
C5
C0
C5
C1
C2
数据结点1
数据结点2
数据结点N

计算节点的物理结构

- 这种新的并行计算架构也称为集群计算，组织方式如下
  - 计算节点放在机架中，每个机架可以安放8~64个节点。
  - 单个机架上的节点之间通过网络互联，通常采用千兆以太网。
  - 计算节点可能需要多个机架来安放，这些机架之间采用另一级网络或交换机互连。
  - 机架间的通信带宽，一般略微高于机架内以太网的带宽

集群计算图

- 计算节点安放在机架上，通过交换机互连



并行计算编程策略

- 1)文件必须多副本存储
  - 把文件放在多个计算节点上备份
  - 一旦某个节点出故障，该节点被替换，它上面的所有文件将继续使用。
- 2)计算过程必须要分成多个任务
  - 一旦某个任务失败，可以在不影响其他任务的情况下，重启这个任务。
  - MapReduce编程系统采用了这种策略

分布式文件系统的结构

- 在DFS中，文件被分成文件块(chunk)，文件块的大小通常为64MB。
  - 文件块会被复制多个副本(如复制三份)放在三个不同的计算节点上。
  - 存放同一文件块不同副本的节点，应分布在不同机架上，这样在某个机架发生故障时就不至于丢失所有副本。
  - 文件块的大小和复制的次数，可以由用户指定

Master Node

- 通过存在主节点(master node)或名字节点(name node)的小文件，来寻找某个文件的文件块。
  - 主节点本身可以有多个副本，文件系统的总目录可以用于寻找主节点的副本。
  - 总目录本身也可以有多个副本，所有使用DFS的用户，都知道这些目录副本所在的位置

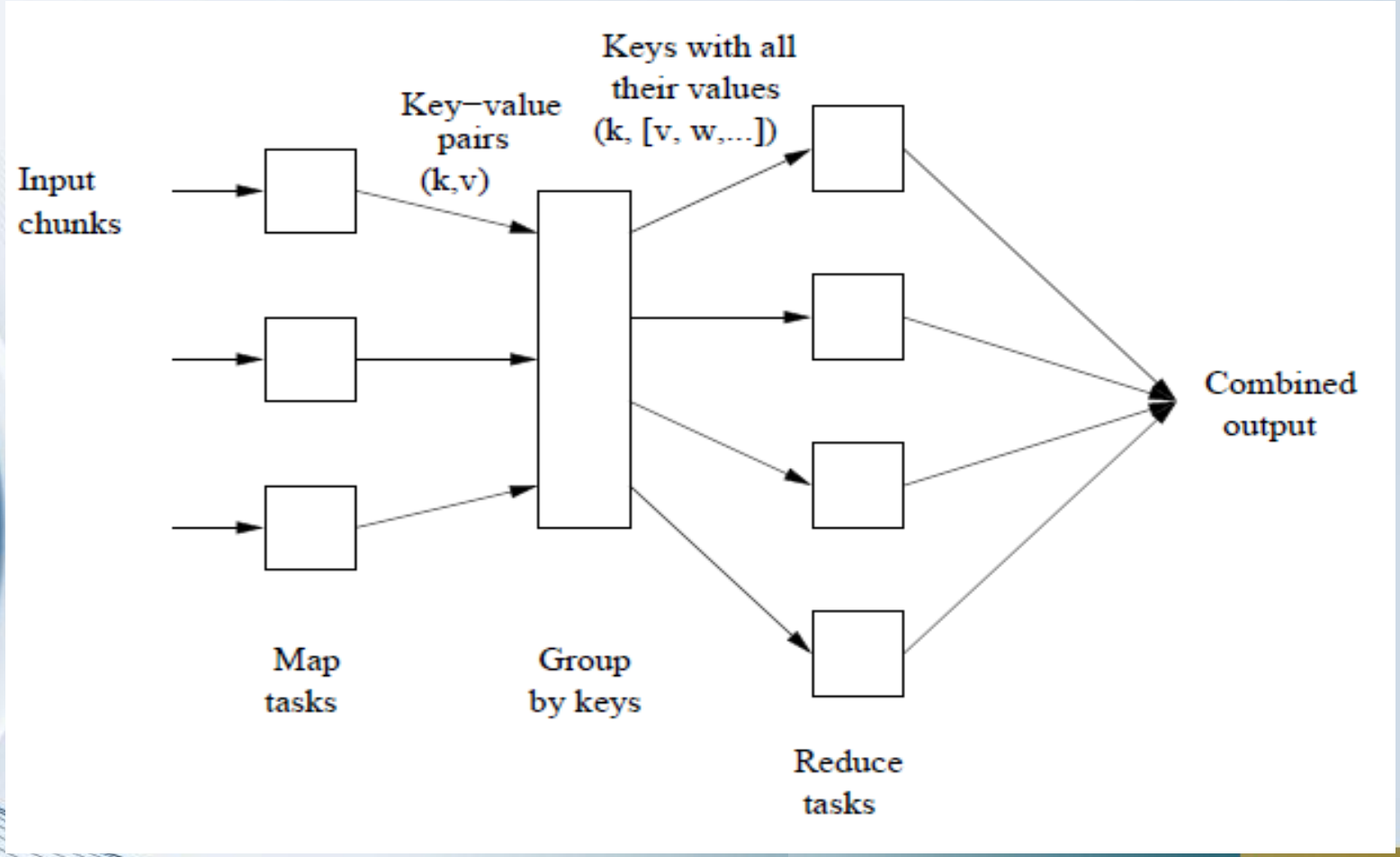
MapReduce

- MapReduce是一种并行计算模式，已有多个实现系统，例如Hadoop。
  - 可通过某个MapReduce实现系统，来管理多个大规模计算过程，并且同时能够保障对硬件故障的容错性。
  - 需编写两个称做Map和Reduce的函数，系统能够管理Map或Reduce并行任务的执行，以及任务之间的协调，并且能处理上述某个任务执行失败的情况

基于MapReduce的计算过程

- 1)有多个Map任务，每个Map任务的输入是DFS中的一个或多个文件块。
  - Map任务将文件块转换为一个键值对(KVP) 序列
- 2)主控制器从每个Map任务中收集一系列键值对，并将其分到所有的Reduce任务中
  - 具有相同键值对的数据，归到同一Reduce任务中
- 3)Reduce任务每次作用于一个键
  - 并将与此键关联的所有值，以某种方式组合起来

MapReduce计算过程示意图



Map任务

- Map任务的输入文件，可以看成由多个元素(element)组成，而元素可以是任意类型
  - 比如一个元组或一篇文档。
  - 文档文件块是一系列元素的集合
  - 同一个元素不能跨文件块存储。
- Map函数将输入元素转换成键值对。
  - 其中的键和值都可以是任意类型。

一个MapReduce计算的经典例子

- 计算每个词语在整个文档集中的出现次数。
  - 输入文件是一个文档集，每篇文档都是一个元素
  - Map函数使用键类型是词语，值类型是整数
  - Map任务读入一篇文档并将它分成词语序列
    - $w_1, w_2, \ldots, w_n$
  - 输出一个键值对序列，其中所有的值都是1。
  - Map任务作用于文档的输出结果，是键值对序列
    - $(w_1, 1), (w_2, 1), \ldots, (w_n, 1)$

Hadoop中单词统计实现

假设有一批海量的数据，每个数据都是由26个字母组成的字符串，原始的数据集合是完全无序的，怎样通过MapReduce完成排序工作，使其有序（字典序）呢？

排序通常用于衡量分布式数据处理框架的数据处理能力

实验结果

Input:
File containing words

Hello World Bye World
Hello Hadoop Bye Hadoop
Bye Hadoop Hello Hadoop

Output:
Number of occurrences of each word

Bye 3
Hadoop 4
Hello 3
World 2

对原始的数据进行分割（Split），得到N个不同的数据分块

Split1：
nklklacdcdd
gfgdfsdfdfd
……
annnbnbvgh

Split2：
dfgmdhjydf
kghfgcxnkil
……
gjghyotewgbb

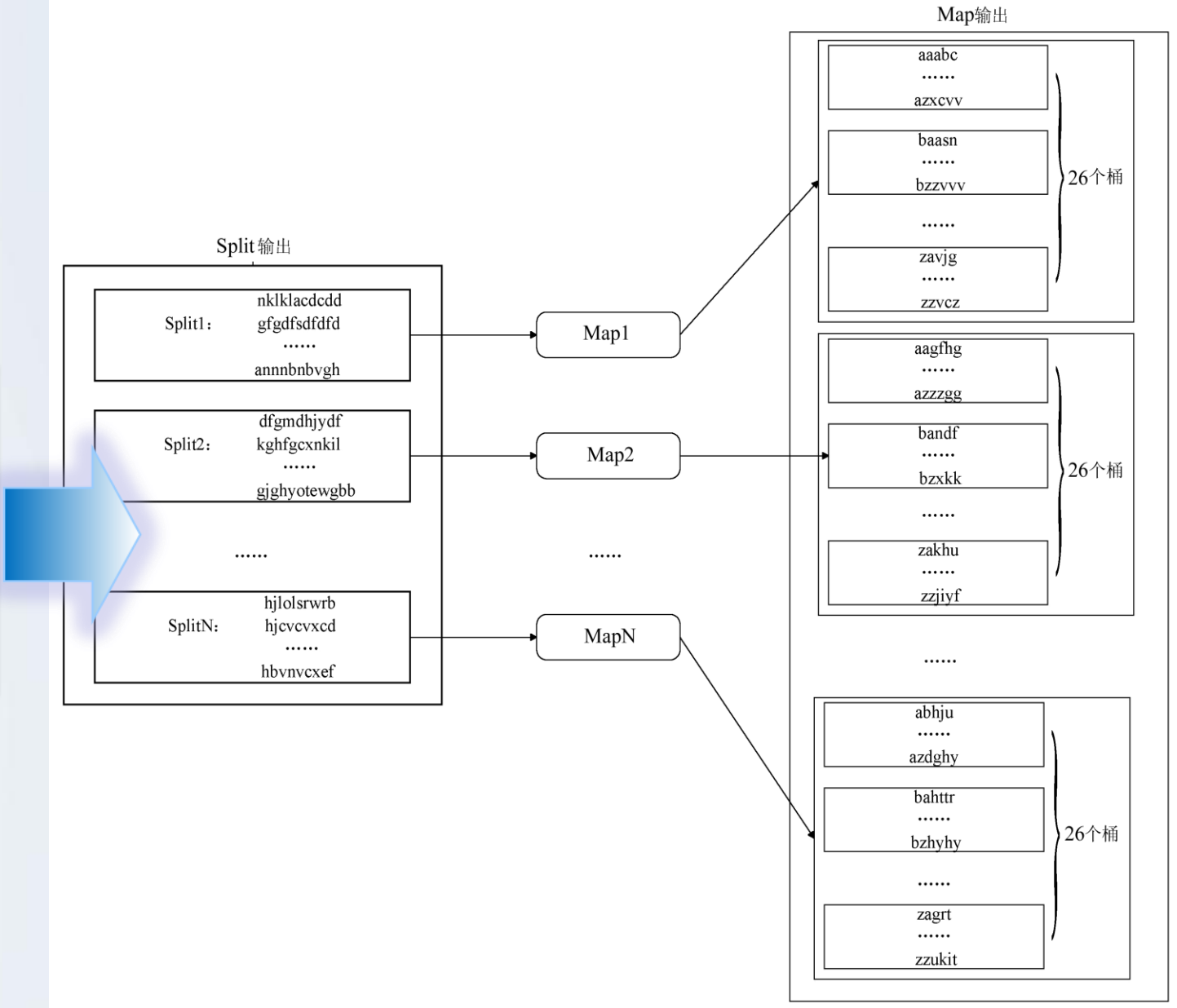
……

SplitN：
hjlolsrwrb
hjcvcvxcd
……
hbvnvcxef

Splits分片

- 将 MapReduce 的输入数据，划分成等长的小数据块split.
  - Split意味着处理每个分片所需的时间，将少于处理整个输入数据所化时间.
- 为每个分片建立一个map任务
  - 并由该任务来执行用户定义的map函数，从而处理分片中的每条记录.

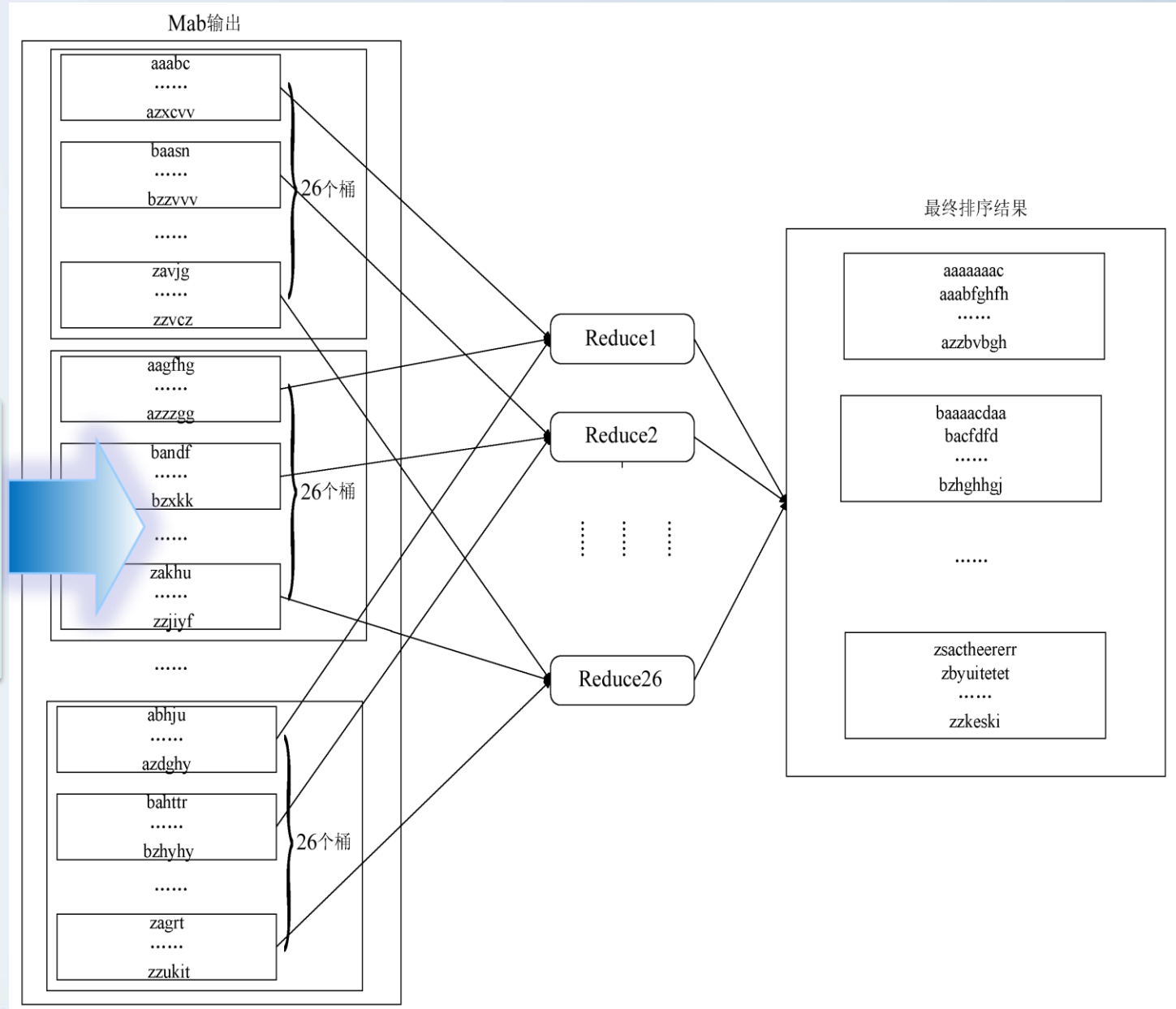
每一个数据分块都启动一个Map进行处理。
每个Map中按照首字母将字符串分配到26个不同的桶中



实现Map类

- 这个类实现 Mapper 接口中的 map 方法，输入参数中的 value 是文本文件中的一行，
- 利用StringTokenizer 将这个字符串拆成单词，然后将输出结果 <单词,1> 写入到输出中。

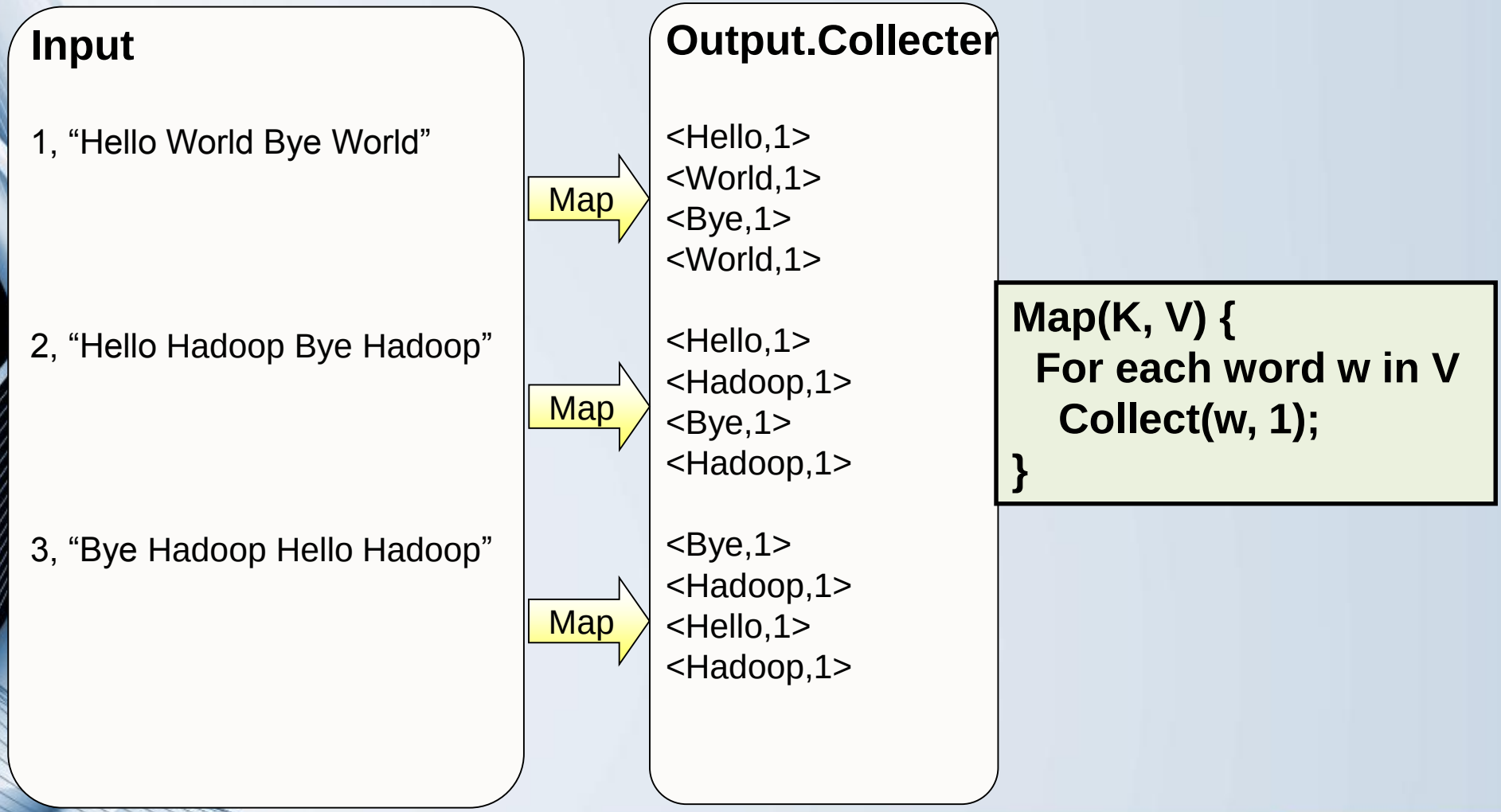
按照首字母将Map中不同桶中的字符串集合放置到相应的Reduce中进行处理。



实现 Reduce 类

- 这个类实现 Reducer 接口中的 reduce 方法, 输入参数中的 key, values 是由 Map 任务输出的中间结果，
- values 是一个 Iterator, 遍历这个 Iterator, 就可以得到属于同一个 key 的所有 value.
  - 此处，key是一个单词，value 是词频。只需要将所有的 value 相加，就可以得到这个单词的总的出现次数。

WordCount-Map



WordCount-Reduce

Reduce Input

<Hello,1>
<World,2>
<Bye,1>

<Hello,1>
<Hadoop,2>
<Bye,1>

<Bye,1>
<Hadoop,2>
<Hello,1>

Internal Grouping

<Bye → 1, 1, 1>

<Hadoop → 2, 2>

<Hello → 1, 1, 1>

<World → 2>

Reduce

Reduce

Reduce

Reduce

Reduce Output

<Bye, 3>
<Hadoop, 4>
<Hello, 3>
<World, 2>

```
Reduce(K, V[ ]) {
  Int count = 0;
  For each v in V
    count += v;
  Collect(K, count);
}
```

运行 Job

- 在 Hadoop 中一次计算任务称之为一个 job, 可以通过一个 Job 对象设置如何运行这个 job。
- 然后将 Job对象作为参数，调用 runJob, 开始执行这个计算任务。

实例分析：WordCount

```
public int run(String[] args) throws Exception {
    JobConf conf = new JobConf(getConf(), WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));

    JobClient.runJob(conf);
    return 0;
  }

public static void main(String[] args) throws Exception {
    if(args.length != 2){
      System.err.println("Usage: WordCount <input path> <output path>");
      System.exit(-1);
    }
    int res = ToolRunner.run(new Configuration(), new WordCount(), args);
    System.exit(res);
  }
}
```

分组和聚合

- 分组和聚合基于相同的方式来处理。
- 主控进程知道Reduce任务的数目(如r个)
  - r通常由用户指定并通知MapReduce系统
- 主控进程通常选择一个哈希函数，作用于键，并产生一个0到r-1的桶编号。

- Map任务输出的每个键都被哈希函数作用，根据哈希结果，其键值对将被放入r个本地文件中的一个。
- 当所有Map任务都成功完成之后，主控进程将每个Map任务输出的，面向某个特定Reduce任务的文件进行合并
- 并将合并文件以“键值表”对(KVP)序列传给该进程。

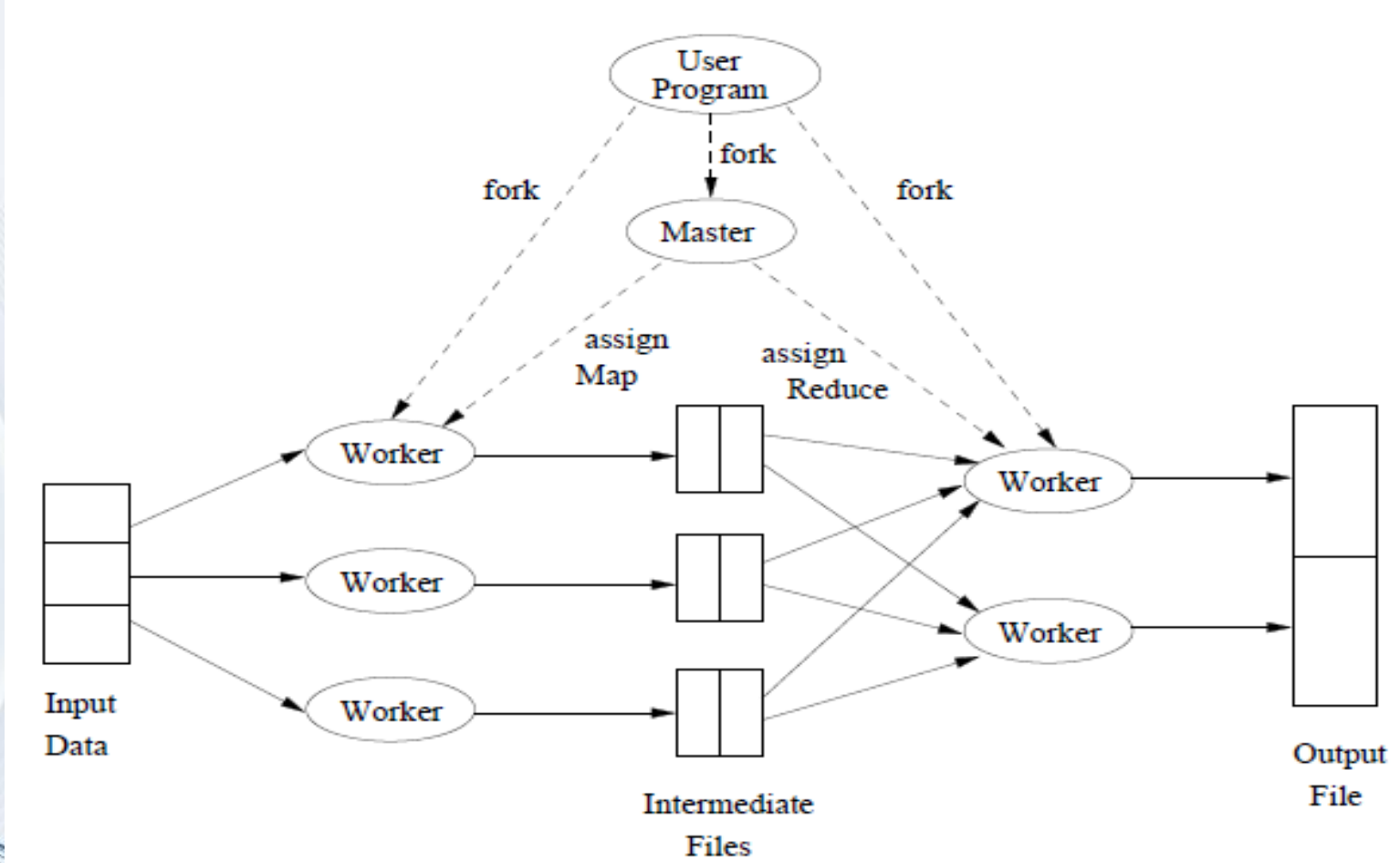
Reduce任务

- Reduce函数将输入的一系列键值表中的值，以某种方式组合。
  - Reduce任务的输出是键值对序列，其中每个键值对中的键k，是Reduce任务接收到的输入键，而值是其接收到的与k关联的值表的组合结果。
  - 所有Reduce任务的输出结果，会合并成单个文件

组合器

- 通常Reduce函数都满足交换律和结合律
  - 所有需要组合的值，可以按照任何次序组合，其结果不变。
- Reduce函数满足交换律和结合律时，就可以将Reduce任务中的部分工作，放到Map任务中来完成
  - 如求最大值，最小值等，但求平均值等可能不适用

MapReduce执行细节



Map节点失效的处理

- 当某个运行Map任务的计算节点崩溃，主控进程会定期检查，发现节点的崩溃情况。
  - 所有分配到该工作进程的Map任务将重新执行，甚至已经完成的Map任务都可能要重启，因为输出的结果还在计算节点上，不再可用
  - 主控进程将需要重启的所有Map任务的状态都置为“空闲”，并当某个工作进程可用时安排它们重新运行。
  - 主控进程通知每个Reduce任务，它们的输入位置(即对应Map任务的输出位置)已经发生改变

Reduce节点失效的处理

- 如果运行Reduce任务的计算节点失效，处理上要简单一些。
  - 主控进程只是将失效节点上运行的Reduce任务的状态置为“空闲”
  - 并安排给另外的工作节点，按计划日程重新运行

使用MapReduce有一定的限制

- MapReduce在分布式文件巨大的时候，效率很高
- 但MapReduce并不能解决所有问题，如：
  - 在管理Amazon com在线零售数据时，不合适采用MapReduce，即使是使用数千计算节点来处理Web请求。
  - 主要原因是，Amazon在线销售数据上的主要操作包括：应答商品搜索需求、记录销售情况，计算量相对较小，频繁更改数据等。

适合MapReduce

- 但Amazon可以使用MapReduce来执行大数据上的某些分析型查询，
  - 为每个用户，找到和他购买模式最相似的用户
- 谷歌采用MapReduce的目的，是为处理PageRank计算过程中，必需的大矩阵-向量乘法。
  - 矩阵-向量及矩阵-矩阵计算，MapReduce计算框架非常适合
  - 另一类有效采用MapReduce的重要运算，是关系代数运算

MapReduce的矩阵向量乘法实现

- 设定有一个n×n的矩阵M，其第i行第j列的元素记为m_{ij}。
- 假定有一个n维向量v，其j个元素记为v_j
- 矩阵M和向量v的乘积结果是一个n维向量x，其第i个元素x_i为

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

- 此处的n可以达到百亿！

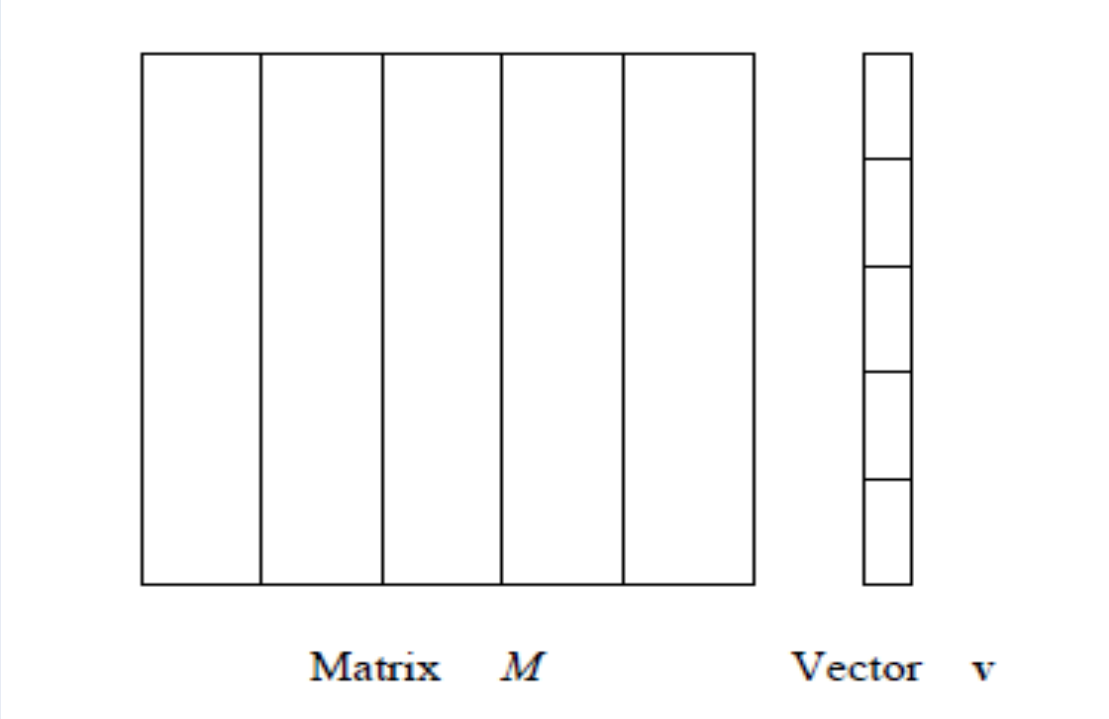
- 矩阵M和向量v各自都会在DFS中存成一个文件
- **Map函数**
  - 每个Map任务将整个向量v，和矩阵M的一个文件块作为输入。
  - 对每个矩阵元素mij，Map会产生键值对(i, mijvj)，计算xi的所有n个求和项的键值mijvj都相同。
- **Reduce函数**
  - Reduce任务将所有与给定键值i关联的值相加，即可得到(i, xi).

向量v无法放入内存时的处理

- 如果向量v很大，在内存中可能无法完整存放
  - 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
- 目标是使用足够的条，以保证向量的每个条，能够方便地放入计算节点内存中

矩阵M和向量v的分割示意图

- 分割示意图，其中矩阵和向量都分割成5个条



- 矩阵第i个垂直条，和向量的第i个水平条相乘
  - 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和Reduce任务，可以按照前面所描述的过程来运行
  - 此时Map任务获得了完整的向量

使用MapReduce算法的例子

- 美国气象局提供的气象数据
  - 数据按行并以 ASCII格式存储，每行是一条记录.
- 存储格式中包含众多的数据元素
  - 其中很多元素可以选择性的输入
  - 其数据存储的长度是可变的

问题分析

- 对气象数据中的年份和温度进行分析
- 找出
  - 最高气温
  - 平均气温
  - …
  - 待处理的气象数据如下：

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

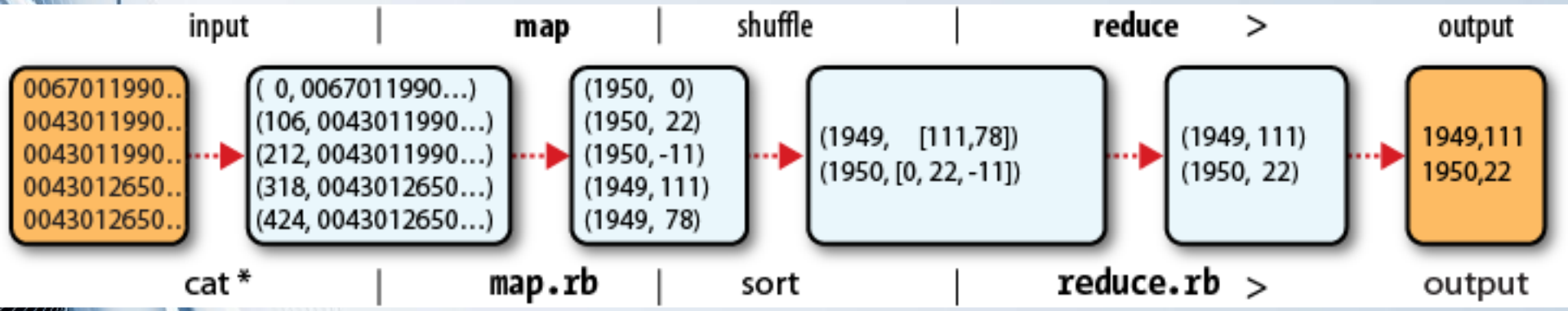
气象数据的格式

0057 332130	# USAF weather station identifier
99999	# WBAN weather station identifier
19500101	# observation date
0300	# observation time
+51317	# latitude (degrees x 1000)
+028783	# longitude (degrees x 1000)
00450	# sky ceiling height (meters)
1	# quality code
010000	# visibility distance (meters)
1	# quality code
	.....
-0128	# air temperature (degrees Celsius x 10)
1	# quality code
-0139	# dew point temperature (degrees Celsius x 10)
1	# quality code
10268	# atmospheric pressure (hectopascals x 10)
1	# quality code

方法：使用MapReduce分析

- 根据MapReduce的设计思路，采用Map 和 Reduce两阶段：
- 以键值对(KVP)作为输入/输出, 其类型由程序员选择.
- 为此要定义两个函数:
  - map 和 reduce函数.

MapReduce 数据流

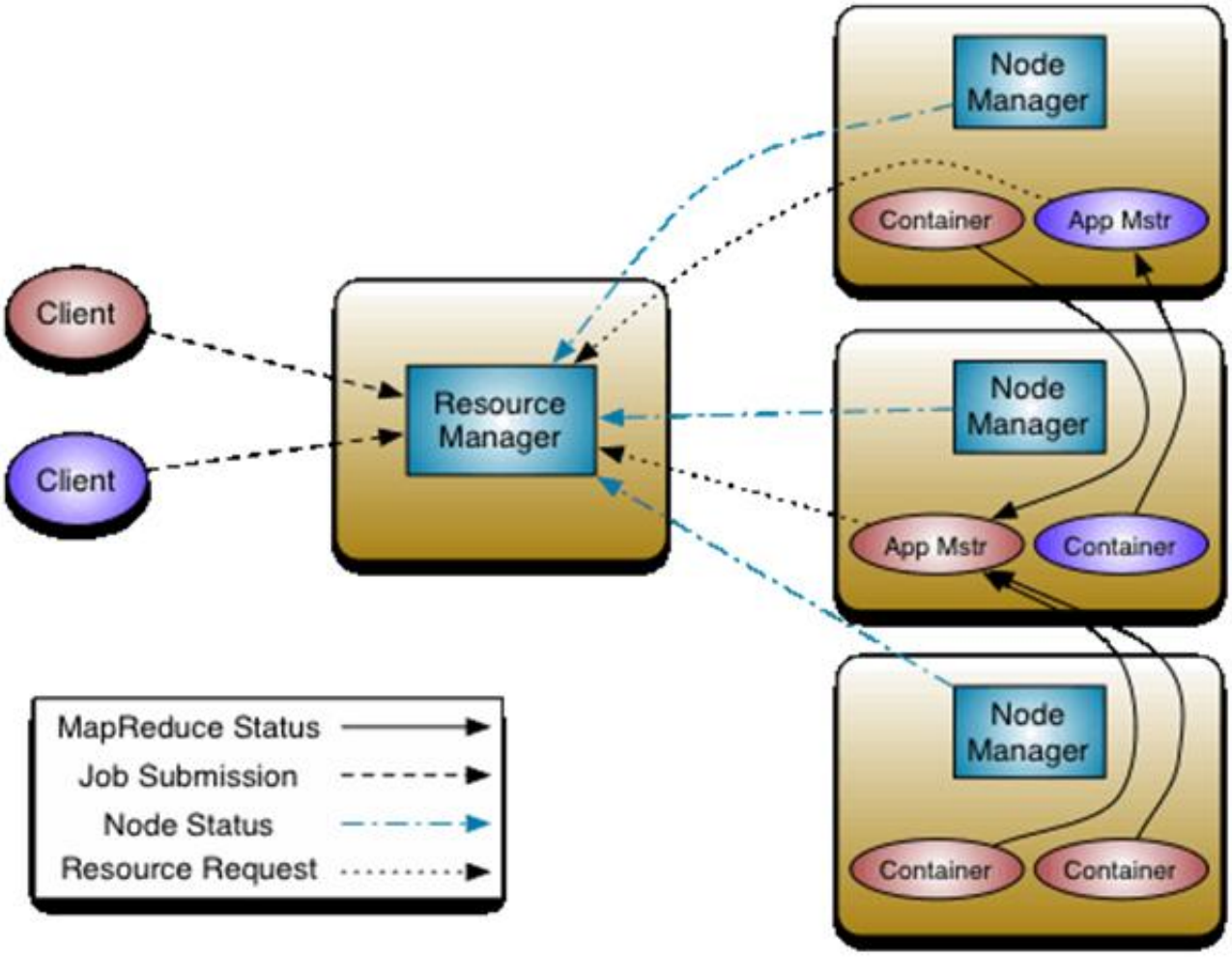


- 方法：不断的转换关注焦点

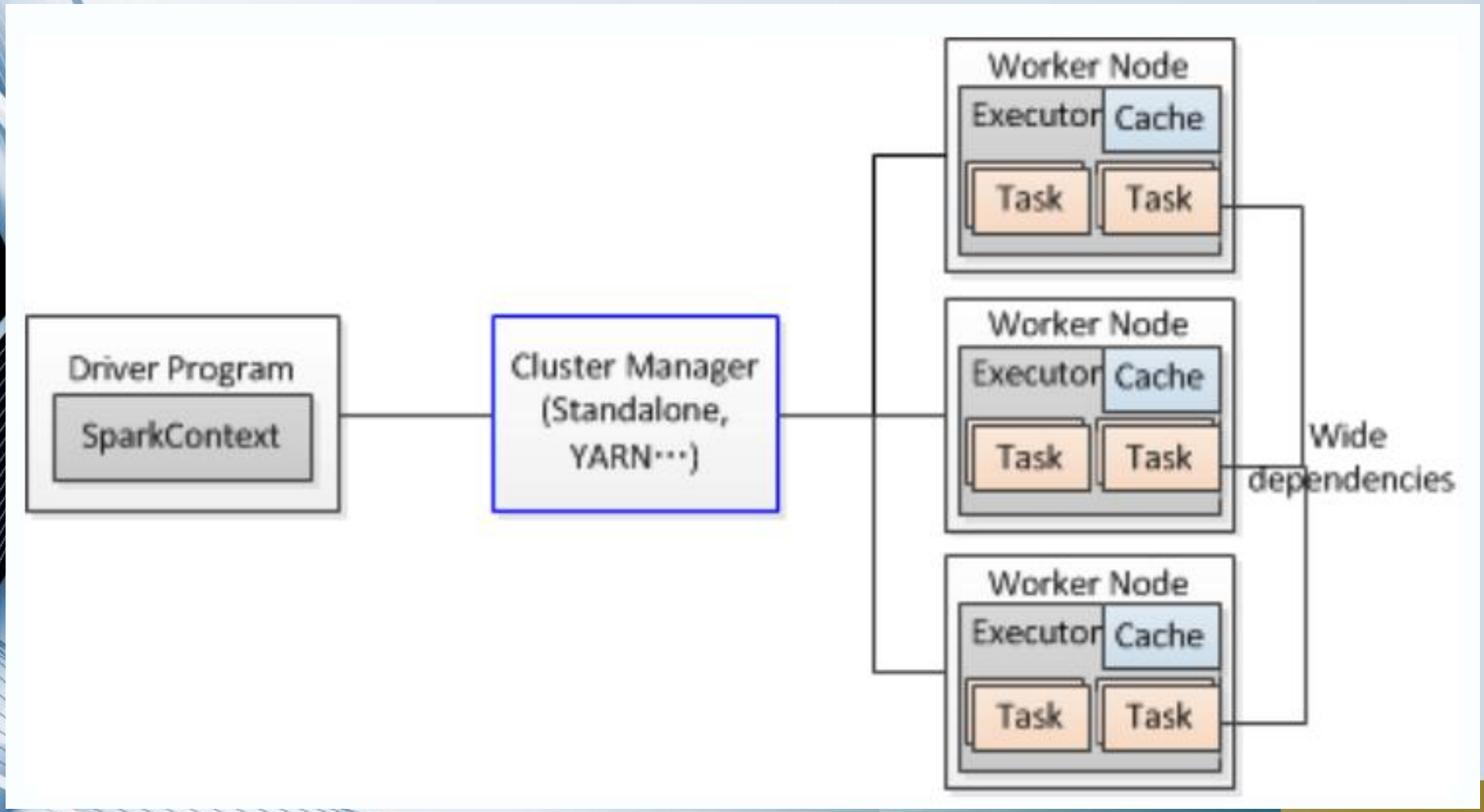
Spark vs Hadoop

- Spark是一个计算框架， 是MapReduce的替代方案
- Hadoop中不仅包含计算框架MapReduce、分布式文件系统HDFS，还包括HBase、Hive等子系统。
- Spark兼容HDFS、Hive等分布式存储层，可融入Hadoop的生态系统，弥补MapReduce的不足。

Yarn架构


Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

Spark的应用框架


Driver Program
SparkContext
Cluster Manager
(Standalone,
YARN…)
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task
Wide
dependencies

谢　谢