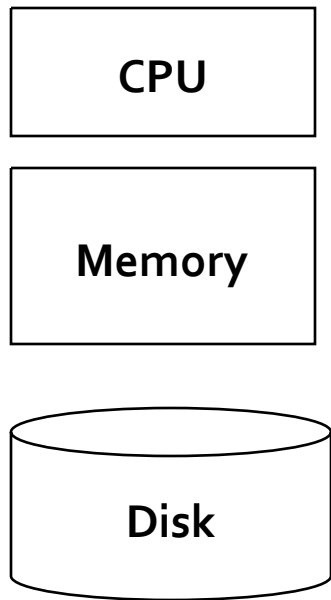


# Map-Reduce

- 分布式系统
- 计算模型
- 调度与数据流
- 改进与优化

by @寒小阳

# 单节点结构



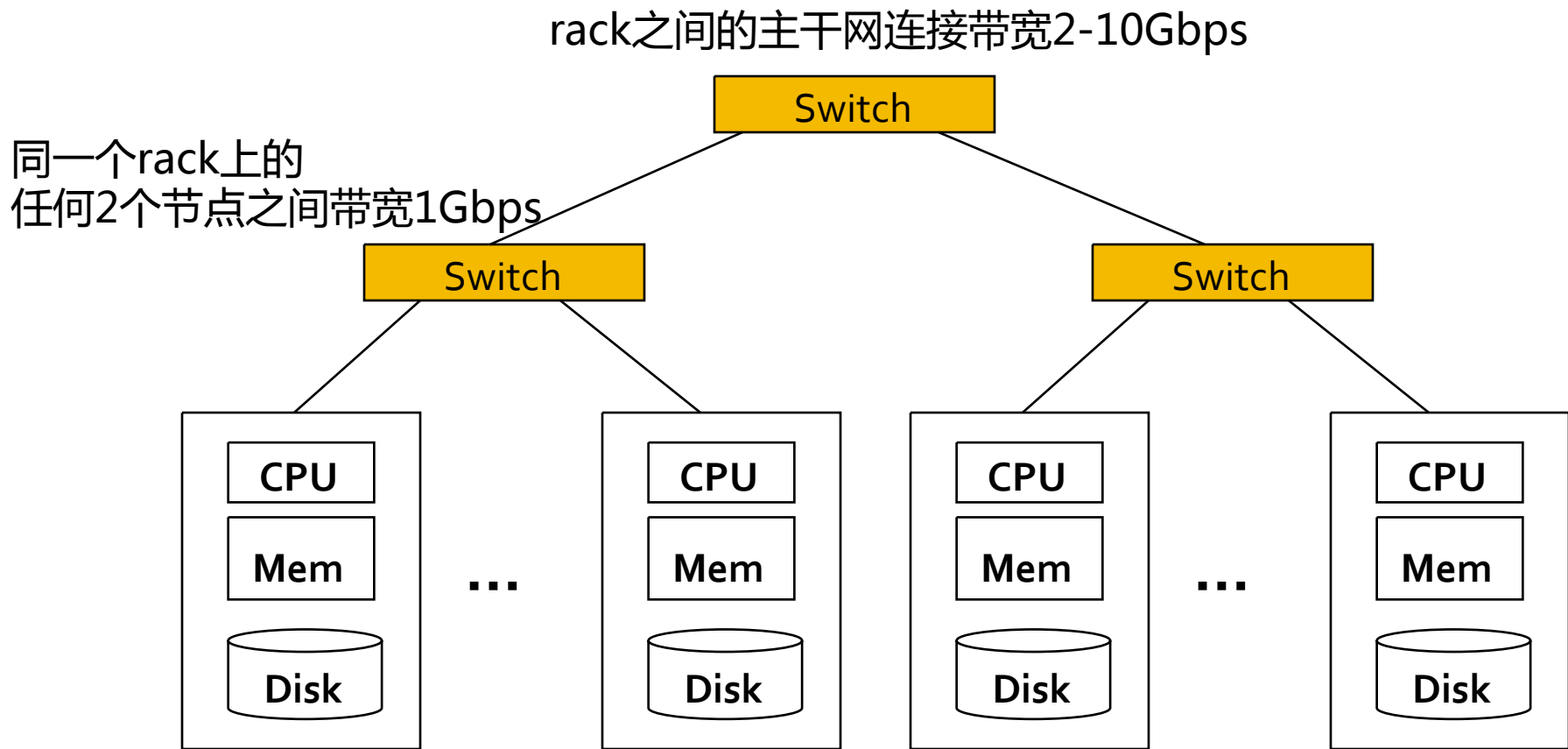
机器学习, 统计

“经典” 数据挖掘结构

# 动机：谷歌的例子

- ■ 100亿个网页
- ■ 平均网页大小 = 20KB
- ■ 100亿 \* 20KB = 200 TB
- ■ 磁盘读取带宽 = 50 MB/sec
- ■ 读取数据所需时间 = 400万秒 = 46+ 天
- ■ 后续的数据处理与操作花费的时间可能会更多

# 集群的架构



每一个rack 包含16到64个Linux 节点

2011年据统计，google约有100万台机器，详见 <http://bit.ly/Shh0RO>



# 集群计算需要面对的问题(1)

## 节点故障

- 单台服务器平均可以
- 1000台服务器的集群  $\Rightarrow$  平均故障率 1 次/天
- 100万台服务器的集群  $\Rightarrow$  平均故障率 1000 次/天

如何保持数据的持续性，  
即在某些节点故障的情形下不影响依旧能够使用数据

在运行时间较长的集群运算中，如何应对节点故障呢

# 集群计算需要面对的问题(2)

## 网络带宽瓶颈

- 网络带宽 = 1 Gbps
- 移动10TB 数据需要花费将近一天

## 分布式编程非常复杂

- 需要一个简单的模型能够隐去所有的复杂性

# Map-Reduce

## Map-Reduce集群运算时问题的解决方案

- 在多节点上冗余地存储数据，以保证数据的持续性和一直可取性
- 将计算移向数据端，以最大程度减少数据移动
- 简单的程序模型隐藏所有的复杂度



# 冗余化数据存储结构

## 分布式文件存储系统

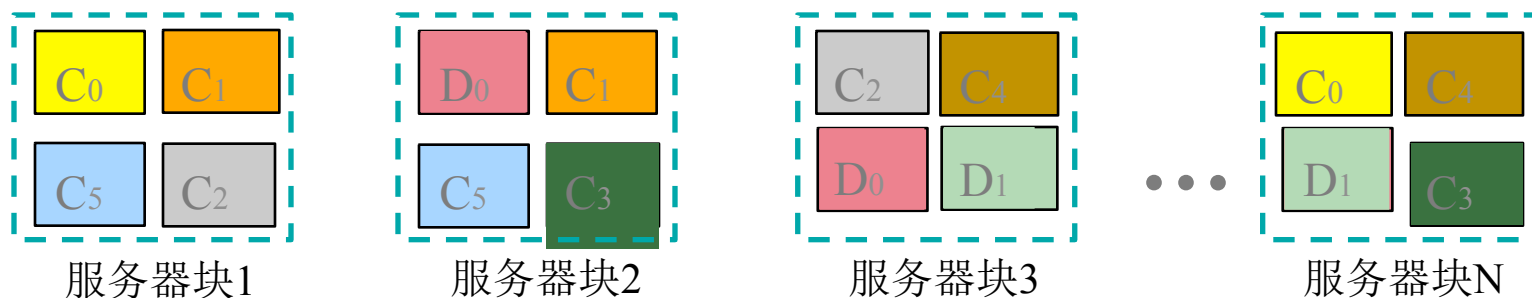
- 提供全局的文件命名空间，冗余度和可获取性
- 例如Google 的 GFS; Hadoop 的 HDFS

## 典型的应用场景与模式

- 超大级别的数据量(100GB到100TB级别)
- 数据很少就地整个被替换
- 最常见的操作为读取和追加数据

# 分布式文件系统

- 数据以“块状”形式在多台机器上存储
- 每个数据块都会重复地在多台机器上存储
- 保证数据的持续性和随时可取性



服务器块同时也用作计算服务器。

把运算挪向数据处！

# 分布式文件系统

## 服务器块

- 文件被分作16-64MB大小的连续块
- 每个文件块会被重复地存储2到3次
- 尽量保证重复的数据块在不同的机架上

## 主节点

- Hadoop的HDFS里叫做Name节点
- 存储元数据记录文件存储结构和地址
- 也可以重复

## 文件访问的客户端库

- 询问主节点以获取块服务器地址
- 直接连接相应服务器块获取数据

# Map-Reduce

## 计算模型示例

# 编程模型： Map-Reduce

## 热身练习：

- 我们手头上有个超大的文本文件
- 我们需要统计每个文本中的词，所出现的次数
- 实际应用场景
  - 从web服务器日志中找出高频热门url
  - 搜索词统计

# 练习：词频统计

## 场景1:

- 文件本身太大无法全部载入内存
- 所有的词和频次对<word, count> 可以全部载入内存

# 练习：词频统计

## 场景2:

- 所有的词和频次对<word, count> 都超出了内存大小
- linux命令

■ **words (doc.txt) | sort | uniq -c**

- 其中words命令输出一个文本内容中所有词，一个一行
- 场景2体现了MapReduce的精髓
- 好事是它是纯天然并行化的

# Map-Reduce: 总览

```
words (doc.txt) | sort | uniq -c
```

## Map

- 逐个文件逐行扫描
- 扫描的同时抽取出我们感兴趣的内容 (Keys)

## Group by key

- 排序和洗牌

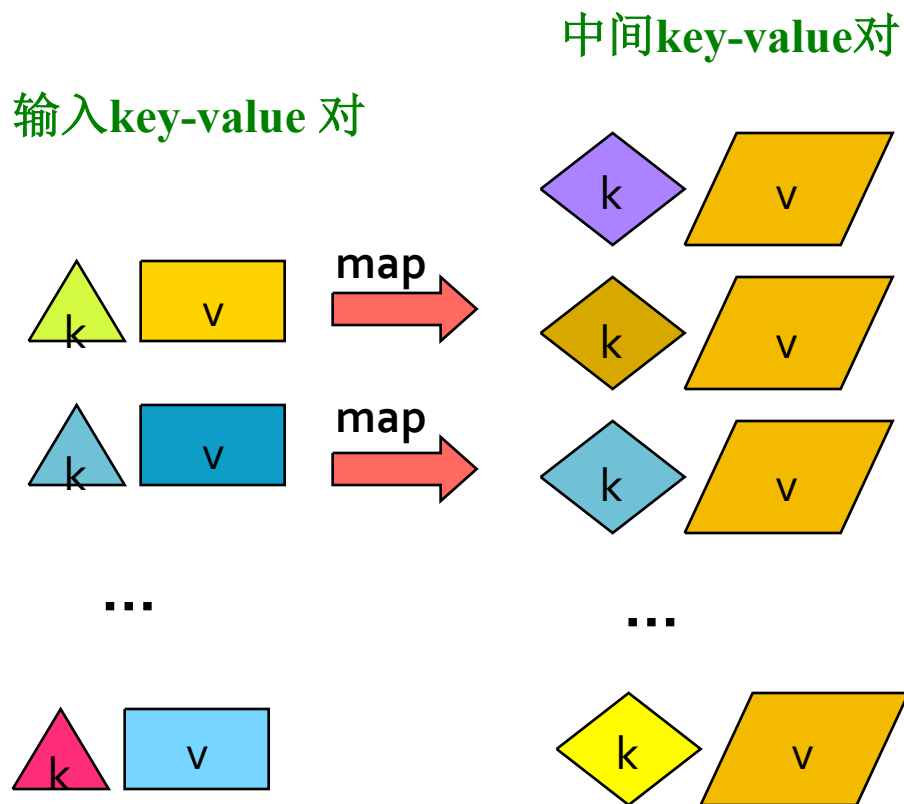
## Reduce

- 聚合、总结、过滤或转换
- 写入结果

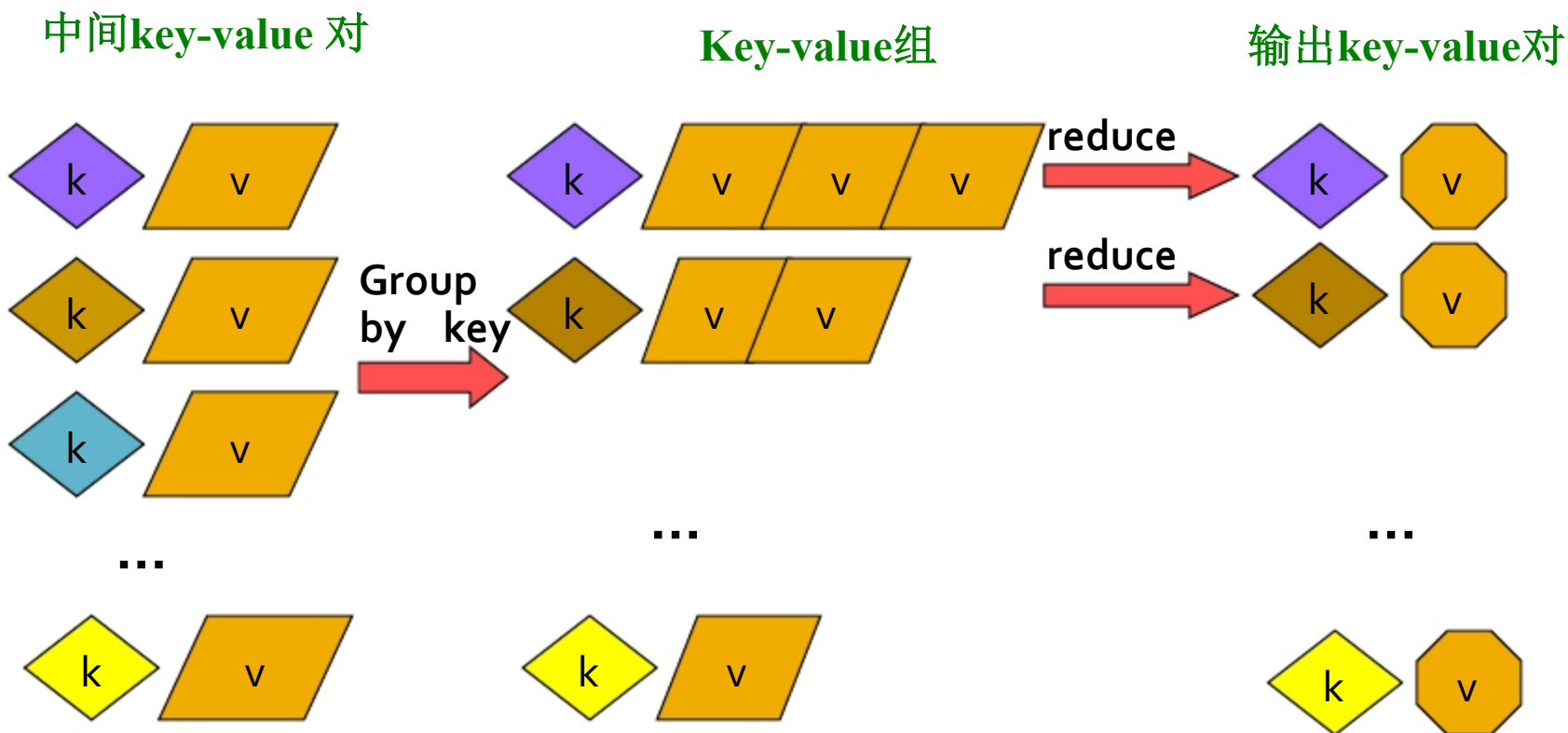
总体框架都和上述描述过程一致  
只是Map和Reduce函数要根据具体问题具体实现



# Map-Reduce: Map步骤



# Map-Reduce: Reduce步骤



# Map-Reduce步骤

输入： 一些键值对

- 编程人员需要定义两个函数：
  - $\text{Map}(k, v) \rightarrow \langle k', v' \rangle^*$ 
    - 对一个键值对输入产生一序列中间键值对
    - Map函数将对所有输入键值对操作
  - $\text{Reduce}(k', \langle v' \rangle^*) \rightarrow \langle k', v'' \rangle^*$ 
    - 所有有相同key  $k'$  的值 $v'$  被reduce到一起
    - Reduce函数对每一个不同的Key  $k'$  进行操作

# Map-Reduce: 词频统计

## Map函数

**MAP:**  
读取输入文本  
产生一系列  
键值对

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need .....

超大文本文档

(The, 1)  
(crew, 1)  
(of, 1)  
(the, 1)  
(space, 1)  
(shuttle, 1)  
(Endeavor, 1)  
(recently, 1)  
....

(key, value)

**按照key排序:**  
将所有  
有相同key的  
键值对排在一起

(crew, 1)  
(crew, 1)  
(space, 1)  
(the, 1)  
(the, 1)  
(the, 1)  
(shuttle, 1)  
(recently, 1)  
...

(key, value)

## Reduce函数

**Reduce:**  
收集和统计  
对应同一个key  
的value并输出

(crew, 2)  
(space, 1)  
(the, 3)  
(shuttle, 1)  
(recently, 1)  
...

(key, value)

序列化读取

# 使用Map-Reduce统计词频

```
map(key, value):
```

```
// key: 文档名称;           value: 文档的文本内容
```

```
    for each word w in value:
```

```
        emit(w, 1)
```

```
reduce(key, values):
```

```
// key: 一个单词;           value: 一个计数的迭代器
```

```
    result = 0
```

```
    for each count v in values:
```

```
        result += v
```

```
    emit(key, result)
```

# Map-Reduce示例2: Host大小

- 假设我们有一个大型网络语料库  
元数据文件格式如下
  - 每一条记录的格式都是: (URL, size, date, ...)
- 对于每一个**Host**, 获取其字节数(大小)

## Map

- 顺序扫描, 对每一条记录, 生成键值对  
(hostname(URL), size)

## Reduce

- 对相同host的键值对的值(字节数)进行求和

# Map-Reduce示例3：语言模型

- 即计算一个大型语料库中5词/字序列出现的次数

## Map

- 顺序扫描，从原始语料库中生成键值对：  
(5-word sequence, count)

## Reduce

- 对相同5词序列的计数(count)进行求和

# Map-Reduce

## 调度与数据流



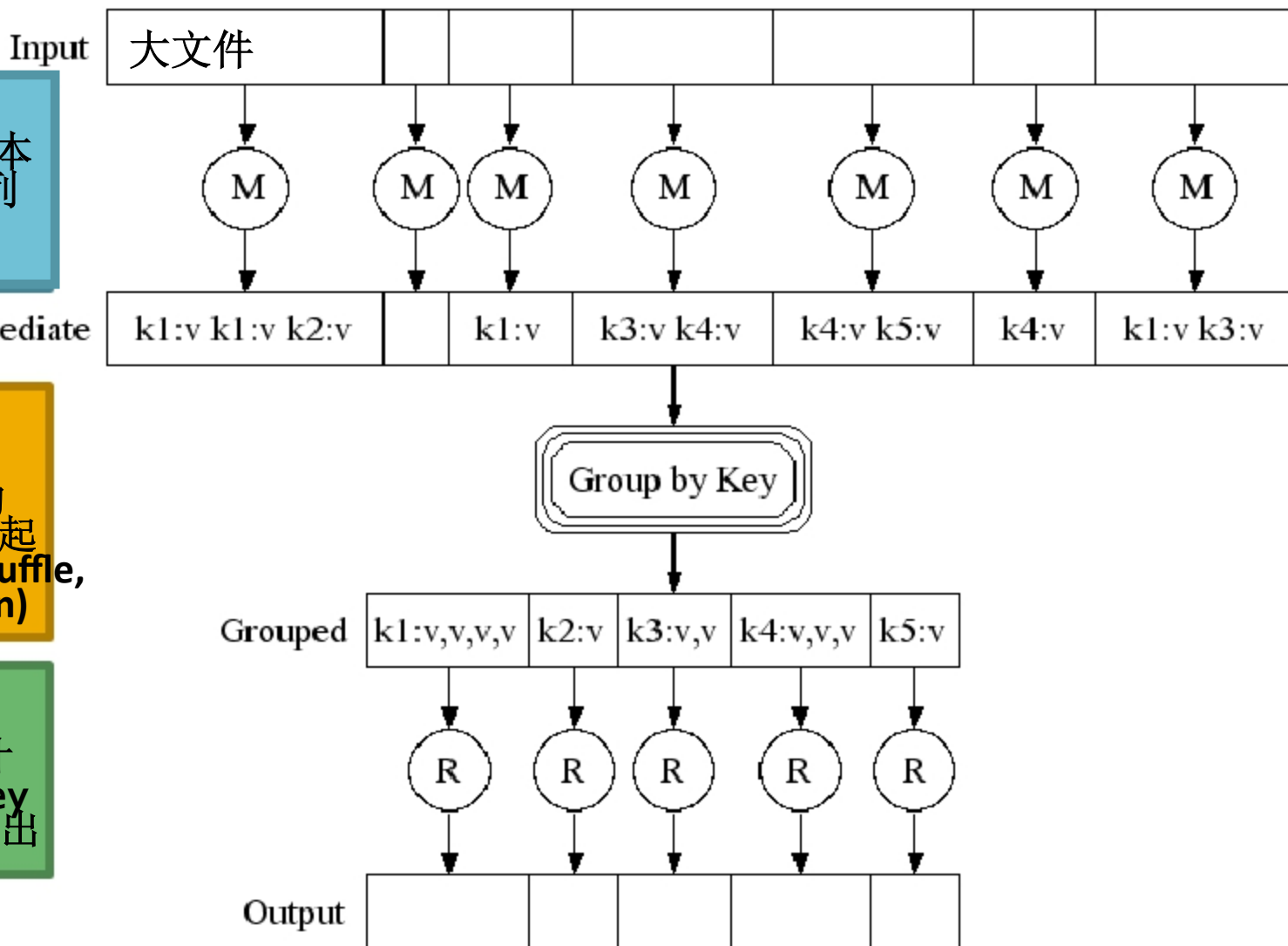
# Map-Reduce示例图:

**MAP:**  
读取输入文本  
产生一序列  
键值对

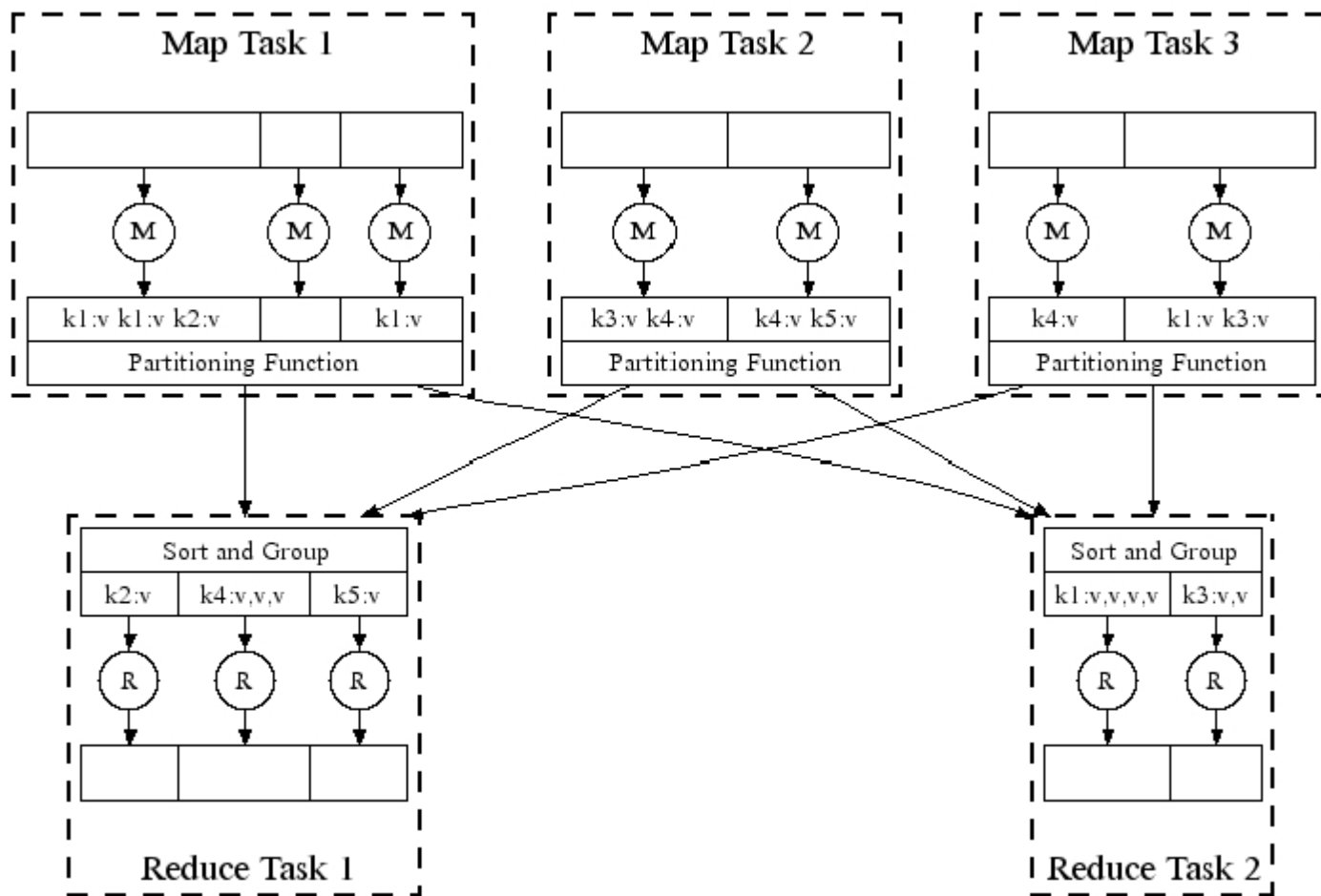
Intermediate

**按照key排序:**  
将所有  
有相同key的  
键值对排在一起  
(Hash merge, Shuffle,  
Sort, Partition)

**Reduce:**  
收集和统计  
对应同一个key  
的value并输出



# Map-Reduce: 并行化



每一个阶段都有大量同类型的工作并行进行

# Map-Reduce: 环境

## Map-Reduce 的背景环境需要负责:

- 对原始数据进行分区 (**partition**)
- 调度程序在一系列的机器集群上都并行运行
- 执行中间过程的**group by key**步骤
- 处理运行过程中的突发节点故障
- 处理并行运行过程中的节点和节点之间的通信

# 数据流

- 输入和输出都被存储在分布式文件系统(DFS)上:
  - 实际调度操作时，调度器会尽可能将map任务移至靠近数据物理存储的节点上
  - 中间结果将会被存储在Map和Reduce操作的本地文件系统上
  - 实际运行过程中，一个Map-Reduce产生的结果，很有可能作为另一个Map-Reduce任务的输入

# 主节点的协调功能

## 主节点主要负责系统的协调

- 任务状态：等待初始，进行中，完成
- 一旦有能工作的worker，待初始任务被调度运行起来
- 一个Map任务完成后，它会向主节点发送它产生的R个中间文件的位置和大小，每个文件对应一个reducer
- 主节点将这些信息传送至reducer

# 处理节点故障

## Map任务节点故障

- 所有运行中和已经完成的map任务，都被重置为待初始
- 所有这些待初始Map任务，将重新被分配到能工作的节点worker上。

## Reduce任务节点故障

- 只有运行中而未完成的reduce任务被设定为待初始。
- 这些待初始reduce任务被重新分配至其他worker上。

11

## 主节点故障

- 整个Map-Reduce任务中断，同时通知客户端管理员。

# 启动多少个Map和Reduce任务呢？

- **M个Map任务和R个Reduce任务**
- **实际操作经验法则：**
  - 通常情况下我们会让**M**远大于集群中的节点数
  - 通常设置为一个分布式文件系统块一个**Map**任务
  - 提升动态加载平衡，同时加速节点故障时的任务恢复
- **通常R比M要小**
  - 因为输出要分布在**R**个文件上

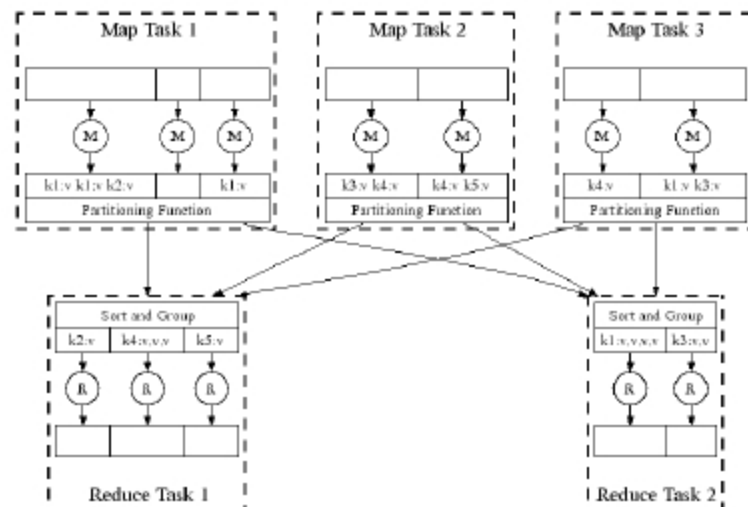
# Map-Reduce

## 改进与优化



# 改进: combiners(1)

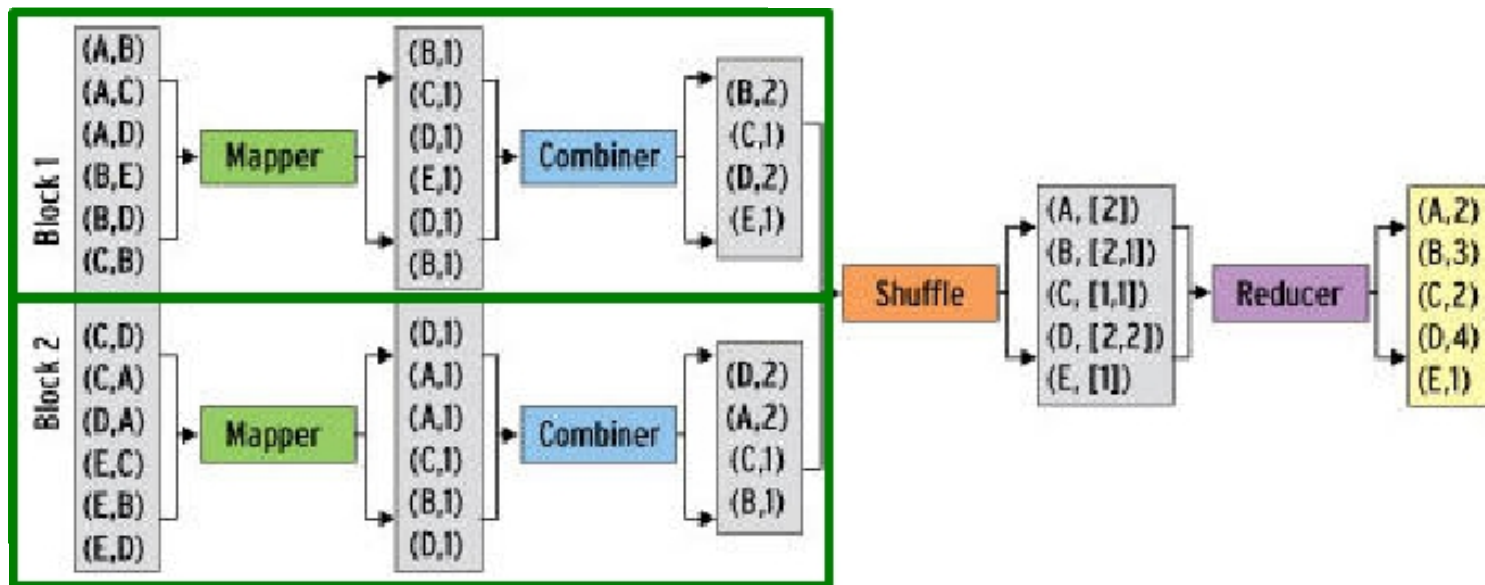
- 很多时候一个Map任务为同一个key  $k$  会产生形如 $(k, v1)$ ,  $(k, v2)$ 的键值对:
  - 例如, 词频统计任务中的高频词产生的中间结果
- 我们通过在Mapper中, 进行预聚合(pre-aggregating)操作, 来节约网络的时间成本。
  - 合并  $(k, \text{list}(v1)) \rightarrow v2$
  - 合并器(combiner)通常和reduce函数是一致的



# 改进: combiners(2)

- 以词频统计为例:

- 合并器(Combiner)预先合并了单个mapper(单个节点)中的键值对。



- 优点: 后续步骤只需要传输和重组更少的数据即可

# 改进: combiners(3)

- 注意:只有在满足**交换律**和**结合律**的条件下,  
**combiner**才能起作用
  - Sum ( 求和 )
  - Average ( 求平均 )
  - Median ( 求中间值 )

# 改进：分区函数

- 控制键值对是如何分区/划分的
  - 保证指定key对应的键值对都分配到同一个reduce的worker中
- 系统有默认的分区函数
  - $\text{hash}(\text{key}) \bmod R$
- 有时候自己指定划分分区的hash函数是很有用的
  - 例如, 定义  $\text{hash}(\text{hostname}(\text{URL})) \bmod R$  可以确保所有同一个host的URL在同一个输出文件

# Map-Reduce系统

- **Google MapReduce**

- 使用google文件系统(GFS)用于稳定存储
- google之外不开放使用

- **Hadoop**

- 使用java实现的Map-Reduce开源框架
- 使用HDFS进行稳定存储
- 可以在<http://lucene.apache.org/hadoop/>下载使用

- **Hive, Pig**

- 在Hadoop的Map-Reduce之上提供的类SQL数据提取操作功能

# 云计算

- 按小时计费的租赁计算系统
  - 可能会提供额外的服务：例如持续性存储
- 例如，亚马逊提供的“Elastic Compute Cloud” (EC2)
  - S3 (稳定性存储)
  - Elastic Map Reduce (EMR)

# 进一步阅读与相关资源

# 阅读

- Jeffrey Dean and Sanjay Ghemawat:  
MapReduce: Simplified Data Processing on  
Large Clusters  
■ <http://labs.google.com/papers/mapreduce.html>
- Sanjay Ghemawat, Howard Gobioff, and  
Shun-Tak Leung: The Google File System  
■ <http://labs.google.com/papers/gfs.html>



# 资源

## ■ Hadoop Wiki

- Introduction

- <http://wiki.apache.org/lucene-hadoop/>

- Getting Started

- <http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop>

- Map/Reduce Overview

- <http://wiki.apache.org/lucene-hadoop/HadoopMapReduce>

- <http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses>

- Eclipse Environment

- <http://wiki.apache.org/lucene-hadoop/EclipseEnvironment>

## ■ Javadoc

- <http://lucene.apache.org/hadoop/docs/api/>

# 资源

- Releases from Apache download mirrors
  - <http://www.apache.org/dyn/closer.cgi/lucene/hadoop>
- Nightly builds of source
  - <http://people.apache.org/dist/lucene/hadoop/nightly>
- Source code from subversion
  - [http://lucene.apache.org/hadoop/version\\_control.html](http://lucene.apache.org/hadoop/version_control.html)