MapReduce 单

击，编辑主字幕样式

大数据分析|何铁科
http: // hetieke.cn

南京大学
南京大学

Mapreduce的目标

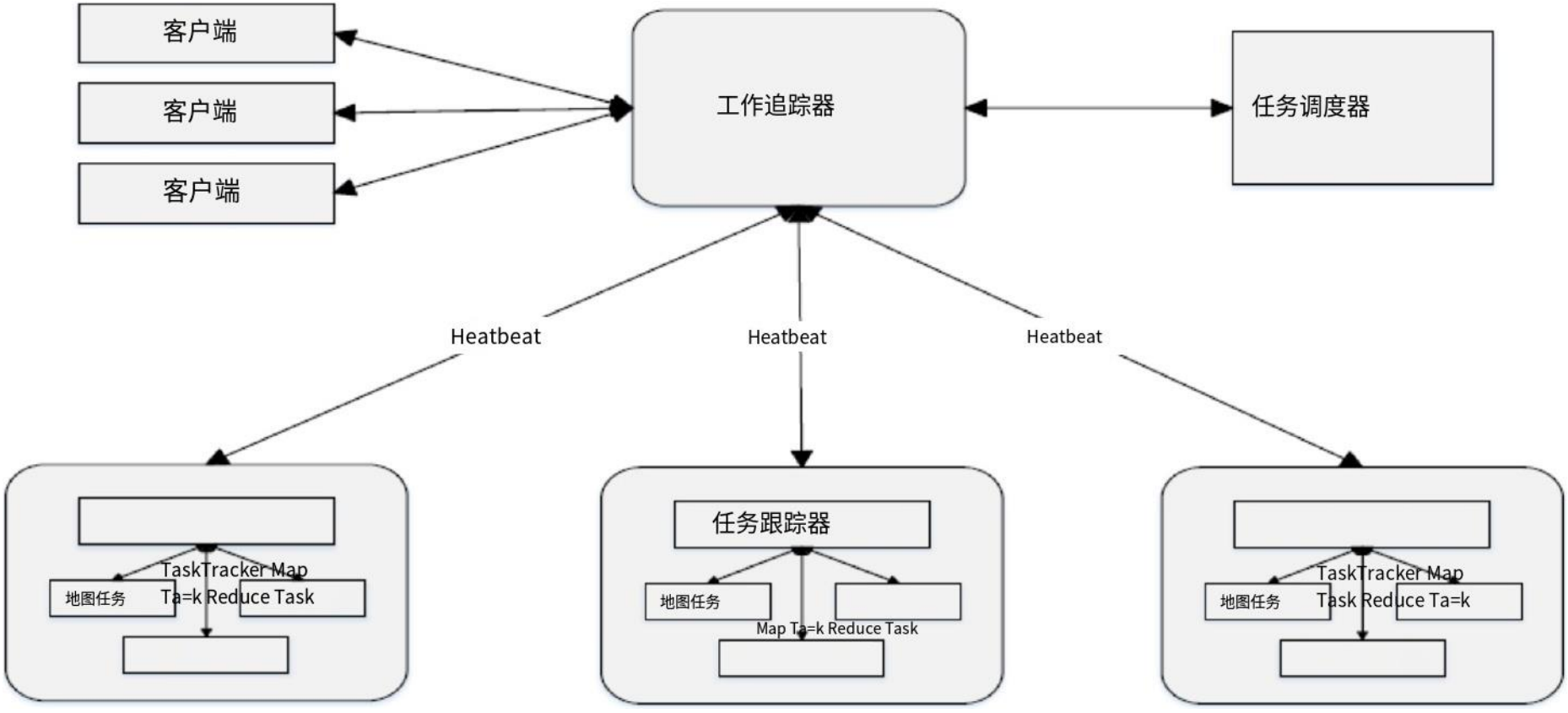
1.并行化
2.容错
3.数据分布
4.负载平衡

动机

q 需要计算大规模数据

q 隐藏了图书馆的细节

q 并行化、容忍度、分布和负载平衡 q 受 **Lisp** 中 **map** 和 **reduce** 原语的启发


客户端
客户端
客户端
工作追踪器
任务调度器
Heatbeat
Heatbeat
Heatbeat
TaskTracker Map
地图任务
Ta=k Reduce Task
任务跟踪器
地图任务
Map Ta=k Reduce Task
TaskTracker Map
地图任务
Task Reduce Ta=k

主要组件

q 客户端 q
JobTracker q
TaskTracker q
任务

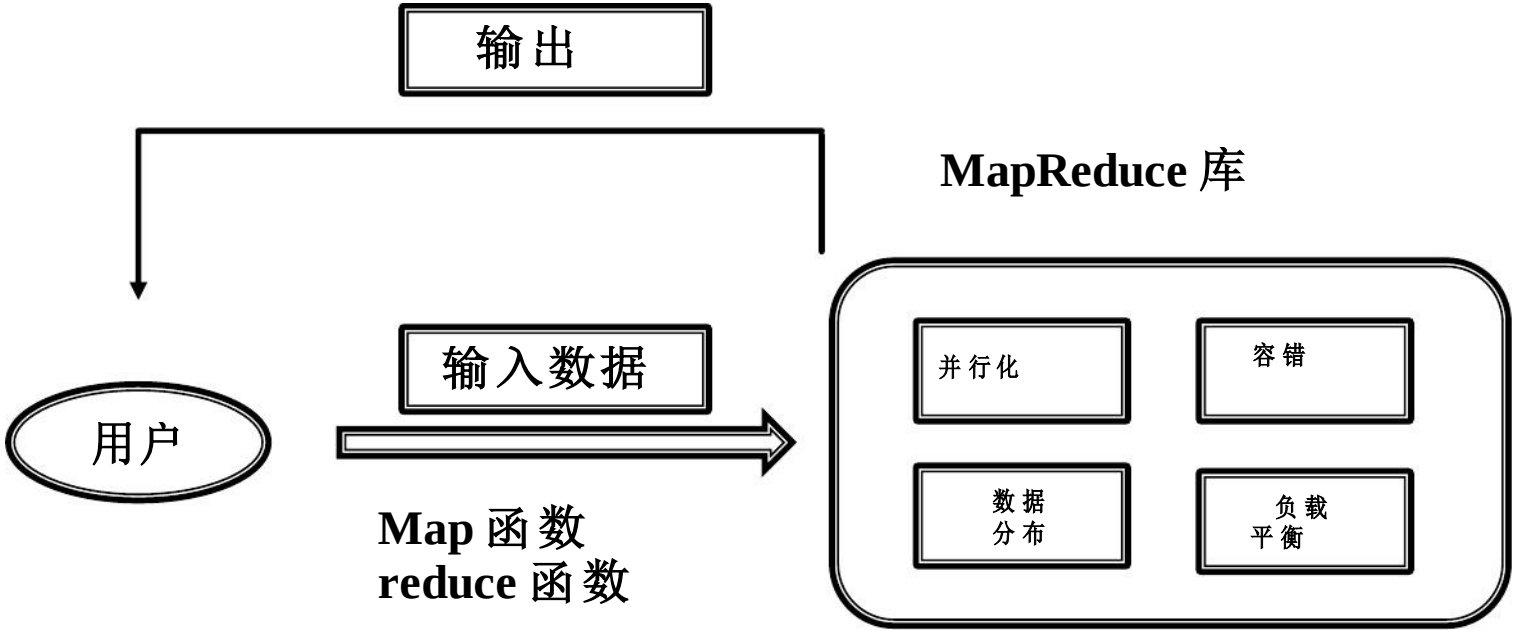
1日计算

1.输入数据通常很大。

2.计算必须分布在数百台机器上。

3.模糊了原始的简单计算。

2日隐秘



编程模型

- **MapReduce** 是谷歌提出的用于处理和生成大型数据集的**编程模型**。

该框架包含两个用户实现的接口:**Map 和 Reduce**。

- Map 接收一个键值对并生成一个中间键值对的集合。

- Reduce 接收这个中间键和键值的集合，将这些值合并在一起，并产生一个更小的值集合。

例子

计算每个单词在大量文档集合中的出现次数的问题。

```
映射(字符串键，字符串值)
    // key:文档名称
    // value:文档内容
    对于每个单词w in value:
        EmitIntermediate (w, " 1 ");


reduce(String key, Iterator values): // key:一个单词

    // values:一个计数列表
    Int result = 0;
    对于values中的每个v:
        result += ParseInt (v);
    Emit (AsString (result));
```

类型

地图减	(F1，V1)	→ List (k2,v2) List
少	(k2,列表(v2))	→ (v2)

- 输入的键和值与输出的键和值来自不同的域。

- 中间的键和值与输出的键和值来自同一个域。

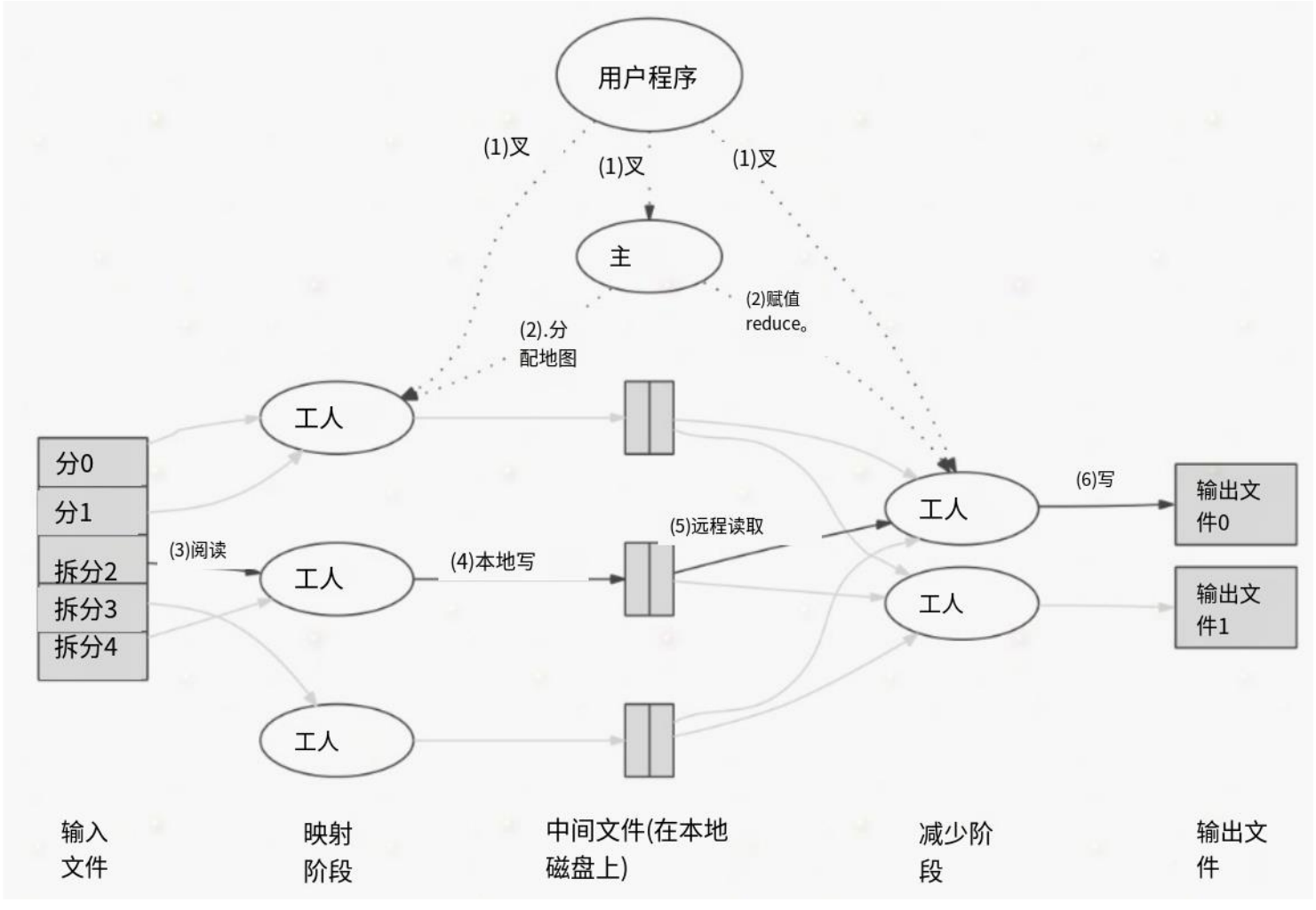
更多的例子

1.分布式 Grep
2.URL 访问频率的计数
3.反向网络链接图
4.每个主机的词向量
5.反向索引
6. 分布式排序

整个设计

1. 执行概述主数据结构容
2. 错局部性
3. 任务粒度备份任务
4.
5.
6.

1. 执行概述


用户程序
(1)叉
(1)叉
(1)叉
主
(2)赋值
reduce。
(2).分
配地图
工人
分0
分1
拆分2
拆分3
拆分4
(3)阅读
工人
(4)本地写
工人
(5)远程读取
工人
工人
(6)写
输出文
件0
输出文
件1
输入
文件
映射
阶段
中间文件(在本地
磁盘上)
减少阶
段
输出文
件

2. 掌握数据结构

- Map 和 Reduce 任务
  - 状态(空闲、进行中、完成)
  - 工人机器的身份

Master 是中间文件区域的位置从 map 任务传播到 reduce 任务的管道

3.容错

- 工人失败
  - 在这个 worker 上完成的所有 map 任务都被重置为空闲状态，并移交给其他 worker 来执行这些 map 任务。
  - 正在这个 worker 上执行的 *MAP* 任务或 reduce 任务被重置到 idle 状态，等待重调度。
- 主服务器失败
  - 当前实现选择中断 MapReduce 计算。

4. 位置

- 输入数据保存在本地硬盘。
- GFS 将每个文件分割为大小为 64MB 的块，
- GFS 保存每个块的多个副本(通常在不同的机器上有 3 个副本)。
- Master 将尝试在包含相关输入数据副本的机器上执行 map 任务。
- 如果任务失败，master 将尝试通过在相邻机器上执行 map 任务来节省网络带宽。

5. 任务粒度

- **Map phase into M pieces, reduce phase into R pieces。**
- **理想情况下，M 和 R 应该远大于 worker machines 的数量，以改善动态负载平衡。**
- **实际上，M 和 R 在实际实施中有限制。**
- **Master 必须做出 O(M + R)调度决策，并在内存中保持 O(M* R)状态。**

6. 备份任务

- 这种模式旨在缓解**掉队者问题。**

- **掉队问题**-机器需要异常大量的时间来完成最后几张地图，导致较长的计算时间。

- **解决方案**

  当一个 *MapReduce* **即将完成时，master 为正在**执行的任务执行一个备用任务。

  - 当任务完成时标记 task 为 complete，无论主任务还是备用任务完成。

细化

- 分区函数
- 排序保证
- **Combiner** 函数
- 输入和输出类型
- 副作用
- 跳过不良记录
- 本地执行
- 状态信息
- 计数器

1. 分区功能

指定所需的 reduce 任务/输出文件的数量
使用哈希（例如;“ hash(key) mod R”）
•得到相当平衡的分区

2. 订购担保

- 在给定的分区内，中间的键值对按键递增顺序处理
- 易于生成每个分区的有序输出文件

3.组合器函数

允许用户指定一个可选的 **Combiner** 函数，在数据通过网络发送之前对其进行部分合并

- 在执行 **map** 任务的每台机器上执行
- 与 **reduce** 函数相同的代码
- reduce 函数和 **combiner** 函数之间唯一的区别是如何处理函数的输出
  - **reduce** 函数:写入最终的输出文件
  - **combiner** 函数:写入一个中间文件，该文件将被发送给 **reduce** 任务

4. 输入和输出类型

- 支持读取几种不同格式的输入数据
- 将输入分割成有意义的范围，作为单独的 map 任务处理
- 通过提供一个简单阅读器接口的实现来增加对新的输入类型的支持

reader 不需要提供从文件中读取的数据

支持一组输出类型以产生不同格式的数据

5. 副作用

方便生成辅助文件作为 map 和/或 reduce 操作符的额外输出

依赖应用程序编写人员使这些副作用具有原子性和幂等性

- 应用程序写入一个临时文件，并在该文件完全生成后原子性地重命名该文件
- 不支持由单个任务生成的多个输出文件的原子两阶段提交
- 产生具有跨文件一致性要求的多个输出文件的任务应该是确定的
- 这一限制在实践中从未成为问题

6. 跳过坏唱片

- 可以忽略一些不良记录
- 提供一种可选的执行模式，其中 MapReduce 库检测哪些记录导致确定性崩溃，并跳过这些记录以向前推进

每个工作进程安装一个信号处理程序，用于捕获分段违规和总线错误

- MapReduce 库将参数的序列号存储在一个全局变量中
- 如果用户代码生成一个信号，信号处理程序发送一个“最后喘息”UDP 数据包，其中包含序列号到 MapReduce 主机。
- 当 master 在某条记录上看到多个失败时，这意味着该记录应该被跳过

7. 本地执行

- 当实际计算发生在分布式系统中时，Map 或 Reduce 函数的调试问题可能会很棘手
- 开发 MapReduce 库的替代实现，在本地机器上顺序执行 MapReduce 操作的所有工作

8. 分区功能

显示计算的进度

- 包含到每个任务生成的标准错误和标准输出文件的链接
- 可以使用这些数据来预测计算需要多长时间，以及是否应该向计算中添加更多的资源

用于判断何时计算比预期慢得多

- 顶级状态页面显示哪些 worker 失败了，以及它们失败时正在处理哪些 map 和 reduce 任务

9. 计数器

提供计数器设施来计算各种事件的发生次数

要使用此功能，用户代码创建一个命名计数器对象，然后在 Map 和/或 Reduce 函数中适当地增加计数器

- 消除重复执行相同 map 或 reduce 任务的影响，以避免重复计算
- 部分计数器值由 MapReduce 库自动维护

用于全面检查 MapReduce 操作的行为

表演

- 集群配置

Grep(全局搜索正则表达式并打印)

- 排序
- 备份任务的效果
- 机器故障

1. 集群配置

在由大约 1800 台机器组成的集群上执行

所有的机器:

2 个 2GHz 的 Intel Xeon 处理器，支持超线程

4GB 内存，两个 160GB IDE 磁盘，一个千兆以太网链路

- 布置在一个两级的树形交换网络中，在根处有大约 100-200 Gbps 的聚合带宽
- 在相同的托管设施中

大约预留了 1-1.5GB

- 在周末下午执行

2. 格雷普

- 扫描 10^10 个 100 字节的记录，搜索一个相对罕见的三字符模式
- 分割成大约 64MB 的片段(M = 15000)，整个输出放在一个文件中(R = 1)

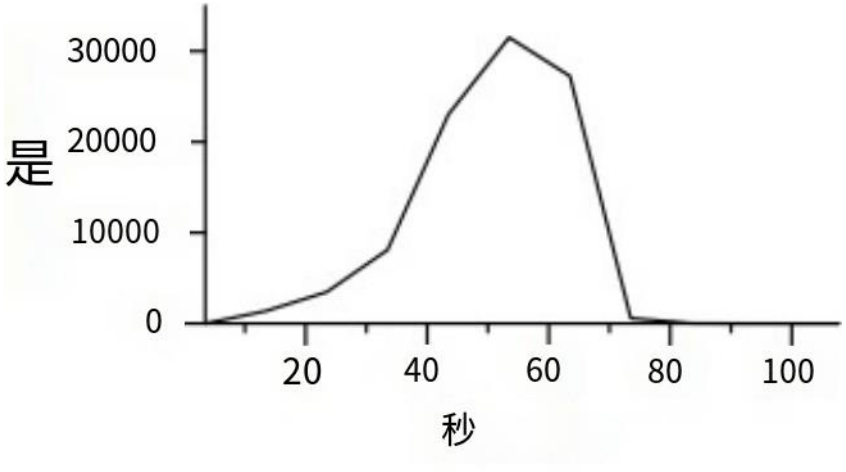


图2:随时间变化的数据传输速率

3.排序

对 10^10 个 100 字节的记录(大约 1 tb 的数据)进行排序

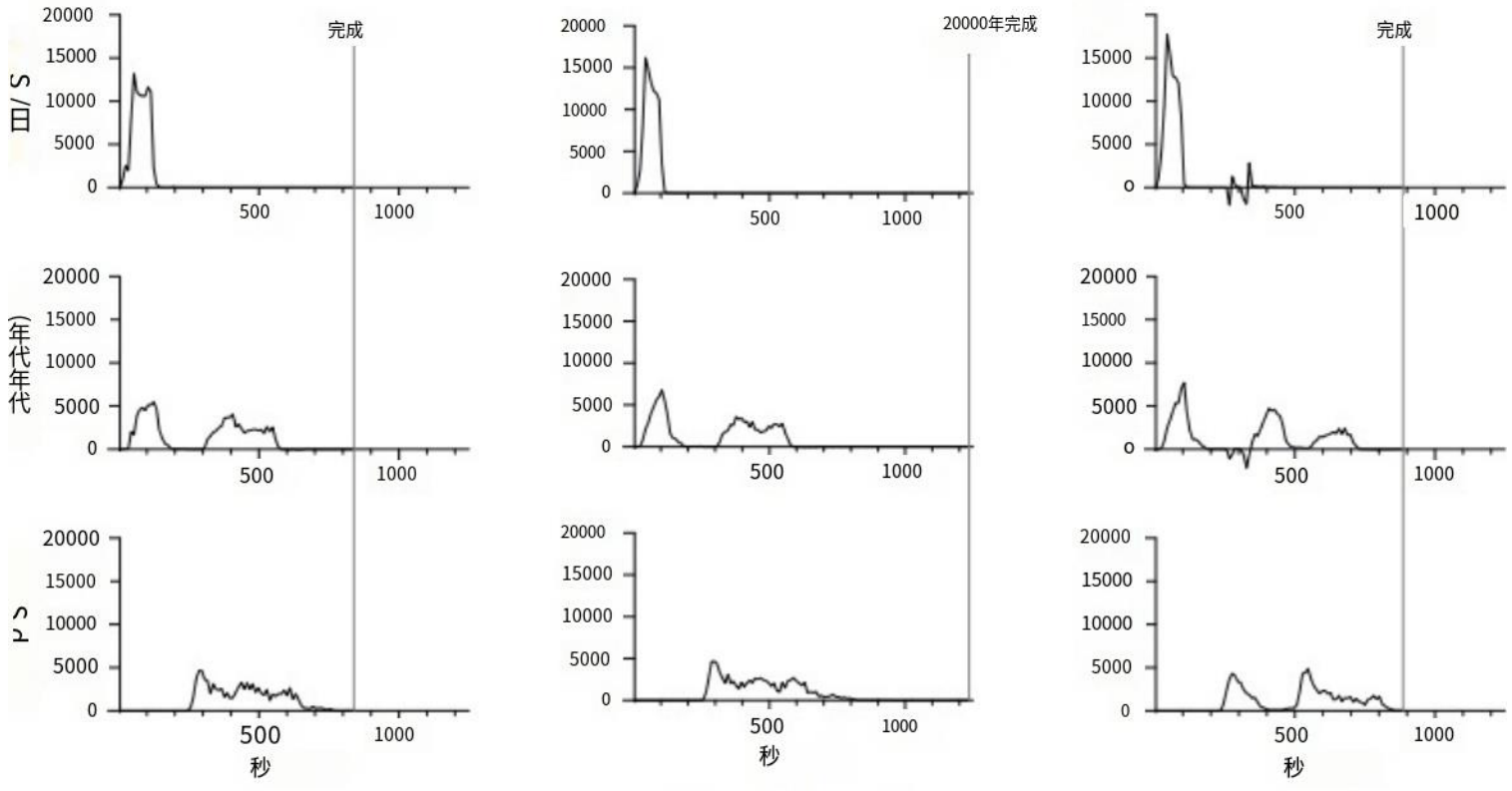
3 行 Map 函数从文本行中提取一个 10 字节的排序键，并将键和原始文本行作为中间键值对输出

- 使用内置的 Identity 函数作为 Reduce 操作符

传递中间的键值对作为输出的键值对

- 此基准的分区函数具有键分布的内置知识
- 添加一个 pre-pass MapReduce 操作来收集一个键的样本
- 使用采样键的分布来计算最终排序的分割点

3.排序



(a)正常执行　(b)无备份任务　(c) 200个任务被杀死

图3:排序程序不同执行时间的数据传输率

4. 备份任务的效果

- 图 3 (b)显示了在禁用备份任务的情况下 sort 程序的执行
- 960 秒后，除 5 个 reduce 任务外，所有任务都完成
- 最后几个掉队者直到 300 秒后才完成
- 整个计算耗时 1283 秒，耗时增加 44%

5. 机器故障

图 3 (c)显示了 sort 程序的执行过程

在计算进行了几分钟后，在 1746 个 worker 进程中故意杀死 200 个进程

底层的集群调度器立即在这些机器上重启新的工作进程

- 工作进程死亡显示为负输入速率，因为一些之前完成的 map 工作消失了，需要重做

重新执行这些地图工作的速度相对较快

整个计算在 933 秒内完成，包括启动开销(仅比正常执行时间增加 5%)

经验

2003 年 2 月发布了 MapReduce 库的第一个版本，并在 2003 年 8 月对其进行了重大增强

- 在谷歌内广泛使用，包括:
  - 大规模机器学习问题
  - b 谷歌 News 和 Google 产品的聚类问题
  - 提取用于生成流行查询报告的数据
  - 提取用于新实验和产品的网页属性
  - 大规模的图计算

平等就业机会
0
2003/03
2003/06
2003/09
2003/12
2004/03
2004/06
2004/09

作业数作业平均完成时间使用的机器天数	29,423 634秒 79186天
输入数据读取产生中间数据 输出数据写入	3288年结核病 758年结核病 193 电视
每个工作的平均工人机器数每个工作的平均工人死亡数每个工作的平均地图任务数每个工作的平均减少任务数	157 1.2 3,351 55
唯一的map实现唯一的reduce实现 独特的地图/减少组合	395 269 426

表1:MapReduce作业在2004年8月运行

1. 大规模索引

到目前为止，**MapReduce** 最重要的用途之一是完全重写了生产索引系统

提供以下几个好处:

索引代码更简单、更小、更容易理解

将概念上不相关的计算分开，而不是混合在一起，以避免对数据的额外传递

变得更容易操作

•通过向索引集群添加新机器，容易提高索引过程的性能

在b谷歌申请

可以被认为是基于我们对大型真实世界计算的经验，对其中一些模型的简化和提炼

- 提供一个可扩展到数千个处理器的容错实现

- 批量同步编程和一些 MPI 原语提供了更高层次的抽象，使程序员更容易编写并行程序

利用受限编程模型自动并行化用户程序，并提供透明的容错

- 局部性优化的灵感来自于活动磁盘等技术
- 备份任务机制类似于 Charlotte 系统中采用的热切调度机制
- 依赖于一个内部的集群管理系统，该系统负责在大量的共享机器集合上分发和运行用户任务
- 排序工具是 MapReduce 库的一部分，在操作上类似于 NOW-Sort

- River 提供了一个编程模型，其中进程通过分布式队列发送数据来相互通信
- BAD-FS 有一个与 MapReduce 非常不同的编程模型，与 MapReduce 不同的是，BAD-FS 的目标是跨广域网执行作业
- TACC 是一个旨在简化高可用网络服务的构建的系统，并依赖于作为实现容错机制的重新执行

总结

MapReduce