MapReduce：大型集群上的简化数据处理

Jeffrey Dean和Sanjay Ghemawat

jeff@google.com，sanjay@google.com

Google，Inc.

抽象

MapReduce是一种编程模型和一种关联-
处理和生成大型的有效实现
数据集。用户指定一个*地图*函数来处理
键/值对以生成一组中间键/值
对，以及归并所有中间元素的*reduce*函数
与同一中间键关联的值。许多
如图所示，现实世界中的任务可以在此模型中表达
在本文中。

以这种功能风格编写的程序可以自动执行。
Cally在大型的COM-群集上并行化并执行
机器。运行系统负责
划分输入数据的详细信息，安排程序
克在一组机器上执行，处理大量
机器故障，并管理所需的机器间
通讯。这使程序员无需任何
具有并行和分布式系统的经验，可以轻松实现
仅利用大型分布式系统的资源。

我们对MapReduce的实现在很大程度上
商业机器集群，并且具有高度可扩展性：
典型的MapReduce计算过程会处理许多
数千台计算机上的数十亿字节的数据。程式设计师
找到易于使用的系统：数百种MapReduce pro-
克已经实施，并且超过一千分之一
MapReduce作业在Google的集群上执行
每天。

1引言

在过去的五年中，作者和许多其他作者
Google已经实现了数百种特殊用途
处理大量原始数据的计算，
例如抓取的文档，Web请求日志等，以
等等。大多数此类计算都是概念上的
盟友直率。但是，输入数据通常是
大，计算必须分布在
成百上千台机器才能完成
合理的时间。如何解析的问题
分配计算，分配数据并处理
失败共同掩盖了原始的简单计算机
处理大量复杂代码
这些问题。

为了应对这种复杂性，我们设计了一种新的
使我们能够表达简单计算的抽象
我们试图执行的操作，但隐藏了混乱的设计
并行化，容错，数据分布
和库中的负载平衡。我们的抽象是
受*地图*启发并*减少* Lisp中存在的基元
和许多其他功能语言。我们意识到
我们大部分的计算都涉及应用*地图*操作
对我们输入中的每个逻辑“记录”进行排序，以便
计算一组中间键/值对，然后
对所有共享的值应用*归约*运算
相同的密钥，以便将派生的数据
适当地。我们对用户使用功能模型的情况如下：
指定的地图和归约运算允许我们对-
轻松进行大型计算并使用重新执行
作为容错的主要机制。

这项工作的主要贡献是简单和
强大的界面可实现自动并行化
和大规模计算的分布
该接口的实现可以实现
大型商用PC群集上的高性能。

第2节介绍了基本的编程模型和
举几个例子。第3节介绍了一个
针对的MapReduce接口的改进
我们基于集群的计算环境。第4节-
描述了编程模型的一些改进

计算各种派生数据，例如反演
索引，图结构的各种表示形式
Web文档，页数摘要
每个主机抓取的是

出现在OSDI 2004中

我们发现有用，第7节有表现
我们对各种实施的衡量
任务。第6节探讨了MapReduce在
Google包括我们使用它作为基础的经验

1个

第2页

重写我们的生产索引系统。秒
第7节讨论了相关的和将来的工作。

2编程模型

该计算采用一组*输入*键/值对，并且
产生一组*输出*键/值对。的用户
MapReduce库将计算表示为两个
功能：*映射*和*缩小*。

由用户编写的*Map*接受一个输入对，然后
产生一组*中间*键/值对。MapRe-
duce库将所有中间值关联在一起
用相同的中间键I引用并传递它们
到*减少*功能。

还由用户编写的*Reduce*函数接受
中间键I和该键的一组值。它
将这些值合并在一起以形成可能较小的值
一组值。通常只有零或一个输出值是
根据*Reduce*调用生成。中间价
ues通过迭代提供给用户的reduce函数
讲师。这使我们能够处理太高的值列表
大以适合内存。

2.1范例

考虑计算oc-数量的问题
大量文档中每个单词的出现
。用户将编写类似于以下内容的代码-
ing伪代码：

```
map（字符串键，字符串值）：
    //键：文档名称
    //值：文档内容
    对于每个单词w的值：
        EmitIntermediate（w，“ 1”）；

reduce（字符串键，迭代器值）：
    //键：一个字
    //值：计数列表
    int结果= 0;
    对于每个v值：
        结果+ = ParseInt（v）；
    Emit（AsString（result））；
```

map函数发出每个单词以及一个关联的单词
出现次数（在此简单示例中为“ 1”）。
reduce函数将所有发出的计数汇总在一起
对于一个特定的词。

另外，用户编写代码以填写*mapreduce*
具有输入和输出名称的*规范*对象-
放置文件以及可选的调整参数。用户然后
调用*MapReduce*函数，并将其传递给指定的

2.2类型

即使先前的伪代码是用术语编写的
字符串输入和输出，从概念上讲，
减少用户提供的功能已关联
类型：

```
地图      （k1，v1）          →清单（k2，v2）
减少（k2，list（v2））→list（v2）
```

即，输入键和值是从不同的位置绘制的
域比输出键和值。此外，
中间键和值来自相同的用途
main作为输出键和值。

我们的C ++实现将字符串往返传递
用户定义的函数并将其留给用户代码
在字符串和适当的类型之间转换。

2.3更多例子

以下是一些有趣的程序的简单示例
可以很容易地表示为MapReduce计算
位置。

分布式Grep：如果map函数发出一行，
匹配提供的模式。减少功能是
身份功能，仅复制提供的中间
将数据输入输出。

URL访问频率计数：地图功能-
处理网页请求和输出的日志
（URL，1）。reduce函数将所有值相加
对于相同的URL并发出（URL，总数）
对。

反向Web链接图：地图功能输出
（目标，源）对到目标的每个链接对
在名为source的页面中找到的URL。减少
函数将所有源URL的列表串联为-
与给定的目标URL相关联并发出该对：
（目标，列表（源））

每个主机的术语向量：术语向量总结了
文档或集中出现的最重要的单词
文档（单词，频率）对的列表。的
map函数发出一个（主机名，术语向量）
每个输入文档（主机名是
从文档的URL中提取）。那里-
将duce函数传递给所有按文档的术语向量

阳离子对象。用户代码与
MapReduce库（在C ++中实现）。附录A
包含此示例的完整程序文本。

对于给定的主机。它将这些术语向量加在一起，
扔掉不常用的字词，然后发出最终字词
（主机名，术语向量）对。

第3页

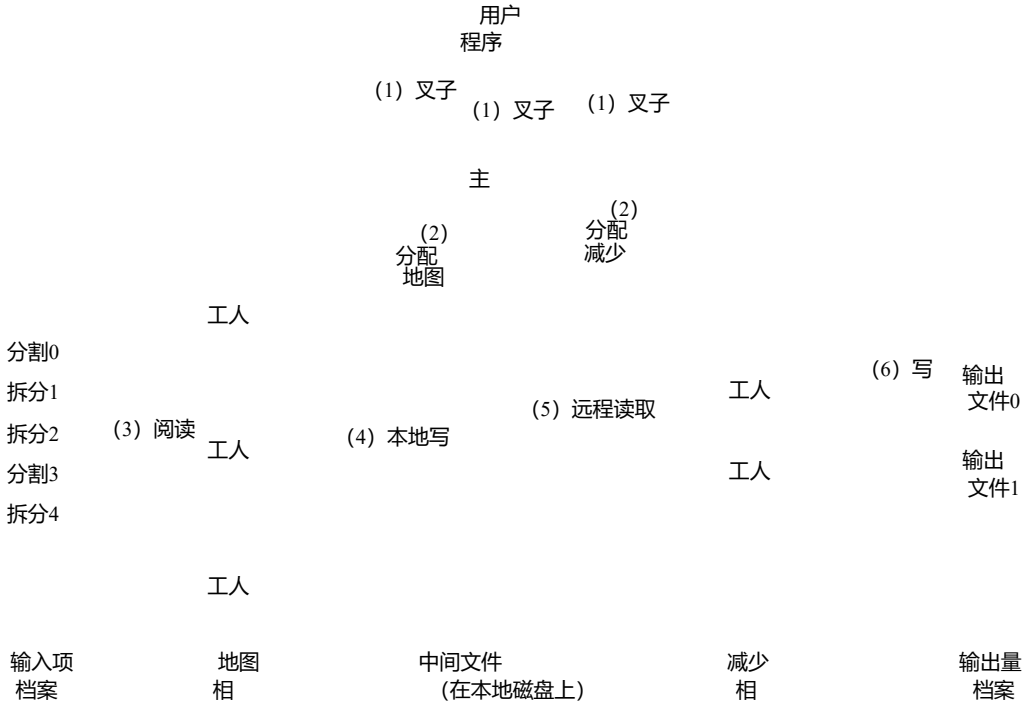


图1：执行概述

倒排索引： map函数解析每个文档
并发出一系列（单词，文档ID）
对。reduce函数接受给定的所有对
单词，对相应的文档ID进行排序并发出一个
（单词，列表（文档ID））对。所有输出的集合
对形成简单的倒排索引。容易扩充
此计算可跟踪单词位置。

分布式排序： map函数提取密钥
从每个记录中，并发出一个（键，记录）对。的
reduce函数将所有对保持不变。这个电脑
时间取决于
第4.1节和Sec-中描述的订购属性
4.2。

3实施

MapReduce的许多不同实现-
界面是可能的。正确的选择取决于
环境。例如，一种实现可以是
适用于小型共享内存计算机，另一台适用于
大型NUMA多处理器，另一个用于
甚至更多的联网机器。

大型商用PC集群与
交换式以太网[4]。在我们的环境中：

（1）机器通常是双处理器x86处理器
运行Linux，每台计算机有2-4 GB的内存。

（2）通常使用商品联网硬件
100兆比特/秒或1吉比特/秒
机器级别，但平均
所有二等分带宽。

（3）集群由数百或数千个
会出现故障，因此机器故障很常见。

（4）存储由便宜的IDE磁盘提供，位于-
直接连接到单个计算机。分布式文件
内部开发的系统[8]用于管理数据
存储在这些磁盘上。文件系统使用复制来
在不可靠的基础上提供可用性和可靠性
硬件。

（5）用户将作业提交到调度系统。每个工作
由一组任务组成，并由调度程序映射
到群集中的一组可用计算机。

3.1执行概述

本节介绍了针对性的实现
Google广泛使用的计算环境：

该*地图*调用在多个分布式
通过自动划分输入数据来划分机器

第4页

成一组M的*分割*。输入分割可以是
由不同的机器并行进行。*减少*言语
通过对中间键进行分区来分配位置
使用分区函数（例如，
hash（key）mod R）。分区数（R）和
分区功能由用户指定。

图1显示了MapReduce操作的总体流程
在我们的实施中。当用户程序
调用MapReduce函数，顺序如下
动作发生（图1中的编号标签对应-
跳到下面列表中的数字）：

1.首先在用户程序中添加MapReduce库
将输入文件分为M个（通常为16个）
每块兆字节至64兆字节（MB-
用户可以通过可选参数将其拖曳）。它
然后在群集上启动该程序的许多副本-
一台机器。

2.该程序的副本之一是特殊的-
主。其余的是被分配工作的工人
由主人。有M个地图任务，R个减少
分配任务。主人挑选闲散的工人
为每个任务分配一个映射任务或一个简化任务。

3.被分配了地图任务的工人读取
相应输入拆分的内容。它解析
输入数据中的键/值对，并传递每个
与用户定义的*Map*函数配对。中间
*Map*函数生成的diate键/值对
被缓冲在内存中。

4.定期将缓冲对写入本地
磁盘，通过分区分为R个区域
功能。这些缓冲对的位置
本地磁盘被传递回主服务器
负责将这些位置转发到
减少工人。

5.船长通知还原工人时
关于这些位置，它使用远程过程调用
从本地磁盘读取缓冲的数据
地图工作者。当reduce工作者阅读了所有内容后，
中间数据，按中间键排序
这样，所有出现在同一键上的都将分组
一起。需要排序，因为通常
许多不同的键映射到相同的reduce任务。如果
中间数据量太大，无法容纳
内存，使用外部排序。

6.减少工人迭代排序的中介
数据，并为每个唯一的中间键输入-
被反击，它传递密钥和对应的
用户的*Reduce*函数的一组中间值

7.完成所有地图任务和归约任务后
完成后，主机将唤醒用户程序。
此时，用户可以通过以下方式调用MapReduce
gram返回用户代码。

成功完成后，mapre的输出-
R输出文件中提供了两次执行（每个执行一次）
使用用户指定的文件名来简化任务）。
通常，用户不需要合并这些R输出
文件合并为一个文件–他们通常将这些文件作为输入传递给
另一个MapReduce调用，或从另一个Dis-
能够处理输入的贡献应用程序
分区成多个文件。

3.2主数据结构

主机保留几个数据结构。对于每张地图
任务和归约任务，它存储状态（*空闲*，*进行中*，
或*完成*），以及工作计算机的身份
（用于非空闲任务）。

主线是位置的管道
地图任务传播中间文件区域的数量
减少任务。因此，对于每个完成的地图任务，
母版存储R间的位置和大小
中介由地图任务产生的文件区域。更新
到此位置和尺寸信息作为地图接收
任务完成。信息被推送-
心理上对*正在进行的*减少任务的工人。

3.3容错

由于MapReduce库旨在帮助处理
使用数百或数千个大量数据
的机器，库必须容忍机器故障
优雅地。

工人失败

主人定期对每个工人执行ping操作。如果没有重新
从工人那里收到一定数量的
时间，主人将工人标记为失败。任何地图
由工作人员完成的任务将重置为初始状态
TiAl基*闲置*状态，因此有资格schedul-
在其他工人身上。同样，任何地图任务或减少
失败的工作程序上正在进行的任务也重置为*空闲*
并有资格进行重新安排。

发生以下故障时，将重新执行已完成的地图任务：
因为它们的输出存储在磁盘的本地磁盘上
故障机器，因此无法访问。已完成
减少任务不需要重新执行，因为它们
输出存储在全局文件系统中。

到此缩减分区的最终输出文件。

*Reduce*函数的输出已附加

当工作者A首先执行地图任务时，

然后稍后由工作人员B执行（因为A失败），所有

第5页

执行减少任务的工人会收到重新通知
执行。任何尚未读取
来自工作程序A的数据将读取工作程序B的数据。

MapReduce可以抵抗大规模的工人故障。
例如，在一次MapReduce操作期间，网络
对正在运行的群集进行维护会导致
一次无法访问80台计算机
普通的分钟。简单地重新执行MapReduce主文件
无法到达的工人机器完成的工作，以及
继续取得进步，最终完成了-
进行MapReduce操作。

主失败

使主控器编写定期检查点很容易
上述主数据结构。如果质量
任务结束后，可以从上一个开始新的副本
检查点状态。但是，鉴于只有一个
单主机，其失败可能性不大；因此，我们的
租金实施中止MapReduce计算
如果主服务器发生故障。客户可以检查这种情况
并根据需要重试MapReduce操作。

存在故障的语义学

当用户提供的*map*和*reduce*运算符被取消时
输入值的术语函数，我们的分布式
实现产生的输出将与
由无故障顺序执行
整个程序。

我们依靠map的原子提交并减少任务
输出以实现此属性。每个正在进行的任务
将其输出写入私有临时文件。减少任务
产生一个这样的文件，而映射任务产生R这样的文件
文件（每个缩减任务一个）。地图任务完成后，
工人向主人发送一条消息，包括
消息中R临时文件的名称。如果
主机收到一个已经完成的消息
完成地图任务后，它将忽略该消息。除此以外，
它在主数据结构中记录R文件的名称。

还原任务完成后，还原工作者
自动将其临时输出文件重命名为最终文件
输出文件。如果相同的归约任务在多
很多机器，将执行多个重命名调用
相同的最终输出文件。我们依靠原子重命名
基础文件系统提供给瓜尔的操作
最终文件系统状态仅包含数据
通过执行reduce任务产生。

我们的*地图*和*归约*运算符绝大多数是
确定性，以及我们的语义等同的事实-
程序员易于推理其程序的原因-
行为。当*map*和/或*reduce*运算符不为
确定性的，我们提供较弱但仍合理的服务
壁炉。在不确定的运算符存在的情况下，
特定化简任务R 1的输出等于
R 1的输出，由顺序执行
非确定性程序。但是，输出为
不同的归约任务R 2可能对应于输出
对于由不同的顺序执行产生的R 2
非确定性程序。

考虑映射任务M并减少任务R 1和R 2。
令e（R i）为已落实的R i的执行（存在
就是这样一种执行）。较弱的语义
出现是因为e（R 1）可能已经读取了产生的输出
通过执行一次M和e（R 2）可能已经读取了
由M的不同执行产生的输出。

3.4地区

网络带宽是我们相对稀缺的资源
计算环境。我们节省网络带宽-
利用输入数据这一事实
（由GFS [8]管理）存储在磁盘的本地磁盘上。
组成集群的计算机。GFS划分每个
文件分成64 MB的块，并分别存储几个副本
在不同的计算机上阻止（通常3份）。的
MapReduce主站获取地图的位置信息
将文件输入帐户并尝试安排地图
包含相关副本的计算机上的任务
生成输入数据。如果失败，它将尝试安排
任务输入数据副本附近的地图任务（例如，
与同一网络交换机上的工作计算机
包含数据的机器）。大跑步时
MapReduce操作在很大一部分
集群中的worker，大多数输入数据在本地读取，
不占用网络带宽。

3.5任务粒度

我们将地图阶段细分为M个部分，然后重新
如上所述，将相切成R块。理想情况下，M
并且R应该比工人人数大得多
机器。让每个工人执行许多不同的操作
任务改善了动态负载平衡，并加快了速度
工作人员失败时加快恢复速度：许多地图任务
它已经完成，可以分散到所有其他地方
工人机器。

M和R可以有多大存在实际界限
在我们的实施中，因为主人必须
O（M + R）调度决策，并保持O（M ∗ R）
状态如上所述在存储器中。（不变的事实-
但是，用于内存使用的参数很小：O（M ∗ R）
状态的一部分由大约一个字节组成

有能力在这种情况下顺序执行使它非常

每个地图任务/减少任务对的数据。）

出现在OSDI 2004中

6页

此外，R通常受用户约束，因为
每个reduce任务的输出最终以单独的out-
放置文件。实际上，我们倾向于选择M，以便每个
单个任务大约需要16 MB到64 MB的输入数据
（这样，上述位置优化是最有效的
有效），我们将R设为以下数字的一小部分倍数-
我们希望使用的劳动者机器。我们经常
形成MapReduce计算，其中M = 200、000和
R = 5，000，使用2,000台工人计算机。

3.6备份任务

延长总时间的常见原因之一
进行MapReduce操作的对象是“散乱者”：
脊椎需要很长时间才能完成
最后几个映射或减少计算中的任务。
流浪者之所以出现，可能是由于多种原因。对于前
足够，磁盘损坏的计算机可能会遇到以下问题-
可能纠正的错误，可能会减慢其读取性能
从30 MB / s到1 MB / s。集群调度系统
他们可能在机器上安排了其他任务，
使它执行MapReduce代码的速度更慢
由于竞争CPU，内存，本地磁盘或网络-
工作带宽。我们最近遇到的一个问题是
机器初始化代码中的错误导致程序
sor缓存将被禁用：受影响主机上的计算
中国的速度减慢了一百倍。

我们有一个通用的机制来缓解问题
流浪者的lem。当MapReduce操作关闭时
完成时，主服务器安排备份执行
其余*正在进行的*任务。任务已标记
无论何时主数据库或备份数据库都已完成
执行完成。我们已经调整了此机制，因此
通常会增加计算资源
该操作使用的百分比不超过百分之几。
我们发现这大大减少了时间
完成大型MapReduce操作。作为考试
例如，第5.3节中描述的排序程序需要44％
当备份任务机制为
禁用的。

4细化

尽管简单地提供了基本功能
编写*Map*和*Reduce*函数对于大多数人就足够了
需求，我们发现了一些扩展很有用。这些是
本节中介绍。

4.1分区功能

MapReduce的用户指定减少的数量
他们想要的任务/输出文件（R）。数据参与
使用分区功能在这些任务上进行分配
中间键。默认的分区功能是
提供使用散列的代码（例如“ hash（key）mod R”）。
这往往会导致分区非常均衡。在
但是，在某些情况下，通过
按键的其他功能。例如，有时
输出键是网址，我们希望所有输入
单个主机最终位于同一输出文件中。支持
这种情况，MapReduce库的用户
可以提供特殊的分区功能。例如，
使用“ hash（Hostname（urlkey））mod R”作为参数
分类功能会导致来自同一主机的所有URL
最终在同一输出文件中。

4.2订购保证

我们保证在给定的分区内，
diate键/值对以递增键或-
der。这种订购保证使其易于生成
每个分区的排序输出文件，在以下情况下很有用
输出文件格式需要支持有效的随机
通过键访问查找，或者输出的用户发现它包含-
方便对数据进行排序。

4.3组合器功能

在某些情况下，
中介每个地图任务产生的键，并且用户-
指定的*Reduce*函数是可交换的并且具有关联性
tive。一个很好的例子就是字数考试-
请参阅第2.1节。由于单词频率倾向于跟随
Zipf分布，每个地图任务将产生数百个
或<the，1>形式的成千上万条记录。所有的
这些计数将通过网络发送到单个
执行任务，然后通过*Reduce*函数加在一起
产生一个数字。我们允许用户指定一个
可选的*合并器*功能，可部分合并
在通过网络发送数据之前。

该*组合器*功能的每个机器上执行
执行地图任务。通常使用相同的代码
同时实现组合器和约简功能
位置。reduce函数和
组合函数是MapReduce库如何处理-
搁置函数的输出。减少输出
函数被写入最终输出文件。输出
将组合器功能写入中间文件，该中间文件
将被发送到reduce任务。

部分合并可显着加快某些
MapReduce操作的类。附录A包含
使用组合器的示例。

4.4输入和输出类型

MapReduce库提供了以下方面的支持：
将数据放入几种不同的格式。例如，“文字”

第7页

模式输入将每行视为键/值对：键
是文件中的偏移量，值是
线。另一种常见的支持格式存储
键/值对的顺序（按键排序）。每个输入
类型实现知道如何将自己拆分为均值-
处理为单独的地图任务的有效范围（例如，
文字模式的范围分割可确保范围分割
仅在线边界处固化）。用户可以添加对
通过提供模拟输入的实现来实现新的输入类型
PLE *读卡器*接口，虽然大部分用户只是使用的一个
少量的预定义输入类型。

一*读者*并不一定需要提供数据
从文件读取。例如，很容易定义一个*阅读器*
从数据库或数据结构中读取记录
映射到内存中。

以类似的方式，我们支持一组输出类型
用于生成不同格式的数据，这很容易
用户代码以添加对新输出类型的支持。

4.5副作用

在某些情况下，MapReduce的用户发现它会-
方便产生辅助文件作为附加输出
从他们的地图和/或减少运营商。我们依靠
应用程序编写者，以使此类副作用原子化并
幂等。通常，应用程序会写入速度-
Rary文件，并在文件被原子命名后对其进行原子重命名
完全生成。

我们不提供对原子两相COM-的支持
单个任务产生的多个输出文件。
因此，使用以下命令生成多个输出文件的任务
跨文件一致性要求应确定-
主义的。实际上，这种限制从来都不是问题。

4.6跳过不良记录

有时用户代码中的错误会导致*地图*
或*Reduce*函数确定性地在某些情况下崩溃
记录。这些错误会阻止MapReduce操作从
完成。通常的做法是修复错误，
但是有时候这是不可行的；也许错误在
第三方库，其源代码不可用-
能够。另外，有时可以忽略一些
记录，例如对
大数据集。我们提供了一种可选的执行方式
MapReduce库检测到哪些记录的位置
导致确定性崩溃，并跳过以下记录-
der取得进步。

每个工作进程都安装一个信号处理程序
捕获分段违规和总线错误。之前
调用用户*Map*或*Reduce*操作后，MapRe-
duce库存储参数的序列号
在全局变量中。如果用户代码产生信号，
信号处理程序发送一个“最后喘息”的UDP数据包，该数据包
包含MapReduce mas-的序列号
ter。当主人看到一个以上的故障
特定记录，它指示该记录应为
发出下一次更正时跳过
创建Map或Reduce任务。

4.7本地执行

*Map*或*Reduce*函数中的调试问题可以是
棘手，因为实际计算发生在
分布式系统，通常在几千台机器上，
通过动态制定的工作分配决定
大师。为了帮助进行调试，分析和
小规模测试，我们已经开发了一种替代的
MapReduce库的序列化
在上执行MapReduce操作的所有工作
本地机器。控件提供给用户，因此
可以将计算限制在特定地图上
任务。用户使用特殊标志调用其程序，然后
然后可以轻松地使用任何调试或测试工具
发现有用（例如gdb）。

4.8状态信息

主服务器运行内部HTTP服务器并导出
一组供人类消费的状态页。sta-
tus页面显示了计算进度，例如
已经完成了多少个任务，正在执行多少个任务
进度，输入字节，中间数据字节，字节
输出，处理速率等。页面还包含
链接到标准错误和标准输出文件
由每个任务负责。用户可以使用此数据进行预
决定计算将花费多长时间，以及是否
不应将更多资源添加到计算中。
这些页面也可以用来确定何时
投放速度比预期慢得多。

此外，顶层状态页面还会显示
工作人员失败了，以及哪个映射和减少任务
他们失败时正在处理。此信息-
尝试诊断设备中的错误时，该功能非常有用。
用户代码。

4.9柜台

MapReduce库为
计算各种事件的发生。例如，用户
代码可能要计算已处理的单词总数
或索引的德国文件数量等

要使用此功能，用户代码将创建一个命名计数器
对象，然后适当增加计数器
在*地图*和/或*缩小*功能。例如：

```
Counter *大写;
大写= GetCounter（“大写”）;

map（字符串名称，字符串内容）：
    对于内容中的每个单词w：
        如果（IsCapitalized（w））：
            大写-> Increment（）;
        EmitIntermediate（w，“ 1”）;
```

来自单个工作机的计数器值
定期传播给主节点（pi带
在ping响应上）。大师汇总计数器
成功映射的价值并减少任务和回报
当他们执行MapReduce操作时将它们发送给用户代码
完成了。当前计数器值也将
在主状态页面上播放，以便人类可以
观看实时计算的进度。聚集时
门控计数器值，主控消除了
复制相同映射的执行或将任务简化为
避免重复计算。（可能会重复执行
从我们使用备份任务以及重新执行
因失败而导致的任务。）

一些计数器值会自动维护
由MapReduce库提供，例如
放置已处理的键/值对和输出数量
产生的键/值对。

用户发现计数器功能对
检查MapReduce操作的行为。对于
例如，在某些MapReduce操作中，用户代码
可能要确保输出对的数量
产生的正好等于输入对的数量
或德国文件的一部分
所收取的费用在总人数的一定容忍范围内
ber处理的文件。

5表现

在本节中，我们将评估MapRe-
在运行于大型集群上的两次计算上
机器。一种计算可搜索大约
大约1 TB的数据在寻找特定的模式
燕鸥 其他计算将大约一个
一个字节的数据。

这两个程序代表了
由MapReduce用户编写的真实程序集–
一类程序将一个代表的数据洗牌
归于另一种，另一类提取少量
来自大型数据集的有趣数据。

5.1集群配置

所有程序都在集群上执行，
由大约1800台机器组成。每个ma-
chine拥有两个2GHz的Intel Xeon处理器和Hyper-
启用线程，4GB内存，两个160GB IDE

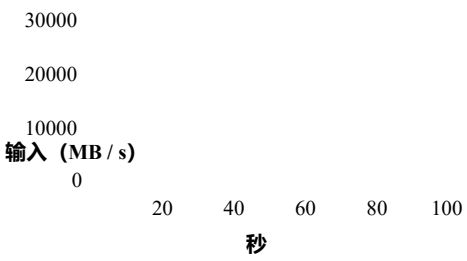


图2：随着时间的数据传输速率

磁盘和一个千兆以太网链接。机器是
安排在两级树形交换网络中
具有大约100-200 Gbps的总带宽
根处可用的宽度。所有的机器都是
在同一托管设施中，因此往返
两台机器之间的时间少于一百万
一秒钟。

4GB内存中约有1-1.5GB
已由群集上运行的其他任务保留。的
程序在周末的下午执行，
CPU，磁盘和网络大部分处于空闲状态。

5.2 Grep

该*grep的*程序通过扫描10 10 100字节记录，
搜索相对罕见的三字符模式（
模式出现在92337条记录中）。输入分为
大约64MB（M = 15000），并且
轮胎输出放在一个文件中（R = 1）。

图2显示了计算过程
时间。Y轴显示输入数据的速率
扫描。随着更多机器的出现，速度逐渐提高
被分配给此MapReduce计算，并且峰
当分配了1764个工人时，速度超过30 GB / s。
地图任务完成后，速率开始下降并达到
零大约80秒进入计算。整个
从开始算起大约需要150秒
完成。这包括大约一分钟的启动过度-
头。开销是由于pro-
克到所有工作机，并且延迟与之交互
GFS打开一组1000个输入文件并获取
位置优化所需的信息。

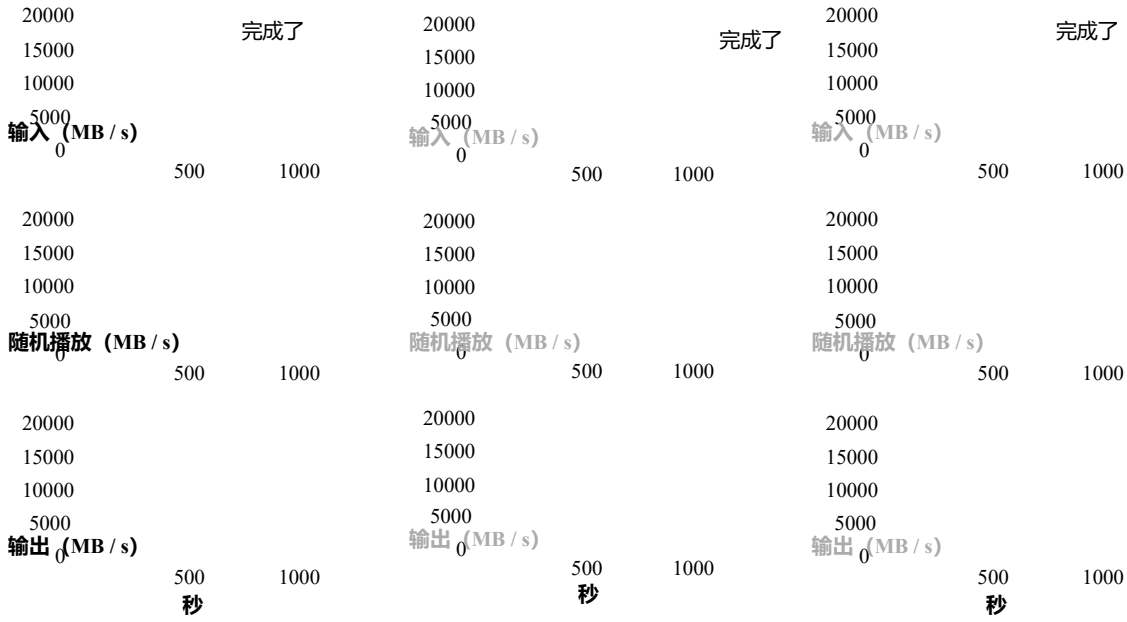
5.3排序

该*类型的*节目种类10 10 100字节记录（内约
至少1 TB的数据）。该程序是仿照
TeraSort基准测试[10]。

排序程序由少于50行组成
用户代码。三行*Map*函数提取10字节
从文本行对键进行排序，并发出键和

出现在OSDI 2004中

第9页



（a）正常执行 （b）没有备份任务 （c）200个任务被杀死

图3：不同程序执行时随时间的数据传输速率

原始文本行作为中间键/值对。我们
使用内置的*Identity*函数作为*Reduce*运算符。
此函数将中间键/值对不传递
更改为输出键/值对。最终排序
输出写入一组两路复制的GFS文件
（即，将2 TB写入程序的输出）。

和以前一样，输入数据被分成64MB
（M = 15000）。我们将排序的输出划分为4000
档案（R = 4000）。分区功能使用ini-
密钥的字节字节将其分离为R个片段之一。

我们针对此基准测试的分区功能具有以下特点：
了解密钥的分配。总的来说
排序程序，我们将添加一个预传递MapReduce
将收集密钥样本的操作，以及
使用采样键的分布来计算
最终排序过程的分数。

图3（a）显示了正常执行的进度
排序程序。左上图显示了费率
在读取输入时。速率达到约13 GB / s的峰值
并且由于所有地图任务均在以下情况下完成而很快死亡：
前200秒已经过去。注意输入速率
小于*grep*。这是因为排序图任务
花大约一半的时间和I / O带宽写入-
中间输出到其本地磁盘。相应的
grep的中间输出大小可以忽略不计。

左中图显示了数据的速率
通过网络从地图任务发送到重新
完成任务。这种改组最早在第一次
地图任务完成。图中的第一个驼峰是
第一批约有1700个简化任务（
整个MapReduce分配了大约1700台机器，
而每台机器最多只能执行一次reduce任务
时间）。大约需要300秒的计算时间
这些第一批的reduce任务完成后，我们开始
为剩余的reduce任务改组数据。全部
改组大约需要600秒才能完成。

左下图显示了排序的速率
减少任务将数据写入最终输出文件。
在第一次改组结束之前有一个延迟
骚动和写作时期的开始，因为
中国正忙于整理中间数据。写
继续以大约2-4 GB / s的速度运行一会儿。所有的
写入大约需要850秒才能完成。
包括启动开销，整个计算需要
891秒。这类似于目前报道得最好的
TeraSort基准测试[10]的结果为1057秒。

需要注意的几件事：输入速率高于
本地化造成的洗牌率和产出率
优化–大多数数据是从本地磁盘读取的，
绕过我们相对带宽受限的网络。
混洗率高于输出率，因为
输出阶段写入已排序数据的两个副本（我们
制作两个输出副本以提高可靠性和可用性-
能力原因）。我们写两个副本是因为
提供的可靠性和可用性机制
通过我们的基础文件系统。网络带宽重新
如果取消了写入数据的要求，将会减少
底层文件系统使用擦除编码[14]而不是
复制。

第10页

5.4备份任务的效果

在图3（b）中，我们显示了排序程序的执行-
备份任务已禁用的语法。执行流程是
与图3（a）相似，除了
一条很长的尾巴，几乎没有写操作发生。
960秒后，除5个reduce任务外，所有其他任务
完成。但是，这最后的几个散客并没有找到
直到300秒后消失。整个计算需要
1283秒，经过时间增加了44％。

5.5机器故障

在图3（c）中，我们显示了排序程序的执行
在这里，我们故意杀死了1746名工人中的200名
处理几分钟到计算中。的
基础群集调度程序立即重新启动
工人在这些计算机上进行处理（因为只有
机油被杀死，机器仍在运转
正确地）。

工人死亡显示为负投入率
因为一些以前完成的地图工作消失了
（因为相应的地图工作人员被杀）和
需要重做。重新执行此地图工作
发生得比较快。整个计算过程
在933秒内刷新一次，包括启动开销（
比正常执行时间增加5％）。

6经验

我们在中编写了MapReduce库的第一个版本
2003年2月，对
它是在2003年8月进行的，包括位置优化，
跨工作人员执行任务的动态负载平衡
机器等。自那时以来，我们一直很愉快
对MapReduce li-的广泛适用性感到惊讶
我们一直在努力解决各种问题。
它已被广泛应用于各个领域
Google，包括：

- •大规模的机器学习问题，
- •Google新闻和
  Froogle产品，
- •提取用于生成流行报告的数据
  查询（例如Google Zeitgeist），
- •提取网页属性以进行新的实验-
  要素和产品（例如地理信息提取
  来自大量网页的正确位置
  本地化搜索），以及
- •大规模图形计算。

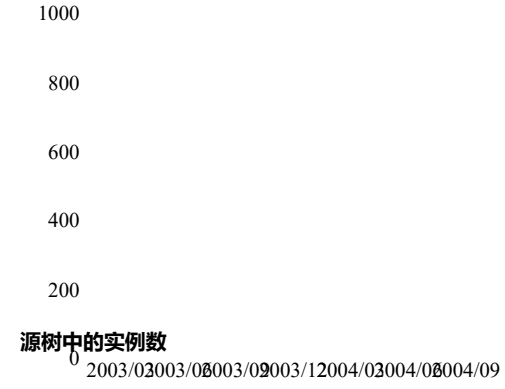


图4：一段时间内的MapReduce实例

工作数量	29,423
平均工作完成时间	634秒
使用机器天数	79,186天
读取输入数据	3,288 TB
产生的中间数据	758 TB
写入输出数据	193 TB
每个作业的平均工人机器	157
每个工作的平均工人死亡人数	1.2
每个作业的平均地图任务	3,351
每个作业平均减少任务	55
独特的*地图*实现	395
独特的*reduce*实现	269
独特的*地图/缩小*组合	426

表1：2004年8月运行的MapReduce作业

图4显示了
将单独的MapReduce程序签入我们的主程序
源代码管理系统随着时间的推移，从0 in
2003年初，到后期为止将近900个独立实例
2004年9月。MapReduce在以下方面非常成功：
因为它使得编写一个简单的程序成为可能
在课程中在一千台机器上高效运行
半小时，大大加快了开发和
原型制作周期。此外，它允许程序员
没有分布式和/或并行的经验
系统以轻松利用大量资源。

在每个作业结束时，MapReduce库将记录
有关使用的计算资源的统计信息
工作。在表1中，我们显示了一些统计信息
MapReduce作业于2004年8月在Google运行。

6.1大规模索引

迄今为止，我们最重要的MapReduce用途之一
已经完全重写了生产指标-

第11页

生成用于
Google网页搜索服务。索引系统需要
作为输入，已检索到大量文档
由我们的抓取系统存储为一组GFS文件。的
这些文件的原始内容超过20个
字节的数据。索引过程按顺序运行
五到十个MapReduce操作。使用MapReduce
（而不是先前版本中的临时分布式通行证，
索引系统的版本）提供了多种好处-
适合：

- 索引代码更简单，更小且更容易
  了解，因为处理错误的代码
  公差，分布和并行化被隐藏
  在MapReduce库中。例如，
  计算的一个阶段的大小从
  大约3800行C ++代码到大约-
  使用MapRe-表达时大约有700行
  ce子

- MapReduce库的性能很好
  足以使我们在概念上保持不相关
  计算是分开的，而不是将它们混合为-
  避免额外的数据传递。这个
  使更改索引过程变得容易。对于
  例如，一项更改花了几个月的时间
  在我们旧的索引系统中制作仅需几个
  天在新系统中实施。

- 索引过程变得更加容易
  操作，因为大多数问题是由
  机器故障，慢速机器和网络
  打Re由MapRe-自动处理
  杜斯图书馆，无需操作员干预。进一步-
  更多，很容易提高性能
  通过将新机器添加到
  德兴集群。

7相关工作

许多系统提供了受限的编程
并使用限制来并行化组合
自动排名。例如，关联函数
可以在N元素的所有前缀上计算tion
使用并行前缀在N个处理器上以对数N时间记录数组
计算[6、9、13]。可以考虑MapReduce
一些模型的简化和提炼
根据我们在大型真实计算机上的经验，
约会。更重要的是，我们提供了容错功能
可扩展到数千个处理器的实施。
相反，大多数并行处理系统具有
仅在较小的规模上实施，而
向程序员处理机器故障的详细信息。

批量同步编程[17]和一些MPI
原语[11]提供了更高层次的抽象，
使程序员更容易编写并行程序
克。这些系统与
MapReduce是MapReduce利用受限的Pro-
语法模型以并行化用户程序自动
并提供透明的容错能力。

我们的位置优化从中汲取了灵感
技术，例如活动磁盘[12、15]，其中
将时间推入接近的处理元素中
到本地磁盘，以减少跨设备发送的数据量
I / O子系统或网络。我们依靠商品
少量磁盘直接连接到的处理器
连接，而不是直接在磁盘控制器上运行
处理器，但一般方法是相似的。

我们的备份任务机制类似于热切的
夏洛特系统中使用的调度机制
tem [3]。渴望的缺点之一
日程安排是，如果给定任务导致重复失败，
整个计算无法完成。我们修复了一些
我们的跳过机制对此问题的立场
不良记录。

MapReduce实现依赖于内部
集群管理系统负责
在大量的任务集上分发和运行用户任务
共享机器。尽管不是本文的重点，
集群管理系统在精神上与其他类似
系统，例如Condor [16]。

属于MapReduce的排序工具
库的操作类似于NOW-Sort [1]。资源
机器（地图工作者）对要排序的数据进行分区
并将其发送给R减少工人之一。每次减少
worker在本地对数据进行排序（如果可能，在内存中）。的
当然NOW-Sort没有用户可定义的地图
和Reduce功能使我们的图书馆得到广泛应用-
电缆。

River [2]提供了一个编程模型，其中
通过发送数据彼此通信
在分布式队列中。就像MapReduce，河
系统尝试提供良好的平均案例性能
即使存在由以下情况引起的不均匀性
异构硬件或系统扰动。河
通过精心安排磁盘和网络来实现此目的
转移以达到平衡的完成时间。MapRe-
ce子有不同的方法。通过限制亲
语法模型，MapReduce框架能够
将问题划分为大量的罚款-
细粒度的任务。这些任务是动态安排的
在可用的工人上，以便更快的工人处理更多
任务。受限的编程模型还允许
我们计划在附近执行冗余的任务执行
工作的结束，大大减少了完成时间
存在不均匀性（例如缓慢或卡住）
工人）。

BAD-FS [5]具有非常不同的编程模型
来自MapReduce，与MapReduce不同，它的目标是

第12章

跨广域网执行作业。怎么样-
曾经有两个基本相似之处。（1）两者
系统使用冗余执行从数据中恢复
故障造成的损失。（2）都使用位置感知
调度以减少跨连接发送的数据量
孕育的网络链接。

TACC [7]是旨在简化控制的系统。
高可用性网络服务的构建。喜欢
MapReduce，它依赖于重新执行作为一种机制
实现容错。

8结论

MapReduce编程模型取得了成功-
在Google上已完全用于许多不同目的。我们
将此成功归因于几个原因。一，模型
易于使用，即使对于没有经验的程序员也是如此
与并行和分布式系统，因为它隐藏了
并行化，容错，局部性优化的详细信息
小型化和负载平衡。二，种类繁多
问题可以很容易地表达为MapReduce com-
推定。例如，MapReduce用于生成
Google生产网络搜索服务的数据分级
副，用于分类，用于数据挖掘，用于机器学习，
和许多其他系统。第三，我们开发了
MapReduce的实现，可扩展为大型集群-
一千个机器组成了数千个机器。的
实施可有效利用这些机器
来源，因此适用于许多
Google遇到的大量计算问题。

我们从这项工作中学到了几件事。第一，
限制编程模型可以轻松解析
分配和分配计算并进行计算
计算容错。二，网络带宽
是一种稀缺资源。我们的许多优化
因此，系统旨在减少
跨网络发送的数据：位置优化算法
使我们无法从本地磁盘读取数据并写入单个
将中间数据复制到本地磁盘可节省网络
带宽。第三，冗余执行可以用来
减少慢速机器的影响，并处理
机器故障和数据丢失。

致谢

Josh Levenberg在修订和
使用num-扩展用户级MapReduce API
根据他的使用经验，获得了许多新功能
MapReduce和其他人的增强建议-
。MapReduce读取其输入并写入其
输出到Google文件系统[8]。我们想要
感谢Mohit Aron，Howard Gobioff，Markus Gutschke，
David Kramer，梁信德和Josh Redstone
他们在开发GFS方面的工作。我们也想
感谢Percy Liang和Olcan Sercinoglu的工作
在开发由...使用的集群管理系统
MapReduce。Mike Burrows，Hilson Hsieh，Josh Leven-
伯格，沙龙·佩尔，罗伯·派克和黛比·沃拉克亲
在本文档的早期草案中提供了有用的评论
每。匿名的OSDI评论者和我们的牧羊人，
埃里克·布鲁尔（Eric Brewer），提供了许多有关该领域的有用建议
需要改进的地方。最后，我们感谢所有人
Google工程中的MapReduce用户，或者-
提供有用的反馈，建议，
和错误报告。

参考资料

[1] Andrea C. Arpaci-Dusseau，Remzi H. Arpaci-Dusseau，David E. Culler，Joseph M. Hellerstein和David A. Pat-特森。高效的工作网络分类
站。在*1997年ACM SIGMOD论文集中，跨国数据管理大会*，图森，
1997年5月，亚利桑那州。

[2] Remzi H. Arpaci-Dusseau，埃里克·安德森，诺亚
Treuhaft，David E. Culler，Joseph M.Hellerstein，David
Patterson和Kathy Yelick。使用River的集群I / O：
使快速案例变得普遍。在*第六届会议论文集并行和分布式输入/输出研讨会系统（IOPADS '99）*，第10-22页，佐治亚州亚特兰大，
1999年5月。

[3] Arash Baratloo，Mehmet Karaul，Zvi Kedem和Peter
威科夫。夏洛特：网络上的元计算。在*Pro-第九届国际并行会议论文集and Distributed Computing Systems*，1996年。

[4] Luiz A. Barroso，Jeffrey Dean和UrsHölzle。网页
搜索行星：Google集群架构。*电气工程师学会Micro*，23（2）：22-28，2003年4月。

[5]约翰·本特，道格拉斯·塞恩，安德里亚·阿尔帕奇·杜索，
雷姆齐·H·阿尔帕基·杜梭和米隆·利夫尼。明确的
批处理分布式文件系统中的控件。在*Pro-第一届USENIX网络研讨会论文集系统设计和实现NSDI*，2004年3月。

[6] Guy E. Blelloch。扫描为原始并行操作。
IEEE电脑交易，C-38（11），11月
1989年。

[7] Armando Fox，Steven D. Gribble，Yatin Chawathe，
Eric A. Brewer和Paul Gauthier。基于集群的规模
能够提供网络服务。在*第16届ACM会议论文集中操作系统原理专题讨论会*，第78页–
91，法国圣马洛，1997年。

[8] Sanjay Ghemawat，Howard Gobioff和Shun-Tak Le-ung。Google文件系统。在*第19届Op-erating Systems Principles*，第29-43页，乔治湖，
纽约，2003年。

第13页

[9] S. Gorlatch。系统有效的扫描并行化
和其他列表同态。在L. Bouge，P。Fraigni-
aud，A。Mignotte和Y. Robert，*Euro-Par'96的*编辑。
并行处理，计算机科学讲座
1124，第401-408页。施普林格出版社，1996年。

[10]吉姆 灰色。 排序基准 主页。
http://research.microsoft.com/barc/SortBenchmark/。

[11] William Gropp，Ewing Lusk和Anthony Skjellum。
使用MPI：可移植并行编程
消息传递接口。麻省理工学院出版社
1999年。

[12] L. Huston，R。Sukthankar，R。Wickremesinghe，M。Satya-
narayanan，GR Ganger，E。Riedel和A.Ailamaki。-
amond：一种存储架构，可在内部
主动搜索。在*2004 USENIX文件的会议记录中*
和存储技术FAST会议，2004年4月。

[13]理查德·拉德纳和迈克尔·菲舍尔。并行前缀
计算。*ACM杂志*，27（4）：831–838，1980年。

[14] Michael O. Rabin。信息的有效传播
安全性，负载平衡和容错能力。*期刊*
ACM，36（2）：335-348，1989。

[15] Erik Riedel，Christos Faloutsos，Garth A. Gibson和
大卫·纳格（David Nagle）。活动磁盘用于大规模数据处理-
ing。*IEEE计算机*，第68-74页，2001年6月。

[16] Douglas Thain，Todd Tannenbaum和Miron Livny。
实践中的分布式计算：Condor实验
ence。*并发与计算：实践与实践*
《perence》，2004年。

[17] LG Valiant。并行计算的桥接模型。
ACM通讯，1997，33（8）：103-111。

[18]吉姆·韦利。Spsort：如何快速排序1 TB。
http://alme1.almaden.ibm.com/cs/spsort.pdf。

词频

本节包含一个计算数字的程序
一组输入文件中每个唯一单词的出现次数
在命令行上指定。

```
#include“ mapreduce / mapreduce.h”

//用户的地图功能
class WordCounter：公共映射器{
 上市：
  虚拟void Map（const MapInput＆input）{
    const string＆text = input.value（）；
    const int n = text.size（）；
    为（int i = 0; i <n;）{
      //跳过领先的空格
      而（（i <n）&& isspace（text [i]））
        i ++;

      //查找单词结尾
      int start = i;
      while（（i <n）&&！isspace（text [i]））
        i ++;
      如果（开始<i）
        Emit（text.substr（start，i-start），“ 1”）；
    }
  }
};
REGISTER_MAPPER（WordCounter）；

//用户的reduce函数
class Adder：public Reducer {
  虚拟虚空Reduce（ReduceInput *输入）{
    //使用
    //相同的键并添加值
    int64值= 0;
    while（！input-> done（））{
      值+ = StringToInt（input-> value（））；
      输入-> NextValue（）；
    }

    //发射input-> key（）的总和
    Emit（IntToString（value））；
  }
};
REGISTER_REDUCER（加法器）；

int main（int argc，char ** argv）{
  ParseCommandLineFlags（argc，argv）；

  MapReduceSpecification规格；

  //将输入文件列表存储到“ spec”中
  for（int i = 1; i <argc; i ++）{
    MapReduceInput *输入= spec.add_input（）；
    input-> set_format（“ text”）；
    输入-> set_filepattern（argv [i]）；
    输入-> set_mapper_class（“ WordCounter”）；
  }

  //指定输出文件：
  //         / gfs / test / freq-00000-of-00100
  //         / gfs / test / freq-00001-of-00100
  //         ...
  MapReduceOutput * out = spec.output（）；
  out-> set_filebase（“ / gfs / test / freq”）；
  out-> set_num_tasks（100）；
  out-> set_format（“ text”）；
  out-> set_reducer_class（“ Adder”）；

  //可选：在地图中进行部分和
  //节省网络带宽的任务
  out-> set_combiner_class（“ Adder”）；

  //调整参数：最多使用2000
  //机器和每个任务100 MB的内存
  spec.set_machines（2000）；
  spec.set_map_megabytes（100）；
  spec.set_reduce_megabytes（100）；

  //现在运行它
  MapReduceResult结果；
  如果（！MapReduce（spec，＆result））abort（）；

  //完成：“结果”结构包含信息
  //关于计数器，花费的时间，
  //使用的机器等

  返回0;
}
```