



# 云计算 (第三版)

CLOUD COMPUTING Third Edition

## 第 2 章

# Google云计算原理与应用 (一)

# 目录

2.1 Google文件系统GFS

2.2 分布式数据处理MapReduce

2.3 分布式锁服务Chubby

2.4 分布式结构化数据表Bigtable


2.5 分布式存储系统Megastore

2.6 大规模分布式系统的监控基础架构Dapper

2.7 海量数据的交互式分析工具Dremel

2.8 内存大数据分析系统PowerDrill

2.9 Google应用程序引擎

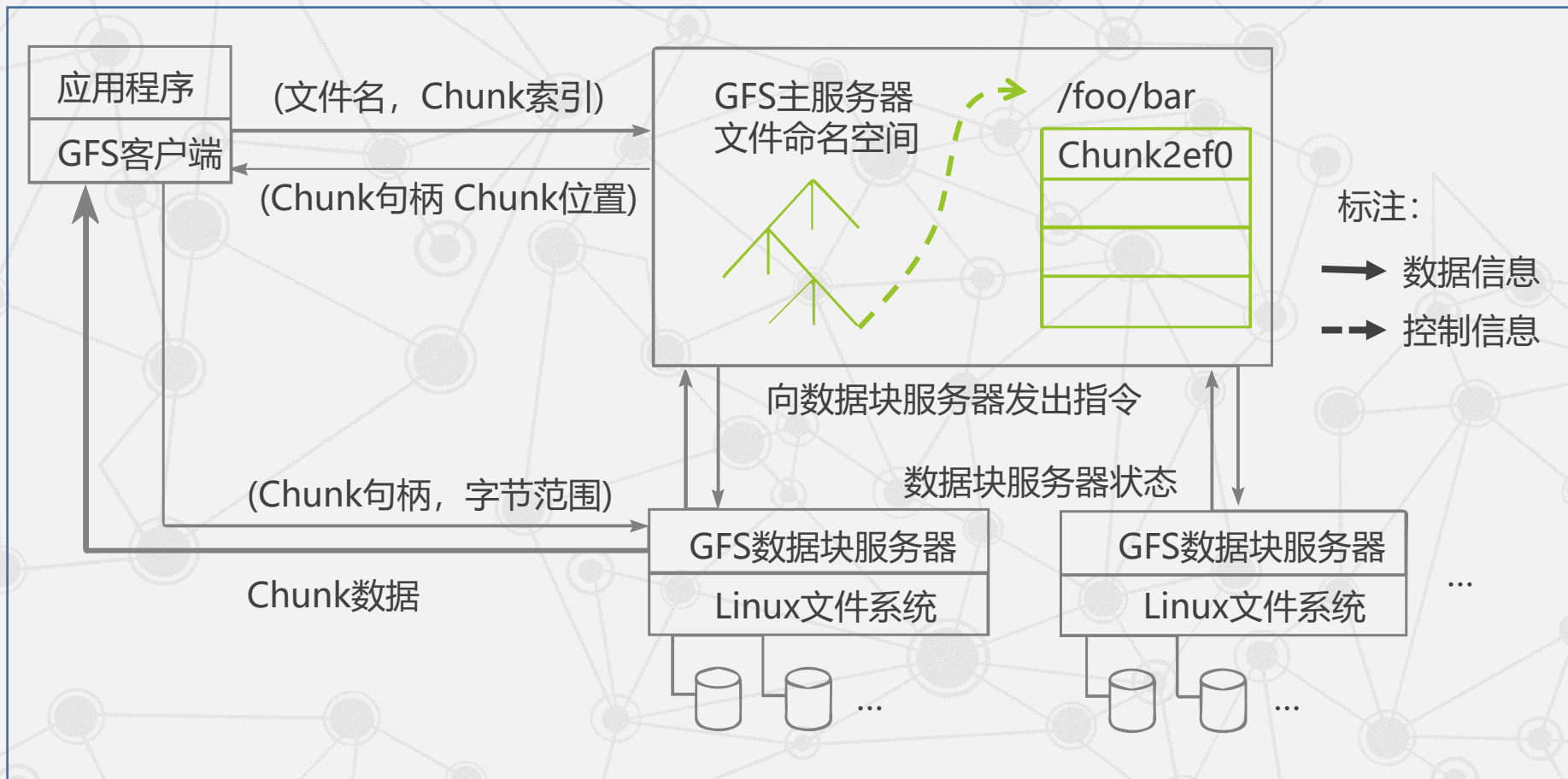


全球最大搜索引擎、Google Maps、Google Earth、Gmail、YouTube等。这些应用的共性在于数据量巨大，且要面向全球用户提供实时服务。

## 2.1 Google文件系统GFS

- ▶ 2.1.1 系统架构
- 2.1.2 容错机制
- 2.1.3 系统管理技术

## ● GFS的系统架构





- GFS将整个系统节点分为三类角色



### ● GFS的实现机制

- 客户端首先访问Master节点，获取交互的Chunk Server信息，然后访问这些Chunk Server，完成数据存取工作。这种设计方法实现了控制流和数据流的分离。
- Client与Master之间只有控制流，而无数据流，极大地降低了Master的负载。
- Client与Chunk Server之间直接传输数据流，同时由于文件被分成多个Chunk进行分布式存储，Client可以同时访问多个Chunk Server，从而使得整个系统的I/O高度并行，系统整体性能得到提高。

- GFS的特点

# 1

## 采用中心服务器模式

- 可以方便地增加Chunk Server
- Master掌握系统内所有Chunk Server的情况，方便进行负载均衡
- 不存在元数据的一致性问题



- GFS的特点

## 2 不缓存数据

- 文件操作大部分是流式读写，不存在大量重复读写，使用Cache对性能提高不大
- Chunk Server上数据存取使用本地文件系统从可行性看，Cache与实际数据的一致性维护也极其复杂

- GFS的特点

# 3

## 在用户态下实现

- 利用POSIX编程接口存取数据降低了实现难度，提高通用性
- POSIX接口提供功能更丰富
- 用户态下有多种调试工具
- Master和Chunk Server都以进程方式运行，单个进程不影响整个操作系统
- GFS和操作系统运行在不同的空间，两者耦合性降低

## 2.1 Google文件系统GFS

2.1.1 系统架构

► 2.1.2 容错机制

2.1.3 系统管理技术

### • Master容错

Master

命名空间 (Name Space) , 也就是整个文件系统的目录结构。

Chunk与文件名的映射表。

Chunk副本的位置信息, 每一个Chunk默认有三个副本。

日志

直接保存在各个  
Chunk Server上

当Master发生故障时, 在磁盘数据保存完好的情况下, 可以迅速恢复以上元数据

为了防止Master彻底死机的情况, GFS还提供了Master远程的实时备份

### ● Chunk Server容错

GFS采用副本的方式实现Chunk Server的容错

每一个Chunk有多个存储副本（默认为三个）

对于每一个Chunk，必须将所有的副本全部写入成功，才视为成功写入

相关的副本出现丢失或不可恢复等情况，Master自动将该副本复制到其他Chunk Server

GFS中的每一个文件被划分成多个Chunk，Chunk的默认大小是**64MB**

每一个Chunk以Block为单位进行划分，大小为64KB，每一个Block对应一个32bit的校验和

## 2.1 Google文件系统GFS

2.1.1 系统架构

2.1.2 容错机制

► 2.1.3 系统管理技术



### ● 系统管理技术

#### 系统 管理技术

GFS集群中通常有非常多的节点，需要相应的技术支撑

大规模集群安装技术

故障检测技术

GFS构建在不可靠廉价计算机之上的文件系统，由于节点数目众多，故障发生十分频繁

新的Chunk Server加入时，只需裸机加入，大大减少GFS维护工作量

节点动态加入技术

节能技术

Google采用了多种机制降低服务器能耗，如采用蓄电池代替昂贵的UPS

# 目录

2.1 Google文件系统GFS

2.2 分布式数据处理MapReduce

2.3 分布式锁服务Chubby

2.4 分布式结构化数据表Bigtable

2.5 分布式存储系统Megastore

2.6 大规模分布式系统的监控基础架构Dapper

2.7 海量数据的交互式分析工具Dremel

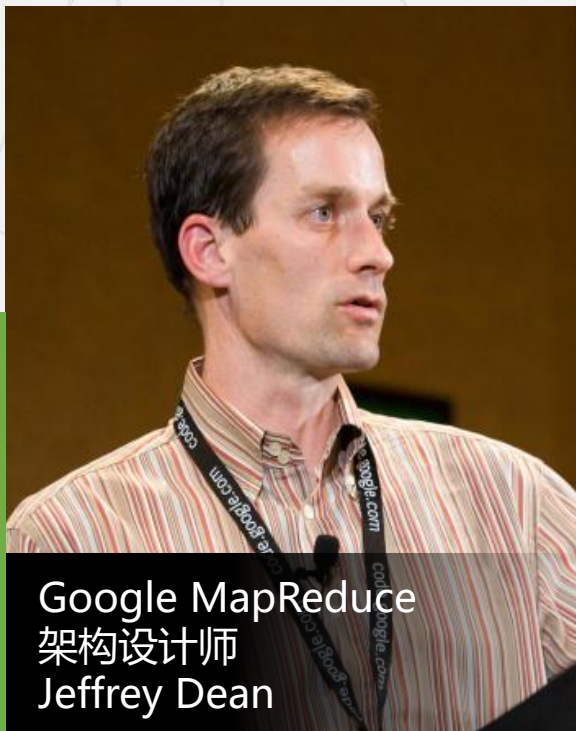
2.8 内存大数据分析系统PowerDrill

2.9 Google应用程序引擎

## 2.2 分布式数据处理MapReduce

- ▶ 2.2.1 产生背景
- 2.2.2 编程模型
- 2.2.3 实现机制
- 2.2.4 案例分析

- 产生背景



Jeffrey Dean设计一个新的抽象模型，封装并行处理、容错处理、本地化计算、负载均衡的细节，还提供了一个简单而强大的接口。

**这就是MapReduce**

### ● 产生背景

MapReduce这种**并行编程模式**思想最早是在**1995年**提出的。

与传统的分布式程序设计相比，MapReduce封装了**并行处理、容错处理、本地化计算、负载均衡**等细节，还提供了一个**简单而强大的接口**。

MapReduce把对数据集的大规模操作，**分发给一个主节点管理下的各分节点**共同完成，通过这种方式**实现任务的可靠执行与容错机制**。

## 2.2分布式数据处理MapReduce

2.2.1 产生背景

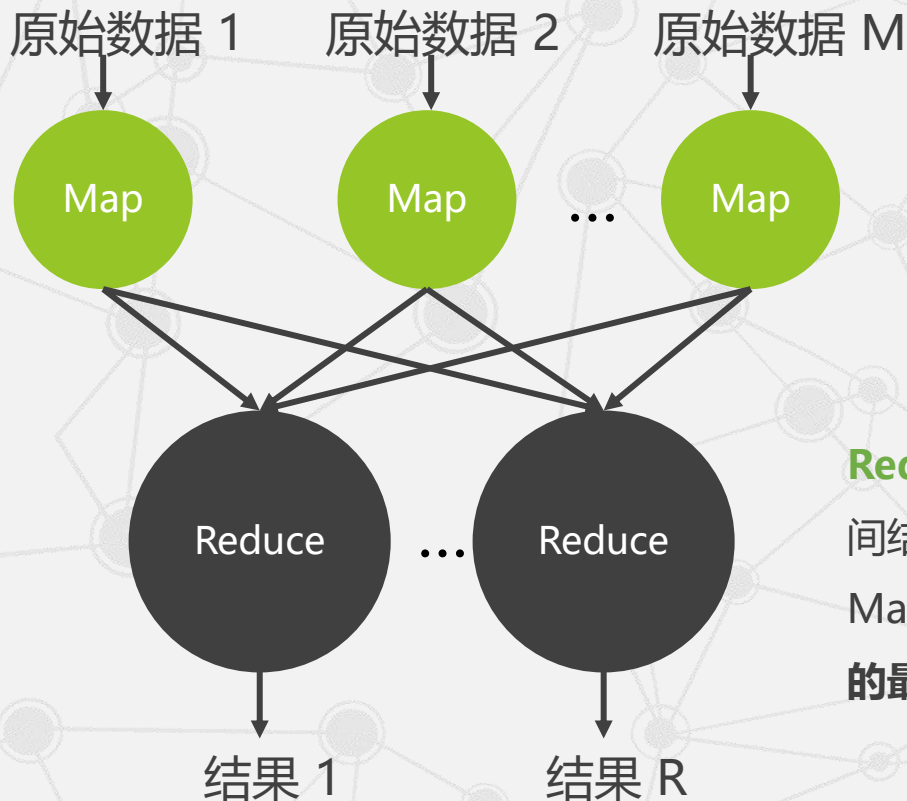
► 2.2.2 编程模型

2.2.3 实现机制

2.2.4 案例分析



### ● 编程模型



**Map函数**——对一部分原始数据进行指定的操作。每个Map操作都针对不同的原始数据，因此Map与Map之间是互相独立的，这使得它们可以充分并行化。

**Reduce操作**——对每个Map所产生的一部分中间结果进行**合并操作**，每个Reduce所处理的Map中间结果是互不交叉的，所有Reduce产生的最终结果经过简单连接就形成了完整的结果集。

### ● 编程模型

Map:  $(in\_key, in\_value) \rightarrow \{(key_j, value_j) \mid j = 1 \dots k\}$

Reduce:  $(key, [value_1, \dots, value_m]) \rightarrow (key, final\_value)$

**Map输入参数:** in\_key和 in\_value, 它指明了Map需要处理的原始数据

**Map输出结果:** 一组  $\langle key, value \rangle$  对, 这是经过Map操作后所产生的中间结果

**Reduce输入参数:**  $(key, [value_1, \dots, value_m])$

**Reduce工作:**  
对这些对应相同key的value值进行归并处理

**Reduce输出结果:**  
 $(key, final\_value)$ , 所有Reduce的结果并在一起就是最终结果

## 2.2 分布式数据处理MapReduce

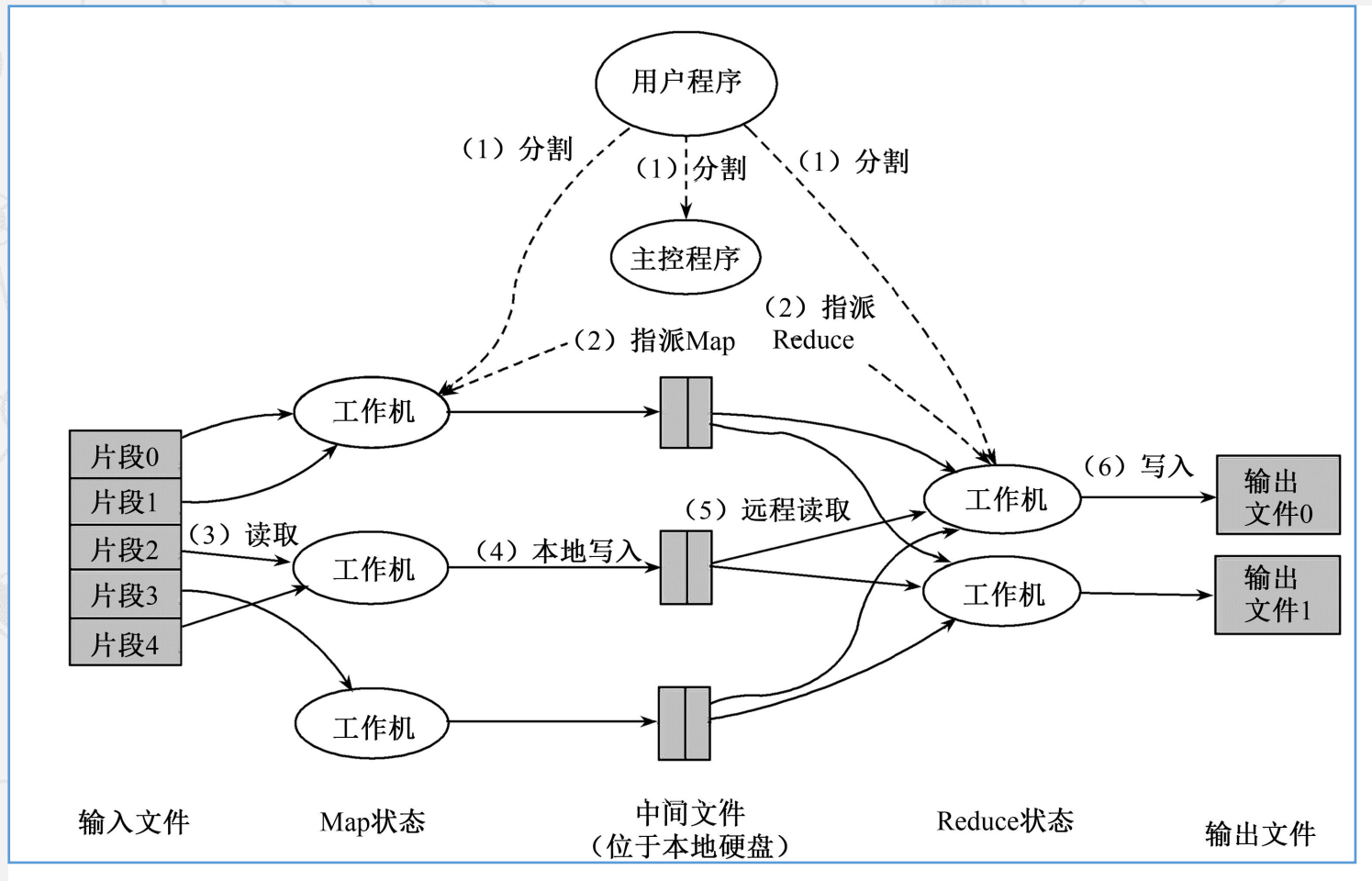
2.2.1 产生背景

2.2.2 编程模型

► 2.2.3 实现机制

2.2.4 案例分析

### ● 实现机制



### ● 实现机制

- (1) MapReduce函数首先把**输入文件分成M块**
- (2) 分派的执行程序中有有一个**主控程序Master**
- (3) 一个被分配了Map任务的Worker读取并处理相关的输入块
- (4) 这些缓冲到内存的中间结果将被定时写到本地硬盘，这些**数据通过分区函数分成R个区**
- (5) 当Master通知执行Reduce的Worker关于中间<key,value>对的位置时，它调用远程过程，从Map Worker的本地硬盘上读取缓冲的中间数据
- (6) Reduce Worker根据每一个唯一中间key来遍历所有的排序后的中间数据，并且把key和相关的中间结果值集合传递给用户定义的Reduce函数
- (7) 当所有的Map任务和Reduce任务都完成的时候，Master激活用户程序

### ● 容错机制

由于MapReduce在成百上千台机器上处理海量数据，所以容错机制是不可或缺的。总的来说，MapReduce通过**重新执行失效的地方**来实现容错。

#### Master失效

Master会周期性地设置检查点（checkpoint），并导出Master的数据。一旦某个任务失效，系统就从最近的一个检查点恢复并重新执行。

由于只有一个Master在运行，如果Master失效了，则只能终止整个MapReduce程序的运行并重新开始。

#### Worker失效

Master会**周期性地给Worker发送ping**命令，如果没有Worker的应答，则Master认为Worker失效，终止对这个Worker的任务调度，把失效Worker的任务调度到其他Worker上重新执行。



## 2.2 分布式数据处理MapReduce

2.2.1 产生背景

2.2.2 编程模型

2.2.3 实现机制

► 2.2.4 案例分析



**怎样通过MapReduce完成排序工作，  
使其有序（字典序）呢？**

### ● 第一个步骤

对原始的数据进行分割 (Split) ,  
得到N个不同的数据分块。

Split1:      nkklacdd  
              gfgdfdfdf  
              .....  
              annnbnbvgh

Split2:      dfgmdhjd  
              kghfgcxnkil  
              .....  
              gjghyotewgbb

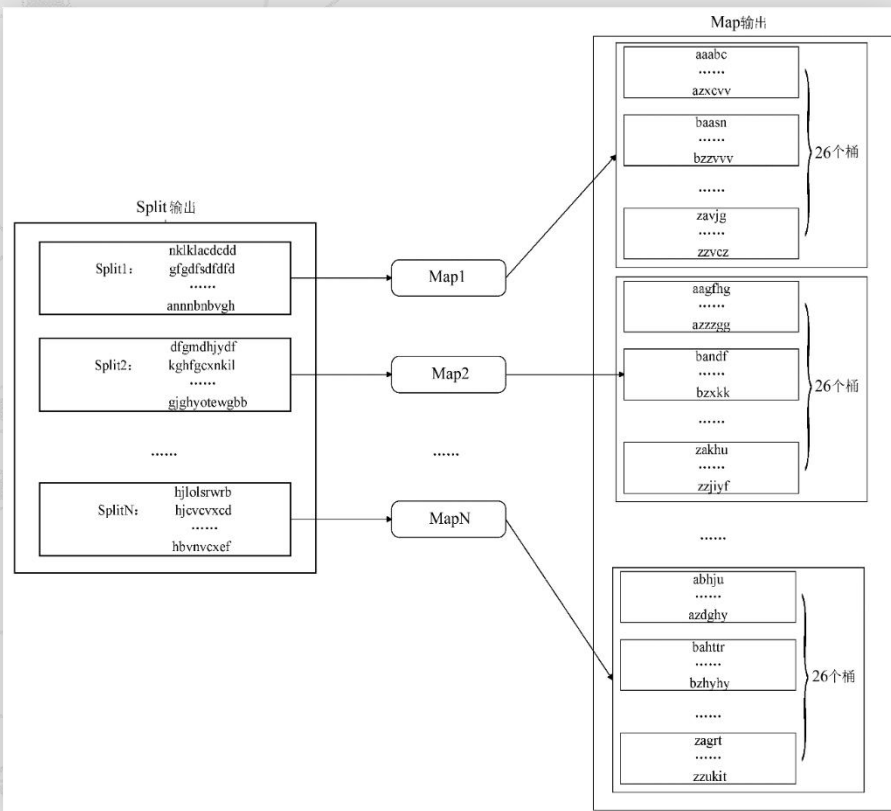
.....

SplitN:      hjlolsrwr  
              hjevexcd  
              .....  
              hbnvxcxef

### ● 第二个步骤

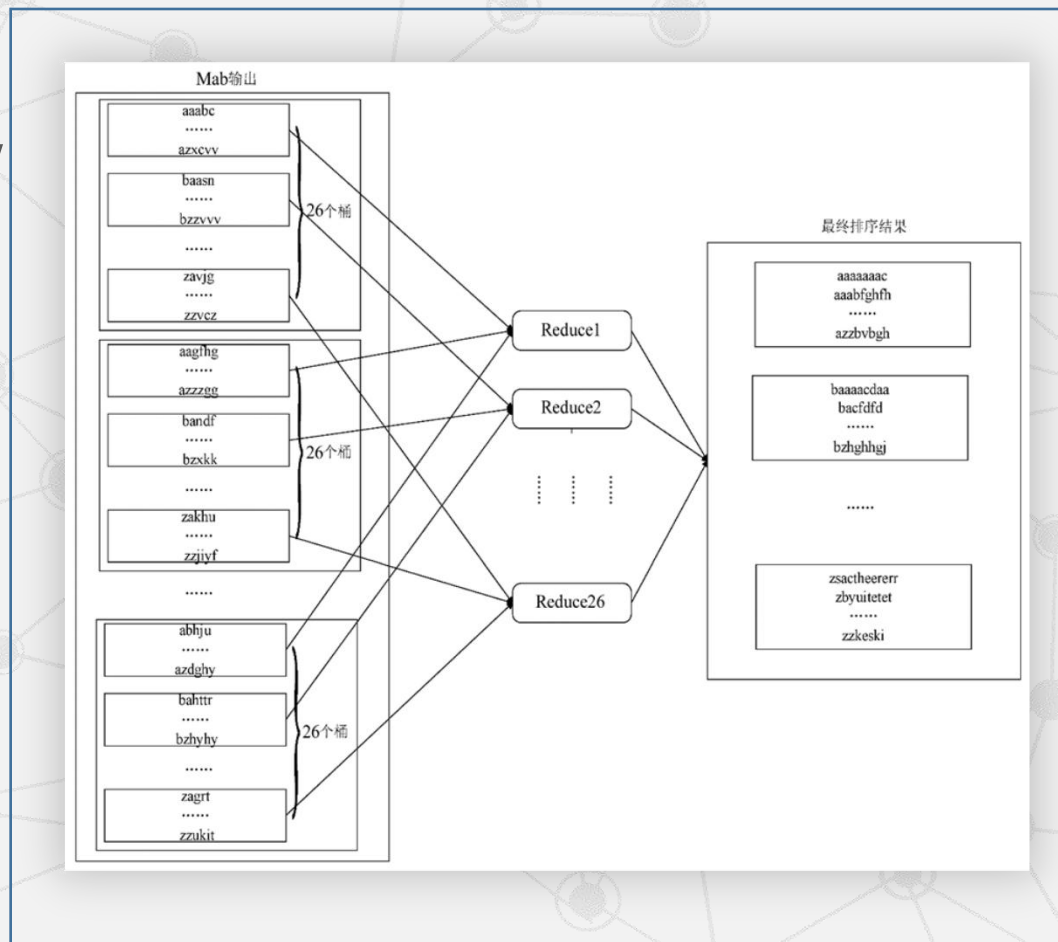
对每一个数据分块都启动一个 Map 进行处理。

采用桶排序的方法，**每个 Map 中按照首字母将字符串分配到 26 个不同的桶中。**



### • 第三个步骤

对于Map之后得到的中间结果，启动26个Reduce。  
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。





**本章未完待续**





# 云计算 (第三版)

CLOUD COMPUTING Third Edition

## 第2章

# 谢谢观看