TURING 图灵程序设计丛书

MANNING

Hadoop in Action

Hadoop 实战

[美] Chuck Lam 著
韩冀中 译

- 纵情享受海量数据之美
- 揭开云计算的神秘面纱
- 深入分析，追本溯源

人民邮电出版社
POSTS & TELECOM PRESS

www.ChinaDBA.net 中国DBA超级论坛

图书在版编目（CIP）数据

Hadoop实战 / （美）拉姆（Lam,C.）著；韩冀中译.
-- 北京：人民邮电出版社，2011.10
（图灵程序设计丛书）
书名原文：Hadoop in Action
ISBN 978-7-115-26448-0

Ⅰ. ①H… Ⅱ. ①拉… ②韩… Ⅲ. ①数据处理—应用软件—网络编程 Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2011)第191474号

内 容 提 要

作为云计算所青睐的分布式架构，Hadoop 是一个用 Java 语言实现的软件框架，在由大量计算机组成的集群中运行海量数据的分布式计算，是谷歌实现云计算的重要基石。本书分为 3 个部分，深入浅出地介绍了 Hadoop 框架、编写和运行 Hadoop 数据处理程序所需的实践技能及 Hadoop 之外更大的生态系统。

本书适合需要处理大量离线数据的云计算程序员、架构师和项目经理阅读参考。

www.ChinaDBA.net 中国DBA超级论坛

图灵程序设计丛书

Hadoop实战

◆ 著　[美] Chuck Lam
译　韩冀中
责任编辑　卢秀丽

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市潮河印业有限公司印刷

◆ 开本：800×1000　1/16
印张：16.75
字数：417千字　2011年 10 月第 1 版
印数：1－5 000册　2011年 10 月河北第 1 次印刷

著作权合同登记号　图字：01-2011-0806 号

ISBN 978-7-115-26448-0

定价：59.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

www.ChinaDBA.net 中国DBA超级论坛

前　言

我很长时间里都痴迷于数据。当我还是一名电气工程本科生的时候，数字信号处理就对我产生了极大的吸引力。我发现音乐、视频、照片和很多其他的东西都可以被视为数据。数据的计算不断带来并加强了这些感性的体验。我当时认为那是最酷的事情。

随着时间的推移，我继续为数据所展现的崭新视野而欣喜。近几年社交数据和大数据崭露头角。特别是大数据，它对我而言是一个智力挑战。早先我已经学会了从统计学角度来观察数据，新的数据类型“只是”需要新的数学方法。这并不简单，但至少我已经得到过训练，了解它们所需的资源也非常丰富。另一方面，大数据涉及系统级的创新和新的编程方法。我从未得到过这样的训练，更重要的是，不只我一个人如此。有关在实践中处理大数据的知识在一定程度上是一种魔法。许多用于扩展数据处理的工具和技术都是如此，包括缓存（例如memcached）、复制及分片，当然还有MapReduce/Hadoop。近几年，我的时间都花在不断地学习这些技术上。

从个人经历看，学习这些技术最大的障碍出现在学习过程的中段。开始时，很容易找到引导性的博客和演示文稿，它们会教你如何做一个“Hello World”的示例。当足够熟悉之后，你就会知道如何在邮件列表中提问，在大小会议中邂逅专家，甚至自己阅读源代码。但在这中间存在一个巨大的知识落差，你的胃口更大了，但又不太清楚下一步该问什么问题。对Hadoop这种最新的技术而言尤为如此。需要一个有组织的说明，将你从开始的“Hello World”引领到可以从容地在实践中应用Hadoop。这就是我希望本书所做到的。幸好我发现了Manning出版社的In Action系列丛书，它们正与此目标相吻合，而且出版社有一群优秀的编辑帮助我达成目标。

我非常享受写作这本书的时光，希望它能为你开启畅游Hadoop的旅途。

www.ChinaDBA.net 中国DBA超级论坛

致　　谢

很多人为这本书提供了灵感并做出了奉献。首先我要感谢James Warren。他将分析引入RockYou，我们一起在公司上下灌输了Hadoop的使用。我从他身上学到了很多东西，他甚至还为初稿出谋划策。

我很幸运有很多人为我提供了Web 2.0行业以外的有趣案例。为此我要感谢罗治国、徐萌、孙少陵、Ken MacInnis、Ryan Rawson、Vuk Ercegovac、Rajasekar Krishnamurthy、Sriram Raghavan、Frederick Reiss、Eugene Shekita、Sandeep Tata、Shivakumar Vaithyanathan以及Huaiyu Zhu。

我也要感谢这本书的许多审阅者。他们为早期的初稿提供了有价值的反馈意见。特别是Paul O'Rorke，他虽是技术审阅人，但却提出了许多超出其职责的中肯建议，告知我如何让这份手稿更为出色。我期待有朝一日能够看到由他自己写的书。我也很享受与Jonathan Cao的长期交谈。他在数据库和大规模系统上的专业知识，为我理解Hadoop的功能提供了广阔的视野。

其他审阅者在此期间对草稿做了大量的反复阅读，多谢他们宝贵的意见，他们是：Paul Stusiak、Philipp K. Janert、Amin Mohammed-Coleman、John S. Griffin、Marco Ughetti、Rick Wagner、Kenneth DeLong、Josh Patterson、Srini Penchikala、Costantino Cerbo、Steve Loughran、Ara Abrahamian、Ben Hall、Andrew Siemer、Robert Hanson、Keith Kim、Sopan Shewale、Marion Sturtevant、Chris Chandler、Eric Raymond以及Jeroen Benckhuijsen。

我很幸运与Manning出版社很出色的一群人工作。特别感谢Troy Mott让我开始本书的撰写工作，并有足够耐心等待我把它完成。也多亏Tara Walsh、Karen Tegtmeyer、Marjan Bace、Mary Piergies、Cynthia Kane、Steven Hong、Rachel Schroeder、Katie Tennant以及Maureen Spencer。他们的支持是了不起的。我想象不出比这更好的工作团队。

不用说，所有对Hadoop及其生态系统做出贡献的人都值得称赞。Doug Cutting发起了它，Yahoo颇具远见地最早支持它。Cloudera现在将Hadoop推广给更多的企业用户。加入成长中的Hadoop社区令人兴奋不已。

最后但重要的是，我要感谢所有的朋友、家人和同事在我编写这本书时给我的支持。

www.ChinaDBA.net 中国DBA超级论坛

作者在线

购买本书可以免费访问 Manning 出版社的内部论坛，在那里可以对这本书进行评论、提出技术问题，并从作者和其他用户那里获得帮助。你可以通过网址 www.manning.com/HadoopinAction 访问并注册论坛。在注册之后，该页面会为你提供如何进入论坛、可以获得的帮助以及论坛的行为规则等信息。

Manning 出版社承诺在读者之间，以及读者和作者之间建立有意义的对话平台。这种承诺并不包含作者的参与，作者在论坛上所作的贡献依然是自愿的（且是无偿的）。我们建议你尝试向作者问一些有挑战性的问题，免得让他兴趣索然！

只要这本书在出版，作者在线论坛和先前的讨论文档都可通过出版商的网站进行访问。

www.ChinaDBA.net 中国DBA超级论坛

关于本书

Hadoop是一个开源框架，它遵循谷歌的方法实现了MapReduce算法，用以查询在互联网上分布的数据集。这个定义自然会导致一个明显的问题：什么是map（映射），为什么它们需要被reduce（归约）？使用传统机制分析和查询大规模数据集会非常困难，当查询自身很复杂时尤为如此。实际上，MapReduce算法将查询操作和数据集都分解为组件——这就是映射。在查询中被映射的组件可以被同时处理（即归约）从而快速地返回结果。

这本书教会读者如何使用Hadoop并编写MapReduce程序。目标读者为不得不离线处理大量数据的程序员、架构师和项目经理。这本书引导读者去获得Hadoop的一个副本、在集群中安装并编写数据分析程序。

在书的开篇，为了让Hadoop和MapReduce的基本理念更易于掌握，本书在Hadoop的默认安装上运行了几个易于理解的任务，例如文档正文中词频变化的分析。然后在使用Hadoop开发MapReduce应用的过程中，探究其基本概念，包括仔细观察框架的组成、Hadoop在各种数据分析中的使用以及Hadoop实战中的大量实例。

MapReduce是一个在概念和实现上都很复杂的想法，要了解运行Hadoop的方方面面对于用户而言是一个挑战。本书除带给你Hadoop的运行机理之外，还会教你在MapReduce框架下写出有意义的程序。

本书假定读者基本掌握了Java，因为大多数代码示例是用Java写的。熟悉基本的统计学概念（如直方图和相关）将有助于读者理解更高级的数据处理示例。

路线图

本书将12章划分为3个部分。

第一部分的3章介绍了Hadoop的框架，涵盖我们理解并使用Hadoop所需的基础知识。这些章节描述了构成一个Hadoop集群的硬件组件，以及建立一个可运行系统的安装及配置方法。第一部分还从高层描述了MapReduce框架，并让你能编写和运行第一个MapReduce程序。

第二部分包含5章，给出编写和运行Hadoop数据处理程序所需的实践技能。在这些章节中，我们将探讨使用Hadoop分析专利数据集的各种实例，包括Bloom filter这样的先进算法。我们还将给出对生产环境下使用Hadoop极其有用的编程和管理技术。

第三部分被称为“Hadoop也疯狂”，包含这本书的最后4章，将探讨Hadoop之外更大的生态系统。云服务提供了创建Hadoop集群的另一种方案，可以替代那种由自己购买并拥有硬件集群的

www.ChinaDBA.net 中国DBA超级论坛

方式。许多附加产品包在MapReduce之上提供了更高级别的编程抽象。最后，我们会看到几个用Hadoop解决实际业务问题的案例。

附录包含HDFS命令的列表及其说明和使用方法。

编码约定及代码下载

所有列表或文本中的源代码都是用固定宽度字体与普通文本相区别的。许多代码清单中都给出了代码注释，重要的概念被突出地显示。有时，随代码清单还会给出由数字符号相连的注释。

本书的示例代码可从Manning出版社的网站www.manning.com/HadoopinAction上下载。

www.ChinaDBA.net 中国DBA超级论坛

关于封面图

本书封面上的图为“一个来自达尔马提亚Kistanja的年轻人”。该图取自克罗地亚19世纪中叶传统服饰影集的一个副本，作者为尼古拉·阿尔塞诺维奇，由Ethnographic博物馆在2003年于克罗地亚的斯普利特出版。该图得自于一位乐于助人的斯普利特Ethnographic博物馆馆员，这个博物馆位于该城镇在中世纪罗马时的核心位置，是公元304年左右罗马皇帝戴克里先的宫殿遗址。这本书包含来自克罗地亚不同地域的颜色精美的插图，附有服饰和日常生活的说明。

Kistanja是一个小镇，位于克罗地亚的布科维卡地区。它坐落在达尔马提亚北部，有悠久的罗马和威尼斯的历史。在克罗地亚，“mamok”一词是指单身汉、花花公子或求婚者（一个在求爱年龄的年轻男人），在封面上的这个年轻人看起来干净利落，很明显正穿着他最好的衣服，小巧玲珑的白色亚麻衬衫，色彩鲜艳的绣花背心，这样的衣服他只有在去教堂和节日时才会穿——或者是去约会一位年轻女士。

过去200年间，着装和生活方式已经发生变化，曾经如此丰富的地域多样性已渐渐消失了。现在，各大洲的居民已经很难分辨，更遑论分隔只有几英里的不同村庄或城镇的人。也许我们用文化差异换来了一个更丰富的个人生活——必然是更为多样和快节奏的技术生活。

Manning出版社取材此类古老书籍中的插图，用两个世纪前丰富多样的地域生活来制作书的封面，用以庆贺计算机行业的创造性和主动性。

www.ChinaDBA.net 中国DBA超级论坛

目　　录

第一部分　Hadoop——一种分布式编程框架

第 1 章　Hadoop简介 ……2
1.1　为什么写《Hadoop 实战》 ……3
1.2　什么是 Hadoop ……3
1.3　了解分布式系统和 Hadoop ……4
1.4　比较 SQL 数据库和 Hadoop ……5
1.5　理解 MapReduce ……6
1.5.1　动手扩展一个简单程序 ……7
1.5.2　相同程序在MapReduce中的扩展 ……9
1.6　用Hadoop统计单词——运行第一个程序 ……11
1.7　Hadoop历史 ……15
1.8　小结 ……16
1.9　资源 ……16

第 2 章　初识Hadoop ……17
2.1　Hadoop 的构造模块 ……17
2.1.1　NameNode ……17
2.1.2　DataNode ……18
2.1.3　Secondary NameNode ……19
2.1.4　JobTracker ……19
2.1.5　TaskTracker ……19
2.2　为 Hadoop 集群安装 SSH ……21
2.2.1　定义一个公共账号 ……21
2.2.2　验证SSH安装 ……21
2.2.3　生成SSH密钥对 ……21
2.2.4　将公钥分布并登录验证 ……22
2.3　运行 Hadoop ……22
2.3.1　本地（单机）模式 ……23
2.3.2　伪分布模式 ……24
2.3.3　全分布模式 ……25
2.4　基于 Web 的集群用户界面 ……28
2.5　小结 ……30

第 3 章　Hadoop组件 ……31
3.1　HDFS 文件操作 ……31
3.1.1　基本文件命令 ……32
3.1.2　编程读写HDFS ……35
3.2　剖析 MapReduce 程序 ……37
3.2.1　Hadoop数据类型 ……39
3.2.2　Mapper ……40
3.2.3　Reducer ……41
3.2.4　Partitioner：重定向Mapper输出 ……41
3.2.5　Combiner：本地reduce ……43
3.2.6　预定义mapper和Reducer类的单词计数 ……43
3.3　读和写 ……43
3.3.1　InputFormat ……44
3.3.2　OutputFormat ……49
3.4　小结 ……50

第二部分　实战

第 4 章　编写MapReduce基础程序 ……52
4.1　获得专利数据集 ……52
4.1.1　专利引用数据 ……53
4.1.2　专利描述数据 ……54
4.2　构建 MapReduce 程序的基础模板 ……55
4.3　计数 ……60

www.ChinaDBA.net 中国DBA超级论坛

4.4 适应 Hadoop API 的改变……64
4.5 Hadoop 的 Streaming……67
4.5.1 通过Unix命令使用Streaming……68
4.5.2 通过脚本使用Streaming……69
4.5.3 用Streaming处理键/值对……72
4.5.4 通过Aggregate包使用Streaming……75
4.6 使用 combiner 提升性能……80
4.7 温故知新……83
4.8 小结……84
4.9 更多资源……84

第 5 章 高阶MapReduce……85
5.1 链接 MapReduce 作业……85
5.1.1 顺序链接MapReduce作业……85
5.1.2 具有复杂依赖的MapReduce链接……86
5.1.3 预处理和后处理阶段的链接……86
5.2 联结不同来源的数据……89
5.2.1 Reduce侧的联结……90
5.2.2 基于 DistributedCache 的复制联结……98
5.2.3 半联结：map侧过滤后在reduce侧联结……101
5.3 创建一个 Bloom filter……102
5.3.1 Bloom filter做了什么……102
5.3.2 实现一个Bloom filter……104
5.3.3 Hadoop 0.20 以上版本的 Bloom filter……110
5.4 温故知新……110
5.5 小结……111
5.6 更多资源……112

第 6 章 编程实践……113
6.1 开发 MapReduce 程序……113
6.1.1 本地模式……114
6.1.2 伪分布模式……118
6.2 生产集群上的监视和调试……123
6.2.1 计数器……123
6.2.2 跳过坏记录……125
6.2.3 用IsolationRunner重新运行出错的任务……128
6.3 性能调优……129
6.3.1 通过combiner来减少网络流量……129
6.3.2 减少输入数据量……129
6.3.3 使用压缩……129
6.3.4 重用JVM……132
6.3.5 根据猜测执行来运行……132
6.3.6 代码重构与算法重写……133
6.4 小结……134

第 7 章 细则手册……135
7.1 向任务传递作业定制的参数……135
7.2 探查任务特定信息……137
7.3 划分为多个输出文件……138
7.4 以数据库作为输入输出……143
7.5 保持输出的顺序……145
7.6 小结……146

第 8 章 管理Hadoop……147
8.1 为实际应用设置特定参数值……147
8.2 系统体检……149
8.3 权限设置……151
8.4 配额管理……151
8.5 启用回收站……152
8.6 删减 DataNode……152
8.7 增加 DataNode……153
8.8 管理 NameNode 和 SNN……153
8.9 恢复失效的 NameNode……155
8.10 感知网络布局和机架的设计……156
8.11 多用户作业的调度……157
8.11.1 多个JobTracker……158
8.11.2 公平调度器……158
8.12 小结……160

第三部分 Hadoop也疯狂

第 9 章 在云上运行Hadoop……162
9.1 Amazon Web Services 简介……162
9.2 安装 AWS……163

www.ChinaDBA.net 中国DBA超级论坛

9.2.1 获得AWS身份认证凭据 ……164
9.2.2 获得命令行工具 ……166
9.2.3 准备SSH密钥对 ……168
9.3 在 EC2 上安装 Hadoop ……169
9.3.1 配置安全参数 ……169
9.3.2 配置集群类型 ……169
9.4 在 EC2 上运行 MapReduce 程序 ……171
9.4.1 将代码转移到Hadoop集群上 ……171
9.4.2 访问Hadoop集群上的数据 ……172
9.5 清空和关闭 EC2 实例 ……175
9.6 Amazon Elastic MapReduce 和其他 AWS 服务 ……176
9.6.1 Amazon Elastic MapReduce ……176
9.6.2 AWS导入/导出 ……177
9.7 小结 ……177
第 10 章 用Pig编程 ……178
10.1 像 Pig 一样思考 ……178
10.1.1 数据流语言 ……179
10.1.2 数据类型 ……179
10.1.3 用户定义函数 ……179
10.2 安装 Pig ……179
10.3 运行 Pig ……180
10.4 通过 Grunt 学习 Pig Latin ……182
10.5 谈谈 Pig Latin ……186
10.5.1 数据类型和schema ……186
10.5.2 表达式和函数 ……187
10.5.3 关系型运算符 ……189
10.5.4 执行优化 ……196
10.6 用户定义函数 ……196
10.6.1 使用UDF ……196
10.6.2 编写UDF ……197
10.7 脚本 ……199
10.7.1 注释 ……199
10.7.2 参数替换 ……200
10.7.3 多查询执行 ……201
10.8 Pig 实战——计算相似专利的例子 ……201
10.9 小结 ……206
第 11 章 Hive及Hadoop群 ……207
11.1 Hive ……207
11.1.1 安装与配置Hive ……208
11.1.2 查询的示例 ……210
11.1.3 深入HiveQL ……213
11.1.4 Hive小结 ……221
11.2 其他 Hadoop 相关的部分 ……221
11.2.1 HBase ……221
11.2.2 ZooKeeper ……221
11.2.3 Cascading ……221
11.2.4 Cloudera ……222
11.2.5 Katta ……222
11.2.6 CloudBase ……222
11.2.7 Aster Data和Greenplum ……222
11.2.8 Hama和Mahout ……223
11.2.9 search-hadoop.com ……223
11.3 小结 ……223
第 12 章 案例研究 ……224
12.1 转换《纽约时报》1100 万个库存图片文档 ……224
12.2 挖掘中国移动的数据 ……225
12.3 在 StumbleUpon 推荐最佳网站 ……229
12.3.1 分布式 StumbleUpon 的开端 ……230
12.3.2 HBase 和 StumbleUpon ……230
12.3.3 StumbleUpon 上的更多 Hadoop 应用 ……236
12.4 搭建面向企业查询的分析系统——IBM 的 ES2 项目 ……238
12.4.1 ES2 系统结构 ……240
12.4.2 ES2 爬虫 ……241
12.4.3 ES2 分析 ……242
12.4.4 小结 ……249
12.4.5 参考文献 ……250
附录A HDFS文件命令 ……251

www.ChinaDBA.net 中国DBA超级论坛

Part 1 第一部分

Hadoop——一种分布式编程框架

本书第一部分所介绍的是理解与使用 Hadoop 的基础。首先描述 Hadoop 集群的硬件构成及系统安装与配置，然后从高层次阐述 MapReduce 框架，并让你的第一个 MapReduce 程序运行起来。

本部分内容

- 第 1 章　Hadoop 简介
- 第 2 章　初识 Hadoop
- 第 3 章　Hadoop 组件

www.ChinaDBA.net 中国DBA超级论坛

第1章 Hadoop简介

1

本章内容

- 编写可扩展、分布式的数据密集型程序的基础知识
- 理解Hadoop和MapReduce
- 编写和运行一个基本的MapReduce程序

今天，我们正被数据所包围。人们上载视频、用手机照相、发短信给朋友、更新Facebook、网上留言及点击广告等，这使得机器产生和保留了越来越多的数据。你甚至此时此刻可能正在自己的电脑屏幕上阅读本书的电子版，并且可以确定的是，你在书店购买本书的记录已经被存为数据。[1]

数据的指数级增长首先向谷歌、雅虎、亚马逊和微软等这些处于市场领导地位的公司提出了挑战。它们需要遍历TB级和PB级数据来发现哪些网站更受欢迎，哪些书有需求，哪种广告吸引人。现有工具正变得无力处理如此大的数据集。谷歌率先推出了MapReduce，这是个用来应对其数据处理需求的系统。这个系统引起了广泛的关注，因为许多其他的企业同样面临数据膨胀的挑战，并且不是每个人都能够为自己重新量身订制一个专有的工具。Doug Cutting看到了机会并且领导开发了一个开源版本的MapReduce，称为Hadoop。随后，雅虎等公司纷纷响应，为其提供支持。今天，Hadoop已经成为许多互联网公司基础计算平台的一个核心部分，如雅虎、Facebook、LinkedIn和Twitter。许多传统的行业，如传媒业和电信业，也正在开始采用这个系统。我们将在第12章的案例中介绍《纽约时报》、中国移动和IBM等公司如何使用Hadoop。

Hadoop及大规模分布式数据处理，正在迅速成为许多程序员的一项重要技能。关系数据库、网络和安全这些在几十年前被认为是程序员可选技能的知识，今天已经成为一个高效程序员的必修课。同样，基本理解分布式数据处理将很快成为每个程序员的工具箱中不可或缺的一部分。斯坦福和卡内基-梅隆等一流的大学已经开始将Hadoop引入他们的计算机科学课程。这本书将会帮助你，一名执业的程序员，快速掌握Hadoop并用它来处理你的数据集。

---

① 当然，你读的是本正版书，对吗？

www.ChinaDBA.net 中国DBA超级论坛

本章正式介绍Hadoop，找出它在分布式系统和数据处理系统方面的定位，并概述MapReduce编程模型。我们基于现有工具实现一个简单的单词统计示例，来彰显大型数据处理的挑战。然后，在使用Hadoop实现该示例之后，你会深刻体会Hadoop的简洁明了。我们还将讨论Hadoop的历史以及人们对MapReduce范式的一些观点。不过，让我先简单介绍一下为什么我写这本书，以及它为什么对你有用。

1.1 为什么写《Hadoop 实战》

实话实说，我第一次接触Hadoop即被其强大的能力所吸引，但随后在编写基本例程时却经历了一段令人沮丧的过程。虽然Hadoop官方网站上的文档相当全面，但是为简单的疑问找到直截了当的解答却并不总是那么容易。

写作本书的目的就是要解决这个问题。我不会关注过多的细节，相反，我提供的信息会有助于你快速创建可用代码，并会涉及在实践中最常遇到的更高级的话题。

1.2 什么是 Hadoop

按照正式的定义，Hadoop是一个开源的框架，可编写和运行分布式应用处理大规模数据。分布式计算是一个宽泛并且不断变化的领域，但Hadoop与众不同之处在于以下几点。

- 方便——Hadoop运行在由一般商用机器构成的大型集群上，或者如亚马逊弹性计算云（EC2）等云计算服务之上。
- 健壮——Hadoop致力于在一般商用硬件上运行，其架构假设硬件会频繁地出现失效。它可以从容地处理大多数此类故障。
- 可扩展——Hadoop通过增加集群节点，可以线性地扩展以处理更大的数据集。
- 简单——Hadoop允许用户快速编写出高效的并行代码。

Hadoop的方便和简单让其在编写和运行大型分布式程序方面占尽优势。即使是在校的大学生也可以快速、廉价地建立自己的Hadoop集群。另一方面，它的健壮性和可扩展性又使它胜任雅虎和Facebook最严苛的工作。这些特性使Hadoop在学术界和工业界都大受欢迎。

图1-1解释了如何与Hadoop集群交互。Hadoop集群是在同一地点用网络互连的一组通用机器。数据存储和处理都发生在这个机器“云”中[①]。不同的用户可以从独立的客户端提交计算“作业”到Hadoop，这些客户端可以是远离Hadoop集群的个人台式机。

并非所有分布式系统的构建都如图1-1所示的一样。下面，我们简要介绍一下其他的分布式系统，以便更好地展现Hadoop所依据的设计理念。

① 虽非绝对必要，但通常在一个Hadoop集群中的机器都是相对同构的x86 Linux服务器。而且它们几乎总是位于同一个数据中心，并通常在同一组机架里。

www.ChinaDBA.net 中国DBA超级论坛

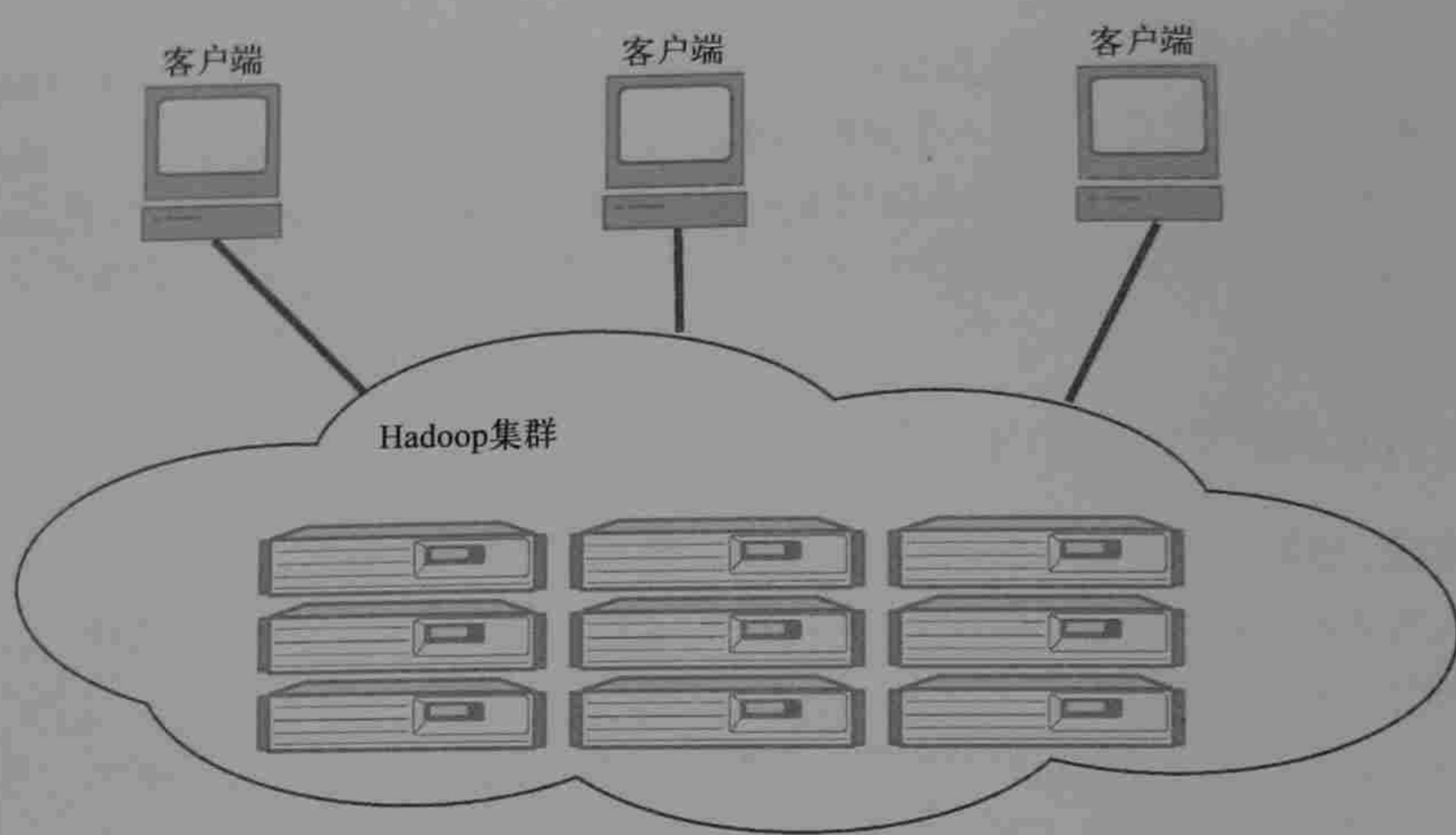


图1-1　一个Hadoop集群拥有许多并行的计算机，用以存储与处理大规模数据集。客户端计算机发送作业到计算云并获得结果

1.3　了解分布式系统和 Hadoop

摩尔定律在过去几十年间对我们都是适用的，但解决大规模计算问题却不能单纯依赖于制造越来越大型的服务器。有一种替代方案已经获得普及，即把许多低端/商用的机器组织在一起，形成一个功能专一的分布式系统。

为了理解盛行的分布式系统（俗称向外扩展）与大型单机服务器（俗称向上扩展）之间的对比，需要考虑现有I/O技术的性价比。对于一个有4个I/O通道的高端机，即使每个通道的吞吐量各为100 MB/sec，读取4 TB的数据集也需要3个小时！而利用Hadoop，同样的数据集会被划分为较小的块（通常为64 MB），通过Hadoop分布式文件系统（HDFS）分布在集群内多台机器上。使用适度的复制，集群可以并行读取数据，进而提供很高的吞吐量。而这样一组通用机器比一台高端服务器更加便宜！

前面的解释充分展示了Hadoop相对于单机系统的效率。现在让我们将Hadoop与其他分布式系统架构进行比较。一个众所周知的方法是SETI @ home，它利用世界各地的屏保来协助寻找外星生命。在SETI @ home，一台中央服务器存储来自太空的无线电信号，并在网上发布给世界各地的客户端台式机去寻找异常的迹象。这种方法将数据移动到计算即将发生的地方（桌面屏保）。经过计算后，再将返回的数据结果存储起来。

Hadoop在对待数据的理念上与SETI@home等机制不同。SETI @ home需要客户端和服务器之间重复地传输数据。这虽能很好地适应计算密集型的工作，但处理数据密集型任务时，由于数据规模太大，数据搬移变得十分困难。Hadoop强调把代码向数据迁移，而不是相反。参考图1-1，我们看到Hadoop的集群内部既包含数据又包含计算环境。客户端仅需发送待执行的MapReduce程序，而这些程序一般都很小（通常为几千字节）。更重要的是，代码向数据迁移的理念被应用

www.ChinaDBA.net 中国DBA超级论坛

在Hadoop集群自身。数据被拆分后在集群中分布，并且尽可能让一段数据的计算发生在同一台机器上，即这段数据驻留的地方。

这种代码向数据迁移的理念符合Hadoop面向数据密集型处理的设计目标。要运行的程序（“代码”）在规模上比数据小几个数量级，更容易移动。此外，在网络上移动数据要比在其上加载代码更花时间。不如让数据不动，而将可执行代码移动到数据所在的机器上去。

现在你知道Hadoop是如何契合分布式系统的设计了，那就让我们看看它和通常的数据处理系统（SQL数据库）比较会怎么样。

1.4 比较 SQL 数据库和 Hadoop

鉴于Hadoop是一个数据处理框架，而在当前大多数应用中数据处理的主力是标准的关系数据库，那又是什么使得Hadoop更具优势呢？其中一个原因是，SQL（结构化查询语言）是针对结构化数据设计的，而Hadoop最初的许多应用针对的是文本这种非结构化数据。从这个角度来看，Hadoop比SQL提供了一种更为通用的模式。

若只针对结构化数据处理，则需要做更细致的比较。原则上，SQL和Hadoop可以互补，因为SQL是一种查询语言，它可将Hadoop作为其执行引擎[①]。但实际上，SQL数据库往往指代一整套传统技术，通过几个主要的厂商，面向一组历史悠久的应用进行优化。许多这些现有的商业数据库无法满足Hadoop设计所面向的需求。

考虑到这一点，让我们从特定的视角将Hadoop与典型SQL数据库做更详细的比较。

1. 用向外扩展代替向上扩展

扩展商用关系型数据库的代价是非常昂贵的。它们的设计更容易向上扩展。要运行一个更大的数据库，就需要买一个更大的机器。事实上，往往会看到服务器厂商在市场上将其昂贵的高端机标称为“数据库级的服务器”。不过有时可能需要处理更大的数据集，却找不到一个足够大的机器。更重要的是，高端的机器对于许多应用并不经济。例如，性能4倍于标准PC的机器，其成本将大大超过将同样的4台PC放在一个集群中。Hadoop的设计就是为了能够在商用PC集群上实现向外扩展的架构。添加更多的资源，对于Hadoop集群就是增加更多的机器。一个Hadoop集群的标配是十至数百台计算机。事实上，如果不是为了开发目的，没有理由在单个服务器上运行Hadoop。

2. 用键/值对代替关系表

关系数据库的一个基本原则是让数据按某种模式存放在具有关系型数据结构的表中。虽然关系模型具有大量形式化的属性，但是许多当前的应用所处理的数据类型并不能很好地适合这个模型。文本、图片和XML文件是最典型的例子。此外，大型数据集往往是非结构化或半结构化的。Hadoop使用键/值对作为基本数据单元，可足够灵活地处理较少结构化的数据类型。在Hadoop中，数据的来源可以有任何形式，但最终会转化为键/值对以供处理。

3. 用函数式编程（MapReduce）代替声明式查询（SQL）

SQL从根本上说是一个高级声明式语言。查询数据的手段是，声明想要的查询结果并让数据

① 其实这是Hadoop社区中的一个热点领域，我们将在第11章讨论其中一些领先的项目。

www.ChinaDBA.net 中国DBA超级论坛

库引擎判定如何获取数据。在MapReduce中，实际的数据处理步骤是由你指定的，它很类似于SQL引擎的一个执行计划。SQL使用查询语句，而MapReduce则使用脚本和代码。利用MapReduce可以用比SQL查询更为一般化的数据处理方式。例如，你可以建立复杂的数据统计模型，或者改变图像数据的格式。而SQL就不能很好地适应这些任务。

另一方面，当数据处理非常适合于关系型数据结构时，有些人可能会发现使用MapReduce并不自然。那些习惯于SQL范式的人可能会发现用MapReduce来思考是一个挑战。我希望本书中的练习和示例能帮你更轻松地掌握MapReduce编程。不过值得注意的是，这里还有很多扩展可用，便于人们采用更熟悉的范式来编程，同时拥有Hadoop的可扩展性优势。事实上，使用某些扩展可采用一种类似SQL的查询语言，并自动将查询编译为可执行的MapReduce代码。我们将在第10章和第11章介绍其中的一些工具。

4. 用离线批量处理代替在线处理

Hadoop是专为离线处理和大规模数据分析而设计的，它并不适合那种对几个记录随机读写的在线事务处理模式。事实上，在本书写作时（以及在可预见的未来），Hadoop最适合一次写入、多次读取的数据存储需求。在这方面它就像SQL世界中的数据仓库。

你已经从宏观上看到Hadoop与分布式系统和SQL数据库之间的关系。那么，让我们开始学习如何用它来编程。为此，我们首先需要了解Hadoop中的MapReduce范式。

1.5　理解 MapReduce

你也许知道管道和消息队列等数据处理模型。这些模型可专用于数据处理应用的方方面面。Unix pipes就是一种最常见的管道。管道有助于进程原语的重用，已有模块的简单链接即可组成一个新的模块；消息队列则有助于进程原语的同步。程序员将数据处理任务以生产者或消费者的形式编写为进程原语，由系统来管理它们何时执行。

同样，MapReduce也是一个数据处理模型，它最大的优点是容易扩展到多个计算节点上处理数据。在MapReduce模型中，数据处理原语被称为mapper和reducer。分解一个数据处理应用为mapper和reducer有时是繁琐的，但是一旦以MapReduce的形式写好了一个应用程序，仅需修改配置就可以将它扩展到集群中几百、几千，甚至几万台机器上运行。正是这种简单的可扩展性使得MapReduce模型吸引了众多程序员。

MapReduce的几种写法

尽管已有不少关于MapReduce的论述，但并没有一个统一的写法。原始的谷歌文章和维基百科条目上使用驼峰大小写方式写为MapReduce。然而，谷歌自己在其网站的一些网页上采用了Map Reduce的写法（例如，http://research.google.com/roundtable/MR.html）。在Hadoop官方网站文档中，可以找到一个指向Map-Reduce Tutorial的链接。点击该链接，会进入一个名为Hadoop Map/Reduce Tuotorial的页面(http://hadoop.apache.org/core/docs/current/ mapred_tutorial.html)，其中是对Map/Reduce框架的解释。写法的差别也存在于不同的Hadoop组件，如NameNode (name node、name-node及namenode)、DataNode、JobTracker和TaskTracker。为了统一起见，我们在本书中全部采用驼峰大小写方式。（即MapReduce、NameNode、DataNode、JobTracker和 TaskTracker。）

www.ChinaDBA.net 中国DBA超级论坛

1.5.1 动手扩展一个简单程序

在正式论述MapReduce之前，让我们先做一个练习，即扩展一个简单的程序让其处理一大段数据。然后，你会认识到扩展数据处理程序所面对的挑战，从而更好地体会MapReduce这种框架帮助你处理繁琐事务时的好处。

我们的练习是统计一组文档中的每个单词出现的次数。在这个例子中，我们的文档仅有一个文件，文件中只有一句话：

Do as I say, not as I do.

我们得到如右图所示单词统计的值。

单　词	计　数
as	2
do	2
i	2
not	1
say	1

我们将这个特定的练习称为单词统计。如果文档集合很小，一个简单的程序即可完成这项工作。可写为如下一段伪代码：

```
define wordCount as Multiset;
for each document in documentSet {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
display(wordCount);
```

该程序循环遍历所有的文档。对于每个文档，使用分词过程逐个地提取单词。对于每个单词，在多重集合wordCount中的相应项上加1。最后，display()函数打印出wordCount中的所有条目。

注意 多重集合中每个元素都有一个计数值。我们试图产生的单词统计是一个多重集合的典型例子。实际上，它通常用一个散列表来实现。

这个程序只适合处理少量文档，一旦文档数量激增，它就不能胜任了。例如，你想编写一个垃圾邮件过滤器，来获取接收到的几百万封垃圾邮件中经常使用的单词。使用单台计算机反复遍历所有文档将会非常费时。重写程序，让工作可以分布在多台机器上。每台机器处理这些文档的不同部分。当所有的机器都完成时，第二个处理阶段将合并这些结果。第一阶段要分布到多台机器上去的伪代码为：

```
define wordCount as Multiset;
for each document in documentSubset {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
sendToSecondPhase(wordCount);
```

第二阶段的伪代码为：

```
define totalWordCount as Multiset;
```

www.ChinaDBA.net 中国DBA超级论坛

```
for each wordCount received from firstPhase {
    multisetAdd (totalWordCount, wordCount);
}
```

这不是很难，对吗？但是，一些细节可能会妨碍它按预期工作。首先，我们忽略了文档读取的性能需求。如果文件都存在一个中央存储服务器上，那么瓶颈就是该服务器的带宽。让更多的机器参与处理的办法不会一直有效，因为有时存储服务器的性能会跟不上。因此，你需要将文档分开存放，使每台机器可以仅处理自己所存储的文档，从而消除单个中央存储服务器的瓶颈。这呼应了先前的观点，即在数据密集型分布式应用中存储和处理不得不紧密地绑定在一起。

该程序的另一个缺陷是wordCount（和totalWordCount）被存储在内存中。当处理大型文档集时，一个特定单词的数量就会超过一台机器的内存容量。英语有大约一百万个词，这个大小可以很轻松地放进一个iPod，但我们的单词统计程序将处理许多不在标准英语单词词典中的特殊单词。例如，我们必须处理诸如Hadoop这样的特定名称。我们必须统计错字，即使它们并不是真正的单词（例如，exampel），我们还需分别统计一个字的所有不同形式（例如，eat，ate，eaten和eating）。即使在文档中特定单词的数量可以被管理在内存中，在问题的定义上略有变化就可能会引起空间复杂度的爆炸。例如，我们不统计文档中的单词，而是去统计日志文件中的IP地址，或者Bigram的频率。那么我们处理的多重集合条目将达到数十亿，这超过了大多数商用计算机的内存容量。

注意 bigram是一对连续的单词。句子“Do as I say, not as I do”可分为以下的bigram：Do as, as I, I say, say not, not as, as I, I do。类似地，trigram是连续的三组词。bigram和trigram在自然语言处理中都非常重要。

---

wordCount也许无法放在内存中；我们将不得不改写我们的程序，以便在磁盘上存储该散列表。这意味着我们将实现一个基于磁盘的散列表，其中涉及大量的编码。

此外，请记住，第二阶段只有一台计算机，它将处理来自所有计算机在第一阶段计算wordCount的结果。wordCount的处理任务原本就相当繁重。当我们为第一阶段的处理提供充足的计算机时，第二阶段的单台计算机将成为瓶颈。最明显的问题是，我们能否按分布模式重写第二阶段，以便它可以通过增加更多的计算机来实现扩展？

答案是肯定的。为了使第二阶段以分布的方式运转，必须以某种方式将其分割到在多台计算机上，使之能够独立运行。需要在第一阶段之后将wordCount分区，使得第二阶段的每台计算机仅需处理一个分区。举一个例子，假设我们在第二阶段有26台计算机。我们让每台计算机上的wordCount只处理以特定字母开头的单词。例如，计算机A在第二阶段仅统计以字母a开头的单词。为了在第二阶段中实现这种划分，我们需要对第一阶段稍作修改。不再采用基于磁盘的散列表实现wordCount，而是划分出26个表：wordCount-a，wordCount-b等。每个表统计以特定字母开头的单词。经过第一阶段，来自该阶段所有计算机的wordCount-a结果将被发送到第二阶段的计算机A上，所有wordCount-b的结果将被发送到计算机B上，依次类推。第一阶段中的每台计算机都会将结果洗牌到第二阶段的计算机上。

www.ChinaDBA.net 中国DBA超级论坛

这个单词统计程序现在正变得复杂。为了使它工作在一个分布式计算机集群上，我们发现需要添加以下功能。

- 存储文件到许多台计算机上（第一阶段）。
- 编写一个基于磁盘的散列表，使得处理不受内存容量限制。
- 划分来自第一阶段的中间数据（即wordCount）。
- 洗牌这些分区到第二阶段中合适的计算机上。

即使对于单词统计这样简单的程序，这都是繁重的工作，而我们甚至还没有涉及容错等问题。(如果在任务执行过程中一个计算机失效该怎么办？）这就是为什么我们需要一个像Hadoop一样的框架。当你用MapReduce模型来写应用程序，Hadoop将替你管理所有与可扩展性相关的底层问题。

1.5.2 相同程序在 MapReduce 中的扩展

MapReduce程序的执行分为两个主要阶段，为mapping和reducing。每个阶段均定义为一个数据处理函数，分别被称为mapper和reducer。在mapping阶段，MapReduce获取输入数据并将数据单元装入mapper。在reducing阶段，reducer处理来自mapper的所有输出，并给出最终结果。

简而言之，mapper意味着将输入进行过滤与转换，使reducer可以完成聚合。可以发现这和我们在扩展单词统计时的两个阶段惊人地相似。这种相似性并非偶然。MapReduce的设计建立在编写可扩展、分布式程序的丰富经验之上。这种两阶段的设计模式可以在许多程序的扩展中看到，成为MapReduce框架的基础。

上一节扩展分布式的单词统计程序时，我们不得不编写了partitioning和shuffling函数。它们与mapping和reducing一起形成常见的设计模式。但不同的是，partitioning和shuffling并不依赖特殊的数据处理应用，它们是通用的功能，MapReduce框架提供了可在大多数情况下工作的默认实现。

为了让mapping、reducing、partitioning和shuffling（以及几个我们尚未涉及的函数）能够无缝地在一起工作，我们需要在数据的通用结构上达成一致。它需要足够灵活和强大，以适应大多数的数据处理应用。MapReduce使用列表和键/值对作为其主要的数据原语。键与值通常为整数或字符串，但也可以是可忽略的假值，或者是复杂的对象类型。map和reduce函数必须遵循以下对键和值类型的约束。

	输　入	输　出
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	list(<k3, v3>)

在MapReduce框架中编写应用程序就是定制化mapper和reducer的过程。让我们看看完整的数据流。

(1) 应用的输入必须组织为一个键/值对的列表`list(<k1, v1>)`。输入格式可能看起来是不受约束的，但在实际中它非常简洁。用于处理多个文件的输入格式通常为`list(<String filename, String file_content>)`。用于处理日志文件这种大文件的输入格式为`list(<Integer line_number, String log_event>)`。

(2) 含有键/值对的列表被拆分，进而通过调用mapper的map函数对每个单独的键/值对<k1,

www.ChinaDBA.net 中国DBA超级论坛

v1>进行处理。在这里，键k1经常被mapper所忽略。mapper转换每个<k1，v1>对并将之放入<k2，v2>对的列表中。这种转换的细节很大程度上决定了MapReduce程序的行为。值得注意的是，处理键/值对可以采用任意的顺序。而且，这种转换必须是封闭的，使得输出仅依赖于一个单独的键/值对。

对于单词统计，<String filename, String file_content>被输入mapper，而其中的filename被忽略。mapper可以输出一个<String word, Integer count>的列表，但也可以有更简单的形式。我们知道，计数值将在后续的阶段聚合，我们可以输出一个有重复条目的<String word, Integer 1>列表，并让完整的聚合稍后执行。这就是说，在输出列表中，可以出现一次键/值对<"foo", 3>，或者出现 3 次<"foo", 1>。可以看到，后者更容易编程。前者会获得一些性能上的优化，但是在完全掌握MapReduce框架之前，我们先不讨论这些优化。

(3) 所有mapper的输出（在概念上）被聚合到一个包含<k2,v2>对的巨大列表中。所有共享相同k2的对被组织在一起形成一个新的键/值对<k2, list(v2)>。框架让reducer来分别处理每一个被聚合起来的<k2, list(v2)>。回到单词统计的例子，一个文档的map输出的列表中可能出现三次<"foo", 1>，而另一个文档的map输出列表可能出现两次<"foo", 1>。reducer所看到的聚合的对为<"foo", list(1,1,1,1,1)>。在单词统计中，reducer的输出为<"foo", 5>，表示“foo”在文档集合中总计出现的次数。每一个reducer负责不同的单词。MapReduce框架自动搜集所有的<k3,v3>对，并将之写入文件。在单词统计例子中需要注意，k2和k3的数据类型是相同的，v2和v3也是相同的。但并不是所有的数据处理应用都是这样。

让我们基于MapReduce重写单词统计程序，看看所有这些是如何结合在一起的。代码清单1-1给出了伪代码。

代码清单1-1　单词统计中map和reduce函数的伪代码

```
map(String filename, String document) {
    List<String> T = tokenize(document);
    for each token in T {
        emit ((String)token, (Integer) 1);
    }
}
reduce(String token, List<Integer> values) {
    Integer sum = 0;
    for each value in values {
        sum = sum + value;
    }
    emit ((String)token, (Integer) sum);
}
```

以前我们提到map和reduce函数的输出都是列表。如伪代码所示，我们在框架中实际使用了一个特殊的函数emit()来逐个生成列表中的元素。这个emit()函数进一步将程序员从管理一个大列表的工作中解放出来。

这个代码看起来与1.5.1节的类似，而此时它却能够实际地以扩展方式运行。可见，Hadoop使得建立一个可扩展的分布式程序变容易了，对吧？现在让我们将这段伪代码变成一个Hadoop程序。

www.ChinaDBA.net 中国DBA超级论坛

1.6 用 Hadoop 统计单词——运行第一个程序

既然已经了解了Hadoop和MapReduce的框架，让我们把它运行起来。在本章，Hadoop是运行在单机上的，可以是台式机或是笔记本。下一章将为你展示如何在集群上运行Hadoop，以满足实际部署的需要。在单机上运行Hadoop主要是为了完成开发的工作。

Linux是Hadoop公认的开发与生产平台。虽然Windows也可以支持开发模式，但你需要在节点上安装cygwin(http://www-cygwin.com/)来支持shell和Unix脚本。

注意 有许多人声称已经成功地将Hadoop以开发模式运行在其他Unix的变体上，如Solaris和Mac OS X。事实上，如果在笔记本上开发，Hadoop的开发者似乎总是选择苹果的MacBook Pro笔记本，这在Hadoop大会上或者用户组会议上随处可见。

运行Hadoop需要Java1.6或更高版本。Mac用户可以从苹果公司获得，而其他操作系统的用户可以从Sun公司的网站下载最新的JDK，网址为http://java.sun.com/javase/downloads/index.jsp。安装后请记住Java的安装根目录，这在以后会用到。

要安装Hadoop，首先需要从网站http://hadoop.apache.org/core/releases.html上下载最近的稳定版本。在打开发布包后，编辑脚本conf/hadoop-env.sh将JAVA_HOME设置为刚才记下来的Java安装根目录。例如，在Mac OS X中，需要把这一行

```
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

替换为

```
export JAVA_HOME=/Library/Java/Home
```

Hadoop脚本以后会经常使用，让我们不加任何参数运行它，来看一看它的用法文档。输入：

```
bin/hadoop
```

我们得到

```
Usage: hadoop [--config confdir] COMMAND
这里COMMAND为下列其中一个:
namenode -format                          格式化DFS文件系统
secondarynamenode                         运行DFS的第二个namenode
namenode                                  运行DFS的namenode
datanode                                  运行一个DFS的datanode
dfsadmin                                  运行一个DFS的admin 客户端
fsck                                      运行一个DFS文件系统的检查工具
fs                                        运行一个普通的文件系统用户客户端
balancer                                  运行一个集群负载均衡工具
jobtracker                                运行MapReduce的jobTracker节点
pipes                                     运行一个Pipes作业
tasktracker                               运行一个MapReduce的taskTracker节点
job                                       处理MapReduce作业
version                                   打印版本
jar <jar>                                 运行一个jar文件
distcp <srcurl> <desturl>                 递归地复制文件或者目录
```

www.ChinaDBA.net 中国DBA超级论坛

archive -archiveName NAME <src>* <dest>	生成一个Hadoop档案
daemonlog	获取或设置每个daemon的log级别
或CLASSNAME	运行名为CLASSNAME的类大多数命令会在使用w/o参数时打出帮助信息。

本书将涵盖Hadoop的各种命令。而当前我们仅需知道运行一个（java）Hadoop程序的命令为bin/hadoop jar <jar>。就像命令中显示的那样，用Java写的Hadoop程序被打包为jar执行文件。

幸运的是，我们无需先写一个Hadoop程序；默认安装中已经有几个我们可以使用的例程。下面的命令可以列出那些jar文件的例程：

```
bin/hadoop jar hadoop-*-examples.jar
```

你将看到一组已经被打包在Hadoop中的例程，其中一个就是单词统计程序，名为wordcount！这个程序重要的（内部）类显示在代码清单1-2中。我们将看到这个Java程序是如何实现代码清单1-1中单词统计伪代码中map和reduce功能的。我们将修改这个程序，理解如何让其行为发生变化。这里我们假设仅需按部就班地执行Hadoop程序，它就能够正常工作。

不指定任何参数执行wordcount将显示一些有关用法的信息：

```
bin/hadoop jar hadoop-*-examples.jar wordcount
```

显示出的参数列表为

```
wordcount [-m <maps>] [-r <reduces>] <input> <output>
```

唯一的参数是所需分析的文本文档的输入目录（<input>），以及程序填充结果的输出目录（<output>）。为了执行wordcount，我们需要首先生成一个输入目录

```
mkdir input
```

并放一些文档在里面。你可以添加任何的文本文档到这个目录中。为了解释，让我们从http://www.gpoaccess.gov/sou/下载一个2002年的国情咨文并添加进去。我们现在分析它的单词统计并看看结果：

```
bin/hadoop jar hadoop-*-examples.jar wordcount input output
more output/*
```

你将看到在文档中用到的所有单词的一个统计结果，它们按照字母顺序进行排列。鉴于你还没有开始写一行代码就能这样，这相当不错！但是，还需注意在这个wordcount程序中有一些不足。分词完全根据空格而不是根据标点符号，这使得“States”、“States.”和“States:”分别成为单独的单词。大小写也是一样，比如States和states会表示为不同的单词。另外，我们还希望能够略去在文档中仅显示一次或者两次的那些单词。

好在可以得到wordcount的源码，安装后它被放在src/examples/org/apache/hadoop/xamples/WordCount.java中。我们可以根据需要来修改它。首先我们建立一个playground的目录结构并复制这个程序。

```
mkdir playground
mkdir playground/src
mkdir playground/classes
cp src/examples/org/apache/hadoop/examples/WordCount.java
➥ playground/src/WordCount.java
```

www.ChinaDBA.net 中国DBA超级论坛

在修改程序之前，我们在Hadoop框架中先编译和执行这个副本：

```
javac -classpath hadoop-*-core.jar -d playground/classes
➥ playground/src/WordCount.java
jar -cvf playground/wordcount.jar -C playground/classes/ .
```

你必须每次在运行Hadoop命令时删掉输出目录，因为它是自动生成的。

```
bin/hadoop jar playground/wordcount.jar
➥ org.apache.hadoop.examples.WordCount input output
```

再看一下输出目录中的文件。既然我们没有改变任何程序代码，结果应该与以前一样。我们只是编译了自己的副本，而不去运行预编译的版本。

现在我们准备修改WordCount来增加额外的功能。代码清单1-2显示了WordCount.java程序的一部分，去掉了注释和支撑性代码。

代码清单1-2 WordCount.java

```
public class WordCount extends Configured implements Tool {

  public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  ...
}
```

❶ 使用空格进行分词

❷ 把Token放入Text对象中[1]

❸ 输出每个Token的统计结果

① Token是分词的基本单位，以单词切分则每个单词为一个Token。——译者注

www.ChinaDBA.net 中国DBA超级论坛

WordCount.java和我们的MapReduce伪代码之间主要的功能区别为，在WordCount.java中map()一次处理一行文本，而伪代码一次处理一个文档。这种区别仅通过观察WordCount.java可能并不容易看出来，因为这是Hadoop的默认配置。

代码清单1-2中的代码和代码清单1-1中的伪代码几乎是一样的，只是Java的语法让代码更加冗长。Map和Reduce函数是在WordCount的内部类中。你可能注意到它们是一些特殊的类，如LongWritable、IntWritable和Text，而不是更为常见的Java类Long、Integer和String。现在可以停下来想一想这些实现的细节。这些新类增加了Hadoop内部所需的串行化能力。

很容易找到我们要对程序进行修改的地方。我们看到在❶的位置上WordCount以默认配置使用了Java的StringTokenizer，这里仅基于空格来分词。为了在分词过程中忽略标准的标点符号，我们将它们加入到StringTokenizer的定界符列表中。

```
StringTokenizer itr = new StringTokenizer(line, " \t\n\r\f,.:;?![]'");
```

当遍历token的集合时，每个token被提取出来并放进一个Text对象❷。（在Hadoop中，特殊的Text类取代了String。）因为希望单词统计忽略大小写，我们在把它们转换为Text对象前先将所有的单词都变成小写。

```
word.set(itr.nextToken().toLowerCase());
```

最后，我们希望仅仅显示出现次数大于4次的单词。我们修改❸来使得输出结果仅搜集符合条件的单词统计。（这等价于在Hadoop中实现伪代码中的emit()）。

```
if (sum > 4) output.collect(key, new IntWritable(sum));
```

在修改了上述3行后，可以重新编译程序并执行，结果显示在表1-1中。

表1-1　2002年国情咨文中出现频次大于4的单词统计

11th (5)	citizens (9)	its (6)	over (6)	to (123)
a (69)	congress (10)	jobs (11)	own (5)	together (5)
about (5)	corps (6)	join (7)	page (7)	tonight (5)
act (7)	country (10)	know (6)	people (12)	training (5)
afghanistan (10)	destruction (5)	last (6)	protect (5)	united (6)
all (10)	do (6)	lives (6)	regime (5)	us (6)
allies (8)	every (8)	long (5)	regimes (6)	want (5)
also (5)	evil (5)	make (7)	security (19)	war (12)
America (33)	for (27)	many (5)	september (5)	was (11)
American (15)	free (6)	more (11)	so (12)	we (76)
americans (8)	freedom (10)	most (5)	some (6)	we've (5)
an (7)	from (15)	must (18)	states (9)	weapons (12)
and (210)	good (13)	my (13)	tax (7)	were (7)
are (17)	great (8)	nation (11)	terror (13)	while (5)
as (18)	has (12)	need (7)	terrorist (12)	who (18)

www.ChinaDBA.net 中国DBA超级论坛

（续）

ask (5)	have (32)	never (7)	terrorists (10)	will (49)
at (16)	health (5)	new (13)	than (6)	with (22)
be (23)	help (7)	no (7)	that (29)	women (5)
been (8)	home (5)	not (15)	the (184)	work (7)
best (6)	homeland (7)	now (10)	their (17)	workers (5)
budget (7)	hope (5)	of (130)	them (8)	world (17)
but (7)	i (29)	on (32)	these (18)	would (5)
by (13)	if (8)	one (5)	they (12)	yet (8)
camps (8)	in (79)	opportunity (5)	this (28)	you (12)
can (7)	is (44)	or (8)	thousands (5)	
children (6)	it (21)	our (78)	time (7)	

我们看到有128个单词出现的次数大于4次。其中许多在几乎所有的英文文本中都经常出现，如a (69)、and (210)、i (29)、in (79)、the (184)等。我们还能发现一些单词，它们概述了美国当前所面临的问题：terror (13)、terrorist (12)、terrorists (10)、security (19)、weapons (12)、destruction (5)、afghanistan (10)、freedom (10)、jobs (11)、budget (7) 等。

1.7 Hadoop 历史

Hadoop开始时是Nutch的一个子项目，而Nutch又是Apache Lucene的一个子项目。这3个项目都是由Doug Cutting所创立的，每个项目在逻辑上都是前一个项目的演进。

Lucene是一个功能全面的文本索引和查询库。给定一个文本集合，开发者就可以使用Lucene引擎方便地在文档上添加搜索功能。桌面搜索、企业搜索，以及许多领域特定的搜索引擎使用的都是Lucene。作为Lucene的扩展，Nutch的目标可谓雄心勃勃，它试图以Lucene为核心建立一个完整的Web搜索引擎。Nutch为HTML提供了解析器，还具有网页抓取工具、链接图形数据库和其他网络搜索引擎的额外组件。Doug Cutting所设想的Nutch是开放与民主的，可以替代Google等商业产品的垄断技术。

除了增加了像抓取器和解析器这样的组件，网络搜索引擎与基本的文档搜索引擎的区别就在于规模。Lucene的目标是索引数百万的文档，但Nutch应该能够处理数十亿的网页，而不会带来过度的操作开销。这样Nutch就得运行在由商用硬件组成的分布式集群上。Nutch团队面临的挑战是解决软件可扩展性问题，即要在Nutch中建立一个层，来负责分布式处理、冗余、自动故障恢复和负载均衡。这些挑战绝非易事。

在2004年左右，Google发表了两篇论文来论述Google文件系统（GFS）和MapReduce框架。Google声称使用了这两项技术来扩展自己的搜索系统。Doug Cutting立即看到了这些技术可以适用于Nutch，接着他的团队实现了一个新的框架，将Nutch移植上去。这种新的实现马上提升了Nutch的可扩展性。它开始能够处理几亿个网页，并能够运行在几十个节点的集群上。Doug认识

www.ChinaDBA.net 中国DBA超级论坛

到设计一个专门的项目可以充实两种网络扩展所需的技术，于是就有了Hadoop。雅虎在2006年1月聘用Doug，让他和一个专项团队一起改进Hadoop，并将其作为一个开源项目。两年后，Hadoop成为Apache的顶级项目。后来，在2008年2月19日，雅虎宣布其索引网页的生产系统采用了在10 000多个核的Linux集群上运行的Hadoop（见http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html）。Hadoop真正达到了万维网的规模！

这个名字是怎么来的

为软件项目命名时，Doug Cutting似乎总会得到家人的启发。Lucene是他妻子的中间名，也是她外祖母的名字。他的儿子在咿呀学语时，总把所有用于吃饭的词叫成Nutch，后来，他把一个黄色大象毛绒玩具叫做Hadoop。Doug说，他“在寻找一个名字，不是已经存在的域名或商标，所以我尝试生活中以前没有人用过的各种词汇。而孩子们很擅长创造单词。”

1.8　小结

Hadoop是一个通用的工具，它让新用户可以享受到分布式计算的好处。通过采用分布式存储、迁移代码而非迁移数据，Hadoop在处理大数据集时避免了耗时的数据传输问题。此外，数据冗余机制允许Hadoop从单点失效中恢复。你已经看到在Hadoop中使用MapReduce框架编写程序非常方便，而且同等重要的是，此时你不必担心如何分割数据、如何分配任务执行节点，或者如何管理节点间的通信。Hadoop为你处理这些事务，使你可以专注于那些最重要的事情——你的数据以及你想用它做什么。

下一章我们将深入Hadoop内部并探究构建Hadoop集群的更多细节。

1.9　资源

Hadoop的官方网站：http://hadoop.apache.org/。

Google文件系统和MapReduce的原始论文很值得一读，可以了解它们的底层设计和架构：

- *The Google File System*——http:// labs.google.com/papers/gfs.html
- *MapReduce: Simplified Data Processing on Large Clusters*——http://labs.google.com/papers/mapreduce.html

www.ChinaDBA.net 中国DBA超级论坛

第2章 初识Hadoop

2

本章内容

- Hadoop的结构组成
- 安装Hadoop及其3种工作模式：单机、伪分布和全分布
- 用于监控Hadoop安装的Web工具

本章将作为路线图来指导Hadoop的全程安装。如果在工作环境中，已经有人为你安装好了Hadoop集群，你也许会打算浏览一下本章，只去了解个人开发环境的安装，而跳过通信配置以及不同节点协同这些细节。

在2.1节讨论Hadoop的物理组件之后，在2.2节和2.3节我们将展开介绍集群安装，其中2.3节关注Hadoop的3种工作模式及其设定。在2.4节，你会了解基于Web界面进行集群监控的辅助工具。

2.1 Hadoop的构造模块

在前一章，我们已经讨论了分布式存储和分布式计算的概念。现在让我们来看Hadoop是如何实现这些思想的。在一个全配置的集群上，“运行Hadoop”意味着在网络分布的不同服务器上运行一组守护进程（daemons）。这些守护进程有特殊的角色，一些仅存在于单个服务器上，一些则运行在多个服务器上。它们包括：

- NameNode（名字节点）；[①]
- DataNode（数据节点）；
- Secondary NameNode（次名字节点）；
- JobTracker（作业跟踪节点）；
- TaskTracker（任务跟踪节点）；

我们将逐一讨论并定位它们在Hadoop中的作用。

2.1.1 NameNode

我们从NameNode开始讨论。也许会有人质疑，但我们认为它是Hadoop守护进程中最重要的

① 因为这些名词有指代的作用，故译文中仍用英文来表示。——译者注

www.ChinaDBA.net 中国DBA超级论坛

一个。Hadoop在分布式计算与分布式存储中都采用了主/从（master/slave）结构。分布式存储系统被称为Hadoop文件系统，或简称为HDFS。NameNode位于HDFS的主端，它指导从端的DataNode执行底层的I/O任务。NameNode是HDFS的书记员，它跟踪文件如何被分割成文件块，而这些块又被哪些节点存储，以及分布式文件系统的整体运行状态是否正常。

运行NameNode消耗大量的内存和I/O资源。因此，为了减轻机器的负载，驻留NameNode的服务器通常不会存储用户数据或者执行MapReduce程序的计算任务。这意味着NameNode服务器不会同时是DataNode或者TaskTracker。

不过NameNode的重要性也带来了一个负面影响——Hadoop集群的单点失效。对于任何其他的守护进程，如果它们所驻留的节点发生软件或硬件失效，Hadoop集群很可能还会继续平稳运行，不然你还可以快速重启这个节点。但这样的方法并不适用于NameNode。

2.1.2　DataNode

每一个集群上的从节点都会驻留一个DataNode守护进程，来执行分布式文件系统的繁重工作——将HDFS数据块读取或者写入到本地文件系统的实际文件中。当希望对HDFS文件进行读写时，文件被分割为多个块，由NameNode告知客户端每个数据块驻留在哪个DataNode。客户端直接与DataNode守护进程通信，来处理与数据块相对应的本地文件。而后，DataNode会与其他DataNode进行通信，复制这些数据块以实现冗余。

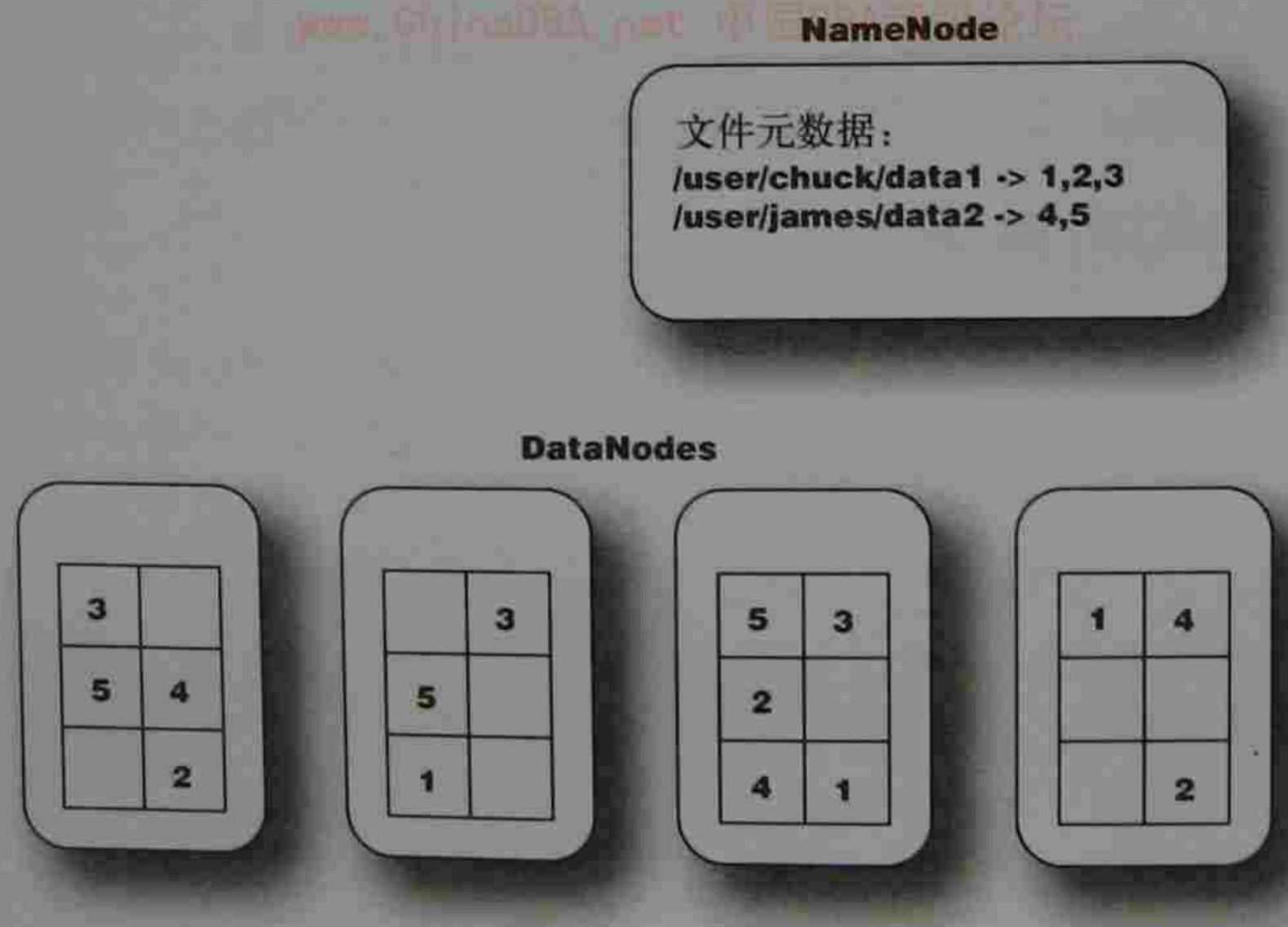


图2-1　NameNode/DataNode在HDFS中的交互。NameNode跟踪文件的元数据——描述系统中所包含的文件以及每个文件如何被分割为数据块。DataNode提供数据块的备份存储，并持续不断地向NameNode报告，以保持元数据为最新状态

图2-1说明了NameNode和DataNode的角色。图中显示了两个数据文件，一个位于目录/user/chuck/data1，另一个位于/user/james/data2。文件data1有3个数据块，表示为1、2和3，而文件data2由数据块4和5组成。这些文件的内容分散在几个DataNode上。这个示例中，每个数据块

www.ChinaDBA.net 中国DBA超级论坛

有3个副本。例如，数据块1（属于文件data1）被复制在图中右侧的3个DataNode上。这确保了如果任何一个DataNode崩溃或者无法通过网络访问时，你仍然可以读取这些文件。

DataNode不断向NameNode报告。初始化时，每个DataNode将当前存储的数据块告知NameNode。在这个初始映射完成后，DataNode仍会不断地更新NameNode，为之提供本地修改的相关信息，同时接收指令创建、移动或删除本地磁盘上的数据块。

2.1.3 Secondary NameNode

Secondary NameNode（SNN）是一个用于监测HDFS集群状态的辅助守护进程。像NameNode一样，每个集群有一个SNN，它通常也独占一台服务器，该服务器不会运行其他的DataNode或TaskTracker守护进程。SNN与NameNode的不同在于它不接收或记录HDFS的任何实时变化。相反，它与NameNode通信，根据集群所配置的时间间隔获取HDFS元数据的快照。

如前所述，NameNode是Hadoop集群的单一故障点，而SNN的快照可以有助于减少停机的时间并降低数据丢失的风险。然而，NameNode的失效处理需要人工的干预，即手动地重新配置集群，将SNN用作主要的NameNode。在第8章，在讲述完集群管理的最佳实践经验之后，我们将讨论到恢复的过程。

2.1.4 JobTracker

JobTracker守护进程是应用程序和Hadoop之间的纽带。一旦提交代码到集群上，JobTracker就会确定执行计划，包括决定处理哪些文件、为不同的任务分配节点以及监控所有任务的运行。如果任务失败，JobTracker将自动重启任务，但所分配的节点可能会不同，同时受到预定义的重试次数限制。

每个Hadoop集群只有一个JobTracker守护进程。它通常运行在服务器集群的主节点上。

2.1.5 TaskTracker

与存储的守护进程一样，计算的守护进程也遵循主/从架构：JobTracker作为主节点，监测MapReduce作业的整个执行过程，同时，TaskTracker管理各个任务在每个从节点上的执行情况。图2-2说明了这种交互关系。

每个TaskTracker负责执行由JobTracker分配的单项任务。虽然每个从节点上仅有一个TaskTracker，但每个TaskTracker可以生成多个JVM（Java虚拟机）来并行地处理许多map或reduce任务。

TaskTracker的一个职责是持续不断地与JobTracker通信。如果JobTracker在指定的时间内没有收到来自TaskTracker的“心跳”，它会假定TaskTracker已经崩溃了，进而重新提交相应的任务到集群中的其他节点中。

讨论了Hadoop的各个守护进程之后，我们在图2-3中描绘了一个典型Hadoop集群的拓扑结构。

这种拓扑结构的特点是在主节点上运行NameNode和JobTracker的守护进程，并使用独立的节点运行SNN以防主节点失效。在小型集群中，SNN也可以驻留在某一个从节点上，而在大型集群中，连NameNode和JobTracker都会分别驻留在两台机器上。每个从节点均驻留一个DataNode和TaskTracker，从而在存储数据的同一节点上执行任务。

www.ChinaDBA.net 中国DBA超级论坛

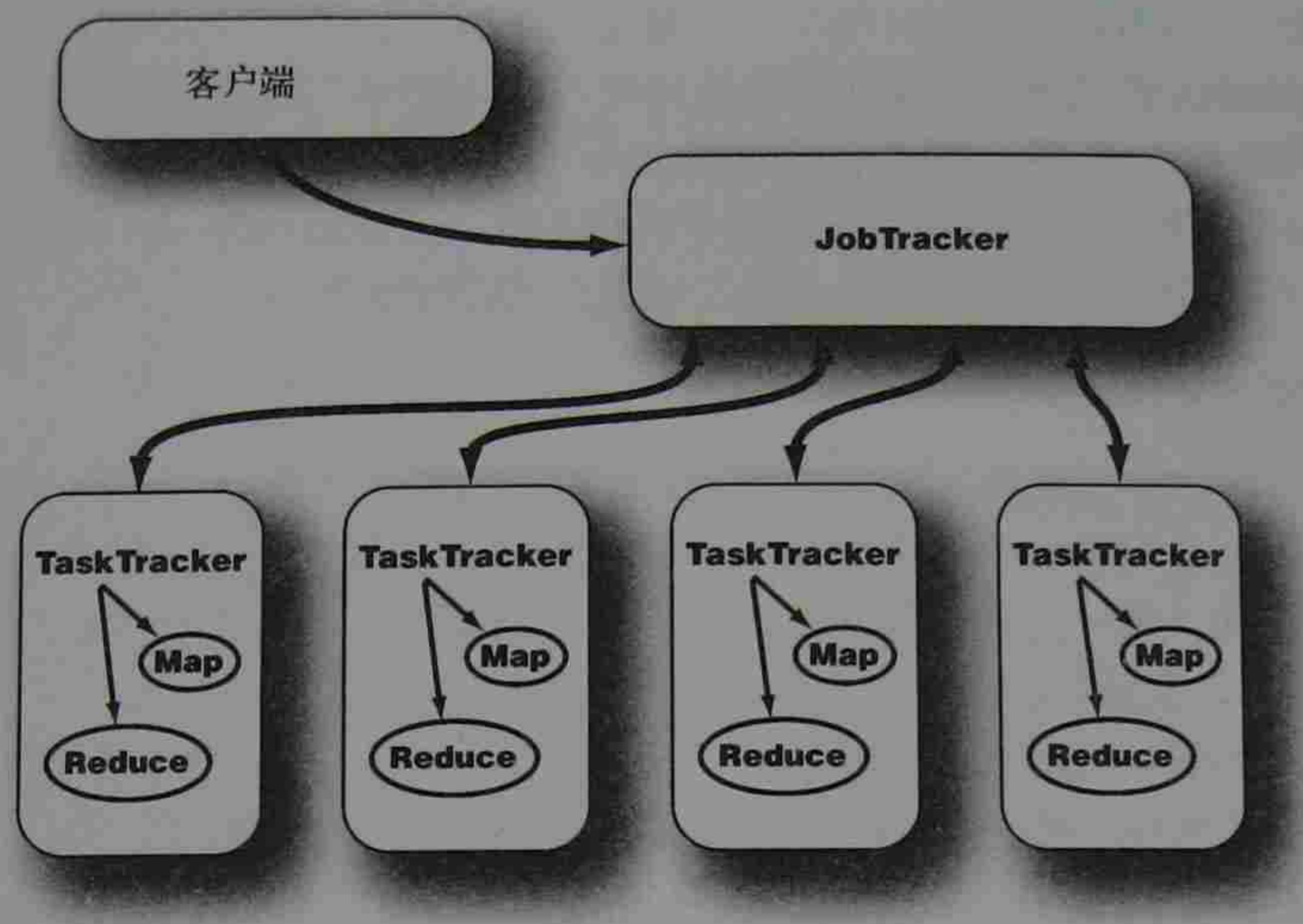


图2-2 JobTracker和TaskTracker的交互。当客户端调用JobTracker来启动一个数据处理作业时，JobTracker会将工作切分，并分配不同的map和reduce任务到集群中的每个TaskTracker上

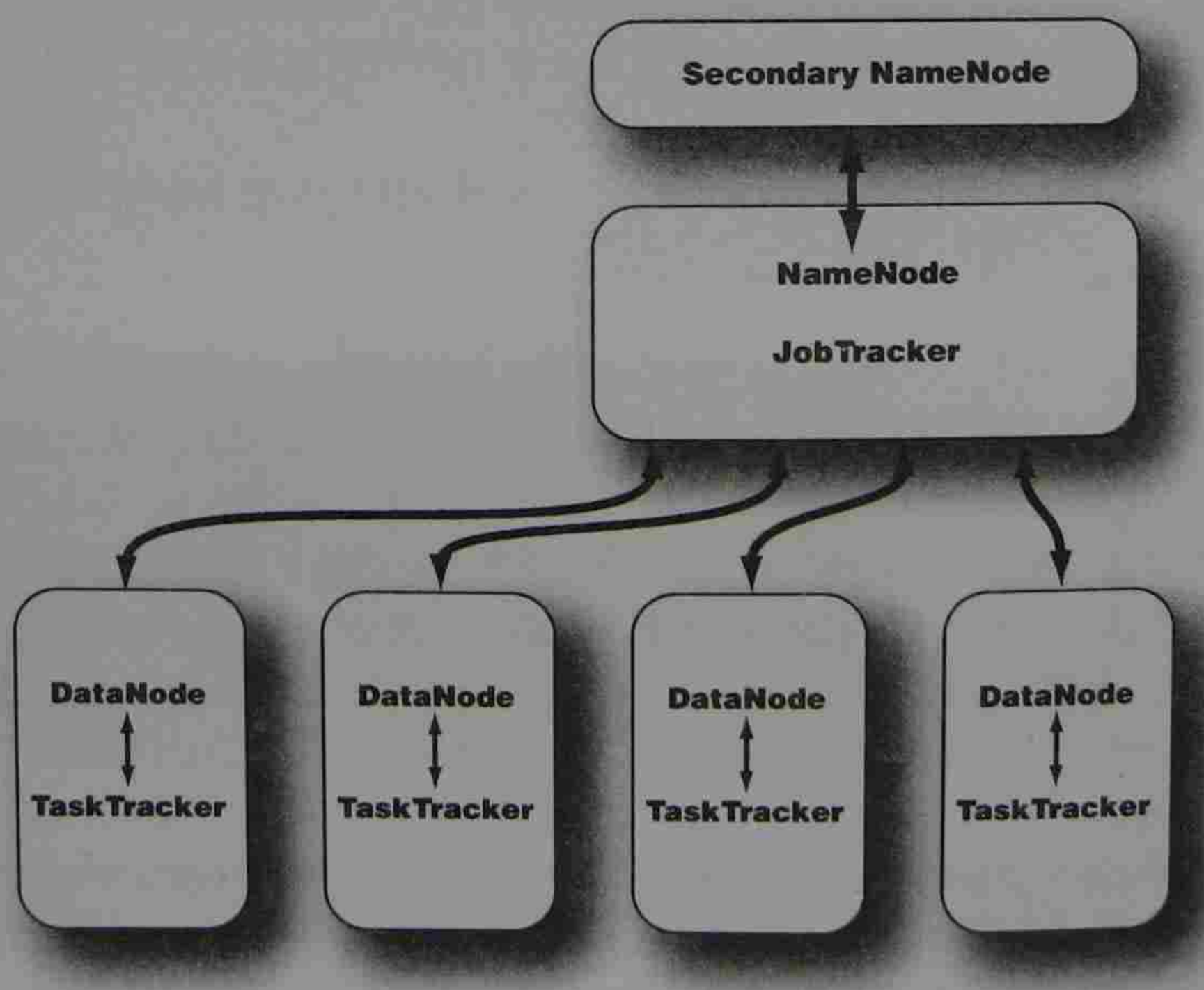


图2-3 一个典型Hadoop集群的拓扑图。这是一个主/从架构，其中NameNode和JobTracker为主端，DataNode和TaskTracker为从端

www.ChinaDBA.net 中国DBA超级论坛

我们致力于来组建这样一个完整的Hadoop集群，这项工作首先得从建立主节点以及节点之间的控制通道开始。如果已经有了一个可用的Hadoop集群，你可以跳过下一节，它主要讲述如何在节点间建立SSH（Secure Shell）通道。另外，还有几个选项允许你仅仅使用一台机器来运行Hadoop，即单机或者伪分布模式。在程序开发时可以使用这些模式。在2.3节中我们将讨论如何把Hadoop配置成上述两种工作模式，或者如何配置成标准的集群安装模式（全分布模式）。

2.2 为 Hadoop 集群安装 SSH

安装一个Hadoop集群时，需要专门指定一个服务器作为主节点。如图2-3所示，这个服务器通常会驻留NameNode和JobTracker的守护进程。它也将作为一个基站，负责联络并激活所有从节点上的DataNode和TaskTracker守护进程。因此，我们需要为主节点定制一种手段，使它能够远程地访问到集群中的每个节点。

为此，Hadoop使用了无口令的（passphraseless）SSH协议。SSH采用标准的公钥加密来生成一对用户验证密钥——一个公钥、一个私钥。公钥被本地存储在集群的每个节点上，私钥则由主节点在试图访问远端节点时发送过来。结合这两段信息，目标机可以对这次登录尝试进行验证。

2.2.1 定义一个公共账号

从一个节点访问另一个节点这种说法一直被我们作为通用术语来使用，但其实更准确的描述应该是从一个节点的用户账号到目标机上的另一个用户账号。对于Hadoop，所有节点上的账号应该有相同的用户名（本书中我们使用hadoop-user），出于安全的考虑，我们建议你把这个账号设置为用户级别。它仅用于管理Hadoop集群。一旦集群的守护进程启动并运行，你就可以从其他的账号运行实际的MapReduce作业。

2.2.2 验证 SSH 安装

第一步是检查节点上是否安装了SSH。我们可以轻松地使用UNIX命令"which"来实现。

```
[hadoop-user@master]$ which ssh
/usr/bin/ssh

[hadoop-user@master]$ which sshd
/usr/bin/sshd

[hadoop-user@master]$ which ssh-keygen
/usr/bin/ssh-keygen
```

如果接收到类似这样的一个错误消息：

```
/usr/bin/which: no ssh in (/usr/bin:/bin:/usr/sbin...
```

你可以通过Linux安装包管理器安装OpenSSH（www.openssh.com）或者直接下载其源码。（但更好的办法是让你的系统管理员帮你去做。）

2.2.3 生成 SSH 密钥对

验证了SSH在集群上所有节点正确安装之后，我们使用主节点上的ssh-keygen来生成一个

www.ChinaDBA.net 中国DBA超级论坛

RSA密钥对。务必要避免输入口令，否则，主节点每次试图访问其他节点时，你都得手动地输入这个口令。

```
[hadoop-user@master]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop-user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop-user/.ssh/id_rsa.
Your public key has been saved in /home/hadoop-user/.ssh/id_rsa.pub.
```

生成密钥对之后，公钥的形式为

```
[hadoop-user@master]$ more /home/hadoop-user/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA1WS3RG8LrZH4zL2/1oYgkV1OmVc1Q2OO5vRi0Nd
K51Sy3wWpBVHx82F3x3ddoZQjBK3uvLMaDhXvncJG31JPfU7CTAfmtgINYv0kdUbDJq4TKG/fuO5q
J9CqHV71thN2M310gcJ0Y9YCN6grmsiWb2iMcXpy2pqg8UM3ZKApyIPx99O1vREWm+4moFTg
YwI15be23ZCyxNjgZFWk5MR1T1p1TxB68jqNbPQtU7fIafS7Sasy7h4eyIy7cbLh8x0/V4/mcQsY
5dvReitNvFVte6on18YdmnMpAh6nwCvog3UeWWJjVZTEBFkTZuV1i9HeYHxpm1wAzcnf7az78jT
IRQ== hadoop-user@master
```

下面我们需要把这个公钥分布到集群中。

2.2.4 将公钥分布并登录验证

尽管有些烦琐，仍需逐一将公钥复制到主节点以及每个从节点上。

```
[hadoop-user@master]$ scp ~/.ssh/id_rsa.pub hadoop-user@target:~/master_key
```

手动登录到目标节点，并设置主节点的密钥为授权密钥（如果还有其他授权密钥，则把主节点密钥追加到授权密钥列表中）：

```
[hadoop-user@target]$ mkdir ~/.ssh
[hadoop-user@target]$ chmod 700 ~/.ssh
[hadoop-user@target]$ mv ~/master_key ~/.ssh/authorized_keys
[hadoop-user@target]$ chmod 600 ~/.ssh/authorized_keys
```

生成该密钥之后，可以尝试从主节点登录到目标节点来验证它的正确性。

```
[hadoop-user@master]$ ssh target
The authenticity of host 'target (xxx.xxx.xxx.xxx)' can't be established.
RSA key fingerprint is 72:31:d8:1b:11:36:43:52:56:11:77:a4:ec:82:03:1d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'target' (RSA) to the list of known hosts.
Last login: Sun Jan 4 15:32:22 2009 from master
```

一旦目标节点确认了对主节点的授权，以后再登录就不会出现这些提示，将显示为

```
[hadoop-user@master]$ ssh target
Last login: Sun Jan 4 15:32:49 2009 from master
```

现在，我们已经有了在集群上运行Hadoop的基础。下面来讨论各种Hadoop模式，在项目中你可能会用到它们。

2.3 运行Hadoop

在运行Hadoop之前需要做一些配置。让我们仔细看看Hadoop的配置目录：

www.ChinaDBA.net 中国DBA超级论坛

```
[hadoop-user@master]$ cd $HADOOP_HOME
[hadoop-user@master]$ ls -l conf/
total 100
-rw-rw-r-- 1 hadoop-user hadoop 2065 Dec 1 10:07 capacity-scheduler.xml
-rw-rw-r-- 1 hadoop-user hadoop 535 Dec 1 10:07 configuration.xsl
-rw-rw-r-- 1 hadoop-user hadoop 49456 Dec 1 10:07 hadoop-default.xml
-rwxrwxr-x 1 hadoop-user hadoop 2314 Jan 8 17:01 hadoop-env.sh
-rw-rw-r-- 1 hadoop-user hadoop 2234 Jan 2 15:29 hadoop-site.xml
-rw-rw-r-- 1 hadoop-user hadoop 2815 Dec 1 10:07 log4j.properties
-rw-rw-r-- 1 hadoop-user hadoop 28 Jan 2 15:29 masters
-rw-rw-r-- 1 hadoop-user hadoop 84 Jan 2 15:29 slaves
-rw-rw-r-- 1 hadoop-user hadoop 401 Dec 1 10:07 sslinfo.xml.example
```

需要做的第一件事是指定包括主节点在内所有节点上Java的位置，即在hadoop-env.sh中定义JAVA_HOME环境变量使之指向Java安装目录。在服务器上，我们将其指定为

```
export JAVA_HOME=/usr/share/jdk
```

(如果你做过了第1章的例子，这一步其实已经完成。)

在hadoop-env.sh文件中还包含定义Hadoop环境的其他变量，但JAVA_HOME是唯一在开始时需要做修正的。其他变量在默认设置下通常都可以很好地工作。你可以在逐渐熟悉Hadoop之后，通过修改这个文件来做个性化的设置（如日志目录的位置、Java类所在的目录，等等）。

Hadoop的设置主要包含在XML配置文件中。在0.20版以前，它们是hadoop-default.xml和hadoop-site.xml。顾名思义，hadoop-default.xml中包含了Hadoop会使用的默认设置，除非这些设置在hadoop-site.xml中被显式地覆盖。因此，在实际操作中你只需要处理hadoop-site.xml。在版本0.20中，这个文件被分离成3个XML文件：core-site.xml，hdfs-site.xml与mapred-site.xml。这次重构更好地对应了它们所控制的Hadoop子系统。在本章的其余部分，我们通常会点明是3个文件中的哪一个被用于配置的调整。如果你使用的是Hadoop的早期版本，请记住，所有这些配置都在hadoop-site.xml中修改。

下面我们将进一步描述Hadoop的不同工作模式并给出其配置文件的示例。

2.3.1 本地（单机）模式

单机模式是Hadoop的默认模式。当首次解压Hadoop的源码包时，Hadoop无法了解硬件安装环境，便保守地选择了最小配置。在这种默认模式下所有3个XML文件（或者0.20版本之前的hadoop-site.xml）均为空。

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

</configuration>
```

当配置文件为空时，Hadoop会完全运行在本地。因为不需要与其他节点交互，单机模式就不使用HDFS，也不加载任何Hadoop的守护进程。该模式主要用于开发调试MapReduce程序的应用

www.ChinaDBA.net 中国DBA超级论坛

逻辑，而不会与守护进程交互，避免引起额外的复杂性。在运行第1章中的MapReduce程序时，采用的就是单机模式。

2.3.2 伪分布模式

伪分布模式在“单节点集群”上运行Hadoop，其中所有的守护进程都运行在同一台机器上。该模式在单机模式之上增加了代码调试功能，允许你检查内存使用情况、HDFS输入输出，以及其他的守护进程交互。代码清单2-1给出了该模式下用于配置单个服务器的简单XML文件。

代码清单2-1 伪分布模式下3个配置文件的示例

```
core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:9000</value>
 <description>The name of the default file system. A URI whose
 scheme and authority determine the FileSystem implementation.
 </description>
</property>

</configuration>

mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>mapred.job.tracker</name>
 <value>localhost:9001</value>
 <description>The host and port that the MapReduce job tracker runs
 at.</description>
</property>

</configuration>

hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>dfs.replication</name>
 <value>1</value>
```

www.ChinaDBA.net 中国DBA超级论坛

```
 <description>The actual number of replications can be specified when the
 file is created.</description>
</property>

</configuration>
```

我们在core-site.xml和mapred-site.xml中分别指定了NameNode和JobTracker的主机名与端口。在hdfs-site.xml中指定了HDFS的默认副本数，因为仅运行在一个节点上，这里副本数为1。我们还需要在文件masters中指定SNN的位置，并在文件slaves中指定从节点的位置。

```
[hadoop-user@master]$ cat masters
localhost
[hadoop-user@master]$ cat slaves
localhost
```

虽然所有的守护进程都运行在同一节点上，它们仍然像分布在集群中一样，彼此通过相同的SSH协议进行通信。在2.2节中已经详细说明了SSH通道的安装，但对于单节点的操作，仅需检查一下机器是否允许对自己运行ssh。

```
[hadoop-user@master]$ ssh localhost
```

如果允许，说明一切正常。否则需要用两行命令来安装。

```
[hadoop-user@master]$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
[hadoop-user@master]$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

现在几乎已经准备好了启动Hadoop。但还是先要输入一个命令来格式化HDFS：

```
[hadoop-user@master]$ bin/hadoop namenode -format
```

我们现在可以使用start-all.sh脚本装载守护进程，然后用Java的jps命令列出所有守护进程来验证安装成功。

```
[hadoop-user@master]$ bin/start-all.sh
[hadoop-user@master]$ jps
26893 Jps
26832 TaskTracker
26620 SecondaryNameNode
26333 NameNode
26484 DataNode
26703 JobTracker
```

当结束Hadoop时，可以通过stop-all.sh脚本来关闭Hadoop的守护进程。

```
[hadoop-user@master]$ bin/stop-all.sh
```

单机模式和伪分布模式均用于开发与调试的目的。真实Hadoop集群的运行采用的是第三种模式，即全分布模式。

2.3.3 全分布模式

在不断强调分布式存储和分布式计算的好处之后，是时候来建立一个完全的集群了。下面的讨论将使用如下的服务器名称。

- master——集群的主节点，驻留NameNode和JobTracker守护进程
- backup——驻留SNN守护进程的节点

www.ChinaDBA.net 中国DBA超级论坛

- hadoop1, hadoop2, hadoop3, ...——集群的从节点，驻留DataNode和TaskTracker守护进程

沿用先前的命名习惯，在伪分布模式配置文件（代码清单2-1）的基础上修改得到代码清单2-2。

代码清单2-2 全分布式模式的配置文件样例

core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>fs.default.name</name>
 <value>hdfs://master:9000</value>
 <description>The name of the default file system. A URI whose
 scheme and authority determine the FileSystem implementation.
 </description>
</property>

</configuration>
```

❶ 定位文件系统的NameNode

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
 <name>mapred.job.tracker</name>
 <value>master:9001</value>
 <description>The host and port that the MapReduce job tracker runs
 at.</description>
</property>

</configuration>
```

❷ 定位JobTracker所在主节点

hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>dfs.replication</name>
 <value>3</value>
 <description>The actual number of replications can be specified when the
 file is created.</description>
</property>

</configuration>
```

❸ 增大HDFS备份参数

主要不同在于以下两点。

- 明确地声明了NameNode❶和JobTracker❷守护进程所在的主机名。
- 增大了HDFS的备份参数以发挥分布式存储的优势❸。回忆一下，数据通过在HDFS上复

www.ChinaDBA.net 中国DBA超级论坛

制可以提高可用性与可靠性。

我们还需要更新masters和slaves文件来指定其他守护进程的位置。

```
[hadoop-user@master]$ cat masters
backup
[hadoop-user@master]$ cat slaves
hadoop1
hadoop2
hadoop3
...
```

在将这些文件复制到集群上的所有节点之后，一定要格式化HDFS以准备好存储数据：

```
[hadoop-user@master]$ bin/hadoop namenode-format
```

现在可以启动Hadoop的守护进程：

```
[hadoop-user@master]$ bin/start-all.sh
```

并验证节点正在运行被指派的任务。

```
[hadoop-user@master]$ jps
30879 JobTracker
30717 NameNode
30965 Jps
[hadoop-user@backup]$ jps
2099 Jps
1679 SecondaryNameNode
[hadoop-user@hadoop1]$ jps
7101 TaskTracker
7617 Jps
6988 DataNode
```

一个可用的集群终于建成了！

模式之间的切换

我发现有一个技巧在开始使用Hadoop时是很有用的，就是使用符号链接而不是不断编辑XML文件来切换Hadoop的模式。为了做到这一点， 需要为每种模式分别生成一个配置目录，并相应地放入恰当版本的XML文件。下面是目录列表的示例：

```
[hadoop@hadoop_master hadoop]$ ls -l
total 4884
drwxr-xr-x 2 hadoop-user hadoop 4096 Nov 26 17:36 bin
-rw-rw-r-- 1 hadoop-user hadoop 57430 Nov 13 19:09 build.xml
drwxr-xr-x 4 hadoop-user hadoop 4096 Nov 13 19:14 c++
-rw-rw-r-- 1 hadoop-user hadoop 287046 Nov 13 19:09 CHANGES.txt
lrwxrwxrwx 1 hadoop-user hadoop 12 Jan 5 16:06 conf -> conf.cluster
drwxr-xr-x 2 hadoop-user hadoop 4096 Jan 8 17:05 conf.cluster
drwxr-xr-x 2 hadoop-user hadoop 4096 Jan 2 15:07 conf.pseudo
drwxr-xr-x 2 hadoop-user hadoop 4096 Dec 1 10:10 conf.standalone
drwxr-xr-x 12 hadoop-user hadoop 4096 Nov 13 19:09 contrib
drwxrwxr-x 5 hadoop-user hadoop 4096 Jan 2 09:28 datastore
drwxr-xr-x 6 hadoop-user hadoop 4096 Nov 26 17:36 docs
...
```

www.ChinaDBA.net 中国DBA超级论坛

然后，就可以使用Linux的ln命令（例如 ln -s conf.cluster conf）在不同配置之间切换。这个技巧还有助于临时把一个节点从集群中分离出来，从而通过伪分布模式来调试一个MapReduce程序，但需要确保这些模式在HDFS上有不同的文件存储位置，并且在改变配置之前还应该停止所有的守护进程。

既然我们已经通览了建立并运行一个Hadoop集群的所有配置，下面将介绍用于集群状态基本监控的Web用户界面。

2.4　基于 Web 的集群用户界面

在讨论了Hadoop的工作模式之后，我们现在介绍Hadoop中用于监控集群健康状态的Web界面。与搜寻日志和目录相比，通过浏览器的界面，我们可以更快地获得所需的信息。

NameNode通过端口50070提供常规报告，描绘集群上HDFS的状态视图。图2-4显示了一个以两节点集群为例的报告。通过这个界面，可以通览文件系统，检查集群每个DataNode的状态，并详细查看Hadoop守护进程的日志来判断集群当前运行是否正确。

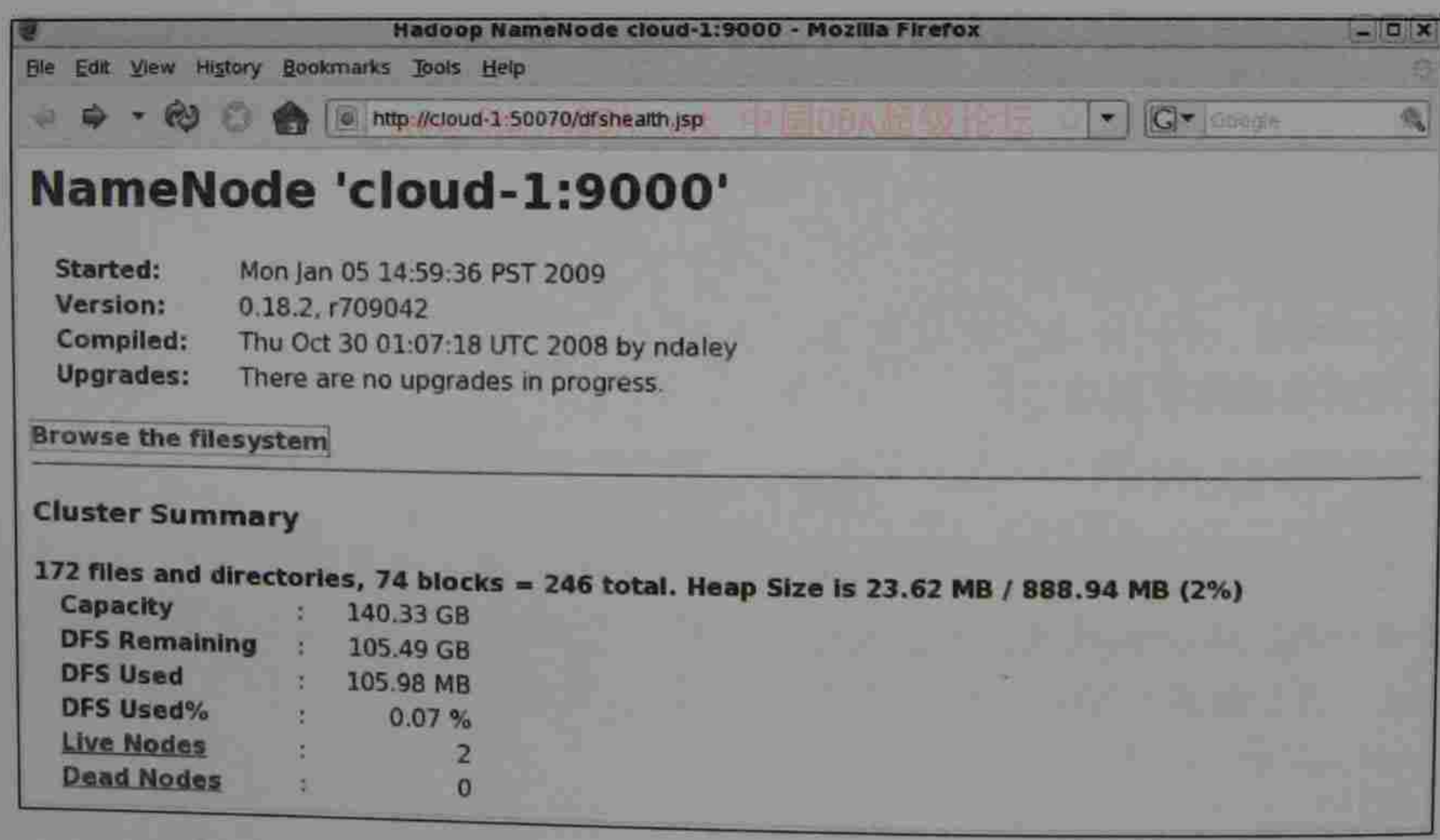


图2-4　HDFS的web界面快照。从这个界面可以通览HDFS文件系统，掌握每个独立节点上可获得的存储资源，并监控集群的整体健康状态

Hadoop提供一个MapReduce作业运行时状态的近似视图。图2-5显示了一个JobTracker通过端口50030给出的视图。

通过这个报告同样可以获得大量的信息，包括MapReduce中任务的运行时状态，以及整个作业的详细报告。后者尤为重要——这些日志描述了哪个节点执行了哪个任务，以及需要完成每个任务所需的时间或资源比。最后，还可以获得Hadoop对每个作业的配置，如图2-6所示。通过所

www.ChinaDBA.net 中国DBA超级论坛

有这些信息，可以合理地管理MapReduce程序，更好地利用集群的资源。

cloud-1 Hadoop Map/Reduce Administration - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://cloud-1:50030/jobtracker.jsp

cloud-1 Hadoop Map/Reduce Administration

State: RUNNING
Started: Mon Jan 05 14:59:43 PST 2009
Version: 0.18.2, r709042
Compiled: Thu Oct 30 01:07:18 UTC 2008 by ndaley
Identifier: 200901051459

Cluster Summary

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
0	0	77	2	4	4	4.00

Running Jobs

Running Jobs
none

图2-5 MapReduce的web界面快照。这个工具可以监控活跃的MapReduce作业，并访问每个map和reduce任务的日志。还可以获得以前提交作业的日志，以辅助程序的调试

Job Configuration: JobId - job_200812231505_0001 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://cloud-1:50030/jobconf_history.jsp?jobid=job_200812231505_0

Job Configuration: JobId - job_200812231505_0001

name	value
fs.s3n.impl	org.apache.hadoop.fs.s3native.NativeS3FileSystem
dfs.client.buffer.dir	${hadoop.tmp.dir}/dfs/tmp
mapred.task.cache.levels	2
hadoop.tmp.dir	/home/hadoop/hadoop-18.2/hadoop-0.18.2/data
hadoop.native.lib	true
map.sort.class	org.apache.hadoop.util.QuickSort
ipc.client.idlethreshold	4000
mapred.system.dir	${hadoop.tmp.dir}/mapred/system
mapred.job.tracker.persist.jobstatus.hours	0
dfs.namenode.logging.level	info
dfs.datanode.address	0.0.0.0:50010

图2-6 一个特定MapReduce作业的详细配置，这些信息在通过调整参数来优化程序性能时可能有用

www.ChinaDBA.net 中国DBA超级论坛

虽然在现阶段这些工具的用处也许不会马上显现，但当在集群上运行更为复杂的任务时，它们的易用性将逐渐展露头脚。随着深入学习Hadoop，我们会逐渐认识到这些工具的重要性。

2.5 小结

本章我们讨论了关键的节点以及它们在Hadoop体系结构中所扮演的角色。学习内容涵盖了如何配置集群，以及利用一些基本工具来监控集群的整体健康状态。

总体上，本章所讨论的是一次性的任务。一旦为集群配置好了一个NameNode，你将永远不需要再管它（至少我们希望如此），也不需要总去变更hadoop-site.xml配置文件，或者为节点分配守护进程。下一章，将学习需要每天和Hadoop打交道的部分，比如管理HDFS的文件。通过这些知识，你就可以开始撰写自己的MapReduce应用，并发掘Hadoop的巨大潜能。

www.ChinaDBA.net 中国DBA超级论坛

第3章 Hadoop组件

本章内容

- 管理HDFS中的文件
- 分析MapReduce框架中的组件
- 读写输入输出数据

在上一章我们着眼于Hadoop的建立与安装，内容涵盖不同节点的功能及配置方法。一旦让Hadoop运转起来，我们就可以从程序员的角度来审视 Hadoop框架。如果把前一章视为教你如何把唱盘机转盘、混音器、放大器和喇叭连接在一起，本章讲的则是混音技术。

我们首先讨论HDFS，它存储着Hadoop应用将要处理的数据。下面我们会更详细地诠释MapReduce框架。在第一章我们已经看到了一个MapReduce程序，但是对程序逻辑的讨论仅限于概念层次。在本章我们将了解Java类和方法，以及底层的处理步骤。我们还会学习如何使用不同的数据格式来读写数据。

3.1 HDFS 文件操作

HDFS是一种文件系统，专为MapReduce这类框架下的大规模分布式数据处理而设计。你可以把一个大数据集（比如说100 TB）在HDFS中存储为单个文件，而大多数其他的文件系统无力实现这一点。我们在第2章中讨论了如何通过复制数据获得可用性，以及将数据副本分布在多台机器上来支持并行处理。HDFS使你不必考虑这些细节，让你感觉就像在处理单个文件一样。

因为HDFS并不是一个天生的Unix文件系统，不支持像`ls`和`cp`这种标准的Unix文件命令[①]，也不支持如`fopen()`和`fread()`这样的标准文件读写操作。另一方面，Hadoop确也提供了一套与Linux文件命令类似的命令行工具。在下一节，我们将讨论Hadoop操作文件的那些shell命令，它们是与HDFS系统的主要接口。3.1.2节讨论了部分Hadoop的Java库，它们用于以编程的方式处理HDFS文件。

---

① 有几个项目正试图使HDFS挂载为一个Unix文件系统。更多详情见http://wiki.apache.org/hadoop/MountableHDFS。在撰写本书时，这些项目还不是Hadoop正式的一部分，它们可能不具备某些生产系统所需的可靠性。

www.ChinaDBA.net 中国DBA超级论坛

注意 一个典型的Hadoop工作流会在别的地方生成数据文件（如日志文件）再将其复制到HDFS中，所使用的命令行工具会在下一节中讨论到。接着由MapReduce程序处理这个数据，但它们通常不会直接读任何一个HDFS文件。相反，它们依靠MapReduce框架来读取HDFS文件，并将之解析为独立的记录（键/值对），这些记录才是MapReduce程序所处理的数据单元。除非需要定制数据的导入与导出，否则你几乎不必编程来读写HDFS文件。

3.1.1 基本文件命令

Hadoop的文件命令采取的形式为

```
hadoop fs -cmd <args>
```

其中cmd是具体的文件命令，而<args>是一组数目可变的参数。cmd的命名通常与UNIX对应的命令名相同。例如，文件列表命令为[1]

```
hadoop fs -ls
```

让我们来看看在Hadoop中最常用的文件管理任务，其中包括

- 添加文件和目录
- 获取文件
- 删除文件

www.ChinaDBA.net 中国DBA超级论坛

指定文件和目录确切位置的URI

Hadoop的文件命令既可以与HDFS文件系统交互，也可以和本地文件系统交互。（正如我们将在第9章看到的，它还能把Amazon S3视为一个文件系统来交互。）URI精确地定位一个特定文件或目录的位置。完整的URI格式为scheme://authority/path。Scheme类似于一个协议。它可以是hdfs或file，来分别指定HDFS文件系统或本地文件系统。对于HDFS，authority是NameNode的主机名，而path是文件或目录的路径。例如，对于在本地机器的9000端口上，以标准伪分布式模型运行的HDFS，访问用户目录user/chuck中文件example.txt的URI大致为hdfs://localhost:9000/user/chuck/example.txt.你可以使用Hadoop的cat命令来显示该文件的内容：

```
hadoop fs -cat hdfs://localhost:9000/user/chuck/example.txt
```

正如我们马上就会看到的，大多数设置不需要指定URI中的scheme://authority部分。对于本地文件系统，你可能会更喜欢用标准的Unix命令，而不是Hadoop文件命令。当在本地文件系统和HDFS之间复制文件时，Hadoop中的命令（如put和get）会分别把本地文件系统作为源和目的，而不需要指定scheme为file。对于其他命令，如果未设置URI中scheme://authority，就会采用Hadoop的默认配置。例如，假如conf/core-site.xml文件已经更改为伪分布式配置，则文件中fs.default.name属性应为

[1] 有些更早的文档以hadoop dfs -cmd <args>的形式表示文件工具。dfs和fs是等价的，但现在更倾向于使用fs。

www.ChinaDBA.net 中国DBA超级论坛

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
</property>
```

在此配置下，URI hdfs://localhost:9000/user/chuck/example.txt缩短为/user/chuck/example.txt。此外，HDFS默认当前工作目录为/user/$USER，其中$ USER是你的登录用户名。如果你作为chuck登录，则URI hdfs://localhost:9000/user/chuck/example.txt就缩短为example.txt。显示文件内容的Hadoop cat命令可写为

```
hadoop fs -cat example.txt
```

1. 添加文件和目录

在运行Hadoop程序处理存储在HDFS上的数据之前，你需要首先把数据放在HDFS上。让我们假设你已经完成了格式化，并启动了一个HDFS文件系统。（出于学习目的，我们建议使用伪分布模式的配置。）让我们创建一个目录并放入一个文件。

HDFS有一个默认的工作目录/user/$USER，其中$USER 是你的登录用户名。不过这个目录不会自动建立，让我们用mkdir命令创建它。为了方便表达，我们使用chuck作为用户名。你需要用你的用户名来替换。

```
hadoop fs -mkdir /user/chuck
```

Hadoop的mkdir命令会自动创建父目录（如果此前不存在的话），类似于UNIX中使用-p选项的mkdir命令。因此上述命令还会创建/user目录。让我们用ls命令对目录进行检查：

```
hadoop fs -ls /
```

该命令返回根目录下目录 /user的信息：

```
Found 1 items
drwxr-xr-x   - chuck supergroup          0 2009-01-14 10:23 /user
```

如果你想看到所有的子目录，则可以使用Hadoop的lsr命令，它类似于Unix中打开-r选项的ls：

```
hadoop fs -lsr /
```

你会看到所有的文件和子目录：

```
drwxr-xr-x   - chuck supergroup          0 2009-01-14 10:23 /user
drwxr-xr-x   - chuck supergroup          0 2009-01-14 10:23 /user/chuck
```

既然有了一个工作目录，我们可以放一个文件进去。在本地文件系统中创建一个名为example.txt的文本文件，用Hadoop的命令put将它从本地文件系统复制到HDFS中去。

```
hadoop fs -put example.txt .
```

注意上面这个命令最后一个参数是句点（.）。这意味着我们把文件放入了默认的工作目录。该命令等价于：

```
hadoop fs -put example.txt /user/chuck
```

www.ChinaDBA.net 中国DBA超级论坛

我们重新执行递归列出文件的命令，可以看到新的文件被添加到HDFS中。

```
$ hadoop fs -lsr /
drwxr-xr-x   - chuck supergroup          0 2009-01-14 10:23 /user
drwxr-xr-x   - chuck supergroup          0 2009-01-14 11:02 /user/chuck
-rw-r--r--   1 chuck supergroup        264 2009-01-14 11:02 /user/chuck/example.txt
```

实际上，我们并不需要递归地检查所有文件，而仅限于在我们自己的工作目录中的文件。我们可以通过最简单的形式来使用Hadoop的ls命令：

```
$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 chuck supergroup        264 2009-01-14 11:02 /user/chuck/example.txt
```

输出结果显示出属性信息，比如权限、所有者、组、文件大小，以及最后修改日期，所有这些都类似于Unix的概念。显示“1”的列给出文件的复制因子。伪分布式配置下它永远为1。对于生产环境中的集群，复制因子通常为3，但也可以是任何正整数。因为复制因子不适用于目录，故届时该列仅会显示一个破折号(-)。

当你把数据放到HDFS上之后，你可以运行Hadoop程序来处理它。处理过程将输出一组新的HDFS文件，然后你可以读取或检索这些结果。

2. 检索文件

Hadoop的get命令与put截然相反。它从HDFS中复制文件到本地文件系统。比方说，我们在本地不再拥有文件example.txt，而想从HDFS中将它取回，我们就可以运行命令：

```
hadoop fs -get example.txt .
```

将它复制到我们在本地的当前工作目录中。

另一种访问数据的方式是显示。由Hadoop的cat命令来支持：

```
hadoop fs -cat example.txt
```

我们可以在Hadoop的文件命令中使用Unix的管道，将其结果发送给其他的Unix命令做进一步处理。例如，如果该文件非常大（正如典型的Hadoop文件那样），并且你希望快速地检查其内容，就可以把Hadoop中cat命令的输出用管道传递给Unix命令head：

```
hadoop fs -cat example.txt | head
```

Hadoop内在支持tail命令来查看最后的一千字节：

```
hadoop fs -tail example.txt
```

文件在HDFS上的任务完成之后，可以考虑删除它们以释放空间。

3. 删除文件

Hadoop删除文件的命令名为rm，现在你也许不会感到太惊讶了：

```
hadoop fs -rm example.txt
```

rm命令还可用于删除空目录。

4. 查阅帮助

在附录中给出了一个Hadoop文件命令的清单，附加了每个命令的用法和说明。多数命令

www.ChinaDBA.net 中国DBA超级论坛

模仿了相应的UNIX命令。你可以执行hadoop fs（无参数）来获取你所用版本Hadoop的一个完整命令列表。你也可以使用help来显示每个命令的用法及简短描述。例如，要了解命令ls，可执行

```
hadoop fs -help ls
```

你就会看到如下的描述：

```
-ls <path>:    List the contents that match the specified file pattern. If
               path is not specified, the contents of /user/<currentUser>
               will be listed. Directory entries are of the form
                       dirName (full path) <dir>
               and file entries are of the form
                       fileName(full path) <r n> size
               where n is the number of replicas specified for the file
               and size is the size of the file, in bytes.
```

虽然命令行工具足以满足大多数与HDFS文件系统交互的需求，但它们并不尽善尽美。某些情况下，你可能会更进一步用到HDFS的API访问。让我们在下一节看看如何去做。

3.1.2 编程读写 HDFS

为了体验HDFS的Java API，我们将开发一个PutMerge程序，用于合并文件后放入HDFS。命令行工具并不支持这个操作，我们将使用API来实现。

需要分析来自许多Web服务器的Apache日志文件时，就有了建立这个例程的动机。虽然我们可以把每个日志文件都复制到HDFS中，但通常而言，Hadoop处理单个大文件会比处理许多个小文件更有效率（这里“小”是相对的，因为它仍会达到几十或几百GB）。此外，从分析目的来看，我们把日志数据视为一个大文件。日志数据被分散在多个文件是由于Web服务器采用分布式架构所带来的副作用。一种解决办法是先将所有的文件合并，然后再复制到HDFS。可是，文件合并需要占用本地计算机的大量磁盘空间。如果我们能够在向HDFS复制的过程中合并它们，事情就会简单很多。

因此，我们需要一个PutMerge类型的操作。Hadoop命令行工具中有一个getmerge命令，用于把一组HDFS文件在复制到本地计算机以前进行合并。但我们想要的则截然相反，故无法在Hadoop的文件管理工具中获得。我们用HDFS API来自己编程实现它。

在Hadoop中用作文件操作的主类位于org.apache.hadoop.fs软件包中。Hadoop的基本文件操作包括常见的open、read、write和close。实际上，Hadoop的文件API是通用的，可用于HDFS以外的其他文件系统。对于我们的PutMerge程序，它读取本地文件系统和写入HDFS都会使用Hadoop的文件API。

Hadoop文件API的起点是FileSystem类。这是一个与文件系统交互的抽象类，存在不同的具体实现子类用来处理HDFS和本地文件系统。你可以通过调用factory方法FileSystem.get(Configuration conf)来得到所需的FileSystem实例。Configuration类是用于保留键/值配置

www.ChinaDBA.net 中国DBA超级论坛

参数的特殊类。它的默认实例化方法是以HDFS系统的资源配置为基础的。如下，我们可以得到与HDFS接口的FileSystem对象：

```
Configuration conf = new Configuration();
FileSystem hdfs = FileSystem.get(conf);
```

要得到一个专用于本地文件系统的FileSystem对象，可用factory方法FileSystem.getLocal(Configuration conf)：

```
FileSystem local = FileSystem.getLocal(conf);
```

Hadoop文件API使用Path对象来编制文件和目录名，使用FileStatus对象来存储文件和目录的元数据。PutMerge程序将合并一个本地目录中的所有文件。我们使用FileSystem的listStatus()方法来得到一个目录中的文件列表：

```
Path inputDir = new Path(args[0]);
FileStatus[] inputFiles = local.listStatus(inputDir);
```

数组inputFiles的长度等于指定目录中的文件个数。在inputFiles中每一个FileStatus对象均有元数据信息，如文件长度、权限、修改时间等。PutMerge程序所关心的是每个文件的Path，即inputFiles[i].getPath()。我们可以通过FSDataInputStream对象访问这个Path来读取文件。

```
FSDataInputStream in = local.open(inputFiles[i].getPath());
byte buffer[] = new byte[256];
int bytesRead = 0;
while( (bytesRead = in.read(buffer)) > 0) {
    ...
}
in.close();
```

FSDataInputStream是Java标准类java.io.DataInputStream的一个子类，增加了对随机访问的支持。类似地有一个FSDataOutputStream对象用于将数据写入HDFS文件：

```
Path hdfsFile = new Path(args[1]);
FSDataOutputStream out = hdfs.create(hdfsFile);
out.write(buffer, 0, bytesRead);
out.close();
```

为了完成PutMerge程序，我们创建一个循环来逐一读取inputFiles中的所有文件，并写入目标HDFS文件。完整的程序见代码清单3-1。

代码清单3-1 PutMerge程序

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class PutMerge {
```

www.ChinaDBA.net 中国DBA超级论坛

```
public static void main(String[] args) throws IOException {
    Configuration conf = new Configuration();
    FileSystem hdfs  = FileSystem.get(conf);
    FileSystem local = FileSystem.getLocal(conf);

    Path inputDir = new Path(args[0]);
    Path hdfsFile = new Path(args[1]);

    try {
        FileStatus[] inputFiles = local.listStatus(inputDir);
        FSDataOutputStream out = hdfs.create(hdfsFile);

        for (int i=0; i<inputFiles.length; i++) {
            System.out.println(inputFiles[i].getPath().getName());
            FSDataInputStream in =
            local.open(inputFiles[i].getPath());
            byte buffer[] = new byte[256];
            int bytesRead = 0;
            while( (bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            in.close();
        }
        out.close();
    } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

❶ 设定输入目录与输出文件

❷ 得到本地文件列表

❸ 生成HDFS输出流

❹ 打开本地输入流

www.ChinaDBA.net 中国DBA超级论坛

程序的大体流程为：❶根据用户定义的参数设置本地目录和HDFS的目标文件；❷提取本地输入目录中每个文件的信息；❸创建一个输出流写入到HDFS文件；❹遍历本地目录中的每个文件，打开一个输入流来读取该文件。剩下就是一个标准的Java文件复制过程了。

FileSystem类还有些方法用于其他标准文件操作，如delete()，exists()，mkdirs()和rename()。关于Hadoop文件API的最新Javadoc在http://hadoop.apache.org/core/docs/current/api/org/apache /hadoop/fs/package-summary.html。

我们已经讨论了如何处理HDFS中的文件。你现在知道一些方法来读写HDFS中的数据，但是仅仅有数据还不够，你还要对它进行处理、分析以及做其他的操作。让我们结束对HDFS的讨论，而转向Hadoop的另一个主要组件——MapReduce框架，看看如何基于它来编程。

3.2 剖析 MapReduce 程序

如前所述，MapReduce程序通过操作键/值对来处理数据，一般形式为

map: (K1,V1) → list(K2,V2)

reduce: (K2,list(V2)) → list(K3,V3)

www.ChinaDBA.net 中国DBA超级论坛

上面是这个数据流的一个相当普通的表现，并无特别之处。而在本节，我们将学习更多的细节，涉及一个典型MapReduce程序的每个阶段。图3-1显示了这个完整过程的高阶视图，我们将逐步遍历这个流程来进一步剖析每一个组成部分。

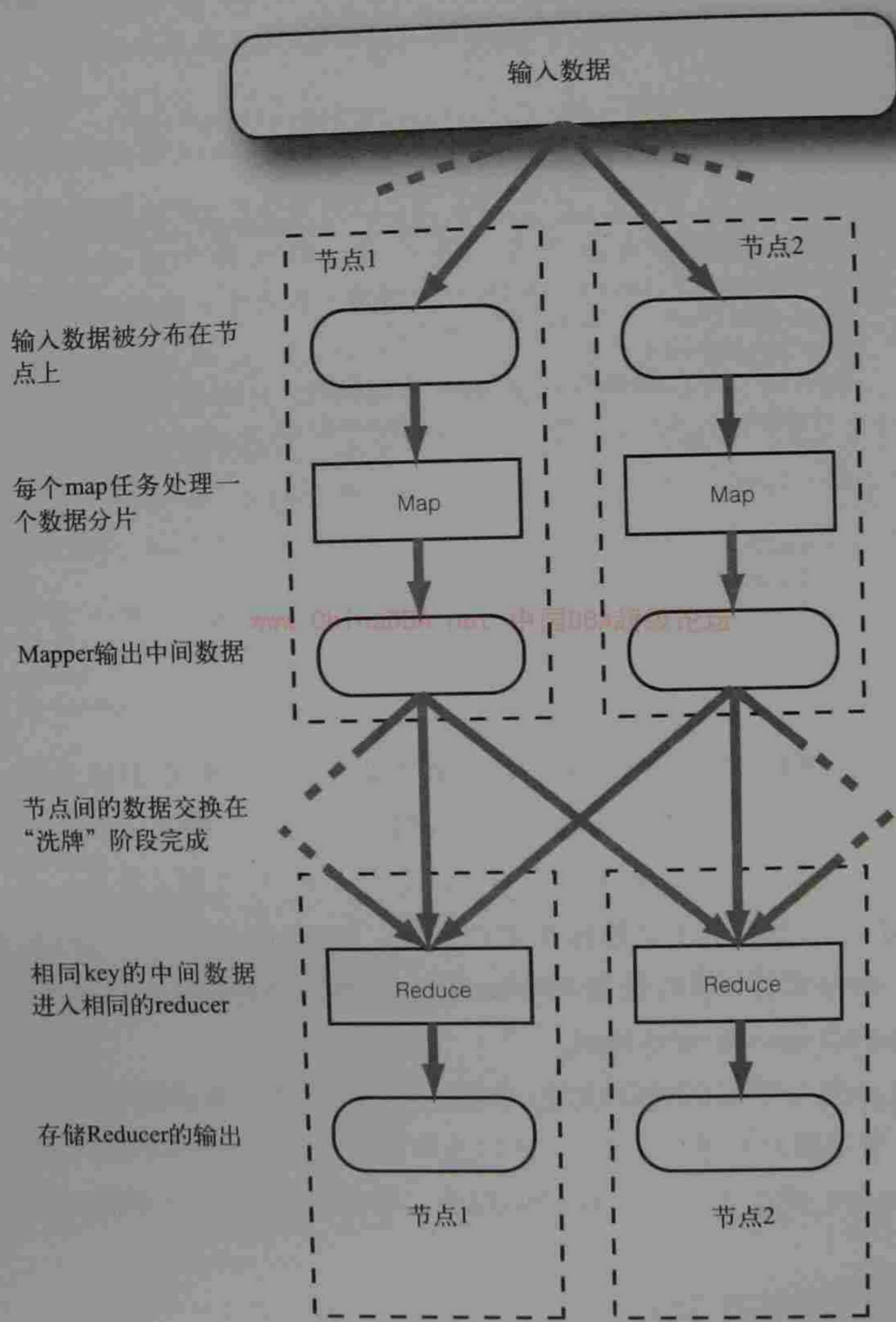


图3-1　普通的MapReduce数据流。注意输入数据被分配到不同节点之后，节点间通信的唯一时间是在“洗牌”阶段。这个通信约束对可扩展性有极大的帮助

在分析数据如何流过每个独立的阶段之前，应该首先熟悉一下Hadoop所支持的数据类型。

www.ChinaDBA.net 中国DBA超级论坛

3.2.1 Hadoop 数据类型

尽管我们的许多讨论只提键和值，但还是得注意它们的类型。MapReduce框架并不允许它们是任意的类。例如，虽然我们可以并且的确经常把某些键与值称为整数、字符串等，但它们实际上并不是Interger、String等那些标准的Java类。这是因为为了让键/值对可以在集群上移动，MapReduce框架提供了一种序列化键/值对的方法。因此，只有那些支持这种序列化的类能够在这个框架中充当键或者值。

更具体而言，实现Writable接口的类可以是值，而实现WritableComparable<T>接口的类既可以是键也可以是值。注意WritableComparable<T>接口是Writable和java.lang.Comparable<T>接口的组合。对于键而言，我们需要这个比较，因为它们将在Reduce阶段进行排序，而值仅会被简单地传递。

Hadoop带有一些预定义的类用于实现WritableComparable，包括面向所有基本数据类型的封装类，如表3-1所示。

表3-1 键/值对经常使用的数据类型列表。这些类均用于实现WritableComparable接口

类	描 述
BooleanWritable	标准布尔变量的封装
ByteWritable	单字节数的封装
DoubleWritable	双字节数的封装
FloatWritable	浮点数的封装
IntWritable	整数的封装
LongWritable	Long的封装
Text	使用UT8格式的文本封装
NullWritable	无键值时的站位符

键和值所采用的数据类型可以超出Hadoop自身所支持的基本类型。你可以自定义数据类型，只要它实现了Writable（或WritableComparable <T>）接口。例如，代码清单3-2给出一个类，用来表示一个网络的边界。这可能代表两个城市之间的航线。

代码清单3-2 示例实现WritableComparable接口的类

```
public class Edge implements WritableComparable<Edge>{

    private String departureNode;
    private String arrivalNode;

    public String getDepartureNode() { return departureNode;}

    @Override
    public void readFields(DataInput in) throws IOException {    ❶ 说明如何读入数据
        departureNode = in.readUTF();
        arrivalNode = in.readUTF();
    }
```

www.ChinaDBA.net 中国DBA超级论坛

```
    @Override
    public void write(DataOutput out) throws IOException {        ❷ 说明如何写出数据
        out.writeUTF(departureNode);
        out.writeUTF(arrivalNode);
    }

    @Override
    public int compareTo(Edge o) {                                 ❸ 定义数据排序
        return (departureNode.compareTo(o.departureNode) != 0)
            ? departureNode.compareTo(o.departureNode)
            : arrivalNode.compareTo(o.arrivalNode);
        }
    }
```

这个Edge类实现了Writable接口中的readFields()❶及write()❷方法。它们与Java中的DataInput和DataOutput类一起用于类中内容的串行化。而Comparable接口中实现的是compareTo()❸方法。如果被调用的Edge小于、等于或者大于给定的Edge，这个方法会分别返回-1，0，1。

利用现在定义的数据类型接口，我们可以开始数据流处理过程的第一阶段，即图3-1中所示的mapper。

3.2.2 Mapper

一个类要作为mapper，需继承MapReduceBase基类并实现Mapper接口。并不奇怪，mapper和reducer的基类均为MapReduceBase类。它包含类的构造与解构方法。

- void configure（JobConf job）。该函数提取XML配置文件或者应用程序主类中的参数，在数据处理之前调用该函数。
- void close ()。作为map任务结束前的最后一个操作，该函数完成所有的结尾工作，如关闭数据库连接、打开文件等。

Mapper接口负责数据处理阶段。它采用的形式为Mapper<K1,V1,K2,V2>Java泛型，这里键类和值类分别实现WritableComparable和Writable接口。Mapper只有一个方法——map，用于处理一个单独的键/值对。

```
void map(K1 key,
         V1 value,
         OutputCollector<K2,V2> output,
         Reporter reporter
        ) throws IOException
```

该函数处理一个给定的键/值对（K1, V1），生成一个键/值对（K2,V2）的列表（该列表也可能为空）。OutputCollector接收这个映射过程的输出，Reporter可提供对mapper相关附加信息的记录，形成任务进度。

Hadoop提供了一些有用的mappper实现，部分如表3-2中所示。

表3-2 一些非常有用的由Hadoop预定义的Mapper实现

类	描 述
IdentityMapper<K,V>	实现Mapper<K,V,K,V>，将输入直接映射到输出
InverseMapper<K,V>	实现Mapper<K,V,V,K>，反转键/值对

www.ChinaDBA.net 中国DBA超级论坛

（续）

类	描 述
RegexMapper<K>	实现Mapper<K,Text,Text,LongWritable>，为每个常规表达式的匹配项生成一个（match,1）对
TokenCountMapper<K>	实现Mapper<K,Text,Text,LongWritable>，当输入的值为分词时，生成一个（token, 1）对

MapReduce，顾名思义在map之后的主要数据流操作是reduce，如图3-1底部所示。

3.2.3 Reducer

reducer的实现和mapper一样必须首先在MapReduce基类上扩展，允许配置和清理。此外，它还必须实现Reducer接口使其具有如下的单一方法：

```
void reduce(K2 key,
            Iterator<V2> values,
            OutputCollector<K3,V3> output,
            Reporter reporter
           ) throws IOException
```

当reducer任务接收来自各个mapper的输出时，它按照键/值对中的键对输入数据进行排序，并将相同键的值归并。然后调用reduce()函数，并通过迭代处理那些与指定键相关联的值，生成一个（可能为空的）列表（K3，V3）。OutputCollector接收reduce阶段的输出，并写入输出文件。Reporter可提供对reducer相关附加信息的记录，形成任务进度。

在表3-3中列出了Hadoop提供的一些基本的reducer实现。

表3-3 一些非常有用的由Hadoop预定义的Reducer实现

类	描 述
IdentityReducer<K,V>	实现Reducer<K,V,K,V>，将输入直接映射到输出
LongSumReducer<K>	实现<K,LongWritable,K,LongWritable>，计算与给定键相对应的所有值的和

虽然我们将Hadoop程序称为MapReduce应用，但是在map和reduce两个阶段之间还有一个极其重要的步骤：将mapper的结果输出给不同的reducer。这就是partitioner的工作。

3.2.4 Partitioner：重定向 Mapper 输出

初次使用MapReduce的程序员通常有一个误解，以为仅需使用一个reducer。毕竟，采用单一的reducer可以在处理之前对所有的数据进行排序——谁不喜欢排序的数据呢？但是，这样理解MapReduce是有问题的，它忽略了并行计算的好处。用一个reducer，我们的计算“云”就被降级成“雨点”了。

但是，当使用多个reducer时，我们就需要采取一些办法来确定mapper应该把键/值对输出给谁。默认的作法是对键进行散列来确定reducer。Hadoop通过HashPartitioner类强制执行这个策略。但有时HashPartitioner会让你出错。让我们回到3.2.1节中介绍的Edge类。

假设你使用Edge类来分析航班信息来决定从各个机场离港的乘客数目。这些数据可以为：

(San Francisco, Los Angeles) Chuck Lam

(San Francisco, Dallas) James Warren

...

www.ChinaDBA.net 中国DBA超级论坛

如果你使用HashPartitioner，这两行可以被送到不同的reducer。离港的乘客数目被处理两次并且两次都是错误的。

如何为你的应用量身定制partitioner呢？在这种情况下，我们希望具有相同离港地的所有edge被送往相同的reducer。这也很容易做到，只要对Edge类的departureNode成员进行散列就可以了：

```
public class EdgePartitioner implements Partitioner<Edge, Writable>
{
    @Override
    public int getPartition(Edge key, Writable value, int numPartitions)
    {
        return key.getDepartureNode().hashCode() % numPartitions;
    }

    @Override
    public void configure(JobConf conf) { }
}
```

一个定制的partitioner只需要实现configure()和getPartition()两个函数。前者将Hadoop对作业的配置应用在patitioner上，而后者返回一个介于0和reduce任务数之间的整数，指向键/值对将要发送到的reducer。

也许你还难以理解Partitioner的确切机制，图3-2给出了更好的诠释。

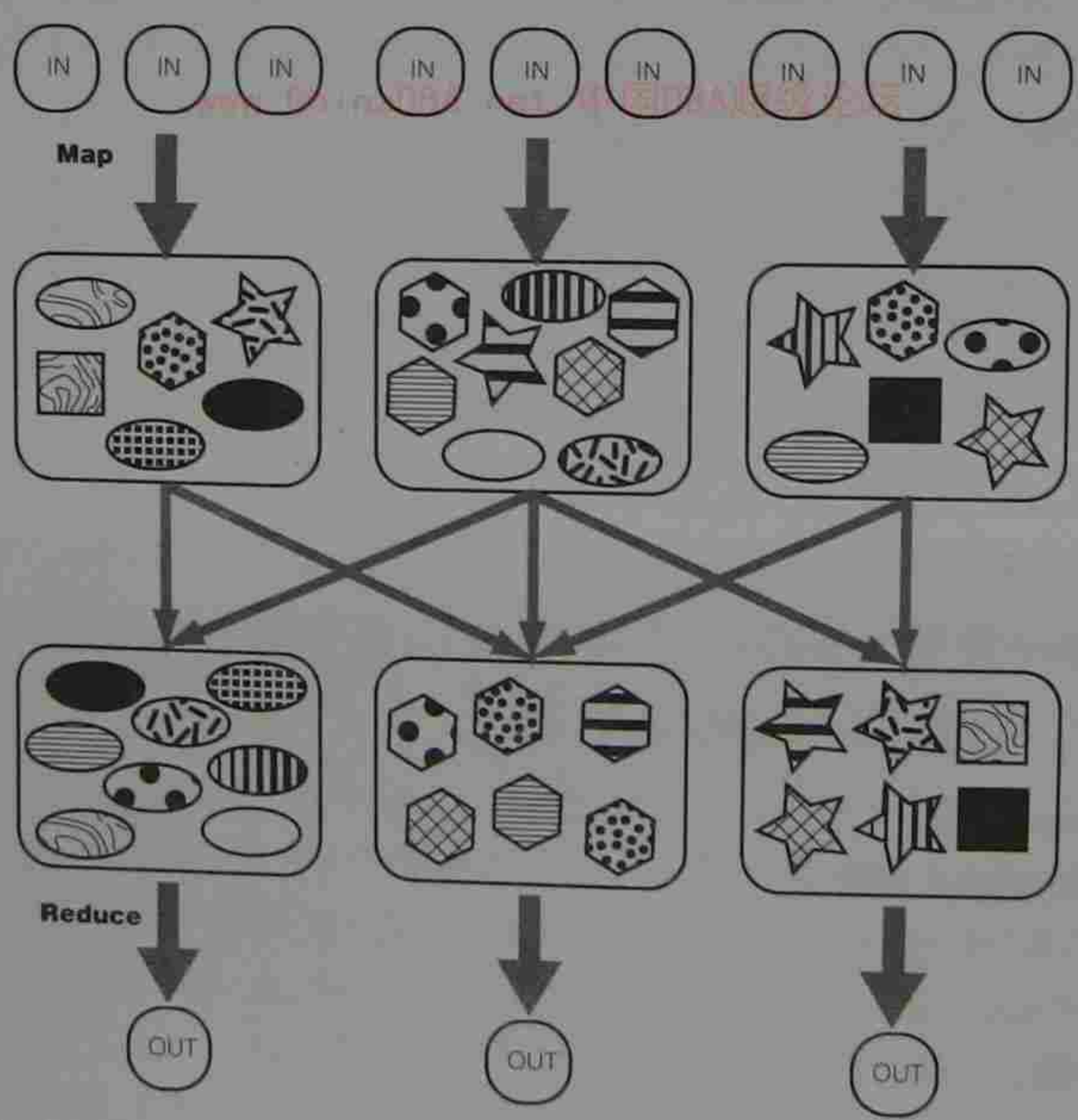


图3-2　MapReduce数据流，重点说明了分区（partitioning）和洗牌（shuffling）。每个图标是一个键/值对（key/value pair）。形状代表键（key），而内部的图案代表值（value）。洗牌之后，相同形状（键）的图标放入相同的Reducer。不同的键也可以放入相同的Reducer，如最右边的Reducer。由partitioner来决定键被放入的位置。注意最左边的Reducer的负载更重，因为更多的数据采用“椭圆形”的键

www.ChinaDBA.net 中国DBA超级论坛

在map和reduce阶段之间，一个MapReduce应用必然从mapper任务得到输出结果，并把这些结果发布给reducer任务。该过程通常被称为洗牌，因为在单节点上的mapper输出可能被送往分布在集群多个节点上的reducer。

3.2.5 Combiner：本地 reduce

在许多MapReduce应用场景中，我们不妨在分发mapper结果之前做一下“本地Reduce”。再考虑一下第1章中WordCount的例子。如果作业处理的文件中单词“the”出现了574次，存储并洗牌一次（“the”，574）键/值对比许多次（“the”，1）更为高效。这种处理步骤被称为合并。我们在4.6节会做详细阐述。

3.2.6 预定义 mapper 和 Reducer 类的单词计数

我们已经完成了对MapReduce所有基本组件的初步讨论。现在你已经看到了更多由Hadoop提供的类。利用我们所学到的一些类来重新审视WordCount示例将会非常有趣（参见代码清单3-3）。

代码清单3-3 修改的WordCount例程

```
public class WordCount2 {
    public static void main(String[] args) {
        JobClient client = new JobClient();
        JobConf conf = new JobConf(WordCount2.class);

        FileInputFormat.addInputPath(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(LongWritable.class);
        conf.setMapperClass(TokenCountMapper.class);
        conf.setCombinerClass(LongSumReducer.class);
        conf.setReducerClass(LongSumReducer.class);

        client.setConf(conf);
        try {
            JobClient.runJob(conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

❶ Hadoop自己的TokenCountMapper

❷ Hadoop自己的LongSumReducer

因为我们使用了Hadoop预定义的类TokenCountMapper❶和LongSumReducer❷，MapReduce程序的撰写就变得非常容易，不是吗？虽然Hadoop也支持生成更复杂程序（这将是本书第二部分的重点），但这里我们希望强调的是，Hadoop允许你通过最小的代码量快速生成实用的程序。

3.3 读和写

让我们看看MapReduce如何读取输入数据，以及如何写入输出数据，并关注它所使用的文件格式。为了易于分布式处理，MapReduce对所处理的数据做了一定的假设，但它也具有一定的灵

www.ChinaDBA.net 中国DBA超级论坛

活性，可支持多种数据格式。

输入数据通常驻留在较大的文件中，通常几十或数百GB，甚至更大。MapReduce处理的基本原则之一是将输入数据分割成块。这些块可以在多台计算机上并行处理。在Hadoop的术语中，这些块被称为输入分片(Input Split)。每个分片应该足够小以实现更细粒度的并行。（如果所有的输入数据都在一个分片中，那就没有并行了。）另一方面，每个分片也不能太小，否则启动与停止各个分片处理所需的开销将占去很大一部分执行时间。

并行处理切分输入数据的原则（输入数据通常为单一的大文件）揭示了其背后Hadoop通用文件系统（尤其是HDFS）的一些设计决策。例如，Hadoop的文件系统提供了`FSDataInputStream`类用于读取文件，而未采用Java中的`java.io.DataInputStream`。`FSDataInputStream`扩展了`DataInputStream`以支持随机读，MapReduce需要这个特性，因为一台机器可能被指派从输入文件的中间开始处理一个分片。如果没有随机访问，而需要从头开始一直读取到分片的位置，效率就会非常低。你还可以看到HDFS为了存储MapReduce并行切分和处理的数据所做的设计。HDFS按块存储文件并分布在多台机器上。笼统而言，每个文件块为一个分片。由于不同的机器会存储不同的块，如果每个分片/块都由它所驻留的机器进行处理，就自动实现了并行。此外，由于HDFS在多个节点上复制数据块以实现可靠性，MapReduce可以选择任意一个包含分片/数据块副本的节点。

回想一下，MapReduce操作是基于键/值对的。目前为止，我们已经看到Hadoop默认地将输入文件中的每一行视为一个记录，而键/值对分别为该行的字节偏移（key）和内容（value）。你也许不会把所有的数据都如此记录。所以，Hadoop也支持一些其他的数据格式，并允许自定义格式。

输入分片与记录边界

请注意，输入分片是一种记录的逻辑划分，而HDFS数据块是对输入数据的物理分割。当它们一致时，效率会非常高，但在实际应用中从未达到完全一致。记录可能会跨过数据块的边界。Hadoop确保全部记录都被处理。处理特定分片的计算节点会从一个数据块中获取记录的一个片段，该数据块可能不是该记录的“主”数据块，而会存放在远端。为获取一个记录片段所需的通信成本是微不足道的，因为它相对而言很少发生。

3.3.1 InputFormat

Hadoop分割与读取输入文件的方式被定义在`InputFormat`接口的一个实现中。`TextInputFormat`是`InputFormat`的默认实现，我们一直暗自使用至今的正是这种数据格式。当你想要一次获取一行内容而输入数据又没有确定的键值时，这种数据格式通常会非常有用。从`TextInputFormat`返回的键为每行的字节偏移量，而我们尚未看到任何的程序使用这个键用于数据处理。

www.ChinaDBA.net 中国DBA超级论坛

1. 常用的INPUTFORMAT类

在表3-4中列出了InputFormat的其他常用实现，并简要描述了每个实现传递给mapper的键/值对。

表3-4 主要的InputFormat类。除非特别说明，TextInputFormat为默认类。下面同时给出了键(key)和值(value)对象类型的描述

InputFormat	描 述
TextInputFormat	在文本文件中的每一行均为一个记录。键（key）为一行的字节偏移，而值（value）为一行的内容 key：LongWritable Value：Text
KeyValueTextInputFormat	在文本文件中的每一行均为一个记录。以每行的第一个分隔符为界，分隔符之前的是键（key），之后的是值（value）。分离器在属性key.value.separator.in.input.line中设定，默认为制表符（\t）。 Key：Text Value：Text
SequenceFileInputFormat<K,V>	用于读取序列文件的InputFormat。键和值由用户定义。序列文件为Hadoop专用的压缩二进制文件格式。它专用于一个MapReduce作业和其他MapReduce作业之间传送数据。 Key：K（用户定义） Value：V（用户定义）
NLineInputFormat	与TextInputFormat相同，但每个分片一定有N行。N在属性mapred.line.input.format.linespermap中设定，默认为1。 key: LongWritable value: Text

KeyValueTextInputFormat在更结构化的输入文件中使用，由一个预定义的字符，通常为制表符（\ t），将每行（记录）的键与值分开。例如，一个以制表符分割的，由时间戳和URL组成的数据文件也许是这样的：

```
17:16:18    http://hadoop.apache.org/core/docs/r0.19.0/api/index.html
17:16:19    http://hadoop.apache.org/core/docs/r0.19.0/mapred_tutorial.html
17:16:20    http://wiki.apache.org/hadoop/GettingStartedWithHadoop
17:16:20    http://www.maxim.com/hotties/2008/finalist_gallery.aspx
17:16:25    http://wiki.apache.org/hadoop/
...
```

你可以设置JobConf对象使用KeyValueTextInputFormat类读取这个文件：

```
conf.setInputFormat(KeyValueTextInputFormat.class);
```

由前面的示例文件可知，mapper读取的第一个记录将包含一个键“17:16:18”和一个值“http://hadoop.apache.org/core/docs/r0.19.0/api/index.html”。而mapper读到的第二个记录将包含键“17:16:19”和值http://hadoop.apache.org/core/docs/r0.19.0/mapred_tutorial.html.，如此递推。

回想一下，我们以前在mapper中曾使用LongWritable和Text分别作为键（key）和值（value）的类型。在TextInputFormat中，因为值为用数字表示的偏移量，所以LongWritable是一个合理的键类型。而当使用KeyValueTextInputFormat时，无论是键和值都为Text类型，你必须改

www.ChinaDBA.net 中国DBA超级论坛

变Mapper的实现以及map()方法来适应这个新的键（key）类型。

输入到MapReduce作业的数据未必都是些外部数据。实际上，一个MapReduce作业的输入常常是其他一些MapReduce的输出。下面将看到，你还可以自定义输出格式。默认的输出格式与KeyValueTextInputFormat能够读取的数据格式保持一致（即记录中的每行均为一个由制表符分隔的键和值）。不过，Hadoop提供了更加有效的二进制压缩文件格式，称为序列文件。这个序列文件为Hadoop处理做了优化，当链接多个MapReduce作业时，它是首选格式。读取序列文件的InputFormat类为SequenceFileInputFormat。

序列文件的键和值的对象类型可由用户来定义。输出和输入类型必须匹配，Mapper实现和map()方法均须采用正确的输入类型。

2. 生成一个定制的InputFormat——InputSplit和RecordReader

有时你会期望采用与标准InputFormat类不同的方式读取输入数据。这时你必须编写自定义的InputFormat类。让我们看看要做哪些事。InputFormat是一个仅包含两个方法的接口。

```
public interface InputFormat<K, V> {
  InputSplit[] getSplits(JobConf job, int numSplits) throws IOException;
  RecordReader<K, V> getRecordReader(InputSplit split,
                                     JobConf job,
                                     Reporter reporter) throws IOException;
}
```

这两个方法总结了InputFormat需执行的两个功能：

- 确定所有用于输入数据的文件，并将之分割为输入分片。每个map任务分配一个分片。
- 提供一个对象(RecordReader)，循环提取给定分片中的记录，并解析每个记录为预定义类型的键与值。

谁又希望去考虑如何将文件划分为分片呢？在创建自己的InputFormat类时，你最好从负责文件分割的FileInputFormat类中继承一个子类。事实上，表3-4中所有的InputFormat类都是FileInputFormat类的子类。FileInputFormat实现了getSplits()方法，不过保留了getRecordReader()抽象让子类填写。FileInputFormat中实现的getSplits()把输入数据粗略地划分为一组分片，分片数目在numSplits中限定，且每个分片的大小必须大于mapred.min.split.size个字节，但小于文件系统的块。在实际情况中，一个分片最终总是以一个块为大小，在HDFS中默认为64 MB。

FileInputFormat有一定数量的protected方法，子类可以通过覆盖改变其行为，其中一个就是isSplitable(FileSystem fs, Path filename)方法。它检查你是否可以将给定文件分片。默认实现总是返回true，因此所有大于一个分块的文件都要分片。有时你可能想要一个文件为其自身的分块，这时你就可以覆盖isSplitable()来返回false。例如，一些文件压缩方案并不支持分割。（你不能从文件的中间开始读数据。）一些数据处理操作，如文件转换，需要把每个文件视为一个原子记录，也不能将之分片。

在使用FileInputFormat时，你关注于定制RecordReader，它负责把一个输入分片解析为记录，再把每个记录解析为一个键/值对。让我们看一下这个接口的样子。

www.ChinaDBA.net 中国DBA超级论坛

```
public interface RecordReader<K, V> {
  boolean next(K key, V value) throws IOException;
  K createKey();
  V createValue();

  long getPos() throws IOException;
  public void close() throws IOException;
  float getProgress() throws IOException;
}
```

我们不用自己去写RecordReader，还是利用Hadoop所提供的类。例如，LineRecordReader实现RecordReader < LongWritable, Text>。它在TextInputFormat中被用于每次读取一行，以字节偏移作为键，以行的内容作为值。而KeyValueLineRecordReader则被用在KeyValueTextInputFormat中。在大多数情况下，自定义RecordReader是基于现有实现的封装，并把大多数的操作放在next()方法中。

我们给出编写自定义InputFormat类的一个用例，它按照特定的类型读取记录，而不使用普通的Text类型。例如，我们以前使用的KeyValueTextInputFormat，是从数据文件中读取以制表符分隔的时间戳和URL数据。这个类最终会把时间戳和URL都作为Text类型。在这个例子中，我们创建一个TimeUrlTextInputFormat，它完成相同的工作，但把URL作为URLWritable类型[①]。如前面提到的，我们通过扩展FileInputFormat生成我们的InputFormat类，并实现一个factory方法来返回RecordReader。

```
public class TimeUrlTextInputFormat extends
    ➥FileInputFormat<Text, URLWritable> {
  public RecordReader<Text, URLWritable> getRecordReader(
      ➥InputSplit input, JobConf job, Reporter reporter)
      ➥throws IOException {
    return new TimeUrlLineRecordReader(job, (FileSplit)input);
  }
}
```

我们的URLWritable类非常简单：

```
public class URLWritable implements Writable {

  protected URL url;

  public URLWritable() { }

  public URLWritable(URL url) {
    this.url = url;
  }

  public void write(DataOutput out) throws IOException {
    out.writeUTF(url.toString());
  }

  public void readFields(DataInput in) throws IOException {
    url = new URL(in.readUTF());
  }
```

① 我们也许还希望时间键（key）采用非Text的类型。例如，我们可以为它定制一个CalendarWritableComparable接口。此处我们只作简单说明，而将它留给读者作练习。

www.ChinaDBA.net 中国DBA超级论坛

```
    public void set(String s) throws MalformedURLException {
      url = new URL(s);
    }
  }
```

除了类的构建函数之外，TimeUrlLineRecordReader会在RecordReader接口中实现6种方法。它主要是在KeyValueTextInputFormat之外的一个封装，但把记录的值从Text类型转换为URLWritable。

```
class TimeUrlLineRecordReader implements RecordReader<Text, URLWritable> {

  private KeyValueLineRecordReader lineReader;
  private Text lineKey, lineValue;

  public TimeUrlLineRecordReader(JobConf job, FileSplit split) throws
      IOException {
      lineReader = new KeyValueLineRecordReader(job, split);

      lineKey = lineReader.createKey();
      lineValue = lineReader.createValue();
  }

  public boolean next(Text key, URLWritable value) throws IOException {
    if (!lineReader.next(lineKey, lineValue)) {
      return false;
    }

    key.set(lineKey);
    value.set(lineValue.toString());

    return true;
  }

  public Text createKey() {
    return new Text("");
  }

  public URLWritable createValue() {
    return new URLWritable();
  }

  public long getPos() throws IOException {
    return lineReader.getPos();
  }

  public float getProgress() throws IOException {
    return lineReader.getProgress();
  }

  public void close() throws IOException {
    lineReader.close();
  }
}
```

TimeUrlLineRecordReader类生成一个KeyValueLineRecordReader对象，并直接把getPos()、getProgress()以及close()方法调用传递给它。而next()方法将lineValue Text对象转换为URLWritable类型。

www.ChinaDBA.net 中国DBA超级论坛

3.3.2 OutputFormat

当MapReduce输出数据到文件时，使用的是OutputFormat类，它与InputFormat类相似。因为每个reducer仅需将它的输出写入自己的文件中，输出无需分片。输出文件放在一个公用目录中，通常命名为part-*nnnnn*，这里*nnnnn*是reducer的分区ID。RecordWriter对象将输出结果进行格式化，而RecordReader对输入格式进行解析。

Hadoop提供几个标准的OutputFormat实现，如表3-5所示。毋庸置疑，几乎我们处理的所有OutputFormat类都是从FileOutputFormat抽象类继承来的，而InputFormat类继承自FileInputFormat。你可以通过调用JobConf对象中的setOutputFormat()定制OutputFormat，这个JobConf对象包含对MapReduce作业的配置信息。

注意 你可能会奇怪，为什么只将OutputFormat（InputFormat）和FileOutputFormat（FileInputFormat）区分开，似乎所有OutputFormat（InputFormat）类都扩展了FileOutputFormat（FileInputFormat）。是否有不处理文件的OutputFormat（InputFormat）类？没错，NullOutputFormat简单地实现了OutputFormat，并不需要继承FileOutputFormat。更重要的是，OutputFormat（InputFormat）类处理的是数据库，并非文件，而且在类的层次关系中OutputFormat（InputFormat）类是区别于FileOutputFormat（FileInputFormat）的一个独立分支。这些类有专门的应用，有兴趣的读者可以在网上搜寻在线Java文档进一步了解DBInputFormat和DBOutputFormat。

表3-5 主要的OutputFormat类，默认为TextOutputFormat

OutputFormat	描 述
TextOutputFormat<K,V>	将每个记录写为一行文本。键和值以字符串的形式写入，并以制表符（\t）分隔。这个分隔符可以在属性mapred.textoutputformat.separator中修改
SequenceFileOutputFormat<K,V>	以Hadoop专有序列文件格式写入键/值对。与SequenceFileInputFormat配合使用
NullOutputFormat<K,V>	无输出

默认的OutputFormat是TextOutputFormat，将每个记录写为一行文本。每个记录的键和值通过toString()被转换为字符串(string)，并以制作符(\t)分隔。分隔符可以在mapred.textoutputformat.separator属性中修改。

TextOutputFormat采用可被KeyValueTextInputFormat识别的格式输出数据。如果把键的类型设为NullWritable，它也可以采用可被TextInputFormat识别的输出格式。在这种情况下，在键/值对中没有键，也没有分隔符。如果想完全禁止输出，应该使用NullOutputFormat。如果让reducer采用自己的方式输出，并且不需要Hadoop写任何附加的文件，可以限制Hadoop的输出。

最后，SequenceFileOutputFormat以序列文件格式输出数据，使其可以通过SequenceFileInputFormat来读取。它有助于通过中间数据结果将MapReduce作业串接起来。

www.ChinaDBA.net 中国DBA超级论坛

3.4 小结

Hadoop是一个在数据处理中与众不同的软件框架。它有自己的文件系统HDFS，采用最适应数据密集型处理的数据存储方式。操作HDFS需要特殊的Hadoop工具，但幸运的是大多数工具采用的语法与Unix或者Java类似。

在Hadoop框架中，数据处理部分有一个更为流行的名称，即MapReduce。毋庸置疑，MapReduce程序的核心是Map和Reduce操作，但实际上这个框架还包含其他操作，如data splitting（数据分割）和shuffling（洗牌），它们对于框架的运行而言至关重要。同时，你还可以定制其他的操作，如Partitioning（分组）和Combining（合并）。另外，Hadoop还支持采用不同的格式读入和输出数据。

既然我们已经对Hadoop如何工作有了更好的了解，下面进入本书的第二部分，看一看编写实用Hadoop程序的各种技巧。

www.ChinaDBA.net 中国DBA超级论坛

www.ChinaDBA.net 中国DBA超级论坛

Part 2

第二部分

实　战

第二部分传授一些实用技能，帮助你在 Hadoop 上编写并运行数据处理程序。我们将通过多个示例来探讨，如何用 Hadoop 分析专利数据集，其中包含 Bloom filter 等高阶算法。我们还会介绍编程和管理技术，这对于将 Hadoop 用于生产非常有用。

本部分内容

- 第 4 章　编写 MapReduce 基础程序
- 第 5 章　高阶 MapReduce
- 第 6 章　编程实践
- 第 7 章　细则手册
- 第 8 章　管理 Hadoop

www.ChinaDBA.net 中国DBA超级论坛

第4章 编写MapReduce基础程序

本章内容

- 基于Hadoop的专利数据处理示例
- MapReduce程序框架
- 用于计数统计的MapReduce基础程序
- 支持用脚本语言编写MapReduce程序的Hadoop流式API
- 用于提升性能的Combiner

MapReduce的编程模型不同于你学过的大多数编程模型，熟悉它需要一些时间和实践。为了帮助你提升技能，我们将在接下来的几章里给出许多示例程序，它们会诠释各种MapReduce的编程技术。通过MapReduce的多种使用方式，你会开始从直觉和习惯上形成“MapReduce思维”。这些示例涵盖了从简单到高阶的应用。在其中一个高阶应用中，我们将介绍Bloom filter数据结构，它通常不会出现在标准的计算机科学课程中。无论是否正在使用Hadoop，你都会发现，处理大型数据集往往需要重新思考底层的算法。

想必你已经对Hadoop有了基本了解，并有能力安装Hadoop并编译和运行一个示例程序，如第1章中的单词计数。现在，让我们用真实的数据集作为示例。

4.1 获得专利数据集

要让Hadoop做的工作有意义就需要数据。本书中的许多示例都将使用专利数据集，它们可以从美国国家经济研究局（National Bureau of Economic Research，NBER）获得，网址为http://www.nber.org/patents/。这些数据集最初是为了“The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools”[①]这篇文章而编辑的。本书使用了专利引用数据集`cite75_99.txt`和专利描述数据集`apat63_99.txt`。

注意 这些数据集每个大约为250 MB，它们足够小，以便我们的示例可以在Hadoop的单机或者伪分布模式下运行。即使并没有一个真正的集群，你也可以用它们练习MapReduce的编

① NBER Working Paper 8498，Hall, B. H., A. B. Jaffe和M. Tratjenberg (2001)。

www.ChinaDBA.net 中国DBA超级论坛

程。你几乎无需改变代码，就能够非常确信你的MapReduce程序可以在100倍或者1000倍大的集群上进行数据处理，这是Hadoop的最大优点。

一个流行的开发策略是为生产环境中的大数据集建立一个较小的、抽样的数据子集，称为开发数据集。这个开发数据集可能只有几百兆字节。当你以单机或者伪分布模式编写程序来处理它们时，你会发现开发周期很短，在自己的机器上运行程序也很方便，而且还可以在独立的环境中进行调试。

我们之所以在示例程序中选择这两个数据集，是因为它们与你将来会遇到的大多数数据类型相似。首先，专利引用数据所构成的关系图与网页链接以及社会网络图可谓大同小异。其次，专利发布以时间为序，有些特性类似于时间序列。再次，每个专利关联到一个人（发明人）和一个位置（发明人的国家）。你可以将之视为个人信息或地理数据。最后，你可以将这些数据视为具有明确模式的普通数据库关系，而格式上简单地以逗号分开[①]。

4.1.1 专利引用数据

专利引用数据集涵盖了自1975年到1999年间对美国专利的引用。它有超过1600万行，前几行如下所示：

```
"CITING（引用）","CITED（被引）"
3858241,956203
3858241,1324234
3858241,3398406
3858241,3557384
3858241,3634889
3858242,1515701
3858242,3319261
3858242,3668705
3858242,3707004
...
```

数据集采用标准的逗号分隔取值（comma-separated values，CSV）格式。在第一行中给出列的描述，其他行每行记录一次引用。例如，第二行表明专利3858241引用了专利956203。文件中的各行按照引用专利进行排序。我们看到专利3858241一共引用了5个专利。进一步的量化分析可以加深我们对该数据的认识。

如果你仅仅读取数据文件，引用数据看起来就是一串数值。不过，你也可以通过更加有趣的方式来“思考”这些数据。一个方法是将之视为一张关系图。图4-1显示了这张引用图的一部分。我们可以看到一些专利经常被引用，另外一些专利基本没有被引用过[②]。还有一些专利如5936972

① 还有许多常见的数据类型超出了这两种数据集所能表达的范围。文本就是此处省略的一种重要数据类型，但它已经出现在单词计数的示例中。其他遗漏的类型包括XML、图像和地理位置（经纬度）。数学矩阵虽然没有被正式提及，但专利引用的关系图可以被认为是一种稀疏的0/1矩阵。

② 对于任何数据分析，当处理有限数据时，我们都要非常小心。当一个专利看似未引用其他专利时，它可能是较早的专利，我们缺少其引用信息。另一方面，较新的专利较少被引用，通常是因为只有更新的专利才能够知道它们的存在。

www.ChinaDBA.net 中国DBA超级论坛

和6009552，它们引用了相似的一组专利（4354269、4486882及5598422），但并没有相互引用。我们使用Hadoop来挖掘描述这些专利数据的统计结果，并找到那些虽不明显但却有趣的模式。

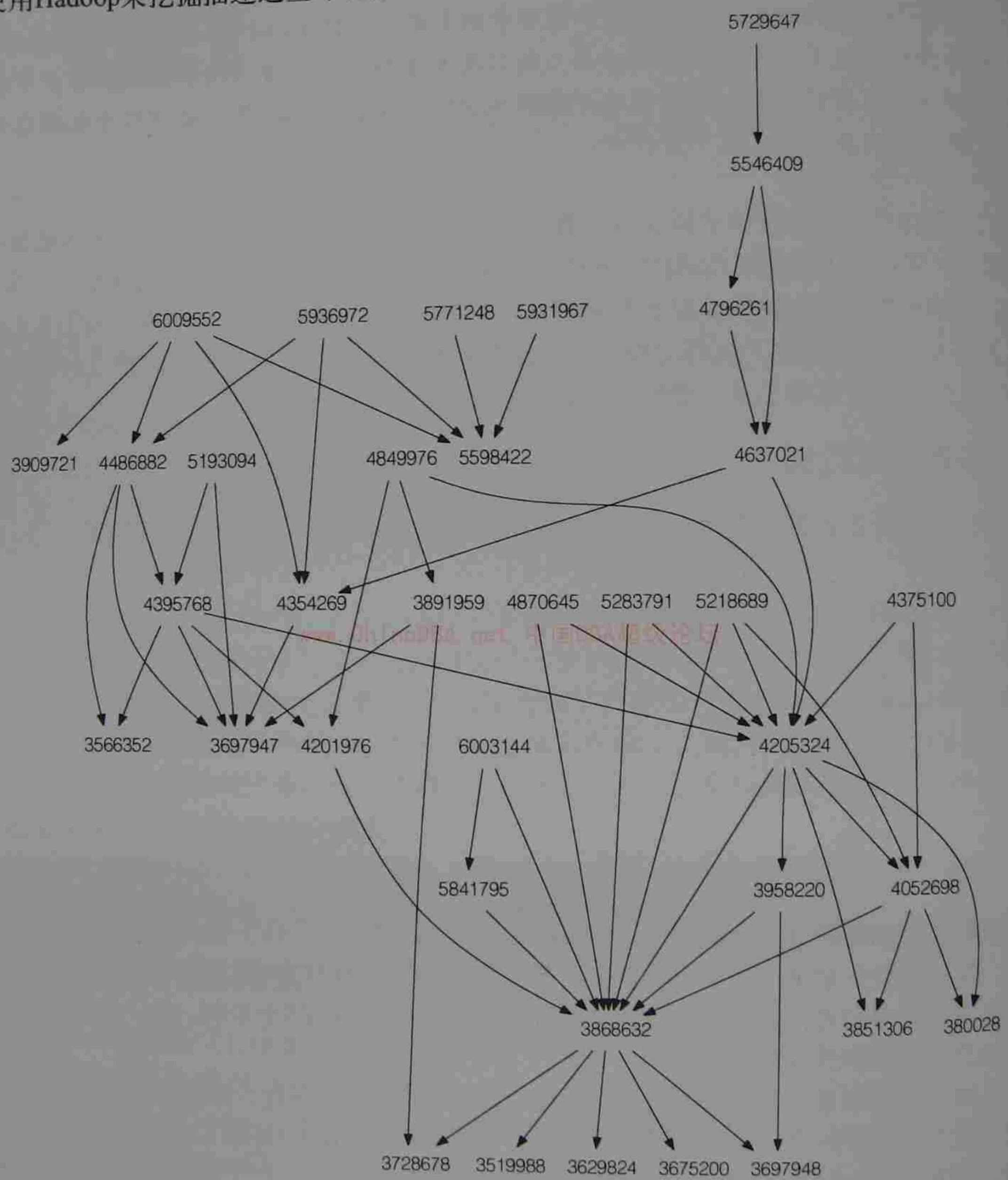


图4-1　专利引用数据集关系图的一部分。每个专利显示为一个顶点（节点），而每次引用为一个有向边（箭头）

4.1.2　专利描述数据

我们使用的另一个数据集是专利描述数据。其中包含专利号、专利申请年份、专利批准年份、声明数目和其他与专利相关的元数据。看看这个数据集的前面几行。它类似关系数据库中的一张

www.ChinaDBA.net 中国DBA超级论坛

表，但采用的是CSV格式。这个数据集有超过290万条记录。与许多真实数据集一样，它的许多值是缺失的。

```
"PATENT","GYEAR","GDATE","APPYEAR","COUNTRY","POSTATE","ASSIGNEE",
➥ "ASSCODE","CLAIMS","NCLASS","CAT","SUBCAT","CMADE","CRECEIVE",
➥ "RATIOCIT","GENERAL","ORIGINAL","FWDAPLAG","BCKGTLAG","SELFCTUB",
➥ "SELFCTLB","SECDUPBD","SECDLWBD"
3070801,1963,1096,,"BE","",,1,,269,6,69,,1,,0,,,,,,,
3070802,1963,1096,,"US","TX",,1,,2,6,63,,0,,,,,,,,,
3070803,1963,1096,,"US","IL",,1,,2,6,63,,9,,0.3704,,,,,,
3070804,1963,1096,,"US","OH",,1,,2,6,63,,3,,0.6667,,,,,,
3070805,1963,1096,,"US","CA",,1,,2,6,63,,1,,0,,,,,,,
3070806,1963,1096,,"US","PA",,1,,2,6,63,,0,,,,,,,,,
3070807,1963,1096,,"US","OH",,1,,623,3,39,,3,,0.4444,,,,,,
3070808,1963,1096,,"US","IA",,1,,623,3,39,,4,,0.375,,,,,,
3070809,1963,1096,,"US","AZ",,1,,4,6,65,,0,,,,,,,,,
```

第一行列出二十来个属性名称，它们仅对专利方面的专家才有价值。而我们即使不理解所有的属性，只要能了解其中的几点也仍然是很有用的。表4-1描述了前10个属性。

表4-1 专利描述数据集中前10个属性的定义

属性名	内容
PATENT	专利号
GYEAR	批准年
GDATE	批准日，自1960年1月1日以来的日期
APPYEAR	申请年（仅适用于自1967年起批准的专利）
COUNTRY	第一发明人国家
POSTATE	第一发明人所在州（若国家为美国）
ASSIGNEE	专利权人（即专利拥有者）数字标识
ASSCODE	1位（1~9）表示的专利权人类型。（专利权人类型包括美国个人、美国政府、美国组织及非美国个人等。）
CLAIMS	声明数目（仅适用于自1975年授权的专利）
NCLASS	3位表示的主要专利类型

既然已经有了两个专利数据集，让我们开始编写Hadoop程序来处理数据。

4.2 构建 MapReduce 程序的基础模板

大多数MapReduce程序的编写都可以简单地依赖于一个模板及其变种。当撰写一个新的MapReduce程序时，我们通常会采用一个现有的MapReduce程序，并将其修改成我们所希望的样子。在本节我们首先撰写第一个MapReduce程序并解释其不同的组成部分。这个程序将作为今后MapReduce程序的模板。

第一个程序将读取专利引用数据并对它进行倒排。对于每一个专利，我们希望找到那些引用它的专利并进行合并。输出大致如下：

www.ChinaDBA.net 中国DBA超级论坛

```
1       3964859,4647229
10000   4539112
100000  5031388
1000006 4714284
1000007 4766693
1000011 5033339
1000017 3908629
1000026 4043055
1000033 4190903,4975983
1000043 4091523
1000044 4082383,4055371
1000045 4290571
1000046 5918892,5525001
1000049 5996916
1000051 4541310
1000054 4946631
1000065 4748968
1000067 5312208,4944640,5071294
1000070 4928425,5009029
```

我们发现专利5312208、4944640及5071294引用了专利1000067。本节不会过多关注第3章阐述过的MapReduce数据流，而会着重讨论MapReduce程序的结构。我们仅用单一文件来形成一个完整的程序，正如你在代码清单4-1中所看到的。

代码清单4-1　典型Hadoop程序的模板

```
public class MyJob extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, Text, Text> {

        public void map(Text key, Text value,
                        OutputCollector<Text, Text> output,
                        Reporter reporter) throws IOException {

            output.collect(value, key);
        }
    }

    public static class Reduce extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key, Iterator<Text> values,
                           OutputCollector<Text, Text> output,
                           Reporter reporter) throws IOException {

            String csv = "";
            while (values.hasNext()) {
                if (csv.length() > 0) csv += ",";
                csv += values.next().toString();
            }
            output.collect(key, new Text(csv));
        }
    }

    public int run(String[] args) throws Exception {
```

www.ChinaDBA.net 中国DBA超级论坛

```
        Configuration conf = getConf();

        JobConf job = new JobConf(conf, MyJob.class);

        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);

        job.setJobName("MyJob");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.set("key.value.separator.in.input.line", ",");

        JobClient.runJob(job);

        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new MyJob(), args);

        System.exit(res);
    }
}
```

我们习惯用单个类来完整地定义每个MapReduce作业，这里称为MyJob类。Hadoop要求Mapper和Reducer必须是它们自身的静态类。这些类非常小，我们的模板将它们包含在MyJob类中作为内部类。这样做的好处是可以把所有的东西放在一个文件内，简化代码管理。但是需要记住这些内部类是独立的，通常不与MyJob类进行交互。在作业执行期间，采用不同JVM 的各类节点复制并运行Mapper和Reducer，而其他的作业类仅在客户机上执行。

我们稍后再说Mapper和Reducer类。不带有这些类的MyJob类框架为：

```
public class MyJob extends Configured implements Tool {

    public int run(String[] args) throws Exception {
        Configuration conf = getConf();

        JobConf job = new JobConf(conf, MyJob.class);

        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);

        job.setJobName("MyJob");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
```

www.ChinaDBA.net 中国DBA超级论坛

```
    job.set("key.value.separator.in.input.line", ",");
    JobClient.runJob(job);
    return 0;
  }
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new MyJob(), args);
    System.exit(res);
  }
}
```

框架的核心在run()方法中，也称为driver。它实例化、配置并传递一个JobConf对象命名的作业给JobClient.runJob()以启动MapReduce作业。（反过来，JobClient类与JobTracker通信让该作业在集群上启动。）JobConf对象将保持作业运行所需的全部配置参数。Driver需要在作业中为每个作业定制基本参数，包括输入路径、输出路径、Mapper类和Reducer类。另外，每个作业可以重置默认的作业属性，例如InputFormat、OutputFormat等。也可以调用JobConf对象中的set()方法填充任意的配置参数。一旦你传递JobConf对象到JobClient.runJob()，它就被视为作业的总体规划，成为决定这个作业如何运作的蓝本。

JobConf对象有许多参数，但我们并不希望全部的参数都通过编写driver来设置，可以把Hadoop安装时的配置文件作为一个很好的起点。同时，用户可能希望在命令行启动一个作业时传递额外的参数来改变作业配置。Driver可以通过自定义一组命令并自行处理用户参数，来支持用户修改其中的一些配置。因为经常需要做这样的任务，Hadoop框架便提供了ToolRunner、Tool和Configured来简化其实现。当它们在上面的MyJob框架中被同时使用时，这些类使得作业可以理解用户提供的被GenericOptionsParser支持的选项。例如，我们以前用如下的命令行执行过MyJob类：

```
bin/hadoop jar playground/MyJob.jar MyJob input/cite75_99.txt output
```

如果我们运行作业仅仅是想看到mapper的输出（出于调试的目的），可以用选项-D mapred.reduce.tasks=0将reducer的数目设置为0。

```
Bin/hadoop jar playground/MyJob.jar MyJob -D mapred.reduce.tasks=0
➥ input/cite75_99.txt output
```

即使我们的程序不能清楚理解-D选项也没关系。通过使用ToolRunner，MyJob可以自动支持表4-2中的选项。

表4-2 GenericOptionsParser支持的选项

选项	描述
-conf <configuration file>	指定一个配置文件
-D <property=value>	给JobConf属性赋值
-fs <local|namenode:port>	指定一个NameNode，可以是“local”
-jt <local|jobtracker:port>	指定一个JobTracker

www.ChinaDBA.net 中国DBA超级论坛

（续）

选　项	描　述
-files <list of files>	指定一个以逗号分隔的文件列表，用于MapReduce作业。这些文件自动地分布到所有节点，使之可从本地获取
-libjars <list of jars>	指定一个以逗号分隔的jar文件，使之包含在所有任务JVM的classpath中
-archives <list of archives>	指定一个以逗号分隔的存档文件列表，使之可以在所有任务节点上打开

模板中习惯将Mapper类称为MapClass，而将Reducer类称为Reduce。如果Mapper类命名为Map，看起来会更对称，但是Java已经有一个名为Map的类（接口）。Mapper和Reducer都是MapReduceBase的扩展。MapReduceBase是一个小类，分别为configure()和close()方法提供非操作性实现。我们使用configure()和close()方法来建立和清除map（reduce）任务。除非是更高级的作业，通常我们并不需要覆盖它们。

Mapper类和Reducer类如下所示：

```
public static class MapClass extends MapReduceBase
    implements Mapper<K1, V1, K2, V2> {

    public void map(K1 key, V1 value,
                    OutputCollector<K2, V2> output,
                    Reporter reporter) throws IOException { }
}

public static class Reduce extends MapReduceBase
    implements Reducer<K2, V2, K3, V3> {

    public void reduce(K2 key, Iterator<V2> values,
                       OutputCollector<K3, V3> output,
                       Reporter reporter) throws IOException { }
}
```

Mapper类的核心操作为map()方法，Reducer类为reduce()方法。每一个map()方法的调用分别被赋予一个类型为K1和V1的键/值对。这个键/值对由mapper生成，并通过OutputCollector对象的collect()方法来输出。你需要在map()方法中的合适位置调用：

```
output.collect((K2) k, (V2) v);
```

在Reducer中reduce()方法的每次调用均被赋予K2类型的键，以及V2类型的一组值。注意它必须与Mapper中使用的K2和V2类型相同。Reduce()方法可能会循环遍历V2类型的所有值。

```
while (values.hasNext()) {
    V2? v = values.next();
    ...
}
```

Reduce()方法还使用OutputCollector来搜集其键/值的输出，它们的类型为K3/V3。在reduce()方法中可以调用

```
output.collect((K3) k, (V3) v);
```

除了在Mapper和Reducer之间保持K2和V2的类型一致，还需要确保在Mapper和Reducer中

www.ChinaDBA.net 中国DBA超级论坛

使用的键值类型与在driver中设置的输入格式、输出键的类，以及输出值的类保持一致。使用KeyValueTextInputFormat意味着K1和V1必须均为Text类型。Driver则必须调用setOutputKeyClass()和setOutputValueClass()分别指定K2和V2的类。

最终，所有的键与值的类型必须是Writable的子类型，来确保Hadoop的序列化接口可以把数据在分布式集群上发送。事实上，键的类型实现了WritableComparable，它是Writable的子接口。键的类型还需额外支持compareTo()方法，因为在MapReduce框架中键会被用来进行排序。

4.3 计数

在外行看来，大多数统计就是计数，而且许多基础的Hadoop作业都包含了计数过程。我们在第1章已经见过单词计数的例子。我们希望从专利引用数据集中得到专利被引用的次数。这同样是计数。期望的输出结果如下：

```
1       2
10000   1
100000  1
1000006 1
1000007 1
1000011 1
1000017 1
1000026 1
1000033 2
1000043 1
1000044 2
1000045 1
1000046 2
1000049 1
1000051 1
1000054 1
1000065 1
1000067 3
```

在每个记录中，专利号与引用次数相关联。我们可以为此写一个MapReduce程序。如前所述，我们很少会从头写一个MapReduce程序。如果有一个以类似方式处理数据的MapReduce程序，你就可以复制并修改它使之符合你的要求。

我们已经有了一个获得反向引用索引的程序。可以修改这个程序来输出计数结果，而不是一个专利引用列表。我们只需修改Reducer。如果我们选择让计数结果输出的类型为IntWritable，就需要在Reducer代码中的3个地方进行声明。在以前的注释中我们称它们为V3。

```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, Text, Text, IntWritable> {

    public void reduce(Text key, Iterator<Text> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {

        int count = 0;
        while (values.hasNext()) {
```

www.ChinaDBA.net 中国DBA超级论坛

```
            values.next();
            count++;
        }
        output.collect(key, new IntWritable(count));
    }
}*
```

通过修改几行代码并匹配好类型，我们就有了一个新的MapReduce程序。这个程序看上去做的修改很少。让我们再看一个修改较多的例程，不过它依然保留了MapReduce基础程序的结构。

运行前面的例子之后，我们现在得到了一个数据集，包含了对每个专利的引用次数的统计。要想做一个很好的练习，可以对这些统计值再做统计。让我们生成一个引用计数的直方图。我们预计会看到一个有趣的引用计数分布，即大多数的专利仅被引用一次，而少部分被引用上百次。

注意 因为专利引用数据集仅包含1975～1999年之间的专利，所以引用统计的数目必然被低估。（在此时间段外的引用没有被计入。）我们也没有统计那些被引用次数为0的专利。尽管如此，分析依然是有价值的。

编写MapReduce程序的第一步是了解数据流。这个例子中，当mapper读取一个记录时，它忽略专利号并输出一个键/值对<citation_count，1>作为中间结果。Reducer将所有的“1”加起来算出每种引用次数的和，并将这个总值输出。

基于对数据流的理解，我们可以为输入、中间结果、输出的键/值对K1、V1、K2、V2、K3和V3设定类型。这里使用KeyValueTextInputFormat，它自动将每个输入记录根据分隔符划分为键/值对。在输入格式中设置K1和V1为文本。我们选择K2、V2、K3和V3的数据为IntWritable类型，因为它们的数据必然为整数，且使用该类型更加高效。

根据数据流和数据类型，你会很容易理解代码清单4-2所示的最终程序。你会发现它在结构上类似于目前我们看到的其他MapReduce程序。我们会在代码清单之后详细介绍这个程序。

代码清单4-2 CitationHistogram.java：计算不同引用次数专利的数目（1次，2次等）

```
public class CitationHistogram extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, IntWritable, IntWritable> {

        private final static IntWritable uno = new IntWritable(1);
        private IntWritable citationCount = new IntWritable();

        public void map(Text key, Text value,
                        OutputCollector<IntWritable, IntWritable> output,
                        Reporter reporter) throws IOException {

            citationCount.set(Integer.parseInt(value.toString()));
            output.collect(citationCount, uno);
        }
```

www.ChinaDBA.net 中国DBA超级论坛

```
    }
    public static class Reduce extends MapReduceBase
        implements Reducer<IntWritable,IntWritable,IntWritable,IntWritable>
    {
        public void reduce(IntWritable key, Iterator<IntWritable> values,
                           OutputCollector<IntWritable, IntWritable>output,
                           Reporter reporter) throws IOException {

            int count = 0;
            while (values.hasNext()) {
                count += values.next().get();
            }
            output.collect(key, new IntWritable(count));
        }
    }
    public int run(String[] args) throws Exception {
        Configuration conf = getConf();

        JobConf job = new JobConf(conf, CitationHistogram.class);

        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);

        job.setJobName("CitationHistogram");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);

        JobClient.runJob(job);

        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                                 new CitationHistogram(),
                                 args);
        System.exit(res);
    }
}
```

现在类名为CitationHistogram，新的名字将替换程序中所有与MyJob相关的部分。Main()方法基本相同。Driver几乎没有改变。输入格式和输出格式仍然分别为KeyValueTextInputFormat和TextOutputFormat。主要的变化在输出的键和值的类，它们现在改为IntWritable来匹配K2和V2的新类型。我们还去掉了这一行：

```
job.set("key.value.separator.in.input.line", ",");
```

它用于设置KeyValueTextInputFormat所采用的分隔符，把每个输入行划分为一个键/值

www.ChinaDBA.net 中国DBA超级论坛

对。以前使用逗号来处理原始的专利引用数据。这里不对这个属性进行设置，分隔符默认采用制表符（tab），它适合于分隔引用计数数据。

这个mapper的数据流与以前mapper的相似，只是这里我们选择定义和使用了一对类变量citationCount和uno。

```
public static class MapClass extends MapReduceBase
    implements Mapper<Text, Text, IntWritable, IntWritable> {

    private final static IntWritable uno = new IntWritable(1);
    private IntWritable citationCount = new IntWritable();

    public void map(Text key, Text value,
                    OutputCollector<IntWritable, IntWritable> output,
                    Reporter reporter) throws IOException {

        citationCount.set(Integer.parseInt(value.toString()));
        output.collect(citationCount, uno);
    }
}
```

Map()方法中多出了一行，用于设置citationCount做类型转换。出于对效率的考虑，citationCount和uno的定义被放在类中而不是方法中。有多少记录，Map()方法就会被调用多少次（对每个JVM而言，就是一个分片中的记录数）。减少在map()方法中生成的对象个数可以提高性能，并减少垃圾回收。由于citationCount和uno被传递给output.collect()，我们依赖output.collect()方法的约定不会修改这两个对象[1]。

Reducer计算每个key对应的值的总数。这似乎并不高效，因为我们知道所有的值都是1（确切来说是uno）。为什么我们还要去加它们呢？原因在于，它会让我们在以后能更容易地增加一个combiner来提高性能。与MapClass不同，在Reduce中调用output.collect()会实例化一个新的IntWritable而非重用一个现有的。

```
output.collect(key, new IntWritable(count));
```

我们也可以通过使用一个IntWritable类的变量来提高reduce()的性能。但是在这个特定程序中，reduce()被调用的次数要小得多，大概不足一千次（包含所有的reducer）。我们不太需要优化这段代码。

在引用计数数据上运行MapReduce作业将显示如下的结果。正如我们推测的那样，大量的专利（900K+）仅有一个引用，而有些则有上百个引用。最多的一个专利有779个引用。

```
1       921128
2       552246
3       380319
4       278438
5       210814
6       163149
7       127941
8       102155
```

① 我们看到在5.1.3节中，这个依赖将禁止ChainMapper使用pass-by-reference。

www.ChinaDBA.net 中国DBA超级论坛

```
9       82126
10      66634
...
411     1
605     1
613     1
631     1
633     1
654     1
658     1
678     1
716     1
779     1
```

因为这个直方图输出只有几百行，我们可以将它放在数据表中并绘制成图。图4-2显示了不同引用频率的专利个数。图形采用双对数坐标系。在双对数图中，当分布为直线时，可以被认为呈指数分布。引用计数的直方图似乎与此相符，不过其曲率接近于抛物线，也可以被认为呈对数正态分布。

迄今为止，在示例程序中显示的MapReduce程序通常都不太大，你可以采用特定的结构来简化开发，而把大部分工作放在对数据流的思考上。

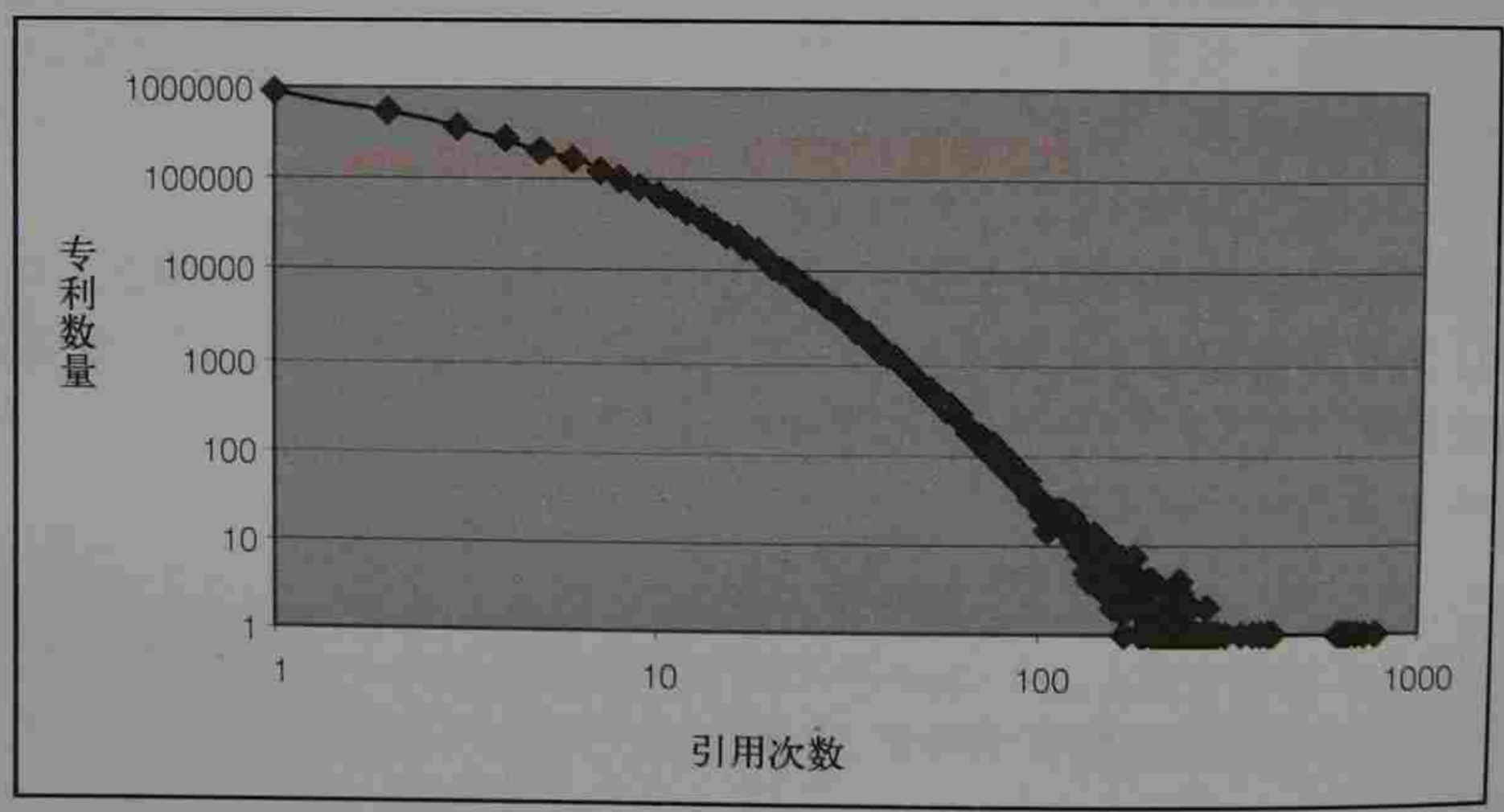


图4-2　本图给出了不同引用频率的专利数目。许多专利被引用一次（未被引用的没有显示在图中）。一些专利则被引用上百次。在双对数坐标中，该数据线看起来非常接近直线，可以被认为呈指数分布

4.4　适应 Hadoop API 的改变

稳定且可扩展的MapReduce API是推动Hadoop 1.0主版本发布的核心设计目标之一。本书撰写时，最新版本为0.20，并被视为是在（我们在本书通篇使用的）旧API和未来稳定API之间的过渡。为了保持向后兼容，版本0.20支持下一代API并将旧API标记为弃用。0.20之后的版本将停止支持旧API。本书撰写时，我们并不推荐马上转向新API，原因如下。

(1) 在0.20中，许多Hadoop自身的类库还没有基于新的API重写。如果MapReduce代码使用

www.ChinaDBA.net 中国DBA超级论坛

0.20中的新API，这些类将无法被使用。

(2) 在本书撰写时，0.18.3仍被很多人认为是最完备与最稳定的版本。虽然一些用户已经准备接受0.20版，不过我们建议你可以再稍等一段时间。[①]

当你阅读本书时情况也许会有所变化。本节内容将涵盖新API所做的改变。值得庆幸的是，几乎所有的改变仅仅影响基础的MapReduce模板。我们基于新API来重写这个模板以备将来使用。

首先值得注意的是，在新的API中org.apache.hadoop.mapred的许多类都被移走了。多数被放入org.apache.hadoop.mapreduce，而且类库都放在org.apache.hadoop.mapreduce.lib的一个包里。当转为使用新API时，org.apache.hadoop.mapred下所有类的import声明（或者完全引用）就不存了，它们都被弃用。

新API中最有益的变化是引入了上下文对象context。最直接的影响在于替换了map()和reduce()方法中使用的OutputCollector和Reporter对象。现在将通过调用Context.write()而不是OutputCollector.collect()输出键/值对。深远的影响是统一了应用代码和MapReduce框架之间的通信，并固定了Mapper和Reducer的API，使得添加新功能时不会改变基本方法签名。新的功能仅仅是在context对象上增加的方法。在引入这些方法之前写的程序不会感知到这些新方法，它们可以在更新的版本中继续编译与运行。

新的map()和reduce()方法分别被包含在新的*抽象类*Mapper和Reducer中。它们取代了原始API中的Mapper和Reducer*接口*（org.apache.hadoop.mapred.Mapper和org.apache.hadoop.mapred.Reducer）。新的抽象类也替换了MapReduceBase类，使之被弃用。

新的map()和reduce()方法多了一两处细微的改变。它们可以抛出InterruptedException而非单一的IOException。而且，reduce()方法不再以Iterator而以Iterable来接受一个值的列表，这样更容易使用Java的foreach语义来实现迭代。我们可以总结迄今为止讨论过的对MapClass和Reduce方法签名的修改。回顾一下在原始API中的签名：

```
public static class MapClass extends MapReduceBase
    implements Mapper<K1, V1, K2, V2> {

    public void map(K1 key, V1 value,
                    OutputCollector<K2, V2> output,
                    Reporter reporter) throws IOException { }
}

public static class Reduce extends MapReduceBase
    implements Reducer<K2, V2, K3, V3> {

    public void reduce(K2 key, Iterator<V2> values,
                       OutputCollector<K3, V3> output,
                       Reporter reporter) throws IOException { }
}
```

新API一定程度上对它们做了简化：

① 你可能会奇怪为什么不提版本0.19。一般的意见认为它的初始版本问题比较多，有许多bug。一些副版本试图解决这些问题，但社区似乎希望直接跳到版本0.20。

www.ChinaDBA.net 中国DBA超级论坛

```
public static class MapClass extends Mapper<K1, V1, K2, V2> {
    public void map(K1 key, V1 value, Context context)
                    throws IOException, InterruptedException { }
}
public static class Reduce extends Reducer<K2, V2, K3, V3> {
    public void reduce(K2 key, Iterable<V2> values, Context context)
                    throws IOException, InterruptedException { }
}
```

你还需要改变driver中的一些内容来支持新的API。在新API中JobConf和JobClient被替换了。它们的功能已经被放入Configuration类（它原本是JobConf的父类）和一个新的类Job中。Configuration类纯粹为了配置作业而设，而Job类负责定义和控制一个作业的执行。诸如setOutputKeyClass()和setOutputValueClass()等方法被从JobConf转移到了Job。作业的构造和提交执行现在放在Job中。原本需要使用JobConf来构造一个作业：

```
JobConf job = new JobConf(conf, MyJob.class);
job.setJobName("MyJob");
```

而现在可通过Job类完成：

```
Job job = new Job(conf, "MyJob");
job.setJarByClass(MyJob.class);
```

以前是通过JobClient提交作业去执行：

```
JobClient.runJob(job);
```

现在同样通过Job类来完成：

```
System.exit(job.waitForCompletion(true)?0:1);
```

在代码清单4-3中，使用Hadoop 0.20中的新API重写了代码清单4-1中的模板程序。它包含了所有我们在本节所提到的改变。

代码清单4-3 基于版本0.20新API重写的Hadoop基础程序模板（代码清单4-1）

```
public class MyJob extends Configured implements Tool {
    public static class MapClass
                    extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context context)
                    throws IOException, InterruptedException {
        String[] citation = value.toString().split(",");
        context.write(new Text(citation[1]), new Text(citation[0]));
    }
}
public static class Reduce extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values,
                    Context context)
                    throws IOException, InterruptedException {
        String csv = "";
```

www.ChinaDBA.net 中国DBA超级论坛

```
        for (Text val:values) {
            if (csv.length() > 0) csv += ",";
            csv += val.toString();
        }

        context.write(key, new Text(csv));
    }
}

public int run(String[] args) throws Exception {
    Configuration conf = getConf();

    Job job = new Job(conf, "MyJob");
    job.setJarByClass(MyJob.class);

    Path in = new Path(args[0]);
    Path out = new Path(args[1]);
    FileInputFormat.setInputPaths(job, in);
    FileOutputFormat.setOutputPath(job, out);

    job.setMapperClass(MapClass.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    System.exit(job.waitForCompletion(true)?0:1);

    return 0;
}

public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new MyJob(), args);

    System.exit(res);
}
}
```

Iterable类型允许foreach循环

❶ 兼容的InputFormat类

这段代码实现的反向索引功能与代码清单4-1相同，但使用了版本0.20的API。不过我们在代码清单4-1中使用的KeyValueTextInputFormat类未被移植到版本0.20的新API中。重写这个模板时我们不得不使用TextInputFormat❶。希望发布版本0.21时，新的API能够支持所有的Hadoop类。为了让本书后续章节中示例程序的展示保持一致，我们继续使用版本0.20以前的API。

4.5 Hadoop 的 Streaming

目前，我们的Hadoop程序都是用Java写的。Hadoop也支持用其他语言来编程，这需要用到名为Streaming的通用API。在实际应用中，Streaming主要用作编写简单、短小的MapReduce程序，它们可以通过脚本语言编程，开发更加快捷，并能够充分利用非Java库。

Hadoop Streaming使用Unix中的流与程序进行交互。从STDIN输入数据，而输出到STDOUT。数据必须为文本而且每行被视为一个记录。注意，这正是许多Unix命令的工作方式，而Hadoop Streaming允许这些命令被用作mapper和reducer。如果你熟悉Unix命令的使用，如用于数据处理的

www.ChinaDBA.net 中国DBA超级论坛

wc、cut或uniq，你就可以通过Hadoop Streaming把它们用在大数据集处理上。

Hadoop Streaming的整个数据流就像一个管道，数据流先流过mapper，然后在reducer中排序，最后输出。我们采用Unix命令行来表示伪代码，写为：

```
cat [input_file] | [mapper] | sort | [reducer] >[output_file]
```

下面的例子将解释如何通过Unix命令来使用Streaming。

4.5.1　通过 Unix 命令使用 Streaming

在第一个例子中，让我们从cite75_99.txt中读取一列被引用的专利。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➡ -input input/cite75_99.txt
        ➡ -output output
        ➡ -mapper 'cut -f 2 -d ,'
        ➡ -reducer 'uniq'
```

就是这样！只有一行命令。我们来看看命令的每个部分都做了什么。

Streaming API位于contrib软件包中的contrib/streaming/hadoop-*-streaming.jar。第一部分和-input和-output参数为我们正在运行的Streaming程序设定了相应的输入输出文件或目录。Mapper和reducer通过引号中的参数进行设定。我们看到mapper使用了Unix的cut命令来提取第二列的数据，并声明列是被逗号分隔的。在引用数据集中，这个列对应被引用专利的专利号。接着，这些专利号被排序并传递给reducer。Reducer中的uniq命令对排序后的数据进行去重。命令的输出结果为：

```
"CITED"
1
10000
100000
1000006
...
999973
999974
999977
999978
999983
```

第一行为来自原始文件的列描述“CITED”。请注意，行是按字母排序的，这是因为Streaming完全采用文本方式处理数据，而不知道其他的数据类型。

在得到被引用专利的列表后，我们可能还希望了解其数量。我们再用Streaming来做快速统计，这次使用Unix命令wc-l。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➡ -input output
        ➡ -output output_a
        ➡ -mapper 'wc -l'
        ➡ -D mapred.reduce.tasks=0
```

这里使用wc -l作为mapper来统计每个分片中的记录数。Hadoop Streaming（自版本0.19.0起）

www.ChinaDBA.net 中国DBA超级论坛

支持GenericOptionsParser。参数-D被用于设定配置属性。我们希望mapper直接输出记录的统计结果而不经过reducer，于是设mapred.reduce.tasks为0，并不设置-redcuer选项[①]。最终结果为3258984，即根据我们的数据，有超过3百万个专利被引用。

4.5.2 通过脚本使用 Streaming

Hadoop Streaming允许我们使用任何可执行的脚本来处理按行组织的数据流，数据取自UNIX的标准输入STDIN，并输出到STDOUT。例如，在代码清单4-4中的Python脚本随机地从STDIN中采样数据。可能有读者不了解Python，我们解释一下这个程序。它有一个for循环每次从STDIN中读取一行。我们在每一行从1~100随机选择一个整数，并核对用户设定的参数（sys.argv[1]）。通过比较决定是否输出该行还是忽略它。你可以使用Unix中的脚本来均匀采样一个按行组织的数据文件，如：

```
cat input.txt | RandomSample.py 10 >sampled_output.txt
```

上面的命令调用Python脚本，参数设为10；sampled_output.txt（大约）包含input.txt中十分之一的记录。我们实际上可以设定从1~100任意的整数来得到相应比例的输入数据。

代码清单4-4 RandomSample.py：一个随机打印STDIN输入行的Python脚本

```
#!/usr/bin/env python
import sys, random

for line in sys.stdin:
    if (random.randint(1,100) <= int(sys.argv[1])):
        print line.strip()
```

我们可以在Hadoop中应用同样的脚本来得到一个数据集样本。采样的数据集通常用于程序开发，因为你可以基于它在单机或伪分布模式下递归地快速调试Hadoop程序。而且，当你在寻找数据的“描述”信息时，处理较小数据集所带来的速度和便利比精度损失更为重要。这种描述信息的一个例子是求数据的聚类。在R、MATLAB以及其他软件包中，已经有了各种聚类算法的优化实现。更好的选择是在采样数据上应用一些标准的软件包，而非试图用Hadoop上的分布式聚类算法处理所有的数据。

警告 数据采样所带来的精度损失重要与否取决于要计算的是什么以及数据集如何分布。例如，通过一个采样数据集来计算平均值通常是有效的，但是如果数据集偏斜得很严重，会使平均值取决于少数几个值，这时采样就会带来问题。类似地，如果仅仅为了得到数据的概貌，对采样数据做聚类通常是有效的。但是如果希望找到一个较小且不规则的聚类，采样会使得它们被排除在外。另外，最大值和最小值等功能也不适合对数据进行采样。

① 你可能注意到统计的是每个分片中的记录数，而不是整个文件。对于一个更大的文件，或者多个文件，用户必须自己将记录数合并才能得到完整的结果。为使整个计数过程能够自动完成，用户不得不在reducer中写一个脚本来合并所有部分的统计值。

www.ChinaDBA.net 中国DBA超级论坛

使用Streaming运行RandomSample.py就像使用Streaming运行Unix命令，不同在于Unix命令已经可以在集群上所有节点运行，而RandomSample.py则不行。Hadoop Streaming支持-file选项，可以把可执行文件打包成作业提交的一部分[①]。用来执行RandomSample.py的命令为：

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➡ -input input/cite75_99.txt
        ➡ -output output
        ➡ -mapper 'RandomSample.py 10'
        ➡ -file RandomSample.py
        ➡ -D mapred.reduce.tasks=1
```

通过设定mapper为'RandomSample.py 10'，我们按十分之一的比例采样。注意reducer的个数(mapred.reduce.tasks)已被设置为1。因为没有设定特殊的reducer，这里默认使用IdentityReducer。顾名思义，IdentityReducer是把输入直接转向输出。这里可以设置reducer的个数为任一非零的值，从而得到一组确定数目的输出文件。如果不这样做，也可以设置reducer为0，而让输出文件的个数等于mapper的数目。这也许对采样任务而言还不够完美，因为每个mapper的输出仅为输入的很小一部分，而且结果会是许多小文件。接下来还需要做一些简单的修补工作，通过HDFS的Shell命令getmerge或其他文件操作，可以获得正确数目的输出文件。用户可以根据个人喜好来选择用上述哪种方法。

随机采样脚本是用Python实现的，但只要是基于STDIN和STDOUT的脚本语言都可以。为了说明，我们用PHP[②]（代码清单4-5）重写了一个相同的脚本。执行如下的Streaming脚本。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➡ -input input/cite75_99.txt
        ➡ -output output
        ➡ -mapper 'php RandomSample.php 10'
        ➡ -file RandomSample.php
        ➡ -D mapred.reduce.tasks=1
```

代码清单4-5 RandomSample.py：一个随机打印STDIN输入行的PHP脚本

```
<?php

while (!feof(STDIN)) {
    $line = fgets(STDIN);
    if (mt_rand(1,100) <= $argv[1]) {
        echo $line;
    }
}
```

① 这里假设集群上每个节点已经安装了Python语言。

② 你可能已经注意到在代码清单4-5中，没有终括号?>来关闭开括号<?php。回想一下PHP最初是设计用来处理静态HTML内容的。在PHP括号<?php ... ?>之外的都被视为被输出的静态内容。当使用PHP作为一个纯脚本语言时，你需要注意不要在括号之外留有空格。否则它们会被输出，并可能导致难以调试的意外操作。（在输出数据中会莫名其妙地出现空格。）
通过把括号放在脚本文件的开始，就很容易保证在开括号<?php之前没有空格。但是，很容易在闭括号?>之后意外地留出空格，因为结尾的空格不容易被注意到。当把一个文件用作PHP脚本时，更安全的办法是省略闭括号?>。PHP解释器将很安静地将文件尾之前的内容当作PHP命令读完，而不是将之视为静态内容。

www.ChinaDBA.net 中国DBA超级论坛

随机的采样脚本不需要任何定制化的reducer，但你并不总是能够写出这样的Streaming程序。因为Streaming总被用来解决实际问题，让我们再看另外一个练习。这次我们生成一个定制化的Reducer。

假设我们想找到声明个数最多的一个专利。在专利描述数据集中，特定专利的声明个数位于第九列。我们的任务就是从专利描述数据中找到第九列的最大值。

在Streaming中，每个mapper都会看到完整的数据流，也正是由mapper负责（按行）将数据流分割为记录。而在标准的Java模式中，是由框架自身将输入数据分割为记录，每次仅将一个记录传给map()方法。Streaming模式可以很容易地维护一个分片中跨记录的状态信息，我们将利用这一点来计算最大值。虽然标准的Java模式也可以维护一个分片中跨记录的状态，但更为复杂。我们将在下一章讨论它。

在生成一个计算最大值的Hadoop程序时，我们可利用最大值的分配律特征。给定一个分割为多片的数据集，全局最大值为每个分片最大值的最大值。这听起来绕口，但用一个简单的例子就能说清楚。如果有4个记录X1、X2、X3和X4，并且它们被分为两个分片(X1, X2)和(X3, X4)，我们找4个记录的最大值可以通过先搜寻每个分片，再挑出最大的一个，或写为

```
max(X1,X2,X3,X4) = max(max(X1,X2), max(X3,X4))
```

我们的策略是让mapper独自计算每个分片的最大值。每个mapper将最终输出一个值。我们让一个reducer看到所有这些值，并输出一个全局最大值。代码清单4-6描述了让mapper计算一个分片最大值的Python脚本。

代码清单4-6 AttributeMax.py：找到某个属性中最大值的Python脚本

```
#!/usr/bin/env python

import sys

index = int(sys.argv[1])
max   = 0
for line in sys.stdin:
    fields = line.strip().split(",")
    if fields[index].isdigit():
        val = int(fields[index])
        if (val > max):
            max = val
else:
    print max
```

这个脚本并不复杂。For循环每次读取一个记录，将记录放入fields中。如果记录中用户设定的field值更大，则用它来更新最大值。注意mapper没有输出任何值，直到最后才输出整个分片的最大值。这与以前我们看到的不同，那时每个记录会输出一个或多个中间结果提供给reducer进行处理。

考虑到mapper的输出非常少，可以使用默认的IdentityReducer来记录（排序的）mapper输出结果。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➥ -input input/apat63_99.txt
```

www.ChinaDBA.net 中国DBA超级论坛

```
➥ -output output
➥ -mapper 'AttributeMax.py 8'
➥ -file playground/AttributeMax.py
➥ -D mapred.reduce.tasks=1
```

Mapper为'AttributeMax.py 8'。它输出一个分片第9列的最大值。由一个reducer搜集所有的mapper输出。假设有7个mapper，上述命令最终的输出为：

```
0
260
306
348
394
706
868
```

每一行记录了一个特定分片的最大值。我们看到其中一个分片上所有记录的声明个数均为0。这很可疑，但是回想一下就会明白，这是因为1975年前的专利数据中缺少了声明数属性。

可以看到mapper给出了正确的结果。我们使用一个reducer得到mapper输出值中的最大值。有趣的是，由于最大值的分配律特征，AttributeMax.py还可以被用作reducer。只是现在reducer是要去寻找第“1”列中的最大值。

```
bin/hadoop jar contrib/streaming/hadoop-0.18.1-streaming.jar
    ➥ -input input/apat63_99.txt
    ➥ -output output
    ➥ -mapper 'AttributeMax.py 8'
    ➥ -reducer 'AttributeMax.py 0'
    ➥ -file AttributeMax.py
    ➥ -D mapred.reduce.tasks=1
```

上述命令会输出一个单行的文件，并且你会看到专利中声明数的最大值为868。

聚合函数的类别

我们使用聚合函数来计算描述统计。它们通常分为3类：分配型、代数型和全集型。最大值函数是分配型的一个例子。其他分配型函数包括最小值、总和和计数。顾名思义，分配型函数具有分配律特征。与最大值函数相似，你可以在逻辑上循环地将这些函数应用到更小块的数据上。

另一个聚合函数类型是代数型函数。这个类型的例子包括平均值和方差。它们不遵循分配律特征，并且它们的推导是要在简单的函数上做一些“代数”运算。我们将在下一节讨论其实例。

最后，像中值和K个最小/最大值这样的函数属于聚合函数中的全集型。喜欢挑战的读者可以尝试使用Hadoop以有效的方式实现一个中值函数。

4.5.3　用 Streaming 处理键/值对

至此你可能会想知道，以键/值对的方式来处理已编码的记录究竟会怎样。迄今为止，在我

www.ChinaDBA.net 中国DBA超级论坛

们对Streaming的讨论中，将每个记录视为一个原子单元，而不是由键和值组成的。事实上，Streaming处理键/值对的方法与标准Java的MapReduce模型并无二致。默认情况下，Streaming使用制表符分离记录中的键与值。当没有制表符时，整个记录被视为键，而值为空白文本。对于我们的数据集，由于没有制表符，好像是将每个单独记录作为一个整体来处理。而且，即使记录确实含有制表符，只会引起Streaming API对记录进行洗牌和排序的顺序不同。只要我们的mapper和reducer以记录形式工作，我们总会有这种对记录做处理的错觉。

使用键/值对允许我们利用基于键的洗牌和排序来做一些有趣的数据分析。为了解释基于Streaming的键/值对处理，我们可以写一个程序为每个国家找到专利声明数的最大值。这不同于`AttributeMax.py`，因为它是试图为每个键寻找最大值，而不是在所有记录范围内的最大值。为了让练习更有趣一些，我们计算其平均值，而不是寻找最大值。（正如我们所见，Hadoop已经包含了一个名为Aggregate的包，它所包含的类可以用来为每个键寻找最大值。）

首先让我们看一看，在MapReduce数据流的每一步上，如何通过Streaming API来操作键/值对。

(1) 正如我们所看到的，Streaming下的mapper通过`STDIN`读取一个分片，并将每一行提取为一个记录。Mapper可以选择是把每条输入记录翻译为一个键/值对，还是翻译为一行文本。

(2) 对于mapper输出的每一行，Streaming API都将之翻译为一个用制表符分隔的键/值对。类似于标准的MapReduce模型，我们用patitioner来处理键，通过洗牌得到记录对应的reducer。所有键一致的键/值对最终均进入相同的reducer。

(3) 通过Streaming API，每个reducer都以键为基准来排序键/值对。回顾在Java模式中，所有相同键的键/值对被结组为一个键和一列值，于是`reducer()`方法就会得到这个组。而在Streaming API中，是由reducer来负责处理分组。这还不至于太糟，因为键/值对已经按键来排序了。相同键的全部记录都在一个连续块中。你的reducer会每次从`STDIN`中读取一行，并跟踪那些新的键。

(4) 若应用于实践，reducer的输出（`STDOUT`）直接被写入到一个文件中。技术上讲，在写文件之前还有一个无操作的步骤。在这个步骤中，Streaming API把reducer输出的每一行用制表符分开，并将键/值对送到默认的`TextOutputFormat`中，即在结果被写入文件以前，默认地重新插入一个制表符。虽然缺少由reducer输出的制表符，Streaming API给出了相同的无操作行为。你可以重新配置默认的行为来做些不同的事情，但最好还是让它仍为无操作，而把处理放入reducer中。

为了更好地理解这个数据流，我们写一个Streaming程序来计算每个国家专利声明数的平均值。mapper提取每个专利的国家和声明个数，并将它们封装为一个键/值对。根据Streaming默认的假设，mapper输出的这个键/值对是以制表符分隔的。Streaming API将取得这个键，洗牌过程将保证所有来自一个国家的声明数最终都被送给同一个reducer。我们可以在代码清单4-7中看到这个Python代码。对于每个记录，mapper提取国名字段（`fields[4][1:-1]`）为键，提取声明个数字段（`fields[8]`）为值。需要额外考虑到我们的数据集存在缺失的值。我们增加一个条件判断来跳过那些缺少声明个数的记录。

www.ChinaDBA.net 中国DBA超级论坛

代码清单4-7　AverageByAttributeMapper.py：输出专利的国家和声明数

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    fields = line.split(",")
    if (fields[8] and fields[8].isdigit()):
        print fields[4][1:-1] + "\t" + fields[8]
```

在写reducer之前，让我们在两种情况下运行mapper：不用任何reducer和使用默认的IdentityReducer。这是一个学习它的好办法，因为我们可以准确地看到mapper形成了什么样的输出（通过无reducer的方式），以及reducer接收了什么样的输入（通过使用IdentityReducer）。你今后在调试MapReduce程序时会发现它很便利。至少你可以检查mapper是否正在输出正确的数据，以及正确的数据是否被发送给了reducer。首先，让我们在无reducer的方式下运行mapper。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➥ -input input/apat63_99.txt
        ➥ -output output
        ➥ -file playground/AverageByAttributeMapper.py
        ➥ -mapper 'AverageByAttributeMapper.py'
        ➥ -D mapred.reduce.tasks=0
```

输出由行组成，每行包含一个国家代码，后面跟着一个制表符，然后是一个计数值。输出记录的顺序不是按（新的）键排序的。虽然从输出结果上看并不明显，但事实上，它的顺序和输入记录是一致的。

更有趣的是使用具有非零个reducer的IdentityReducer。我们看一下在redcuer上洗牌和排序后的记录是怎样的。简单起见，让我们通过-D mapred.reduce.tasks=1把reducer数设为1个，来看看前32个记录。

AD	9
AD	12
AD	7
AD	28
AD	14
AE	20
AE	7
AE	35
AE	11
AE	12
AE	24
AE	4
AE	16
AE	26
AE	11
AE	4

AE	23
AE	12
AE	16
AE	10
AG	18
AG	12
AG	8
AG	14
AG	24
AG	20
AG	7
AG	3
AI	10
AM	18
AN	5
AN	26

使用Streming API时，reducer从STDIN看到这些文本数据。我们必须让reducer用制表符将每一行分隔开以恢复键/值对。排序之后键相同的记录就“结组”在一起了。当从STDIN读取每一行时，你有责任把带有不同键的记录之间的边界找到。注意虽然键已经排序过，但值并不遵循任何顺序。

www.ChinaDBA.net 中国DBA超级论坛

最后，reducer必须执行它的计算，这里是计算一个键上所有值的平均数。代码清单4-8给出了reducer的完整python代码。

代码清单4-8 AverageByAttributeReducer.py

```
#!/usr/bin/env python
import sys
(last_key, sum, count) = (None, 0.0, 0)
for line in sys.stdin:
    (key, val) = line.split("\t")

    if last_key and last_key != key:
        print last_key + "\t" + str(sum / count)
         (sum, count) = (0.0, 0)

    last_key = key
    sum   += float(val)
    count += 1

print last_key + "\t" + str(sum / count)
```

这段程序不断地为每个键做求和与计数操作。当在输入流中检测到新的键或者读到文件结尾时，它计算前一个键的平均值并输出到STDOUT中。运行完整的MapReduce作业后，我们可以很容易地检查前几个结果的正确性。

```
AD      14.0
AE      15.4
AG      13.25
AI      10.0
AM      18.0
AN      9.625
```

注意 对那些感兴趣的人来说，我们获得专利数据的NBER网站上还有一个文件(list_of_countries.txt)，显示了每个国家代码的完整国名。观察我们作业的输出和国家代码，可以看到安道尔（AD）专利平均声明数为 14个。阿拉伯酋长国（AE）专利的平均声明数为15.4个。安提瓜和巴布达（AG）专利的平均声明数为 13.25个，等等。

4.5.4 通过 Aggregate 包使用 Streaming

Hadoop包括一个称为Aggregate的软件包，它让数据集的汇总统计更为简单。尤其在使用Streaming时，该软件包可以简化Java统计程序的编写，这是本节的重点[①]。

在Streaming中Aggregate包作为reducer来做聚集统计。你只需提供一个mapper来处理记录并以特定格式输出。Mapper输出的每行如下：

① 在Java中使用Aggregate包的解释见http://hadoop.apache.org/core/docs/current/api/org/apache/hadoop/mapred/lib/aggregate/package-summary.html。

www.ChinaDBA.net 中国DBA超级论坛

function:*key*\t*value*

字符串输出的开头为一个值聚合函数的名称（从Aggregate包预定义的函数中获得）。紧接着一个冒号和一个以制表符分隔的键/值对。Aggregate的Reducer用函数来处理每个键的一组值。例如，如果函数为LongValueSum，那么输出为每个键对应值的总和。（如函数名所示，每个值被视为Java中的long型）。如果函数为LongValueMax，那么输出为每个键对应值的最大值。你可以在表4-3中看到Aggregator函数的列表。

表4-3　Aggregate包支持的值聚合器函数列表

值聚合器	描　述
DoubleValueSum	一个double值序列的求和
LongValueMax	求一个long值序列的最大值
LongValueMin	求一个long值序列的最小值
LongValueSum	一个long值序列的求和
StringValueMax	求一个string值序列的字母序最大值
StringValueMin	求一个string值序列的字母序最小值
UniqValueCount	（为每个键）求单一值的个数
ValueHistogram	求每个值的个数、最小值、中值、最大值、平均值和标准方差（详见正文）

让我们使用Aggregate包做一个练习，看看它的操作是多么简单。我们希望计算每年授权的专利数。处理这个问题，我们可以采用与第1章单词计数同样的方法。对于每个记录，我们的mapper将授权年度输出为键，并设值为“1”。reducer会将对所有值（“1”）求和后得到一个计数值。只是这里我们基于Aggregate包的Streaming来实现。如代码清单4-9所示，它是一个很简单的mapper。

代码清单4-9　AttributeCount.py

```
#!/usr/bin/env python

import sys

index = int(sys.argv[1])
for line in sys.stdin:
    fields = line.split(",")
    print "LongValueSum:" + fields[index] + "\t" + "1"
```

AttributeCount.py可处理任何CSV格式的输入文件。用户仅需指定列的索引来统计该列上每个属性的记录个数。打印语句包含了这个小程序的主要“动作”。它告知Aggregate包根据用户在指定列（index）中的定义对每个键的所有值（为1）求和。为了计算每年授权的专利个数，我们用Aggregate包来运行这个Streaming程序，告知mapper使用输入文件的第2列（Index=1）作为属性。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➥ -input input/apat63_99.txt
        ➥ -output output
        ➥ -file AttributeCount.py
        ➥ -mapper 'AttributeCount.py 1'
        ➥ -reducer aggregate
```

www.ChinaDBA.net 中国DBA超级论坛

你会发现运行Streaming程序使用的大多数选项都很常见。关键是要指定reducer为'aggregate'。这是一个信号，告诉Streaming API我们正在使用Aggregate包。MapReduce作业的输出（排序后）为：

```
"GYEAR"  1
1963     45679
1964     47375
1965     62857
...
1996     109645
1997     111983
1998     147519
1999     153486
```

第一行与众不同是因为输入数据的第一行是列的描述。接下来MapReduce作业输出每年度的专利个数。如图4-3所示，我们可以对数据进行制图以便有更好的可视效果。你会看到它大体呈稳定上升的趋势。

观察表4-3中Aggregate包的函数列表，你会发现它们大多是原子数据类型与求最大值、求最小值或求和操作的组合。（由于某些原因，DoubleValueMax和DoubleValueMin未被支持。它们相比LongValueMax和LongValueMin只有细微区别和改进。）略有不同的是UniqValueCount和ValueHistogram，我们通过例子来看一下它们的使用方法。

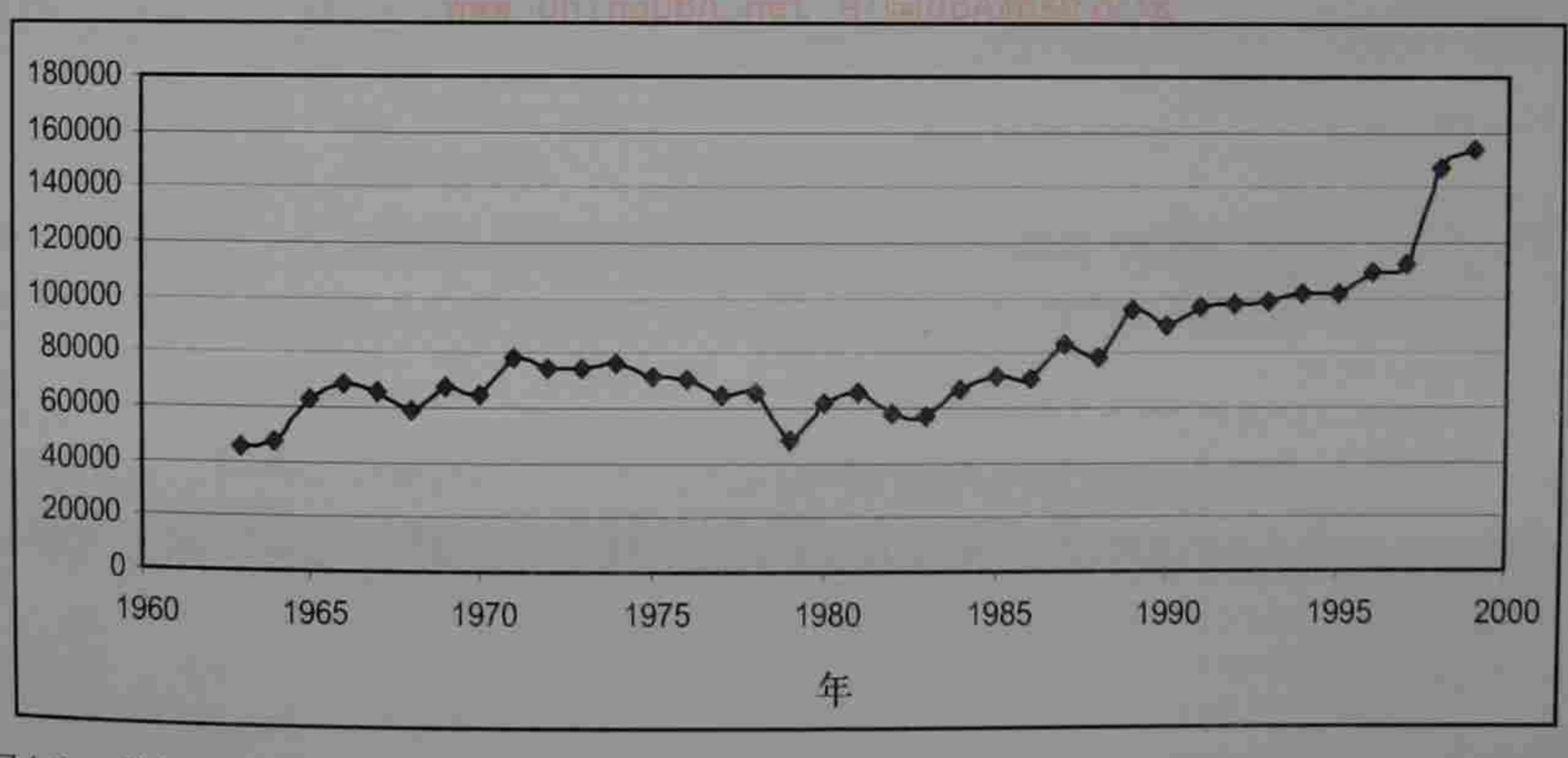


图4-3 使用Hadoop来计算每年专利的发表个数，并用Excel来对结果图形化。这个使用Hadoop的分析很快显示出每年的专利产出在40年间几乎增长到4倍

UniqValueCount给出了每个键对应的唯一值的个数。例如，我们也许希望知道随着时间推移，是否有更多的国家参与到美国的专利系统中。我们可以通过观察每年授权专利的国家个数来获知。我们在代码清单4-10中直接对UniqValueCount做了一个封装，并用它处理apat63_99.txt的包含年度和国家的列（分别为列1和列4）。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
    ➥ -input input/apat63_99.txt
    ➥ -output output
    ➥ -file UniqueCount.rb
```

www.ChinaDBA.net 中国DBA超级论坛

```
➥ -mapper 'UniqueCount.py 1 4'
➥ -reducer aggregate
```

输出结果中每年都有一个记录。画成图形后如图4-4所示。我们可以看到从1960年～1990年增加的专利数（如图4-3所示）并不是来自于更多的国家（如图4-4所示）。同样多的国家贡献了更多的专利。

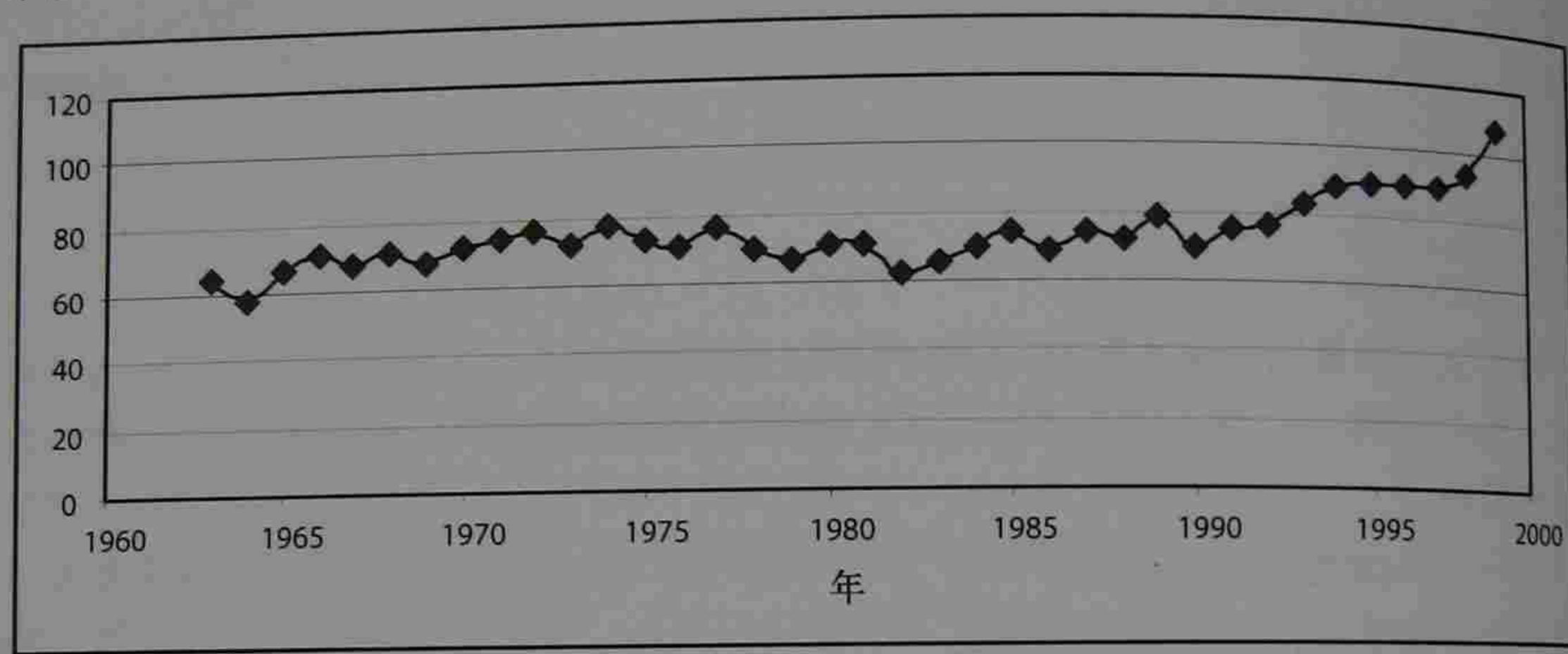


图4-4　每年美国授权专利的国家个数，我们执行一个MapReduce作业并用Excel画出结果图

代码清单4-10　UniqueCount.py：对UniqValueCount函数的一个封装

```
#!/usr/bin/env python

import sys

index1 = int(sys.argv[1])
index2 = int(sys.argv[2])
for line in sys.stdin:
    fields = line.split(",")
    print "UniqValueCount:" + fields[index1] + "\t" + fields[index2]
```

ValueHistogram是Aggregate包中最全能的函数。对于每个键，它输出如下内容：

1　唯一值个数
2　最小个数
3　中值个数
4　最大个数
5　平均个数
6　标准方差

mapper最常见的输出格式为：

```
ValueHistogram:key\tvalue\tcount
```

我们指定ValueHistogram函数后面跟着一个冒号，然后是用制表符分隔键、值和个数的一个三元组。Aggregate的reducer为每个键输出上面的6个统计。注意对于第1个（唯一值个数）以外的所有统计，统计个数是对每个键/值对的数据累计。统计mapper输出的如下两个记录：

www.ChinaDBA.net 中国DBA超级论坛

```
ValueHistogram:key_a\tvalue_a\t10
ValueHistogram:key_a\tvalue_a\t20
```

与统计它们求和之后输出的单个记录相比，结果是一样的

```
ValueHistogram:key_a\tvalue_a\t30
```

一个对mapper有益的变化是仅输出键和值，而不带有个数及其制表符。这里ValueHistogram自动假定个数为1。代码清单4-11给出了ValueHistogram的一个简单封装。

代码清单4-11 ValueHistogram.py：对Aggregate包中ValueHistogram的封装

```
#!/usr/bin/env python

import sys

index1 = int(sys.argv[1])
index2 = int(sys.argv[2])
for line in sys.stdin:
    fields = line.split(",")
    print "ValueHistogram:" + fields[index1] + "\t" + fields[index2]
```

我们运行这个程序来得到每年授权专利的国家分布情况。

```
bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar
        ➥ -input input/apat63_99.txt
        ➥ -output output
        ➥ -file ValueHist.py
        ➥ -mapper 'ValueHist.py 1 4'
        ➥ -reducer aggregate
```

输出为一个包含7个列的制表符间隔值格式（tab-separated value, TSV）文件。第一列的专利授权年份是键。其他6列为ValueHistogram计算得到的6种统计。输出的部分结果如下（由于格式原因，我们跳过了前两行）：

```
1964    58    1    7    38410    816.8103448275862    4997.413601595352
1965    67    1    5    50331    938.1641791044776    6104.779230296307
1966    71    1    5    54634    963.4507042253521    6443.625995189338
1967    68    1    8    51274    965.4705882352941    6177.445623039149
1968    71    1    7    45781    832.4507042253521    5401.229955880634
1969    68    1    8    50394    993.5147058823529    6080.713518728092
1970    72    1    7    47073    894.8472222222222    5527.883233761672
1971    74    1    9    55976    1058.337837837838    6492.837390992137
```

在年度之后第一列的值为唯一值的个数。它与UniqValueCount的输出结果完全一致。第2、3和4列分别为最小值、中值和最大值。对于我们使用的专利数据集，我们看到（每年）接收授权专利最少的国家（除0之外）接收的专利数为1。特别注意1964年的输出，接收专利最多的国家接收了38410项专利，而一半的国家接收的专利少于7项。一个国家接收专利的平均数在1964年为816.8项，标准方差为4997.4。不用说，鉴于中值（7）和平均值（816.8）的落差，每个国家授权的专利数是高度倾斜的。

我们已经看到在Streaming下使用Aggregate包来做一些常见的评测是如此简单。这是对Hadoop能够简化大数据集分析的一个很好证明。

www.ChinaDBA.net 中国DBA超级论坛

4.6 使用 combiner 提升性能

我们在AverageByAttributeMapper.py和AverageByAttributeReducer.py（代码清单4-7和代码清单4-8）中了解了如何计算每个属性的平均值。mapper读取每个记录，并用键/值对的形式输出该记录的属性与计数。在网络上洗牌键/值对之后，reducer计算每个键的平均值。在计算出每个国家专利声明个数平均值的例子中，我们可以看到至少有两个效率上的瓶颈。

(1) 如果我们有10亿条输入记录，mapper就会生成10亿个键/值对在网络上洗牌。如果我们正在计算求最大值这样的一个函数，显然mapper仅需输出它所见到的键中最大的一个。如此便能减少网络流量并提升性能。对于求平均值这样的函数，事情会更复杂一些，但我们可以重新定义算法，使得在每个mapper上，每个键仅有一个记录会参与洗牌。

(2) 使用专利数据集中的国家作为键，来观察数据倾斜的问题。数据远非均匀分布，绝大多数的记录的键都是美国。不仅输入中的每一个键/值对都被映射到中间数据的一个键/值对，而且大多数中间的键/值对最终会进入单一的reducer中，使其不堪重负。

Hadoop通过扩展MapReduce框架，在mapper和reducer之间增加了一个combiner解决了这些瓶颈。你可以将combiner视为reducer的助手。它致力于减少mapper的输出以降低网络和reducer上的压力。如果我们指定了一个combiner，mapReduce框架使用它的次数可以是0次，1次或者是多次。为了使combiner工作，它在数据的转换上必须与reducer等价。如果我们去掉combiner，reducer的输出也应保持不变。而且，当combiner被应用于中间数据的任意子集时，仍需保持等价的转换特性。

如果Reducer仅仅处理分配型函数，如最大值、最小值和求和（计数），则我们可以使用reducer自身作为combiner。但是许多有用的函数并不是分配型的。我们可以重写其中的一些函数，如求平均值，从而利用combiner。

AverageByAttributeMapper.py采用的求均值方法仅输出每个键/值对。AverageByAttributeReducer.py计算它接收到的键/值对的个数，并将它们的值相加，再将两者相除得到一个最终的结果。使用combiner的主要障碍是计数操作，因为reducer假设它接收到的键/值对的个数就是输入数据的键/值对个数。我们可以重构MapReduce程序来准确地跟踪这个计数操作。combiner就变为一个具有分配律特征的简单求和函数。

因为MapReduce作业需要在缺少combiner的情况下保证正确性，在写combiner之前，让我们首先重构mapper和reducer。由于combiner必须为一个Java类，我们用Java写一个新的求均值程序。

注意 Streaming API允许你使用-combiner选项来指定combiner。至少到0.20版本，cominber仍然必须为一个Java类。最好用Java语言来编写你的mapper和reducer。幸运的是，支持本地的Streaming脚本作为combiner已经在Hadoop的路线图中。实践中，可以通过设置mapper到一个Unix管道'mapper.py | sort | combiner.py'中，来获得一个等效的combiner。而且，如果你正在使用Aggregate包，每个值聚合器中已经包含了一个内置的（Java）combiner。Aggregate包会自动使用这些combiner。

让我们写一个Java的mapper（代码清单4-12），它类似于代码清单4-7中的AverageByAttributeMapper.py。

www.ChinaDBA.net 中国DBA超级论坛

代码清单4-12 与AverageByAttributeMapper.py等效的Java代码

```
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text> {

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, Text> output,
                    Reporter reporter) throws IOException {

        String fields[] = value.toString().split(",", -20);
        String country = fields[4];
        String numClaims = fields[8];
        if (numClaims.length() > 0 && !numClaims.startsWith("\"")) {
            output.collect(new Text(country),
                           new Text(numClaims + ",1"));      ❶ 包含1的计数输出
        }
    }
}
```

这个新的Java版mapper根本不同在于，现在输出附带了对1的计数❶。虽然我们本来可以定义一个新的数据类型Writable来包含值与计数，但这里用的方法更简单，我们仅需在Text中维护一个用逗号分隔的字符串。

在reducer中，解析每个key所对应的一列值。于是通过求和得到总和与计数，并最终相除来得到平均值。

```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, Text, Text, DoubleWritable> {

    public void reduce(Text key, Iterator<Text> values,
                       OutputCollector<Text, DoubleWritable> output,
                       Reporter reporter) throws IOException {
        double sum = 0;
        int count = 0;
        while (values.hasNext()) {
            String fields[] = values.next().toString().split(",");
            sum += Double.parseDouble(fields[0]);
            count += Integer.parseInt(fields[1]);
        }
        output.collect(key, new DoubleWritable(sum/count));
    }
}
```

重构后MapReduce作业的逻辑并不太难理解，对吗？我们为每个键/值对增加一个明确的计数。这个重构允许中间数据在被发送到网上之前在每个mapper上进行合并。

从编程角度看，combiner必须实现Reducer接口。在cominber中的reduce()方法实现了合并操作。这看似是一个糟糕的命名方案，但是回想一下，对分配型函数中的重要类，combiner和reducer执行了相同的操作。因此，combiner采取reducer的签名可以让重用变得简单。你不必重新命名Reduce类就能让它成为combiner类。此外，因为combiner执行了等价的转换，在输出中的键/值对类型必然与其输入相匹配。最后，我们创建了一个Combine类，它看起来近似于Reduce类，除了它最终仅输出（部分）求和和计数，而reducer会计算最终的平均值。

```
public static class Combine extends MapReduceBase
```

www.ChinaDBA.net 中国DBA超级论坛

```
implements Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterator<Text> values,
                   OutputCollector<Text, Text> output,
                   Reporter reporter) throws IOException {
    double sum = 0;
    int count = 0;
    while (values.hasNext()) {
        String fields[] = values.next().toString().split(",");
        sum += Double.parseDouble(fields[0]);
        count += Integer.parseInt(fields[1]);
    }
    output.collect(key, new Text(sum + "," + count));
  }
}
```

为了使combiner可行，driver必须为Jobconf对象指定combiner的类。你可以使用setCombinerClass()方法来实现。driver设置mapper、combiner和reducer：

```
job.setMapperClass(MapClass.class);
job.setCombinerClass(Combine.class);
job.setReducerClass(Reduce.class);
```

Combiner未必会提高性能。你应该监控作业的行为来判断是否由combiner输出的记录数明显地少于输入记录的数量。这种减少必须能够证明花费额外的时间运行combiner是值得的。我们会在第6章讲到，你可以通过JobTracker的Web用户接口方便地进行检查。

观察图4-5，注意在map阶段，combine有1 984 625个输入记录，并且仅有1063个输出记录。显然combiner降低了中间数据的数量。注意reduce侧也执行了combiner，不过在这个例子中它的作用微不足道。

	Counter	Map	Reduce	Total
Job Counters	Data-local map tasks	0	0	4
	Launched reduce tasks	0	0	2
	Launched map tasks	0	0	4
Map-Reduce Framework	Reduce input records	0	151	151
	Map output records	1 984 055	0	1 984 055
	Map output bytes	18 862 764	0	18 862 764
	Combine output records	1 063	151	1 214
	Map input records	2 923 923	0	2 923 923
	Reduce input groups	0	151	151
	Combine input records	1 984 625	493	1 985 118
	Map input bytes	236 903 179	0	236 903 179
	Reduce output records	0	151	151
File Systems	HDFS bytes written	0	2 658	2 658
	Local bytes written	20 554	2 510	23 064
	HDFS bytes read	236 915 470	0	236 915 470
	Local bytes read	21 112	2 510	23 622

图4-5 监控AveragingWithCombiner作业中combiner的效果

www.ChinaDBA.net 中国DBA超级论坛

4.7 温故知新

正所谓熟能生巧。如下的练习可以帮助你领悟如何用MapReduce范式来思考。

(1) Top K记录——改变AttributeMax.py（或AttributeMax.php）来输出完整记录，而不仅为最大值的记录。重写程序使得MapReduce作业输出排名前K个的记录，而不仅为最大值的记录。

(2) 网络流量测量——获取一个web服务器的日志文件，并用Aggregate软件包写一个Streaming程序来计算该站点每个小时的流量。

(3) 两个稀疏向量内乘——一个向量是一列值。给定两个向量，X = [x1, x2, …]和Y = [y1, y2, …]，它们的内乘为Z = x1 * y1 + x2 * y2 + …，当X和Y为long型，但许多值为0时，它们通常表现为稀疏的形式

1, 0.46
9, 0.21
17, 0.92
⋮

这里键（第一列）是向量的索引。所有未被明确指定的元素值均被视为0。注意无需对键进行排序。事实上，键甚至可能不是数字。（对于自然语言处理，键为文档中的词，而内乘为文档相似性的评估。）写一个Streaming作业来计算两个稀疏向量的内部乘积。你可以在MapReducer作业之后增加一个后处理的步骤来完成计算。

(4) 时序处理——考虑时序数据，每个记录以时间戳为键，以量测量（该时间段内）为值。我们想要的输出结果的形式是一个时间序列的线性函数，表示为y(t) = a0 * x(t) + a1 * x(t−1) + a2 * x(t−2) + …+ aN * x(t−N)。这里t代表时间而a0,…,aN为已知的常量。在信号处理中，它被称为FIR滤波。移动平均线（moving average）是一个非常通用的实例，其中a0 = a1 = … = aN = 1/N。在y中的每个点都是x中前N个点的平均值。这是平滑时间序列的一个简单方法。

用MapReduce实现这个线性滤波器。务必使用combiner。如果你将时间序列数据按时间排序（它们通常如此）并且N相对较小，当combiner被使用时，洗牌时减少了哪些网络流量？还有额外的加分项，即写出你自己的partitioner让输出保持为时间序。

对于更高级的开发人员，这个例子说明了可扩展性和性能之间的差别。在Hadoop中实现一个FIR滤波器让它可以扩展到TB级或更多数据。信号处理专业的学生会明白一个FIR滤波器的高性能实现需要快速傅里叶变换(FFT)技术。一个可扩展和高性能的解决方案需要一个FFT的MapReduce实现，这超出了本书的范围。

(5) 交换律——回顾基础数学中交换律的概念，它是指操作的顺序是不相关的。例如，加法遵循交换律，如a+b=b+a以及a+b+c=b+a+c=b+c+a=c+a+b=c+b+a。在功能上，MapReduce框架是为实现交换律函数所设计的吗？是或不是，为什么？

(6) 乘法（乘积）——许多机器学习和统计分类算法包含了大量概率值的相乘。通常我们将一个概率集的乘积与另一个不同概率集的乘积做比较，来选择与较大乘积相对应的分类。我们已经看到最大值为一个分配型函数。那么乘积是否也是分配型呢？写一个MapReduce程序将一个数据集的所有值相乘。要得满分的话，将这个程序应用在一个足够大的数据集上。实现这个

www.ChinaDBA.net 中国DBA超级论坛

MapReduce程序解决了所有的扩展性问题吗？你应该如何来修正它？（自己写浮点库是一个常见的解决方法，但并不是一个好的办法。）

(7) *向虚构方言的翻译*——在计算机科学课程的引论中，一个流行的作业是编写程序将英语转换为“黑话”。这个练习存在许多变种，用于生成其他半虚构的方言，如“Snoop Dogg”和“E-40”。通常的做法包括：可准确匹配单词的字典查询（“for”变为“fo”，“sure”变为“sho”，“the”变为“da”，等），简单的文本规则（以“ing”结尾的单词现在以“in”结尾，替换单词最后一个元音及其之后的所有字母为“izzle”等），以及随机的插入（“kno’ wha’ im sayin’?”）。写这个翻译程序并使用Hadoop将之应用于一个大的资料库，如Wikipedia。

4.8　小结

MapReduce程序遵循一个模板。通常整个程序定义在一个Java类中。在这个类中，driver创建一个MapReduce作业的配置对象作为作业创建和运行的蓝本。你可以在`Mapper`和`Reducer`的子类中分别找到map和reduce的函数。这些类通常不过几十行，因此为了方便起见，它们通常被写为内部类。

Hadoop提供了一个Streaming API，支持用非Java语言来编写MapReduce程序。使用Streaming API，许多MapReduce程序可以更容易地通过脚本语言来开发，尤其是即时数据分析的程序。当使用Streaming时，Aggregate软件包能让你很快地编写出用于计数和基础统计的程序。

MapReduce程序主要是关于map和reduce函数的，但是Hadoop允许在reduce阶段之前，用combiner函数对mapper产生的中间结果进行“预归约”以提升性能。

在标准的编程中（MapReduce范式之外），对数据的计数、求和、平均等操作通常是简单的一次性运算。正如我们在本章中所做的那样，在MapReduce中重构这些程序在概念上相对简单直观。更为复杂的数据分析算法则需要更深层的算法重构，我们将在下一章讨论这部分内容。

4.9　更多资源

虽然本章我们专注于专利数据集，但另外还有一些大型的可供下载和使用的开放数据集。下面是几个例子。

- http://www.netflixprize.com/index——Netflix是一个在线电影租赁网站。其核心业务是基于用户对以前电影的评分为用户推荐新的电影。作为一种竞争策略，它发布了一个用户评分数据集来鼓励用户开发更优的推荐算法。未压缩的数据有2 GB多，包含48万个用户对1.7万个电影的1亿多个电影评分。
- http://aws.amazon.com/publicdatasets/——Amazon为EC2的用户提供了几个免费开放的大型数据集。本书撰写时，数据集包含生物、化学和经济三类。例如，其中一个生物数据集为大约550 GB的人类基因组数据。还有经济学的数据集，如2000年的美国人口普查数据（大约200 GB）。
- http://boston.lti.cs.cmu.edu/Data/clueweb09/——卡内基–梅隆大学的语言技术研究所发布了ClueWeb09数据集来辅助大规模的web研究。它抓取了包含10种语言的10亿个网页。未压缩的数据集达到25 TB。鉴于数据集的规模，获取数据最有效的方法是用硬盘来传递压缩数据（大约为5 TB）。（有一定数据规模时，通过FedEx来传递可以获得高“带宽”。）本书撰写时，该大学寄送4个1.5 TB硬盘的压缩数据的费用为790美元。

www.ChinaDBA.net 中国DBA超级论坛

第5章 高阶MapReduce

本章内容

- 链接多个MapReduce作业
- 执行多个数据集的联结
- 生成Bloom filter

当数据处理变得更加复杂时，你就需要深入了解Hadoop的不同功能。本章将重点讨论一些更高阶的技术。

在高阶数据处理中，你经常会发现无法把将整个流程写在单个MapReduce作业中。因此Hadoop支持将多个MapReduce程序链接成更大的作业。你还会发现，高阶数据处理涉及多个数据集。我们将探讨Hadoop中用于同时处理多个数据集的各种联结技术。当一次处理一组记录时，我们就可以写出一些更高效的数据处理任务的代码。我们已经看到Streaming天生具有一次处理整个分片的能力，最大值函数的Streaming实现充分运用了这种能力。我们将要了解的是Java程序也可以做同样的事情。我们还会剖析Bloom filter，并把它的实现放在一个保持着跨记录信息的mapper中。

5.1 链接 MapReduce 作业

你一直在做的数据处理任务都可由单个MapReduce作业完成。当能够更熟练地编写MapReduce程序，并处理更费时费力的数据处理任务时，你会发现许多复杂的任务需要分解成一些简单的子任务，每个均需通过单独的MapReduce作业来完成。例如，你可能想从引用数据集中找到10个被引用最多的专利。这种操作可以通过由两个MapReduce作业组成的一个序列来完成。第一个创建“倒排”引用数据集，并计算每个专利的引用数，而第二个作业再去寻找这个“倒排”数据中最大的10个。

5.1.1 顺序链接 MapReduce 作业

虽然上述两个作业可以手动地逐个执行，但更为便捷的方式是生成一个自动化的执行序列。你可以将MapReduce作业按照顺序链接在一起，用一个MapReduce作业的输出作为下一个的输入。MapReduce作业的链接类似于Unix的管道。

www.ChinaDBA.net 中国DBA超级论坛

```
mapreduce-1 | mapreduce-2 | mapreduce-3 | ...
```

顺序地链接MapReduce作业是非常简单的。回顾一下，driver为MapReduce作业创建一个带有配置参数的JobConf对象，并将该对象传递给JobClient.runJob()来启动这个作业。当JobClient.runJob()运行到作业结尾处被阻止时，MapReduce作业的链接会在一个MapReduce作业之后调用另一个作业的driver。每个作业的driver都必须创建一个新的JobConf对象，并将其输入路径设置为前一个作业的输出路径。你可以在最后阶段删除在链上每个阶段生成的中间数据。

5.1.2 具有复杂依赖的MapReduce链接

有时，在复杂数据处理任务中的子任务并不是按顺序运行的，因此它们的MapReduce作业不能按线性方式链接。例如，mapreduce1处理一个数据集，mapreduce2独立处理另一个数据集，而第3个作业mapreduce3，对前两个作业的输出结果做内部联结。（在下一节中我们将讨论数据联结。）mapreduce3依赖于其他两个作业，仅当mapreduce1和mapreduce2都完成后才可以执行。而mapreduce1和mapreduce2之间并无相互依赖。

Hadoop有一种简化机制，通过Job和JobControl类来管理这种（非线性）作业之间的依赖。Job对象是MapReduce 作业的表现形式。Job对象的实例化可通过传递一个JobConf对象到作业的构造函数中来实现。除了要保持作业的配置信息外，Job还通过设定addDependingJob()方法维护作业的依赖关系。对于Job对象x和y，

www.ChinaDBA.net 中国DBA超级论坛

```
x.addDependingJob(y)
```

意味着x在y完成之前不会启动。鉴于Job对象存储着配置和依赖信息，JobControl对象会负责管理并监视作业的执行。通过addJob()方法，你可以为JobControl对象添加作业。当所有作业和依赖关系添加完成后，调用JobControl的run()方法，生成一个线程来提交作业并监视其执行。JobControl有诸如allFinished()和getFailedJobs()这样的方法来跟踪批处理中各个作业的执行。

5.1.3 预处理和后处理阶段的链接

大量的数据处理任务涉及对记录的预处理和后处理。例如，在处理信息检索的文档时，可能一步是移除stop words（像a、the和is这样经常出现但不太有意义的词），另一步做stemming（转换一个词的不同形式为相同的形式，例如转换finishing和finished为finish）。你可以为预处理与后处理步骤各自编写一个MapReduce作业，并把它们链接起来。在这些步骤中可以使用IdentityReducer（或完全不用reducer）。由于过程中每一个步骤的中间结果都需要占用I/O和存储资源，这种做法是低效的。另一种方法是自己写mapper去预先调用所有的预处理步骤，再让reducer调用所有的后处理步骤。这将强制你采用模块化和可组合的方式来构建预处理和后处理。Hadoop在版本0.19.0中引入了ChainMapper和ChainReducer类来简化预处理和后处理的构成。

如5.1.1节所述，你可以通过伪正则表达式将MapReduce作业的链接符号化地表达为：

```
[MAP | REDUCE]+
```

www.ChinaDBA.net 中国DBA超级论坛

这里REDUCE为reducer，位于名为MAP的mapper之后。这个[MAP | REDUCE]序列可以重复一次或多次，一个跟着一个。使用ChainMapper和ChainReducer所生成的作业表达式与此类似，为：

```
MAP+ | REDUCE | MAP*
```

作业按序执行多个mapper来预处理数据，并在运行reduce之后可选地按序执行多个mapper来做数据的后处理。这一机制的优点在于你可以将预处理和后处理步骤写为标准的mapper。如果你愿意，你可以逐个运行它们。（这有助于分别做调试。）你可以在ChainMapper和ChainReducer中调用addMapper()方法来分别组合预处理和后处理的步骤。全部预处理和后处理步骤在单一的作业中运行，不会生成中间文件，这大大减少了I/O操作。

考虑这样一个例子，有4个mapper（Map1、Map2、Map3，和 Map4）和一个reducer（Reduce），它们被链接为单个MapReduce作业，顺序如下：

```
Map1 | Map2 | Reduce | Map3 | Map4
```

在这个组合中，你可以把Map2和Reduce视为MapReduce作业的核心，在mapper和reducer之间使用标准的分区和洗牌。可以把Map1视为前处理步骤，而将Map3和Map4作为后处理步骤。处理步骤的数目可以变化。这里仅为一个示例。

你可以使用driver设定这个mapper和reducer序列的构成。见代码清单5-1。你需要确保的是，一个任务输出的键和值类型能够匹配的下一个任务的输入类型（类）。

代码清单5-1 用于链接MapReduce作业中mapper的driver

```
Configuration conf = getConf();
JobConf job = new JobConf(conf);

job.setJobName("ChainJob");
job.setInputFormat(TextInputFormat.class);
job.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(job, in);
FileOutputFormat.setOutputPath(job, out);

JobConf map1Conf = new JobConf(false);
ChainMapper.addMapper(job,
                      Map1.class,
                      LongWritable.class,
                      Text.class,
                      Text.class,
                      Text.class,
                      true,
                      map1Conf);

JobConf map2Conf = new JobConf(false);
ChainMapper.addMapper(job,
                      Map2.class,
                      Text.class,
                      Text.class,
                      LongWritable.class,
                      Text.class,
                      true,
                      map2Conf);
```

在作业中添加Map1阶段

在作业中添加Map2阶段

www.ChinaDBA.net 中国DBA超级论坛

```
JobConf reduceConf = new JobConf(false);
ChainReducer.setReducer(job,
                        Reduce.class,
                        LongWritable.class,
                        Text.class,
                        Text.class,
                        Text.class,
                        true,
                        reduceConf);
```

在作业中添加Reduce阶段

```
JobConf map3Conf = new JobConf(false);
ChainReducer.addMapper(job,
                       Map3.class,
                       Text.class,
                       Text.class,
                       LongWritable.class,
                       Text.class,
                       true,
                       map3Conf);
```

在作业中添加Map3阶段

```
JobConf map4Conf = new JobConf(false);
ChainReducer.addMapper(job,
                       Map4.class,
                       LongWritable.class,
                       Text.class,
                       LongWritable.class,
                       Text.class,
                       true,
                       map4Conf);
JobClient.runJob(job);
```

在作业中添加Map4阶段

www.ChinaDBA.net 中国DBA超级论坛

driver首先会设置“全局”的JobConf对象，包含作业名、输入路径及输出路径等。它一次性地添加这个由5个步骤链接在一起的作业，以步骤执行先后为序。它用ChainMapper.addMapper()添加位于Reduce之前的所有步骤。用静态的ChainReducer.setReducer()方法设置reducer。再用ChainReducer.addMapper()方法添加后续的步骤。全局的JobConf对象(作业)经历所有的5个add*方法。此外，每个mapper和reducer都有一个本地JobConf对象(map1Conf、map2Conf、map3Conf、map4Conf和reduceConf)，其优先级在配置各自mapper/reducer时高于全局的对象。建议本地JobConf对象采用一个新的JobConf对象，且在初始化时不设默认值——new JobConf(false)。

让我们通过ChainMapper.addMapper()方法的签名来详细了解如何一步步地链接作业。ChainReducer.setReducer()的签名和功能与ChainReducer.addMapper()类似，我们不再详述。

```
public static <K1,V1,K2,V2> void
                    addMapper(JobConf job,
                              Class<? extends Mapper<K1,V1,K2,V2>> klass,
                              Class<? extends K1> inputKeyClass,
                              Class<? extends V1> inputValueClass,
                              Class<? extends K2> outputKeyClass,
                              Class<? extends V2> outputValueClass,
```

www.ChinaDBA.net 中国DBA超级论坛

```
boolean byValue,
JobConf mapperConf)
```

该方法有8个参数。第一个和最后一个分别为全局和本地的JobConf对象。第二个参数(Klass)是Mapper类，负责数据处理。余下4个参数inputValueClass、inputKeyClass、outputKeyClass和outputValueClassMapper是这个Mapper类中输入/输出类的类型。

需要稍微解释一下byValue这个参数。在标准的Mapper模型中，键/值对的输出在序列化之后写入磁盘[①]，等待被洗牌到一个可能完全不同的节点上。形式上认为这个过程采用的是值传递(passed by value)，发送的是键/值对的副本。在目前的情况下我们可以将一个Mapper与另一个相链接，在相同的JVM线程中一起执行。因此，键/值对的发送有可能采用引用传递(passed by reference)，初始Mapper的输出放在内存中，后续的Mapper直接引用相同的内存位置。当Map1调用OutputCollector.collect (K k, V v)时，对象k和v直接传递给Map2的map()方法。mapper之间可能有大量的数据需要传递，避免去复制这些数据可以让性能得以提高。但是，这样做会违背Hadoop中MapReduce API的一个更为微妙的“约定”，即对OutputCollector.collect(K k, V v)的调用一定不会改变k和v的内容。Map1调用OutputCollector.collect(K k, V v)之后，可以继续使用对象k和v，并完全相信它们的值会保持不变。但如果我们将这些对象通过引用传递给Map2，接下来Map2可能会改变它们，这就违反了API的约定。如果你确信Map1的map()方法在调用OutputCollector.collect(K k, V v)之后不再使用k和v的内容，或者Map2并不改变k和v在其上的输入值，你可以通过设定byValue为false来获得一定的性能提升。如果你对Mapper的内部代码不太了解，安全起见最好设byValue为true，依旧采用值传递模式，确保Mapper会按预期的方式工作。

5.2 联结不同来源的数据

在数据分析中不可避免地需要从不同的来源提取数据。例如，对于我们所用的专利数据集，也许你想知道某些国家引用的专利是否来自另一个国家。这时必须查看引用数据(cite75_99.txt)以及专利数据中的国家信息(apat63_99.txt)。在数据库领域中，这只是两个表的联结，而大多数数据库都会自动提供对联结的处理。不过Hadoop中数据的联结更为复杂，并有几种可能的方法，需做不同的权衡。

为了更好地诠释Hadoop中的联结，我们用一对极小的数据集。我们采用一个以逗号分隔的Customers文件，其中每个记录有3个域：Customer ID（顾客ID）、Name（名字）和Phone Number（电话号码）。我们在该文件中放入4个记录：

```
1,Stephanie Leung,555-555-5555
2,Edward Kim,123-456-7890
3,Jose Madriz,281-330-8004
4,David Stork,408-555-0000
```

① 键和值实现为Writables使得它们能够被复制和序列化。

www.ChinaDBA.net 中国DBA超级论坛

我们在另一个文件Orders中存储顾客的订单。它采用CSV格式，有4个域：Customer ID（顾客ID）、Order ID（订单ID）、Price（价格）和Purchase Date（购买日期）。

```
3,A,12.95,02-Jun-2008
1,B,88.25,20-May-2008
2,C,32.00,30-Nov-2007
3,D,25.02,22-Jan-2009
```

如我们想对上面两个数据集做内联结，预期的输出应该如代码清单5-2所示。

代码清单5-2 Customers和Orders数据做内联结的预期输出结果

```
1,Stephanie Leung,555-555-5555,B,88.25,20-May-2008
2,Edward Kim,123-456-7890,C,32.00,30-Nov-2007
3,Jose Madriz,281-330-8004,A,12.95,02-Jun-2008
3,Jose Madriz,281-330-8004,D,25.02,22-Jan-2009
```

Hadoop还可以执行外联结，但简单起见我们仅关注内联结。

5.2.1 Reduce侧的联结

Hadoop有一个名为datajoin的contrib软件包，在Hadoop中它是一个用作数据联结的通用框架。它的jar文件位于contrib/datajoin/hadoop-*-datajoin.jar。因为大多数的处理都被放在reduce侧，为了与其他的联结技术相区别，我们称它为reduce侧联结。它也采用与数据库技术中相同的命名，故也被称为repartitioned join（重分区联结），或者repartitioned sort-merge join（重分区排序-合并联结）。虽然并非最有效的联结技术，但它是最通用的，进而成为一些更高阶技术的基础（如Semijoin）。

Reduce侧联结引入了一些新的术语与概念，名为data source（数据源）、tag（标签）和group key（组键）。一个数据源类似关系数据库中的一张表。在极小数据示例中，我们有两个数据源：Customers和Orders。一个数据源可以是单个或多个文件。最重要的一点是，在一个数据源中的所有记录有相同的结构，类似于一个schema。

MapReduce范式每次以无状态地方式处理一个记录。如果我们想保持一些状态信息，就不得不标记记录的状态。例如，对于我们所用的两个文件，mapper看到的记录可能为：

```
3,Jose Madriz,281-330-8004
```

或

```
3,A,12.95,02-Jun-2008
```

这里记录类型（Customers或者Orders）失去了与记录自身的关联。标记这个记录将确保特定的元数据可以一直跟随着记录。为了联结数据，我们希望为每个记录标记数据源。

Group key的作用就像关系数据库中的join key（联结键）。在我们的例子中，组键为Customer ID（顾客ID）。由于datajoin软件包允许Group key可由用户自由定义，Group key比关系数据库中的join key更为通用。

在解释如何使用contrib软件包之前，我们先遍历一下对这个极小数据集做repartitioned sort-merge join（重分区排序-合并联结）的所有主要步骤。在明白这些步骤如何组装在一起之后，我们将看到哪一个步骤是由datajoin软件包所处理的，而哪一个是由我们编程实现的。我们还会

www.ChinaDBA.net 中国DBA超级论坛

通过代码来说明如何利用钩子将我们的代码与datajoin软件包进行集成。

1. Reduce侧联结的数据流

图5-1解释了在极小数据集Customers和Orders上所做的一个重分区联结的数据流，直到reduce阶段前为止。我们将在以后更详细地说明reduce阶段的工作。

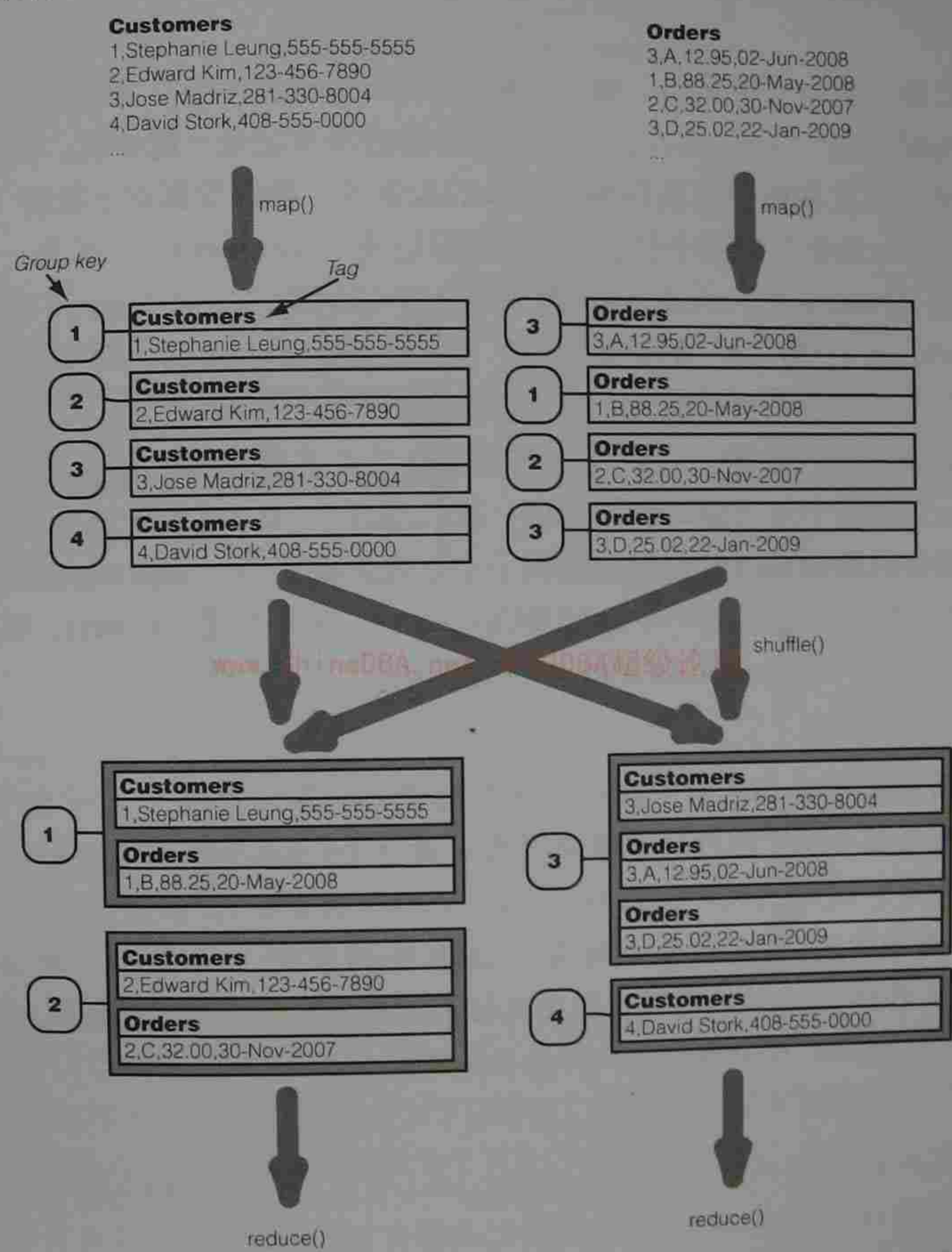


图5-1 在重分区联结中，mapper首先用一个组键和一个标签封装每个记录。组键为连接属性，而标签为记录的数据源（在SQL术语中为table）。在分区和洗牌阶段，相同组键的所有记录被分为一组。最后，在具有相同组键的数据集上调用Reducer

首先，我们看到mapper接收的数据来自两个文件，Customers及Orders。每个mapper都已知它所处理数据流的文件名。调用map()函数处理每个记录，其主要目的是将每个记录打包，使其可以在reduce侧进行联结。

回顾在MapReduce框架中，map()的输出被记录为键/值对，并按键来分区，而且相同键的所有记录最终都会在单一的reducer中被一起处理。对于联结，我们希望map()函数输出一个记录包，

www.ChinaDBA.net 中国DBA超级论坛

采用组键作为联结的键——这里为Customer ID。在这个键/值包中的值为原始记录，并用数据源来标记（即文件名）。例如，对于来自Orders文件的记录，Map()会输出一个键/值对

```
3,A,12.95,02-Jun-2008
```

用于和来自文件Customers的记录进行联结；它的键为Customer ID“3”，值为带有标签“Orders”的完整记录。

在map()封装输入的每个记录后，就执行MapReduce标准的分区、洗牌和排序操作。注意，因为组键被设置为联结键，reduce()会对相同联结键的所有记录一起处理。reduce()函数通过解包得到原始记录，以及根据标签所得到的记录的数据源。我们看到对于组键（Custormer ID）“1”和“2”，reduce()函数得到两个值。一个值被标记为“Customers”，而另一个为“Orders”。对于（组）键“4”的map输出，reduce()仅看到一个标记为“Customers”的值。这与预期一致，因为没有Custumer ID为“4”的记录存在。另一方面，reduce()得到3个（组）键为“3”的值。这是因为有一个记录来自Customers，而另外两个来自Orders。

reduce()函数接收输入数据，并对其值进行完全交叉乘积。reduce()生成这些值的所有合并结果，并限定一个合并中每个值最多被标记一次。当reduce()见到的值分别有不同的标签时，交叉乘积就是这些值的原始集合。在我们的例子中，就是组键为1、2和4的场景。图5-2解释了组键为3的交叉乘积。我们有3个值，一个标记为Customers，两个标记为Orders。交叉结果产生了两个合并。每个合并包含一个Customers值和一个Orders值。

注意 在我们所用的极小数据示例中有一个隐含的模式，即Customer ID标识唯一的Customers记录，这使得在交叉之后的结果数目通常就是这个Customer ID对应的Orders记录数（除非记录数为零，这时交叉的结果则为Customers记录本身）。在更为复杂的情况下，交叉结果所生成的合并数目是每个标签下记录数的乘积。如果reduce()同时看到两个Customers记录和3个Orders记录，则交叉结果就会产生6个（2*3）合并。如果两个记录之外还有第3个数据源（Accounts），则交叉结果就会产生12个（2*2*3）合并。

交叉乘积得到的每个合并结果被送入函数combine()。（不要与4.5节中讲解的combiner混淆。）交叉乘积的本质确保了combine()看到的记录都有相同的联结键，且来自于每个数据源（标签）的记录不超过一个。正是combine()函数决定了整个操作是内联结、外联结，还是其他方式的联结。在内联结中，combine()丢弃所有未含有全部标签的合并结果，如在我们示例中组键为“4”的情形。否则combine()合并来自不同数据源的记录到一个输出记录中。

现在，你看到了为什么我们将联结过程称为重分区排序–合并连接。在原始输入数据源中的记录可以是任意顺序。它们按照正确的组被重新划分给reducer。接着reducer可以将相同联结键的记录合并到一起，生成所期望的联结输出结果。（这里存在排序，但对理解这个操作并不是很关键。）

www.ChinaDBA.net 中国DBA超级论坛

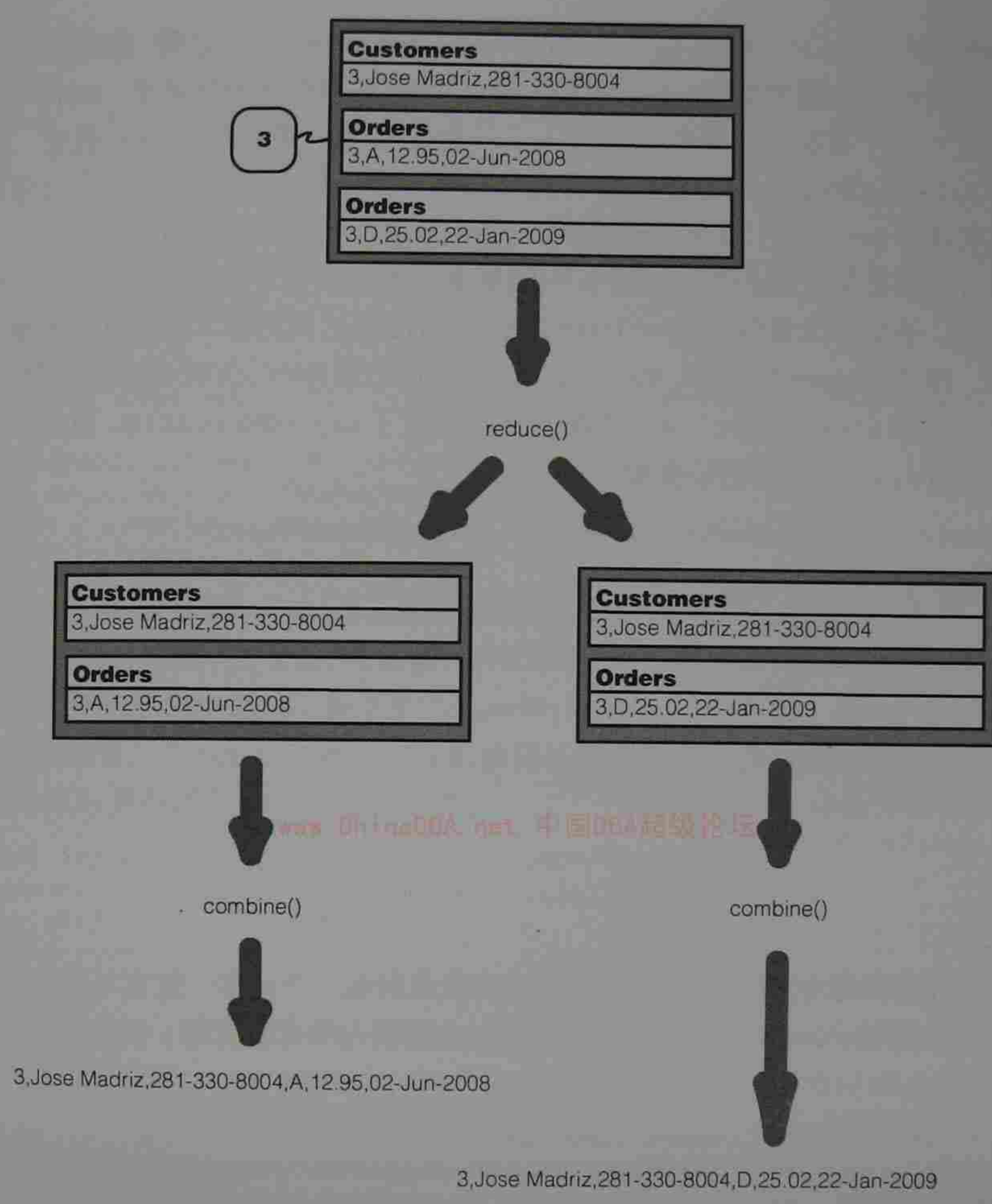


图5-2 重分区联结的reduce侧。给定一个联结键，reduce任务所形成的值来自于不同数据源的完全交叉乘积。它发送每个合并到combine()来生成一个输出记录。对于任意特定的合并，Combine()函数都可以选择不做输出

2. 使用DATAJOIN软件包实现联结

如前所述，Hadoop的datajoin软件包实现了联结的数据流。我们有一些钩子可用于处理特定数据结构中的详细信息，还有一个特殊的钩子可用于定义combine()的确切功能。

Hadoop的datajoin软件包有3个可供继承和具体化的抽象类：DataJoinMapperBase、DataJoinReducerBase和TaggedMapOutput。顾名思义，我们的MapClass会扩展DataJoinMapperBase，而Reduce类会扩展DataJoinReducerBase。Datajoin软件包已经分别在这些基类上实现了map()和reduce()方法，可用于执行上一节描述的联结数据流。我们的子类仅需实现几种用于配置详细信息的新方法。

www.ChinaDBA.net 中国DBA超级论坛

在解释如何使用DataJoinMapperBase和DataJoinReducerBase之前，你需要了解在代码通篇所使用的一个新抽象数据类型TaggedMapOutput。回顾在数据流的描述中，mapper输出的包带有一个（组）键和一个被标签记录的值。datajoin软件包指定（组）键为Text类型，而值（即被标记的记录）为TaggedMapOutput类型。TaggedMapOutput是一种用Text标签封装记录的数据类型。它具体实现了getTag()和setTag(Text tag)方法，并指定抽象方法getData()。我们的子类将实现该方法来处理记录的类型。没有明确要求子类必须包含setData()方法，但我们必须将记录数据传入。子类可以实现这样一个setData()方法来达到对称，或者从构造函数中取得一个记录。此外，作为mapper的输出，TaggedMapOutput必须为Writable类型。因此，我们的子类都必须实现readFields()和write()方法。我们创建了TaggedWritable，这是一个简单的子类，用于处理所有Writable的记录类型。

```
public static class TaggedWritable extends TaggedMapOutput {

    private Writable data;

    public TaggedWritable(Writable data) {
        this.tag = new Text("");
        this.data = data;
    }

    public Writable getData() {
        return data;
    }
    ...
}
```

www.ChinaDBA.net 中国DBA超级论坛

回顾在联结操作的数据流中，mapper的主要功能是封装一个记录，使它与其他有相同联结键的记录都被送到相同的reducer。DataJoinMapperBase 执行所有的封装，但这个类为我们的子类指定了3个可以填充的抽象方法：

```
protected abstract Text generateInputTag(String inputFile);
protected abstract TaggedMapOutput generateTaggedMapOutput(Object value);
protected abstract Text generateGroupKey(TaggedMapOutput aRecord);
```

GenerateInputTag()在map任务开始时被调用，来为这个map任务所处理的所有记录指定一个全局的标签。这个标签定义为Text类型。请注意我们使用记录的文件名来调用generateInputTag()。处理Customers文件的mapper将接收字符串“Customers”作为generateInputTag()的参数。鉴于我们正在使用这个标签来标识数据源，而我们的文件名又是用来反映数据源的，generateInputTag()写为

```
protected Text generateInputTag(String inputFile) {
    return new Text(inputFile);
}
```

如果数据源横跨几个文件（part-0000、part-0001等），你可能不想把完整的文件名作为标签，而用它的一些前缀。例如，标签（数据源）可以为短划线（ - ）前的文件名。

```
protected Text generateInputTag(String inputFile) {
    String datasource = inputFile.split('-')[0];
```

www.ChinaDBA.net 中国DBA超级论坛

```
    return new Text(datasource);
}
```

我们把generateInputTag()的结果存在DataJoinMapperBase对象的变量inputTag中供以后使用。同样，如果想再次提及它，还可以把文件名存入DataJoinMapperBase的变量inputFile中。

完成map任务的初始化后，为每个记录调用DataJoinMapperBase的map()。它再调用我们迟早必须要实现的两个抽象方法。

```
public void map(Object key, Object value,
                OutputCollector output, Reporter reporter) throws IOException
{
    TaggedMapOutput aRecord = generateTaggedMapOutput(value);
    Text groupKey = generateGroupKey(aRecord);
    output.collect(groupKey, aRecord);
}
```

GenerateTaggedMapOutput()方法封装记录的值为TaggedMapOutput 类型。回顾我们正在用的TaggedMapOutput具体实现为TaggedWritable。GenerateTaggedMapOutput()方法可以返回一个带有任何我们所想要的Text标签的TaggedWritable。原则上，在同一文件中，不同的记录可以用不同的标签。在标准的情况下，我们希望标签代表一个数据源，它早先已由generate-InputTag()计算好并存在this.inputTag中。

```
protected TaggedMapOutput generateTaggedMapOutput(Object value) {
    TaggedWritable retv = new TaggedWritable((Text) value);
    retv.setTag(this.inputTag);
    return retv;
}
```

GenerateGroupKey()方法取得被标记的记录（TaggedMapOutput类型），并返回用作联结的组键。就现在的目的而言，我们解开有标签的记录，并取CSV格式的值的第一个域作为联结键。

```
protected Text generateGroupKey(TaggedMapOutput aRecord) {
    String line = ((Text) aRecord.getData()).toString();
    String[] tokens = line.split(",");
    String groupKey = tokens[0];
    return new Text(groupKey);
}
```

在更普遍的实现中，用户能够指定哪些字段为联结键，以及是否记录的分隔符可以为逗号之外的一些字符。

DataJoinMapperBase 是一个简单的类，而mapper代码的大部分在我们的子类中。另外，DataJoinReducerBase是datajoin软件包的核心，它通过执行一个完整的外部联结，简化了我们的编程。我们的reducer子类只是必须实现combine()方法来筛选掉不需要的组合，以获取所需的联结操作（内部联结，左外部联结等）。还是在combine()方法中，我们将合并结果格式化为合适的输出格式。

我们为combine()方法提供一个有相同联结（组）键的标记记录的交叉乘的合并。这听起来似乎很复杂，但回顾图5-1和图5-2中的数据流图，对两个典型的数据源而言，交叉乘是很简单的。

www.ChinaDBA.net 中国DBA超级论坛

每个合并的记录数要么为2个（意味着在每个有组键的数据源中至少有一个记录），要么为1个（意味着仅有一个数据源有该联结键）。

让我们看看combine()的签名：

```
protected abstract TaggedMapOutput
                    combine(Object[] tags, Object[] values);
```

一个合并结果由一个标签数组和一个值的数组来表示。这两个数组的大小必须保证相同，且等于合并结果中被标记记录的个数。合并结果中第1个被标记的记录表示为tags[0]和values[0]，第2个为tags[1] 和values[1]，以此类推。此外，这些标签总是按顺序排列。

由于标签与数据源对应，在两个数据源相联结的典型情况下，combine()的标签数组长度不会超过2。图5-2显示 combine()被调用了两次。左图中，标签和值的数组如下所示：[1]

```
tags = {"Customers", "Orders"};
values = {"3,Jose Madriz,281-330-8004", "A,12.95,02-Jun-2008"};
```

对于内部联结，combine()会忽略那些没有全部标签的合并结果，而返回空值。在一个合法的合并中，combine()的作用是将所有值都连接成一个单个记录来输出。联结的顺序完全是由combine()决定的。在内部联结中，values[]的长度总是等于可用数据源的个数（典型情况下为两个），而且标签总是排好序的。一个合理的选择是循环遍历values[]数组，使其按照数据源名称的默认字母顺序进行排序。

像任何reducer一样，DataJoinReducerBase输出键/值对。对于每个合法的合并，键总是为联结键，而值为combine()的输出。请注意联结键还出现在values[]数组的每个单元中。Combine()在连接它们之前，会去掉这些单元中的联结键。否则联结键会在输出的一条记录中被多次显示。

最后，DataJoinReducerBase希望combine()返回TaggedMapOutput。至于为什么DataJoinReducerBase忽略在TaggedMapOutput对象中的标签，目前尚不清楚原因。

代码清单5-3展示了完整的代码段，包括reduce的子类。

代码清单5-3　来自两个reduce侧连接数据的内联结

```
public class DataJoin extends Configured implements Tool {

    public static class MapClass extends DataJoinMapperBase {

        protected Text generateInputTag(String inputFile) {
            String datasource = inputFile.split("-")[0];
            return new Text(datasource);
        }

        protected Text generateGroupKey(TaggedMapOutput aRecord) {
            String line = ((Text) aRecord.getData()).toString();
            String[] tokens = line.split(",");
            String groupKey = tokens[0];
            return new Text(groupKey);
        }
```

www.ChinaDBA.net 中国DBA超级论坛

```
        protected TaggedMapOutput generateTaggedMapOutput(Object value) {
            TaggedWritable retv = new TaggedWritable((Text) value);
            retv.setTag(this.inputTag);
            return retv;
        }
    }

    public static class Reduce extends DataJoinReducerBase {

        protected TaggedMapOutput combine(Object[] tags, Object[] values){
            if (tags.length < 2) return null;
            String joinedStr = "";
            for (int i=0; i<values.length; i++) {
                if (i > 0) joinedStr += ",";
                TaggedWritable tw = (TaggedWritable) values[i];
                String line = ((Text) tw.getData()).toString();
                String[] tokens = line.split(",", 2);
                joinedStr += tokens[1];
            }
            TaggedWritable retv = new TaggedWritable(new Text(joinedStr));
            retv.setTag((Text) tags[0]);
            return retv;
        }
    }

    public static class TaggedWritable extends TaggedMapOutput {

        private Writable data;

        public TaggedWritable(Writable data) {
            this.tag = new Text("");
            this.data = data;
        }

        public Writable getData() {
            return data;
        }

        public void write(DataOutput out) throws IOException {
            this.tag.write(out);
            this.data.write(out);
        }

        public void readFields(DataInput in) throws IOException {
            this.tag.readFields(in);
            this.data.readFields(in);
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = getConf();

        JobConf job = new JobConf(conf, DataJoin.class);

        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setJobName("DataJoin");
```

www.ChinaDBA.net 中国DBA超级论坛

```
        job.setMapperClass(MapClass.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormat(TextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(TaggedWritable.class);
        job.set("mapred.textoutputformat.separator", ",");
        JobClient.runJob(job);
        return 0;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                                 new DataJoin(),
                                 args);
        System.exit(res);
    }
}
```

下一步，我们将介绍另一种联结方法，它在某些常见应用中更为高效。

5.2.2 基于 DistributedCache 的复制联结

上一节中讨论的reduce侧的联结技术非常灵活，但其效率也可能很低。直到Reduce阶段才会做联结。我们先让所有数据在网络上重排，然后在许多情况下连接过程中的大部分收据又被丢弃了。如果我们在map阶段就去除不必要的数据，连接会更有效率。甚至于在map阶段就执行完整个联结操作，效果会更好。

在map阶段执行联结的主要障碍是一个mapper正在处理的记录可能会与另一个不易访问（甚至定位）的记录进行联结。如果我们可以保证在联结一条记录时可以访问所有需要的数据，在map侧的联结就可以工作。例如，如果我们知道两个数据源被划分到相同数量的分区，而且分区是按键排序的，且这个键又是所需的联结键，则每个mapper（加上适当的`InputFormat`和`RecordReader`）就可以精确地查找和检索到执行联结所需的所有数据。事实上，Hadoop的`org.apache.hadoop.mapred.join`软件包含有`helper`类，可以很容易地实现这个map侧的联结。不过只有有限的场景下我们才能自然地使用它，而且运行额外的MapReduce作业来重新划分数据源使之可以被该软件包所使用，似乎又损害了效率。因此，我们不会进一步讨论这个包。

但并非完全没有希望。有另一种非常频繁出现的数据模式可供利用。当连接大数据时，往往只有一个数据源较大；第二个数据源可能会小几个数量级。例如，本地电话公司的客户数据可能只有几千万记录（每个记录包含一个客户的基本信息），但其事务日志可以有数十亿包含详细的通话历史的记录。当较小的数据源可以装入mapper的内存时，我们可以通过将较小的数据源复制到所有的mapper，并在map阶段执行联结，以实现效率上的极大提高。借用数据库的术语，它被称为*复制联结*（replicated join），即一个数据表被复制到集群中的所有节点上。（下一节将讨论当较小的数据源不适合放在内存中的情况。）

Hadoop有一个称为*分布式缓存*的机制，设计主旨在于将文件分布到集群中所有节点上。它通常用于分发包含所有mapper所需“背景”数据的文件。例如，如果使用Hadoop进行文档分类，你

www.ChinaDBA.net 中国DBA超级论坛

可能让每个类有一个关键词列表。(或者更进一步，让每个类有一个概率模型，不过这跑题了……)你将使用分布式缓存来保证所有mapper都有机会获得关键词的列表，即“背景”数据。为了执行复制联结，我们认为较小的数据源为背景数据。

管理分布式缓存的类有一个恰如其分的名字——DistributedCache。使用这个类要采取两个步骤。第一步，当配置作业时，你可以调用静态方法DistributedCache.addCacheFile()来设定要传播到所有节点上的文件。这些文件被指定为URI对象，除非设置了不同的文件系统，它们都默认地放在HDFS上。JobTracker在开始工作时，将取得URI列表，并在所有TaskTrackers中创建文件的本地副本。第二步，在每个单独TaskTracker上的mapper会调用静态方法DistributedCache.getLocalCacheFiles()来获取数组本地副本所在的本地文件的路径。这里mapper可以使用标准的Java文件I/O技术读取本地副本。

使用DistributedCache的复制联结比reduce侧的联结更简单。让我们从标准的Hadoop模板开始讲述。

```
public class DataJoinDC extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, Text, Text> {

        ...
    }

    public int run(String[] args) throws Exception {
        ...
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                                 new DataJoinDC(),
                                 args);

        System.exit(res);
    }
}
```

请注意，我们已经去掉了Reduce类。我们计划在map阶段执行联结，并将配置该作业为没有reducer。你会发现我们的driver方法也很熟悉。

```
public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    JobConf job = new JobConf(conf, DataJoinDC.class);

    DistributedCache.addCacheFile(new Path(args[0]).toUri(), conf);

    Path in = new Path(args[1]);
    Path out = new Path(args[2]);
    FileInputFormat.setInputPaths(job, in);
    FileOutputFormat.setOutputPath(job, out);

    job.setJobName("DataJoin with DistributedCache");
    job.setMapperClass(MapClass.class);
    job.setNumReduceTasks(0);

    job.setInputFormat(KeyValueTextInputFormat.class);
    job.setOutputFormat(TextOutputFormat.class);
```

www.ChinaDBA.net 中国DBA超级论坛

```
    job.set("key.value.separator.in.input.line", ",");
    JobClient.runJob(job);
    return 0;
}
```

这里关键的增加是我们取一个文件并将其添加到DistributedCache中，该文件由第1个参数所指定。当我们运行该作业时，每个节点将从HDFS创建该文件的本地副本。第2个和第3个参数表示标准Hadoop 作业的输入和输出路径。请注意我们限制了数据源个数为2。这并非技术上的固有局限，而是为了让代码更易于理解而作的一种简化。

到目前为止，我们的MapClass仅需定义一个方法，map()。事实上，Mapper接口（以及Reducer接口）还有另外两个抽象方法，configure()和close()。当我们最初实例化MapClass时，调用configure()方法，而在mapper结束处理其分片时，调用close()方法。MapReduceBase类为这些方法提供默认的no-op实现。在这里我们需要重写configure()，在mapper第一次初始化时，将连接的数据加载到内存中。通过这种方式，我们在每次调用map()处理一条新纪录时，都可以获得这个数据。

```
public static class MapClass extends MapReduceBase
    implements Mapper<Text, Text, Text, Text> {

    private Hashtable<String, String> joinData =
                                    new Hashtable<String, String>();
    @Override
    public void configure(JobConf conf) {
        try {
            Path [] cacheFiles = DistributedCache.getLocalCacheFiles(conf);
            if (cacheFiles != null && cacheFiles.length > 0) {
                String line;
                String[] tokens;

                BufferedReader joinReader = new BufferedReader(
                                new FileReader(cacheFiles[0].toString()));
                try {
                    while ((line = joinReader.readLine()) != null) {
                        tokens = line.split(",", 2);
                        joinData.put(tokens[0], tokens[1]);
                    }
                } finally {
                    joinReader.close();
                }
            }
        } catch (IOException e) {
            System.err.println("Exception reading DistributedCache: " + e);
        }
    }

    public void map(Text key, Text value,
                    OutputCollector<Text, Text> output,
                    Reporter reporter) throws IOException {

        String joinValue = joinData.get(key);
        if (joinValue != null) {
```

www.ChinaDBA.net 中国DBA超级论坛

```
            output.collect(key,
                           new Text(value.toString() + "," + joinValue));
        }
    }
}
```

当调用configure()时，我们得到一个文件路径数组，指向由DistributedCache填入的文件本地副本。因为我们的driver方法仅填入一个文件到DistributedCache中（由第一个参数给出），该数组的长度为1。我们使用标准的Java文件I/O读取该文件。我们这里假定程序的每一行都是一个记录，键/值对以逗号分隔，且键是唯一的并会用于联结。该程序将此源代码文件读入名为joinData的Java散列表（Hashtable），它可以在mapper的整个生命周期中获得。

连接发生在map()方法中，这显而易见，因为其中一个数据源以joinData的形式驻留在内存中。如果在joinData中找不到联结键，我们就丢弃这个记录。否则我们将这个（联结）键与joinData中的值进行匹配，并连接这些值。结果直接输出到HDFS中，这是因为我们不需要任何的reducer去做进一步的处理。

在使用DistributedCache时，还会出现一种并不罕见的情况，即背景数据（在我们的数据连接中较小的数据源）存储于客户端的本地文件系统中，而不是在HDFS中。处理这种情况的一种方法是添加代码，在调用DistributedCache.addCacheFile()之前将客户端上的本地文件上载到HDFS中。幸好该过程可以在GenericOptionsParser中，通过一个通用的Hadoop命令行参数来自然地得到支持。该选项为-files，它自动地复制一组以逗号分隔的文件到所有的任务节点上。命令行语句为

```
bin/hadoop jar -files small_in.txt DataJoinDC.jar big_in.txt output
```

既然我们不再需要自己调用DistributedCache.addCacheFile()，也不必再将较小数据源的文件名作为参数之一。参数的索引也因此移动了。

```
Path in = new Path(args[0]);
Path out = new Path(args[1]);
```

通过这些微调，我们的DistributedCache联结程序就可以让客户端计算机上的本地文件成为一个输入源。

5.2.3 半联结：map 侧过滤后在 reduce 侧联结

使用复制连接的限制之一是其中一个连接表必须小到可以放在内存中。即使输入源中的数据大小通常不对称，但较小的一个仍然可能不够小。要解决这个问题，可以重新组织处理的步骤让处理变得更为有效。例如，如果在寻找区号415中所有客户的订单历史记录的时候，在找出那些区号为415的客户记录之前，先去连接Orders和Customers表，结果虽然是正确的，但效率会很低下。Orders和Customers表对于复制联结来说可能都太大了，你将不得不去做效率低下的reduce侧联结。更好的办法是首先找出区号在415中的顾客。我们将之存储在一个称为 Customers415 的临时文件中。通过联结Orders与Customers415，就可以最终得到相同的结果，而现在Customers415足够小，这使得做一次复制连接就行了。创建和分发Customers415文件的过程会有一些开销，但它一般会由效率上的总增益而得到补偿。

www.ChinaDBA.net 中国DBA超级论坛

有时你可能有大量的数据要做分析。不管如何重新排列处理步骤，你都无法使用复制联结。不过不用担心。仍有办法让Reduce侧的联结更为有效。回顾reduce侧联结的主要问题是数据仅仅由mapper做了标签，然后全部在网络上重排，而它们大多都被reducer所忽略。如果mapper在网络重排以前，用一个额外的预过滤函数去除大多数或甚至所有不必要的数据，效率低的问题就会得到改善。我们需要建立这种筛选机制。

仍用Customers415和Orders的连接示例，联结键为Customer ID，我们希望mapper过滤掉任何不属于区号415的顾客，而不是将这些记录发送到reducer。我们创建一个数据集CustomerID415来存储所有属于区号415的顾客的Customer ID。CustomerID415小于 Customers415，因为它只有一个数据字段。假设CustomerID415现在可以在内存中，我们可以通过使用分布式缓存来把CustomerID415传播到所有的mapper上。当处理来自Customers和Orders的记录时，mapper将丢弃所有键不在CustomerID415中的记录。我们有时采用数据库中的术语，把它称之为半联结。

最后，但同样重要的是，如果CustomerID415这个文件仍然太大而无法放入内存怎么办？或者CustomerID415 也许可以装入内存，但它太大使得将它复制到所有mapper的效率很低。这时需要用到一个名为Bloom filter的数据结构。Bloom filter是一种集合的紧凑表示法，仅支持特定的查询。（如："该集合包含这个元素吗？"）此外，查询答案并不完全准确，但它保证了没有漏报，而且误报仅为小概率事件。数据结构的紧凑性抵消了这些少量的不精确。通过使用Bloom filter表示CustomerID415，mapper将给出区号415的所有顾客。它仍可保证数据联结算法的正确性。Bloom filter传给reduce阶段的顾客还有少数不在区号415中。这没有关系，因为它们会在reduce阶段中被忽略。通过极大地减少在网络上重排所需的通信量，我们仍会获得性能的提升。使用Bloom filter实际上是分布式数据库中在联结中的一个标准技术，并在商业产品如Oracle 11g中得到运用。在下一节中，我们将介绍Bloom filter和其他应用程序的更多详细内容。

5.3 创建一个 Bloom filter

当你把Hadoop用于对大数据集进行批处理时，也许在数据密集型计算中也包含了对事务型处理的需求。我们不会讨论实时分布式数据处理中的所有技术（缓存及分区等）。它们与Hadoop没有必然联系，且超出了本书的范围。在实时数据处理中， Bloom filter是一个相对而言较少为人所知的工具，它是一个数据集的摘要，它的使用让其他的数据处理技术更为有效。当数据集很大时，Hadoop经常被用来生成一个Bloom filter的数据集表示。如前所述，Bloom filter有时还被用作Hadoop内部的数据联结。作为一个数据处理专家，掌握Bloom filter会让你受益匪浅。在本节，我们将更详细地解释这个数据结构，将通过一个在线广告网络的例子来说明使用Hadoop生成Bloom filter的方法。

5.3.1 Bloom filter 做了什么

在最基本的形态中，一个Bloom filter对象支持两个方法：`add()` 和`contains()`。这两个方法的运行方式与Java中`Set`接口是相似的。方法`add()`在集合中增加一个对象，而方法`contains()`返回一个true/false布尔值，它表示一个对象是否在这个集合中。但是，对于一个Bloom filter，`contains()`不会总是给出一个准确的答案。但它不会漏报。如果`contains()`返回false，你可以

www.ChinaDBA.net 中国DBA超级论坛

确定这个集合没有所查询的对象。不过，它会有小概率的误报。Contains()会为一些不在集合中的对象返回true。误报的概率依赖于集合中的元素个数，以及一些Bloom filter自身的配置参数。

Bloom filter的主要优势在于它的大小（比特位个数）为常数且在初始化时被设置。增加更多的元素到一个Bloom filter中不会增加它的大小。它仅增加误报的概率。Bloom filter还有另一个参数，为其所用的散列函数个数。稍后，在讲到Bloom filter的实现时，我们将讨论采用这个参数的原因，以及散列函数是如何被使用的。现在，其主要寓意是它影响误报率。误报率可近似地表示为如下公式

$$(1 - \exp(-kn/m))k$$

这里k是所用散列函数的个数，m是用于存储Bloom filter的比特个数，而n是被添加到Bloom filter的元素个数。实践中，m和n由系统需求决定，因此，在给定m和n的情况下，应通过k来将误报最少化，即为（略作计算可得）

$$k = \ln(2) * (m/n) \approx 0.7 * (m/n)$$

将k代入得到误报率为0.618 5 m/n，这里k必须为一个整数。误报率仅为一个估计值。从设计的角度看，不能仅考虑m，还应该考虑（m/n），即每个元素的比特个数。例如，我们必须存储一个有1千万条URL的集合（n=10 000 000）。每个URL分配8个比特（m/n=8）将需要一个10 MB的Bloom filter（m= 80 000 000比特）。这个Bloom filter的误报率为(0.618 5)8，即大约2%。如果我们通过存储原始的URL来实现这个Set类，并假定平均的URL长度为100字节，我们将不得不占用1 GB的空间。Bloom filter将存储的需求减少了2个数量级，而仅带来2%的误报率！略微增加一些分配给Bloom filter存储空间就可以进一步减少误报率。当每个URL为10个比特时，Bloom filter将占用12.5 MB，而误报概率仅为0.8%。

归纳一下，Bloom filter类的签名如下所示

```
class BloomFilter<E> {
    public BloomFilter(int m, int k) { ... }
    public void add(E obj) { ... }
    public boolean contains(E obj) { ... }
}
```

Bloom filter的更多应用

Bloom filter的第一次成功应用出现在内存还十分稀缺的年代。其中一个用途是拼写检查。因为不能将整个字典存在内存中，拼写检查器使用了一个（字典的）Bloom filter形式来捕获大多数的拼写错。当内存容量增加并变得更廉价时，这种对空间的顾虑就消失了。在当前的大规模数据密集型操作中，由于数据量迅速地超过内存和带宽，Bloom filter又重获新生。

我们已经看到在商业产品中，如Oracal 11g，使用Bloom filter实现跨分布式数据库的数据连接。在网络领域，开源分布式web代理Squid是一个使用Bloom filter的成功案例。Squid会缓存访问频繁的web内容以节省带宽，并给用户更快的web体验。在一个Squid服务器集群中，每个节点缓存了不同数据集合。接收到的请求会被路由到某个Squid服务器上，上面存有一份该请求内容的副本；如果缓存缺失，请求就会被转到原来的服务器。路由机制需要知道每个Squid服务器的内容，因为向每个服务器发送一个URL的列表并在内存中保存的代价太高，就需要使用Bloom filter。一次误报代表一个请求被发送给错误的Squid服务器，但是该服务器可以把它

www.ChinaDBA.net 中国DBA超级论坛

识别成一次缓存缺失，并发送给原来的服务器，可以保证整个操作的正确性。一次误报所造成一些性能损失与整体的性能提升相比微不足道。

数据分片（Sharding）系统是一种与此类似但更为高阶的应用。简单而言，数据库分片是把一个数据库分到多台机器上，使得每台机器仅需处理记录的一个子集。每条记录都拥有某个ID，决定它被分派到哪台机器上。在更基础的设计中，数据库机器的个数是固定的，这个ID会被静态地散列到其中一台机器上。但这个方法不能灵活地添加更多的分片或重组现有的分片。为了提高灵活性，系统用一个动态查找表来管理每个记录ID，但是如果通过一个数据库来实现查找表（即使用磁盘），则会增加处理的延迟。像Squid一样，更多高级的数据分片系统采用内存级的Bloom filter作为快速查找表。虽然需要一些机制来处理误报，但误报率足够小，从而不会影响整体性能的提高。

对于在线显示的广告网络，重要的一点是能够为访问者提供恰当类别来定位广告。对于一个典型的广告网络，当流量规模和延迟需求固定时，为了能够实时地检索目录，人们就需要投入大量资金购买硬件。如果基于Bloom filter来设计，就可以大大减少成本。通过一个离线的处理过程，将web页面（或访问者）标记到有限数量的类别上（体育、以家庭为导向的节目、音乐等）。为每个类建立一个Bloom filter，并把它存在广告服务器的内存中。当广告请求到达时，广告服务器可以迅速并廉价地确定要显示哪一类的广告。误报的数量很小，可以忽略不计。

5.3.2　实现一个 Bloom filter

Bloom filter的实现在概念上是非常直观的。在把它应用到Hadoop的分布式模式之前，我们先在一个单机系统中描述其实现。Bloom filter的内在表现为一个m个比特位的数组。我们有k个独立的散列函数，这里每个散列函数的输入为一个对象，而输出为介于0和$m-1$之间的一个整数。我们使用这个输出的整数作为位数组的索引。当我们“添加”一个元素到Bloom filter时，我们使用散列函数来生成位数组的k个索引。我们设这k个比特为1。图5-3显示了我们在一个使用3个散列函数的Bloom filter中添加了几个对象（x、y和z）的过程。注意无论其以前状态是什么，比特位都被设为1。在位数组中1的个数只能增长。

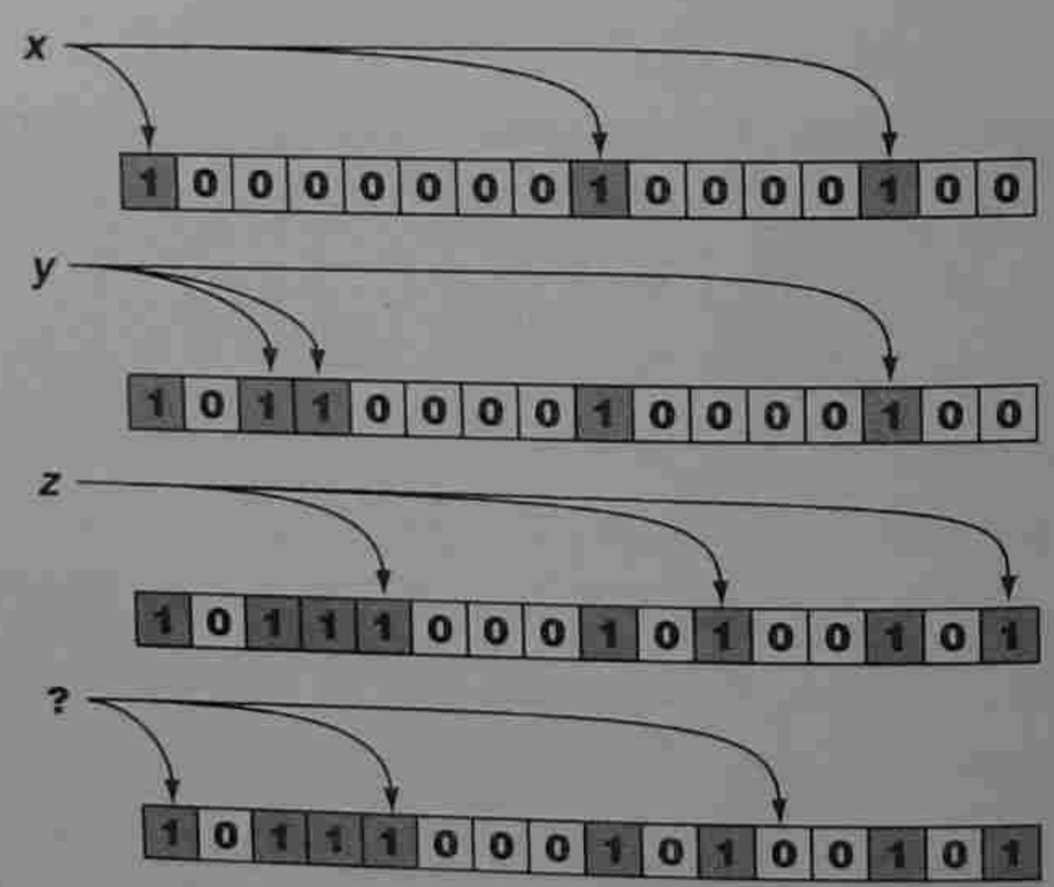


图5-3　Bloom filter是一个比特数组，表示具有一定误报概率的一个集合。对象（如x、y、和z）被确定地散列到数组中的位上，而这些位被设置为1。通过散列并检查那些位置上的比特值，你可以查看一个对象是否在这个集合中

www.ChinaDBA.net 中国DBA超级论坛

当有一个对象到来时，我们如要检查它是否已经被加入到Bloom filter中，则使用在添加对象时相同的k个散列函数来生成一个位数组的索引。现在检查是不是比特数组中的所有的k个比特均为1，如果所有k个比特都是1，我们返回true并声明Bloom filter包含这个对象。否则返回false。我们看到如果对象以前已经被添加，那么Bloom filter一定会返回true。这里没有漏报（当对象的确在集合时却返回false）。不过，即使这个对象从来没有被添加到集合中，与所查询对象相对应的k个比特也可能都为1。这是因为其他对象的增加会设置这些位，从而导致误报。[①]

我们基于Java实现的Bloom filter会使用Java的`BitSet`类作为其内部表示。我们用函数`getHashIndexes(obj)`获取一个对象，并返回一个长度为k的整数数组，它包含`BitSet`的索引。Bloom filter主函数`add()`和`contains()`的功能是非常直观的：

```
class BloomFilter<E> {

    private BitSet bf;

    public void add(E obj) {
        int[] indexes = getHashIndexes(obj);

        for (int index : indexes) {
            bf.set(index);
        }
    }

    public boolean contains(E obj) {
        int[] indexes = getHashIndexes(obj);

        for (int index : indexes) {
            if (bf.get(index) == false) {
                return false;
            }
        }

        return ture;
    }

    protected int[] getHashIndexes(E obj) { ... }
}
```

要真正把`getHashIndexes()`实现为k个彼此独立的散列函数还是比较复杂的。但是，在代码清单5-4中，我们给出了一个粗略的实现方法，来生成大致独立且均匀分布的k个索引。`GetHashIndexes()`方法用对象的MD5散列作为Java随机数发生器的种子，并取k个“随机”数作为索引。更为严谨的`getHashIndexes()`能够优化Bloom filter类的实现，但是我们的粗略实现对于说明而言足够用了。

在两个集合的归并时，有一个巧妙的方法可以生成Bloom filter，即让Bloom filter的两个独立集进行OR（或）操作。由于增加一个对象就是置一个位数组中的某些位为1，很容易就能明白这个归并的规则是正确的：

```
public void union(BloomFilter<E> other) {
```

① Bloom Filters的相关介绍见http://en.wikipedia.org/wiki/Bloom_filter。

www.ChinaDBA.net 中国DBA超级论坛

```
        bf.or(other.bf);
    }
```

我们会在分布式模式下利用这个归并的技巧来生成Bloom filter。每个mapper根据自己的数据分片构造一个Bloom filter。我们再把这些Bloom filter发送到一个单一的reducer上，将它们归并且记录最终的输出。

由于Bloom filter会随着mapper的输出被打乱，Bloom filter类必须实现Writable接口，它包括write()和readFields()方法。这些方法实现了在内部的BitSet表示和一个字节数组之间的转换，从而让这些数据可以被序列化到DataInput/DataOutput。最终的代码如下：

代码清单5-4 基础的Bloom filter实现

```
class BloomFilter<E> implements Writable {

    private BitSet bf;
    private int bitArraySize = 100000000;
    private int numHashFunc = 6;

    public BloomFilter() {
        bf = new BitSet(bitArraySize);
    }

    public void add(E obj) {
        int[] indexes = getHashIndexes(obj);

        for (int index : indexes) {
            bf.set(index);
        }
    }

    public boolean contains(E obj) {
        int[] indexes = getHashIndexes(obj);

        for (int index : indexes) {
            if (bf.get(index) == false) {
                return false;
            }
        }
        return true;
    }

    public void union(BloomFilter<E> other) {
        bf.or(other.bf);
    }

    protected int[] getHashIndexes(E obj) {
        int[] indexes = new int[numHashFunc];

        long seed = 0;
        byte[] digest;
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(obj.toString().getBytes());
            digest = md.digest();

            for (int i = 0; i < 6; i++) {
```

www.ChinaDBA.net 中国DBA超级论坛

```
            seed = seed ^ (((long)digest[i] & 0xFF))<<(8*i);
        }
    } catch (NoSuchAlgorithmException e) {}

    Random gen = new Random(seed);

    for (int i = 0; i < numHashFunc; i++) {
        indexes[i] = gen.nextInt(bitArraySize);
    }

    return indexes;
}

public void write(DataOutput out) throws IOException {
    int byteArraySize = (int)(bitArraySize / 8);

    byte[] byteArray = new byte[byteArraySize];
    for (int i = 0; i < byteArraySize; i++) {
        byte nextElement = 0;
        for (int j = 0; j < 8; j++) {
            if (bf.get(8 * i + j)) {
                nextElement |= 1<<j;
            }
        }

        byteArray[i] = nextElement;
    }
    out.write(byteArray);
}

public void readFields(DataInput in) throws IOException {
    int byteArraySize = (int)(bitArraySize / 8);
    byte[] byteArray = new byte[byteArraySize];

    in.readFully(byteArray);

    for (int i = 0; i < byteArraySize; i++ ) {
        byte nextByte = byteArray[i];
        for (int j = 0; j < 8; j++) {
            if (((int)nextByte & (1<<j)) != 0) {
                bf.set(8 * i + j);
            }
        }
    }
}
```

然后，我们使用Hadoop中的MapReduce程序生成Bloom filter。如前所述，每个mapper会实例化一个BloomFilter对象，并将其分片中每个记录的键添加到它的BloomFilter实例中。（我们使用记录的键是为了遵循数据连接的示例程序。）通过把它们收集到单个reducer中，我们就生成了这些BloomFilter的一个合并。

MapReduce程序的driver很简单。我们在mapper中会输出一个键/值对，其中值为一个BloomFilter实例。

```
job.setOutputValueClass(BloomFilter.class);
```

www.ChinaDBA.net 中国DBA超级论坛

因为只有一个reducer，输出的键不会涉及分区的问题。

```
job.setNumReduceTasks(1);
```

我们希望reducer将最终的BloomFilter输出为一个二进制文件。Hadoop的OutputFormat输出要么是文本文件，要么是一个键/值对。因此，我们的reducer不会使用Hadoop的MapReduce输出机制，而我们会自己把结果写入文件中。

```
job.setOutputFormat(NullOutputFormat.class);
```

警告 通常当你脱离了MapReduce的输入/输出框架并开始自己操作文件时，事情会变得更危险。不再能保证你的任务的多次执行结果是一致的，你会需要了解不同的故障情况如何影响你的任务。例如，当一些任务被重启时，你的文件可能只有部分被写入。我们这里的例子是安全（或更安全）的，因为所有的文件操作都仅在close()方法中一起执行一次，且都在一个reducer中。更为小心/极端的实现应该是更仔细地检查每个单独文件的操作。

回顾我们在mapper上的策略，是为整个分片生成一个单独的Bloom filter，并在分片处理结束时将它输出到reducer。鉴于MapClass的map()方法并不知道处理的是分片中的哪条记录，我们应当在close()方法中输出BloomFilter，来确保所有分片中的记录都被读取了。虽然OutputCollector被传递给map()方法用于收集mapper的输出，但结果并没有传给close()方法。在Hadoop中应付这种情况的标准做法是当OutputCollector被传递到map()时，由MapClass自己来维护一个到它的引用。OutputCollector对函数可见，甚至在close()方法中。MapClass如下所示：

```
public static class MapClass extends MapReduceBase
        implements Mapper<K1, V1, K2, V2> {

    OutputCollector<K2, V2> oc = null;

    public void map(K1 key, V1 value,
                    OutputCollector<K2,V2> output,
                    Reporter reporter) throws IOException {

        if (oc == null) oc = output;
        ...
    }

    public void close() throws IOException {
        oc.collect(k, v);
    }
}
```

所有mapper生成的BloomFilter都被送入一个单独的reducer中。在Reducer类中的reducer()方法会把它们全部合并为一个Bloom filter。

```
while (values.hasNext()) {
    bf.union((BloomFilter<String>)values.next());
}
```

如前所述，我们希望最终的BloomFilter按我们自己的格式被写入一个文件，而非使用Hadoop的某

www.ChinaDBA.net 中国DBA超级论坛

个OutputFormat格式。在driver中，我们已经通过设置reducer的OutputFormat为NullOutputFormat，关闭了它的输出机制。现在close()方法必须自己处理文件的输出。它必须知道由用户定义的输出路径，该路径可以在JobConf对象的mapred.output.dir属性中找到。但是我们采用与mapper处理OutputCollector相同的方式。我们在Reduce类中维护一个对JobConf对象的引用，使之可以被close()方法使用。然后剩余的close()方法就可以使用Hadoop的文件I/O，把BloomFilter以二进制方式写入HDFS文件中。代码清单5-5给出了完整的代码。

代码清单5-5 一个用于生成Bloom filter的MapReduce程序

```
public class BloomFilterMR extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<Text, Text, Text, BloomFilter<String>> {

        BloomFilter<String> bf = new BloomFilter<String>();
        OutputCollector<Text, BloomFilter<String>> oc = null;

        public void map(Text key, Text value,
                        OutputCollector<Text, BloomFilter<String>> output,
                        Reporter reporter) throws IOException {
            if (oc == null) oc = output;

            bf.add(key.toString());
        }

        public void close() throws IOException {
            oc.collect(new Text("testkey"), bf);
        }
    }

    public static class Reduce extends MapReduceBase
        implements Reducer<Text, BloomFilter<String>, Text, Text> {

        JobConf job = null;
        BloomFilter<String> bf = new BloomFilter<String>();

        public void configure(JobConf job) {
            this.job = job;
        }

        public void reduce(Text key, Iterator<BloomFilter<String>> values,
                           OutputCollector<Text, Text> output,
                           Reporter reporter) throws IOException {

            while (values.hasNext()) {
                bf.union((BloomFilter<String>)values.next());
            }
        }

        public void close() throws IOException {
            Path file = new Path(job.get("mapred.output.dir") +
                                 "/bloomfilter");
            FSDataOutputStream out = file.getFileSystem(job).create(file);
            bf.write(out);
            out.close();
```

www.ChinaDBA.net 中国DBA超级论坛

```
    }
}
public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    JobConf job = new JobConf(conf, BloomFilterMR.class);

    Path in = new Path(args[0]);
    Path out = new Path(args[1]);
    FileInputFormat.setInputPaths(job, in);
    FileOutputFormat.setOutputPath(job, out);

    job.setJobName("Bloom Filter");
    job.setMapperClass(MapClass.class);
    job.setReducerClass(Reduce.class);
    job.setNumReduceTasks(1);

    job.setInputFormat(KeyValueTextInputFormat.class);
    job.setOutputFormat(NullOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(BloomFilter.class);
    job.set("key.value.separator.in.input.line", ",");

    JobClient.runJob(job);

    return 0;
}
public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(),
                             new BloomFilterMR(),
                             args);

    System.exit(res);
}
}
```

5.3.3 Hadoop 0.20 以上版本的 Bloom filter

Hadoop版本0.20中有一个Bloom filter类。它支持版本0.20中引入的一些新类，在将来的版本中它很可能会继续保留。它的功能很像代码列表5-4中的BloomFilter类，但是它在散列函数实现上更为严格。作为一个内部类，它对在Hadoop中实现半联结是一个很好的选择。但并不容易将它从Hadoop的框架中分离出来，作为一个单独的类来使用。如果你需要为非Hadoop应用开发Bloom filter，Hadoop内置的BloomFilter可能并不是理想的选择。

5.4 温故知新

通过这些练习，你可以检验自己对更高阶MapReduce技术的理解：

(1) 异常检测　取一个web服务器的日志文件。写MapReduce程序累计每个IP地址的访问次数。写另一个MapReduce程序查找访问数最多的前K个IP地址。这些频繁的访问者可能是合法的ISP 代理（多个用户之间共享），也可能是爬虫和骗子（如果服务器日志是来自与一个广告网络）。将这两个MapReduce作业结合在一起，使它们可以易于按日执行。

www.ChinaDBA.net 中国DBA超级论坛

(2) 筛选输入记录 在我们使用过的两个专利数据集（cite75_99.txt和apat63_99.txt）中，第一行为元数据（列名）。到目前为止，我们不得不显式或隐式地在mapper中剔除这一行，或者当知道该元数据记录确定的影响时，对结果进行解释。一个更终极的解决方案是从输入数据中删除这个元数据行，并在别处跟踪它。另一个解决方案是把一个mapper写为预处理器，筛选所有看起来像元数据的记录。（例如，元数据记录不会以专利号的数字为起始。）写这样一个mapper，使用ChainMapper/ChainReducer将它集成在你的MapReduce程序中。

(3) 不相交的选择 用datajoin软件包处理相同的Customers和Orders示例，你如何改变代码来输出那些不在Orders数据源中的顾客？也许Orders数据只包含最后*N*个月内的订单，并且这些顾客在这个时间段没有购买任何东西。在商业策略上，也许会选择联系这些客户提供折扣优惠或其他奖励。

(4) 计算比率 比率通常是一个比原始数字更易于分析的单位。例如，假设你有一个当天股价的数据集和一个昨日股价的数据集。相比于其绝对价格，你可能对每支股票的增长率更感兴趣。使用datajoin框架编写一个程序，取出两个数据源并输出其比率。

(5) 矩阵向量乘 在线性代数课本中找一段你最喜欢的矩阵乘法的定义。实现一个计算一个向量和一个矩阵乘积的MapReduce作业。你应使用DistributedCache来保存向量的值，可以假设矩阵是稀疏的。

(6) 空间联结 让我们再大胆一些。考虑*x*和*y*坐标范围均从−1 000 000 000到+1 000 000 000的一个二维空间。你有一个文件位于foos，而另一个文件位于bars。在这些文件中的每条记录都是以逗号分隔的(*x*，*y*)坐标。例如，可能为如下两行

145999.32455, 888888880.001

834478899.2, 5656.87660922

写一个MapReduce作业，查找距离一个bar小于1个单元的所有foo。距离的计算可使用常见的欧氏距离公式$\mathrm{sqrt}[(x1-x2)^2+(y1-y2)^2]$。虽然foos和bars在此二维空间中均相对稀疏，它们各自的文件却太大，而无法存储在内存中。你不能使用DistributedCache做空间连接。

提示1 当前实现的datajoin软件包也不能很好地解决这个问题，但你可以自己写mapper和reducer，采取与datajoin软件包相类似的数据流来解决。

提示2 在迄今为止我们讨论的所有MapReduce 程序中，键仅用于提取和传递，而值会经历各种计算。应该考虑计算键作为mapper的输出。

(7) 用Bloom filter强化空间联结 在回答上一个问题后，找出如何使用Bloom filter来加速联结操作的方法。假定bar远比foo少，但仍然无法将所有的位置放入内存中。

5.5 小结

在我们运行作业处理数据集时，这个作业通常都可以通过编写基本的MapReduce程序来生成。但有时，我们也会需要编写更高级的程序来形成多个作业，或者我们会用它们处理多个数据集。Hadoop有几种不同的方式把多个作业协调在一起，包括按照顺序链接或者按照预定义的依赖

www.ChinaDBA.net 中国DBA超级论坛

关系来执行。如果需要频繁地将多个仅包含map的作业联结成一个完整MapReduce作业，Hadoop提供了特殊的类使之高效实现。

连接是多个数据源处理数据的一个经典示例。虽然Hadoop有一个强大的datajoin软件包可以做任意的联结，但它的通用性是以效率为代价的。在大多数的数据联结中，数据源大小的相对不对称性是一个典型现象，通过挖掘该特征，就可以采用几种不同的联结方法来实现更快的联结。其中一种方法是利用Bloom filter，这是一个在许多数据处理任务中都很有用的数据结构。

到此，你对MapReduce编程的了解应该能够使你开始编写自己的程序。所有程序员都知道，编程并非仅是编写代码，而且需要遵循各种技术与流程——历经开发、部署、测试和调试。MapReduce编程和分布式计算的内在特性给这些过程增添了复杂性，并产生出细微的差别，我们将在下一章中进行阐述。

5.6　更多资源

http://portal.acm.org/citation.cfm?doid=1247480.1247602—MapReduce缺乏对数据集连接的支持是广为人知的。许多用于强化Hadoop的工具（如Pig、Hive和CloudBase）提供数据连接作为首要操作。更正式的解决方案出现在Hung-chih Yang及其合作者发表的一篇论文“Map-reduce-merge: simplified relational data processing on large clusters”中，他们提出了一个MapReduce的改进形式，即添加一个“merge”步骤来支持数据连接。

http://umiacs.umd.edu/~jimmylin/publications/Lin_etal_TR2009.pdf—Section 5.2.2 描述了如何使用分布式缓存给任务提供辅助数据。这种技术的局限性是辅助数据会被复制到每个TaskTracker上，而且必须装入内存中。Jimmy Lin及其同事们探索使用memcached（一种分布的缓存系统）来提供对辅助数据的全局访问。他们的经验总结在论文“Low-Latency, High-Throughput Access to Static Global Resources within the Hadoop Framework.”中。

www.ChinaDBA.net 中国DBA超级论坛

第6章

编程实践

6

本章内容

- Hadoop程序开发的独门绝技
- 在本地、伪随机和全分布模式下调试程序
- 程序输出的完整性检查和回归测试
- 日志和监控
- 性能调优

既然你已经学习了MapReduce中的各种编程技术，本章将回溯到对编程实践的讨论。

Hadoop编程与传统编程的不同主要体现在两个方面。其一，Hadoop 程序主要是关于数据处理的；其二，Hadoop程序运行在一组分布的计算机上。这两点差异将使开发和调试过程发生改变，我们会在6.1节和6.2节中具体介绍它们。

在性能优化技术上，Hadoop没有什么特别之处，也是与特定的编程平台相关。在6.3节会讨论Hadoop程序的优化工具和方法。

让我们从Hadoop的开发技术开始讨论。想必你已经对标准的Java软件工程方法很熟悉了。这里我们专门讨论Hadoop中数据处理程序的编程实践。

6.1 开发 MapReduce 程序

第2章讨论了Hadoop的3种模式：本地（独立）、伪分布与全分布。它们大致对应于开发、筹划和运营3种部署方式。开发过程会逐步经历这3种模式，因此必须能够方便地进行配置切换。在实践中甚至可能有几个采用全分布模式的集群。例如，更大型的企业在把MapReduce程序真正在生产型集群上运行之前，会在“开发”集群上让程序更为稳定。你还可能需要有运行不同工作负载的多个集群。例如，在内部集群上可以运行许多中小规模的作业；在云上的集群可以更高效地运行庞大但不频繁发生的作业。

2.3节讨论了如何为不同安装环境设置不同版本的`hadoop-site.xml`配置文件，然后用符号链接切换到当前想要使用的配置。你还可以通过带有`-conf`选项的Hadoop命令来明确地指定配置文件。例如，

www.ChinaDBA.net 中国DBA超级论坛

```
bin/hadoop fs -conf conf.cluster/hadoop-site.xml -lsr
```

将列出在全分布模式下集群的所有文件，即使你可能当前正工作在不同的模式或不同的集群上（假定conf.cluster/hadoop-site.xml这个位置存放了全分布模式的集群配置文件）。

在运行和测试Hadoop程序之前，你需要让正在运行的配置可以获得这些数据。3.1节描述了多种从HDFS中读写数据的方法。对于本地和伪分布模式，你只需要完整数据的一个子集。根据4.4节的介绍，对一个HDFS中的数据集，你可以用Streaming程序（RandomSample.py）按比例地进行随机采样。因为该程序是一个Python脚本，你还可以通过Unix管道使用它来采样本地文件：

```
cat datafile | RandomSample.py 10
```

这条命令提取datafile文件中十分之一的数据。

既然你已经安装了各种不同的配置，并且知道如何为每个配置提供数据。现在让我们看一看如何开发和调试本地和伪分布模式。这些技术环环相扣，引领你了解生产环境。我们将把全分布式集群调试的讨论推迟到下一节。

6.1.1　本地模式

本地模式下的Hadoop将所有的运行都放在一个单独的Java虚拟机（JVM）中完成，并且使用的是本地文件系统（即非HDFS）。在一个JVM中运行允许你使用所有熟悉的Java开发工具，如debugger。使用来自本地文件系统的文件，意味着你可以快速应用Unix命令或简单的脚本来处理输入和输出数据，而检查HDFS文件就只能使用Hadoop命令行所提供的命令。例如，要计算在输出文件中有多少条记录，如果该文件是在本地文件系统中，可以使用wc-l。如果该文件位于HDFS，就必须写MapReduce程序，或者先将文件下载到本地存储，然后再用Unix命令。正如你所看到的，能够很容易地访问输入和输出文件，对于我们在本地模式下的程序开发实践是非常重要的。

注意　本地模式严格遵循着Hadoop的MapReduce编程模型，但它并不支持每一项功能。例如，它不支持分布式缓存，以及它最多只允许有一个reducer。

在本地模式中运行的程序将所有的日志和错误消息都输出到控制台。最后，它还会给出所处理数据的总量。例如，运行我们写的最基本的MapReduce作业（MyJob.java）来对专利引用数据做倒排，其输出会相当冗长，图6-1是作业输出结果中间部分的快照。

在作业结束时，Hadoop会打印出各种内部计数器的值，包括记录的个数以及MapReduce不同阶段所处理的字节数：

```
09/05/27 03:34:37 INFO mapred.TaskRunner: Task
➥ attempt_local_0001_r_000000_0' done.
09/05/27 03:34:37 INFO mapred.TaskRunner: Saved output of task
➥ attempt_local_0001_r_000000_0' to
➥ file:/Users/chuck/Projects/Hadoop/hadoop-0.18.1/output/test
```

www.ChinaDBA.net 中国DBA超级论坛

```
Terminal — bash — 132x30
09/05/27 03:33:04 INFO mapred.MapTask: kvstart = 262144; kvend = 196607; length = 327680
09/05/27 03:33:05 INFO mapred.MapTask: Index: (0, 4716118, 4716118)
09/05/27 03:33:05 INFO mapred.MapTask: Finished spill 1
09/05/27 03:33:06 INFO mapred.MapTask: Spilling map output: buffer full = false and record full = true
09/05/27 03:33:06 INFO mapred.MapTask: bufstart = 8383919; bufend = 12575596; bufvoid = 99614720
09/05/27 03:33:06 INFO mapred.MapTask: kvstart = 196607; kvend = 131070; length = 327680
09/05/27 03:33:07 INFO mapred.MapTask: Index: (0, 4715965, 4715965)
09/05/27 03:33:07 INFO mapred.MapTask: Finished spill 2
09/05/27 03:33:07 INFO mapred.LocalJobRunner: file:/Users/chuck/Projects/Hadoop/hadoop-0.18.1/input/cite75_99.txt:234881024+29194407
09/05/27 03:33:07 INFO mapred.JobClient:  map 94% reduce 0%
09/05/27 03:33:07 INFO mapred.MapTask: Spilling map output: buffer full = false and record full = true
09/05/27 03:33:07 INFO mapred.MapTask: bufstart = 12575596; bufend = 16767227; bufvoid = 99614720
09/05/27 03:33:07 INFO mapred.MapTask: kvstart = 131070; kvend = 65533; length = 327680
09/05/27 03:33:09 INFO mapred.MapTask: Index: (0, 4715919, 4715919)
09/05/27 03:33:09 INFO mapred.MapTask: Finished spill 3
09/05/27 03:33:09 INFO mapred.MapTask: Spilling map output: buffer full = false and record full = true
09/05/27 03:33:09 INFO mapred.MapTask: bufstart = 16767227; bufend = 20958850; bufvoid = 99614720
09/05/27 03:33:09 INFO mapred.MapTask: kvstart = 65533; kvend = 327677; length = 327680
09/05/27 03:33:10 INFO mapred.LocalJobRunner: file:/Users/chuck/Projects/Hadoop/hadoop-0.18.1/input/cite75_99.txt:234881024+29194407
09/05/27 03:33:10 INFO mapred.JobClient:  map 96% reduce 0%
09/05/27 03:33:11 INFO mapred.MapTask: Index: (0, 4715913, 4715913)
09/05/27 03:33:11 INFO mapred.MapTask: Finished spill 4
09/05/27 03:33:11 INFO mapred.MapTask: Spilling map output: buffer full = false and record full = true
09/05/27 03:33:11 INFO mapred.MapTask: bufstart = 20958850; bufend = 25150768; bufvoid = 99614720
09/05/27 03:33:11 INFO mapred.MapTask: kvstart = 327677; kvend = 262140; length = 327680
09/05/27 03:33:13 INFO mapred.MapTask: Index: (0, 4716206, 4716206)
09/05/27 03:33:13 INFO mapred.MapTask: Finished spill 5
09/05/27 03:33:13 INFO mapred.MapTask: Starting flush of map output
09/05/27 03:33:13 INFO mapred.MapTask: bufstart = 25150768; bufend = 29194395; bufvoid = 99614720
09/05/27 03:33:13 INFO mapred.MapTask: kvstart = 262140; kvend = 187317; length = 327680
```

图6-1　在本地模式中运行一个Hadoop程序会输出所有的log信息到终端上

```
09/05/27 03:34:37 INFO mapred.LocalJobRunner: reduce > reduce
09/05/27 03:34:37 INFO mapred.JobClient: Job complete: job_local_0001
09/05/27 03:34:37 INFO mapred.JobClient: Counters: 11
09/05/27 03:34:37 INFO mapred.JobClient:   Map-Reduce Framework
09/05/27 03:34:37 INFO mapred.JobClient:     Map output records=16522439
09/05/27 03:34:37 INFO mapred.JobClient:     Reduce input records=33044878
09/05/27 03:34:37 INFO mapred.JobClient:     Map output bytes=264075431
09/05/27 03:34:37 INFO mapred.JobClient:     Map input records=16522439
09/05/27 03:34:37 INFO mapred.JobClient:     Combine output records=0
09/05/27 03:34:37 INFO mapred.JobClient:     Map input bytes=264075431
09/05/27 03:34:37 INFO mapred.JobClient:     Combine input records=0
09/05/27 03:34:37 INFO mapred.JobClient:     Reduce input groups=6517968
09/05/27 03:34:37 INFO mapred.JobClient:     Reduce output records=6517968
09/05/27 03:34:37 INFO mapred.JobClient:   File Systems
09/05/27 03:34:37 INFO mapred.JobClient:     Local bytes written=4246405780
09/05/27 03:34:37 INFO mapred.JobClient:     Local bytes read=4276658154
```

MapReduce作业的输入和输出都在本地文件系统中。我们可以用标准的Unix命令（如wc-l或head）来检查他们。因为我们在开发过程中特意使用了较小的数据集，我们甚至可以把它们加载到文本编辑器或电子表格中。这些应用程序有很多功能可用于对程序做正确性检查。

1. 完整性检查

大多数MapReduce 程序至少会涉及一些计数或算术操作，但是在数学公式中的一些错误（尤其是拼写错误）不会抛出异常或威胁性的错误消息，故不引人注意。即使程序在技术上是“正确的”并且运行顺利，但数学上它可能是错误的，最终的结果也会没有用。发现算术错误的方法都比较复杂，但一些完整性检查可以帮助我们。你可以宏观地查看各项指标的整体计数、最大值、平均值等，看它们是否达到期望。在底层，你可以选择一个特定的输出记录，来验证它所生成结果的正确性。例如，当我们创建倒排引用图时，前几行如下

www.ChinaDBA.net 中国DBA超级论坛

```
"CITED" "CITING"
1       3964859,4647229
10000   4539112
100000  5031388
1000006 4714284
1000007 4766693
1000011 5033339
1000017 3908629
1000026 4043055
1000033 4190903,4975983
```

作业得到的结果为：专利号1的引用为两次，分别为3964859和4647229。为了验证这个结果，我们可以在采样后的输入数据中查找专利号1被引用的记录。

```
grep ",1$" input/cite75_99.txt
```

我们的确得到了与预期相一致的两项纪录。你可以多验证一些记录，来获得对程序的数学和逻辑正确性的信心。[①]

不过，这个倒排引用图有一个让人感觉不顺眼的输出，即第一行并不是真实的数据。

```
"CITED" "CITING"
```

它来自输入数据的第一行，是用来做数据定义的。我们在mapper中增加一段代码来过滤掉非数字的键和值，该过程就是回归测试。

2. 回归测试

我们的处理以数据为中心，回归测试就是“比较”代码改变前后输出文件的差异。对于我们做的这个改变，只需从该作业的输出中提取一行。要验证的确如此，让我们先保存当前作业的输出。在本地模式中，我们最多只能有一个reducer，所以该作业的输出只有一个文件，我们称之为`job_1_output`。

对于回归测试，保存map阶段的输出也会很有帮助。这将有助于我们隔离出bug是在map阶段还是在reduce阶段。我们可以通过运行不带reducer的MapReduce作业，来保存map阶段的输出。使用选项`-D mapred.reduce.tasks=0`可以很容易地做到这一点。在这个仅有mapper的作业中，会生成多个文件，每个map任务都会将它的输出写入自己的文件中。我们将它们全部复制到一个名为`job_1_intermediate`的目录中。

保存了输出文件，我们就可以在MapClass的`map()`方法中修改一些代码。代码本身并不重要，我们专注于对它进行测试。

```
public void map(Text key, Text value,
                OutputCollector<Text, Text> output,
                Reporter reporter) throws IOException {
  try
  {
     if (Integer.parseInt(key.toString()) > 0 &&
```

① 此时，你可能会怀疑是否专利号1真的被那两项专利应用。感觉专利号1有问题，是因为它在被引用专利号正常范围之外。这可能是原始输入数据中存在错误。如果想验证它，我们不得不自己查找专利。在任何情况下，确保数据的质量是一个重要的话题，但这超出了我们讨论Hadoop的范围。

www.ChinaDBA.net 中国DBA超级论坛

```
            Integer.parseInt(value.toString()) > 0)
        {
            output.collect(value, key);
        }
    } catch (NumberFormatException e) { }
}
```

编译这段新代码并在相同的输入数据上执行。让我们先用只有map的作业来执行，并比较中间数据。因为我们只修改了mapper，任何bug都会首先引起在中间数据上的差异。

```
diff output/job_1_intermediate/ output/test/
```

通过diff工具，我们得到以下输出：

```
Binary files output/job_1_intermediate/.part-00000.crc and
➥ output/test/.part-00000.crc differ
diff output/job_1_intermediate/part-00000 output/test/part-00000
1d0
< "CITED"       "CITING"
```

我们在二进制文件.part-00000.cr中发现了差异。这是一个HDFS用于为文件part-00000保存校验和的内部文件。校验和上的差异意味着part-00000已经更改，然后diff打印了具体的不同。新的中间文件，在output/test下，没有了引用字段的描述。更重要的是，我们没有发现任何其他的不同。目前为止一切还不错。如果我们用一个reducer来运行整个作业，我们期待最终的输出也仅有一行会不同。

嗯，不过并非如此。如果你用一个reducer来运行整个作业，并与原始的运行结果job_1_output进行比较，就会发现有许多不同之处。想想究竟是怎么回事？让我们看看不同之处的前几行。

```
$ diff output/job_1_output output/test/part-00000 | head -n 15
1,2c1
< "CITED"       "CITING"
< 1     3964859,4647229
---
> 1     4647229,3964859
19c18
< 1000067       5312208,4944640,5071294
---
> 1000067       4944640,5071294,5312208
22,23c21,22
< 1000076       4867716,5845593
< 1000083       5566726,5322091
---
> 1000076       5845593,4867716
> 1000083       5322091,5566726
```

我们看到包含（“CITED”和“CITING”）字段描述符的行都如预期一样被去掉了。剩下的不同存在一定的模式。

在我们的reduce()方法中，每个键对应的一组值被联结在一起，它们的顺序是由Hadoop决定的。但是Hadoop并不保证这些值总按一样的顺序排列。我们看到去掉中间数据中的一行就会影响到reducer多个键对应值的顺序。正如我们所知，这个作业的正确性是与顺序无关的，我

www.ChinaDBA.net 中国DBA超级论坛

们可以忽略这个差异。回归测试本身是保守的，往往会给出错误的警告。你在使用它的时候应该谨记这一点。

我们主张在开发中使用采样后的数据集，因为它相比于生产环境中所使用的数据集，在结构和属性上更具代表性。我们在回归测试中使用了相同的样本数据集，而你也可以手动地构造另一个考虑边界情况的输入数据集，这种数据集在生产数据中并不典型。例如，你可以在这个结构化的数据中插入空值、额外的制表符或其他不寻常的记录。这个测试数据集是为了确保你的程序始终能够处理各种边界情况。这个测试数据集不会多于几十个记录。你可以目测检查整个输出，以查看程序的运行是否仍然与预期一致。

3. 考虑使用LONG而非INT

大多数Java程序员本能地用int类型（或Integer或IntWritable）来表示整数值。在Java中int类型可以容纳从$2^{31}-1$至-2^{31}之间，或者从2 147 483 647至-2 147 483 648之间的任何整数。这对大多数应用程序已经足够了，很少有程序员会考虑太多。可是，当你正在处理Hadoop这种规模的数据时，一些计数器的变量会需要更大的范围，这并不罕见。在开发数据集的环境下是不会有这种需求的，它的设计规模很小。甚至在你当前的生产型数据中也看不到这种需求，但是随着业务的扩展，数据集将变得更大。你最终会遇到一些变量超过int能表示的范围而导致算术错误。以经典的单词统计为例，当处理数以百万计的文档时，任何单词的计数值都不会超过20亿个，[①]用int型表示是足够的。但是，随着你处理的文档数增长到上千万或上亿个时，统计像the这样的频繁使用的词就可能超过int类型的界限。若这种bug影响到你的生产系统，这会更难调试和修复，为了避免这种情况的发生，你应该现在就通读代码，并仔细考虑数值变量应该变为long还是LongWritable，以应对未来的规模扩展。[②]

6.1.2　伪分布模式

本地模式不具备生产型Hadoop集群的分布式特征。一些bug在运行本地模式时是不会出现的。Hadoop提供了一个伪分布模式，它具备生产集群的所有功能及“节点”——Name Node、SecondaryNameNode、DataNode、JobTracker和TaskTracker，分别运行在不同的JVM上。所有的软件组件都是分布的，伪分布模式与生产型集群的区别仅仅在于系统和硬件规模上。它仅使用一个物理机器——你自己的本地计算机。在把作业部署到一个完全的生产集群上以前，我们应该确保作业能够在伪分布模式中运行。

第2章描述了启动伪分布模式的配置和命令。你在自己的计算机上启动所有的守护进程，让它们就像工作在集群上一样。你和这个计算机的交互也像和一个Hadoop集群交互一样。你把数据放入它自己的HDFS文件系统中。你通过向它提交作业来运行，而不是放在相同的用户空间来运行。更重要的是，你现在是通过日志文件和web界面“远程地”监视它。这些工具和以后在监控

① 这不是绝对真实的，且将取决于你的文档大小和内容。

② 超过数字区间的问题不是Hadoop所独有的。你会记得在早期的程序中著名的Y2K问题，就是仅分配两个数字来表示一年。最近，几乎所有的web业务都经历了爆炸式的增长（如Facebook、Twitter和RockYou），它们不得不调整系统来处理比他们原来预期范围更大的用户数或文件数。

www.ChinaDBA.net 中国DBA超级论坛

生产集群时用的工具是相同的。

1. 日志

让我们把本地模式下同样的作业放在伪分布集群中运行。使用hadoop fs命令可以把输入文件放入HDFS中。使用与本地模式中相同的hadoop jar命令来提交作业去运行。

首先你会注意到在终端上不会再显示不断滚动的消息。你仅会得到对map阶段和reduce阶段执行进度的度量，并在最后会出现一个与本地模式相同的汇总。如图6-2所示。

```
Terminal — bash — 124x59
09/05/29 04:47:38 INFO mapred.JobClient:  map 36% reduce 0%
09/05/29 04:47:43 INFO mapred.JobClient:  map 44% reduce 0%
09/05/29 04:47:48 INFO mapred.JobClient:  map 50% reduce 0%
09/05/29 04:48:16 INFO mapred.JobClient:  map 61% reduce 0%
09/05/29 04:48:21 INFO mapred.JobClient:  map 63% reduce 0%
09/05/29 04:48:26 INFO mapred.JobClient:  map 70% reduce 16%
09/05/29 04:48:31 INFO mapred.JobClient:  map 77% reduce 16%
09/05/29 04:48:36 INFO mapred.JobClient:  map 82% reduce 16%
09/05/29 04:48:41 INFO mapred.JobClient:  map 89% reduce 16%
09/05/29 04:48:46 INFO mapred.JobClient:  map 95% reduce 16%
09/05/29 04:48:50 INFO mapred.JobClient:  map 98% reduce 16%
09/05/29 04:48:55 INFO mapred.JobClient:  map 100% reduce 16%
09/05/29 04:49:20 INFO mapred.JobClient:  map 100% reduce 25%
09/05/29 04:49:23 INFO mapred.JobClient:  map 100% reduce 67%
09/05/29 04:49:25 INFO mapred.JobClient:  map 100% reduce 68%
09/05/29 04:49:29 INFO mapred.JobClient:  map 100% reduce 69%
09/05/29 04:49:34 INFO mapred.JobClient:  map 100% reduce 71%
09/05/29 04:49:36 INFO mapred.JobClient:  map 100% reduce 72%
09/05/29 04:49:39 INFO mapred.JobClient:  map 100% reduce 73%
09/05/29 04:49:41 INFO mapred.JobClient:  map 100% reduce 74%
09/05/29 04:49:44 INFO mapred.JobClient:  map 100% reduce 75%
09/05/29 04:49:49 INFO mapred.JobClient:  map 100% reduce 76%
09/05/29 04:49:51 INFO mapred.JobClient:  map 100% reduce 78%
09/05/29 04:49:54 INFO mapred.JobClient:  map 100% reduce 79%
09/05/29 04:49:56 INFO mapred.JobClient:  map 100% reduce 80%
09/05/29 04:49:59 INFO mapred.JobClient:  map 100% reduce 81%
09/05/29 04:50:04 INFO mapred.JobClient:  map 100% reduce 83%
09/05/29 04:50:06 INFO mapred.JobClient:  map 100% reduce 84%
09/05/29 04:50:09 INFO mapred.JobClient:  map 100% reduce 85%
09/05/29 04:50:11 INFO mapred.JobClient:  map 100% reduce 86%
09/05/29 04:50:14 INFO mapred.JobClient:  map 100% reduce 87%
09/05/29 04:50:19 INFO mapred.JobClient:  map 100% reduce 88%
09/05/29 04:50:21 INFO mapred.JobClient:  map 100% reduce 89%
09/05/29 04:50:24 INFO mapred.JobClient:  map 100% reduce 91%
09/05/29 04:50:26 INFO mapred.JobClient:  map 100% reduce 92%
09/05/29 04:50:29 INFO mapred.JobClient:  map 100% reduce 93%
09/05/29 04:50:34 INFO mapred.JobClient:  map 100% reduce 94%
09/05/29 04:50:35 INFO mapred.JobClient: Job complete: job_200905290339_0001
09/05/29 04:50:35 INFO mapred.JobClient: Counters: 16
09/05/29 04:50:35 INFO mapred.JobClient:   Job Counters
09/05/29 04:50:35 INFO mapred.JobClient:     Data-local map tasks=4
09/05/29 04:50:35 INFO mapred.JobClient:     Launched reduce tasks=1
09/05/29 04:50:35 INFO mapred.JobClient:     Launched map tasks=4
09/05/29 04:50:35 INFO mapred.JobClient:   Map-Reduce Framework
09/05/29 04:50:35 INFO mapred.JobClient:     Map output records=16522438
09/05/29 04:50:35 INFO mapred.JobClient:     Reduce input records=16522438
09/05/29 04:50:35 INFO mapred.JobClient:     Map output bytes=264075414
09/05/29 04:50:35 INFO mapred.JobClient:     Map input records=16522439
09/05/29 04:50:35 INFO mapred.JobClient:     Combine output records=0
09/05/29 04:50:35 INFO mapred.JobClient:     Map input bytes=264075431
09/05/29 04:50:35 INFO mapred.JobClient:     Combine input records=0
09/05/29 04:50:35 INFO mapred.JobClient:     Reduce input groups=3258983
09/05/29 04:50:35 INFO mapred.JobClient:     Reduce output records=3258983
09/05/29 04:50:35 INFO mapred.JobClient:   File Systems
09/05/29 04:50:35 INFO mapred.JobClient:     HDFS bytes written=158078522
09/05/29 04:50:35 INFO mapred.JobClient:     Local bytes written=1040839754
09/05/29 04:50:35 INFO mapred.JobClient:     HDFS bytes read=264087722
09/05/29 04:50:35 INFO mapred.JobClient:     Local bytes read=735943048
chuck-lams-computer:~/Projects/Hadoop/hadoop-0.18.1 chuck$
```

图6-2 在伪分布模式中，终端仅输出一个作业的进度和最终的计数值

www.ChinaDBA.net 中国DBA超级论坛

Hadoop并没有停止输出调试信息。事实上，现在它的输出更多了。这些信息并没有打印在终端屏幕上，而是被保存在日志文件中了。

你可以在/logs目录中找到这些日志文件。不同的服务（NameNode、JobTracker等）会各自产生不同的日志文件。从文件名上就能分辨出服务所对应的日志文件。Hadoop每天循环地记录日志。结尾为.log的日志是最近的。较早的日志都被添加了日期。默认设置下，Hadoop并不会自动删除旧的日志文件。你应该主动地归档并删除它们，以确保它们不会占用太多的空间。

NameNode、SecondaryNameNode、DataNode和JobTracker的日志文件分别用于各自服务的调试。在伪分布模式下它们并不太重要。而在生产集群中，系统管理员可以通过它们来调试相应节点上出现的问题。不过作为程序员，你会始终关心TaskTracker的日志，因为它记录了抛出异常的事件。

MapReduce程序可以将其自己的日志信息输出到STDOUT和STDERR（在Java里为System.out和System.err）。Hadoop分别将它们记录在文件stdout 和stderr中。任务的每次执行都会生成一个不同的文件。（若第一个任务执行失效，可能会再试多次。）这些用户日志文件放在子目录/logs/userlogs中。

除了将日志输出到STDOUT 和STDERR之外，程序还可以使用setStatus()方法传送实时的状态信息，这个方法位于被传递给map()和reduce()方法的Reporter对象中。（对于Streaming程序，状态信息的更新是以reporter:status:message的字符串形式发送给STDERR的。）这对长期运行的作业而言是非常有用的，你可以在它们运行时进行监控。状态信息显示在JobTracker的Web用户界面中，我们下面会进行介绍。

2. JOBTRACKER的WEB用户界面

显然事件是发生在分布式程序中多个不同位置上的。这使得监控更加困难。系统更像是一个黑盒，我们需要特殊的监控工具才能探测到系统的各种状态。JobTracker提供了一个web界面来跟踪作业的进度和各种状态。在默认配置下，你可以通过浏览器访问

```
http://localhost:50030/jobtracker.jsp
```

来查看伪分布集群中监控工具的起始页面。[1]它给出了Hadoop集群的总体信息，以及正在运行、完成以及失效的作业列表，见图6-3。

Hadoop内部通过作业ID来跟踪作业。一个作业ID是前缀为job_的字符串，加集群的ID（为集群启动时的时间戳），再加一个自动增量的作业号。Web用户界面列出了带有用户名和作业名的每个作业。在伪分布模式中，分辨当前正在执行的作业相对而言是很容易的，因为你一次仅执行一个作业。而在多用户的生产环境下，你就不得不通过Hadoop的用户名和当前的作业名来定位你的作业了。作业名是通过JobConf对象中的setJobName()方法来设定的。一个Streaming作业的名称是通过如表6-1所示的一个配置属性来设定的。

① 在全分布模式中，“localhost”会被替代为JobTracker中主节点的域名。

www.ChinaDBA.net 中国DBA超级论坛

localhost Hadoop Map/Reduce Administration

http://localhost:50030/jobtracker.jsp

localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Fri May 29 03:39:32 PDT 2009
Version: 0.18.2-dev, r
Compiled: Thu May 14 15:55:01 PDT 2009 by chuck
Identifier: 200905290339

Cluster Summary

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
1	0	2	1	2	2	4.00

Running Jobs

Running Jobs								
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_200905290339_0002	chuck	DataJoin	0.00%	4	0	0.00%	2	0

Completed Jobs

www.ChinaDBA.net 中国DBA超级论坛

Completed Jobs								
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_200905290339_0001	chuck	DataJoin	100.00%	4	4	100.00%	1	1

Failed Jobs

Done

图6-3 JobTracker Web用户界面的主页面

表6-1 用于设置一个作业名称的配置属性

属 性	描 述
mapred.job.name	表示作业名称的字符串属性

在管理页面，你可以看到每个作业都有一个map阶段的完成比例。它显示了作业的map任务总数以及已经完成的个数。在reduce侧也有相同的信息。这让你对作业的进度有一个大致了解。要掌握一个特定作业的更多细节，你可以点击这个作业ID，这是一个链接，它会把你带到作业的管理页面，见图6-4。

这个作业页面显示了运行作业所产生的各种输入/输出量。这个页面周期性地自动刷新，不过你也可以手工地去刷新它来得到更及时的数字。在这个页面提供的链接中，你可以从多方面了解这个作业的情况。例如，点击map链接将为你展示作业中所有map任务的列表，见图6-5。

www.ChinaDBA.net 中国DBA超级论坛

Hadoop job_200905290339_0002 on localhost

http://localhost:50030/jobdetails.jsp?jobid=jo

Hadoop job_200905290339_0002 on localhost

User: chuck
Job Name: DataJoin
Job File: hdfs://localhost:9000/tmp/hadoop-chuck/mapred/system/job_200905290339_0002/job.xml
Status: Running
Started at: Fri May 29 06:19:54 PDT 2009
Running for: 1mins, 58sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	4	0	2	2	0	0 / 0
reduce	0.00%	2	0	2	0	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Data-local map tasks	0	0	4
	Launched reduce tasks	0	0	2
	Launched map tasks	0	0	4
Map-Reduce Framework	Map output records	8,400,584	0	8,400,584
	Map output bytes	134,217,712	0	134,217,712
	Combine output records	0	0	0
	Map input records	8,400,585	0	8,400,585
	Combine input records	0	0	0
	Map input bytes	134,217,729	0	134,217,729
File Systems	Local bytes written	380,381,217	0	380,381,217
	HDFS bytes read	134,225,921	0	134,225,921
	Local bytes read	226,849,189	0	226,849,189

Done YSlow 0.373s

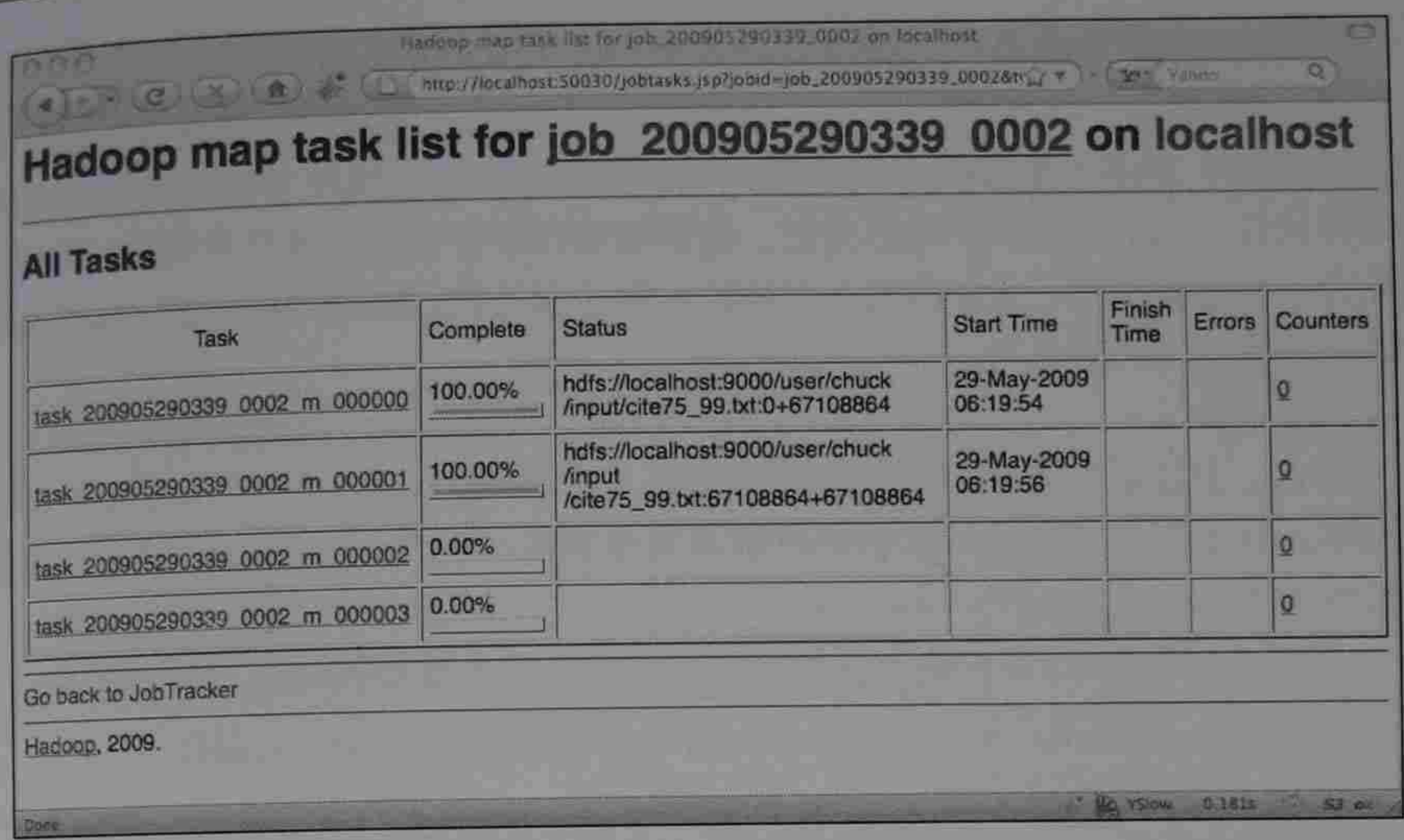
图6-4 单个作业的JobTracker管理页面

任务是通过一个任务ID来标识的。为了构建这个任务ID，你采用任务所在的作业ID并将前缀job_替换为task_。接着在它后面为map任务加上_m 或者为reduce任务加上_r。然后再在其后加上一个在每个组中自动增量的数字。在TaskTracker的Web用户界面中，你会看到每个任务都有一个状态，通过以前提到的setStatus()方法可以对它进行设置。

点击任务ID会带你进入一个新的页面，它进一步描述了一个任务几次不同的尝试。Hadoop会多次尝试重新执行一个失效的任务，避免让它影响到整个作业。

JobTracker和TaskTracker的用户界面还提供了许多其他的链接和信息。大多数都很容易理解。

www.ChinaDBA.net 中国DBA超级论坛



Hadoop map task list for job_200905290339_0002 on localhost

http://localhost:50030/jobtasks.jsp?jobid=job_200905290339_0002&t

Hadoop map task list for job_200905290339_0002 on localhost

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200905290339_0002_m_000000	100.00%	hdfs://localhost:9000/user/chuck /input/cite75_99.txt:0+67108864	29-May-2009 06:19:54			0
task_200905290339_0002_m_000001	100.00%	hdfs://localhost:9000/user/chuck /input /cite75_99.txt:67108864+67108864	29-May-2009 06:19:56			0
task_200905290339_0002_m_000002	0.00%					0
task_200905290339_0002_m_000003	0.00%					0

Go back to JobTracker

Hadoop, 2009.

图6-5 TaskTracker Web用户界面的任务列表。该图显示了单一作业中所有的map任务。每个任务可以更新自己的状态信息

3. 杀掉作业

不过，有时一个作业在启动后会出错，但是并没有真正失效。它会花很长时间来执行，或者可能陷入一个无限循环中。在（伪）分布模式中，你可以手动地杀掉这个作业，命令为

```
bin/hadoop job -kill job_id
```

这里的*job_id*就是在JobTracker的Web用户界面显示的作业ID。

6.2 生产集群上的监视和调试

在成功地在伪分布集群中运行作业之后，你可以将实际数据放在生产集群上运行。我们可以将使用过的所有技术都应用于生产集群的开发和调试中，虽然具体的使用可能会略有不同。集群仍会有JobTracker的Web用户界面，但域名不再是`localhost`，而是集群中JobTracker的地址。除非配置被更改，否则端口号仍然是50030。

在伪分布模式下，只有一个节点，所有日志文件都放在一个单独的目录`/logs`中，你可以在本地访问到它。在全分布集群中，每个节点都有其自己的`/logs`目录保存其日志文件。你可以通过特定节点上的日志文件来诊断该节点的问题。

除了迄今为止我们提到过的开发和测试技术，还有一些在生产集群中对实际数据处理更为有用的监控和调试技术，我们将在这一节探讨它们。

6.2.1 计数器

你可以在Hadoop作业中插桩计数器来分析其整体运作。在程序中定义不同的计数器，分别累计特定事件的发生次数。对于来自（同一个作业）所有任务的相同计数器，Hadoop会自动对它们

www.ChinaDBA.net 中国DBA超级论坛

进行求和，以反映整个作业的情况。这些计数器的数值会在JobTracker的Web用户界面中与Hadoop的内部计数器一起显示。

计数器的典型应用是用来跟踪不同的输入记录类型，特别是跟踪“坏”记录。回顾4.4节中的例子，即寻找每个国家专利声明的平均数。我们知道在许多记录中没有声明数。我们的程序会忽略这些记录，知道被忽略记录的数量是有用的。除了满足我们的好奇心，这种插桩让我们可以理解程序的操作并对其正确性做一些“实地检查”。

我们通过Reporter.incrCounter()方法来使用计数器。Reporter对象被传递给map()和reducer()方法。你以计数器名以及增量为参数来调用incrCounter()。每个不同的事件都有一个独立命名的计数器。当你用一个新的计数器名来调用incrCounter()，这个计数器会被初始化并进行值的累加。

Reporter.incrCounter()方法有两种签名，取决于你希望如何指定一个计数器的名字：

```
public void incrCounter(String group, String counter, long amount)
public void incrCounter(Enum key, long amount)
```

第一种形式较为常用，它允许你在运行时用一个动态的字符串来指定这个计数器的名字。它是group和counter这两个字符串的合并，这可以唯一地定义一个计数器。当计数器输出报告时(在Web用户界面或者作业执行后以文本输出)，相同group的计数器会一起报告。

第二种形式使用一个Java的enum来指定计数器的名字，它强制你在编译时定义它们，但这样也使得你可以作类型检查。enum的名字被用作group字符串，而enmu的域被用作counter字符串。

代码清单6-1是对代码清单4-12中MapClass的改写，它用计数器来跟踪丢弃值以及“quoted”值的个数。(只有列描述的第一行才能是“quoted”的值。)被称为ClaimsCounters的enum通过值MISSING和QUOTED来定义。这段代码的逻辑是累加计数器来反映所处理的记录。

代码清单6-1 使用计数器统计缺失值个数的MapClass

```
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, Text> {

    static enum ClaimsCounters { MISSING, QUOTED };

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, Text> output,
                    Reporter reporter) throws IOException {

        String fields[] = value.toString().split(",", -20);
        String country = fields[4];
        String numClaims = fields[8];

        if (numClaims.length() == 0) {
            reporter.incrCounter(ClaimsCounters.MISSING, 1);
        } else if (numClaims.startsWith("\"")) {
            reporter.incrCounter(ClaimsCounters.QUOTED, 1);
        } else {
            output.collect(new Text(country), new Text(numClaims + ",1"));
        }
    }
}
```

www.ChinaDBA.net 中国DBA超级论坛

程序运行后，可以看到我们定义的计数器和Hadoop内部的计数器都被显示在JobTracker的web用户界面中，如图6-6所示。

	Counter	Map	Reduce	Total
Job Counters	Data-local map tasks	0	0	4
	Launched reduce tasks	0	0	2
	Launched map tasks	0	0	4
Map-Reduce Framework	Reduce input records	0	151	151
	Map output records	1,984,055	0	1,984,055
	Map output bytes	18,862,764	0	18,862,764
	Combine output records	1,063	151	1,214
	Map input records	2,923,923	0	2,923,923
	Reduce input groups	0	151	151
	Combine input records	1,984,625	493	1,985,118
	Map input bytes	236,903,179	0	236,903,179
	Reduce output records	0	151	151
File Systems	HDFS bytes written	0	2,658	2,658
	Local bytes written	20,554	2,510	23,064
	HDFS bytes read	236,915,470	0	236,915,470
	Local bytes read	21,112	2,510	23,622
AveragingWithCombiner$MapClass$ClaimsCounters	QUOTED	1	0	1
	MISSING	939,867	0	939,867

图6-6 JobTracker的Web用户界面搜集并显示计数器信息

我们看到enum完全限定的Java名（按照内部类层次以$分隔）被用作group名。MISSING和QUOTED域被用来定义各自的计数器。与预期一样，QUOTED计数器只累加了一次，而MISSING计数器累加了939 867次。这个数据集有这么多行缺失的声明数吗？根据数据集提供者所说，1975年前的授权专利都没有声明数。我们看一眼图4-3，可以估计出在我们的数据集中于1975年前授权的专利大概占了总数的1/3。由map输入记录数统计（来自图6-6），可知总共有2.9M+的记录。数字看起来是对得上的，我们由此对处理的正确性更有信心了。

Streaming过程也可以使用计数器。它需要向STDERR发送一个特别格式的行，形式为

```
reporter:counter:group,counter,amount
```

这里group、counter 和amount都已经以相应参数形式传递到Java中incrCounter()中。例如，在Python中，可以通过如下形式累计ClaimsCounters.MISSING计数器

```
sys.stderr.write("reporter:counter:ClaimsCounters,MISSING,1\n")
```

一定要在最后包含换行符（“\n”）。没有它，Hadoop Streaming就无法正确地解析这个字符串。

6.2.2 跳过坏记录

在处理大数据集时，不可避免地在一些记录上会出错。你经常需要经历几轮开发过程才能实

www.ChinaDBA.net 中国DBA超级论坛

现程序在异常数据处理时的鲁棒性。[1]但是，程序可能永远也无法达到完全可靠。你需要程序处理新的数据，而新数据可能引发新的异常。而且，你的程序可能还会用到依赖于第三方软件库的解析器，从而无法控制它。虽然你应该让程序尽可能地可以处理不正常的记录，但是你还应该有一个恢复机制来处理那些你无法预见的情况。你并不希望仅仅因为无法处理一个坏记录，而导致整个作业的失效。

Hadoop对硬件失效所采取的恢复机制无法应对由于坏记录所导致的确定性软件失效。相反，它提供了一个特征，用来跳过那些被确信会导致任务崩溃的记录。如果这个特征被启动，任务在重试几次后就会进入skipping模式。只要在skipping模式中，TaskTracker就会跟踪并确定哪个记录区域会导致失效。于是TaskTracker会重启这个任务，并忽略这个坏记录区域。

1. 在JAVA中配置记录跳读

Hadoop从0.19版本起就已经支持skipping特征了，但是默认状态是关闭的。在Java中，这个特征由类SkipBadRecords来控制，全部由静态方法组成。作业的driver需要调用如下的一个或全部方法：

```
public static void setMapperMaxSkipRecords(Configuration conf,
        long maxSkipRecs)
public static void setReducerMaxSkipGroups(Configuration conf,
        long maxSkipGrps)
```

来分别为map任务和reduce任务打开记录跳读的设置。Driver使用配置对象和跳读区域中的最大记录数调用这个方法。如果最大的跳读区域大小被设置为0（默认），那么记录跳读就处于关闭状态。Hadoop使用分而治之的方法来寻找跳读区域。它每次取跳读区域中一半的任务执行，并判断这一半有没有坏记录。这个过程不断迭代，直到跳读区域的大小在可接受的范围内。这是个非常耗时的操作，特别当最大跳读区域的设置很小的时候。你可能需要在Hadoop常规的任务恢复机制中增加任务重试的最大次数来适应这些额外的重试。可以使用JobConf.setMaxMapAttempts()和JobConf.setMaxReduceAttempts()方法，或者设置等效的属性mapred.map.max.attempts和mapred.reduce.max.attempts来做到这点。

如果skipping被启用，Hadoop在任务失效两次后就进入skipping模式。你可以在SkipBadRecords的setAttemptsToStartSkipping()方法中设置触发skipping模式的任务失效次数。

```
public static void setAttemptsToStartSkipping(Configuration conf,
        int attemptsToStartSkipping)
```

Hadoop会把被跳过的记录写入HDFS以供以后分析。它们以序列文件的形式写入_log/skip目录。我们将在6.3.3节进一步介绍序列文件。现在我们可以将之视为一种特定的Hadoop压缩格式。它可以用如下的命令解压并读取。

```
bin/hadoop fs -text <filepath>
```

你可以使用方法SkipBadRecords.setSkipOutputPath(JobConf conf, Path path)修改当前用于存放被跳过记录的目录_log/skip。如果path被设为空，或者一个值为“none”的字符串Path，

---

① 异常数据并不总是错误数据。有人告诉我他有一个程序在处理用户的地理信息时崩溃了。通过深入分析发现有一个用户来自一个名为Null的实境城。

www.ChinaDBA.net 中国DBA超级论坛

Hadoop就会放弃记录被跳过的记录。

2. 在Java之外配置记录跳读

虽然你可以在driver中调用SkipBadRecords方法来设置记录跳读属性，但是有时你可能会希望使用在GenericOptionsParser的通用选项来设置它。这是因为运行程序的人对于处理坏记录，会有比原始开发者更好的主意，能够更合理地设置这些参数。而且，Streaming程序不能访问SkipBadRecords；必须使用Streaming的-D属性来设置记录跳读属性（在版本0.18中为-jobconf）。表6-2显示了与SkipBadRecords方法调用等效的JobConf属性。

表6-2 与SkipBadRecords方法调用等效的JobConf属性

SkipBadRecords方法	JobConf属性
setAttemptsToStartSkipping()	mapred.skip.attempts.to.start.skipping
setMapperMaxSkipRecords()	mapred.skip.map.max.skip.records
setReducerMaxSkipGroups()	mapred.skip.reduce.max.skip.groups
setSkipOutputPath()	mapred.skip.out.dir
setAutoIncrMapperProcCount()	mapred.skip.map.auto.incr.proc.count
setAutoIncrReducerProcCount()	mapred.skip.reduce.auto.incr.proc.count

我们还没有解释最后两个属性。它们的默认值对于大多数Java程序都是适用的，但我们需要为Streaming来修改它。

为了确定要跳过的记录区域，Hadoop需要知道一个任务已经处理的精确记录数。Hadoop使用一个内部计数器，它默认地在每次调用map（reduce）函数之后累加。对于Java程序，这是一个很好的跟踪处理记录数的方法。某些时候它也会失效，比如程序对记录异步地进行处理（即通过复制并发线程）或者将记录缓存并按块来处理，但在通常情况下这个计数器是可用的。在Streaming程序中，这个默认模式根本不起作用，因为并不存在可供调用的等效map（reduce）函数来处理每个记录。这该情况下，你不得不通过设置布尔属性为false来关闭默认模式，而你的任务必须自己更新记录的计数器。

在Python中，map任务可以用如下方式更新计数器

```
sys.stderr.write(
    ➥"reporter:counter:SkippingTaskCounters,MapProcessedRecords,1\n")
```

而在reduce任务中可以用

```
sys.stderr.write(
    ➥"reporter:counter:SkippingTaskCounters,ReduceProcessedGroups,1\n")
```

对于不能依靠默认记录计数器的Java程序，当它分别在Mapper和Reducer中处理了一个键/值对时，应该使用

```
reporter.incrCounter(SkipBadRecords.COUNTER_GROUP,
    SkipBadRecords.COUNTER_MAP_PROCESSED_RECORDS, 1);
```

以及

www.ChinaDBA.net 中国DBA超级论坛

```
reporter.incrCounter(SkipBadRecords.COUNTER_GROUP,
    SkipBadRecords.COUNTER_REDUCE_PROCESSED_GROUPS, 1);
```

6.2.3 用 IsolationRunner 重新运行出错的任务

基于日志文件的调试是通过普通的历史记录来重建事件。有时，日志并不能提供足够的信息让我们可以追溯失败的原因。Hadoop中有一个名为IsolationRunner的工具，它就像一台用于调试的时间机器。这个工具可以隔离失败的任务，并在同一节点上用完全相同的输入重新运行它。你可以添加一个调试器在任务运行时控制它，并集中收集有关失败原因的证据。

要使用IsolationRunner特征，你必须在运行作业时配置属性keep.failed.tasks.files为true。这会告诉每个TaskTracker保存所有必要的数据来重新运行失效的任务。

当一个作业失效，你可以使用JobTracker的Web用户界面来定位失效任务的节点、作业ID和该任务的重试ID。你登录到任务失效的节点并进入work目录，它位于该重试任务的目录下。进入目录：

```
local_dir/taskTracker/jobcache/job_id/attempt_id/work
```

这里*job_id*和*attempt_id*是作业ID和失效任务的重试ID。（作业ID应该以“job_”开头，而任务重试ID应该以“attempt_”开头。）根目录*local_dir*就是在配置属性mapred.local.dir中设置的目录。注意Hadoop允许一个节点使用多个本地目录（通过设置mapred.local.dir为一个以逗号分隔的目录列表）来将磁盘I/O分散到多个驱动器上。如果节点以这种方式配置，你将不得不查看所有的本地目录并找到包含*attempt_id*子目录的那一个。

在这个work目录中，你可以通过IsolationRunner采用与以前相同的输入来重新运行失效的任务。在重新运行时，我们希望JVM支持远程调试。因为我们并不直接运行JVM，而是通过bin/hadoop脚本，所以我们通过HADOOP_OPTS来指定JVM的调试选项：

```
export HADOOP_OPTS="-agentlib:jdwp=transport=dt_socket,
    ➥ server=y,address=8000"
```

它告诉JVM在端口8000上侦听调试器，并在运行代码前等待调试器的接入。[①]我们现在使用IsolationRunner来重新运行这个任务：

```
bin/hadoop org.apache.hadoop.mapred.IsolationRunner ../job.xml
```

job.xml文件中包含IsolationRunner所需的所有配置信息。鉴于本场景，JVM会在执行任务前等待调试器的接入。你可以将任意支持Java Debug Wire Protocol（JDWP）的Java调试器附加到JVM上。所有主流的Java IDE环境都是这样做的。例如，如果你正在使用jdb，你可以按如下方式将它附加到JVM上。

```
jdb -attach 8000
```

（当然，这仅仅是一个例子。我希望你在用的工具会比jdb更好！）可以通过查看你的IDE的文档来了解如何将调试器联结到JVM上。

① 用于调试的Sun JVM的配置选项详见Sun公司的文档：http://java.sun.com/javase/6/docs/technotes/guides/jpda/conninv.html#Invocation。

www.ChinaDBA.net 中国DBA超级论坛

6.3 性能调优

完成了MapReduce程序的开发和全部调试工作之后，你可能希望优化它的性能。在做任何优化之前，请注意Hadoop的主要吸引力之一是它的线性可扩展性。许多作业都可以通过添加更多机器来加速。如果你用的是一个小集群，这是一个比较节约的办法。考虑一下将程序性能提高10%所带来的时间价值。对于一个10个节点的集群，你可以通过添加一台机器来获得10%的性能收益(而且这个收益适用于该集群的所有作业)。你在开发上的时间成本很可能要高于增加计算机的成本。另外，在一个1000节点的集群中，10%的性能改进可以减少100台新机器的投入。通过简单地添加硬件的方法来提升性能，所带来的成本效益就很低了。

Hadoop在性能调优上有一些特定的手段和技巧，总体来说颇有成效。我们在下一章讨论系统管理问题时会涉及。在本节中，我们将以单个作业为基础研究其所用的技术。

6.3.1 通过 combiner 来减少网络流量

Combiner可以减少在map和reduce阶段之间洗牌的数据量，较低的网络流量缩短了执行时间。使用combiner的细节及其优点我们已经在4.6节做了充分的论述。我们在这里再次提起它是出于表述完整性的需要。

6.3.2 减少输入数据量

在处理大型数据集时，有时相当一部分处理时间都花在了扫描磁盘中的数据。减少需要读取的字节数可以提高总体的吞吐量。有几种方法可以做到这一点。

减少处理字节数的最简单方法是减少处理的数据量。我们可以选择只处理数据采样后的子集。对于某些分析应用而言，这是一个可行的选择。对于这些应用程序，抽样降低的是精度而非准确性。对于一些决策支持系统而言，它们的结果依然有用。

通常MapReduce作业不会用到输入数据集中的所有信息。回顾第4章的专利描述数据集。它几乎有几十个字段，但我们的作业大多数情况下仅访问其中几个常见的。对于该数据集中的每个作业，每次都去读取不曾用过的字段是低效的。可以“重构”输入数据为几个较小的数据集。每个仅包含数据处理所需的特定类型。精确的重构依赖于应用。这项技术的思想类似于在关系数据库管理系统(RDBMS)中垂直分区和列数据库。

最后，可以通过压缩数据来减少磁盘和网络的I/O数量。这些技术可以用于中间数据，也可以用于输出数据集。Hadoop在数据压缩上有许多选项，下一小节我们会讨论这个话题。

6.3.3 使用压缩

即使使用了combiner，在map阶段的输出也可能会很大。这些中间数据必须被存储在磁盘上，并在网络上重排。压缩这些中间数据会提高大多数MapReduce作业的性能，而且非常简单。

Hadoop内置支持压缩与解压。启用对map输出的压缩涉及对两个属性的配置，见表6-3。

www.ChinaDBA.net 中国DBA超级论坛

表6-3　控制mapper输出压缩的配置属性

属　性	描　述
mapred.compress.map.output	Boolean属性，表示mapper的输出是否应被压缩
mapred.map.output.compression.codec	Class属性，表示哪种CompressionCodec被用于压缩mapper的输出

设置mapred.compress.map.output为true就可以允许对mapper的输出进行压缩。而且，你还应在mapred.map.output.compression.codec中设置合适的codec（编/解码器）类。Hadoop中所有的编/解码器类都实现了CompressionCodec接口。Hadoop支持一些压缩编解码器（见表6-4）。例如，使用GZIP压缩，你可以设置配置对象：

```
conf.setBoolean("mapred.compress.map.output", true);
conf.setClass("mapred.map.output.compression.codec",
              GzipCodec.class,
              CompressionCodec.class);
```

你也可以使用JobConf中的便捷方法setCompressMapOutput()和setMapOutputCompressorClass()而不必直接设置这些属性。

表6-4　在软件包org.apache.hadoop.io.compress中可用的编/解码器列表

编/解码器	Hadoop版本	描　述
DefaultCodec	0.18, 0.19, 0.20	处理zlib格式的文件，这些文件采用Hadoop传统的文件扩展名.deflate
GzipCodec	0.18, 0.19, 0.20	处理gzip格式的文件，这些文件采用的文件扩展名为.gz
BZip2Codec	0.19, 0.20	处理bzip格式的文件，这些文件采用的文件扩展名为.bz2。这种压缩格式独特之处在于它可以在Hadoop上分割，甚至在不采用序列文件格式的时候也可以

来自一个作业map阶段的数据输出仅在作业的内部使用，因此允许对这个中间文件进行压缩对开发者是透明的，而且不费力气。因为许多MapReduce应用包含多个作业，所以让作业用压缩的形式输入和输出数据是有意义的。强烈推荐在Hadoop作业之间使用Hadoop专用的序列文件格式传递数据。

序列文件是用于存储键/值对的可压缩的二进制文件格式。它被设计用于支持压缩并保持可分割。记得Hadoop并行性之一是它有能力分片读取输入文件，并用多个map任务来处理。如果输入的文件采用压缩格式，Hadoop将不得不将文件拆分，并让map任务独立地解压缩每个分片。否则，如果Hadoop必须先将文件作为一个整体来解压缩，就毁掉了并行性。不是所有的压缩文件格式的设计主旨都在于分片和按块解压缩。序列文件是为支持此功能而特别开发的。这种文件格式为Hadoop提供了同步标志（sync marker）来表示可拆分的边界。[①]

除了可压缩性和可拆分性，序列文件还支持二进制的键和值。因此，序列文件常被用作处理二进制文档，如图像，而且它也可以很好地处理文本文档和其他大型的键/值对对象。每个文档

① 迄今我们看到的所有输入文件都是未压缩的文本文件，每一行为一条记录。换行符（\n）可以被简单地视为同步标志，它指出了可拆分的边界和记录边界。

www.ChinaDBA.net 中国DBA超级论坛

被视为这个序列文件中的一条记录。

通过设置MapReduce作业的输出格式为SequenceFileOutputFormat，你可以让它以序列文件的形式输出。你也许希望将默认压缩格式从RECORD改为BLOCK。采用记录压缩，每条记录被分别压缩。采用块压缩，记录中的块会被一起压缩进而得到更高的压缩率。最后，你必须调用FileOutputFormat中的静态方法setCompressOutput()和setOutputCompressorClass()（或者SequenceFileOutputFormat，它继承了这些方法）让输出的压缩使用某个特定的编/解码器。Hadoop支持的编/解码器如表6-4所示。在driver中增加如下的行：

```
conf.setOutputFormat(SequenceFileOutputFormat.class);
SequenceFileOutputFormat.setOutputCompressionType(conf,
    ➡ CompressionType.BLOCK);
FileOutputFormat.setCompressOutput(conf, true);
FileOutputFormat.setOutputCompressorClass(conf, GzipCodec.class);
```

表6-5列出了等价于与配置与序列文件输出的属性。给出如下选择后，Streaming程序输出序列文件。

```
-outputformat org.apache.hadoop.mapred.SequenceFileOutputFormat
-D mapred.output.compression.type=BLOCK
-D mapred.output.compress=true
-D mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCode
```

表6-5 输出压缩序列文件的配置属性

属 性	描 述
mapred.output.compression.type	String属性，表示序列文件的压缩类型。可以为NONE/RECORD或BLOCK中的一个。默认为RECORD但通常BLOCK压缩效果更优 **便捷方法：** SequenceFileOutputFormat. ➡ setOutputCompressionType()
mapred.output.compress	Boolean属性，表示是否压缩作业的输出 **便捷方法：** FileOutputFormat.setCompressOutput()
mapred.output.compression.codec	Class属性，用于指定用于压缩作业输出的编/解码器 **便捷方法：** FileOutputFormat. ➡ setOutputCompressorClass()

要将一个序列文件作为输入，设置输入格式为SequenceFileInputFormat。

在Streaming中使用

```
conf.setInputFormat(SequenceFileInputFormat.class);
```

或

```
-inputformat org.apache.hadoop.mapred.SequenceFileInputFormat
```

SequenceFile.Reader类（由SequenceFileRecordReader使用）会通过文件头自动地确定这些设置，因此不需要配置压缩类型或codec类。

www.ChinaDBA.net 中国DBA超级论坛

6.3.4 重用 JVM

默认情况下，TaskTracker将每个Mapper和Reducer任务作为子进程分别运行在独立的JVM中。这必然给每个任务引入JVM的启动开销。如果Mapper对自身进行初始化，例如把一个大数据结构读到内存里（参见5.2.2节联合使用分布式缓存的例子），那么这个初始化就会延缓启动过程。在每个任务运行时间很短，或者说Mapper初始化要花较长时间的情况下，启动过程就会在整个任务运行时间中占去大半时间。

Hadoop从版本0.19.0开始，允许在相同作业的多个任务之间重用JVM。因此，启动开销被平摊到多个任务中。一个新属性（mapred.job.reuse.jvm.num.tasks）指定了一个JVM可以运行的最大任务数（相同的作业）。它的默认值为1，此时JVM不能被重用。你可以增大该属性值来启用JVM重用。如果将其设置为-1，则意味着在可重复使用JVM的任务数量上没有限制。在JobConf对象中有一个便捷方法，setNumTasksToExecutePerJvm(int)，可以用它很方便地设置作业的属性。上述说明总结在表6-6中。

表6-6 允许JMV重用的配置属性

属 性	描 述
mapred.job.reuse.jvm.num.tasks	整数属性，用于设置一个JVM可以运行的最大任务数。值为-1意味着没有限制

www.ChinaDBA.net 中国DBA超级论坛

6.3.5 根据猜测执行来运行

MapReduce原始设计的一个基本假设是节点并不可靠（如谷歌的MapReduce文章所述），框架必须在作业执行过程中能够处理部分节点失效的情况。基于这个假设，最初的MapReduce框架将map任务和reduce任务均设定为等效的。这意味着当一个任务失败时，Hadoop可以重新启动该任务，而整体的作业依然会得到同样的结果。Hadoop可以监视运行节点的健康状态，并自动重启失效节点上的任务。因此容错对于开发人员是透明的。

通常节点并不会突然失效，而会经历一段降速的过程，就像输入/输出设备变坏一样。在这种情况下，系统仍在工作，但任务运行得更慢了。有时任务变慢还因为发生了临时的拥塞。这并不影响正在执行作业的正确性，但肯定会影响其性能。一个运行速度缓慢的任务甚至会延误整个MapReduce作业的完成。在所有的mapper完成之前，reducer都不会启动。类似地，在所有的reducer完成之前，一个作业也不会结束。

为了解决任务执行变慢的问题，Hadoop再次使用了幂等特性。不再仅当任务失败后才重启任务，Hadoop会注意到运行速度缓慢的任务，并安排在另一个节点上并行执行相同的任务。幂等特性保证同步执行的任务会产生相同的输出。Hadoop 将监视这些同步执行的任务。只要一个任务成功完成，Hadoop会采用它的输出，并杀死其他并行的任务。整个过程被称为猜测执行。

请注意map任务猜测执行的时机，是仅当所有的map任务已经安排执行之后，且仅对于那些比其他map任务平均执行进度远远落后的map任务。reduce任务的猜测执行与此相同。猜测执行并

www.ChinaDBA.net 中国DBA超级论坛

不是在任务的多个副本之间“赛车”来获得最佳完成时间。它仅为防止整个作业的完成时间受到较慢任务的拖累。

默认情况下，猜测执行处于启用状态。可以分别为map任务和reduce任务关闭这个设置。要做到这一点，可以设置表6-7中的一个或两个属性为false。它们仅作用于单个作业，但你也可以更改集群配置文件来改变整个集群的默认设置。

表6-7 启动和禁止猜测执行的配置属性

属 性	描 述
mapred.map.tasks.speculative.execution	布尔属性，表示是否运行map任务猜测执行
mapred.reduce.tasks.speculative.execution	布尔属性，表示是否运行reduce任务猜测执行

通常应该让猜测执行保持在启动状态。将其关闭的主要原因是：当map或者reduce任务产生副作用时，它们就不再满足等效性。例如，如果一个任务写入外部文件，猜测执行可能会导致一个任务的多个副本都试图创建相同的外部文件，从而产生冲突。你可以关闭猜测执行，以确保一次只有一个任务副本在运行。

注意 如果任务有副作用，你还要仔细考虑这些副作用与Hadoop的恢复机制之间的相互影响。例如，如果任务写入外部文件，有可能任务恰恰在写入外部文件之后死掉。在这种情况下，Hadoop重新启动的任务会再次尝试写入外部文件。你需要确保在这种情况下，任务的操作依然是正确的。

6.3.6 代码重构与算法重写

如果为了优化性能而打算重新编写MapReduce程序，有一些简单的技巧以及一些重要而且依赖于应用的重写方法，可以让事情做得更快。

对于Streaming程序，一个简单的技术是把它重写为Hadoop的Java程序。Streaming擅长于快速生成一个用于特定数据分析的MapReduce作业，但它无法取得Hadoop中Java程序的运行速度。Streaming作业的优势在于处理一次性的查询，但它不善于处理频繁执行的任务，因此将其重写为Java实现可以提高性能。

如果你有几个作业都在相同的输入数据集上运行，就有可能通过改写让作业数变少。例如，如果要计算数据集的最大值，还要计算最小值，你可以写一个单独的MapReduce作业同时计算它们，而不是用两个不同的作业分别来计算它们。这听起来是显而易见的，但在实践中，许多作业原本只是为恰当地完成一个功能而编写的。这是一个很好的设计方法，因为作业的简洁性可以让它广泛适用于在不同的数据集上的不同目的的工作。只有在运行了一段时间之后，你才应该尝试将作业结组，通过改写来加快其速度。

加速MapReduce程序的要点之一是认真考虑底层的算法，寻找是否有一种更有效的算法，可以更快地得到相同计算结果。这对于任何编程都一样，但对MapReduce程序更为重要。常规课本

www.ChinaDBA.net 中国DBA超级论坛

上的算法和数据结构（sorting、lists、maps等）涵盖了最传统的编程设计。另外，Hadoop程序则触及“独特”的领域，例如分布式计算、函数式编程、统计与数据密集型处理，大多数程序员对此了解不多，目前仍有许多令人兴奋的研究工作在探索新的途径。

我们已经见过一个利用新数据结构加快MapReduce程序的例子，即在semijoin（5.3节）中使用Bloom filter。它在分布式计算领域广为人知，而其他领域对它却不太熟悉。

使用新算法来加速MapReduce程序的另一个典型例子来自方差计算中的统计。非统计学专业的人在计算方差时可能会采用如下的公认定义：

$$(1/N) * Sum_i[(X_i - X_{avg})^2]$$

这里Sum_i代表对数据集求和。方差X_{avg}代表数据集的平均值。如果我们预先不知道平均值，非统计学专业的人可能会决定首先运行一个MapReduce作业求平均，再用第二个MapReduce作业计算方差。一些更熟悉计算统计的人会使用一个等效的定义：

$$(1/N) * Sum_i[(X_i)^2] - ((1/N) * Sum_i[X_i])^2$$

从这个定义中，我们需要求X和X^2的和，但你可以在一次数据扫描过程中将两次求和一起计算，只需用一个单独的MapReduce作业。（这类似于在单一作业中计算最大值和最小值。）一点统计背景就可以将计算方差的时间减半。[①]

你还需要注意算法的计算复杂性。Hadoop提供的“仅仅”是线性可扩展性，它无法胜任在大数据集上解二次或更高阶方程的计算密集型算法。此时，需要寻找更为有效的算法，有时你可能为了得到速度更快的算法，不得不给出近似的结果。

6.4　小结

Hadoop开发方法建立在Java编程的良好实践基础之上，如单元测试、测试驱动开发等。Hadoop在数据处理中的核心作用需要更多以数据为中心的测试流程。数学和逻辑错误在数据密集型程序中更加普遍，而它们往往并不显眼。Hadoop的分布式特性也使得调试更加困难。若要减轻负担，你应该逐个阶段地测试，从非分布式的（即本地）模式到单点伪分布模式，最后到全分布的模式。

著名计算机科学家高德纳·克努特（Donald Knuth）曾说过“过早的优化是一切罪恶的根源。”只有在程序已完全调试好之后，你才应该调节Hadoop程序的性能。改善性能不能仅靠对一般性算法和计算问题的思考，它也是与平台相关的，可以利用Hadoop的一些特定技术让作业运行得更为高效。

① 大规模数据上的数值计算带来许多麻烦的问题。在这个方差计算的例子中，我们注意到重构后MapReduce作业的数值精度更低，更可能遇到溢出的问题。

www.ChinaDBA.net 中国DBA超级论坛

第7章 细则手册

本章内容

- 向任务传递定制参数
- 获取任务特定的信息
- 生成多个输出
- 与关系数据库交互
- 让输出做全局排序

本书到此已经涵盖了编写MapReduce程序的核心技术。Hadoop是一个巨大的框架，它所支持的功能要比那些核心技术多得多。在这个必应和谷歌的时代，你可以很容易地搜索到特定的MapReduce技术，我们并不试图成为百科全书式的参考书。通过自身的使用经验以及与其他Hadoop用户的讨论，我们发现了一些通常很有用的技巧，例如可以用一个标准的关系数据库作为MapReduce作业的输入或输出的技巧。本章收集了我们最喜欢的一些“配方”。

7.1 向任务传递作业定制的参数

在编写Mapper和Reducer时，通常会想让一些地方可以配置。例如，第5章的联结程序被固定地写为取第一个数据列作为联结键。如果用户可以在运行时指定某个列作为联结键，就会让程序更具普适性。Hadoop自身使用一个配置对象来存储所有作业的配置属性。你也可以使用这个对象将参数传递到Mapper和Reducer。

我们已经知道MapReduce的driver是如何用属性来配置JobConf对象的，这些属性包括输入格式、输出格式、Mapper类等。若要引入自己的属性，需要在这个配置对象中，给属性一个唯一的名称并设置它的值。这个配置对象会被传递给所有的TaskTracker，然后作业中的所有任务就能够看到配置对象中的属性。Mapper和Reducer也就可以读取该配置对象并获得它的属性值。

Configuration类（JobConf的父类）有许多通用的setter方法。属性采用键/值对的形式，键必须是一个String，而值可以是常用类型的任意一个。常用setter方法的签名为

```
public void set(String name, String value)
public void setBoolean(String name, boolean value)
public void setInt(String name, int value)
```

www.ChinaDBA.net 中国DBA超级论坛

```
public void setLong(String name, long value)
public void setStrings(String name, String... values)
```

请注意在Hadoop内部，所有的属性都存为字符串。在set(String,String)方法之外的所有其他方法都是它的便捷方法。例如，setStrings(String, String...)方法取一个String数组，把它变成一个单一的以逗号分隔的String，并把这个String设置为属性值。同样，GetStrings()方法将这个连结好的字符串拆分后放入一个数组中。考虑到这一点，在原始数组的字符串中不要出现逗号。如果需要有逗号，则应使用自定义的字符串编码函数。

Driver会首先设置配置对象中的属性，让它们在所有任务中可见。Mapper和Reducer可以访问configure()方法中的配置对象。任务初始化时会调用configure()，它已经被覆写为可以提取和存储你设置的属性。之后，map()和reduce()方法会访问这些属性的副本。在下面的示例中，我们调用新的属性myjob.myproperty，它用一个由用户指定的整数值。

```
public int run(String[] args) throws Exception {

    Configuration conf = getConf();
    JobConf job = new JobConf(conf, MyJob.class);

    ...

    job.setInt("myjob.myproperty", Integer.parseInt(args[2]));   ◁ 设置特定属性 ❶

    JobClient.runJob(job);
    return 0;
}
```

在MapClass中，configure()方法取出属性值，并将它存储在对象的范围中。Configuration类的getter方法需要指定默认的值，如果所请求的属性未在配置对象中设置，就会返回默认值。在这个例子中，我们取默认值为0：

```
public static class MapClass extends MapReduceBase
    implements Mapper<Text, Text, Text, Text> {

    int myproperty;

    public void configure(JobConf job) {
        myproperty = job.getInt("myjob.myproperty", 0);   ◁ 得到特定属性 ❶
    }
    ...
}
```

如果你希望在Reducer中使用该属性，Reducer也必须检索这个属性。

```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, Text, Text, Text> {

    int myproperty;

    public void configure(JobConf job) {
        myproperty = job.getInt("myjob.myproperty", 0);
    }
```

www.ChinaDBA.net 中国DBA超级论坛

Configuration类中getter方法的列表比setter方法更长，但是它们大多一目了然。几乎所有的getter方法都需要将参数设置为默认值。唯一例外的是get(String)，如果没有设置特定的名称，它就返回null。

```
public String get(String name)
public String get(String name, String defaultValue)
public boolean getBoolean(String name, boolean defaultValue)
public float getFloat(String name, float defaultValue)
public int getInt(String name, int defaultValue)
public long getLong(String name, long defaultValue)
public String[] getStrings(String name, String... defaultValue)
```

既然我们的job类实现了Tool接口并使用了ToolRunner，我们还可以让用户直接使用通用的选项来配置定制化的属性，方法与用户设置Hadoop的配置属性相同。

```
bin/hadoop jar MyJob.jar MyJob -D myjob.myproperty=1 input output
```

我们可以将driver中总是需要用户通过参数来设定属性值的那行代码删掉。如果在大多数时间里默认值是可用的，这样做会让用户感觉更加方便。

```
public int run(String[] args) throws Exception {

    Configuration conf = getConf();
    JobConf job = new JobConf(conf, MyJob.class);

    ...

    Int myproperty = job.getInt("myjob.myproperty", 0);
    if (myproperty< 0) {
        System.err.println("Invalid myjob.myproperty: " + myproperty);
        System.exit(0);
    }

    JobClient.runJob(job);
    return 0;
}
```

当你允许用户设定特定属性时，在driver中最好对用户的输入进行验证。上面的例子可以确保用户设定的myjob.myproperty不是负数。

7.2 探查任务特定信息

除了获取自定义属性和全局配置之外，我们还可以使用配置对象上的getter方法获得当前任务和作业状态的一些信息。例如，在Mapper中，你可以通过map.input.file属性来得到当前map任务的文件路径。在datajoin软件包的DataJoinMapperBase中，这正是configure()方法中所做的，即用一个标签来表示数据源。

```
this.inputFile = job.get("map.input.file");
this.inputTag = generateInputTag(this.inputFile);
```

表7-1列出了一些其他的任务特定状态信息。

www.ChinaDBA.net 中国DBA超级论坛

表7-1 在配置对象中可获得的任务特定状态信息

属性	类型	描述
mapred.job.id	String	作业ID
mapred.jar	String	作业目录中jar的位置
job.local.dir	String	作业的本地空间
mapred.tip.id	String	任务ID
mapred.task.id	String	任务重试ID
mapred.task.is.map	boolean	标志量，表示是否为一个map任务
mapred.task.partition	int	作业内部的任务ID
map.input.file	String	Mapper读取的文件路径
map.input.start	long	当前Mapper输入分片的文件偏移量
map.input.length	long	当前Mapper输入分片中的字节数
mapred.work.output.dir	String	任务的工作（即临时）输出目录

配置属性还可以通过环境变量为Streaming程序所用。在执行脚本之前，Streaming的API会把所有的配置属性添加到运行环境中。属性名被重新格式化，使得非字母数字编码的字符都被下划线替代。例如，一个Streaming脚本会在环境变量map_input_file中查找当前mapper所读取文件的完整路径。

```
import os

filename = os.environ["map_input_file"]
localdir = os.environ["job_local_dir"]
```

上面的代码显示了在Python中如何访问配置属性。

7.3 划分为多个输出文件

直到现在我们看到的所有MapReduce作业都输出一组文件。然而，经常在有些场景下，输出多组文件或把一个数据集分为多个数据集更为方便。一个常见的案例将一个大的日志文件按天划分到不同的日志文件中。

MultipleOutputFormat提供了一个简单的方法，将相似的记录结组为不同的数据集。在写每条输出记录之前，这个OutputFormat类调用一个内部方法来确定要写入的文件名。更具体地说，你将扩展MultipleOutputFormat的某个特定子类，并实现generateFileNameForKeyValue()方法。你扩展的子类将决定输出的格式。例如，MultipleTextOutputFormat将输出文本文件，而MultipleSequenceFileOutputFormat将输出序列文件。无论哪种情况，你会覆写下面的方法以返回每个输出键/值对的文件名：

```
protected String generateFileNameForKeyValue(K key, V value, String name)
```

默认实现返回参数name，即叶文件名。你可以让该方法根据记录的内容返回文件名。

在此示例中，我们取专利的元数据并按国家对它进行分区。来自美国发明者的所有专利将都进入一组文件，而来自日本的所有专利进入另一个，以此类推。该示例程序的框架是一个map-only的作业，取得输入后立即将其输出。我们做的主要变化是创建自己的MultipleTextOutputFormat

www.ChinaDBA.net 中国DBA超级论坛

子类，名为PartitionbyCountryMTOF。（请注意MTOF是MultipleTextOutputFormat的缩写。）我们的子类将基于该记录列出的发明专利所属国家，将每条记录存储到相应的位置。因为我们将generateFileNameForKeyValue()的返回值作为文件路径，通过返回的country + "/" + filename，我们可以为每个国家创建一个子目录。请参阅代码清单7-1。

代码清单7-1 根据国家将专利元数据分割到多个目录中

```
public class MultiFile extends Configured implements Tool {

  public static class MapClass extends MapReduceBase
      implements Mapper<LongWritable, Text, NullWritable, Text> {

      public void map(LongWritable key, Text value,
                      OutputCollector<NullWritable, Text> output,
                      Reporter reporter) throws IOException {

          output.collect(NullWritable.get(), value);
      }
  }

  public static class PartitionByCountryMTOF
      extends MultipleTextOutputFormat<NullWritable,Text>
  {
      protected String generateFileNameForKeyValue(NullWritable key,
                                                   Text value,
                                                   String filename)
      {
          String[] arr = value.toString().split(",", -1);
          String country = arr[4].substring(1,3);
          return country + "/" + filename;
      }
  }

  public int run(String[] args) throws Exception {
      Configuration conf = getConf();

      JobConf job = new JobConf(conf, MultiFile.class);

      Path in = new Path(args[0]);
      Path out = new Path(args[1]);
      FileInputFormat.setInputPaths(job, in);
      FileOutputFormat.setOutputPath(job, out);

      job.setJobName("MultiFile");
      job.setMapperClass(MapClass.class);

      job.setInputFormat(TextInputFormat.class);
      job.setOutputFormat(PartitionByCountryMTOF.class);
      job.setOutputKeyClass(NullWritable.class);
      job.setOutputValueClass(Text.class);

      job.setNumReduceTasks(0);

      JobClient.runJob(job);

      return 0;
```

www.ChinaDBA.net 中国DBA超级论坛

```
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(),
                                 new MultiFile(),
                                 args);
        System.exit(res);
    }
}
```

在上述程序执行以后，我们会在输出目录中看到，现在每个国家都有一个单独的目录。

```
ls output/
AD    BN    CS    GE    IN    LC    MT    PH    SV    VE
AE    BO    CU    GF    IQ    LI    MU    PK    SY    VG
AG    BR    CY    GH    IR    LK    MW    PL    SZ    VN
AI    BS    CZ    GL    IS    LR    MX    PT    TC    VU
AL    BY    DE    GN    IT    LT    MY    PY    TD    YE
AM    BZ    DK    GP    JM    LU    NC    RO    TH    YU
AN    CA    DO    GR    JO    LV    NF    RU    TN    ZA
AR    CC    DZ    GT    JP    LY    NG    SA    TR    ZM
AT    CD    EC    GY    KE    MA    NI    SD    TT    ZW
AU    CH    EE    HK    KG    MC    NL    SE    TW
AW    CI    EG    HN    KN    MG    NO    SG    TZ
AZ    CK    ES    HR    KP    MH    NZ    SI    UA
BB    CL    ET    HT    KR    ML    OM    SK    UG
BE    CM    FI    HU    KW    MM    PA    SM    US
BG    CN    FO    ID    KY    MO    PE    SN    UY
BH    CO    FR    IE    KZ    MQ    PF    SR    UZ
BM    CR    GB    IL    LB    MR    PG    SU    VC
```

而且在每个国家的目录中的文件都是仅由这些国家所创建的记录（专利）。

```
ls output/AD
part-00003     part-00005     part-00006

head output/AD/part-00006
5765303,1998,14046,1996,"AD","",,1,12,42,5,59,11,1,0.4545,0,0,1,67.3636,,,,
5785566,1998,14088,1996,"AD","",,1,9,441,6,69,3,0,1,,0.6667,,4.3333,,,,
5894770,1999,14354,1997,"AD","",,1,,82,5,51,4,0,1,,0.625,,7.5,,,,
```

我们编写的这个简单的分区练习是一个map-only程序。你也可以对reducer的输出应用相同的技术。请注意不要与MapReduce框架中的partitioner混淆。partitioner查看中间记录的键，并决定哪些Reducer将处理它们。这里我们要做的分割是查看输出的键/值对，并决定存储到哪个文件中。

MultipleOutputFormat很简单，但也有局限。例如，我们可以按行拆分输入数据，但如果我们想按列拆分会该怎么做？假设我们要从专利的元数据中创建两个数据集：一个包含每个专利的时间相关信息（例如发布日期），另一种包含地理信息（例如发明的国家）。这两个数据集可能有不同的输出格式以及不同数据类型的键和值。我们可以用在Hadoop 0.19版本中引入的MultipleOutputs，以获得更强的能力。

MultipleOutputs所采用的方法不同于MultipleOutputFormat。它不是要求给每条记录请求文件名，而是创建多个OutputCollector。每个OutputCollector可以有自己的OutputFormat和键/值对的类型。MapReduce程序将决定如何向每个OutputCollector输出数据。代码清单7-2

www.ChinaDBA.net 中国DBA超级论坛

显示了一个程序，取出专利的元数据，并输出两个数据集。一个包含时间顺序的信息，例如发布日期。另一个数据集包含与每个专利相关的地理信息。它也是个map-only程序，但你可以直接在reducer上使用多个输出收集器。

代码清单7-2 将输入数据的不同列提取为不同文件的程序

```
public class MultiFile extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<LongWritable, Text, NullWritable, Text> {

        private MultipleOutputs mos;
        private OutputCollector<NullWritable, Text> collector;

        public void configure(JobConf conf) {
            mos = new MultipleOutputs(conf);
        }

        public void map(LongWritable key, Text value,
                        OutputCollector<NullWritable, Text> output,
                        Reporter reporter) throws IOException {

            String[] arr   = value.toString().split(",", -1);
            String chrono = arr[0] + "," + arr[1] + "," + arr[2];
            String geo    = arr[0] + "," + arr[4] + "," + arr[5];

            collector = mos.getCollector("chrono", reporter);
            collector.collect(NullWritable.get(), new Text(chrono));
            collector = mos.getCollector("geo", reporter);
            collector.collect(NullWritable.get(), new Text(geo));
        }

        public void close() throws IOException {
            mos.close();
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = getConf();

        JobConf job = new JobConf(conf, MultiFile.class);

        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);

        job.setJobName("MultiFile");
        job.setMapperClass(MapClass.class);

        job.setInputFormat(TextInputFormat.class);
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setNumReduceTasks(0);

        MultipleOutputs.addNamedOutput(job,
                                       "chrono",
                                       TextOutputFormat.class,
```

www.ChinaDBA.net 中国DBA超级论坛

```
                                        NullWritable.class,
                                        Text.class);
    MultipleOutputs.addNamedOutput(job,
                                        "geo",
                                        TextOutputFormat.class,
                                        NullWritable.class,
                                        Text.class);

    JobClient.runJob(job);

    return 0;
}

public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(),
                             new MultiFile(),
                             args);

    System.exit(res);
  }
}
```

若要使用MultipleOutputs，MapReduce程序的driver必须设置它想要使用的输出收集器。创建收集器涉及调用MultipleOutputs的静态方法addNamedOutput()。我们已经创建了两个输出收集器，一个称为chrono和另一个称为geo。它们都使用TextOutputFormat，并具有相同的键/值类型，但我们也可以选择使用不同的输出格式或数据类型。

设置完driver中的输出收集器后，我们需要获取MultipleOutputs对象，当configure()方法中的mapper被初始化时，它会跟踪这些输出收集器。该对象必须在整个map任务的生命期中可用。在map()函数自身，我们可以在MultipleOutputs对象中调用getCollector()方法取回chrono和geo的OutputCollectors。我们将不同的数据输出到每个合适的输出收集器中。

我们已经为MultipleOutputs中的每个输出收集器赋予了一个名字，而MultipleOutputs会自动生成输出文件名。我们可以通过脚本来查看这些文件，看看MultipleOutputs是如何生成输出文件名的：

```
ls -l output/
total 101896
-rwxrwxrwx 1 Administrator None 9672703 Jul 31 06:28 chrono-m-00000
-rwxrwxrwx 1 Administrator None 7752888 Jul 31 06:29 chrono-m-00001
-rwxrwxrwx 1 Administrator None 6884496 Jul 31 06:29 chrono-m-00002
-rwxrwxrwx 1 Administrator None 6933561 Jul 31 06:29 chrono-m-00003
-rwxrwxrwx 1 Administrator None 7164558 Jul 31 06:29 chrono-m-00004
-rwxrwxrwx 1 Administrator None 7273561 Jul 31 06:29 chrono-m-00005
-rwxrwxrwx 1 Administrator None 8281663 Jul 31 06:29 chrono-m-00006
-rwxrwxrwx 1 Administrator None 9428951 Jul 31 06:28 geo-m-00000
-rwxrwxrwx 1 Administrator None 7464690 Jul 31 06:29 geo-m-00001
-rwxrwxrwx 1 Administrator None 6580482 Jul 31 06:29 geo-m-00002
-rwxrwxrwx 1 Administrator None 6448648 Jul 31 06:29 geo-m-00003
-rwxrwxrwx 1 Administrator None 6432392 Jul 31 06:29 geo-m-00004
-rwxrwxrwx 1 Administrator None 6546828 Jul 31 06:29 geo-m-00005
-rwxrwxrwx 1 Administrator None 7450768 Jul 31 06:29 geo-m-00006
-rwxrwxrwx 1 Administrator None       0 Jul 31 06:28 part-00000
-rwxrwxrwx 1 Administrator None       0 Jul 31 06:28 part-00001
```

www.ChinaDBA.net 中国DBA超级论坛

```
-rwxrwxrwx 1 Administrator None      0 Jul 31 06:29 part-00002
-rwxrwxrwx 1 Administrator None      0 Jul 31 06:29 part-00003
-rwxrwxrwx 1 Administrator None      0 Jul 31 06:29 part-00004
-rwxrwxrwx 1 Administrator None      0 Jul 31 06:29 part-00005
-rwxrwxrwx 1 Administrator None      0 Jul 31 06:29 part-00006
```

我们有一组以chrono为前缀的文件，以及另一组以geo为前缀的文件。请注意该程序创建了默认的输出文件part-*，即使它没有显示地写任何东西。这些文件是完全有可能通过map()方法中的原始OutputCollector来写的。事实上，如果这不是一个map-only的程序，只有那些被写到原始OutputCollector中的记录才会被传递到reducer上加工。

使用MultipleOutputs需要权衡的一个地方是它有一个比MultipleOutputFormat更为严格的命名结构。输出收集器的名称不能为part，因为它已经被用作默认值。输出文件名还严格定义为在输出收集器的名称之后，要跟m或r，这取决于是在Mapper还是在Reducer上收集输出。最后，还要跟一个分区号。

```
head output/chrono-m-00000
"PATENT","GYEAR","GDATE"
3070801,1963,1096
3070802,1963,1096
3070803,1963,1096
3070804,1963,1096
3070805,1963,1096
3070806,1963,1096
3070807,1963,1096
3070808,1963,1096
3070809,1963,1096

head output/geo-m-00000
"PATENT","COUNTRY","POSTATE"
3070801,"BE",""
3070802,"US","TX"
3070803,"US","IL"
3070804,"US","OH"
3070805,"US","CA"
3070806,"US","PA"
3070807,"US","OH"
3070808,"US","IA"
3070809,"US","AZ"
```

查看输出文件，可以看到我们已经成功地将专利数据集中的列抽取到不同的文件中。

7.4 以数据库作为输入输出

虽然Hadoop善于处理较大数据，但在许多数据处理应用中，关系数据库是仍然是主力。很多时候Hadoop会需要与数据库进行接口。

虽然有可能建立一个MapReduce程序通过直接查询数据库来取得输入数据，而不是从HDFS中读取文件，但其性能不甚理想。更多时候，你需要将数据集从数据库复制到HDFS中。你可以很容易地通过标准的数据库工具dump，来取得一个flat文件。然后使用HDFS的shell命令put将它上传到HDFS中。

www.ChinaDBA.net 中国DBA超级论坛

但是有时更合理的做法是让MapReduce程序直接写入数据库。许多MapReduce程序获取大型数据集，并把它们处理到数据库可管理的大小。例如，我们经常在ETL之类的过程中用MapReduce获取堆积如山的日志文件，并计算出一个更小、更易于管理的统计数据集供分析者查看。

DBOutputFormat是用于访问数据库的关键类。在driver中，你可以将输出格式设置为这个类。你需要指定配置，以便能够联结到该数据库。你可以通过在DBConfiguration中的静态方法configureDB()做到这一点：

```
public static void configureDB(JobConf job, String driverClass,
    ➥ String dbUrl, String userName, String passwd)
```

之后，你要指定将要写入的表，以及那里有哪些字段。这是通过在DBOutputFormat中的静态setOutput()方法做到的。

```
public static void setOutput(JobConf job, String tableName,
    ➥ String... fieldNames)
```

你的driver应该包含如下样式的几行代码：

```
conf.setOutputFormat(DBOutputFormat.class);
DBConfiguration.configureDB(job,
                            "com.mysql.jdbc.Driver",
                            "jdbc:mysql://db.host.com/mydb",
                            "username",
                            "password")
DBOutputFormat.setOutput(job, "Events", "event_id", "time");
```

使用DBOutputFormat将强制你输出的键实现DBWritable接口。只有这个键会被写入到数据库中。通常，键必须实现Writable接口。Writable和DBWritable的签名类似；只是它们的参数类型是不同的。在Writable中的write()方法用DataOutput，而DBWritable中的write()用PreparedStatement。类似地，用在Writable中的readFields()方法采用DataInput，而DBWritable中的readFields()采用ResultSet。除非你打算使用DBInputFormat直接从数据库中读取输入的数据，在DBWritable中的readFields()将永远不会被调用。

```
public class EventsDBWritable implements Writable, DBWritable {
    private int id;
    private long timestamp;

    public void write(DataOutput out) throws IOException {
        out.writeInt(id);
        out.writeLong(timestamp);
    }

    public void readFields(DataInput in) throws IOException {
        id = in.readInt();
        timestamp = in.readLong();
    }

    public void write(PreparedStatement statement) throws SQLException {
        statement.setInt(1, id);
        statement.setLong(2, timestamp);
    }
```

www.ChinaDBA.net 中国DBA超级论坛

```
    public void readFields(ResultSet resultSet) throws SQLException {
        id = resultSet.getInt(1);
        timestamp = resultSet.getLong(2);
    }
}
```

我们想再次强调的是，从Hadoop内部读写数据库仅适用于凭借Hadoop的标准来说相对较小的数据集。除非你的数据库是与Hadoop并行的（如果你Hadoop集群相对较小而你的数据库系统中有许多分片，就可以这样设置），否则，你的数据库将成为性能瓶颈，而无法利用Hadoop集群的可扩展性优势。很多时候，最好批量地将数据装载到数据库，而不是从Hadoop上直接写入。对于非常大的数据库，你需要采用定制化的解决方案。[①]

7.5 保持输出的顺序

MapReduce框架保证每个reducer的输入都按键来排序。在许多情况下，reducer只是对键/值对中值的部分做简单的计算。输出仍保持顺序排序。请记住MapReduce框架并不能保证reducer输出的顺序。它只是已经排序好的输入以及reducer所执行的典型操作类型的一种副产品。

对于某些应用，这种排序是没有必要的，有时就会有疑问，是否可以关闭排序操作来消除reducer中不必要的步骤。事实上，排序操作的目的并不是强制对reducer的输入进行排序。相反，排序是将相同键的所有记录进行组合的一种有效途径。如果分组功能是不必要的，那么我们就可以直接从单个输入记录中生成输出记录。在这种情况下，你就可以消除整个reduce阶段来提高性能。你可以将reducer的个数设置为0，从而让应用程序成为一个map-only作业。

另外，对于某些应用程序，很好将所有的输出做整体排序。（由一个reducer生成的）每个输出文件都已经排好序了；很好的结果是所有`part-00000`的记录都小于`part-00001`，而`part-00001`又都小于`part-00002`，以此类推。要做到这点，关键还在于框架中的partitioner操作。

Partitioner的任务是确定地为每个键分配一个reducer。相同键的所有记录都结成组并在reduce阶段被集中处理。Partitioner的一个重要设计需求是在reducer之间达到负载平衡；没有一个reducer会比其他的reducer被分配更多的键。由于没有以前对键的分配信息，partitioner默认使用散列函数来均匀地将键分配给reducer。这通常可以很好地在reducer之间均匀地分配工作，但分配是完全随意的，而不存在任何顺序。如果事先知道键是大致均匀分布的，我们就可以使用一个partitioner给每个reducer分配一个键的范围，仍然可以确保reducer的负载是相对均衡的。

注意 如果某些键的处理比其他键要占用更多的时间，散列分区可能也无法均匀分布作业。例如，在高度倾斜的数据集中，大量的记录可能有相同的键。如果可能的话，你应该使用combiner在map阶段尽可能多地做预处理，来减轻reduce阶段的负载。此外，你还可以选择写一个特殊的partitioner来非均匀地分配键，使其可以平衡这个倾斜的数据及其处理。

① LinkedIn有一个有趣的博客，讨论了将离线处理（即Hadoop）的海量数据结果移动到在线系统所面临的挑战：http://project-voldemort.com/blog/2009/06/building-a-1-tb-per-day-data-platform-with-hadoop-and-project-voldemort/。

www.ChinaDBA.net 中国DBA超级论坛

TotalOrderPartitioner是一个可以保证在输入分区之间，而不仅仅是分区内部排序的partitioner。大规模数据的排序（即TeraSort基准测试）最初使用的就是这个类的一个类似版本。这个类利用一个排好序的分区键组读取一个序列文件，并进一步将不同区域的键分配到reducer上。

7.6 小结

本章讨论了许多工具和技术来让你的Hadoop作业对用户更友好，或使它可以更好地与数据处理平台中的其他组件接口。一个Hadoop作业可获得的全部支持在Hadoop的API中有详细的描述：http://hadoop.apache.org/common/docs/current/api/index.html。你还可以使用更多的抽象概念（如pig和Hive）以简化你的编程。我们会在第10章和第11章介绍这些工具。

如果你的角色涉及管理一个Hadoop集群，你会在下一章发现管理Hadoop集群的那些有用的技巧。

www.ChinaDBA.net 中国DBA超级论坛

第8章

管理Hadoop

8

本章内容

- 生产系统的配置
- 维护HDFS文件系统
- 建立作业调度器

遵照第2章中的安装说明可以很快建立一个运行的Hadoop集群。这种配置较为简单，但是它不能胜任生产集群中持久而繁重的工作。有许多不同的配置参数可用于生产集群的调优，8.1节会介绍这些参数。

此外，任何系统都会随时间发生变化，Hadoop集群也一样。而你（或某些管理员）将不得不了解如何维护它，使之保持良好的运行状态。HDFS文件系统尤其如此。从8.2节到8.5节，我们将介绍各种标准的文件系统维护任务，如健康检查、权限设置、配额管理以及已删除文件（回收站）的恢复。从8.6节到8.10节，我们将讨论更复杂也是更为少见的管理任务，且更专注于HDFS。其中包括添加/删除节点（容量）和NameNode的故障恢复。本章最后一节将讨论调度器的建立，用以管理多个运行中的作业。

8.1 为实际应用设置特定参数值

Hadoop有各种各样的参数。它们的默认值通常仅针对于单机模式，而且致力于变得让用户使用更为方便。设置默认值时要让它们能够适用于更多的系统，且不会导致任何错误。然而，大多时候它们和生产集群对优化的需求相去甚远。表8-1显示了配置生产集群所需修改的系统属性。

表8-1　可用于生产集群调优的Hadoop属性

属　性	描　述	推 荐 值
dfs.name.dir	在NameNode的本地文件系统中，用于存储HDFS元数据的目录	/home/hadoop/dfs/name
dfs.data.dir	在DataNode的本地文件系统中，用于存储HDFS文件块的目录	/home/hadoop/dfs/data
mapred.system.dir	在HDFS中用于存储共享的MapReduce系统文件的目录	/hadoop/mapred/system

www.ChinaDBA.net 中国DBA超级论坛

（续）

属　性	描　述	推荐值
mapred.local.dir	在TaskNode的本地文件系统中，用于存储临时数据的目录	
mapred.tasktracker.{map|reduce}.tasks.maximum	在一个TaskTracker上可同时运行的map和reduce任务的最大值	
hadoop.tmp.dir	Hadoop临时目录	/home/hadoop/tmp
dfs.datanode.du.reserved	DataNode应具备的最小空闲空间	1073741824
mapred.child.java.opts	分配给每个子任务的堆栈大小	-Xmx512m
mapred.reduce.tasks	一个作业的reduce任务个数	

dfs.name.dir和dfs.data.dir默认值指向/tmp目录，这个目录在几乎所有的Unix系统中仅用来存储临时的数据。在生产集群中，这些属性必须做修改。[①] 此外，这些属性可以被设为一个以逗号分隔的目录列表。对于dfs.name.dir，多个目录可以更好地实现备份。如果DataNode有多个磁盘驱动器，应该在每一个磁盘上建立一个数据目录，并全部列在dfs.data.dir中。DataNode将通过并行访问来提高I/O性能。[②]你还应该把来自多个磁盘的目录列表指向mapred.local.dir，以加快临时数据的处理。

hadoop.tmp.dir为Hadoop的临时目录，其默认配置依赖于用户名。因为提交作业的用户名会和启动Hadoop节点的用户名可能并不一致，你应该避免任何Hadoop属性与用户名相关。应将其设置为/home/hadoop/tmp这样的目录，以独立于任何用户名。使用hadoop.tmp.dir默认值的另一个问题是它指向/tmp目录。虽然这个位置适用于临时存储数据，但是Linux的大部分默认配置给/tmp的配额对Hadoop而言太小了。如果不去增加/tmp的配额，最好让hadoop.tmp.dir指向一个已知有很多空间的目录。

默认配置下，HDFS不需要在DataNode上预留可用空间。但在实践中，大多数系统在可用空间太低时稳定性就会出问题。你应通过设置dfs.datanode.du.reserved在DataNode中预留1 GB的可用空间。当空闲的空间低于预留量时，DataNode会停止接受数据块的写入。

可以为每个TaskTracker配置最多被允许运行的map和Reduce任务数。Hadoop的默认值为4个任务（2个map任务和2个reduce任务）。合适的数量取决于许多因素，虽然大多数情况下每个核仅会调用1到2个任务。你可以设置一个四核机器的map和reduce任务数最多为6个（每种各3个），因为TaskTracker和DataNode已经分别有一个任务，总计为8个。同样，你可以设置双四核机器的map和reduce任务最多为14，不过这是基于大多数MapReduce作业有很重的I/O负载。如果你希望负载为CPU密集的，就应该减少允许的最大任务数。

---

① /tmp的使用印证了默认值是为操作简单而设的。每个Unix系统都有/tmp目录，这样就不会遇到“目录找不到”的错误了。

② 在Hadoop论坛上曾讨论过是否应将DataNode上的多个硬盘驱动器配置为RAID或JBOD。Hadoop并不需要RAID的数据冗余，因为HDFS已经在计算机之间复制了数据。此外，雅虎已表示它们能够使用JBOD获得明显的性能改进。因为，即使是相同型号的硬盘，它们的速度也有很大差异。RAID配置会使I/O性能受限于最慢的驱动器。另外，让每个驱动器的功能独立，将允许其以最高速度运行，使系统的整体吞吐量更高。

www.ChinaDBA.net 中国DBA超级论坛

在考虑允许的任务数时，还应考虑分配给每个任务的堆栈大小。Hadoop默认给每个任务200 MB是远远不够的。很多配置将它提升为512 MB，有些甚至在1 GB。这并非一个终极属性。每个作业给一个任务所请求的堆栈空间都可以或多或少。你需要确保在机器上有足够的可用内存适应这些配置参数。请记住DataNode和TaskTracker每个都已经占用了1 GB的RAM。

虽然你可以在每个单独的MapReduce作业中设置reduce任务的数量，最行之有效的办法还是让默认值能够适应大部分时间的工作。Hadoop默认一个作业一个reduce任务，在大多数情况下当然不够理想。通常建议把默认值设为集群中运行reduce的TaskTracker的最大值的0.95倍或1.75倍。这意味着在一个作业中reduce任务的数量应等于0.95或1.75乘以工作节点数，再乘以mapred.tasktracker.reduce.tasks.maximum。因子为0.95时会让所有的reduce任务立即装载，并当map任务结束时复制它们的输出结果。当因子为1.75时，一些reduce任务会被立即装载，而其他一些则会等待。更快的节点会较早地完成第一轮reduce任务，并开始第二轮。最慢的节点在第二轮不需要处理任何reduce任务。这可以带来更好的负载平衡。

8.2 系统体检

Hadoop提供的文件系统检查工具叫做fsck。如参数为文件路径时，它会递归地检查该路径下所有文件的健康状态。如果参数为/，它就会检查整个文件系统。如下为一个输出的示例：

```
bin/hadoop fsck /
Status: HEALTHY
 Total size:    143106109768 B
 Total dirs:    9726
 Total files:   41532
 Total blocks (validated):      42419 (avg. block size 3373632 B)
 Minimally replicated blocks:   42419 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          8
 Number of racks:               1
```

大多数信息是不言而喻的。默认情况下，fsck会忽略正在被客户端写入而打开的文件。如果你想看到这些文件的列表，可以在fsck使用-openforwrite参数。

在fsck检查文件系统时，它每发现一个健康的文件就会打印出一个圆点（未在上面的输出中显示）。遇到不健康的文件时，它会打出相应信息，包括过度复制的块、复制不足的块、未复制的块、损坏的块和失踪的副本。因为HDFS是自我修复的，所以过度复制的块、复制不足的块、未复制的块都不足为虑。但是，损坏的块和失踪的副本意味着数据已永久丢失。默认情况下，fsck对损坏的文件什么也不做，但你可以运行fsck的-delete选项将其删除。但更好的方式是用-move选项运行fsck，它会把已损坏的文件移动到/lost+found目录中备用。

www.ChinaDBA.net 中国DBA超级论坛

你还可以在fsck中加上-files、-blocks、-locations和-racks选项，以打印出更多的信息。每个后续的选项需要前面选项的存在。-blocks需要-files；-locations既要有-files，也要有-blocks，以此类推。-files选项让fsck每检查一个文件便打印一行信息，包含文件路径、文件字节数与块个数，以及文件的状态。-blocks选项进一步让fsck为文件中的每个块打印出一行信息。该行包括块名、长度及其副本数。-locations选项会让每一行包含块的副本位置。-racks选项则会在位置信息中增加机架名。例如，一个小到仅有一个数据块的文件会给出如下的报告

```
bin/hadoop fsck /user/hadoop/test -files -blocks -locations -racks
/user/hadoop/test/part-00000 35792 bytes, 1 block(s):  OK
0. blk_-4630072455652803568_97605 len=35792 repl=3
➥ [/default-rack/10.130.164.71:50010, /default-rack/10.130.164.177:50010,
➥ /default-rack/10.130.164.186:50010]

Status: HEALTHY
 Total size:    35792 B
 Total dirs:    0
 Total files:   1
 Total blocks (validated):      1 (avg. block size 35792 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          8
 Number of racks:               1
```

虽然fsck会给出HDFS中每个文件的报告，但获知DataNode的情况却需要用dfsadmin命令。你可以使用dfsadmin命令中的-report选项：

```
bin/hadoop dfsadmin -report
Total raw bytes: 535472824320 (498.7 GB)
Remaining raw bytes: 33927731366 (31.6 GB)
Used raw bytes: 379948188541 (353.85 GB)
% used: 70.96%

Total effective bytes: 0 (0 KB)
Effective replication multiplier: Infinity
-------------------------------------------------
Datanodes available: 8

Name: 123.45.67.89:50010
State          : In Service
Total raw bytes: 76669841408 (71.4 GB)
Remaining raw bytes: 2184594843(2.03 GB)
Used raw bytes: 56598956650 (52.71 GB)
% used: 73.82%
Last contact: Sun Jun 21 16:13:32 PDT 2009
Name: 123.45.67.90:50010
State          : In Service
```

www.ChinaDBA.net 中国DBA超级论坛

```
Total raw bytes: 76669841408 (71.4 GB)
Remaining raw bytes: 6356175381(5.92 GB)
Used raw bytes: 54220537856 (50.5 GB)
% used: 70.72%
Last contact: Sun Jun 21 16:13:33 PDT 2009

Name: 123.45.67.91:50010
State          : In Service
Total raw bytes: 76669841408 (71.4 GB)
Remaining raw bytes: 6106387206(5.69 GB)
Used raw bytes: 52412190091 (48.81 GB)
% used: 68.36%
Last contact: Sun Jun 21 16:13:33 PDT 2009

...
```

若要看NameNode的当前活动状态，可以在dfsadmin中使用-metasave选项：

```
bin/hadoop dfsadmin -metasave filename
```

这会将一部分NameNode的元数据保存到日志目录下filename文件中。在这些元数据中，你会发现待复制的块、正在复制的块以及待删除块的列表。每个块也会有一个复制列表，指向它被复制到的DataNode。最后，这个文件还会给出每个DataNode的统计摘要。

8.3 权限设置

HDFS采用类似于POSIX语义的基本文件权限管理系统。每个文件有9种权限设置：与每个文件关联的所有者组和其他用户分别有读(r)、写(w)和执行(x)权限。不是所有的权限设置都有意义。在HDFS中我们不能执行文件，所以不设置文件的x权限。

目录的权限设置也严格遵循POSIX语义。权限r允许列出目录。权限w允许创建或删除文件或目录。权限x允许访问子目录。

当前版本的HDFS没有太多安全上的考虑。使用HDFS的权限管理系统只是为了防止在共享Hadoop集群的受信任用户之间发生数据的意外误用和覆盖。HDFS不会对用户进行身份验证，并相信用户就是主机操作系统所声称的身份。Hadoop的用户名就是登录名，等同于whoami所示的用户。组列表与bash -c groups所列出的一致。启动name node的用户名是一个例外。该用户名有一个特别的Hadoop用户名superuser。这个超级用户可以执行任何文件操作，不受权限设置的约束。此外，管理员可以在一个超级用户组中通过配置参数dfs.permissions.supergroup来指定成员。所有超级用户组的成员也都是超级用户。

更改权限设置和所有权可以使用bin/hadoop fs -chmod、-chown以及-chgrp。使用方式与Unix中对应的命令类似。

8.4 配额管理

默认情况下，HDFS 不设任何配额来限制一个目录中可以放多少内容。你可以在指定目录上启用和指定所谓的名字配额，限制放在该目录下的文件名和目录名的个数。其主要用途是防止用户生成过多的小文件，令NameNode负担过重。以下命令用于设置和清除名字配额：

www.ChinaDBA.net 中国DBA超级论坛

```
bin/hadoop dfsadmin -setQuota <N> directory [...directory]
bin/hadoop dfsadmin -clrQuota directory [...directory]
```

HDFS从0.19版开始，还支持对每个目录做空间配额。它有助于管理一个用户或应用程序可占用的存储量。

```
bin/hadoop dfsadmin -setSpaceQuota <N> directory [...directory]
bin/hadoop dfsadmin -clrSpaceQuota directory [...directory]
```

在SetSpaceQuota命令中，代表每个目录配额的参数以字节为单位。这个参数还可以用后缀来表示单位。例如，20 g将代表20 GB，而5 t则代表5 TB。所有的副本都计入配额。

若要获取目录的配额，以及它使用的名字个数和字节数，可使用带有-q选项的HDFS shell命令count。

```
bin/hadoop fs -count -q directory [...directory]
```

8.5 启用回收站

除了文件权限之外，还有一个保护机制可以防止在HDFS上意外删除文件，这就是回收站。默认情况下该功能被禁用。当它启用后，用于删除文件的命令行程序不会立即删除文件。相反，它们暂时把文件移动到用户工作目录下的.Trash/文件夹下。在用户设置的时间延迟到来之前，它都不会被永久删除。只要文件仍在.Trash/文件夹下，你就可以通过将它移回到原来位置的方法来恢复它。

若要启用回收站功能并设置清空回收站的时间延迟，可以通过设置core-site.xml的fs.trash.interval属性（以分钟为单位）。例如，如果你希望用户有24个小时（1440分钟）的时间来还原已删除的文件，应该在core-site.xml中设置

```
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
</property>
```

如将该值设置为0，则将禁用回收站功能。

8.6 删减 DataNode

有时你会从HDFS集群中删除DataNode。例如离线升级或者维护一台机器。从Hadoop中删除节点是非常简单的。虽然不推荐这样做，但你的确可以杀死节点或将之从集群中断开。HDFS的设计非常有弹性。让一两个DataNode离线不会影响操作的正常运行。NameNode会检测到节点的死亡，并开始复制那些低于约定副本数的数据块。为了让操作更为顺畅和安全，特别当删减大批DataNode时，你应该使用Hadoop的退役（decommissioning）功能。该功能确保所有块在剩余的活动节点上仍达到所需的副本数。要使用此功能，你必须在NameNode的本地文件系统上生成一个排除文件（最初是空的），并让参数dfs.hosts.exclude在NameNode的启动过程中指向该文件。当你想删减DataNode时，把它们列在排除文件中，每行列一个节点。还必须用完整的主机名、IP或IP:port的格式来指定节点。执行

```
bin/hadoop dfsadmin -refreshNodes
```

www.ChinaDBA.net 中国DBA超级论坛

来强制Name Node重新读取排除文件，并开始退役过程。当此过程结束后，NameNode的日志文件中会出现像“Decommission complete for node 172.16.1.55:50010”这样的消息，此时你就可以将节点从集群中移出。

如果你在启动HDFS时没有让dfs.hosts.exclude指向排除文件，退役DataNode的正确方法是：关闭NameNode。设置dfs.hosts.exclude指向一个空的排除文件。重新启动NameNode。在NameNode成功重启后，就可以按照上面的步骤操作。请注意如果你在重启NameNode之前在排除文件中列出了需要删减的DataNode，那么NameNode就会混淆，而在日志中引发“ProcessReport from unregistered node: node055:50010”这样的消息。NameNode会认为它接触到的是系统之外的DataNode，而不是即将去除的节点。

如果退役的机器在后来的某个时刻还会重新加入集群，你应该在外部文件中删除它们，并重新执行bin/hadoop dfsadmin -refreshNodes来更新NameNode。当机器已准备好重新加入集群时，你可以按照下一节中介绍的步骤来添加它们。

8.7 增加 DataNode

除了让离线维护的机器重新上线，你可能还会在Hadoop集群中增加DataNode，以便有更多的作业来处理更多的数据。采用与集群中所有DataNode都一样的方式在新节点上安装Hadoop并设置配置文件。手动启动DataNode的守护进程（bin/hadoop datanode）。它会自动联系NameNode并加入集群。你还应把新节点添加到主服务器的conf/slaves文件中。脚本命令会识别到这个新节点。

当你添加一个新的DataNode时，它最初会是空的，然而早先的DataNode已经存了一些内容。这时，文件系统被认为是不平衡的。新的文件将有可能进入新节点，但其副本块仍会进入先前的节点。我们应该主动地启动HDFS平衡器来获得最优性能。平衡器的运行脚本为：

```
bin/start-balancer.sh
```

该脚本将在后台运行，直到集群达到平衡为止。管理员还可以提前终止它，即运行：

```
bin/stop-balancer.sh
```

当所有DataNode的利用率处于平均利用率加减一个阈值的范围内时，集群就被认为是平衡的。当启动一个平衡器脚本时，可以指定一个不同的阈值。默认情况下的阈值为10%，当启动平衡器脚本时，也可以指定一个与此不同的阈值。例如，要设置阈值为5%以便让集群达到更优的均匀分布，需这样启动平衡器

```
bin/start-balancer.sh -threshold 5
```

因为均衡操作会占用网络资源，我们建议在晚上或者周末做，此时集群可能不会太忙。或者，你可以设置dfs.balance.bandwidthPerSec参数，以限制用于做均衡的带宽。

8.8 管理 NameNode 和 SNN

NameNode是HDFS体系结构中最为重要的组件之一。它保存文件系统的元数据，并在RAM中缓存集群的块映射来获得不错的性能。除非集群非常小，不然，你应当为NameNode单独

www.ChinaDBA.net 中国DBA超级论坛

指定一个节点，不在上面运行任何DataNode、JobTracker或TaskTracker服务。NameNode节点应该是集群中最强大的机器。给它尽可能多的RAM。虽然DataNode使用JBOD磁盘设备会获得更高的性能，但你绝对应该把RAID分给NameNode用以提高可靠性，以防单个驱动器出现故障。

减轻NameNode负担的一种方法是增加数据块的大小，以降低文件系统中元数据的数量。将块大小增加一倍，元数据几乎减少一半。不过对于那些不是很大的文件，它们的访问并行度也因此降低了。理想的块大小会由具体场景来决定。在配置参数`dfs.block.size`中可以设置块的大小。例如，若要让块的大小提高一倍，从默认值64 MB增加到128 MB，可以将`dfs.block.size`设置为134217728。

默认情况下，NameNode和SNN（Secondary NameNode）[①] 运行在同一台计算机上。对于中等规模的集群（10个或者更多节点），应该为SNN分配专用的机器，规格应相当于NameNode。但是，在讨论如何安装一个独立的服务器作为SNN之前，我会先解释SNN会做什么，以及不会做什么，继而讨论NameNode的一些内在机制。

由于取了一个不妥的名字，SNN有时会被混淆为NameNode的失效备份。它绝对不是。SNN仅仅是定期清理NameNode上文件系统的状态信息，使之紧凑，进而帮助 NameNode变得更有效率。NameNode使用`FsImage`和`EditLog`这两个文件来管理文件系统的状态信息。文件`FsImage`是文件系统在一些检查点上的快照，而`EditLog`记录在此检查点之后文件系统的每个增量修改（delta）。通过这两个文件可以完全确定文件系统的当前状态。当初始化NameNode时，它将合并这两个文件来创建新的快照。在NameNode初始化结束时，`FsImage`将包含新的快照，而`EditLog`将为空。之后，任何导致HDFS状态改变的操作都被追加到`EditLog`，而`FsImage`将保持不变。NameNode关闭并重新启动时，将再次进行合并，并产生新的快照。请注意这两个文件仅为在NameNode不运行时（有计划的关闭或系统故障）文件系统保留的状态信息。NameNode将文件系统的状态信息常驻在内存中，以快速地响应对文件系统的相关查询。

在繁忙的集群中，`EditLog`文件会变得非常大，并且NameNode在下次重新启动时将花很长的时间才能把`EditLog`合并到`FsImage`中。而且，集群繁忙时，NameNode两次重启之间的时间也会很长，你可能需要更频繁地做快照以便于存档。此时SNN便有了用武之地。它合并`FsImage`和`EditLog`到一个新的快照中，并让NameNode专注于活动的事务。因此，把SNN视为一个检查点服务器更为合适。合并`FsImage`和`EditLog`是很耗内存的，需要SNN与正常NameNode操作相当的内存容量。最好把SNN放在一个与主NameNode同等级别的独立的服务器上。

若要配置HDFS让其使用单独的服务器作为SNN，首先需在`conf/masters`上列出该服务器的主机名或IP地址。不过这个文件名同样令人困惑。Hadoop上的主节点（NameNode和JobTracker）为你在其上运行`bin/start-dfs.sh`和`bin/start-mapred.sh`的机器。在`conf/masters`中所列出的却是SNN，与主节点没有什么关系。

你还需在SNN上修改conf/hdfs-site.xml文件，让属性`dfs.http.address`指向NameNode主机

① 在本书撰写时，Secondary NameNode被提出将在Hadoop版本0.21中被弃用，该版本将在本书印刷时被发行。Secondary NameNode将会被一个更鲁棒的热备设计方案所取代。你应该检查自己所使用的Hadoop版本的在线文档，来确定它是否仍在使用Secondary NameNode。这个修改的专用补丁详见https://issues.apache.org/jira/browse/HADOOP-4539。

www.ChinaDBA.net 中国DBA超级论坛

地址的端口50070，如

```
<property>
    <name>dfs.http.address</name>
    <value>namenode.hadoop-host.com:50070</value>
</property>
```

你必须设置此属性，因为SNN从NameNode中获取FsImage和EditLog，是通过向下面的网址发送HTTP的Get请求得到的：

- *FsImage*——http://*namenode.hadoop-host.com*:50070/getimage?getimage=1
- *EditLog*——http://*namenode.hadoop-host.com*:50070/getimage?getedit=1

SNN也把合并的元数据通过相同的地址和端口更新到NameNode上。

8.9 恢复失效的 NameNode

故障总会发生，所以Hadoop被设计得非常有弹性。不过NameNode仍然是薄弱环节。如果NameNode当机，HDFS就会失效。一种常见的设计是通过重用SNN来建立后备的NameNode服务器[1]。毕竟，SNN具有与NameNode相似的硬件规格，而且在Hadoop上安装的目录配置应该也是一样的。如果我们做一些额外的工作，维护SNN为NameNode的功能镜像，我们可以在NameNode出现故障的情况下快速启动这个备份机。启动备份节点作为新的NameNode需要人工的干预和时间，但至少我们不会丢失任何数据。

NameNode在dfs.name.dir目录下保存了所有的文件系统元数据，包括FsImage和EditLog文件。请注意SNN服务器根本不会使用该目录。它把系统的元数据下载到fs.checkpoint.dir目录下，并在那里进一步合并FsImage和EditLog。因为SNN上的dfs.name.dir目录未被使用，我们可以通过网络文件系统（NFS）把它暴露给NameNode。我们让NameNode每次写入本地元数据目录时，也把数据写入这个挂载目录。HDFS支持在多个目录中写入元数据。你需要用一个逗号分隔的列表来设定NameNode上的dfs.name.dir，就像下面这样。

```
<property>
    <name>dfs.name.dir</name>
    <value>/home/hadoop/dfs/name,/mnt/hadoop-backup</value>
    <final>true</final>
</property>
```

这里假定NameNode和SNN上的本地dfs.name.dir目录都位于/home/hadoop/dfs/name，而且SNN上的目录被挂载到NameNode上的/mnt/hadoop-backup。当HDFS在dfs.name.dir中看到以逗号分隔的列表时，它就会将其元数据写入到列表中的每个目录。

采用这种安装方式，当NameNode当机时，NameNode和备份节点（SNN）上的本地dfs.name.dir目录将拥有相同的内容。但是，要让备份节点作为NameNode，你必须把它的IP地址改为原始NameNode的IP地址。（遗憾的是，仅更改主机名是不够的，因为DataNode缓存了DNS entry。）你还得通过执行bin/start-dfs.sh让备份节点运行起来，让它成为NameNode。

[1] 不过这种常见的设计也促成了将SNN 视为备份节点的误解。你可以在NameNode和SNN之外的机器上设置备份节点，但该机器应该在绝大部分时间处于空闲状态。

www.ChinaDBA.net 中国DBA超级论坛

若要更安全，这个新的NameNode也需要在启动之前安装备份节点。否则如果新的NameNode再失效会给你带来麻烦。如果你没有现成的节点作备份机，你至少应创建一个挂载NFS的目录。这样文件系统的状态信息可以放在多个位置。

因为HDFS会将元数据写入到`dfs.name.dir`所列出的所有目录中，如果NameNode中有多个硬盘，你可以将元数据的副本保存在属于不同硬盘的目录中。这样如果一个驱动器出现故障，就可以在没有坏盘的节点上重启NameNode。这比切换到备份节点更容易，因为切换会涉及IP地址改变及安装新的备份节点等。

回顾一下，SNN在目录`fs.checkpoint.dir`中创建文件系统元数据的快照。因为它只会周期性做检查点（默认每小时一次），元数据对于故障恢复而言不够新。但是，定期将这个目录存档到远程存储器上仍然是一个好主意。在发生灾难的情况下，从陈旧的数据中恢复起码比没有数据可用要强。NameNode及其备份节点同时失效的情况是存在的（比如一次电涌影响到了两台计算机）。另一种不幸的情况是文件系统的元数据早已损坏（比如因为人为错误或软件错误），而导致所有副本都毫无用处。从检查点的映像中恢复的详细解释可见http://issues.apache.org/jira/browse/HADOOP-2585。

本书撰写时HDFS的备份和恢复机制仍在积极地完善。你可以检查HDFS的在线文档来了解最新的消息。也有一些特殊的Linux软件，如DRBD[①]，可用在Hadoop集群上来实现很高的可用性。你可以在http://www.cloudera.com/blog/2009/07/22/hadoop-ha-configuration/看到一个示例。

8.10 感知网络布局和机架的设计

随着Hadoop集群变大，节点将分布在多个机架中，而集群的网络拓扑结构会开始影响可靠性和性能。你也许会希望集群能够在整个机架的故障中得以幸免。如上一节所述，你应在一个NameNode机架之外放置一个备份服务器。这样任何机架的失效都不会导致文件系统中元数据的所有副本被销毁。

当超过一个机架时，块副本和任务的位置就变得更加复杂。块的副本应放在彼此独立的机架上，以减少潜在的数据丢失。标准的副本数为3，在写入一个块时的默认放置策略是：如果执行写操作的客户端是Hadoop集群的一部分，第一个副本应放在客户端所在的DataNode中。否则随机放置在集群中。第二个副本随机放置在与第一个副本不同的机架中。第三个副本放在与第二个副本相同机架上的不同节点上。如果副本数大于3，后续的副本在节点上随机放置。撰写本文时，这种块放置策略被并入NameNode中。计划在版本0.21中将实现可插拔的策略。[②]

除了块的放置，任务的放置也是机架感知的。任务被放置的节点通常拥有该任务所处理块的副本。当没有这种节点可用来承担这项新任务时，任务就随机分配给机架上的一个节点，只要该机架的某个节点上可以获得这个副本。就是说，当数据局部性无法在节点级别达成，Hadoop就尝试在机架级别实现。如果还不成功，任务就随机分配给剩余节点中的一个。

① 详见http://www.drbd.org。

② 关于这个改变的描述见http://issues.apache.org/jira/browse/HDFS-385。

www.ChinaDBA.net 中国DBA超级论坛

在这一点上，你可能想知道Hadoop是如何知道节点所处机架的。这需要你来告诉它。它假定你的Hadoop集群为层次式网络拓扑，结构类似于图8-1。每个节点都有类似于文件路径的机架名。例如，图8-1中的节点H1、H2及H3全都有机架名/D1/R1。图8-1给出了一个例子，如果有多个数据中心（D1和D2），每个有多个机架（R1至R4）。在大多数情况下你需要对多个机架做统一的管理。你的机架名将采用平面名称空间，如/R1和/R2。

为了帮助 Hadoop知道每个节点的位置，你必须提供一个可执行脚本，能够把IP地址映射到机架名。这个网络拓扑的脚本必须驻留在主节点上，它的位置在core-site.xml的topology.script.file.name属性中指定。Hadoop调用脚本时会使用一组IP地址作为彼此独立的参数。该脚本会以相同的顺序（通过STDOUT）打印出每个IP地址所对应的机架名，中间以空格分隔。属性topology.script.number.args控制在任一时刻Hadoop所请求的IP地址的最大数目。简便的办法是将这个值设为1，以简化你的脚本。下面是一个网络拓扑脚本的示例。

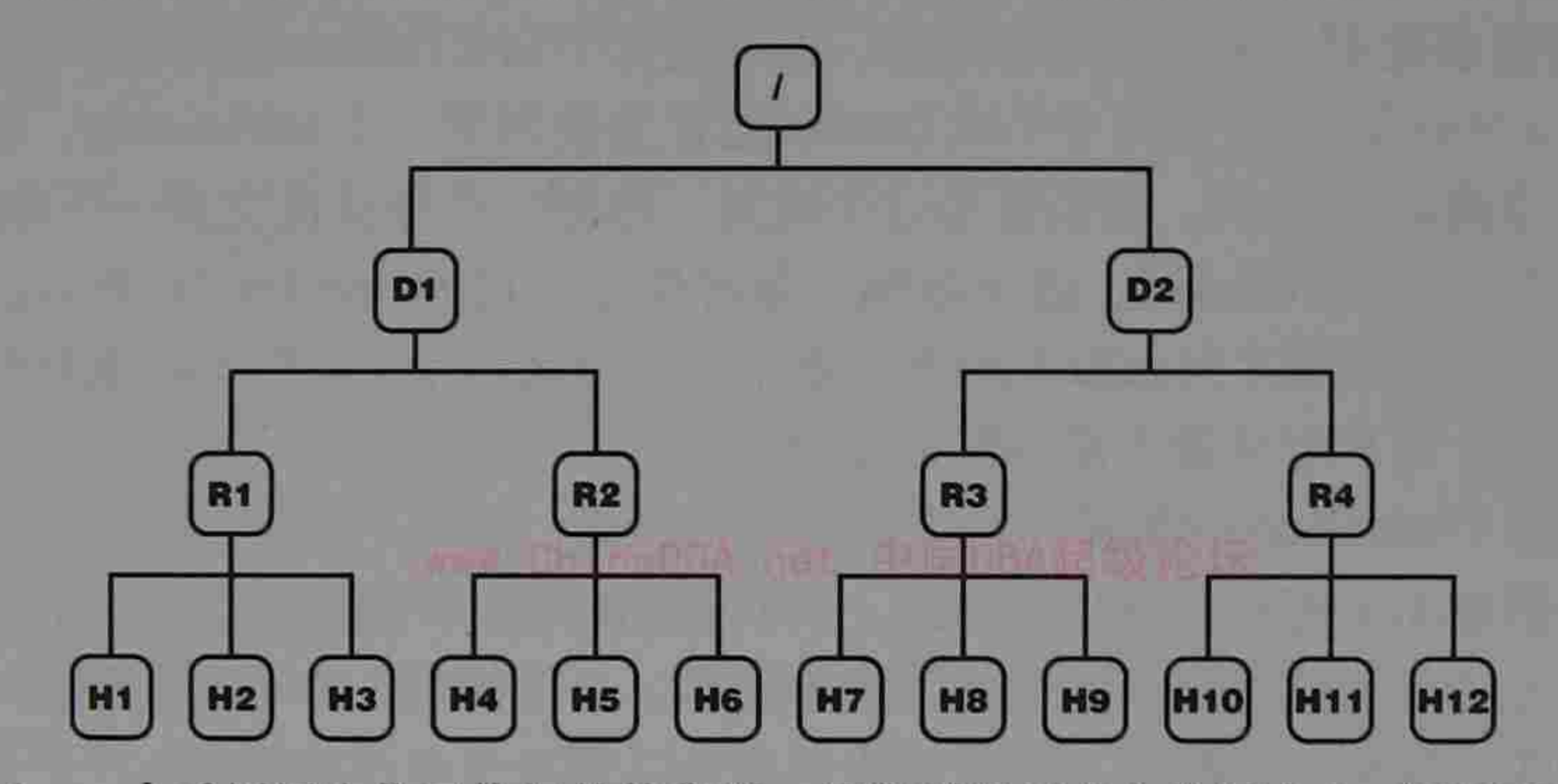


图8-1 一个采用层次化网络拓扑的集群。该集群跨越多个数据中心（D1和D2）。每个数据中心拥有多个机架(R)，每个机架有多台计算机

```
#!/bin/bash

ipaddr=$1
segments='echo $ipaddr | cut --delimiter=. --fields=4'
if [ "$segments" -lt 128 ]; then
   echo /rack-1
else
   echo /rack-2
fi
```

这个bash脚本取得IPv4地址，并提取4个八位字节（假定为十进制点符号）中的最后一个。如果最后一个八位字节少于128，节点被认为在机架1中，否则在机架2中。在较复杂的集群拓扑结构中使用查找表会更合适。另外，如果没有网络拓扑的脚本，Hadoop将假定采用平坦拓扑，所有节点都被分配到/default-rack中。

8.11 多用户作业的调度

当Hadoop集群上有越来越多来自于多个用户的作业时，你需要采取一些控制措施来避免冲突。Hadoop默认采用FIFO调度器，只要有一个作业被送到Hadoop上执行，JobTracker就会根据作

www.ChinaDBA.net 中国DBA超级论坛

业处理的需求为它分配尽可能多的TaskTracker。当事务不繁忙而你有大量闲置的处理能力时，这种调度工作得很好。但是，一些大的Hadoop作业会很容易长时间占用集群，而让较小的作业等待。如果在Hadoop集群中为较小的作业建立一个类似快速收银通道的东西，不是很棒吗？

8.11.1 多个 JobTracker

回到Hadoop版本0.19以前的时代，你不得不在物理上设置多个MapReduce集群，把基本的CPU资源在作业间分配。但是为了保持高效合理的存储利用率，这仍然是一个单一的HDFS集群。就是说，你的Hadoop集群有Z个从节点。你必须让一个NameNode把所有Z个节点作为DataNode管理起来。所有Z个节点也都将为TaskTracker。到目前为止，所有这些TaskTracker将指向相同或唯一的JobTracker。

多集群安装的一个诀窍是用多个JobTracker，每个JobTracker（互斥地）控制一个TaskTracker子集。例如，若要创建两个MapReduce集群，必须让X个TaskTracker指向一个JobTracker（利用mapred.job.tracker属性），让Y个TaskTracker配置为使用第二个JobTracker。两个MapReduce集群之间的从节点满足X+Y=Z。若要使用这个设置，你把一些作业提交到一个JobTracker，而把其他的作业分给另一个JobTracker。这就限制了每种作业可用的TaskTracker数目。作业的类型并不一定会左右作业分配到哪个MapReduce池。更常见的是将每个MapReduce池划给一个用户组来使用。这将强制一个用户组只能占用有限的资源。

物理上设置多个MapReduce集群这种方式有很多缺点。它并不方便，因为需要人们记住使用的是哪个池。数据也不太可能总在任务本地。（有时所有的副本可能都处于池外的DataNode上。）而且，这种设置无法灵活地适应不断变化的资源需求。可喜的是，从0.19版开始，Hadoop在调度器上采用了插件式的架构，并提供了两个新的调度器以解决作业冲突问题。一个是Facebook开发的公平调度器（Fair Scheduler）；另一个是雅虎开发的容量调度器（Capacity Scheduler）。

8.11.2 公平调度器

公平调度器引入了池的概念。作业都标记为属于特定的池，并且每个池被配置为必须拥有一定数量的map slot和reduce slot。当task slot被释放时，公平调度器会首先满足这些最低限度的保障。在保障被满足后，slot再在作业之间“公平共享”，使得每个作业获取大致相同的计算资源。你也可以设置作业的优先级，让更高优先级的作业获得更多资源（一些作业实际上比其他作业更有特权）。

公平调度器可以从Hadoop安装目录contrib/fairscheduler下的hadoop-*-fairscheduler.jar文件中获得。安装时，可将此jar文件直接移动到Hadoop的lib/目录下。或者，可以修改脚本conf/hadoop-env.sh 中的HADOOP_CLASSPATH来包含此jar文件。

你需要在hadoop-site.xml中设置几个属性来完全启用与配置公平调度器。首先，将mapred.jobtracker.taskScheduler设置为org.apache.hadoop.mapred.FairScheduler，让Hadoop使用公平调度器，而非默认调度器。然后，对公平调度器的几个属性进行配置。其中最重要的属性为mapred.fairscheduler.allocation.file，它指向用于定义不同池的文件。该文件通常命名为pools.xml，指定了每个池的名称和容量。mapred.fairscheduler.poolnameproperty定义了jobconf属性，调度器会通过它来确定作业使用哪个池。一种实用的配置方式是将它设置

www.ChinaDBA.net 中国DBA超级论坛

为一个新属性，如pool.name，并将其默认值设为${user.name}。公平调度器自动赋予每个用户其独有的池。默认情况下这个特定的pool.name 将分配每个作业到其宿主的池。你可以在作业的jobconf中修改pool.name的属性来分配这个作业到一个不同的池中[①]。最后，如果mapred.fairscheduler.assignmultiple属性被设置为true，会允许调度器在每个心跳到来时指定map任务和reduce任务，从而提高了构建速度和吞吐量。简而言之，mapred-site.xml中的属性设置如下所示：

```
<property>
  <name>mapred.jobtracker.taskScheduler</name>
  <value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
<property>
  <name>mapred.fairscheduler.allocation.file</name>
  <value>HADOOP_CONF_DIR/pools.xml</value>
</property>
<property>
  <name>mapred.fairscheduler.assignmultiple</name>
  <value>true</value>
</property>
<property>
  <name>mapred.fairscheduler.poolnameproperty</name>
  <value>pool.name</value>
</property>
<property>
  <name>pool.name</name>
  <value>${user.name}</value>
</property>
```

调度器中对池的定义放在配置文件pools.xml中。它给出每个池的名字及其容量限制。约束可以包括map slot或reduce slot的最小数目。还可以包含正在运行作业的最大数目。此外，你可以设置每个用户运行作业的最大数目，并为特定的用户重新定义这个最大值。pools.xml的示例如下所示：

```
<?xml version="1.0"?>
<allocations>
  <pool name="ads">
     <minMaps>2</minMaps>
     <minReduces>2</minReduces>
  </pool>
  <pool name="hive">
     <minMaps>2</minMaps>
     <minReduces>2</minReduces>
     <maxRunningJobs>2</maxRunningJobs>
  </pool>
  <user name="chuck">
     <maxRunningJobs>6</maxRunningJobs>
  </user>
  <userMaxJobsDefault>3</userMaxJobsDefault>
</allocations>
```

① 是的，你可以在另一个用户的池中运行你的作业，但这可不太礼貌。这样做的主要用途是将特殊的作业分配在特定的池中。例如，你可能希望所有cron作业被归入一个单独的池中，而不是让它们在每个用户的池中运行。

www.ChinaDBA.net 中国DBA超级论坛

这个pools.xml定义了两个特殊的池——“ads”和“hive”。每个都保证有至少两个map slot和两个reduce slot。“hive”池被限定一次最多运行两个作业。若要使用这些池，需设置作业配置中的pool.name属性为“ads”或“hive”。这个pools.xml还限制一个用户最多有3个同时运行的作业，但用户“chuck”最多可以运行6个。

请注意pools.xml文件每15秒被重新读取一次。你可以在运行时修改此文件，并动态地重新分配容量。任何在此文件中未定义的池都没有容量保证，也没有同时运行作业数量的限制。

当你让Hadoop集群运行公平调度器时，有一个Web用户界面可用于管理这个调度器。网页位于http://<jobtracker url>/scheduler。除了让你知道作业被如何调度之外，它还允许更改作业所属的池，以及作业的优先级。图8-2显示了该页面的示例截图。

localhost Job Scheduler Administration

Pools

Pool	Running Jobs	Min Maps	Min Reduces	Running Maps	Running Reduces
ads	0	2	2	0	0
chuck	1	0	0	1	1
hive	0	2	2	0	0
default	0	0	0	0	0

Running Jobs

Submitted	JobID	User	Name	Pool	Priority	Maps			Reduces		
						Finished	Running	Fair Share	Finished	Running	Fair Share
Aug 11, 04:08	job_200908110346_0002	chuck	streamjob6351400141424754280.jar	chuck	NORMAL	3 / 4	1	2.0	0 / 1	1	2.0

Scheduling Mode

The scheduler is currently using **Fair Sharing mode** Switch to FIFO mode

图8-2　监视Hadoop公平调度器的Web用户界面。上面的表显示了所有可用的池和每个池的使用率。下面的表有一个名为Pool的列，可用于监视或更改每个作业的池

容量调度器与公平调度器有相似的目标。但是，容量调度器作用于队列，而不是池。有兴趣的读者可以通过在线文档了解到有关容量调度器的更多内容：http://hadoop.apache.org/common/docs /r0.20.0 /capacity_scheduler.html。

8.12　小结

分布式集群的管理非常复杂，而Hadoop并非异类。在本章，我们介绍了很多常见的管理任务。如果你的设置复杂并且问题繁多，Hadoop的邮件列表[①]就是一个有用的资源。许多经验丰富的Hadoop管理员在其中非常活跃，有可能他们中的一位就曾遇到过你的问题。另外，如果你主要是希望拥有一个基本的Hadoop集群，而免去管理的麻烦，你可以考虑使用Cloudera的发行版[②]，或者采用一个Hadoop云服务，我们会在下一章中加以讨论。

① http://hadoop.apache.org/common/mailing_lists.html。

② http://www.cloudera.com/distribution。

www.ChinaDBA.net 中国DBA超级论坛

Part 3

第三部分

Hadoop 也疯狂

本部分探讨围绕 Hadoop 的更大的生态系统。云服务提供了创建 Hadoop 集群的另一种方案，可以替代自己购买并拥有硬件以创建 Hadoop 集群的方式。许多附加产品包在 MapReduce 之上提供了更高级别的编程抽象。最后，我们会给出几个 Hadoop 解决实际业务问题的案例。

www.ChinaDBA.net 中国DBA超级论坛

本部分内容

- 第 9 章　在云上运行 Hadoop
- 第 10 章　用 Pig 编程
- 第 11 章　Hive 及 Hadoop 群
- 第 12 章　案例研究

www.ChinaDBA.net 中国DBA超级论坛

第9章 在云上运行Hadoop

本章内容

- 基于亚马逊Web服务（AWS）建立计算云
- 在AWS云上运行Hadoop
- 从AWS Hadoop云上导入导出数据

根据你对数据处理的需求，Hadoop的负载随时间有很大变化。你可能偶尔要用上百个节点来处理几个大型的数据处理作业，但这些节点在其他时间则被闲置。你可能刚开始用Hadoop，并希望在投资购买一个专用集群前先熟悉它。你可能在创业阶段，需要节省现金，不想将资金花在购买Hadoop集群上。凡此种种，去买一个集群倒不如去租用它更有意义。

这种将计算资源作为一个远程的服务，以灵活、符合成本效益的方式提供出来的模型，被称为云计算。云计算中最富盛名的基础架构服务平台就是Amazon Web Services (AWS)。你可以从AWS按需租用计算和存储服务。截至本书写作时，租用一个相当于1.0 GHz 32-bit Operton处理器、1.7 GB内存和160 GB磁盘的计算单元，花费为每小时0.10美元。采用100个这样的节点构成集群，每小时仅花费10美元！在不久以前，仅有少部分人才能拥有如此规模的集群。有赖于AWS和其他此类服务平台，今天许多人已经可以获得大规模的计算能力。

因为其灵活性和成本效益，在AWS云上运行Hadoop成为一种很常见的方式，本章我们将学习如何来安装和配置它。

9.1 Amazon Web Services 简介

Amazon Amazon Web Services的全部功能本身就可以写一本书。而Amazon还在不断添加新的服务和功能。我们建议你浏览AWS网站(http://aws.amazon.com)以便了解更多的细节和最新的产品。这里我们仅讨论运行一个Hadoop集群所需的基础知识。

在所有AWS提供的服务中，弹性计算云（Elastic Compute Cloud, EC2）和简单存储服务（Simple Storage Service, S3）是两个核心的服务。要让Hadoop运行在云上，就需要理解它们。EC2服务提供了可供Hadoop节点运行的计算能力。你可以将EC2视为一个虚拟机的大农场。EC2实例是AWS描述一个虚拟计算单元的术语。每个Hadoop节点都要占用一个EC2实例。你可以按需租用EC2实

www.ChinaDBA.net 中国DBA超级论坛

例，并按小时付费。

汽车租赁公司会把你归还汽车时遗留在车厢中的所有东西都丢掉。同样，在实例终止时，你在EC2实例上的所有数据都会被删除。如果你希望将来仍使用这些数据，就必须保证它们被放在某些持久化的存储中。Amazon的S3服务是一种云存储服务，你可以使用它来做到这一点。

每个EC2实例的功能就像一台可以通过互联网访问与控制的商用Intel计算机。实例是用亚马逊虚拟机镜像（Amazon Machine Image）来启动的，它也被简称为AMI或者镜像。有更高要求的用户可以创建自己的镜像，但是大多数用户都能在预设的许多镜像中找到一个合适的。一些镜像仅有基本的操作系统——EC2上支持的操作系统包括6种以上的Linux，以及Windows Server和OpenSolaris。另一些镜像会在操作系统的基础上加入预装的软件，如数据库服务器、Apache HTTP服务器、Java应用服务器等。AWS提供预先配置好的运行在Linux上的Hadoop镜像，而且Hadoop内置同时支持与EC2和S3一起使用。

9.2 安装 AWS

本节是AWS的安装简介，仅涉及运行Hadoop集群的基本知识。如果你已经熟悉了EC2实例的装载和使用，可以直接跳到下一节，开始在AWS上安装Hadoop。

使用AWS之前，你先要注册一个账户。访问http://aws.amazon.com/，然后点击“Sign Up Now”。注册的过程不言自明，不比从Amazon购买一本书更复杂。注册过程会为你设置一个Amazon的账户（如果你以前在Amazon买过东西，你可能已经有账户了），然后激活它，以便支付AWS的使用费。

注意 Amazon推出了Elastic MapReduce(EMR)服务，大大简化了Hadoop在AWS上的使用。最重要的简化是你不再需要安装和装载自己的EC2实例集群。但不足之处在于你会失去对集群的一些控制，而且还要额外地为EMR服务付费。我们将在9.6节讨论EMR。不过，即使你不需要参与整个安装的过程，我们依然强烈建议你继续阅读并理解如何安装自己的EC2集群来运行Hadoop。至少，更多地了解Hadoop在EC2上运行的细节，就能明白EMR在底层都做了些什么。

在激活AWS上的Amazon账户之后，还需要经历3个步骤才能创建并使用实例。

(1) 获得账户ID以及认证密钥与证书。把它们安装在本地计算机上，来保证与AWS通信的安全。这些安全机制确保只有你可以用该账户租用计算资源。

(2) 下载并安装命令行工具来管理你的EC2实例，包括在虚拟集群上启动和终止EC2实例的一些特殊程序。

(3) 生成一个SSH密钥对。当一个EC2实例启动之后，可以使用SSH（直接或间接地通过特殊工具）登录。在默认的SSH机制中，使用SSH密钥对EC2实例进行认证，以代替密码认证。

下面的各个小节将逐一阐述每个步骤。

www.ChinaDBA.net 中国DBA超级论坛

9.2.1　获得AWS身份认证凭据

AWS支持两种类型的身份认证机制：访问密钥标识符（AWS Acess Key Identifier）和X.509证书。为了在AWS上运行Hadoop，这两种机制你都需要，你可以在管理AWS账户的Access Identifiers网页中（http://aws.amazon.com/account/）安装它们。访问密钥标识符包含一个Access Key ID和Secret Access Key。图9-1显示了Access Identifiers页面中的一部分。Access Key ID是一个有20个字符的字符序列，而Secret Access Key是一个包含40个字符的序列。不要和任何人共享你的Secret Access Key。这个Web页面需要额外地点击Show来显示它（以防有人从后面偷窥）。如果你拥有的Secret Access Key已经泄露了，你应该生成一个新的密钥。当稍后安装Hadoop集群时，需要设定你的Access Key ID和Secret Access Key。

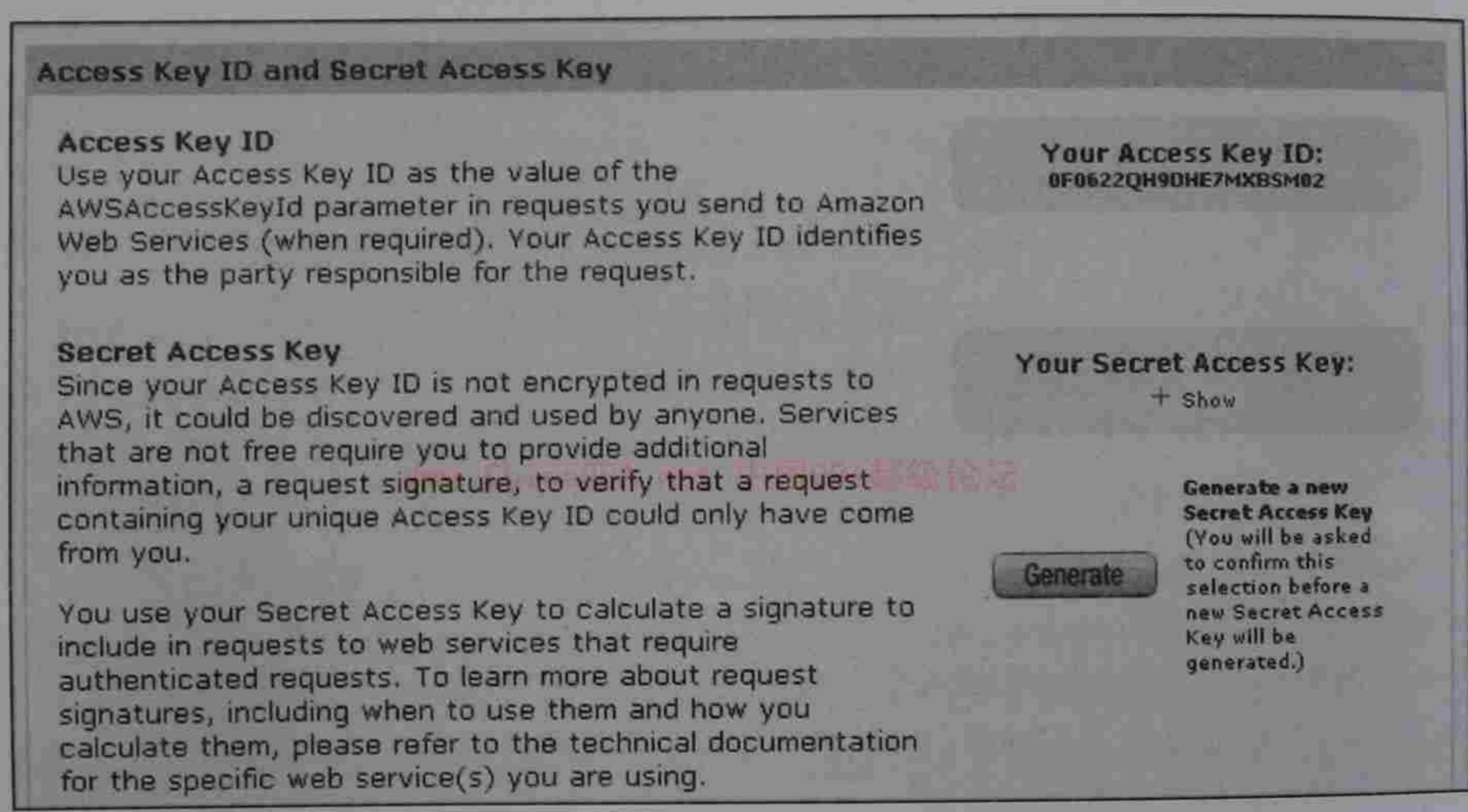


图9-1　获取AWS上的Access Key ID和Secret Access Key

提示　在一些情况下，当你希望Hadoop访问S3的时候，会使用特殊格式的URI告诉Hadoop你在AWS上的Access Key ID和Secret Access Key。不过AWS允许在Secret Access Key中使用斜杠（/），这会导致URI内部出现混淆。虽然有一些方法无需使用URI就可以让Hadoop知道你在AWS上的Access Key ID，但是更方便的办法是重新生成你的Secret Access Key，直到得到一个其中不带有斜杠的密钥。

设置X.509证书要做的事情略多一些。如图9-2所示，在同一个Access Identifiers页面下还有一段名为X.509 Certificate。单击Create New生成一个新的X.509证书。一个证书有两个密钥：公钥和私钥。与Access Key ID和Secret Access Key不同，X.509证书中的公钥和私钥有几百个字符长，而且它们必须按文件来存储和管理。创建一个新的X.509证书后，你会进入一个能够下载这两个密钥/文件的页面，如图9-3所示。

www.ChinaDBA.net 中国DBA超级论坛

X.509 Certificate

Certificate File

An X.509 Certificate consists of Public Key and a Private Key. The file containing the public key, the certificate file, must contain a base64-encoded DER certificate body. The file containing the private key, the Private Key file, must contain a base64-encoded PKCS#8 private key. The Private Key is used to authenticate requests to AWS.

AWS accepts any syntactically and cryptographically valid X.509 certificates. They do not need to be from a formal Certificate Authority (CA).

To learn more about how certificates are used to authenticate requests, please see the Developer Guide for services that support X.509 authentication.

Your X.509 Certificate:
cert-ZF34WZELIV2DZOTL2V6OOKAD2PNDJHRK.pem

Create New — Create a New X.509 Certificate
Download — Download Your X.509 Certificate
Upload — Upload Your Own X.509 Certificate
Delete — Delete Your Current X.509 Certificate from AWS

图9-2　管理X.509证书。你可以上传自己的证书，也可以由AWS创建

Create Success

You have successfully created a new X.509 Certificate for your AWS account.

Please download your Private Key file now. You must download your Private Key file (pk-), by clicking the link below before you navigate away from this page. AWS does not store your private key information. You will not be able to download the Private Key file at any other time. If you do not download the Private Key file now, you will have to create a new certificate and private key.

Your Private Key file

pk-PXTTXYE5BUUEPJ5M6HLVOYWEAULVVGQ4.pem

Download Private Key File

IMPORTANT: You should store your Private Key file in a secure location. If you lose your Private Key file you will need to create a new certificate to use with your account. AWS does not store Private Key Information.

Your Private Key is secret, and should be known only by you. You should never include your Private Key information in a requests to AWS, except encrypted as a signature. You should also never e-mail your Private Key file to anyone. It is important to keep your Private Key confidential to protect your account.

Please download your certificate file. You can download your certificate file now using the link below, or at your convenience from the Access identifiers page.

Your X.509 Certificate file

cert-PXTTXYE5BUUEPJ5M6HLVOYWEAULVVGQ4.pem

Download X.509 Certificate

Return to Access Identifier Page

图9-3　下载X.509证书的私钥和证书文件

www.ChinaDBA.net 中国DBA超级论坛

公钥也被称为证书文件（certificate file）。私钥，顾名思义，不能与任何人共享。甚至Amazon本身也不能存储它的副本。AWS指定证书和私钥的文件名，而你应该在存储时保持它们的文件名不变。证书和私钥所使用的文件名分别带有前缀cert-和pk-，而且都有.pem的文件扩展名。你需要在本机home目录中创建一个名为.ec2的目录，来保存这两个文件。在Linux环境中，本地计算机将会保存如下的文件：

```
~/.ec2/cert-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
~/.ec2/pk-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
```

最后，你还应该注意AWS Account Number。它在Access Identifiers页面的右上角附近，而且是一个12位带连接符的数字，类似“4952-1993-3132”。Account ID是它的一个无连接符版本，如“495219933132”。在EC2上安装Hadoop时会用到Account ID。

要生成和保存的值似乎太多了。总结一下，你一共应该有5个值。

- 20个字符的包含字母与数字的Access Key ID
- 40个字符的Secret Access Key
- 保存在.ec2目录下的X.509认证文件
- 保存在.ec2目录下的X.509私钥文件
- 保存12位（无连接符）的AWS Account ID

这些值在后面将会用于向AWS的认证，进而允许你操控Hadoop集群。

www.ChinaDBA.net 中国DBA超级论坛

9.2.2　获得命令行工具

在获得所有的安全证书后，你应该下载并配置AWS的命令行工具来初始化和管理EC2实例。这些工具是用Java写的，你应该在本地计算机上安装好Java环境。

EC2命令行工具全都包含在一个ZIP文件中，可从AWS EC2资源中心下载[①]。将这个文件解压到你的AWS工作目录中。在解压后的文件中，你会看到Java工具，以及Windows、Linux和Mac OS X的shell脚本。

你不必配置命令行工具，但是在使用它们之前必须设置几个环境变量。环境变量EC2_HOME目录应当指向命令行工具解压的文件目录。除非你重命名了这个目录，否则它的名字应为ec2-api-tools-A.B-nnnn，这里A、B和n为版本/发布号。你还应该让EC2_CERT 和EC2_PRIVATE_KEY分别指向X.509认证文件和私钥文件。一个好办法是用脚本来设置所有AWS命令行工具所需的环境变量。代码清单9-1中给出了用于Linux、Unix和Mac OS X的脚本ec2-init.sh。在使用任何与AWS相关的工具之前，执行如下命令来运行脚本：

```
source ec2-init.sh
```

或

```
. ec2-init.sh
```

① http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351&categoryID=88。

代码清单9-1 ec2-init.sh：设置EC2工具所用变量的Unix脚本

```
export JAVA_HOME = /Library/Java/Home
export EC2_HOME = ~/Projects/Hadoop/aws/ec2-api-tools-1.3-30349
export PATH = $PATH:$EC2_HOME/bin:$HADOOP_HOME/src/contrib/ec2/bin
export EC2_PRIVATE_KEY = ~/.ec2/pk-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
export EC2_CERT = ~/.ec2/cert-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
```

Windows有一个类似的版本，如代码清单9-2所示。你可以在命令提示符方式下执行ec2-init.bat。

代码清单9-2 ec2-init.bat：一个设置EC2工具所用变量的Windows脚本

```
set JAVA_HOME = "C:\Program Files\Java\jdk1.6.0_08"
set EC2_HOME = "C:\Program Files\Hadoop\aws\ec2-api-tools-1.3-30349"
set PATH = %PATH%;%EC2_HOME%\bin;%HADOOP_HOME%\src\contrib\ec2\bin
set EC2_PRIVATE_KEY = c:\ec2\pk-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
set EC2_CERT = c:\ec2\cert-HKZYKTAIG2ECMXYIBH3HXV4ZBZQ55CLO.pem
```

如果你经常使用AWS，则不必每次都单独运行脚本，可以把它直接集成在操作系统的启动脚本中（例如.profile和autoexec.bat）。

在这个脚本中的路径名会与你安装环境中的路径名不同。需要设置环境变量JAVA_HOME让AWS命令行工具可以工作。虽然通常它已经在其他地方设置过了，我们在这里还需对它进行设置。脚本在系统路径PATH中添加了命令行工具的bin目录。这会使工具的执行更为方便，因为你不必每次都设定其完全路径。Hadoop EC2工具的目录也被添加到PATH中，不过我们到下一节才会讨论它们。

AWS的机器部署在世界各地。在本书写作时，AWS支持美国和欧洲这两个地区。作为一个可选的步骤，我们可以选择在哪个地区运行EC2实例以减少延迟。在运行上面的脚本设置环境变量之后，让我们运行第一个AWS命令行工具来询问Amazon当前有哪些地区可用。

```
ec2-describe-regions
```

你会得到与下面类似的返回结果：

```
REGION    us-east-1    us-east-1.ec2.amazonaws.com
REGION    eu-west-1    eu-west-1.ec2.amazonaws.com
```

第二列为地区名（us-east-1和eu-west-1），而第三列为相应的service endpoints（服务点）。默认的地区为us-east-1。如果你要选择不同的服务地区，需要设置环境变量EC2_URL为相应的服务点。你可以在前面提到的AWS初始化shell脚本中设置它。

提示 除了官方的命令行工具之外，还有一些图形化工具用于管理对EC2和S3的使用。这些图形化工具对用户更为友好。较为流行的两个都来自FireFox的扩展：Elasticfox和S3Fox。Elasticfox（http://developer.amazonwebservices.com/connect/entry.jspa?entryID=609）支持基本的EC2管理特性，如启动新的EC2实例并列出当前运行的实例。S3Fox（http://www.suchisoft.com/ext/s3fox.php）是一个第三方工具，用于管理你的S3存储。它的功能非常类似于远程存储管理的图形化FTP客户端。

www.ChinaDBA.net 中国DBA超级论坛

9.2.3 准备 SSH 密钥对

启动EC2实例之后，你可以登录它来运行程序和服务。默认的（在公共镜像中的）登录机制使用基于一个密钥对的SSH。这个密钥对的一半（公钥）被嵌入在EC2实例中，而另一半（私钥）在你的本地机器上。这对密钥共同作用以确保本地机器和EC2实例之间的通信安全。

注意 一些读者可能对用基于密码的SSH来登录远程计算机更为熟悉，而基于密钥对的SSH是与此有别的另一种机制。与密码不同，身份认证是通过存储在本地计算机上的私钥来完成的。就像密码一样，私钥文件不能被未经授权的人获得。

每个SSH密钥对有一个key name（密钥名）来标识。当请求Amazon EC2创建一个实例时，你必须通过其对应的密钥名来指定嵌入在该实例中的一个公钥。在创建任何EC2实例之前，SSH公钥必须存在，且已经在Amazon中注册。

如下的命令会生成一个SSH公钥/私钥对，并以gsg-keypair为密钥名将公钥注册到Amazon EC2中。

```
ec2-add-keypair gsg-keypair
```

有趣的是，这个命令并不在本地文件中存储私钥。相反，它生成类似图9-4所示的标准输出(stdout)，其中一部分为私钥。你必须手工使用文本编辑器将它存储在一个文件中。具体而言，你需要复制和粘贴在如下两行之间（包含这两行）的内容，并将其保存为一个名为id_rsa-gsg-keypair的新文件。

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

为了管理方便，你应该把文件保存到与X.509私钥和认证文件所在的同一目录.ec2中。你还需要锁定文件权限，使得它只能被你读取。在Linux和其他Unix系统中，使用如下的命令：

```
chmod 600 ~/.ec2/id_rsa-gsg-keypair
```

一个Hadoop集群中的所有EC2实例都只有一个相同的公钥。一个私钥可以登录到集群上的所有节点，因此只需要一个SSH密钥对。当在多个Hadoop集群上工作，或者使用到所在Hadoop集群之外的其他EC2实例时，你也可以选择使用一个以上的密钥对。

到此你已经一次性地安装了用来启动Amazon云上计算集群的凭证和证书。你可以手工地使用AWS工具来启动EC2实例，并登录上去启动你的Hadoop集群。但是，这个过程是非常耗时且容易出错的。幸好Hadoop提供了使用AWS的工具，我们将在下一节进行讨论。在结束本节之前，我们建议你花一些时间阅读EC2的文档，它包含在Getting Started Guide中[①]。EC2有许多配置和定制化的选项。理解它们可以让你在优化AWS的Hadoop集群时得心应手。

① http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/

www.ChinaDBA.net 中国DBA超级论坛

```
KEYPAIR gsg-keypair  1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f
-----BEGIN RSA PRIVATE KEY-----
MIIEoQIBAAKCAQBuLFg5ujHrtm1jnutSuoO8Xe56LlT+HM8v/xkaa39EstM3/aFxTHgElQiJLChp
HungXQ29VTc8rc1bW0lkdi23OH5eqkMHGhvEwqa0HWASUMll4o3o/IX+0f2UcPoKCOVUR+jx71Sg
5AU52EQfanIn3ZQ8lFW7Edp5a3q4DhjGlUKToHVbicL5E+g45zfB95wIyywWZfeW/UUF3LpGZyq/
ebIUlq1qTbHkLbCC2r7RTn8vpQWp47BGVYGtGSBMpTRP5hnbzzuqj3itkiLHjU39S2sJCJ0TrJx5
i8BygR4s3mHKBj8l+ePQxG1kGbF6R4yg6sECmXn17MRQVXODNHZbAgMBAAECggEAY1tsiUsIwDl5
91CXirkYGuVfLyLflXenxfI50mDFms/mumTqloHO7tr0oriHDR5K7wMcY/YY5YkcXNo7mvUVD1pM
ZNUJs7rw9gZRTrf7LylaJ58kOcyajw8TsC4e4LPbFaHwS1d6K8rXh64o6WgW4SrsB6ICmr1kGQI7
3wcfgt5ecIu4TZf0OE9IHjn+2eRlsrjBdeORi7KiUNC/pAG23I6MdDOFEQRcCSigCj+4/mciFUSA
SWS4dMbrpb9FNSIcf9dcLxVM7/6KxgJNfZc9XWzUw77Jg8x92Zd0fVhHOux5IZC+UvSKWB4dyfcI
tE8C3p9bbU9VGyY5vLCAiIb4qQKBgQDLiO24GXrIkswF32YtBBMuVgLGCwU9h9HlO9mKAc2m8Cm1
jUE5IpzRjTedc9I2qiIMUTwtgnw42auSCzbUeYMURPtDqyQ7p6AjMujp9EPemcSVOK9vXYL0Ptco
xW9MC0dtV6iPkCN7gOqiZXPRKaFbWADp16p8UAIvS/a5XXk5jwKBgQCKkpHi2EISh1uRkhxljyWC
iDCiK6JBRsMvpLbc0v5dKwP5alo1fmdR5PJaV2qvZSj5CYNpMAy1/EDNTY5OSIJU+0KFmQbyhsbm
rdLNLDL4+TcnT7c62/aH01ohYaf/VCbRhtLlBfqGoQc7+sAc8vmKkesnF7CqCEKDyF/dhrxYdQKB
gC0iZzzNAapayz1+JcVTwwEid6j9JqNXbBc+Z2YwMi+T0Fv/P/hwkX/ypeOXnIUcw0Ih/Y+GBVAC
DQbsz7LcY1HqXiHKYNWNvXgwwO+oiChjxvEkSdsTTIfnK4VSCvU9BxDbQHjdiNDJbL6oar92UN7V
rBYvChJZF7LvUH4YmVpHAoGAbZ2X7XvoeEO+uZ58/BGKOIGHByHBDiXtzMhdJr15HTYjxK7OgTZm
gK+8zp4L9IbvLGDMJO8vft32XPEWuvI8twCzFH+CsWLQADZMZKSsBasOZ/h1FwhdMgCMcY+Qlzd4
JZKjTSu3i7vhvx6RzdSedXEMNTZWN4qlIx3kR5aHcukCgYA9T+Zrvm1F0seQPbLknn7EqhXIjBaT
P8TTvW/6bdPi23ExzxZn7KOdrfclYRph1LHMpAONv/x2xALIf91UB+v5ohy1oDoasL0gij1houRe
2ERKKdwz0ZL9SWq6VTdhr/5G994CK72fy5WhyERbDjUIdHaK3M849JJuf8cSrvSb4g==
-----END RSA PRIVATE KEY-----
```


图9-4　ec2-add-keypair的输出示例。第1行为密钥签名，其余为私钥

9.3 在 EC2 上安装 Hadoop

为了在EC2集群上运行Hadoop，首先需要在本地计算机上安装Hadoop并能够在单机模式下运行。本地的Hadoop安装所包含的脚本可以帮助你启动并登录EC2的Hadoop集群。这些脚本放在Hadoop安装后的目录`src/contrib/ec2/bin`中。

9.3.1 配置安全参数

你必须配置一个单独的初始化脚本src/contrib/ec2/bin/hadoopec2-env.sh。在这个脚本中，依照9.2.1节中所得到的值设置如下三个变量。

- *AWS_ACCOUNT_ID* ——12位AWS账户ID。
- *AWS_ACCESS_KEY_ID* ——20个字符，包含字母数字的Access Key ID。
- *AWS_SECRET_ACCESS_KEY* ——40个字符的Secret Access Key。

EC2上的Hadoop工具从环境变量中得到其他的安全参数（在执行`source aws-init.sh`时被设置）。如果你遵照了9.2节AWS的安装方法，用默认值也可正常工作。

9.3.2 配置集群类型

需要在`hadoop-ec2-env.sh`中指定Hadoop集群的配置，设置3个主要的参数：`HADOOP_VERSION`、`INSTANCE_TYPE`和`S3_BUCKET`。在讲述如何设置这些参数之前，让我们先交待一下背景。

在创建一个实例之前，Amazon EC2必须知道实例的类型和用于启动实例的镜像。实例类型代表虚拟机的物理配置（CPU、RAM及磁盘空间等）。在本书写作时，已经有5种实例类型，分为两组：standard和high-CPU。High-CPU类型是为计算密集的工作准备的，它们很少被用于Hadoop

www.ChinaDBA.net 中国DBA超级论坛

应用这种数据密集型的工作。Standard有3种实例类型，表9-1列出了它们的特征。

表9-1 各种EC2实例类型的配置

类 型	CPU	内 存	存储容量	平 台	I/O	名 字
Small	1个EC2计算单元	1.7 GB	160 GB	32-bit	Moderate	m1.small
Large	4个EC2计算单元	7.5 GB	850 GB	64-bit	High	m1.large
Extra Large	8个EC2计算单元	15 GB	1690 GB	64-bit	High	m1.xlarge

配置越好的实例类型价格越高，你最好到AWS网站上查看最新的报价。

只有Amazon的S3存储服务可以存储启动EC2实例所用的镜像。所有类型的安装都有许多现有的镜像可用。你可以使用一个公共镜像，或购买一个特别定制的镜像，甚至自己去创建一个镜像。彼此类似的镜像存储在相同的S3桶分区（bucket）[①]中。标准的公共Hadoop镜像要么在hadoop-ec2-images桶分区中，要么在hadoop-images桶分区中。事实上，我们仅使用hadoopimages桶分区，因为hadoop-ec2-images桶分区不再提供更新的Hadoop版本（0.17.1之后）。有重大Hadoop版本发布时，Hadoop团队将新的EC2镜像放在hadoopimages桶分区中。在任意时刻，执行如下的EC2命令来查看可获得的Hadoop镜像：

```
Terminal — bash — 164x10
chuck-laws-computer:~/Projects/Hadoop/aws/ec2-api-tools-1.3-30349 chuck$ ec2-describe-images -x all | grep hadoop-images
IMAGE   ami-65987c0c   hadoop-images/hadoop-0.17.1-i386.manifest.xml   914733919441   available   public   i386   machine   aki-a71cf9ce   ari-a51cf9cc
IMAGE   ami-4b987c22   hadoop-images/hadoop-0.17.1-x86_64.manifest.xml 914733919441   available   public   x86_64machine   aki-b51cf9dc   ari-b31cf9da
IMAGE   ami-b0fe1ad9   hadoop-images/hadoop-0.18.0-i386.manifest.xml   914733919441   available   public   i386   machine   aki-a71cf9ce   ari-a51cf9cc
IMAGE   ami-90fe1af9   hadoop-images/hadoop-0.18.0-x86_64.manifest.xml 914733919441   available   public   x86_64machine   aki-b51cf9dc   ari-b31cf9da
IMAGE   ami-ea36d283   hadoop-images/hadoop-0.18.1-i386.manifest.xml   914733919441   available   public   i386   machine   aki-a71cf9ce   ari-a51cf9cc
IMAGE   ami-fe37d397   hadoop-images/hadoop-0.18.1-x86_64.manifest.xml 914733919441   available   public   x86_64machine   aki-b51cf9dc   ari-b31cf9da
IMAGE   ami-fa6a8e93   hadoop-images/hadoop-0.19.0-i386.manifest.xml   914733919441   available   public   i386   machine   aki-a71cf9ce   ari-a51cf9cc
IMAGE   ami-cd6a8ea4   hadoop-images/hadoop-0.19.0-x86_64.manifest.xml 914733919441   available   public   x86_64machine   aki-b51cf9dc   ari-b31cf9da
chuck-laws-computer:~/Projects/Hadoop/aws/ec2-api-tools-1.3-30349 chuck$
```

图9-5 一些AWS中可供使用的Hadoop镜像

图9-5显示了上述命令的输出示例。每一行描述一个可用的EC2镜像。每个镜像列出了11条属性，它们大多仅对AWS高级用户有用。对我们而言，我们所需的所有信息都可以从第三列中读取，这一列也被称为镜像的显示位置（Manifest location）。它们被表示为两层结构，其中上层为镜像所在的S3桶分区。如前所述，我们关注的是hadoop-images桶分区。

```
ec2-describe-images -x all | grep hadoop-images
```

显示位置包含了Hadoop的版本号，还包括一个字段，为i386或x86_64，说明该镜像的实例是32位的还是64位的。在本书写作时，其中一个镜像示例的显示位置为hadoop-images/hadoop-0.19.0-i386.manifest.xml。该镜像使用Hadoop的0.19.0版本，并且可以运行在32位的EC2实例上。

有了可用的Hadoop镜像，我们就可以在`hadoop-ec2-env.sh`中设置`HADOOP_VERSION`、`INSTANCE_TYPE`和`S3_BUCKET`。除非你用的是定制的镜像，否则`S3_BUCKET`应被设为hadoop-images。`INSTANCE_TYPE`默认为m1.small，它可以很好地工作。关键是要记住实例类型直接指定了CPU是32位的还是64位的，而且必须从一个与之兼容的镜像启动（i386或x86_64）。最后，

① S3桶分区位于S3命名空间的最上层。每个桶只属于一个AWS账户，且命名空间在全局范围内必须是独一无二的。

www.ChinaDBA.net 中国DBA超级论坛

HADOOP_VERSION应该被设置为你所需的Hadoop版本。如同在ec2-describe-images命令输出中所看到的一样，HADOOP_VERSION、INSTANCE_TYPE及S3_BUCKET的特定组合必须是可用的。

9.4 在 EC2 上运行 MapReduce 程序

Hadoop的EC2工具位于Hadoop的安装目录src/contrib/ec2/bin中。如前所述，ec2-init.sh脚本已经把这个目录添加到系统PATH中。在这个目录中有一个hadoop-ec2，它是用于执行其他命令的元指令（meta-command）。要在EC2上启动一个Hadoop集群，使用

```
hadoop-ec2 launch-cluster <cluster-name> <number-of-slaves>
```

它首先启动一个主EC2实例。之后，它启动所需数目的从节点来指向主节点。当该指令返回后，它会打印出主节点的公用DNS名，我们将之表示为<master-host>。这里，不是所有的从节点都需要完全启动。你可以查看位于http://<master-host>:50030/的主节点的JobTracker网页界面来监视集群以及从节点的运行状态。

注意 EC2的新用户不能运行超过20个并发实例。你可以在http://www.amazon.com/gp/html-forms-controller/ec2-request填写Amazon EC2 Instance Request Form（实例请求表），申请提高上限。

在启动了一个Hadoop集群之后，你就可以登录到主节点并使用这个集群，就像你把它安装在自己的计算机上一样。为了登录，使用如下命令：

```
hadoop-ec2 login <cluster-name>
```

Hadoop EC2实例的$HADOOP_HOME为/usr/local/hadoop-x.y.z，其中x.y.z为Hadoop的版本号。我们做一个快速测试，来显示Hadoop正在集群上运行。

```
# cd /usr/local/hadoop-*
# bin/hadoop jar hadoop-*-examples.jar pi 10 10000000
```

在本章剩下的部分中，命令行前的井号（#）字符代表该行是在Hadoop EC2集群的主节点而非本地计算机上运行。上述命令运行一个Hadoop示例程序来估算pi的值。你可以在http://<master-host>:50030/上监控这个作业。

9.4.1 将代码转移到 Hadoop 集群上

所有的Hadoop应用都包含两个组件：代码和数据。我们首先移动代码到Hadoop集群上。下一节将讨论让我们的数据变得可以被访问（或许会涉及移动数据到集群）。

用scp可以将应用代码复制到Hadoop EC2集群的主节点上。从本地计算机上执行如下的命令：

```
source hadoop-ec2-env.sh
scp $SSH_OPTS <local-filepath> root@$MASTER_HOST:<master-filepath>
```

这里<local-filepath>指向本地计算机上的应用代码，而<master-filepath>指向主节点上的目标文件路径。

www.ChinaDBA.net 中国DBA超级论坛

9.4.2　访问 Hadoop 集群上的数据

因为Hadoop EC2集群是租用的，存储在集群（包括HDFS）上的数据并不是持久保存的。你的输入数据必须持久化到其他一些地方，并在处理时被放入EC2集群。这里有许多方法可供选择，而它们都各具优势与不足。

1. 直接移动数据到HDFS

当输入数据较小（<100 GB）并且仅处理一次时，最简单的办法是先把数据复制到主节点，再从主节点复制到集群上的HDFS。向主节点复制数据与向它复制应用程序的代码（见9.4.1节）没有区别。一旦数据已经在主节点上，你可以登录主节点并用标准的Hadoop命令把数据复制到HDFS中。

```
# bin/hadoop fs -put <master-filepath> <hdfs-filepath>
```

图9-6图形化地展示了数据流的通路。在这个数据流通路中有一些值得注意的地方。其中一点是AWS与外部通信的带宽是需要付费的（每个EC2实例每小时的费用另算），但是AWS内部的带宽是免费的。在这个案例中，复制数据到主节点会收取费用，而从主节点到HDFS是免费的。（在MapReduce处理过程中的通信，以及MapReduce和HDFS之间的通信都是免费的。）无论用哪种方法将数据放入Hadoop集群，都需至少支付一次网络带宽费用。复制数据到主节点所花费的时间可能会相对比较长，因为在你的计算机和AWS之间的联结会比AWS内部的联结慢得多。而且，无论你采用什么样的数据流机制，都避免不了至少出现一次这种时间损耗。目前的数据流架构的问题在于每当你启用Hadoop集群时，都会带来时间开销和费用支出。如果输入的数据会在多个会话中用不同的方式处理，不推荐使用这种数据流。

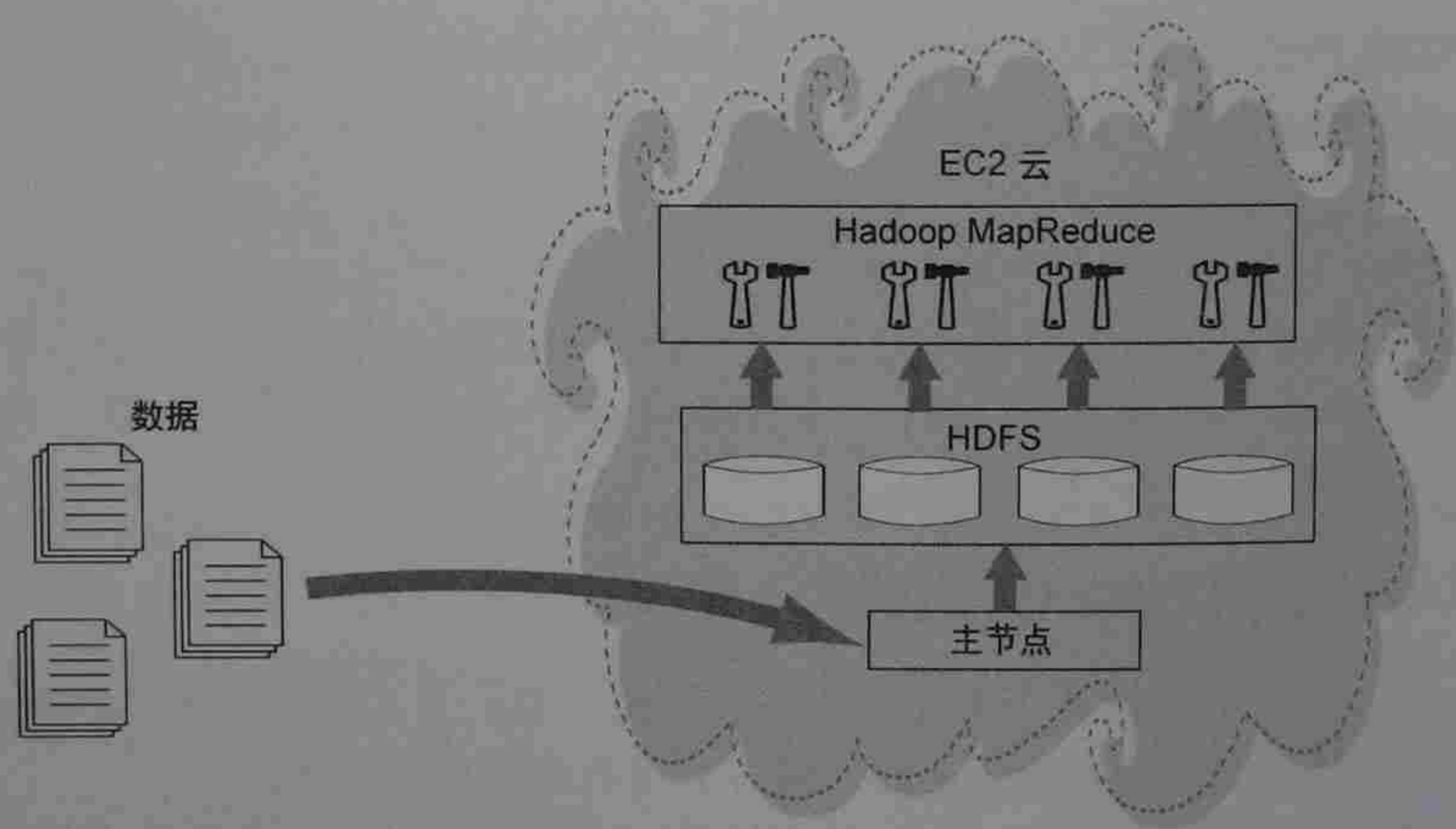


图9-6　直接传输数据到Hadoop EC2云上

现有的数据流通路的另一个缺点是对输入数据有大小限制。所有的数据必须能够先驻留在主节点上，而一个小的EC实例只有150 GB的磁盘分区。这个局限也可以被打破，但需要把输入数据划分为几块，每次传送一块。你还可以选择使用拥有多个420 GB磁盘分区的更大的实例。但是，

www.ChinaDBA.net 中国DBA超级论坛

在尝试这些更复杂的机制之前，应该考虑一下在数据路径中使用S3。

2. 通过S3移动数据到HDFS

S3是AWS提供的一个云存储服务。你实际已经看到S3可以存储EC2的镜像。在S3中存储数据需要支付带宽费，用于和非AWS计算机进行数据I/O，还要支付一个以数据规模按月计费的存储费。该存储服务的收费模式对许多应用非常有吸引力。更特别的是，它非常适合与Hadoop EC2集群一起使用。

在图9-7中可以看到这种数据流模式。与图9-6相比，主要的改变在于输入数据首先被传输到S3云上，而不是到主节点。注意，与主节点不同，S3云存储独立于Hadoop EC2集群存在。你可以在一段时间内创建并终止多个Hadoop EC2集群，而它们均可以从S3中读取相同的输入数据。这个安装方式的优点在于，资金与时间成本仅花费一次在向AWS复制输入数据时。在输入数据被复制到S3之后，因为S3和EC2都在AWS系统内部，从S3云复制到集群的HDFS是非常快的，而且免费。虽然现在增加了一个额外的用于在S3上存储输入数据的月度支出，但是它通常是很小的。如果你需要有一个可扩展的归档存储空间，在此该数据流架构下，S3也可以胜任这个角色，并进一步体现其花费模型的优势。

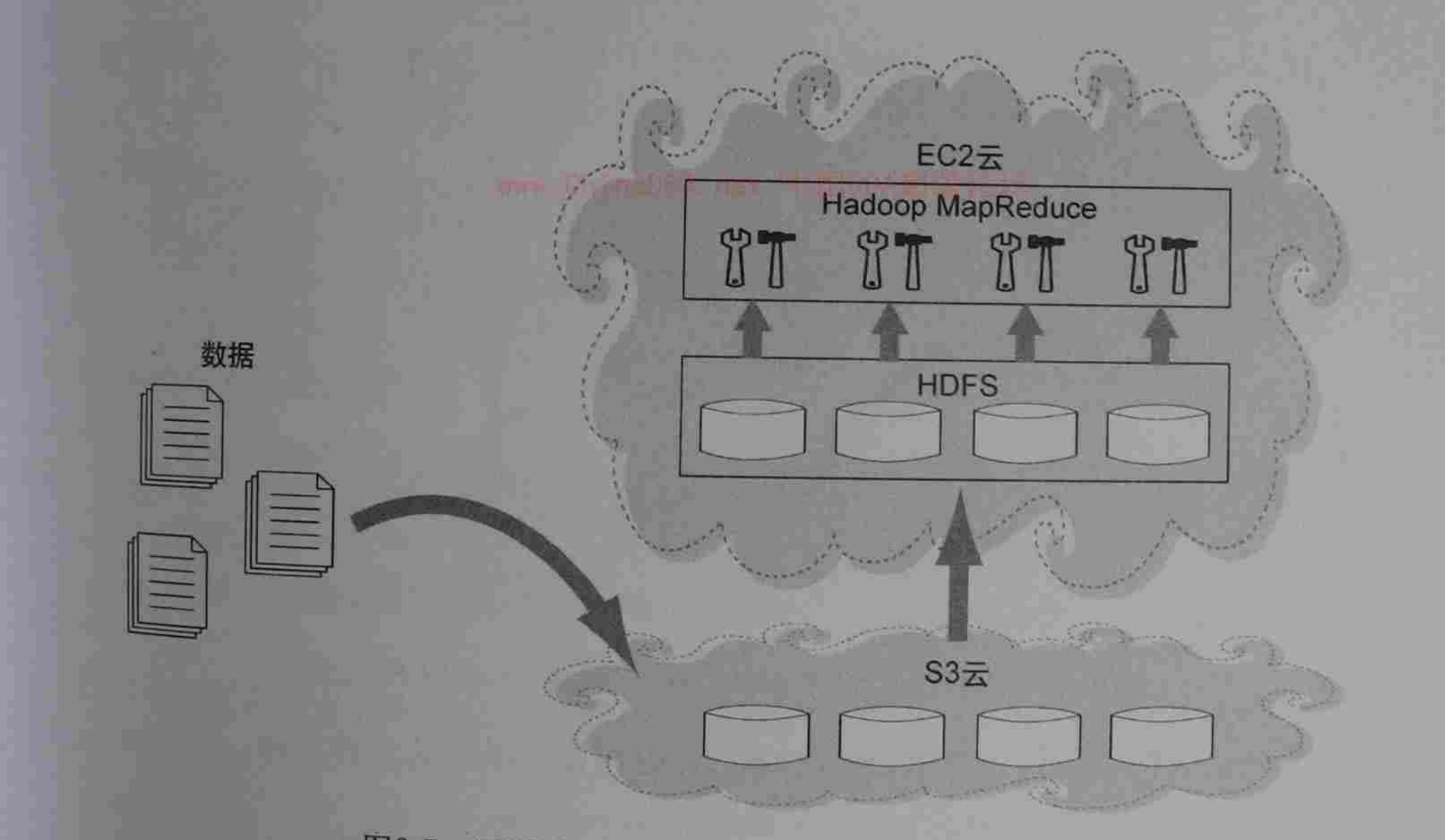


图9-7 同时通过S3和HDFS使用AWS上的Hadoop

默认的Hadoop安装内置支持使用S3。有一个用于S3的特殊文件系统，称为S3 Block FileSystem，架构在S3之上来支持大文件。（S3将文件大小限制为5 GB。）你需要将S3 Block FileSystem看做是一个区别于S3的独立文件系统，就像HDFS区别于底层的Unix文件系统一样。

S3 Block FileSystem需要一个专用的S3桶分区。创建了这个S3桶分区之后，你就可以从本地计算机向S3移动数据了：

www.ChinaDBA.net 中国DBA超级论坛

```
bin/hadoop fs -put <local-filepath>
➥s3://<access-key-id>:<secret-access-key>@<s3-bucket>/<s3-filepath>
```

回顾9.2.1节介绍过<access-key-id>和<secret-access-key>是认证参数，而<s3-bucket>是你为S3 Block FileSystem创建的S3桶分区的名字。

当数据转移S3上之后，可以复制它到任何Hadoop EC2集群。在集群的主节点上，运行：

```
# bin/hadoop distcp s3://<access-key-id>:<secret-access-key>@<s3-bucket>/
➥<s3-filepath> <hdfs-filepath>
```

数据复制到HDFS之后，你就可以按照通常的方式在集群中运行Hadoop程序。

3. 直接访问S3中的数据

迄今为止，我们总是在运行Hadoop应用之前把数据复制到集群的HDFS上。这样保证了存储和MapReduce程序之间的数据局部性。对于非常小的作业，你可以选择不用HDFS而放弃数据局部性，省掉从S3到HDFS的数据复制过程。数据通路如图9-8所示。

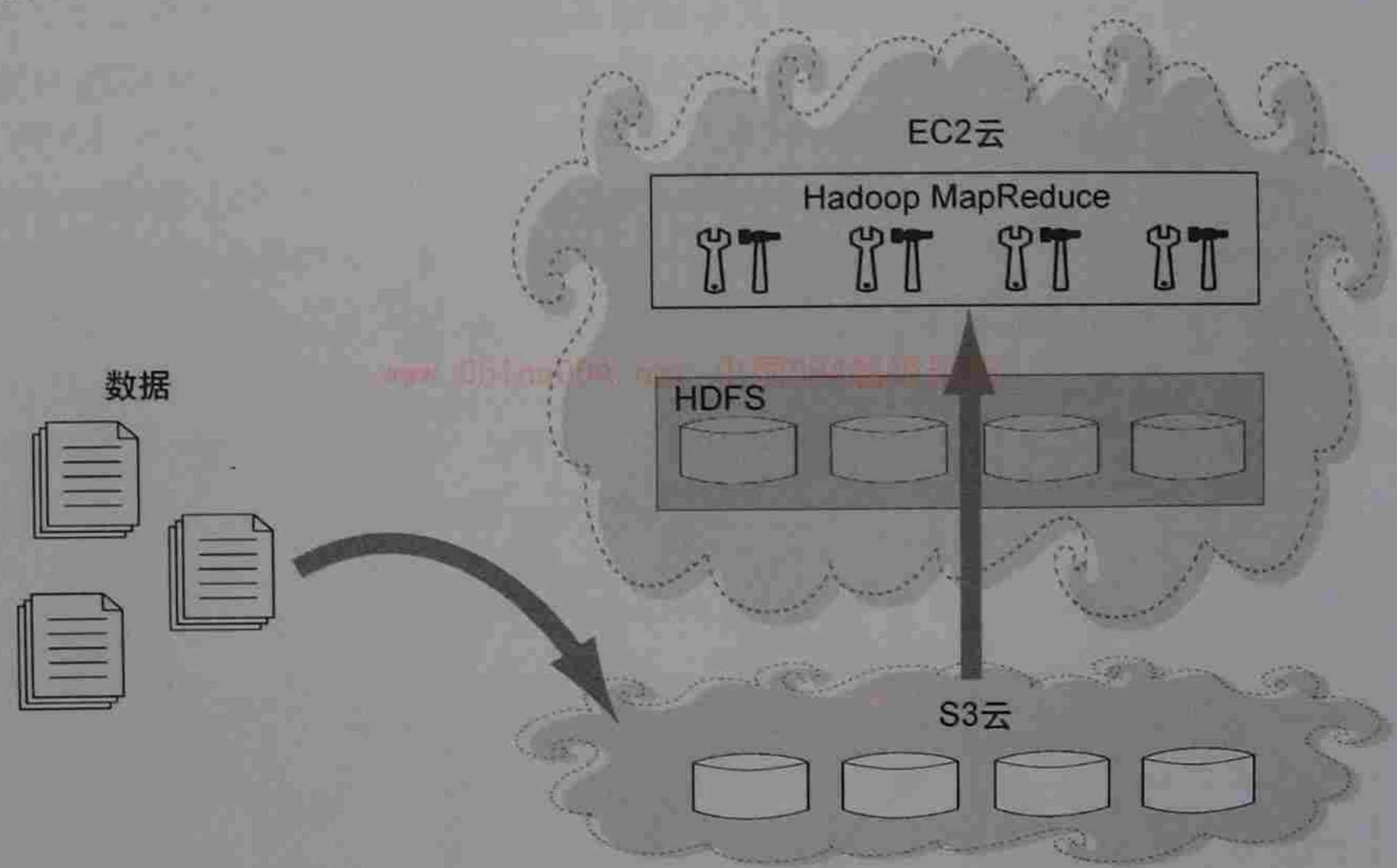


图9-8　运行在EC2上的Hadoop可以直接访问S3上的数据

为了在此体系结构下工作，在执行Hadoop应用时指定S3为输入文件路径

```
# bin/hadoop jar <app-jar-filepath> s3://<access-key-id>:
➥<secret-access-key>@<s3-bucket>/<s3-filepath> <hdfs-filepath>
```

上面的指令将输出文件存储到HDFS中，但是你也可以将它改为S3。

4. 在Hadoop中使用S3的更多选择

关于S3的使用方法还有一些不同的选择，在某些场景下会很有用。

迄今为止，我们已经使用了一个特殊的Hadoop S3文件系统（S3 Block FileSystem）在S3中存储数据。另一个选择是使用S3的本地文件系统。本地文件系统的主要缺点是它限制文件大小为5 GB。如果输入数据文件小于这个限制，本地文件系统会是一个绝好的选择。它与所有的标准

www.ChinaDBA.net 中国DBA超级论坛

S3工具兼容，而Hadoop的S3文件系统采用的是特殊的数据格式。标准的S3工具使得S3本地文件系统的使用更加透明且更容易理解。要使用S3本地文件系统而非S3 Block FileSystem，需要在指定文件位置时用s3n替代s3。例如，使用

```
s3n://<access-key-id>:<secret-access-key>@<s3-bucket>/<s3-filepath>
```

替代

```
s3://<access-key-id>:<secret-access-key>@<s3-bucket>/<s3-filepath>
```

如果经常使用S3，你会发现为每个要访问的文件敲入一个长长的URI是非常麻烦的。缩短URI的一个办法是在conf/core-site.xml文件中增加如下内容：

```
<property>
  <name>fs.s3.awsAccessKeyId</name>
  <value>AWS_ACCESS_KEY_ID</value>
</property>

<property>
  <name>fs.s3.awsSecretAccessKey</name>
  <value>AWS_SECRET_ACCESS_KEY</value>
</property>
```

注意，必须将AWS_ACCESS_KEY_ID替换为20个字符的Access Key ID，将AWS_SECRET_ACCESS_KEY替换为40个字符的Secret Access Key。增加这两个属性到core-site.xml之后，S3文件的URI缩减为

```
s3://<s3-bucket>/<s3-filepath>
```

或

```
s3n://<s3-bucket>/<s3-filepath>
```

成为S3的本地文件系统。

注意 如果你不幸使用了一个包含斜杠的Secret Access Key，就无法把密钥嵌入到URI中。使用这个AWS/S3账户的唯一途径是用如前所述的方法把密钥放在core-site.xml中。(一些文档建议将密钥中的斜杠在URI中替换为字符串*%2F*，但是这在实际中并不好用。)

另一个便捷的办法是把S3作为你的默认文件系统来替代HDFS。为此，在增加了上述两个属性之后，将conf/coresite.xml中的fs.default.name属性改为

```
<property>
  <name>fs.default.name</name>
  <value>s3://S3_BUCKET</value>
</property>
```

这里的S3_BUCKET是被你选择作为Hadoop S3 Block FileSystem使用的S3的桶分区（我们在前面表示为<s3-bucket>）。

9.5 清空和关闭 EC2 实例

默认情况下，Hadoop把其作业的输出数据存储到集群的HDFS中，而你应该将它持久化保存

www.ChinaDBA.net 中国DBA超级论坛

到其他地方。提取输出数据的方法与将输入数据复制到Hadoop EC2集群上的方法相同，只是方向相反。主要的区别在于输出数据通常比输入数据的规模小几个数量级。一般而言，考虑到输出数据较小，从主节点复制可能是最佳选择。

因为你以小时为单位从AWS租用EC2实例，当工作完成后关闭实例并告诉AWS停止计费就非常重要。很容易犯的错误是退出登录后忘记实例仍在运行，而你依然在付费！为了正常地终止集群，使用如下的命令：

```
bin/hadoop-ec2 terminate-cluster <cluster-name>
```

所有集群中的EC2实例将关闭，而在上面的数据将会丢失。不需要做进一步的清理工作。

9.6　Amazon Elastic MapReduce 和其他 AWS 服务

Amazon Web Services仍在不断增加新的功能，许多功能都旨在让Hadoop开发者感觉更加方便。在本书写作时，两个最新推出的服务为Amazon Elastic MapReduce (EMR)和AWS导入/导出。

9.6.1　Amazon Elastic MapReduce

只需额外投入很小的费用，EMR服务就会启动一个预置的Hadoop集群供你运行MapReduce程序。这个服务所提供的主要的简化是让你无需顾及EC2实例的安装，因此也不必处理所有的认证及命令行工具等事务。你与EMR的通信完全通过一个Web终端完成，地址为https://console.aws.amazon.com/elasticmapreduce/home。可以在图9-9中看到它的初始界面。

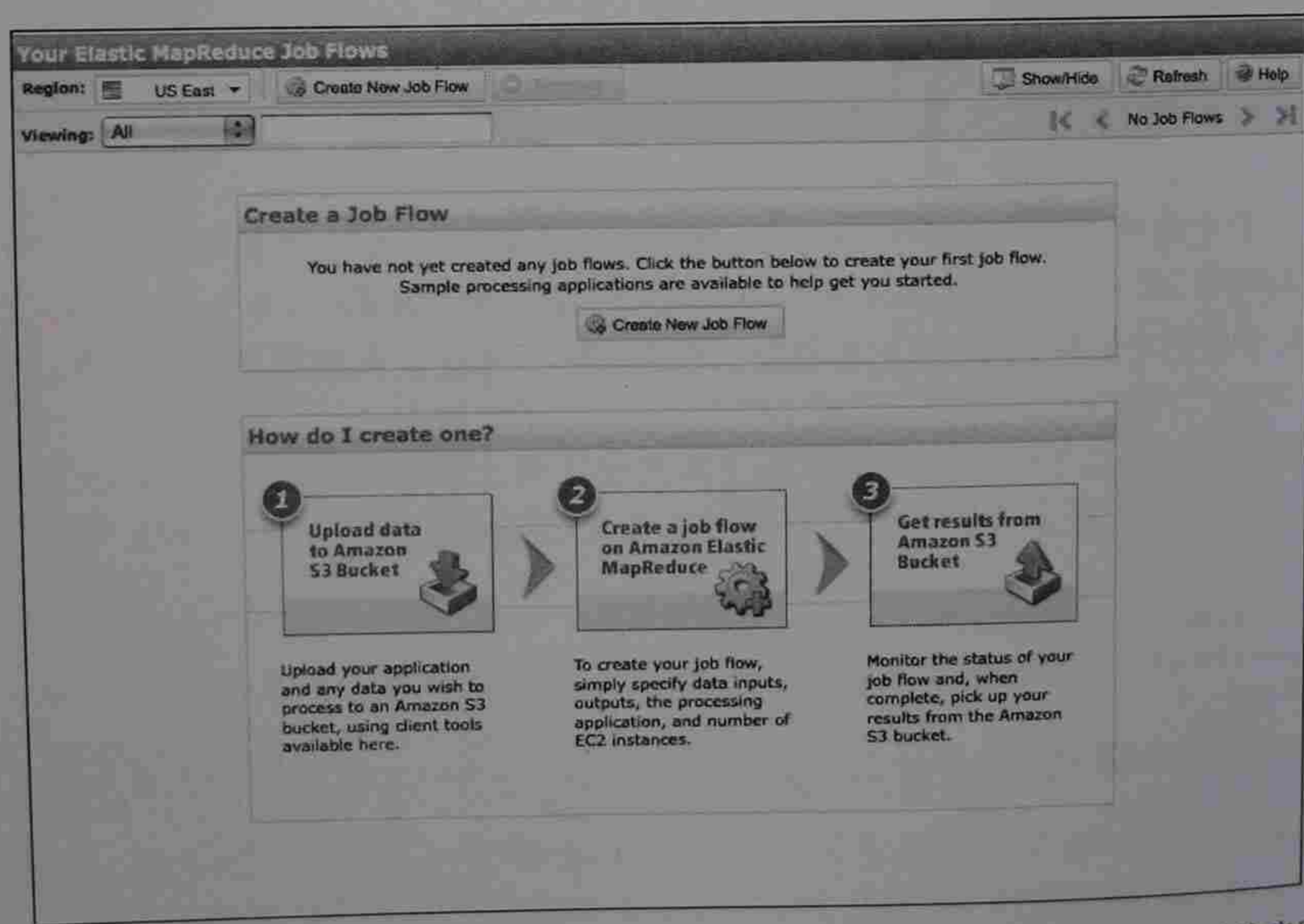


图9-9　Amazon Elastic MapReduce的Web终端初始界面。你可以按照屏幕上显示的步骤创建一个作业流

www.ChinaDBA.net 中国DBA超级论坛

这个设计旨在处理单一的作业。以一个（Streaming、Pig或Hive）脚本或JAR文件的方式提交一个MapReduce作业，EMR就会配置一个集群运行它。默认情况下，集群在作业结束后关闭。作业的输入（输出）直接从S3读取（写入）。如9.4.2节所述，重量级的Hadoop用户通常有许多处理相同数据的作业，会使得这种配置相对低效。但是轻量级用户会发现使用EMR极大地简化了在云上运行MapReduce的过程。而且，不难想象，未来EMR还会变得更加复杂，最终会成为在AWS运行Hadoop的主要方式。

你可以在如下站点找到关于Amazon Elastic MapReduce的更多信息：

http://aws.amazon.com/elasticmapreduce/；

http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/。

9.6.2 AWS 导入/导出

在云上处理大规模数据的一个主要瓶颈在于很难将大规模的数据集搬移到云上。如果你已经将数据存储在S3上，那么比较直接的方法就是在EC2上运行Hadoop来访问这个数据。另一方面，如果你只是为了分析而把数据搬移到Amazon云上，那么数据传输本身可能是一个很大的障碍。利用Amazon推出的AWS导入/导出服务，你可以直接把硬盘交给他们，由他们来使用高速的内部网络把数据上传到S3上。这个服务是否合理取决于你的互联网连接速度。表9-2为AWS给出的大致参考依据。

表9-2 判断AWS导入/导出比互联网上载更有效的数据点

可用的互联网连接	理论上80%网络利用率时传输1TB数据所需的最小天数	考虑使用AWS导入/导出的条件
T1 (1.544 Mbps)	82天	100 GB或更多
10 Mbps	13天	600 GB或更多
T3 (44.736 Mbps)	3天	2 TB或更多
100 Mbps	1～2天	5 TB或更多
1000 Mbps	少于1天	60 TB或更多

你可以在http://aws.amazon.com/importexport/中找到关于AWS导入/导出的更多信息。

9.7 小结

云基础设施是一个运行Hadoop的好地方，因为它允许你轻松地扩展到上百个节点，并给你经济上的灵活性以规避前期的投资。Hadoop天生支持Amazon Web Services（AWS）。本章一开始讲述了AWS的账户设置和计算服务租用，而在你准备好了从AWS租用计算节点时，就会有Hadoop工具来帮你自动地完成Hadoop集群的设置与运行。AWS还有一个云存储服务（S3），可用于与HDFS协同工作或者替代HDFS。你会发现不同设置各有利弊。最后，重要的是记住用完后要关闭Hadoop集群。你是按小时租用这个云基础设施，除非你明确地关机，否则费用就会不断累积。

www.ChinaDBA.net 中国DBA超级论坛

第 10 章 用Pig编程

本章内容

- 安装Pig并使用Grunt Shell
- 理解Pig Latin语言
- 从用户定义函数扩展Pig Latin语言
- 使用一个简单的Pig latin脚本有效地计算近似文档

经常有人抱怨用MapReduce很难编程。以前在第一次考虑一个数据处理任务时，会以数据流操作的形式来思考，如循环和过滤。但是，当用MapReduce实现一个程序时，必须在mapper和reducer函数以及作业链的层次思考。一些在高级语言中被视为运作一流的函数，在MapReduce中却要费力去实现，如我们在第5章所看到的join。Pig是Hadoop的一个扩展，它简化了Hadoop的编程，提供了一个高级数据处理语言，并且保持了Hadoop易于扩展与可靠的特征。Yahoo是Hadoop的一个重量级用户（也是Hadoop Core和Pig的后台支持者），它有40%的Hadooop作业是使用Pig运行的。Twitter是另一个有名的Pig用户[①]。

Pig有两个主要的组成部分：

(1) 高级数据处理语言Pig Latin；

(2) 依据可供选择的评价机制编译与运行Pig Latin脚本的编译器。主要的评价机制是Hadoop。Pig也支持面向开发的本地模式。

Pig能够简化编程缘自于Pig Latin可以很容易地表达你的代码。编译器可以自动地帮助你在脚本中发掘出优化的空间。这让你从手动优化程序中摆脱出来。随着Pig编译器的改进，Pig Latin程序的运行速度也会自动随之提高。

10.1 像 Pig 一样思考

Pig的设计有其特定的理念。所有Hadoop子项目都寄望于获得易用性、高性能和大规模的扩展能力。有3个要素对于理解Pig极为独特和重要，即Pig在编程语言上的设计选择（一种称为Pig Latin的数据流语言）、Pig所支持的数据类型，以及它把用户定义函数（user-defined functions，UDF）

① http://www.slideshare.net/kevinweil/hadoop-pig-and-twitter-nosql-east-2009。

www.ChinaDBA.net 中国DBA超级论坛

视为“一等公民”。

10.1.1 数据流语言

编写Pig Latin程序包含一系列的步骤，每个步骤都是一个单独的高级数据转换。这个转换支持关系型操作，诸如filter（过滤）、union（联合）、group（组合）和join（联结）。一个处理查询日志的Pig Latin例程可能是这样的

```
log = LOAD 'excite-small.log' AS (user, time, query);
grpd = GROUP log BY user;
cntd = FOREACH grpd GENERATE group, COUNT(log);
DUMP cntd;
```

即便操作是关系型的，Pig Latin仍然是一个数据流语言。数据流语言对根据算法来思考的程序员而言更为友好，算法用数据和控制流表达起来更为自然。另一方面，像SQL这样的声明式语言有时对分析师而言更为好用，他们喜欢只陈述从程序中所期望得到的结果。Hive是另外一个Hadoop子项目，面向的正是那些更喜欢SQL的用户。我们将在第11章学习Hive的相关细节。

10.1.2 数据类型

我们可以把Pig在数据类型上的设计理念总结为一句口号：“Pig吃任何东西。”输入数据可以来自于任何格式。Pig天生支持那些流行的格式，如用制表符分隔的文本文件。用户也可以增加函数来支持其他的数据文件格式。Pig不需要元数据或者数据的schema，但如果有，它也可以利用它们。

Pig可以操作relational（关系）、nested（嵌套）、semistructured（半结构化）或unstructured（非结构化）类型的数据。为了支持数据的多样性，Pig还支持复杂的数据类型，如bags（包）和tuples（元组），它们可以嵌套形成相当复杂的数据结构。

10.1.3 用户定义函数

Pig是面向多种应用程序而设计的——处理日志数据、自然语言处理、分析网络图等。这些计算大多都需要定制化的处理。Pig采用自底向上的架构，支持用户定义函数（UDF，user-defined functions）。了解如何编写UDF是学习使用Pig的重要一环。

10.2 安装 Pig

你可以从http://hadoop.apache.org/pig/releases.html下载Pig的最新版本。本书写作时，Pig的最新版本为0.4和0.5，它们都需要Java 1.6。它们之间的主要区别在于Pig 0.4面向Hadoop 0.18，而Pig 0.5面向Hadoop 0.20。按照惯例，请务必将JAVA_HOME设置为Java安装的根目录，而Windows用户应该安装Cygwin。Hadoop集群应当已经被安装好。理想情况下，它是采用全分布模式的真实集群，但是在实际中采用伪分布模式也是可以的。

下载的发布包解压后将Pig安装在本地计算机上。在Hadoop集群上无需修改任何东西。可以

www.ChinaDBA.net 中国DBA超级论坛

认为Pig发布包中含有一个编译器和一些开发和部署工具。它增强了MapReduce的编程手段，但是与运行态Hadoop集群之间的关系并不紧密。

在Pig解压缩后的目录下，你应该创建子目录logs和conf（除非已经有了）。Pig将从conf下的文件中提取定制化的配置。如果你刚刚新建了conf目录，显然其中还没有配置文件，你需要在conf下放置一个名为pig-env.sh的新文件。当运行Pig时这个脚本会被执行，并且它会被用来设置Pig的环境变量。除了JAVA_HOME，需要特别关注的环境变量为PIG_HADOOP_VERSION和PIG_CLASSPATH。通过设置这些变量，可以将集群的情况告知Pig。例如，如下在pig-env.sh中的语句会告诉Pig当前集群所使用的Hadoop版本为0.18，并且将Hadoop本地安装的配置目录添加到Pig的classpath中。

```
export PIG_HADOOP_VERSION=18
export PIG_CLASSPATH=$HADOOP_HOME/conf/
```

我们假定HADOOP_HOME被设置为本地计算机上Hadoop的安装目录。通过将Hadoop的conf目录添加到Pig的classpath中，Pig可以自动找到Hadoop集群中NameNode和JobTracker的位置。

你也可以不使用Pig的classpath，而通过创建一个名为pig.properties的文件来指定Hadoop集群的位置。这个属性文件将位于先前创建的conf目录下。它应该定义fs.default.name和mapred.job.tracker，即文件系统（HDFS的NameNode）以及JobTracker的位置。一个指向伪分布模式Hadoop集群的pig.properties示例如下

```
fs.default.name=hdfs://localhost:9000
mapred.job.tracker=localhost:9001
```

方便起见，我们把Pig的安装目录bin添加到路径中。假定$PIG_HOME指向你的Pig安装目录：

```
export PATH=$PATH:$PIG_HOME/bin
```

由于在命令行路径中包含Pig的bin目录，你可以用命令pig来启动Pig。首先，可以先看看它的使用选项：

```
pig -help
```

让我们启动Pig的交互式shell，可以看到它在恰当地读取配置信息。

```
pig
2009-07-11 22:33:04,797 [main] INFO
➥ org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
➥ Connecting to hadoop file system at: hdfs://localhost:9000
2009-07-11 22:33:09,533 [main] INFO
➥ org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
➥ Connecting to map-reduce job tracker at: localhost:9001
grunt>
```

由Pig报告的文件系统和JobTracker应该与配置信息的设置一致。现在就进入了Pig的交互式shell，它也被称为Grunt。

10.3　运行 Pig

我们可以采用3种方式来运行Pig Latin命令——通过Grunt交互shell、脚本文件和嵌在Java程

www.ChinaDBA.net 中国DBA超级论坛

序中的查询语句。每种方法都有两种模式——本地模式和Hadoop模式。（在Pig文档中Hadoop模式有时被称为MapReduce模式。）在上一节的最后，我们已经通过Grunt shell运行了Hadoop模式。

Grunt shell允许你手工地输入Pig命令。这种方式主要用在即席数据分析或者程序开发的反复调试阶段。大型的Pig程序以及需重复运行的程序是通过脚本文件运行的。进入Grunt，要输入命令pig。而要执行一个Pig脚本，运行相同的pig命令的同时要带上文件名作为参数，如pig myscript.pig。Pig脚本通常使用.pig作为扩展名。

你可以认为Pig程序是与SQL查询类似的，Pig提供了一个PigServer类，允许任意的Java程序执行Pig查询。从概念上看，这类似于使用JDBC来执行SQL查询。嵌入式Pig程序是一个相当高阶的话题，你可以在http://wiki.apache.org/pig/EmbeddedPig找到更多的细节描述。

当在本地模式下运行Pig时，你根本不会用到Hadoop[1]。Pig命令被编译为运行在它们自己本地的JVM上，访问本地文件。其目的主要用于开发，你可以通过在本地运行一个小的开发数据集来快速得到反馈。在Hadoop模式下运行Pig意味着编译的Pig程序会实际运行在安装的Hadoop上。通常Hadoop安装为全分布式集群模式。（我们在10.2节所使用的伪分布式Hadoop设置纯粹是为了演示。除非为了调试配置信息，很少会使用。）命令pig的执行模式是由选项-x或-exectype所设定的。你可以通过如下命令进入本地模式的Grunt shell：

```
pig -x local
```

进入Hadoop模式的Grunt shell为：

```
pig -x mapreduce
```

或者使用不带参数的pig命令，默认它会选择Hadoop模式。

管理 Grunt shell

除了运行（将在后面的小节讨论）Pig Latin语句之外，Grunt shell支持一些基本的实用命令[2]。键入help会打印出一屏关于这些实用命令的帮助信息。键入quit可以退出Grunt shell。键入kill命令后加上Hadoop作业的ID可以终止该Hadoop作业。有些Pig的参数可以通过set命令设置。例如：

```
grunt> set debug on
grunt> set job.name 'my job'
```

参数debug声明是否打开或者关闭调试级别的日志。参数job.name取一个单引号字符串作为Pig程序的Hadoop作业名。这可以允许你为Pig作业取一个有意义的名字，从而更容易地在Hadoop的Web界面上找到它很有用。

Grunt shell还支持文件级的实用命令，如ls和cp。你可以在表10-1中看到实用命令与文件命令的完整列表。这些文件命令大体上是HDFS文件系统shell命令的一个子集，它们的用法是不言自明的。

---

① 有计划要改变Pig，使得它甚至可以在本地模式下使用Hadoop，可以让一些程序的编制更为一致。这个话题的讨论出现在https://issues.apache.org/jira/browse/PIG-1053。

② 从技术角度看，它们仍被认为是Pig Latin命令，但是你不太会在Grunt shell之外用到它们。

www.ChinaDBA.net 中国DBA超级论坛

表10-1　Grunt shell中的实用命令和文件命令

实用命令	help quit kill *jobid* set debug [on|off] set job.name '*jobname*'
文件命令	cat, cd, copyFromLocal, copyToLocal, cp, ls, mkdir, mv, pwd, rm, rmf, exec, run

有两个新的命令：exec和run。它们在Grunt shell的内部运行Pig脚本，这有助于Pig脚本的调试。命令exec执行Pig脚本的空间在Grunt shell之外。在脚本中定义的别名对shell是不可见的，反之亦然。命令run执行Pig脚本的空间与Grunt shell相同（也被称为交互模式）。它和手动将脚本逐行输入Grunt Shell的效果是一样的。

10.4　通过 Grunt 学习 Pig Latin

在正式地描述Pig的数据类型和数据处理操作之前，让我们在Grunt shell下运行几个命令来感觉一下如何在Pig中处理数据。出于学习的目的，更简便的方法是在本地模式下运行Grunt。

```
pig -x local
```

你可能希望首先尝试一些文件命令，如pwd和ls，来让自己熟悉一下文件系统。

让我们查看一些数据。稍后会重新用到我们在第4章所介绍的专利数据，但现在让我们从Excite搜索引擎中找一些有趣的查询日志作为数据集。在Pig安装时就自带了这个数据集，位于Pig安装目录下的tutorial/data/excite-small.log文件中。数据分为3列，中间按制表符分隔。第1列为匿名化的用户ID。第2列为Unix的时间戳，而第3列为查询记录。我们从该文件的4500个记录中选取一段样本如下：

```
3F8AAC2372F6941C   970916093724   minors in possession
C5460576B58BB1CC   970916194352   hacking telenet
9E1707EE57C96C1E   970916073214   buffalo mob crime family
06878125BE78B42C   970916183900   how to make ecstacy
```

在Grunt中，输入如下的语句会将数据装载到一个称为log的“别名”（即变量）。

```
grunt> log = LOAD 'tutorial/data/excite-small.log' AS (user, time, query);
```

注意在你输入这个语句之后，似乎什么也没有发生。在Grunt shell中，Pig解析你的语句，但是不会去实际执行它们，直到你使用DUMP或STORE命令来请求其结果。DUMP命令打印出一个别名的内容，而STORE命令将内容存在一个文件中。在你显式地请求某些最终结果前，Pig都不会实际执行任何命令，这很有意义，因为你知道我们正在处理的是大规模数据集。没有足够的内存空间用于“装载”数据，而且任何情况下，我们希望在花费时间和资源去实际执行它之前，验证执行计划的逻辑是正确的。

我们通常仅在开发时使用DUMP命令。而更多时候会将庞大的结果STORE到一个目录中。（类似Hadoop，Pig会自动将数据分割到以part-nnnnn命名的文件中。）当DUMP一个别名时，你应该确认它的内容足够小，可以很好地打印到屏幕上。通常的做法是通过LIMIT命令生成另一个别名，

www.ChinaDBA.net 中国DBA超级论坛

而DUMP这个新的、更小的别名。LIMIT命令允许你指定有多少元组（行）用于返回。例如，要查看log的4个元组，则

```
grunt> lmt = LIMIT log 4;
grunt> DUMP lmt;
(2A9EABFB35F5B954,970916105432L,+md foods +proteins)
(BED75271605EBD0C,970916001949L,yahoo chat)
(BED75271605EBD0C,970916001954L,yahoo chat)
(BED75271605EBD0C,970916003523L,yahoo chat)
```

表10-2总结了在Pig Latin中的读/写操作符。LIMIT在技术上并不是一个读操作符或者写操作符，但是由于它经常被一起使用，我们也在表中列出。

表10-2 在Pig Latin中的读/写操作符

LOAD	alias = LOAD 'file' [USING function] [AS schema]; 从文件中装载数据到一个关系。如果不用USING选项来特别指定，默认情况下使用PigStorage装载函数。数据可以使用AS选项给出shema
LIMIT	alias = LIMIT alias n; 限制元组个数为*n*。当作用于由ORDER运算处理之后的alias时，LIMIT返回前*n*个元组。否则不能保证哪些元组会被返回。LIMIT运算符难以分类，因为它肯定不是一个读/写运算符，也不是一个真正的关系运算符。我们把它列在这里，是因为稍后要讲到DUMP运算符，在使用它之前会使用到LIMIT运算符
DUMP	DUMP alias; 显示一个关系的内容。主要用于调试。这个关系应该足够小，以便可以打印在屏幕上。你可以对alias使用LIMIT操作来确保它小得可以显示
STORE	STORE alias INTO 'directory' [USING function]; 将一个关系中的数据存储到一个目录中。这个目录在该命令执行时一定不存在。Pig会生成这个目录，并把关系存储在以part-*nnnnn*为名的文件中。除非使用USING选项特别指定，默认使用PigStorage存储函数。

你可能发现装载和存储数据没有那么令人兴奋。让我们执行几个数据处理语句来看一看如何通过Grunt使用Pig Latin。

```
grunt> log = LOAD 'tutorial/data/excite-small.log'
          ➥ AS (user:chararray, time:long, query:chararray);
grunt> grpd = GROUP log BY user;
grunt> cntd = FOREACH grpd GENERATE group, COUNT(log);
grunt> STORE cntd INTO 'output';
```

上述的语句统计每个用户发起的查询个数。输出文件的内容类似于（你必须在Grunt之外来查看这个文件）：

```
002BB5A52580A8ED 18
005BD9CD3AC6BB38 18
00A08A54CD03EB95 3
011ACA65C2BF70B2 5
01500FAFE317B7C0 15
0158F8ACC570947D 3
018FBF6BFB213E68 1
```

从概念上来讲，我们已经执行了一个类似于SQL查询中的聚集操作：

```
SELECT user, COUNT(*) FROM excite-small.log GROUP BY user;
```

www.ChinaDBA.net 中国DBA超级论坛

需要指出，在Pig Latin与SQL查询之间有两个主要的不同。如前所述，Pig Latin是一种数据处理语言。它使用的是一个数据处理步骤的序列，而不是用子句组成的复杂SQL查询。另一个区别更加微妙——SQL中的关系通常有固定的schema。在SQL中，我们在数据填充之前定义关系的schema。Pig则在schema上采用更为宽松的方法。事实上，如果你不愿意，可以不必使用schema，这种情况会出现在半结构化和无结构化数据处理时。这里我们的确指定了关系log的schema，但它仅位于load语句，而且直到装载数据时才会实施。在装载操作过程中，任何不遵循schema的字段都被置为空。这种方式保证了log这个关系在后续操作中仍遵循既定的schema。

基于创建关系所用的操作，Pig会尽可能多地找出它的schema。你可以使用DESCRIBE命令将Pig的schema暴露给所有的关系。这有助于理解Pig语句在做什么。例如，我们下面会查看grpd和cntd的schema。在此之前，让我们先看看DESCRIBE命令是如何描述log的。

```
grunt> DESCRIBE log;
log: {user: chararray,time: long,query: chararray}
```

正如所料，load命令已经为log指定了确切的schema。关系log包含3个字段，名为user、time和query。字段user和query均为字符串（在Pig中为chararray），而time为一个long整型数。

在关系log中的GROUP BY操作生成关系grpd。基于这个操作和log的schema，Pig为grpd生成一个schema：

```
grunt> DESCRIBE grpd;
grpd: {group: chararray,log: {user: chararray,time: long,query: chararray}}
```

group和log为grpd中的两个字段。字段log是一个包含子字段user、time和query的包(bag)。因为我们尚未讨论Pig的类型系统和GROUP BY操作，所以这里并不期望你去理解这个schema。仅为说明Pig的关系可以拥有相当复杂的schema，而DESCRIBE可以有助于你理解正在使用的关系：

```
grunt> DESCRIBE cntd;
cntd: {group: chararray,long}
```

最后，FOREACH命令对关系grpd进行操作以输出cntd。在查看了cntd的输出之后，可知它包含两个字段——用户ID和查询个数统计。正如DESCRIBE所给出的Pig的cntd策略，它也包含两个字段。第一个名为group，取自grpd的schema。第二个字段没有名字，但其类型为long。这个字段由COUNT函数生成，这个函数不会自动提供名字，但的确是由它来告知Pig其类型必须为long。

虽然DESCRIBE可以告诉你一个关系的schema，ILLUSTRATE通过一个样例的执行，来一步步地显示Pig是如何计算这个关系的。此时，Pig试图通过模拟语句的执行来计算一个关系，但它仅使用一小部分样例数据来提高执行速度。理解ILLUSTRATE的最好办法是把它用到一个关系中。这里我们使用cntd。（输出被格式化以适应打印页面的宽度。）

```
grunt> ILLUSTRATE cntd;
-----------------------------------------------------------------------------
| log      | user: bytearray   | time: bytearray | query: bytearray      |
-----------------------------------------------------------------------------
|          | 0567639EB8F3751C | 970916161410     | "conan o'brien"       |
|          | 0567639EB8F3751C | 970916161413     | "conan o'brien"       |
|          | 972F13CE9A8E2FA3 | 970916063540     | finger AND download   |
-----------------------------------------------------------------------------
```

www.ChinaDBA.net 中国DBA超级论坛

```
-----------------------------------------------------------------------
| log      | user: chararray  | time: long    | query: chararray     |
-----------------------------------------------------------------------
         | 0567639EB8F3751C | 970916161410 | "conan o'brien"      |
         | 0567639EB8F3751C | 970916161413 | "conan o'brien"      |
         | 972F13CE9A8E2FA3 | 970916063540 | finger AND download  |
-----------------------------------------------------------------------

-------------------------------------------------------------------------------
| grpd     | group: chararray | log: bag({user: chararray,time: long,        |
|          |                  |          query: chararray})                  |
-------------------------------------------------------------------------------
         | 0567639EB8F3751C | {(0567639EB8F3751C, 970916161410,            |
         |                  |   "conan o'brien"),                          |
         |                  |  (0567639EB8F3751C,970916161413,             |
         |                  |   "conan o'brien")}                          |
         | 972F13CE9A8E2FA3 | {(972F13CE9A8E2FA3, 970916063540,            |
         |                  |   finger AND download)}                      |
-------------------------------------------------------------------------------

-------------------------------------
| cntd     | group: chararray | long |
-------------------------------------
         | 0567639EB8F3751C | 2    |
         | 972F13CE9A8E2FA3 | 1    |
-------------------------------------
```

ILLUSTRATE命令显示出在cntd之前有4次转换。每个表的标题行描述了转换之后所输出关系的schema，表的其他部分显示了样例数据。Log关系显示有两次转换。两次转换之间，数据并没有发生改变，但是schema从通用的bytearray（Pig中二进制对象的类型）变为指定的schema。对log的GROUP操作在样例数据的3个log元组上执行，得到grpd的数据。基于此我们可以推断出GROUP操作取出user字段并把它变为group字段。此外，它把在log中具有相同user值的所有元组放入grpd中的一个包中。通过ILLUSTRATE查看这个样例数据的模拟运行十分有助于理解不同的操作。最后，我们看到作用于grpd的FOREACH操作生成了cntd。看过在上述表格中grpd的数据，可以很容易推断出是COUNT()函数计算出每个包的大小。

虽然DESCRIBE和ILLUSTRATE是理解Pig Latin语句的主力军，Pig还有一个EXPLAIN命令用于显示逻辑和物理执行计划的细节。我们在表10-3中总结了这个诊断运算符。

表10-3 Pig Latin的诊断运算符

DESCRIBE	DESCRIBE alias; 显示一个关系的schema
EXPLAIN	EXPLAIN [-out path] [-brief] [-dot] [-param ...] [-param_file ...] alias; 显示用于计算一个关系的执行计划。当与一个脚本名同时使用时，例如EXPLAIN myscript.pig，它会显示该脚本的执行计划
ILLUSTRATE	ILLUSTRATE alias; 逐步显示数据如何被转换，从load命令开始直到得到最终的结果关系。为了使显示和处理可管理，只有一个（并不完全随机）输入数据的样本被用于模拟执行 不过，Pig的初始样本有时并不理想，无法让脚本生成有意义的数据，Pig就会“伪造”一些类似的初始数据，可以为alias生成结果。例如，考虑如下操作：

www.ChinaDBA.net 中国DBA超级论坛

（续）

```
A = LOAD 'student.data' as (name, age);
B = FILTER A by age > 18;
ILLUSTRATE B;
```

如果Pig为A取样的每个元组恰好都小于或等于18岁，B会为空而没有什么可"解释"的。Pig就会为A构建一些年龄大于18的元组。这样B就不再是一个空的关系，使用户能够看到脚本的工作方式

为了让ILLUSTRATE能够工作，第一步的load命令必须包含一个schema。后续的转换一定不能包含LIMIT或SPLIT运算符，也不能包含嵌套的FOREACH运算符，或者使用map数据类型（将在10.5.1节说明）。

10.5　谈谈 Pig Latin

你现在知道如何使用Grunt来运行Pig Latin语句并观察它们的执行和结果。我们可以回过头来给这个语言一个更为正式的描述。你还是应该尽可能多地使用Grunt来理解我们所给出的这些语言概念。

10.5.1　数据类型和 schema

让我们首先自下向上地看一看Pig的数据类型。Pig有6个简单的原子类型和3个复杂的类型，分别如表10-4和表10-5所示。原子类型包括数字标量以及字符串和二进制对象。对类型转换的支持与实现和常规方式相同。除非特别声明，字段默认为bytearray。

表10-4　在Pig Latin中的原子数据类型

int	Signed 32-bit integer
long	Signed 64-bit integer
float	32-bit floating point
double	64-bit floating point
chararray	Character array (string) in Unicode UTF-8
bytearray	Byte array (binary object)

3个复杂数据类型是元组（tuple）、包（bag）和映射表（map）。

在元组中的一个字段，或者在映射表中的一个值可以为空，或者为任意原子或复杂类型。这使得组建嵌套和复杂数据结构成为可能。虽然数据结构可以是任意复杂的，但其中一些可能更实用并且出现得更为频繁，而且通常嵌套的深度不会超过两级。在前面所用的Excite日志例子中，GROUP BY运算符生成了一个关系grpd，其中每个元组中有一个字段是由一个包构成的。把grpd视为每个用户的查询历史，会让这个关系所用的schema看起来更为自然。每个元组代表一个用户，并有一个字段是由用户查询所构成的包。

我们也可以自上向下地审视Pig的数据模型。在顶端，Pig Latin语句作用于关系，它是由元组构成的一个包。如果你强制让包中的所有元组包含固定数目的字段，而且每个字段有固定的原子类型，那么它的行为就像一个关系数据schema——关系为一个表，元组为行（记录），而字段为

www.ChinaDBA.net 中国DBA超级论坛

列。但是，Pig的数据模型更为强大和灵活，允许嵌套式（nested）数据类型。字段本身可以是元组、包或映射表。映射表有助于处理半结构化数据，如JSON、XML以及稀疏关系型数据。而且，在一个包中的元组不必具有相同数量的字段。这允许元组表示非结构化数据。

表10-5 Pig Latin中的复杂数据类型

元组	(12.5,hello world,-2) 一个元组是一个顺序排列的字段集合。在关系中它经常被作为一行。其表现形式为以逗号分隔的字段，前后用括号包裹
包	{(12.5,hello world,-2),(2.87,bye world,10)} 包是一个无序的元组集合。关系是一种特殊的包，有时也称为外部包。内部包是在某些复杂类型中的一个字段 包表示为逗号分隔的元组，全部由大括号包裹 在一个包中的元组不必有相同的schema，甚至字段数也不必相同。不过，最好让它们相同，除非你处理的是半结构化或者非结构化数据
Map	[key#value] map是一组键/值对。键必须是唯一的并为一个字符串（chararray）。值可以是任何类型

除了为字段声明类型，schema还可以为字段命名让它们更易于引用。用户可以在LOAD、STREAM和FOREACH命令中使用AS关键字为关系定义schema。例如，在LOAD语句中为了得到Excite查询日志，我们为在log中的字段定义数据类型，并将字段命名为user、time和query。

```
grunt> log = LOAD 'tutorial/data/excite-small.log'
➥ AS (user:chararray, time:long, query:chararray);
```

在定义schema时，如果遗漏了类型，Pig会默认将bytearray作为最常用的类型。你还可以不设置字段的名字，此时该字段的状态为未命名，你只能通过位置来引用它。

10.5.2 表达式和函数

你可以将表达式和函数应用在数据字段来计算出各种数值。最简单的表达式为常数值。然后是引用一个字段的值。你可以通过名字直接引用命名字段的值，也可以通过$*n*引用一个未命名的字段，这里*n*是其在元组中的位置。（位置从0开始计数。）例如，LOAD命令通过如下的schema将命名的字段提供给log。

```
log = LOAD 'tutorial/data/excite-small.log'
      ➥AS (user:chararray, time:long, query:chararray);
```

3个命名的字段为user、time和query。例如，我们可以用time或者$1来指向时间字段，因为时间字段是log中的第2个字段（位置号为1）。假设我们希望提取时间字段作为其自身的关系；我们可以使用这个语句：

```
projection = FOREACH log GENERATE time;
```

我还可以用如下语句达成相同效果

```
projection = FOREACH log GENERATE $1;
```

大部分时间你应该为字段命名。通过位置引用字段的用处之一是为处理非结构化数据。

www.ChinaDBA.net 中国DBA超级论坛

当使用复杂的类型，你可以使用点表示法来引用嵌在元组或包中的字段。例如，回顾以前，我们利用用户ID对Excite日志进行组合，得到一个具有嵌套schema的关系grpd。

```
| grpd        | group: chararray | log: bag({user: chararray,time: long,
|             |                  |          query: chararray})
| ------------------------------------------------------------------------
|             | 0567639EB8F3751C | {(0567639EB8F3751C, 970916161410,
|             |                  |   "conan o'brien"),
|             |                  |  (0567639EB8F3751C,970916161413,
|             |                  |   "conan o'brien")}
|             | 972F13CE9A8E2FA3 | {(972F13CE9A8E2FA3, 970916063540,
|             |                  |   finger AND download)}
| ------------------------------------------------------------------------
```

grpd中的第2个字段名为log，采用包类型。每个包中的元组包含3个命名的字段：user、time和query。在这个关系中，当作用于第一个元组时，log.query会指向“conan”“o'brien”的两个副本。你可以用log.$2得到相同的字段。

引用映射表内部的字段是通过井号运算符，而非点运算符。对一个名为m的映射表，与键k相关的值通过m#k来引用。

能够引用数值仅仅是第一步。Pig支持标准算术、比较、条件、类型转换和布尔表达式，它们常见于大多数流行的编程语言中。如表10-6所示。

www.ChinaDBA.net 中国DBA超级论坛

表10-6　Pig Latin中的表达式

Constant	12, 19.2,'hello world'	常数值，如19或者“hello world”。没有小数点的数值被视为int型，如果数字之后带有l或L则为long型。带有小数点的数值被视为double型，如果数字之后带有f或F，则为float型
Basic arithmetic	+,-,*,/	加、减、乘和除
Sign	+x, -x	负号（-）改变一个数字的符号
Cast	(t)x	将x的值转换为t类型
Modulo	x % y	x被y除的余数
Conditional	(x ? y : z)	如果x为真则返回y,否则为z。该表达式必须用圆括号包括
Comparison	==,!=,<,>,<=,>=	等于，或不等于，大于，小于等
Pattern matching	x matches regex	与字符串x匹配的正则表达式。使用Java的正则表达式格式来指定regex（在java.util.regex.Pattern类中）
Null	x is null, x is not null	检查是否x为空
Boolean	x and y, x or y not x	布尔值：与、或、非

进一步而言，Pig还支持函数。表10-7显示了Pig的内置函数，它们大多数不言自明。我们会在10.6节讨论用户定义函数（UDF）。

www.ChinaDBA.net 中国DBA超级论坛

表10-7 Pig Latin中的内置函数

AVG	在一个单列的包中计算数字的平均值
CONCAT	连接两个字符串（chararray）或两个bytearray
COUNT	计算一个包中的元组个数。请参考提供给其他数据类型使用的SIZE
DIFF	比较一个元组中的两个字段。如果这两个字段为包，会返回那些存在于一个包中，而不在另一个包中的元组。如果两个字段为数值，会返回数值不匹配的元组
MAX	计算在一个单列包中的最大值。该列必须为一个数字类型或者一个chararray
MIN	计算在一个单列包中的最小值。该列必须为一个数字类型或者一个chararray
SIZE	计算元素的个数。对于一个包，它计算元组的个数。对于一个元组，它计算元素的个数。对于一个chararray，它计算字符的个数。对于一个bytearray，它计算字节的个数。对于一个数字标量，它总是返回1
SUM	计算在一个单列包中的数值的总和
TOKENIZE	将一个字符串（chararray）拆分为一个由单词组成的包（每个单词为包中的一个元组）。单词的分隔符为空格、双引号（"）、逗号、圆括号和星号（*）
IsEmpty	检查一个bag（包）或map（映射表）是否为空

你不能单独使用表达式和函数，而必须在关系运算符中使用它们来转换数据。

10.5.3 关系型运算符

Pig Latin作为一种语言，关系型运算符是它最为显著的特征。这些运算符将Pig Latin定义为一种数据处理语言。我们首先会很快地浏览较为简单的运算符，来熟悉它们的风格与语法。之后我们会更为详细地探究那些更为复杂的运算符，如COGROUP和FOREACH。

UNION将多个关系归并在一起，SPLIT则将一个关系分割为多个。用一个例子可以清楚地说明它们：

```
grunt> a = load 'A' using PigStorage(',') as (a1:int, a2:int, a3:int);
grunt> b = load 'B' using PigStorage(',') as (b1:int, b2:int, b3:int);
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
grunt> c = UNION a, b;
grunt> DUMP c;
(0,1,2)
(0,5,2)
(1,3,4)
(1,7,8)
grunt> SPLIT c INTO d IF $0 == 0, e IF $0 == 1;
grunt> DUMP d;
(0,1,2)
(0,5,2)
grunt> DUMP e;
(1,3,4)
(1,7,8)
```

www.ChinaDBA.net 中国DBA超级论坛

UNION运算符允许重复。你可以使用DISTINCT运算符来对关系进行去重。我们在c上的SPLIT操作将一个元组传给另一个关系，如果第一个字段（$0）为0，则送到d，如果为1，则送到e。也可以将条件写为把一些行同时送往d和e，或者都不送。你可以用多个FILTER运算符来仿真SPLIT。单独的FILTER运算符会将一个关系裁剪为能够通过某种测试的元组。

```
grunt> f = FILTER c BY $1 > 3;
grunt> DUMP f;
(0,5,2)
(1,7,8)
```

我们已知LIMIT被用于从一个关系中取出指定个数的元组。SAMPLE运算符则根据特定的比例从一个关系中随机地取样出元组。

直到现在还都是相对简单的操作，就这个意义上而言，它们的基本操作单元是一个元组。另一方面，更为复杂的数据处理则需要成组地处理元组。下面我们看一下作用于组的运算符。与以前的运算符不同，这些组运算符会在输出中生成新的schema，这些schema非常依赖于bag（包）和nested（嵌套）数据类型。我们可能先要花些时间才能习惯这些生成的schema。请记住这些组运算符几乎总是被用于生成中间数据。在你计算出最终结果的过程中，这种复杂性仅仅是临时出现的。

这些运算符中比较简单的是GROUP。让我们继续使用前面的关系集合。

```
grunt> g = GROUP c BY $2;
grunt> DUMP g;
(2,{(0,1,2),(0,5,2)})
(4,{(1,3,4)})
(8,{(1,7,8)})
grunt> DESCRIBE c;
c: {a1: int,a2: int,a3: int}
grunt> DESCRIBE g;
g: {group: int,c: {a1: int,a2: int,a3: int}}
```

我们已经生成了一个新的关系g，它源自对c中第3列（$2，也被命名为a3）相同的元组进行组合的结果。GROUP的输出通常有两个字段。第一个字段为组键，这里是a3。第二个字段是一个bag（包），包含组键相同的所有元组。看一下g的dump值，我们看到它有3个元组，对应于c中第3列的3个专有值。第一个元组中的包代表c中第3列等于2的所有元组。在第二个元组中的包代表c中第3类等于4的所有元组，以此类推。在理解了g的数据如何产生之后，我们看这个schema会感觉更舒服。GROUP输出关系的第一个字段总是名为group，代表组键。这里似乎把第一个字段叫做a3更自然些，但是当前Pig并不允许你指定一个名字来取代group。你必须让自己适应将它指向group。GROUP输出关系的第2个字段通常以其操作的关系为名，这里是c，而且正如以前所说，它总是一个包。因为我们使用这个包承载来自c的元组，这个包的schema绝对就是c的schema——由整数构成的3个字段命名为a1、a2和a3。

在进一步讨论之前，我们希望给一个提示，人们可以通过任何函数或者表达式来执行GROUP。例如，如果time是一个时间戳，并且存在一个函数DayOfWeek，可以想象组合会生成一个有7个元组的关系。

www.ChinaDBA.net 中国DBA超级论坛

```
GROUP log BY DayOfWeek(time);
```

最后，可以把关系中的所有元组都放入一个大的包中。这有助于对关系进行聚集分析，因为函数是对包，而不是关系进行操作。例如：

```
grunt> h = GROUP c ALL;
grunt> DUMP h;
(all,{(0,1,2),(0,5,2),(1,3,4),(1,7,8)})
grunt> i = FOREACH h GENERATE COUNT($1);
grunt> dump i;
(4L)
```

这是计算c中元组个数的一种方法。在GROUP ALL的输出中的第一个字段总是字符串all。

既然你已经适应了GROUP，我们可以看一下COGROUP，它将来自多个关系的元组组织在一起。它运行起来很像是join。例如，让我们对a和b的第3列做COGROUP。

```
grunt> j = COGROUP a BY $2, b BY $2;
grunt> DUMP j;
(2,{(0,1,2)},{(0,5,2)})
(4,{(1,3,4)},{})
(8,{},{(1,7,8)})
grunt> DESCRIBE j;
j: {group: int,a: {a1: int,a2: int,a3: int},b: {b1: int,b2: int,b3: int}}
```

虽然GROUP通常在输出中总是生成两个字段，COGROUP总是会生成3个（如果COGROUP两个以上关系时，还会更多）。第1个字段为组键，而第2个和第3个字段为包。这些包为来自与grouping键相匹配的cogroup关系。如果一个grouping key仅匹配了来自一个关系的元组，而不匹配另一个，那么与不匹配关系的相对应的字段将为一个空包。为了忽略对于一个关系而言并不存在的组键，可以在操作中添加一个INNER关键字，如下

```
grunt> j = COGROUP a BY $2, b BY $2 INNER;
grunt> dump j;
(2,{(0,1,2)},{(0,5,2)})
(8,{},{(1,7,8)})
grunt> j = COGROUP a BY $2 INNER, b BY $2 INNER;
grunt> dump j;
(2,{(0,1,2)},{(0,5,2)})
```

从概念上来讲，你可以把COGROUP的默认操作当做一个外部联结，而INNER关键字可以把它修改为left outer join、right outer join或inner join。另一个在Pig中做inner join的方法是使用JOIN运算符。JOIN和inner COGROUP的主要区别是JOIN生成一个输出记录的平坦集合，正如如下schema所显示的

```
grunt> j = JOIN a BY $2, b BY $2;
grunt> dump j;
(0,1,2,0,5,2)
grunt> DESCRIBE j;
j: {a::a1: int,a::a2: int,a::a3: int,b::b1: int,b::b2: int,b::b3: int}
```

我们最后观察的运算符为FOREACH。它浏览一个关系中的所有元组并在输出中生成新的元组。但是在简单的表象之后隐含着巨大的能量，特别是当它被应用于由grouping运算符输出的复杂数据类型时。甚至可以用嵌套形式的FOREACH来处理每个的数据类型。首先让我们熟悉一下对

www.ChinaDBA.net 中国DBA超级论坛

于简单关系的不同FOREACH操作。

```
grunt> k = FOREACH c GENERATE a2, a2 * a3;
grunt> DUMP k;
(1,2)
(5,10)
(3,12)
(7,56)
```

FOREACH后面通常跟着一个别名（一个关系的名字）和关键字GENERATE。在GENERATE之后的表达式控制输出结果。最简单的情况下，我们使用FOREACH将一个关系中的特定列投影到输出中。我们还可应用任意的表达式，例如，在前面的示例中的乘法。

对于有嵌套包（例如，由组合操作生成的嵌套包）的关系，FOREACH有特别的投影语法和一组更丰富的功能。例如，应用嵌套的投影使得每个包仅保留第1个字段：

```
grunt> k = FOREACH g GENERATE group, c.a1;
grunt> DUMP k;
(2,{(0),(0)})
(4,{(1)})
(8,{(1)})
```

要得到每个包中的两个字段:

```
grunt> k = FOREACH g GENERATE group, c.(a1,a2);
grunt> DUMP k;
(2,{(0,1),(0,5)})
(4,{(1,3)})
(8,{(1,7)})
```

大多数内置Pig函数都可以很好地支持对包的操作。

```
grunt> k = FOREACH g GENERATE group, COUNT(c);
grunt> DUMP k;
(2,2L)
(4,1L)
(8,1L)
```

回顾一下，是基于对c的第3列进行组合的结果。因此这个FOREACH语句在C的第3列的值上生成一个频度计数。正如以前所说，组合运算符主要用于生成中间数据，这些中间数据将会被其他运算符（如FOREACH）简化。COUNT函数是一种聚类函数。正如我们所见，你可以通过UDF生成许多其他的函数。

FLATTEN函数被设计为将嵌套式数据类型平坦化。从语句构成上，它很像COUNT和AVG这样的函数，但是，它是一个特殊的运算符，因为它可以改变由FOREACH...GENERATE所产生输出的结构。它的平坦化特性也会根据应用方式和应用目标的不同而有所不同。例如，考虑一个关系，其中元组的形式为(a,(b,c))。语句FOREACH... GENERATE $0, FLATTEN($1)会为每个输入元组生成一个形式为(a,b,c)的输出元组。

当应用于包时，FLATTEN修改FOREACH...GENERATE语句来生成新的元组。它移除嵌套的层次，几乎就是结组的反向操作。如果一个包有N个元组，平坦化会移除这个包并在它的位置上创建N个元组。

www.ChinaDBA.net 中国DBA超级论坛

```
grunt> k = FOREACH g GENERATE group, FLATTEN(c);
grunt> DUMP k;
(2,0,1,2)
(2,0,5,2)
(4,1,3,4)
(8,1,7,8)
grunt> DESCRIBE k;
k: {group: int,c::a1: int,c::a2: int,c::a3: int}
```

理解FLATTEN的另一个方法是通过它产生一个交叉乘积。这样有助于理解在一条FOREACH语句中多次使用FLATTEN的情况。例如，假设我们以某种方式创建关系l。

```
grunt> dump l;
(1,{(1,2)},{(3)})
(4,{(4,2),(4,3)},{(6),(9)})
(8,{(8,3),(8,4)},{(9)})
grunt> describe l;
d: {group: int,a: {a1: int,a2: int},b: {b1: int}}
```

下面的语句，将平坦化两个包，输出其中每个元组那两个包的所有组合：

```
grunt> m = FOREACH l GENERATE group, FLATTEN(a), FLATTEN(b);
grunt> dump m;
(1,1,2,3)
(4,4,2,6)
(4,4,2,9)
(4,4,3,6)
(4,4,3,9)
(8,8,3,9)
(8,8,4,9)
```

www.ChinaDBA.net 中国DBA超级论坛

我们看到在关系l中组键为4的元组在字段a中有一个大小为2的包，而在字段b中也有一个大小为2的包。在m上相应的输出因此有4行代表完整的交叉乘结果。

最后，FOREACH的一个嵌套形式允许对包进行更为复杂的处理。让我们假设有一个关系（如l），其字段之一（如a）是一个包，有嵌套块的FOREACH形式为：

```
alias = FOREACH l {
          tmpl = operation on a;
          [more operations...]
          GENERATE expr [, expr...]
        }
```

语句GENERATE必须始终出现在嵌套块的末尾。它将在l中为每个元组生成一些输出结果。在嵌套块中的操作基于包a创建新的关系。例如，我们在l中元组的每个元素上裁剪出这个包a。

```
grunt> m = FOREACH l {
         tmpl = FILTER a BY a1 >= a2;
         GENERATE group, tmpl, b;
       }
grunt> DUMP m;
(1,{},{(3)})
(4,{(4,2),(4,3)},{(6),(9)})
(8,{(8,3),(8,4)},{(9)})
```

www.ChinaDBA.net 中国DBA超级论坛

在嵌套块中可以有多个语句。甚至每个都可以在不同包上进行操作。

```
grunt> m = FOREACH l {
         tmp1 = FILTER a BY a1 >= a2;
         tmp2 = FILTER b by $0 < 7;
         GENERATE group, tmp1, tmp2;
     };
grunt> DUMP m;
(1,{},{(3)})
(4,{(4,2),(4,3)},{(6)})
(8,{(8,3),(8,4)},{})
```

至本文写作时，在嵌套块中仅允许使用5种运算符：DISTINCT、FILTER、LIMIT、ORDER和SAMPLE。期待将来支持的会更多。

注意　有时FOREACH输出的schema可以完全有别于输入。这种情况下，用户可以在每个字段后使用AS选项来指定输出schema。这个语法不同于LOAD命令中在AS选项之后将schema指定为一个列表，但是在两种情况下我们都使用AS来指定一个schema。

表10-8总结了在Pig Latin中的关系运算符。在许多运算符中都可以看到PARALLEL n选项。数字n为你希望执行运算符的并行度。在实践中，n为在Hadoop中Pig会使用的reduce任务个数。如果你不设置n，它默认为Hadoop集群的缺省配置。Pig文档推荐根据如下原则来设置n的值。

```
n = (#nodes - 1) * 0.45 * RAM
```

这里#nodes为节点的个数，而RAM为每个节点有多少GB的内存容量。

表10-8　Pig Latin中的关系运算符

SPLIT	SPLIT alias INTO alias IF expression, alias IF expression [, alias IF expression ...]; 基于给定的布尔表达式，将一个关系拆分成两个或更多的关系。注意一个元组可以被赋予的关系可以大于1，也可以1个都没有
UNION	alias = UNION alias, alias, [, alias ...] 创建两个或多个关系的并集。注意 ❑ 对于任何的关系，并不保证元组的顺序 ❑ 不需要关系之间具有相同的schema，甚至字段数也可以不同 ❑ 不会移除重复的元组
FILTER	alias = FILTER alias BY expression; 选择基于布尔表达式的元组。用于选择你想要的元组，或删除不想要的元组
DISTINCT	alias = DISTINCT alias [PARALLEL n]; 去除重复元组

SAMPLE	alias = SAMPLE alias factor;
	随机采样一个关系。采样因子在factor中给出。例如，在关系large_data中1%的采样数据为
	small_data = SAMPLE large_data 0.01;
	这个操作的概率是近似的，small_data的大小不会精确地等于large_data的1%，操作并不保证每次所返回的元组数是相同的

www.ChinaDBA.net 中国DBA超级论坛

FOREACH	alias = FOREACH alias GENERATE expression [,expression ...] [AS schema]; （续） 循环遍历每个元组，并生成新的元组。通常用于转换数据的列，例如，添加或删除字段 可以选择性地为输出的关系指定schema；例如，命名新的字段

FOREACH (nested)	alias = FOREACH nested_alias {
	alias = nested_op;
	[alias = nested_op; ...]
	GENERATE expression [, expression ...];
	};
	遍历nested_alias中的每个元组，并生成新的元组。nested_alias中至少一个字段应该是一个包。DISTINCT、FILTER、LIMIT、ORDER和SAMPLE在nested_op中允许对内部包进行操作

JOIN	alias = JOIN alias BY field_alias, alias BY field_alias [, alias BY field_alias …] [USING "replicated"] [PARALLEL n]; 基于共同的字段值计算两个或多个关系的内部联结。当使用replicated选项时，Pig为了加快处理将第一个之后的所有关系都存在内存中。你必须确保所有这些较小的关系的确小到可以放在内存中 在JOIN中，当输入关系是平坦的，其输出关系也是平坦的。此外，输出关系中的字段数是输入关系字段数之和，而输出关系中的schema是输入关系schema的串联
GROUP	alias = GROUP alias { [ALL] | [BY {[field_alias [, field_alias]] | * | [expression]] } [PARALLEL n]; 在单一的关系中，将相同组键的元组组合在一起。组键通常是一个或多个字段，但它也可以是整个元组（*）或表达式。此外可以使用Group alias All将所有的元组组成一组。 输出关系包含两个字段，其名称是自动生成的。第一个字段始终被命名为“group”，且与组键的类型相同。第二个字段取输入关系的名称，并且为包类型。这个包的schema与输入关系的schema相同
COGROUP	alias = COGROUP alias BY field_alias [INNER | OUTER] , alias BY field_alias [INNER | OUTER] [PARALLEL n]; 基于通用的组值，从两个或更多的关系中组合元组 输出关系会为每个唯一的组值生成一个元组。每个元组都把组值当作第一个字段。第二个字段是一个包，它含有来自第一个输入关系中与组值相匹配的元组。同理形成的输出元组作为第三个字段 在默认OUTER联结语义中，任何输入关系中出现的所有组值都表示在输出关系中。如果一个输入关系不含有任何特定组值的元组，它会在相应的输出元组有一个空的包。如果为关系设置了INNER选项，那么只有在输入关系中存在的组值才允许出现在输出关系中。该关系在输出中不能有空包 你可以按多个字段进行组合。为此，必须用在括号内以逗号分隔的列表为field_alias指定字段。 COGROUP（带有INNER）和JOIN是类似的，只是COGROUP会生成嵌套的输出元组
CROSS	alias = CROSS alias, alias [, alias ...] [PARALLEL n]; 计算两个或多个关系的（平坦）交叉乘结果。这是一个耗时的操作，你应该尽量避免这种情况
ORDER	alias = ORDER alias BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] ...] } [PARALLEL n]; 基于一个或多个字段对一个关系进行排序。如果你在ORDER操作之后获取这个关系（通过DUMP或STORE），它一定按照理想的顺序排序。进一步处理（FILTER、DISTINCT等）可能会毁掉这个排序
STREAM	alias = STREAM alias [, alias ...] THROUGH {'command' | cmd_alias } [AS schema] ; 通过外部脚本处理一个关系

至此我们已经学习Pig Lat in语言的各个方面——数据类型、表达式、函数和关系运算符。你可以进一步使用用户定义函数来扩展这个语言。但是在讨论它们之前，我们为Pig Latin的编译和优化做一个注解，以结束本节。

www.ChinaDBA.net 中国DBA超级论坛

10.5.4　执行优化

如许多流行的编译器一样，只要执行计划与原始程序在逻辑上保持一致，Pig编译器可以重排执行顺序以优化性能。例如，想象一下有一个程序对每个记录的特定字段（如社会安全号码）应用了一个庞大的函数（如加密），之后通过一个过滤函数根据另一个字段来选择记录（如限定仅包含特定地域的人群）。编译器将这两个运算符的执行顺序反转，并不会影响最终的结果，但可以让性能大大改善。让过滤步骤先做可以极大地减少加密阶段必须处理的数据量和工作量。

随着Pig的成熟，更多的优化会被添加到编译器中。因此，重要的一点是要一直使用最新版本。但是无论优化什么代码，编译器的能力总有一个局限。你可以阅读Pig网络文档中关于增强性能的技术。在Pig版本0.3中增强性能的要点可见http://hadoop.apache.org/pig/docs/r0.3.0/cookbook. html。

10.6　用户定义函数

Pig Latin设计的基本理念是通过用户定义函数（UDF）获得扩展性，并为编写UDF提供一组良好定义的API。这并不意味着你必须编写所需的全部函数。Pig生态系统[①]中的一部分是PiggyBank[②]，即一个供用户共享函数的在线库。你应该首先检查PiggyBank来寻找所有会用到的函数。仅当你找不到合适的函数时，才应考虑自己编写。你还应考虑将自己的UDF回馈给PiggyBank来造福Pig社区中的其他人。

www.ChinaDBA.net 中国DBA超级论坛

10.6.1　使用 UDF

截至本书写作时，UDF总是用Java编写并打成Jar包。要使用一个特定的UDF，你就需要这个包含有UDF类文件的Jar文件。例如，当使用来自PiggyBank的函数时，你很可能会获得一个piggybank.jar文件。

要使用一个UDF，你必须首先使用REGISTER语句在Pig中注册这个jar文件。之后，就可以通过其完全认证的Java类名调用UDF。例如，在PiggyBank中有一个UPPER函数，它将一个字符串转换为uppercase。

```
REGISTER piggybank/java/piggybank.jar;
b = FOREACH a GENERATE
      ➥org.apache.pig.piggybank.evaluation.string.UPPER($0);
```

如果你需要多次使用一个函数，每次都要把已经完全认证的类名完整地重写一遍会让人很恼火。Pig提供DEFINE语句来为UDF指定一个名称。你可以重写上面的语句

```
REGISTER piggybank/java/piggybank.jar;
DEFINE Upper org.apache.pig.piggybank.evaluation.string.UPPER();
b = FOREACH a GENERATE Upper($0);
```

---

① 我本想叫它Pig pen（猪圈），但是PigPen实际上是Eclipse用于编辑Pig Latin脚本的一个插件的名字。见http://wiki.apache.org/pig/PigPen。

② http://wiki.apache.org/pig/PiggyBank。

表10-9总结了与UDF相关的语句。

表10-9 Pig Latin中的UDF语句

DEFINE	DEFINE alias { function | 'command' [...] }; 将别名赋值为一个函数或命令
REGISTER	REGISTER alias; 向Pig注册UDF。当前UDF仅能用Java编写，而alias为JAR文件的路径。所有的UDF在使用之前必须注册

如果仅使用其他人编写的UDF，你只要知道这么多就够了。但是如果你找不到所需的UDF，就必须自己写了。

10.6.2 编写 UDF

Pig支持两种主要类型的UDF：eval[1]和load/store。我们仅在LOAD和STORE语句中使用load/store函数，来帮助Pig读写特殊的格式。大多数UDF为eval函数，取出一个字段值然后返回另一个字段值。

本书写作时，你仅可通过Pig的Java API来写UDF[2]。要生成一个eval UDF，需要创建一个Java类，这个类继承抽象类EvalFunc<T>。它仅有一个抽象方法需要我们去实现。

```
abstract public T exec(Tuple input) throws IOException;
```

在一个关系的每个元组上都会调用这个方法，这里每个元组都由一个Tuple对象表示。方法exec()处理这个元组并返回一个类型T，对应于一个有效的Pig Latin类型。T可以为表10-10中任何一个Java类，其中一些为原有Java类，另一些为Pig的扩展。

表10-10 Pig Latin的类型及其在Java中等效的类

Pig Latin类型	Java类
Bytearray	DataByteArray
Chararray	String
Int	Integer
Long	Long
Float	Float
Double	Double
Tuple	Tuple
Bag	DataBag
Map	Map<Object, Object>

学习编写UDF的最好办法是在PiggyBank中剖析一个现有的UDF。即使在自己编写时，一个通常有用的办法是从一个与所需功能相近的可用UDF开始，再做一些处理逻辑的修改。对我们而言，可以从以前所用的PiggyBank中编写UPPER这个UDF。exec()方法如下所示：

① 一些eval函数非常通用并被特别关注。它们有时在自己的分类中描述。这包含过滤函数（返回一个布尔值的eval函数）以及聚合函数（读取一个bag并返回一个标量值的eval函数）。

② 这个API的Javadoc在http://hadoop.apache.org/pig/javadocs/documentation/。

www.ChinaDBA.net 中国DBA超级论坛

```
public class UPPER extends EvalFunc<String>
{
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try {
            String str = (String)input.get(0);
            return str.toUpperCase();
        } catch(Exception e){
            System.err.println("Failed to process input; error - " +
                ➡e.getMessage());
            return null;
        }
    }
}
```

对象input属于Tuple类，它有两个方法可用于获取其内容。

```
List<Object> getAll();
Object get(int fieldNum) throws ExecException;
```

方法getAll()将元组中的所有字段返回为一个有序的列。而UPPER使用get()方法来请求一个特定的字段（在位置0）。如果被请求的字段数大于元组中的字段数，该方法会抛出一个ExecException异常。在UPPER中，获取的字段被转换为一个Java String，这通常是有效的，但如果我们要转换的是不兼容的数据类型，就会导致一个转换异常。稍后我们会介绍如果使用Pig来确保转换正常运行。在任何情况下，try/catch部分都会捕获并其处理所有的异常。如果一切正常，Upper的exec()方法将返回一个字符大写的String。此外，大多数UDF有一个默认动作，即在输入元组为空时，输出也为空。

除了实现exec()，UPPER还覆盖了EvalFunc中的一些方法，其中一个是getArgToFuncMapping:

```
@Override
public List<FuncSpec> getArgToFuncMapping() throws FrontendException {
    List<FuncSpec> funcList = new ArrayList<FuncSpec>();
    funcList.add(new FuncSpec(this.getClass().getName(),
        ➡new Schema(new Schema.FieldSchema(null, DataType.CHARARRAY))));
    return funcList;
}
```

此getArgToFuncMapping()方法返回FuncSpec对象的一个List，表示input元组中每个字段的schema。Pig会负责类型的转换，在将元组传递给exec()之前，会把元组中的所有字段类型进行转换以便和该schema保持统一。对于无法做类型转换的字段会传递空值。

UPPER仅关心第一个字段的类型，因此它在列表中仅增加一个FuncSpec，而这个FuncSpec声明该字段必须为chararray类型，表示为DataType.CHARARRAY。FuncSpec的实例化是相当复杂的，这是由于Pig可以处理复杂的嵌套类型。幸好除非你要处理不常用的复杂类型，你通常都会在PiggyBank的现有UDF中找到一个所需类型的实例。进而在代码中重用。如果你有和另一个UDF相同的元组schema，你甚至可以重用全部getArgToFuncMapping()函数。

除了告知Pig输入的schema之外，你还可以告诉Pig输出的schema。如果UDF的输出是一个简

www.ChinaDBA.net 中国DBA超级论坛

单的标量，你大可不必这样做，因为Pig会使用Java的Reflection机制来自动地推断这个schema。但是如果你的UDF返回的是一个元组或一个包，Reflection机制无法完整地找到这个schema。此时你应该指定它，使得Pig可以正确地传播这个schema。

对于UPPER这个案例，因为它仅输出一个简单的字符串，所以不需要为它指定输出的schema。但是UPPER的确可以通过覆写outputSchema()告诉Pig它返回的是一个字符串(DataType.CHARARRAY)。

```
@Override
public Schema outputSchema(Schema input) {
    return new Schema(
        new Schema.FieldSchema(
            getSchemaName(this.getClass().getName().toLowerCase(), input),
            DataType.CHARARRAY
        )
    );
}
```

同样，由于Pig具有复杂的嵌套式类型，Schema对象的构建看上去也很复杂。一个特例是如果UDF输出的schema与输入一致，我们可以返回该输入schema的副本：

```
public Schema outputSchema(Schema input) {
    return new Schema(input);
}
```

至于FuncSpec的构造，你可能会在PiggyBank中发现一个已存在的UDF满足所需的输出schema。

有几种UDF的类型需要特别注意。过滤函数为返回布尔值的eval函数，我们在Pig Latin的FILTER和SPLIT语句中使用它们。它们应当扩展FilterFunc而非EvalFunc。聚合函数为读取一个包而返回一个标量的eval函数。它们通常用作计算聚集，例如COUNT，我们有时可以在Hadoop中使用一个combiner来优化它们。我们没有讨论用于读写数据集的load/save UDF。这些更为高阶的内容请参见Pig的UDF文档：http://hadoop.apache.org/pig/docs/r0.3.0/udf.html。

10.7 脚本

编写Pig Latin脚本很大程度上是将已经在Grunt中测试成功的Pig Latin语句打包在一起。但Pig脚本也确有几个特别之处，它们是注释、参数替换和多查询执行。

10.7.1 注释

因为Pig Latin脚本还会重用，为他人（或自己）留下些注释，以便在将来还可以理解显然是个不错的主意。Pig Latin支持两种形式的注释：单行和多行。单行注释以双斜杠开始，在行尾结束。多行注释以/*和*/标记包裹，类似Java中的多行注释。例如，一个带有注释的Pig Latin脚本如下

```
/*
 * Myscript.pig
 * Another line of comment
```

www.ChinaDBA.net 中国DBA超级论坛

```
 */
log = LOAD 'excite-small.log' AS (user, time, query);
lmt = LIMIT log 4; -- Only show 4 tuples
DUMP lmt;
-- End of program
```

10.7.2 参数替换

当你编写一个可重用的脚本时，它通常是参数化的，便于在每次运行时做出改变。例如，脚本可能每次是从用户那里获得输入和输出的文件路径。Pig支持参数替换，允许用户在运行时指定这些信息。这些参数通过脚本中的前缀$来表示。例如，如下的脚本显示了来自用户指定的日志文件的一组由用户设定的元组。

```
log = LOAD '$input' AS (user, time, query);
lmt = LIMIT log $size;
DUMP lmt;
```

在此脚本中的参数为$input和$size。如果使用pig命令运行此脚本，则使用-param name=value指定这些参数。

```
pig -param input=excite-small.log -param size=4 Myscript.pig
```

请注意你不需要在参数中使用$前缀。如果一个参数值包含多个单词，你可以将它用单引号或双引号标记。一个实用的技术是使用Unix命令来生成这个参数值，尤其是日期。这可以通过Unix的命令替换方法，替换由后撇号(`)括起来的部分得到的。

```
pig -param input=web-'date +%y-%m-%d'.log -param size=4 Myscript.pig
```

通过执行此操作，用于Myscript.pig的输入文件将基于该脚本运行的日期来生成。例如，如果该脚本在2009年7月29日执行，那么输入文件将为web-09-07-29.log。

如果必须指定多个参数，一种更方便的做法是把它们放在一个文件中，并告知Pig执行脚本时使用基于该文件的参数替换。例如，我们可以创建内容如下的Myparams.txt文件。

```
# Comments in a parameter file start with hash
input=excite-small.log
size=4
```

参数文件通过变量-param_file filename传递给pig命令。

```
pig -param_file Myparams.txt Myscript.pig
```

你可以指定多个参数文件，或者把参数文件与在命令行用-param直接指定参数的方式相混合。如果你多次定义一个参数，最后定义者有优先权。如果对脚本结束后使用了哪些参数值有疑问，你可以通过-debug选项运行pig命令。这会告诉Pig运行该脚本，并且输出一个名为original_ script_name.substituted的文件，它具有原始脚本，但其中所有的参数都被替换了。使用-dryrun选项执行pig输出相同的文件，但并不执行该脚本。

exec和run命令允许你在Grunt Shell内部运行Pig Latin脚本，它们支持使用同样的-param和-param_file参数来进行参数替换；例如：

```
grunt> exec -param input=excite-small.log -param size=4 Myscript.pig
```

www.ChinaDBA.net 中国DBA超级论坛

但是，在exec和run中所做的参数替换不支持Unix命令，也没有debug和dryrun选项。

10.7.3 多查询执行

在Grunt shell中，DUMP或STORE操作会处理所有在生成结果之前需要执行的语句。另一方面，Pig将一个Pig脚本作为一个整体来优化与处理。如果你的脚本最后仅有一个DUMP或STORE命令，这两者之间的区别根本不会造成影响。如果你的脚本有多个DUMP/STORE，Pig脚本的多查询执行（multiquery execution）会通过避免冗余评估而提高效率。例如，假设你有一个存储中间数据的脚本：

```
a = LOAD ...
b = some transformation of a
STORE b ...
c = some further transformation of b
STORE c ...
```

如果你在Grunt中输入这些语句，因为没有多查询执行，就会在STORE b命令处生成一串作业来计算b。到了STORE c时，Grunt会运行另一串作业来计算c，但这时它又会再次评估a和b。你可以在STORE b之后手动地插入一条语句b = LOAD...来避免重新评估，强制c的计算使用b的保存值。这依赖于一个假设，即b的保存值没有被修改，因为Grunt自己是没办法知道的。

另一方面，如果你将所有的语句作为脚本来运行，多查询执行可以通过智能地处理中间数据来优化执行。Pig将所有的语句一起编译，从而可以发现依赖与冗余。多查询执行默认是开启的，通常对计算的结果没有影响。但是如果Pig不知道数据依赖关系，多查询执行可能会失效。这极少会发生，但也有可能出现，例如在UDF中。考虑脚本：

```
STORE a INTO 'out1';
b = LOAD ...
c = FOREACH b GENERATE MYUDF($0,'out1');
STORE c INTO 'out2';
```

如果定制化函数MYUDF像这样从文件out1中获取a，Pig编译器是无从知道的。看不到这种依赖，Pig会认为在评估a之前评估b和c也是OK的。要关闭多查询执行，用选项-M或-no_multiquery运行pig命令即可。

10.8 Pig 实战——计算相似专利的例子

鉴于Pig所提供的更多功能，我们可以实现更具挑战性的数据处理应用。专利数据集有一个有趣的应用，即基于引用数据发现类似的专利。在某种程度上，经常在一起被引用的专利一定是类似的（或者至少是相关的）。这些应用本质上与Amazon.com的协同过滤和相似文档查找是相同的，协同过滤即“买了这个的顾客还会买那个”，相似文档查找即找到有相似单词集合的文档。为了便于说明，我们假设要查找那些一起被引用超过N次的专利，这里N是我们指定的一个固定值[1]。

[1] 这种变化可能会包含更高级的评分函数，例如频繁项的归一化，或者相似性评价而非一刀切。我这里之所以选择一刀切的方式，是为了更容易实现，而且同样可以阐释计算相似性的本质。

www.ChinaDBA.net 中国DBA超级论坛

对于涉及成对计算的应用（例如，计算每对专利联合引用的数量），通常很容易想到采用一对嵌套的循环来枚举出所有专利对的合并，并对每一对进行计算。即便Hadoop通过增加更多硬件可以让扩展更为容易，我们依然应该记住计算复杂性这个基本概念。二阶复杂度仍会使得线性扩展成为泡影。即使300万个专利的小数据集也会带来9万亿个专利对。我们需要更为精巧的算法。

所用方法的本质在于结果数据是稀疏的。大多数专利对没有相似性，因为它们永远不会被一起引用。如果我们重新设计相似度计算，让它仅作用于已知被一起引用的专利对，计算将变得更加容易。从数据的角度来看，这种做法是很自然的。这个实现对每个专利做以下的步骤的操作：

- 获得它所引用专利的列表；
- 生成这个列表上所有专利按对归并的结果，并记录每一个专利对；
- 统计每个专利对的个数；

如果每个专利引用固定数量的专利，比如10个，该实现方法会为每个专利生成45个专利对。（45是可能来自10个专利的按对归并结果，可通过数学推导得到，即$10 \times 9/2$。）那么300万个专利会生成1.35亿个专利对，这比暴力方法得到的专利对要少几个数量级。当原始数据集变得更大时，这种优势会更加明显。

即便我们有了应用程序的算法，在MapReduce中实现它依然很繁琐。它需要将多个作业链接在一起，而每个作业都需要自己的类。而Pig Latin仅用十几行就可以实现这3个阶段的程序（见代码清单10-1），而且进一步优化可以让行数降得更低，还能提高效率。

代码清单10-1　用Pig Latin脚本找到经常被一起引用的专利

```
          cite = LOAD 'input/cite75_99.txt' USING PigStorage(',')
                 ➥AS (citing:int, cited:int);
    cite_grpd = GROUP cite BY citing;
cite_grpd_dbl = FOREACH cite_grpd GENERATE group, cite.cited AS cited1,
                 ➥cite.cited AS cited2;
       cocite = FOREACH cite_grpd_dbl
                 ➥GENERATE FLATTEN(cited1), FLATTEN(cited2);
  cocite_fltrd = FILTER cocite BY cited1 != cited2;
  cocite_grpd = GROUP cocite_fltrd BY *;
   cocite_cnt = FOREACH cocite_grpd
                 ➥GENERATE group, COUNT(cocite_fltrd) as cnt;
    cocite_flat = FOREACH cocite_cnt GENERATE FLATTEN(group), cnt;
cocite_cnt_grpd = GROUP cocite_flat BY cited1;
    cocite_bag = FOREACH cocite_cnt_grpd
                 ➥GENERATE group, cocite_flat.(cited2, cnt);
cocite_final = FOREACH cocite_cnt_grpd {
    similar = FILTER cocite_flat BY cnt > 5;
    GENERATE group, similar;
}
STORE cocite_final INTO 'output';
```

Pig Latin以及通常的复杂数据处理可能很难读懂。幸好我们可以对cocite_bag使用Grunt的ILLUSTRATE命令来得到语句的仿真样本运行，并查看正在生成的每个操作。（我们已经重新格式化了输出以适应打印页面的宽度。）

www.ChinaDBA.net 中国DBA超级论坛

```
---------------------------------------------------------------
| cite        | citing: bytearray | cited: bytearray |
---------------------------------------------------------------
            | 3858554           | 3601095          |
            | 3858554           | 3685034          |
            | 3859004           | 1730866          |
            | 3859004           | 3022581          |
            | 3859572           | 3206651          |
---------------------------------------------------------------

---------------------------------------
| cite        | citing: int | cited: int |
---------------------------------------
            | 3858554     | 3601095    |
            | 3858554     | 3685034    |
            | 3859004     | 1730866    |
            | 3859004     | 3022581    |
            | 3859572     | 3206651    |
---------------------------------------

---------------------------------------------------------------------------
| cite_grpd     | group: int | cite: bag({citing: int,cited: int})        |
---------------------------------------------------------------------------
                | 3858554    | {(3858554, 3601095), (3858554, 3685034)}   |
                | 3859004    | {(3859004, 1730866), (3859004, 3022581)}   |
                | 3859572    | {(3859572, 3206651)}                       |
---------------------------------------------------------------------------

---------------------------------------------------------------------------------
| cite_grpd_dbl | group:     | cited1:                    | cited2:                |
                |    int     |     bag({cited: int})      |     bag({cited: int})|
---------------------------------------------------------------------------------
                | 3858554    | {(3601095), (3685034)}     | {(3601095),(3685034)}|
                | 3859004    | {(1730866), (3022581)}     | {(1730866),(3022581)}|
                | 3859572    | {(3206651)}                | {(3206651)}          |
---------------------------------------------------------------------------------
```

关系cite_grpd让每个专利包含一个bag（包），在这个包中是引用的专利。从这个关系（如示例），我们可以看到专利3601095和3685034在专利3858554中被一起引用。在创建cite_grpd时通过GROUP操作对共同引用的专利进行组合。关系cite_grpd_dbl仅去除了冗余的“引用”专利并创建了一个重复的列。cited1和cited2列的值是相同的。该重复可以支持交叉乘操作，以便生成所有专利对的合并结果。

```
---------------------------------------------------------
| cocite      | cited1::cited: int | cited2::cited: int |
---------------------------------------------------------
                | 3601095            | 3601095            |
                | 3601095            | 3685034            |
                | 3685034            | 3601095            |
                | 3685034            | 3685034            |
                | 1730866            | 1730866            |
                | 1730866            | 3022581            |
                | 3022581            | 1730866            |
                | 3022581            | 3022581            |
                | 3206651            | 3206651            |
---------------------------------------------------------
```

www.ChinaDBA.net 中国DBA超级论坛

cite_grpd_dbl对每一行平坦化后再用交叉乘生成cocite[①]。它记录了被一起引用的所有专利对，也是我们算法的一个主要检查点。我们知道，cocite 是一个大的关系，在我们的机制下比暴力方法更为有效。有3种方法可以进一步裁剪cocite。我们将讨论所有方法但只会实现其中一个。

第一个潜在的裁剪方法是注意每个引用的专利被认为是已经cocite了自身。我们知道对应用程序而言，找出某一专利与自身相似是毫无意义的，我们可以忽略所有这类专利对。请注意如果我们在计算中保留这些“身份”对，它们的共同引用计数最终会完全是引用计数。如果我们在寻找专利cocited的时间的百分比，这些数字仍很有用。我们不计算百分比，故无需考虑这种情况。

因为共同引用是对称的，专利对总是出现2次，但次序相反。例如，(3601095, 3685034) 和 (3685034, 3601095) 会在一起出现。鉴于当前应用需要找到一起共同引用超过*N*次的专利对，我们可以用一个简单的规则，只保留两个冗余专利对中的一个，将cocite的大小减半。这个规则可以是这样的：仅保留第一个字段小于第二个字段的专利对。但是在某些应用中，保留冗余的专利对会对今后的查找有用。例如，我们可以通过在第一个字段中搜索X，找到与X有共同引用的所有专利。而在裁剪之后，我们必须在两个字段中查找X。

最终，我们可以使用启发式算法来删除我们认为不重要的共同引用专利对。即用精度来换取效率。启发式算法的适用性和有效性依赖于应用程序的语义和数据分布。在我们的例子中，当一个专利同时引用多项专利时，它在cocite中生成的行数将达到两位数。如果我们认可如此“详尽”的专利对理解相似专利对并无帮助，就可以将它们移除以显著减少处理数据的大小，而这样做对最终结果的影响很小。这种启发式算法的好处是比较大的，如果我们要找反向专利引用或者文本文档，其中各个项目的出现频率是极度不平均的，主要的数据处理都会被花费在几个常见项的扩充上。实际上，此时近似的启发式算法通常是必需的。

这里需要注意的是到我们始终在关注于高层次的数据处理，而完全避免了从底层对MapReduce进行讨论。

```
-------------------------------------------------------------
| cocite_fltrd      | cited1::cited: int | cited2::cited: int |
-------------------------------------------------------------
|                   | 3601095            | 3685034            |
|                   | 3685034            | 3601095            |
|                   | 1730866            | 3022581            |
|                   | 3022581            | 1730866            |
-------------------------------------------------------------
```

如前面所述，我们决定将那些“相同”的专利对排除掉。

① 请注意cocite可以直接从cite_grpd中计算得到，通过使用一个更为复杂的FOREACH语句，当你更习惯pig Latin时，你就可以选择这样做。

www.ChinaDBA.net 中国DBA超级论坛

cocite_grpd	group: tuple({cited1::cited: int, cited2::cited: int})	cocite_fltrd: bag({cited1::cited: int, cited2::cited: int})
	(1730866, 3022581)	{(1730866, 3022581)}
	(3022581, 1730866)	{(3022581, 1730866)}
	(3601095, 3685034)	{(3601095, 3685034)}
	(3685034, 3601095)	{(3685034, 3601095)}

cocite_cnt	group: tuple({cited1::cited: int,cited2::cited: int})	cnt: long
	(1730866, 3022581)	1
	(3022581, 1730866)	1
	(3601095, 3685034)	1
	(3685034, 3601095)	1

cocite_flat	group::cited1::cited: int	group::cited2::cited: int	cnt: long
	1730866	3022581	1
	3022581	1730866	1
	3601095	3685034	1
	3685034	3601095	1

我们将专利对的引文组合在一起、统计并拼合为关系。不过ILLUSTRATE生成仅有联合引用数为1的样本数据。然而，这些操作基本上完成了我们想要做的事情。如果我们坚持满足应用程序原来的要求，只寻找联合应用多于*N*次的专利对，可以在cocite_flat上应用一个过滤器来做到。但其他类型的过滤可能会需要进一步地组合元组。例如，你可能希望为每个专利找到联合引用最多的前*K*个专利。让我们看看余下的输出结果：

cocite_cnt_grpd	group: int	cocite_flat: bag({group::cited1::cited: int,group::cited2::cited: int,cnt: long})
	1730866	{(1730866, 3022581, 1)}
	3022581	{(3022581, 1730866, 1)}
	3601095	{(3601095, 3685034, 1)}
	3685034	{(3685034, 3601095, 1)}

cocite_bag	group: int	cocite_flat: bag({group::cited2::cited: int,cnt: long})
	1730866	{(3022581, 1)}
	3022581	{(1730866, 1)}
	3601095	{(3685034, 1)}
	3685034	{(3601095, 1)}

www.ChinaDBA.net 中国DBA超级论坛

如果希望为每个专利找到前K个联合引用最多的专利，可以使用FOREACH语句来处理cocite_bag中的每个元组，再写自己的UDF来取一个包（cocite_flat）并返回一个包含前K个元组包（联合引用最多的）。最后一步可以作为一个练习。让我们通过一个嵌套FOREACH语句的例子来排除在包中计数值为5个及5个以下的元组。

```
cocite_final = FOREACH cocite_cnt_grpd {
    similar = FILTER cocite_flat BY cnt > 5;
    GENERATE group, similar;
}
```

如你所见，Pig极大地简化了数据处理应用程序的实现。“相似项”特征对于不同的应用程序都是有用的，但实现起来非常困难。而使用Pig和Hadoop，只需一个下午即可完成。此外，其易用性的进步支持了其他功能的快速开发。你在自己练习时，可以查找那些有相似引文的专利，而不是寻找常被一起引用的专利。

10.9　小结

Pig是架构在Hadoop之上的高级数据处理层。Pig Latin语言提供给编程者一种更直观的定制数据流的方法。它支持结构化数据的处理schema，也可以足够灵活地处理非结构化文本或半结构化XML数据。用UDF可以对它进行扩展。它大幅简化了数据联结和作业链接——对许多开发者而言非常复杂的两种MapReduce编程。为了说明其效用，我们给出计算专利联合引用的示例，显示一个复杂的MapReduce程序在Pig Latin中用几十行代码就能实现。

www.ChinaDBA.net 中国DBA超级论坛

第11章 Hive及Hadoop群

本章内容

- 什么是Hive
- Hive的安装
- 在数据仓库中使用Hive
- 其他与Hadoop相关的软件包

即使像Hadoop这样强大的工具，也不能满足每个人的需求。许多项目如雨后春笋般涌现出来，为特定目的扩展了Hadoop。那些突出并且得到很好维护的项目已经正式成为Apache Hadoop项目下的子项目。[1]这些子项目如下所示。

www.chinadba.net 中国DBA超级论坛

- Pig——一种高级数据流语言。
- Hive——一种类SQL数据仓库基础设施。
- HBase——一种模仿Google Bigtable的分布式的、面向列的数据库。
- ZooKeeper——一种用于管理分布式应用之间共享状态的可靠的协同系统。
- Chukwa——一种用于管理大型分布式系统的数据收集系统。

我们在第10章详细讨论了Pig，本章将学习Hive。此外，11.2节还会简单介绍其他与Hadoop相关的项目。其中一些（例如Cascading和CloudBase）与Apache没有关系。有些则仍在新生阶段（例如Hama和Mahout）。你会在第12章中看到这些工具在实战中的一些应用。

11.1 Hive

Hive[2]是建立在Hadoop基础上的数据仓库软件包。在开始阶段，它被Facebook用于处理大量的用户数据和日志数据。它现在是Hadoop的子项目并有许多的贡献者。其目标用户仍然是习惯了SQL的数据分析师，他们需要在Hadoop规模的数据上做即席查询、汇总和数据分析[3]。通过称为

① 我们至今在本书中所提到的“Hadoop”（HDFS和MapReduce）在技术上被称为Apache Hadoop的子项目“Hadoop Core”，不过人们倾向于把它通俗地称为Hadoop。

② http://hadoop.apache.org/hive/。

③ 请注意因为Hive构建在Hadoop的基础之上，它仍然设计用作处理低延迟与批量类型的作业。故而它并不会直接取代传统的SQL数据仓库，比如那些由Oracle提供的数据仓库。

www.ChinaDBA.net 中国DBA超级论坛

HiveQL的类SQL语言，你可以发起一个查询来实现与Hive的交互。例如，一个从user表获取所有活跃用户的查询如下所示：

```
INSERT OVERWRITE TABLE user_active
SELECT user.*
FROM user
WHERE user.active = 1;
```

Hive的设计体现出它是一个管理和查询结构化数据的系统。通过专注结构化数据，Hive可以实现MapReduce一般所不具备的某些优化和可用性功能。受到SQL影响的Hive语言让用户可以脱离MapReduce编程的复杂性。它沿用了关系数据库中的常见概念，如表、行、列和schema，以便于学习。此外，虽然Hadoop天生支持平坦文件，但Hive可以使用目录结构来“划分”数据，以提高某些查询的性能。为支持这些额外的功能，Hive中有一个全新且重要的组件，它就是用于存储schema信息的metastore。这个metastore通常只有在关系数据库中才有。

你可以通过几种方法与Hive交互，包括Web GUI和JDBC（Java数据库连接）接口。但是大多数交互倾向于利用命令行界面（CLI），这正是我们所关注的焦点。你可以在图11-1上看到Hive的高层体系结构框图。

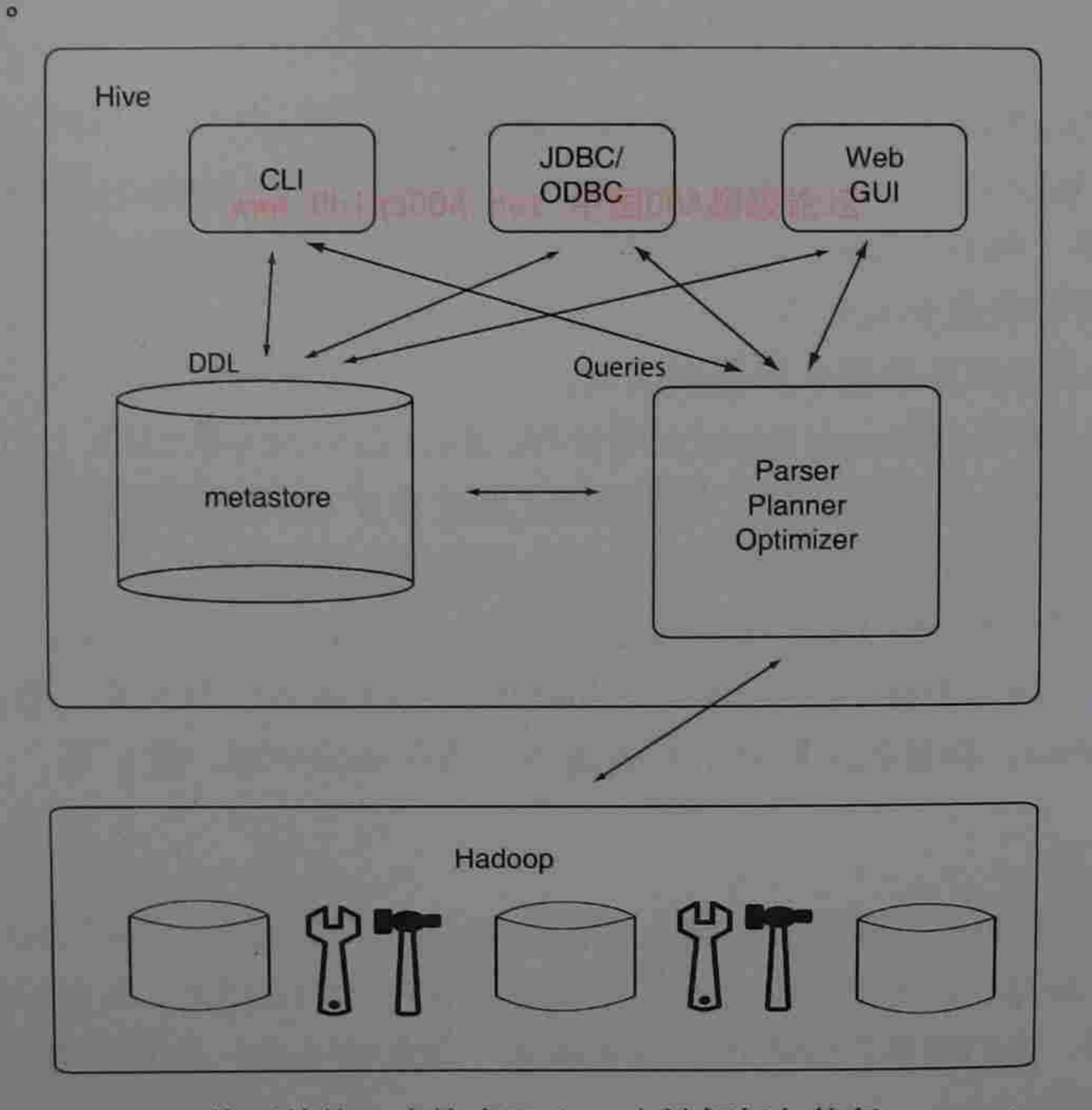


图11-1　Hive体系结构。查询在Hadoop上被解析与执行。metastore是一个重要组件，它用于确定查询如何执行

11.1.1　安装与配置 Hive

Hive需要Java 1.6和Hadoop 0.17及以上版本的支持。你可以在http://hadoop.apache.org/hive

www.ChinaDBA.net 中国DBA超级论坛

/releases.html上找到Hive的最新版本。下载并将压缩包解压到HIVE_HOME目录。Hive需要Hadoop已经启动并运行。此外，还需要在HDFS中为Hive设置几个目录备用。

```
bin/hadoop fs -mkdir /tmp
bin/hadoop fs -mkdir /user/hive/warehouse
bin/hadoop fs -chmod g+w /tmp
bin/hadoop fs -chmod g+w /user/hive/warehouse
```

如果把数据完全交由Hive管理，Hive会把它们存储在/user/hive/warehouse目录下。它还会自动压缩数据并加入特殊的目录结构（例如分区）以提高查询性能。如果你计划使用Hive查询数据，最好让Hive来管理它。但如果数据已经在HDFS的其他目录中，而你还想让它们继续存在那里，Hive也可以直接对它们进行操作。此时，Hive会照原样读取数据，而不会为了查询处理而优化数据存储。有的用户不理解这种区别，他们以为Hive需要数据采用某些特殊的Hive格式。这绝对不是真的。

Hive将元数据存储在标准关系数据库中。Hive自带了Derby，它是一个开源、轻量、嵌入式的SQL数据库[①]，它与Hive一起安装并运行在客户端机器上。如果你是Hive的唯一用户，使用默认设置就足够了。但除非是在做初始测试和评价，最有可能的情况还是将Hive部署在多用户环境中，而你不会想让每个用户都创建自己的元数据。这就需要有一个集中的地方来存储元数据。通常会使用一个共享的SQL数据库，如MySQL，但任何符合JDBC的数据库都是可以的。你还需要一个数据库服务器，在其上创建一个数据库作为Hive的metastore。这个数据库通常命名为metastore_db。一旦创建了这个数据库，就把每个Hive的安装都配置成指向它，将它作为metastore。这个安装的配置是通过修改文件hive-site.xml和jpox.properties来完成的，它们都放在$HIVE_HOME/conf目录下。原始的安装中没有hive-site.xml文件，你必须创建它。此文件中的属性会重写hive-default.xml中的属性，类似于hadoop-site.xml重写hadoop-default.xml。文件hive-site.xml会重写3个属性，如下所示：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
 <name>hive.metastore.local</name>
 <value>false</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionURL</name>
 <value>jdbc:mysql://<hostname>/metastore_db</value>
</property>

<property>
 <name>javax.jdo.option.ConnectionDriverName</name>
 <value>com.mysql.jdbc.Driver</value>
</property>

</configuration>
```

① http://db.apache.org/derby/。

www.ChinaDBA.net 中国DBA超级论坛

表11-1解释了这些属性。再在文件jpox.properties中指定javax.jdo.option.ConnectionURL和javax.jdo.option.ConnectionDriverName属性。此外，在jpox.properties中还指定了登录到数据库的用户名和密码。jpox.properties文件应包含以下4行：

```
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionURL=jdbc:mysql://<hostname>/metastore_db
javax.jdo.option.ConnectionUserName=<username>
javax.jdo.option.ConnectionPassword=<password>
```

表11-1　在多用户模式下将MySQL数据库用作元数据存储的配置

属　性	描　述
hive.metastore.local	控制是否在客户端机器创建和使用一个本地metastore服务器。设为false，则使用远端的metastore服务器
javax.jdo.option.ConnectionURL	JDBC连接的URL，指定用于metastore的数据库[①]。例如jdbc:mysql://<hostname>/metastore_db
javax.jdo.option.ConnectionDriverName	JDBC驱动器的类名，例如com.mysql.jdbc.Driver
javax.jdo.option.ConnectionUserName	登录数据库的用户名
javax.jdo.option.ConnectionPassword	登录数据库的密码

一旦安装好了数据库，或者你仅为评估Hive而使用了它默认的单用户模式，都可以开始使用Hive的CLI。在$HIVE_HOME目录中键入/bin/hive，就会出现Hive的提示符，供你输入Hive命令。

```
bin/hive
Hive history file=/tmp/root/hive_job_log_root_200908240830_797162695.txt
hive>
```

11.1.2　查询的示例

在正式讲解HiveQL之前，先在命令行方式下运行几条命令是有好处的。可以感觉一下HiveQL是如何工作的，也可以自己随便探索一下。

假设你在本机上有专利引用数据cite75_99.txt，这是一个以逗号分隔的专利引用数据集。在Hive中，我们首先定义一个存储该数据的表：

```
hive> CREATE TABLE cite (citing INT, cited INT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.246 seconds
```

HiveQL语句以分号结尾。如上所示，一条语句可以跨多行，直到分号才结束。

这个4行命令的大部分动作在第一行。这里我们定义一个包含两个列的表，名为cite。第一列叫citing，类型为INT，而第二列叫cited，也是INT类型。命令的其他行告诉Hive该数据的存储方式（文本文件）和解析方式（以逗号分隔的字段）。

① MySQL JDBC驱动器的完整格式描述见http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html。

www.ChinaDBA.net 中国DBA超级论坛

通过SHOW TABLES命令，我们可以看到Hive中当前的表：

```
hive> SHOW TABLES;
OK
cite
Time taken: 0.053 seconds
```

在Hive的“OK”和“Time taken”消息之间，可以看到有一个名为cite的表。还可以通过DESCRIBE命令检查它的schema：

```
hive> DESCRIBE cite;
OK
citing  int
cited   int
Time taken: 0.13 seconds
```

和预期的一样，表中包含我们定义的两个列。在HiveQL中对表的管理和定义类似于标准的关系数据库。下面将专利引用数据加载到这个表中。

```
hive> LOAD DATA LOCAL INPATH 'cite75_99.txt'
    > OVERWRITE INTO TABLE cite;
Copying data from file:/root/cite75_99.txt
Loading data to table cite
OK
Time taken: 9.51 seconds
```

这告诉Hive从本地文件系统上一个名为cite75_99.txt的文件中把数据加载到cite表中。在此过程中，本地计算机会把数据上传到HDFS，放在Hive管理的某些目录下。（除非你更改了配置，否则这些目录会位于/user/hive/warehouse下。）

当加载数据时，Hive不会让任何违反其schema的数据进入表中。Hive会将这些数据替换为空值。我们可以使用一个简单的SELECT语句来浏览cite表中的数据：

```
hive> SELECT * FROM cite LIMIT 10;
OK
NULL    NULL
3858241 956203
3858241 1324234
3858241 3398406
3858241 3557384
3858241 3634889
3858242 1515701
3858242 3319261
3858242 3668705
3858242 3707004
Time taken: 0.17 seconds
```

我们的schema将这两个列定义为整数。我们看到有一行为空值，表明有一条记录违反了schema。这是由于cite75_99.txt的第一行包含的是列名，而不是专利号码。总的来说，不是大问题。

既然我们已经确信Hive读到了数据并进行了管理，就可以对它进行各种查询。首先统计表的行数。SQL中有一个人们所熟知的实现，即SELECT COUNT(*)。这里HiveQL在语法上略有不同：

www.ChinaDBA.net 中国DBA超级论坛

```
hive> SELECT COUNT(1) FROM cite;
Total MapReduce jobs = 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_200908250716_0001, Tracking URL = http://ip-10-244-199-
143.ec2.internal:50030/jobdetails.jsp?jobid=job_200908250716_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=ip-10-
244-199-143.ec2.internal:9001 -kill job_200908250716_0001
 map = 0%,  reduce =0%
 map = 12%,  reduce =0%
 map = 25%,  reduce =0%
 map = 30%,  reduce =0%
 map = 34%,  reduce =0%
 map = 43%,  reduce =0%
 map = 53%,  reduce =0%
 map = 62%,  reduce =0%
 map = 71%,  reduce =0%
 map = 75%,  reduce =0%
 map = 79%,  reduce =0%
 map = 88%,  reduce =0%
 map = 97%,  reduce =0%
 map = 99%,  reduce =0%
 map = 100%,  reduce =0%
 map = 100%,  reduce =67%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0001
OK
16522439
Time taken: 85.153 seconds
```

通过这些消息，可以知道该查询生成了一个MapReduce作业。Hive之美在于用户根本不需要知道MapReduce的存在。用户所需关心的，仅仅是使用一种类似于SQL的语言来查询数据库。

上述查询的结果被直接输出在屏幕上。大多数情况下，查询结果应该被存盘，而且通常被放在其他的Hive表中。我们的下一个查询会查找每个专利的引用频率。首先生成一个表来存储它的结果：

```
hive> CREATE TABLE cite_count (cited INT, count INT);
OK
Time taken: 0.027 seconds
```

我们可以执行一个查询来得到引用频率。这个查询再次使用与SQL相类似的COUNT和GROUP BY功能。再加上INSERT OVERWRITE TABLE，就可以让Hive将结果写入表中：

```
hive> INSERT OVERWRITE TABLE cite_count
    > SELECT cited, COUNT(citing)
    > FROM cite
    > GROUP BY cited;
```

www.ChinaDBA.net 中国DBA超级论坛

```
...
 map = 100%,  reduce =89%
 map = 100%,  reduce =90%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0002
Loading data to table cite_count
3258984 Rows loaded to cite_count
OK
Time taken: 103.331 seconds
```

查询执行告诉我们有3 258 984行被装载到引用频率表中。我们可以执行更多的HiveQL语句来浏览这个引用频率表：

```
hive> SELECT * FROM cite_count WHERE count > 10 LIMIT 10;
Total MapReduce jobs = 1
Number of reduce tasks is set to 0 as there's no reduce operator
...
 map = 80%,  reduce =0%
 map = 100%,  reduce =100%
Ended Job = job_200908250716_0003
OK
163404  13
164184  16
217584  13
246144  14
288134  11
347644  11
366494  11
443764  11
459844  13
490484  13
```

这个查询的一个有趣的地方是Hive足够智能，它知道“Number of reduce tasks is set to 0 as there's no reduce operator”（由于没有reduce运算符，reduce任务个数被置为0）。

当你已经使用完这个表时，可以使用DROP TABLE命令来删除它：

```
hive> DROP TABLE cite_count;
OK
Time taken: 0.024 seconds
```

使用这个命令时要小心。它不会让你确认是否真的想删除这个表。一旦你删除了这个表，就会很难恢复。

最终，你可以使用exit命令来退出Hive会话。

11.1.3 深入 HiveQL

在对Hive有了实际体验之后，我们现在就可以正式地深入观察HiveQL的不同方面及其用法了。

1. 数据模型

我们看到Hive将表作为基本的数据模型。物理上，Hive将表以目录形式放在/user/hive/warehouse下。例如，我们前面生成的cite表将数据放在/user/hive/warehouse/cite目录下。输出结果

www.ChinaDBA.net 中国DBA超级论坛

的cite_count表则被放在/user/hive/warehouse/cite_count目录下。在大多数基本设置中，一个表下面的目录结构只有一层，表中的数据分散在一个目录下的多个文件中。

关系数据库在列上使用索引来加速对这些列的查询。Hive则使用了分区列（partition column）的概念，根据列的值将表划分为多个分区。例如，state列将表划分为50个分区，每个州有一个分区[①]。date列是用作日志数据的分区列，每天的数据归属于其自己的分区。Hive将分区列与常规的数据列区别看待，在分区列上执行查询要高效得多。这是因为Hive在物理上将不同的分区存在不同的目录中。例如，假设你有一个名为users的表，包含data和state两个分区列（加上常规的数据列）。Hive将为这个表生成如下的目录结构：

```
/user/hive/warehouse/users/date=20090901/state=CA
/user/hive/warehouse/users/date=20090901/state=NY
/user/hive/warehouse/users/date=20090901/state=TX
...
/user/hive/warehouse/users/date=20090902/state=CA
/user/hive/warehouse/users/date=20090902/state=NY
/user/hive/warehouse/users/date=20090902/state=TX
...
/user/hive/warehouse/users/date=20090903/state=CA
/user/hive/warehouse/users/date=20090903/state=NY
/user/hive/warehouse/users/date=20090903/state=TX
...
```

所有加利福尼亚州（state=CA）2009年9月1日（date=20090901）的用户数据放在一个目录中，而其他分区的数据放在其他目录中。如果进来一个查询，它请求加利福尼亚州在2009年9月1日的用户数据，Hive仅需处理那个目录下的数据，而不用管其他分区中存储的users表中的数据。对分区列的跨区查询涉及对多个目录的处理，但是Hive仍能避免去扫描users中的所有数据。某种程度上，分区为Hive带来的好处类似于索引对传统关系数据库的作用，但是分区在细粒度方面远远不及索引。你会想让每个分区的数据仍然足够大，使得对其运行MapReduce作业仍然有不错的效率。

除了分区，Hive数据模型还应用了桶（bucket）的概念，它可以提供对随机样本数据的高效查询。（例如，当计算一个列的平均值时，数据的随机样本可以提供一个很好的近似结果。）基于对桶列（bucket column）的散列，桶将分区的数据进一步划分为特定数目的文件。如果根据users表中的用户id，我们将桶的个数设定为32，那么Hive中的表将会有如下的完整文件结构：

```
/user/hive/warehouse/users/date=20090901/state=CA/part-00000
...
/user/hive/warehouse/users/date=20090901/state=CA/part-00031
/user/hive/warehouse/users/date=20090901/state=NY/part-00000
...
/user/hive/warehouse/users/date=20090901/state=NY/part-00031
/user/hive/warehouse/users/date=20090901/state=TX/part-00000
...
```

每个分区有32个桶。通过基于用户id的分桶，Hive会知道part-00000 ... part-00031中的

① 实际中你还必须处理哥伦比亚特区和各个准州。

www.ChinaDBA.net 中国DBA超级论坛

每个文件都是一个用户的随机样本。许多汇总统计的计算在样本数据集上依然有相当好的精度。分桶对于加速一些查询特别有用。例如，Hive的查询可以用part-00000这段数据，它仅为一个分区上所有用户的1/32，而不用去读其他文件。不使用分桶，Hive也可以进行采样（基于列而不是桶列），但是这会涉及对所有数据的扫描，再随机地忽略大部分数据。这样，由采样所带来的大部分效率都因而丧失了。

2. 表的管理

我们已经看到如何为专利引用数据集生成一个样本表。现在让我们剖析一个更为复杂的表生成语句。这次生成一个名为page_view的表。

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
            page_url STRING, referrer_url STRING,
            ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY (dt STRING, country STRING)
CLUSTERED BY (userid) INTO 32 BUCKETS
ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      LINES TERMINATED BY '\n'
STORED AS SEQUENCEFILE;
```

第一部分看起来很像SQL中等价的语句：

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
            page_url STRING, referrer_url STRING,
            ip STRING COMMENT 'IP Address of the User')
```

它指定了表名（page_view）及其schema，包含列名和它们的类型。Hive支持如下的数据类型。

- TINYINT——单字节整数。
- SMALLINT——双字节整数。
- INT——四字节整数。
- BIGINT——八字节整数。
- DOUBLE——双精度浮点数。
- STRING——字符序列。

注意这里没有布尔类型，通常用TINYINT来替代。Hive还支持复杂的数据类型，如结构、映射和数组，它们可以嵌套。但是它们目前在语言中的支持还不够好，这涉及更高级的话题。

我们可以在每一列上附加一个描述性注释，就像这里对ip列所做的一样。而且，我们在表上也增加了一个描述性注释：

```
COMMENT 'This is the page view table'
```

CREATE TABLE语句的下一部分是指定分区列：

```
PARTITIONED BY (dt STRING, country STRING)
```

我们在前面讨论过，分区列面向查询进行了优化。它们不同于数据列viewTime、userid、page_url、referrer_url和ip。对于特定的行，分区列的值并没有显式地在行中存储，它隐含

www.ChinaDBA.net 中国DBA超级论坛

在目录路径中。但是，分区列和数据列的查询在语法上并无差别。

```
CLUSTERED BY (userid) INTO 32 BUCKETS
```

CLUSTERED BY (...) INTO ... BUCKETS语句指定了分桶信息，其中包含参与随机采样的列，以及生成的桶数。桶数的选择依赖于以下标准：

(1) 每个分区下数据的大小；

(2) 打算使用的样本大小。

第一个标准很重要，因为在分区被进一步划分为指定个数的桶之后，你并不想让每个桶文件太小而导致Hadoop的效率很低。另一方面，桶应该和你所用样本有相同的大小或者比它更小。如果样本大小为用户基数的百分之三（1/32）左右，基于用户将分桶数设为32就是一个很好的选择。

注意　与分区不同，在数据被写入表时，Hive不会自动地强制分桶。指定分桶信息仅会告知Hive当数据写入一个表时，你会手动地强制分桶（取样）的标准，而Hive可以在处理查询时利用这一优势。为了强制设置分桶的标准，你需要在填充表时正确地设置reducer的个数。更多细节见http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL/BucketedTables。

ROW FORMAT子句告诉Hive表中数据按行存储的方式。若无此子句，Hive默认以换行符作为行分隔符，以001（Ctrl+A）的ASCII值作为字段分隔符。我们的子句告诉Hive改为使用制表符字符作为字段分隔符。我们还告诉Hive使用换行符作为行分隔符，但那已经是默认值，我们在这里包括它只是为了演示而已：

```
ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      LINES TERMINATED BY '\n'
```

最终，最后一个子句告诉Hive用于存储表中数据的文件格式：

```
STORED AS SEQUENCEFILE;
```

目前Hive支持两种格式，SEQUENCEFILE和TEXTFILE。序列文件是一种压缩的格式，通常可提供更高的性能。

我们可以在CREATE TABLE语句中添加一个EXTERNAL修饰符，这样表被创建为指向一个现有的数据目录。你需要指定此目录的位置。

```
CREATE EXTERNAL TABLE page_view(viewTime INT, userid BIGINT,
                    page_url STRING, referrer_url STRING, ip STRING)
LOCATION '/path/to/existing/table/in/HDFS';
```

在生成一个表之后，你可以用DESCRIBE命令在Hive中查询这个表的schema：

```
hive> DESCRIBE page_view;
```

你还可以用ALTER命令更改表的结构，包括更改表的名称：

```
hive> ALTER TABLE page_view RENAME TO pv;
```

www.ChinaDBA.net 中国DBA超级论坛

或增加新的列：

```
hive> ALTER TABLE page_view ADD COLUMNS (newcol STRING);
```

或删除一个分区：

```
hive> ALTER TABLE page_view DROP PARTITION (dt='2009-09-01');
```

要删除整个表，使用DROP TABLE命令：

```
hive> DROP TABLE page_view;
```

要知道Hive正在管理着哪些表，你可以用SHOW命令把它们都显示出来：

```
hive> SHOW TABLES;
```

如果正在使用的表有很多，将难以把它们都列出来，你可以用Java正则表达式缩小其结果：

```
hive> SHOW TABLES 'page_.*';
```

3. 装载数据

将数据装载到一个Hive表中的方法有很多。主要是LOAD DATA命令：

```
hive> LOAD DATA LOCAL INPATH 'page_view.txt'
    > OVERWRITE INTO TABLE page_view;
```

它取一个名为page_view.txt的本地文件，并将其内容加载到page_view表。如果我们忽略了OVERWRITE修饰符，内容会被添加到表中，而不是替换已存在的所有内容。如果我们忽略LOCAL修饰符，文件会取自HDFS而不是本地文件系统。此外，还可以通过LOAD DATA命令在表中指定一个分区来加载数据：

```
hive> LOAD DATA LOCAL INPATH 'page_view.txt'
    > OVERWRITE INTO TABLE page_view
    > PARTITION (dt='2009-09-01', country='US');
```

当操作来自本地文件系统中的数据时，可以执行Hive CLI中的本地Unix命令，这很有用。需要在命令前加上感叹号（!），命令尾部加上分号（;）。例如，可以获取一个文件的列表

```
hive> ! ls ;
```

或检查文件的前几行

```
hive> ! head hive_result ;
```

请注意,! 和 ; 周围的空格不是必要的。我们添加它们是为了提高可读性。

4. 执行查询

大多数情况下，运行HiveQL查询与SQL查询惊人地相似。一个通常的差别为HiveQL查询的结果相对较大。几乎总是有一个INSERT子句，以便告诉Hive把查询结果存起来。往往存到其他的表中：

```
INSERT OVERWRITE TABLE query_result
```

也会是HDFS的一个目录：

```
INSERT OVERWRITE DIRECTORY '/hdfs_dir/query_result'
```

www.ChinaDBA.net 中国DBA超级论坛

有时还会是一个本地目录：

```
INSERT OVERWRITE LOCAL DIRECTORY '/local_dir/query_result'
```

可以看到基本查询几乎与SQL完全相同：

```
INSERT OVERWRITE TABLE query_result
SELECT *
FROM page_view
WHERE country='US';
```

请注意查询作用于分区列（country）上，但如果是数据列，查询看起来也完全一样。只有一个语法的调整，即HiveQL使用了COUNT(1)以替代SQL通常在此处使用的COUNT(*)。例如，将使用如下的HiveQL查询查找来自美国的页面浏览数量：

```
SELECT COUNT(1)
FROM page_view
WHERE country='US';
```

像SQL一样，GROUP BY子句允许在组上执行聚合查询。此查询将列出每个国家的页面浏览数量：

```
SELECT country, COUNT(1)
FROM page_view
GROUP BY country;
```

www.ChinaDBA.net 中国DBA超级论坛

而这个查询将列出每个国家的唯一用户数：

```
SELECT country, COUNT(DISTINCT userid)
FROM page_view
GROUP BY country;
```

表11-2显示在HiveQL中支持的所有运算符。这些运算符在SQL和编程语言中都是很标准的，我们不会详细解释它们。主要的例外是正则表达式的匹配。HiveQL提供了两个正则表达式匹配的命令——LIKE和REGEXP。（RLIKE等同于REGEXP。）LIKE仅执行简单的SQL正则表达式匹配，如B中的下划线（_）字符匹配A中任意单个字符，而百分号（%）字符匹配A中任意的数字符号。REGEXP将B视为一个完整的正则表达式[①]。表11-3和表11-4列出了主要的HiveQL函数。

表11-2　HiveQL中的标准运算符

运算符类型	运　算　符
比较运算符	A = B , A <> B , A < B , A <= B , A > B , A >= B , A IS NULL , A IS NOT NULL , A LIKE B , NOT A LIKE B , A RLIKE B , A REGEXP B
算术运算符	A + B , A - B , A * B , A / B , A % B
按位运算符	A & B , A | B , A ^ B, ~A
逻辑运算符	A AND B, A && B, A OR B, A || B, NOT A, !A

www.ChinaDBA.net 中国DBA超级论坛

① http://jakarta.apache.org/oro/index.html 的按此解释是 Java 实现

表11-3 内置函数

函 数	描 述
concat(string a, string b)	返回字符串a与字符串b的串接结果
substr(string str, int start) substr(string str, int start, int length)	返回从start开始的str的子字符串。结果直到字符串尾部，除非特别设定了length参数
round(double num)	返回最近的整数（BIGINT）
floor(double num)	返回等于或小于num的最大整数（BIGINT）
ceil(double num) ceiling(double num)	返回等于或大于num的最小整数（BIGINT）
sqrt(double num)	返回num的平方根
rand() rand(int seed)	返回一个随机数（每行不同）。指定选项seed的值形成确定的随机数序列
Ln(double num)	返回num的自然对数
log2(double num)	返回num以2为底的对数
log10(double num)	返回num以10为底的对数
log(double num) log(double base, double num)	返回num的自然对数，或者返回以base为底的对数
exp(double a)	a的e次幂（自然对数的底）
power(double a, double b) pow(double a, double b)	返回a的b次幂
upper(string s) ucase(string s)	返回大写的字符串s
lower(string s) lcase(string s)	返回小写的字符串s
trim(string s)	返回字符串s两端去掉空格的结果
ltrim(string s)	返回字符串s左端去掉空格的结果
rtrim(string s)	返回字符串s右端去掉空格的结果
regexp(string s, string regex)	返回字符串s是否与Java正则表达式regex相匹配
regexp_replace(string s, string regex, string replacement)	返回一个字符串，其中所有与Java的正则表达式regex匹配的都被替换为replacement
day(string date) dayofmonth(string date)	返回日期或时间戳字符串中天的部分
month(string date)	返回日期或时间戳字符串中月的部分
year(string date)	返回日期或时间戳字符串中年的部分
To_date(string timestamp)	返回一个时间戳字符串中日期的部分（年-月-日）
unix_timestamp(string timestamp)	将时间戳字符串转换为Unix时间
from_unixtime(int unixtime)	将整型的Unix时间转换为时间戳字符串
date_add(string date, int days)	在date字符串中增加天数
date_sub(string date, int days)	从date字符串中减去天数
datediff(string date1, string date2)	计算天数的不同，如果date1更早，则结果为负。

www.ChinaDBA.net 中国DBA超级论坛

表11-4 内置聚合函数

函 数	描 述
count(1) count(DISTINCT *col*)	返回一个组中的成员个数，或者该列中不重复值的数量
sum(*col*) sum(DISTINCT *col*)	返回列上值的总和，或者该列中不重复值的和
avg(*col*) avg(DISTINCT *col*)	返回列的平均值，或者该列中不重复值的平均值
max(*col*)	返回该列中的最大值
min(*col*)	返回该列中的最小值

对于用户而言，寻求Pig Latin和HiveQL这种更高级的语言的一个主要动力在于支持联结操作。目前HiveQL只支持等联结。连接查询的示例如下：

```
INSERT OVERWRITE TABLE query_result
SELECT pv.*, u.gender, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.id);
```

在语法上，将关键字JOIN添加到FROM子句中两个表之间，然后在关键字ON后面指定联结的列。若要联结两个以上的表，我们重复这种模式：

```
INSERT OVERWRITE TABLE pv_friends
SELECT pv.*, u.gender, u.age, f.friends
FROM page_view pv JOIN user u ON (pv.userid = u.id)
                  JOIN friend_list f ON (u.id = f.uid);
```

我们可以通过修改FROM子句为任何查询添加采样。该查询会去计算平均浏览时间，除非这个平均值仅取自32个桶中第一桶的数据：

```
SELECT avg(viewTime)
FROM page_view TABLESAMPLE(BUCKET 1 OUT OF 32);
```

TABLESAMPLE的一般语法如下：

```
TABLESAMPLE(BUCKET x OUT OF y)
```

查询样本的大小为1/y左右。此外，y为表创建时所指定的桶数的倍数或因数。例如，如果我们将y改为16，查询为

```
SELECT avg(viewTime)
FROM page_view TABLESAMPLE(BUCKET 1 OUT OF 16);
```

那么样本的大小为大约每16个用户中取一个（桶列为userid）。表仍有32个桶，但Hive仅取1和17并一起处理以满足该查询。另一方面，如果设y为64，则Hive会对单个桶中的一半数据执行查询。x的值仅用于选择要使用的桶。在真正随机的抽样下，它的取值不会产生什么影响。

除了avg，Hive还有很多其他的内置函数。你可以在表11-3和表11-4中看到一些较为常见的函数。程序员还可以在Hive中添加UDF来定制处理函数。如何创建UDF的简介见http://wiki.apache.org/hadoop/Hive/AdminManual/Plugins。

www.ChinaDBA.net 中国DBA超级论坛

11.1.4 Hive 小结

Hive是建立在Hadoop大规模可扩展系统结构之上的数据仓库层。通过把重点放在结构化数据上，Hive添加了许多性能强化技术（如分区）以及可用性特征（如类SQL语言）。它使某些常见的任务（如联结）变得容易。Hive将Hadoop技术推广给更多的数据分析师和其他非编程人员。至2009年8月，Facebook统计其雇员中有29%是Hive的用户，其中一半以上为非工程人员[①]。

11.2 其他 Hadoop 相关的部分

Hadoop生态系统每天都在成长。下面为与Hadoop相关的项目或第三方软件，它们很实用或是有巨大的潜力。除了Aster Data和Greenplum之外，它们在某种程度上都是开源的。

11.2.1 HBase

HBase（http://hadoop.apache.org/hbase/）是一个可扩展的数据存储系统，旨在用于随机读写访问（相当于）结构化的数据。它的设计源自谷歌的Bigtable[②]，旨在支持大表，即包含数十亿计的行和数以百万计的列。它使用HDFS作为底层文件系统，以达到完全分布式和高可用性。版本0.20在性能上有了显著的提升。

11.2.2 ZooKeeper

ZooKeeper（http://hadoop.apache.org/zookeeper/）是用于构建大型分布式应用的一种协作式服务。你可以在Hadoop Core框架之外独立使用它。它实现了许多在大型分布式应用中常见的服务，如配置管理、命名、同步和组服务。从历史上看，开发人员不得不为每个分布式应用重塑这些服务，这样做不仅耗时，而且容易出错，因为这些服务难以正确实现。通过将底层的复杂性剥离出来，ZooKeeper使应用能够很方便地实现consensus（会商）、leader election（领导者选举）、presense protocol（存在协议）和其他原语。这解放了开发人员，使其可以专注于应用程序的语义。ZooKeeper常作为其他Hadoop相关项目的主要组件，如HBase和Katta。

11.2.3 Cascading

Cascading（http://www.cascading.org/）是Hadoop上用于组装和执行复杂数据处理工作流的一个API。它从MapReduce模型中抽象出包括元组（tuple）、管道（pipe）和出水/入水口（tap）的数据处理模型。管道对元组的流进行操作，其操作包括Each、Every、GroupBy和CoGroup。管道还可以组装和嵌套，以创建“程序集”（assembly）。当我们将管道组件接上（数据）入水口和（数

11

① 这件事引自http://www.facebook.com/note.php?note_id=114588058858。在http://www.facebook.com/note.php?note_id=16121578919中解释了Facebook决定如何构建其Hadoop基础设施。在http://www.slideshare.net/zshao/hive-data-warehousing-analytics-on-hadoop-presentation中详细介绍了Facebook如何围绕Hive来设计其数据仓库及分析系统。

② “Bigtable: A Distributed Storage System for Structured Data” by Chang et al., OSDI ‘06—Seventh Symposium on Operating System Design and Implementation，http://labs.google.com/papers/bigtable.html。

www.ChinaDBA.net 中国DBA超级论坛

据）出水口时，就生成了一个可以执行的“流”（flow）。

Cascading与Pig之间有许多相似的设计与目标。不过，一个区别在于Pig的Grunt便于执行临时查询。还有一个区别是Pig程序都用Pig Latin编写，而Cascading更像一个Java框架，你要通过实例化各种Java类（Each和Every等）来创建数据的处理流程。使用Cascading不需要学习新的语言，而且创建的数据处理流程效率更高，因为它是直接由你自己写的。

11.2.4 Cloudera

Cloudera（http://www.cloudera.com/）试图在Hadoop中充当RedHat在Linux中的角色。它支持和封装Hadoop，以便对企业用户易用、友好。它在主要城市开设了实时培训课程，并在他们的网站上提供了教育视频。通过使用他们以RPM或Ubuntu/Debian包的形式提供的免费Hadoop发行版，可以简化Hadoop的部署。他们的Hadoop发行版基于Hadoop最新的稳定版本、未来发行版中有用（且已测试）的补丁，以及Pig和Hive等额外的工具。Cloudera还提供咨询和支持服务，帮助企业使用Hadoop。

11.2.5 Katta

其网址为http://katta.sourceforge.net/。因为Hadoop起源可以追溯到搜索引擎，它自然可以被应用到分布式索引和查询中。Nutch是以Hadoop为基础构建的Web搜索引擎[①]。但作为一个Web搜索引擎，Nutch有许多特别的限定。这使它往往无法成为与特定搜索应用相匹配的解决方案。

Katta是一个可扩展、容错、分布式的索引系统。它比Nutch更轻量、灵活。在某种意义上它为Lucene添加了一些额外的功能（如复制、冗余、容错和可扩展性），同时保留了基本应用程序的语义。

11.2.6 CloudBase

CloudBase（http://cloudbase.sourceforge.net/）是一个架构在Hadoop之上的ANSI SQL数据仓库层。与Hive不同，CloudBase直接工作在平面文件上，没有任何元数据存储区。它更严格地与ANSI SQL保持一致，而交互主要是通过JDBC驱动器，这使其更容易连接到商业智能报告工具。大多数情况下，CloudBase充当了编译器的角色，它获取SQL查询并将它们编译为MapReduce程序。写这篇文章时，相比于Pig和Hive而言，CloudBase开发者社区不够活跃，而其GPL许可证比Apache许可的限制性更强。

11.2.7 Aster Data 和 Greenplum

Aster Data Systems（http://www.asterdata.com/）和Greenplum（http://www.greenplum.com/）这两个商业公司提供了高性能、可扩展的数据仓库解决方案，将SQL与MapReduce紧密结合在一起。虽然它们支持MapReduce编程模型，但均独立于Hadoop之外，且采取了许多不同的底层设计

① 也许更准确的说法是Nutch激励了Hadoop的产生。更完整的故事见第1章。

www.ChinaDBA.net 中国DBA超级论坛

方案。与Hadoop不同的是，它们的产品采用了特别针对企业用户的设计架构，以实现更高性能的SQL数据仓库。因为它们在MapReduce范式的实现上采用了与Hadoop不同的角度，学习它们有助于理解Hadoop在架构上所做的一些权衡。

11.2.8 Hama 和 Mahout

Hama（http://incubator.apache.org/hama/）和Mahout（http://lucene.apache.org/mahout/）都是使用Hadoop进行科学数据处理的项目。Hama是矩阵计算软件包，用于计算乘积、逆、特征值、特征向量和其他的矩阵运算。Mahout更专门针对基于Hadoop实现机器学习算法（更多的信息请参阅Manning出版社的*Mahout in Action*）。Mahout 0.1版于2009年4月发布，包含诸如Naïve Bayes分类、k-means聚类和协同过滤等算法的实现。

在写本书时，这两个项目都相对较新，位于Apache incubator（孵化器）中。感兴趣的读者可以考虑成为这些项目的参与者。

11.2.9 search-hadoop.com

作为一名Hadoop程序员，你经常需要找一些有关Hadoop或其子项目的文档。Sematext，一家专业从事搜索和分析的公司，运行了网站http://search-hadoop.com/，可以供你搜索所有Hadoop的子项目和数据源——邮寄列表档案、Wiki、问题跟踪系统、源代码等。基本的查询索引仍在不断更新。搜索结果允许按照项目、数据源和作者进行筛选，并可以按日期、关联或二者的组合进行排序。

11.3 小结

本章介绍了许多附加的Hadoop工具。我们特别关注了Hive，它是一个数据仓库软件包，允许你使用类SQL语言来处理Hadoop的数据。围绕着Hadoop的支撑性软件如雨后春笋般涌现，丰富了这个生态系统。在下一章中，你会在实际案例中看到它们的作用。

www.ChinaDBA.net 中国DBA超级论坛

第 12 章 案例研究

12

本章内容

- 《纽约时报》
- 中国移动
- StumbleUpon
- IBM

我们一起做了许多练习和示例程序。下一步是将你所学到的Hadoop知识和真实世界中的应用相结合。为了帮助你完成这一转变，本章提供一些企业案例，说明它们是如何通过Hadoop为数据处理问题提供解决方案的。

案例研究有两个目的。一个是从更大的范围来审视那些将Hadoop作为其核心部分的系统。你会发现一些辅助性的工具，如Cascading、HBase和Jaql。第二个目的是说明对于多样性的业务，Hadoop均为它们解决了在运营中所面临的挑战。我们的案例研究横跨多个行业，包括媒体（《纽约时报》）、电讯（中国移动）、互联网（StumbleUpon）和企业软件（IBM）。

12.1 转换《纽约时报》1100 万个库存图片文档

2007年，《纽约时报》决定在其网站上免费发布他们在1851年和1922年之间所有公开的文章。这样做需要一个可扩展的图像转换系统。由于《纽约时报》已经把较早的文章存储为扫描的TIFF图像，需要处理这些图像来合并每篇文章的不同部分，以得到一个预期的PDF格式文件。在此之前，这些文章需要付费，没有多少流量。因此，《纽约时报》可以按照实时的处理方式来伸缩、拼接并将其转换为TIFF图像。这种方式对少量请求是足够的，但是它无法扩展以应对开放文档后的大规模请求增长。《纽约时报》需要一个更好的架构来支持文档开放。

解决方案是把所有文章预处理为PDF文件，让它就像所有其他静态内容一样。《纽约时报》已经有将TIFF图像转换为PDF文件的代码。它看起来就像一个简单的批处理，对所有的文章一视同仁，而不会对每篇文章的请求进行单独处理。该方案存在的挑战是人们知道这里归档了由1100万篇文章组成的4 TB数据。

Derek Gottfrid是《纽约时报》的一名软件程序员，他认为这是一个绝好的机会来使用AWS

www.ChinaDBA.net 中国DBA超级论坛

和Hadoop。在Amazon S3上存储最后的PDF数据集并提供服务，这已经被认为是一项比升级网站的后端存储更划算的办法。为什么不把PDF的处理过程也放在AWS云上呢？

Derek将4 TB的TIFF图像复制到S3上。他“开始编写代码来从S3上获取文章的各个部分，从中生成一个PDF，并将其写回S3。使用JetS3t令其非常容易，JetS3t包括S3的开源Java工具包、iText PDF库，以及安装Java高级图像扩展”[①]。在调整他的代码使其可以在Hadoop框架运行之后，Derek将之部署到Hadoop上，它在Amazon EC2的100个节点上运行。这个作业运行了24个小时，生成了另外1.5 TB的数据，被存储到S3中。

按每小时每个实例10美分计算，整个工作的计算最终只需花费240美元（100实例×24小时×0.10美元）。这不包括S3的存储开销，不过《纽约时报》已经决定在S3上存放它的文件，这个成本已经被分摊了。S3和EC2之间的数据传输是免费的，Hadoop的作业不会形成任何的带宽费用。

全部工作只由一个员工完成。借助Derek的努力，它让人们可以更容易地找到《纽约时报》对历史事件的记录。

12.2 挖掘中国移动的数据

由中国移动通信公司研究院的罗治国、徐萌和孙少陵撰写

CMCC（中国移动通信公司）是世界上最大的移动电话运营商。以CHL代称在纽约证券交易所（NYSE）上市，中国移动在BrandZ全球品牌2009年资产排名中位列第七名，在麦当劳和苹果公司之后，而在通用电器之前。CMCC占有超过三分之二的中国移动电话市场，为5亿个用户的通信需求提供服务。即便在规模上，中国移动也经历了快速的增长。例如，2006年，当用户基数只有3亿时，其用户的年增长率也达到了22%，而话务量每年增长30%，短信业务每年增长41%。

和所有电信运营商一样，中国移动在其通信网络的正常经营过程中也产生了很多数据。例如，每次打电话会生成一个呼叫数据记录（Call Data Record，CDR），其中包括的信息有呼叫者的电话号码、接听者的电话号码、呼叫开始时间、呼叫持续时间、呼叫路由信息，等等。除了CDR，电话网也会在该网络中的不同开关、节点和终端之间产生信号数据。最起码，我们需要这些数据来完成通话，并准确地为客户记账。我们还需要将它用于市场分析、网络调节以及其他用途。

中国移动网络的规模自然导致生成大量的数据。每天网络会产生5 TB到8 TB的CDR数据。一个中国移动的分公司就有超过2000万用户，每天生成的语音业务CDR数据超过100 GB，短信（SMS）业务的CDR数据在100 GB到200 GB之间。此外，一个普通的分公司每天会产生大约48 GB的GPRS（通用分组无线业务）信号数据，以及300 GB的3G信号数据。

中国移动期望通过数据仓库和数据挖掘来提取数据的内在信息，以改善市场营销、网络优化和服务优化。一些典型应用包括：

- 分析用户行为
- 预测客户流失
- 分析服务关联

① http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/。

www.ChinaDBA.net 中国DBA超级论坛

- 分析网络服务质量（QoS）
- 分析信号数据
- 过滤垃圾消息

中国移动曾使用一些知名厂商的商业数据挖掘工具。这些工具的架构设计限制了中国移动当前数据挖掘系统，因为它要求所有的数据在单一的服务器上处理，从而使硬件能力成为性能瓶颈。有一个分公司，它的现有系统是基于商业解决方案的，采用了一台由8个CPU内核、32 GB内存和一个存储阵列组成的Unix服务器。它最多只能支持140万个客户的用户行为分析，仅占该公司用户数据的十分之一。即使处理的数据量在受限的范围内，当前系统对许多应用的执行时间也过长。此外，高端Unix服务器和存储阵列非常昂贵，而且商业软件包并不能很好地支持自定义的算法。

由于现有系统的局限性，中国移动启动了一个试验性项目，在Hadoop上开发一种并行数据挖掘工具，并将其与现有系统对比。他们给这个项目取名为BC-PDM（大云数据挖掘系统），其架构欲达到以下4个目标。

- 高可扩展性——利用Hadoop实现向外扩展的体系结构。
- 低成本——采用廉价商用硬件和免费软件构建。
- 可定制——面向应用的特定业务需求。
- 易使用——类似于商用工具的图形化用户界面。

BC-PDM实现了许多标准的ETL操作和MapReduce数据挖掘算法。ETL操作包括汇总统计、属性处理、数据采样、冗余去除等。它实现了3类算法，其中包括9个数据挖掘算法。3个类包括聚类（如K-means）、分类（如C4.5算法）和关联分析（例如Apriori算法）。MapReduce程序的执行和评估是在一个由256个节点的Hadoop集群中完成的，节点之间通过一台264口的千兆网交换机互连。节点的硬件配置如下所示。

- Datanode/TaskTracker——1路4核Xeon 2.5 GHz CPU，8 GB内存，4×250 GB SATA II硬盘。
- Namenode/JobTracker——2路2核AMD Opteron 2.6 GHz CPU，16 GB内存，4×146 GB SAS硬盘。

中国移动将BC-PDM与现有的数据挖掘方案进行比较，使用了来自中国移动商业分析支撑系统（Business Analysis Support System，BASS）的真实数据。有3个不同的数据集。它们都相当大，而某些较小的评测任务则需要采样后的数据子集。在表12-1中，你会看到数据集的原始大小（大规模）以及采样子集的大小（中小规模）。

表12-1　用于评测的数据集大小

数　据	大　规　模	中等规模	小　规　模
用户行为	12 TB	120 GB	12 GB
用户访问	16 TB	160 GB	16 GB
新服务关联	120 GB	12 GB	1.2 GB

www.ChinaDBA.net 中国DBA超级论坛

中国移动从4个方面评价BC-PDM：正确性、性能、成本和可扩展性。当然，正确性是新系统可用的必要条件。他们通过确认BC-BPM的并行ETL操作可以产生与串行ETL相同的结果，验证了其正确性。另一方面，数据挖掘算法不会产生与现有系统完全相同的结果。这是因为它在实现与执行细节上有所不同，如输入数据的初始条件和顺序，这可能会影响到输出的准确性。他们仔细核对了结果并进行了一致性检查。此外，还采用UCI 数据集来验证BC-PDM并行数据挖掘算法。UCI数据集易于理解且在机器学习社区中的研究者中间深受欢迎。中国移动可以在已知的期望模型下验证BC-PDM的输出。

在确立了MapReduce实现的正确性后，我们将BC-PDM的性能与现有系统进行比较。这次比较仅使用了16个节点的Hadoop集群。我们将看到，小型集群比现有系统所用的单个大型服务器要便宜很多。图12-1显示了ETL操作（左图）和数据挖掘任务（右图）在两种配置下的运行时间。

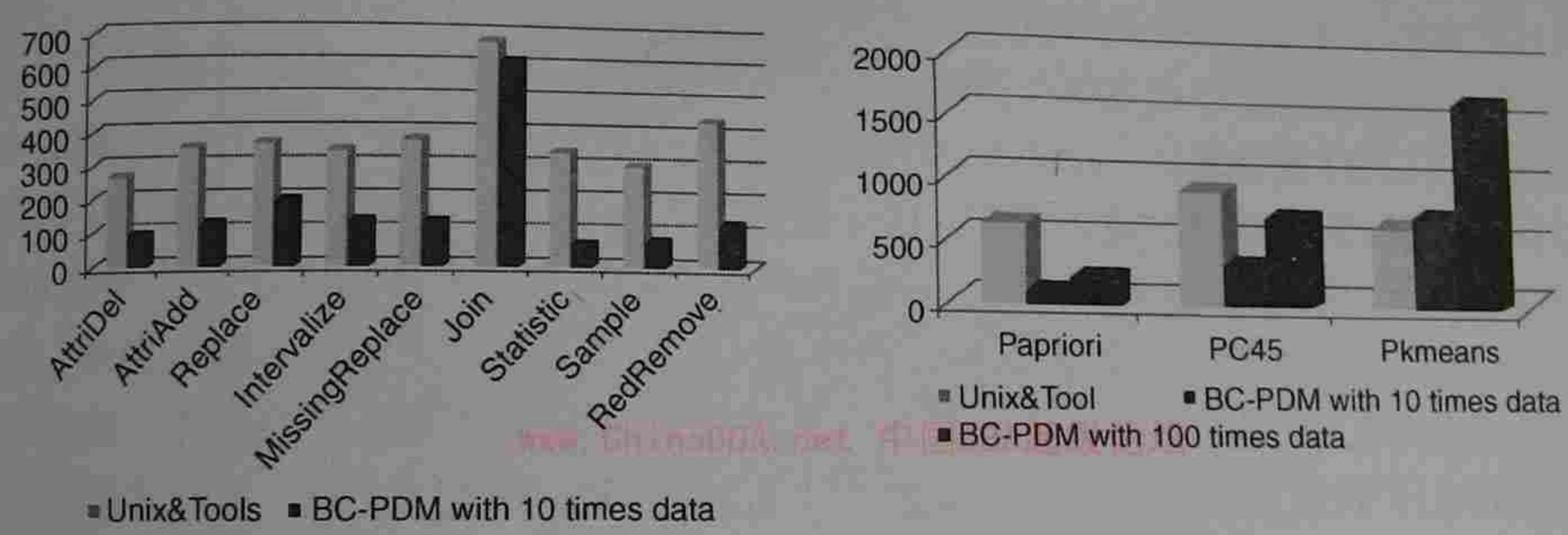


图12-1 Hadoop集群和现有商用Unix服务器的性能比较。左图为ETL测试，而右图为数据挖掘算法

请注意BC-PDM所处理的数据为现有系统的10倍。BC-PDM在所有的ETL操作上都更快，导致12～16倍的性能提升。对于数据挖掘任务，BC-PDM进一步用相当于现有系统100倍的数据量进行了压力测试。即使在高出两个数量级的数据规模下，BC-PDM在Apriori（关联）和C4.5（分类）算法上也比现有系统快得多。在数据规模为10倍时，K-means聚类算法所花的处理时间仅比现有系统稍长一些。对上海分公司3个应用程序进行完整的端到端测试显示，性能提升了3～7倍。这些实际应用程序包括通道偏好建模、新服务关联建模以及订阅者细分建模。回想一下，BC-PDM评测都是基于16个节点的较小Hadoop集群的。稍后我们将看到，当节点增加时，BC-PDM和Hadoop具有良好的扩展性。在具有256个节点的完整集群上，BC-PDM有望存储、处理和挖掘100 TB的数据。

16个节点的BC-PDM集群不只是在性能上超越当前的系统，它也要便宜得多。表12-2给出了这两个系统的成本。16个节点的Hadoop/BC-PDM集群的成本大约为当前商业解决方案的五分之一。最大的节约来自使用了低成本的商用服务器。事实上，16个节点的集群的硬件成本比当前方案成本的十六分之一还要少。

www.ChinaDBA.net 中国DBA超级论坛

表12-2　现有方案和16个节点Hadoop集群的成本与配置比较

		BC-PDM（16个节点）	现有商用Unix服务器
硬件成本	计算能力	CPU：64 核 内存：128 GB	CPU：8 核 内存：32 GB
	存储能力	16 TB (每个节点4 × 256 GB SATA II)	存储阵列
	成本	240 000元	4 000 000元
软件成本	数据库	500 000元	1 000 000元
	应用软件	300 000元	500 000元
	维护成本	200 000元	500 000元
总计		1 240 000元	6 000 000元

至此，我们已经调研了新BC-PDM系统的正确性、性能和成本。我们在集群中添加更多节点来检查其可扩展性。我们在3个集群规模下运行ETL操作和数据挖掘算法：32个节点、64个节点和128个节点。我们以32个节点集群为基准，在更大的集群上评估执行时间的加速比。如图12-2所示，左图为ETL操作的加速比，而右图为数据挖掘操作的加速比。请注意横轴为指数分布（代表集群大小），按照32、64到128成倍设置。由于操作的完全线性可扩展性是我们可以期待的最好结果，理想的加速比曲线会从1到2再到4。我们看到许多ETL操作非常接近理想的线性可扩展性。事实上，当集群大小扩展了4倍，即从32个节点增加到128个节点，除了两个操作之外，都获得了高于2.5倍的加速比[①]。数据挖掘算法更为复杂，但它们也取得了可观的可扩展性。我们早先的性能测试（见图12-1）仅使用了16个节点的集群，在当前的可扩展性测试中，我们甚至没有去考虑这个规模。但是16个节点的集群仍可以比现有的商业解决方案多处理一个数量级的数据。这些测测共同证明了我们的BC-PDM 集群有能力进一步处理100 TB级规模的数据。

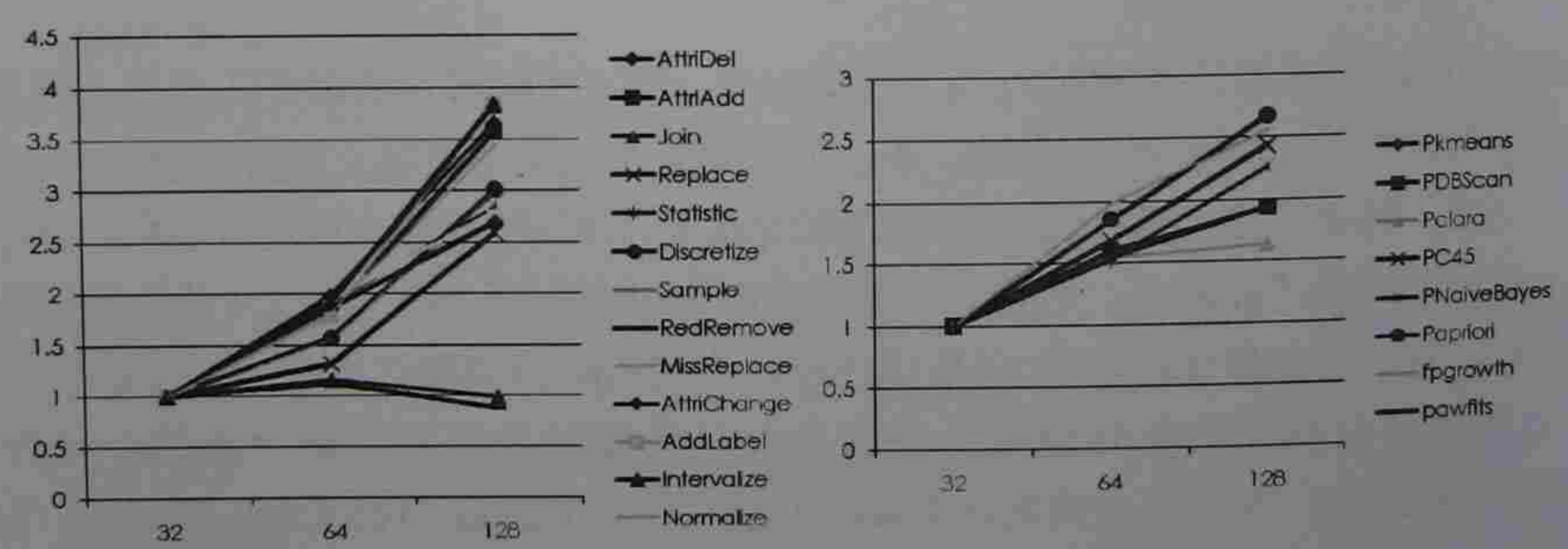


图12-2　ETL（左图）和数据挖掘算法（右图）在Hadoop集群节点增加时的可扩展性。横轴代表BC-PDM集群的节点个数。纵轴代表加速比，这是相对于在32个节点集群上的执行时间来说的

① 这两个例外是Join和Duplicate Removal操作。它们的运行时间基本上不随集群规模改变。我们目前正在研究其改层的原因。Hadoop以固定时间运行作业（独立于集群规模）的一个可能解释是作业分布不均衡，一个任务成为整个作业的瓶颈。

www.ChinaDBA.net 中国DBA超级论坛

彻底评价BC-PDM系统后，我们与上海分公司合作，要将我们的系统应用于其需要的一些业务。一个应用程序是描述其用户群以便精准营销。更具体地说，他们想知道他们的用户是如何分群的，每一个群的特征和差异，为每个用户提供有针对性的市场营销。我们使用数据挖掘工具集中的并行K-means算法来聚类其用户群，并产生市场分割图，如图12-3所示。根据平均账单以及所使用的服务类型做进一步分析，将有助于描述每个群。比起现有的Unix解决方案，BC-PDM可以以3倍速度执行这种分析。

图12-3 对中国移动上海分公司用户群使用K-means算法进行聚类分析，结果可以用作公司的市场营销活动

总之，中国移动作为大型的移动通信提供商，需要不断分析大量数据。当前商业解决方案非常昂贵，且无法支持我们对用户数据的分析。通过对Hadoop的调研，我们在MapReduce和HDFS之上建立了称为BC-PDM的数据挖掘系统，并使之准确、快捷、廉价且可扩展。再向前发展，我们会提高BC-PDM的效率，并通过实现更多的ETL操作和数据挖掘算法对它进行扩展。更重要的是，我们打算在整个中国移动的分公司中推行BC-PDM作为数据分析的服务平台。

12.3 在 StumbleUpon 推荐最佳网站

由Ken Macinnis和Ryan Rawson撰写

StumbleUpon将个人喜好和机器学习相结合，可以立即为使用者提供相关的网页内容。它仅

www.ChinaDBA.net 中国DBA超级论坛

显示那些由其他志趣相投的Stumbler所推荐的网站，每当你单击Stumble按钮时，那些高质量的网站就会根据这些同道网友的意见为你呈现出来。

StumbleUpon 使用“喜欢”和“不喜欢”评分来评价网站质量。当你单击“Stumble”时，你只会看到由朋友和志趣相投的Stumbler所建议的页面。这将有助于你发现在传统搜索引擎中很难找到的内容。

12.3.1 分布式 StumbleUpon 的开端

为了收集和分析这些数据，StumbleUpon需求有高可用的后端平台，以采集、分析和变换每天数以百万计的评级。当前近1000万的用户规模让StumbleUpon的需求很快超过了传统LAMP（Linux、Apache、MySQL、PHP）的支撑能力，因此我们构建分布式平台的原因如下所示。

- 可扩展性——某些情况下，商用硬件容易扩展。20个Hadoop节点的成本可能仅相当于一对冗余互备的数据库。
- 自由开发——与围绕设计完善却有些脆弱的RDBMS进行开发相比，对开发者的限制更少。
- 操控性——尽可能地消除了单点故障对于提供平稳的一流服务操作很重要。
- 数据处理速度——许多系统端的计算任务无法在单机上运行。

12.3.2 HBase 和 StumbleUpon

HBase在StumbleUpon的分布式平台中起着非常重要的作用。HBase是一个分布式的、面向列的数据库，能够利用底层的Hadoop和HDFS平台的能力。但是，与任何复杂系统一样，这里存在权衡： 一方面，HBase搭建了传统关系数据库的概念，如连接、外键关系和触发器；另一方面，系统希望以可扩展的方式在商用硬件上维护庞大而不常用的数据。

1. HBase的介绍

HBase源自于Google的分布式存储系统Bigtable[①]。让我们重温一下Bigtable的基本概念以及与Bigtable类似的系统。

- 分享面向列的数据库和面向行的数据库的概念。如作者所述，Bigtable是一个“稀疏的、分布式的多维度排序的映射”。存储的基本单位是表，它被分成为多个子表（tablet），而在HBase中被称为区域（region）。
- 写操作被缓存在内存中，在一段时间后刷新到只读文件中。
- 为了保持较小的文件个数，它们在紧缩（compaction）过程中被合并，并将N重置为1。
- 特殊的子表或区域被用于跟踪数据的位置。
- 由于数据存储的面向列特征，稀疏表——多数单元为空值——几乎不占空间，因为空值不会显式地存储。
- 列簇用于组合行列。一个簇中的所有列被一起存储（出于局部性考虑），并共享存储和配置参数。

① Bigtable: A Distributed Storage System for Structured Data.Chang,et al.http://labs.google.com/papers/bigtable. html。

www.ChinaDBA.net 中国DBA超级论坛

- 表的单元被存储为多个版本，而不是覆盖现有数据。
- 通过简单地在集群中添加机器，就可以增加容量（存储大小和处理速度），而无须特殊的配置。

HBase提供了许多额外的特征，具体如下所示。

- REST和Thrift[①]网关允许从非Java开发环境方便地进行访问。
- 易于与Hadoop MapReduce集成，用于数据处理。
- 利用Hadoop和HDFS被证明的可靠性和扩展性。
- 基于Web的用户界面，用于管理主服务器和区域服务器。
- 强大的开源社区。

图12-4描述了在HBase 区域服务器中数据写入路径的简化版本。写操作被追加到服务器的写前日志中。

(1) 数据被插入MemStore。

(2) 随着MemStore的增长超出阈值，它被刷新到磁盘上的新文件中。

(3) 当磁盘上的文件太多，数据存储文件将被压缩成较少的文件。

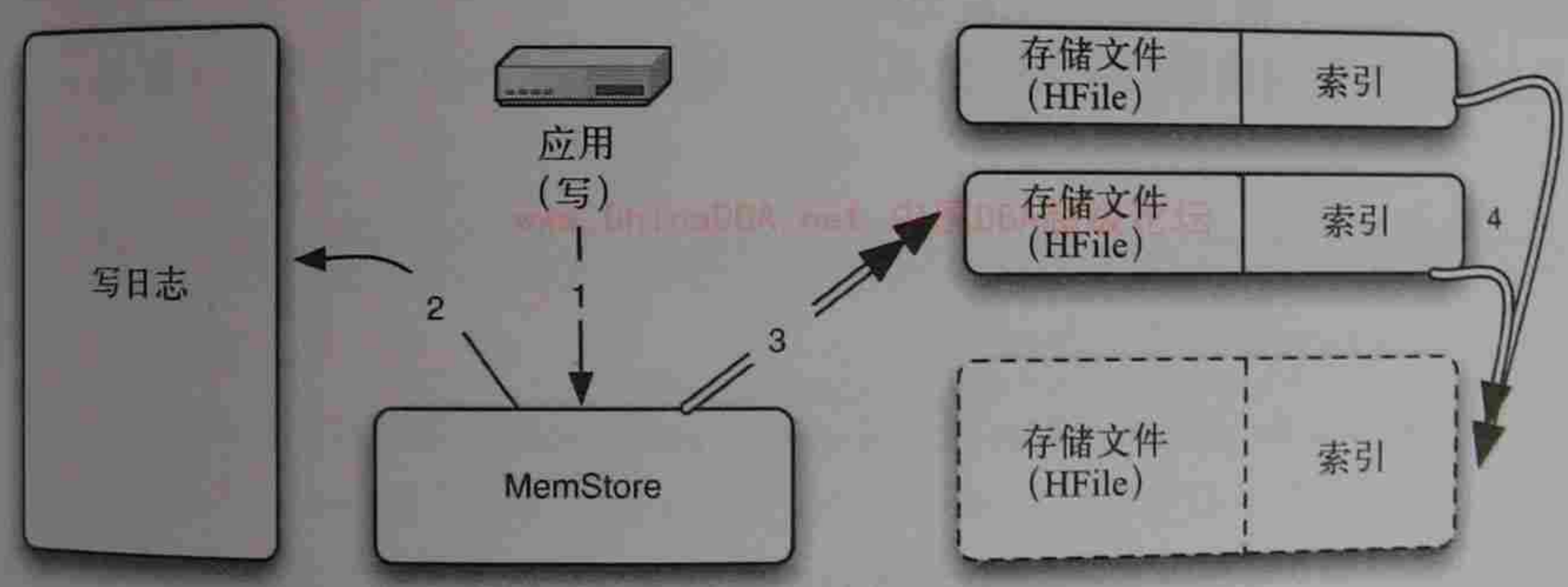


图12-4 HBase写操作

关于获得、运行和增强HBase的更多信息，请访问项目站点[②]。

2. 在StumbleUpon中使用HBase

StumbleUpon通过认真考虑，从大量候选数据库和类数据库的存储和检索系统中挑中了HBase。我们考量完全一致性，即任何在一个写操作之后的查询都能确保得到写操作的结果。此外，StumbleUpon致力于开源模式，我们完全免费地回馈到社区中，而HBase强大的开发社区不仅印证了这一承诺，而且提供了一个改进产品的宝贵资源。

我们第一个大型的HBase测试是从我们基于MySQL的系统中导入现有的合法数据。过去，我们仅在绝对必要的时候才完成这个过程（如迁移表或主机），需要花费几天或几周的时间才能完成。

① Thrift是由Facebook开发的一个远程过程调用库，现在为Apache孵化器，见http://incubator.apache.org/thrift/。

② http://hadoop.apache.org/hbase/。

www.ChinaDBA.net 中国DBA超级论坛

3. 行存储与列存储

我们看一个列存储设计模式的例子，它可以为包含多个逻辑属性组的用户存储任意的属性值。在这个例子中，我们假设用户具有如下属性。

- 联系方式——Email和Web地址、即时消息账号、个人资料照片URL。
- 统计信息——“注册日期”、“上次登录时间”、“上次客户端版本”。
- 属性——用于远端登录证书，向第三方服务进行认证。
- 权限——用于访问站点属性与数据。

在传统的RDBMS领域，我们可以任意地将每个组赋给一个表。可以提取用户属性并同时将之关联到外键与连接操作。如果设计足够认真[1]且数据规模适当，这样的系统是灵活的、可维护的。但是，当访问模式改变时（例如，我们希望为每个用户存储多个证书，而开始时假设只有一个），就会发现设计很难被改变。

此外，当数据量超过一定规模以及schema需要进行重构时，这种设计就会遭遇致命的缺陷。对包含数百万或数十亿行的生产型数据库表进行ALTER TABLE操作，或对系统schema的更改进行正确性和完整性检查的想法都是非常糟糕的。即使有一个完美的、静态的、具体的表，数据分析依然会由于记录选择、输入和输出而成为瓶颈。

让我们看一下代码清单12-1中的简单示例，每个Stumble中的用户通常仅有一个ID和一个记录。

代码清单12-1　逐个用户、逐个URL地确定Stumble

```
public class CountUserUrlStumbles {
  public static class Map extends MapReduceBase
      implements Mapper<ImmutableBytesWritable, RowResult,
      Text, Text> {
    @Override
    public void map(ImmutableBytesWritable key,
                    RowResult value,
                    OutputCollector<Text, Text> output,
                    Reporter reporter) throws IOException {
      byte [] row = value.getRow();
      int userid = StumbleUtils.UserIndex.getUserId(row);
      int urlid = StumbleUtils.UserIndex.getUrlId(row);

      Text one = new Text("1");
      output.collect(new Text("U:" + Integer.toString(userid)), one);
      output.collect(new Text("Url:" + Integer.toString(urlid)), one);
    }
  }

  public static class Reduce extends MapReduceBase
      implements Reducer<Text,Text,Text,Text> {
    @Override
    public void reduce(Text key,
                       Iterator<Text> values,
```

① 第一次很少奏效，因为最后的schema很难被预知！

www.ChinaDBA.net 中国DBA超级论坛

```
                    OutputCollector<Text, Text> output,
                    Reporter reporter) throws IOException {
    int count = 0;
    while (values.hasNext()) {
      values.next();
      count++;
    }
    output.collect(key, new Text(Integer.toString(count)));
  }
}

public static void main(String []args) throws IOException {
  if (args.length < 2) {
    System.out.println("Give the name of the by-userid stumble table");
    return;
  }
  JobConf job = new JobConf(CountUserUrlStumbles.class);
  job.setInputFormat(TableInputFormat.class);
  FileInputFormat.setInputPaths(job, args[0]);
  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);
  job.setOutputFormat(TextOutputFormat.class);
  TextOutputFormat.setOutputPath(job, new Path(args[1]));
  job.setNumMapTasks(5000);
  JobClient jc = new JobClient(job);
  jc.submitJob(job);
 }
}
```

在这个例子中，我们看到例程的StumbleUpon任务：统计每个用户以及每个URL的stumble。虽然这项任务不是特别复杂或深奥，我们在这里为读者提供的是一个具体实例，是以我们每天执行的分析任务为基础的。最有趣的是这个小的示例在处理一个键的数百亿个计数时，需要大约1个小时才能完成（使用20个商用节点）。而基于MySQL的对应实现无法在合理的时间内完成——要想完成的话，至少不能缺少对从MySQL中转储数据、将行拆分为合理的块大小再做结果合并的特殊处理和支持。

你可能会发现这一系列操作很熟悉：先进行map操作，然后进行reduce操作！通过使用HBase和Hadoop这些通用平台，我们就能够结合需求进行类似的统计调查，不需要做特别的准备和运行时处理工作。为了把这个直观的例子应用到真实的世界，我们现在可以在同一天完成所有的分析任务。我们可以提供快速实现即席查询的能力，在Hadoop和HBase支撑我们的平台之前，这种速度是不可想象的。因为商业的成败取决于数据的分析，从前台的数字处理到后台研究人员对内容的即时垃圾邮件分析，周转时间的减少都会产生令人难以置信的影响。

如果没有分布式处理平台能够支撑提取、转换和分析，当数据schema比这个示例更为复杂时，可以想象重构自定义处理管道的难度。

4. 超越单机

如前所述，可扩展性是HBase最重要的一个特性，它能够（最终）超越单机的写限制。

www.ChinaDBA.net 中国DBA超级论坛

通常情况下，扩展数据库涉及增加读操作的从节点以及系统的缓存。只有当你的应用程序读多写少时，增加读操作的从节点才有作用。如果你的数据集更改并不频繁，缓存才有作用。即便如此，这些系统结构的特征也总是会在应用层增加巨大的复杂性。

HBase驻留在集群上任何一个机器的每个区域上（每个都是区域服务器）。写操作涉及托管该区域的区域服务器，而HBase的区域服务器（默认情况下）写入3个HDFS数据节点[①]。基于一个大表和一个同样大的集群，写操作被分散到很多不同的机器上，从根本上避免了主/从数据存储所具有的单机写瓶颈问题。

这个特征可以帮助你使用传统关系数据库管理系统成本的很小一部分来获得扩展。随着大型硬件系统与其所提供的实际性能相比越来越昂贵，这是一个影响相当深远而重要的能力。对于在StumbleUpon的大型工作负载，单从字面上就可能节省数百万美元。还有一些问题在单机系统中根本无法得到解决！

对于高度动态的数据集，我们经常读取刚刚写入的内容，这样系统中的缓存，如memcached，可能无法提供很多帮助。HBase在写缓冲区中保存最近写入的数据。读取的数据直接来自内存。此操作可以完全避免使用缓存层。

高度动态的数据集的一个例子是事件计数器。这是一个困难的问题，因为大多数高速解决方案往往是只有利用内存才能满足性能（比如memcached），但又无法满足持久性。考虑HBase及其`incrementColumnValue()`调用。通过在磁盘中记录日志并缓冲到写缓冲区，读取可以直接来自写缓冲区，达到高性能和高持久性。StumbleUpon利用HBase的能力来对网站的每个事件进行统计——单击、点击率、广告送达等。

此外，HBase为典型的分区方案提供了绝佳的选择。大多数传统的分区方法需要对键空间的先验假设。当散列函数分布不均匀时，或键的分布违背了分区的假设时，就会对性能造成严重的影响。

HBase根据数据大小来将表分割到各个区域。当区域中的数据增长并达到配置大小时（当前默认为256 MB），就会从数据中取一个中间值，将区域分割为大致相同的两块。每个块成为一个新的区域，可以进一步添加内容。重复此过程数千次会让一个表被分裂成大约2000个大小相等的区域。图12-5显示了在一个简化的HBase集群上同时执行3次写操作和一次读操作的情况，该空间中包含256个键（0x00至0xFF）。

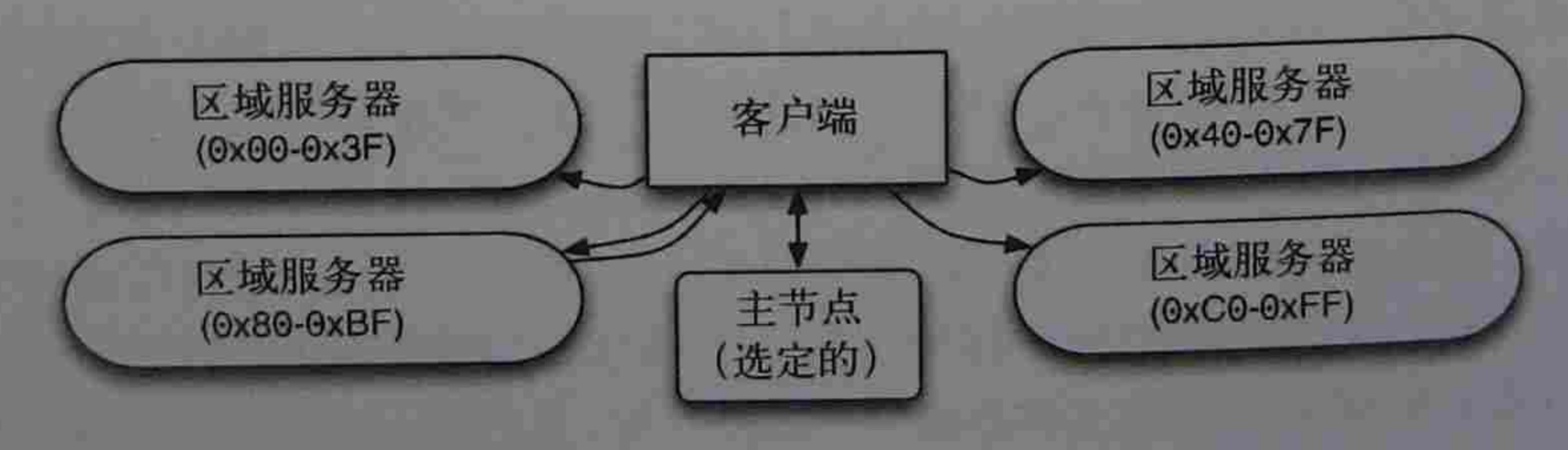


图12-5　HBase可方便地自动“分区”

www.ChinaDBA.net 中国DBA超级论坛

① HDFS在多个数据节点写入数据以获得数据持久性，以及基于局部的性能改进。

- 客户端从ZooKeeper中导出ROOT表位置。
- ROOT表的指针指向包含用户表位置的META表。
- 客户端为所选操作寻找区域。位置缓存在客户端中。
- 请求被发送到该区域中去执行。

随着StumbleUpon的数据继续不规则地增长，我们以后不再需要手动地调整不平衡的分片或区域，而这在包含RDBMS的大多数手工分区解决方案中则是经常遇到的棘手问题。

5. HBase的惊人速度

谈了许多HBase的好处，不过在初期，我们发现系统性能并不能满足在线的数据服务。为了解决这个问题，Ryan为HBase项目增强性能和可靠性作出了许多贡献。

第一个主要贡献是HFile格式。以前的格式在索引策略、读取路径和内部API上效率低下。发现的几个问题如下所示。

- 面向流的数据格式不易缓存。
- 索引效率与数据大小紧密相关。
- 要复制大量的字节数组。
- 对象创建的速度太高。

HFile是一个不变的文件格式。一旦写入，其中的值就不能再更改。在代码中reader和writer是分开的，没有改变或者更新的方法。由于大多数HFile放在HDFS中，无论如何修改都不可能，因为 HDFS文件也是不可变的。

HFile的writer有一个直接的写路径，具有4个要素。

- 打开文件，提供压缩、块大小和比较参数，这些参数在文件的生命周期内不会改变。
- 追加键，按比较参数排序，任何不按序增加键的操作将导致异常。
- 选择性地追加元数据块，用于附加的数据或特征，如Bloom过滤器。
- 关闭文件，完成索引并写入文件尾。

当键和值被追加到HFile中时，代码将跟踪当前块的大小。一旦它超出指定的块大小，当前块就会被刷新，数据就开始被压缩到下一个块中。当HFile的writer追加一个块时，就在内存中形成了对每个块第一个键的索引及其在文件中的偏移量。当调用close方法时，块索引立即被写到最后一个块之后。可选的元数据块紧跟着元数据块的索引进行追加。最后，追加索引指针结尾并关闭该文件。

当打开文件进行读取时，加载数据块索引和元数据块索引。它们始终驻留，直到reader对象被回收。索引允许快速查找并读取数据块。为了在文件找到一个键，reader首先对索引进行二进制搜索。为了找到块的编号，它读入并解压缩数据块，并将其存储到数据块的缓存中。然后，代码遍历内存中的块来查找这个键，或者与之最匹配的键。接着返回指针，允许客户端看到数据的单个副本。

HFile的优势来自于其概念和实现的简单性。实现都在一个文件中（测试除外），reader和writer加起来大约有1600行代码。

www.ChinaDBA.net 中国DBA超级论坛

HFile提供了一个新的内部平台来重写剩余的区域服务器。随着时间的推移，合并读取多个文件到单个扫描结果的内部算法不断发展，并需要重新审视。Streamy.com的Jon Gray和Erik Holstad设计和实现了一个全新的读操作，添加了新的删除语义并重构了内部键的格式。通过使用更高效的算法并重新实现了零副本HFile，代码的执行速度得到了进一步增强。

令人印象深刻的是，整体速度可以获得30倍到100倍的提高，这取决于特定的API调用。在低端，扫描一系列行可以获得30倍的提速。在高端，获取单个行可以加快100倍。通过这些性能改进，HBase才真正可以被认为是“web ready”。

6. HBase与并行化

HBase展示了在读写负载上优秀的并行加速性能。由于StumbleUpon在MySQL中存储太多的数据，插入性能是非常重要的。为了将数据复制到HBase，用一个仅有mapper的Hadoop作业从MySQL中读取数据并随后写入HBase。在20个节点的集群上运行可以获得大约80倍的并行度，插入性能从每秒100 000个操作提高到每秒300 000个操作。所涉及的行大约为100字节。

与写性能同样令人印象深刻，读性能异常高效。使用具有80倍并行度的MapReduce读聚合作业，它可以让读取速度达到每秒450万行。以这个速度读取我们最大的表只需不到一个小时。对整个表的分析是以前并不存在的强大功能。

上述所有机器都是双四核Intel，带16 GB内存。每个节点有两个SATA磁盘，每个容量为1 TB 。这些相对中档的标准节点提供了极高的性能，而集群只是执行得更好、更多。

12.3.3 StumbleUpon 上的更多 Hadoop 应用

在StumbleUpon上，我们的口号是“Log early, log often, log everything”（早记、常记、全记）。任何数据片段都不会太小或太乱而在将来没用。在分布式处理这个传统领域，Hadoop在搜集并分析log-and-click（登录并单击）操作上具有明显优势。StumbleUpon利用Hadoop这个天然特性来处理各种分析任务，包括Apache日志文件收集和用户会话分析。

举一个例子，对于任何Web产品，虽然每天都要应对搜索引擎优化专家和“黑帽子”攻击者，但其基本需求还是Web浏览器的用户代理字符串加上（显示的）原始IP地址和运行环境。现在想象一下所做的工作需要涉及海量的Web 服务器前端、数以百万计的用户和上亿次的单击，会是什么样子。

Scribe[①]，一个开放的Facebook项目，是在这种场景下用于聚集实时流日志数据的平台。该服务在计算机和网络级别都具有容错能力，并能方便地集成到任何基础设施中。

StumbleUpon使用Scribe将数据直接搜集到HDFS中，再由许多系统来审查和处理。Cascading和普通的基于MapReduce分析作业的组合从vanilla统计日志（如单击计数）中提取数据，但是更为复杂的消费者将数据送入基于BerkeleyDB和TokyoCabinet的实时反馈系统。后续的系统再将此数据用于搜索索引更新和缩略图生成。图12-6解释了Hadoop相关的几个数据处理模块。

① http://github.com/facebook/scribe。

www.ChinaDBA.net 中国DBA超级论坛

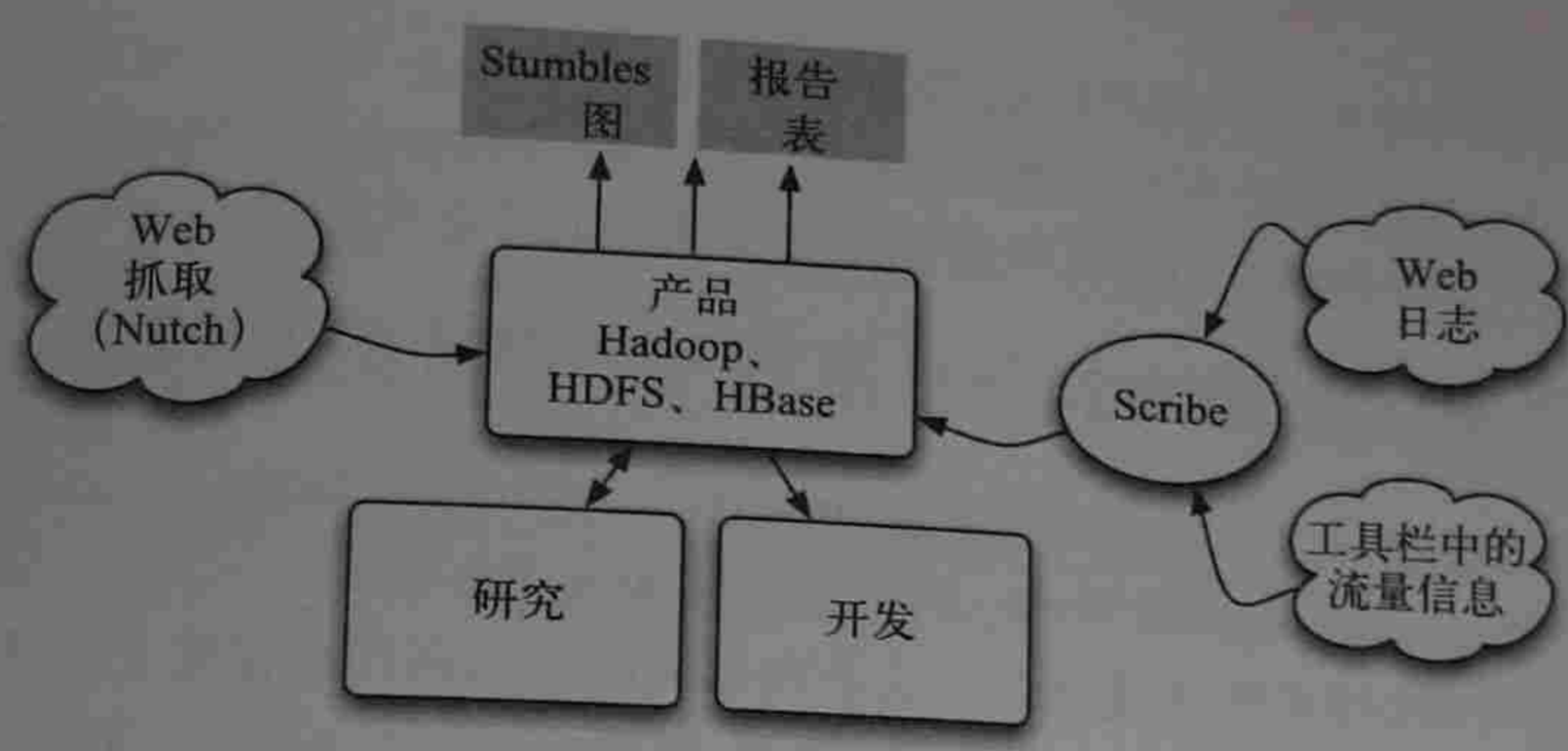


图12-6 StumbleUpon使用Hadoop进行数据搜集、分析和存储

我们利用Cascading日志分析的例子来说明对10 GB的标准Apache日志文件的处理[①]。使用Hadoop 0.19.1、Cascading 1.0.9以及前面提到的节点配置，可以获得每分钟Apache的点击数。在此示例中，点击数被分割到MapReduce作业中。我们写了一个简单的单节点基于散列的Perl程序，作为系统管理员可以创建的典型快速解决方案示例。如表12-3所示，可以确认我们的结果通过简单地在集群中增加节点，就很容易实现线性（或更高）加速比。时间取10次混合执行的平均值以消除偏差。

表12-3 使用Cascading的Apache日志处理

系　统	性能测量	结　果
1个节点	运行时间	21分46秒
	s/MB	0.127
	s/MB/节点	0.127
3个节点	运行时间	8分3秒
	s/MB	0.0471
	s/MB/节点	0.0157
15个节点	运行时间	1分30秒
	s/MB	0.00878
	s/MB/节点	0.000585
原始Perl应用	运行时间	42分49秒
	s/MB	0.251
	s/MB/节点	0.251

我们看到即使是单节点Cascading方案，也可以获得一般Perl应用的两倍性能，这是因为MapReduce框架内置的智能数据分段机制，与将所有数据映射到单个Perl散列机制不同。如果熟悉Cascading，也可以用更复杂的Perl代码优化（与维护）启动！

也就是说，StumbleUpon使用Hadoop及相关产品中的原始map（映射）和reduce（化简）功能，包括Nutch和自己编写的内容评测，以执行数据检索、分析和存储功能。将结果数据放在靠近处

① http://code.google.com/p/cascading/wiki/ApplicationCascade

www.ChinaDBA.net 中国DBA超级论坛

理流程的地方可以最大地发挥局部性的优势。

总之，StumbleUpon通过应用并扩展Hadoop、HDFS和HBase，使得MapReduce范式的优势得到最大程度的发挥。StumbleUpon很高兴能带你走向未来的分布式处理。

12.4 搭建面向企业查询的分析系统——IBM 的 ES2 项目

撰写人为Vuk Ercegovac、Rajasekak Krishnamurthy、Sriram Raghavan、Frederick Relss、Eugene Shekita、Sandeep Tata、Shivakumar Vaithyanathan与Huaiyu Zhu

相比过去几年的Web搜索的飞速进步，企业内部网的搜索在很大程度上仍是一个未被解决的难题。基于对IBM内部网的研究，Fagin等人[1]指出内部网搜索与Web搜索之间存在一些重要差异。他们观察到内部网上的查询绝大多数都是“导航式的”。正确答案只是一个很小的集合[2],[3]。例如，通过人工检查IBM内部网（至 2008年7月）最常用的6500个查询，他们发现这些查询的90%以上都是导航式的。

有几个要素是企业所特有的，它们复杂化了找到查询的“正确”答案的任务。

- 缺乏经济上的激励，（相比于在Web上进行内容发布）创作者没有动力让页面内容更易于被发现。
- 在搜索查询语句和内部网网页上使用企业特定词汇、缩写和首字母缩写词。
- 依据发起请求的人的所在地及其在组织中的角色，相同的查询有不同的“正确”答案（对于像IBM这种雇员和站点分布在80多个国家的公司而言特别重要）。

从IBM[4]先前的努力中，我们了解到使用传统信息检索技术很难克服这些问题。后来，在参考文献[5]中，我们提出了一种方法，其中包括用详尽的离线分析来预先标识导航页，并使用专用的导航索引。我们对IBM内部网上550万多页的语料库进行了试验，验证了这种方法的可行性。参考文献[5]所述的系统使用了专有平台和关系数据库的混合方式。我们还曾抓取了IBM内部网上大部分数据，共找到1亿多个URL，编制了超过1600万份的索引文档。为了应付如此大的规模，我们汲取了前人的努力成果[4],[5]，开发了ES2——一种可扩展的、高质量的IBM内部网搜索引擎。ES2以参考文献[5]中所述的分析学为基础，但它利用了大量的开放源码平台和工具，如Hadoop、Nutch、Lucene和Jaql①。

原则上，Nutch爬虫、Hadoop MapReduce框架与Lucene索引引擎提供了构建一个完全的搜索引擎所需的所有软件组件。但要真正解决前面所述的挑战，仅仅拼接这些系统是不够的。我们描述如何对被抓取的网页进行复杂的分析和挖掘，并采用专用导航索引与智能的查询处理相结合的方式，以确保有效的搜索质量。为了了解这些元素是如何一起工作的，现在我们查看ES2中的一些说明性查询及其相应的结果，如图12-7所示。

图12-7显示在ES2上查询idp的结果。IDP是Individual Development Plan（个人发展计划）的缩写，它是IBM中一个基于Web的人力资源应用程序，协助跟踪员工的职业生涯发展。由ES2返回的前两个结果表示两个不同的URL，它们都允许用户启动IDP Web应用。第三个结果实际上为组

① http://code.google.com/p/jaql/。

www.ChinaDBA.net 中国DBA超级论坛

合在一起用于描述IDP过程的一组网页，每个国家有一个入口点（通过缩进和地球图标直观地表示）。通过观察可以得到以下结论。

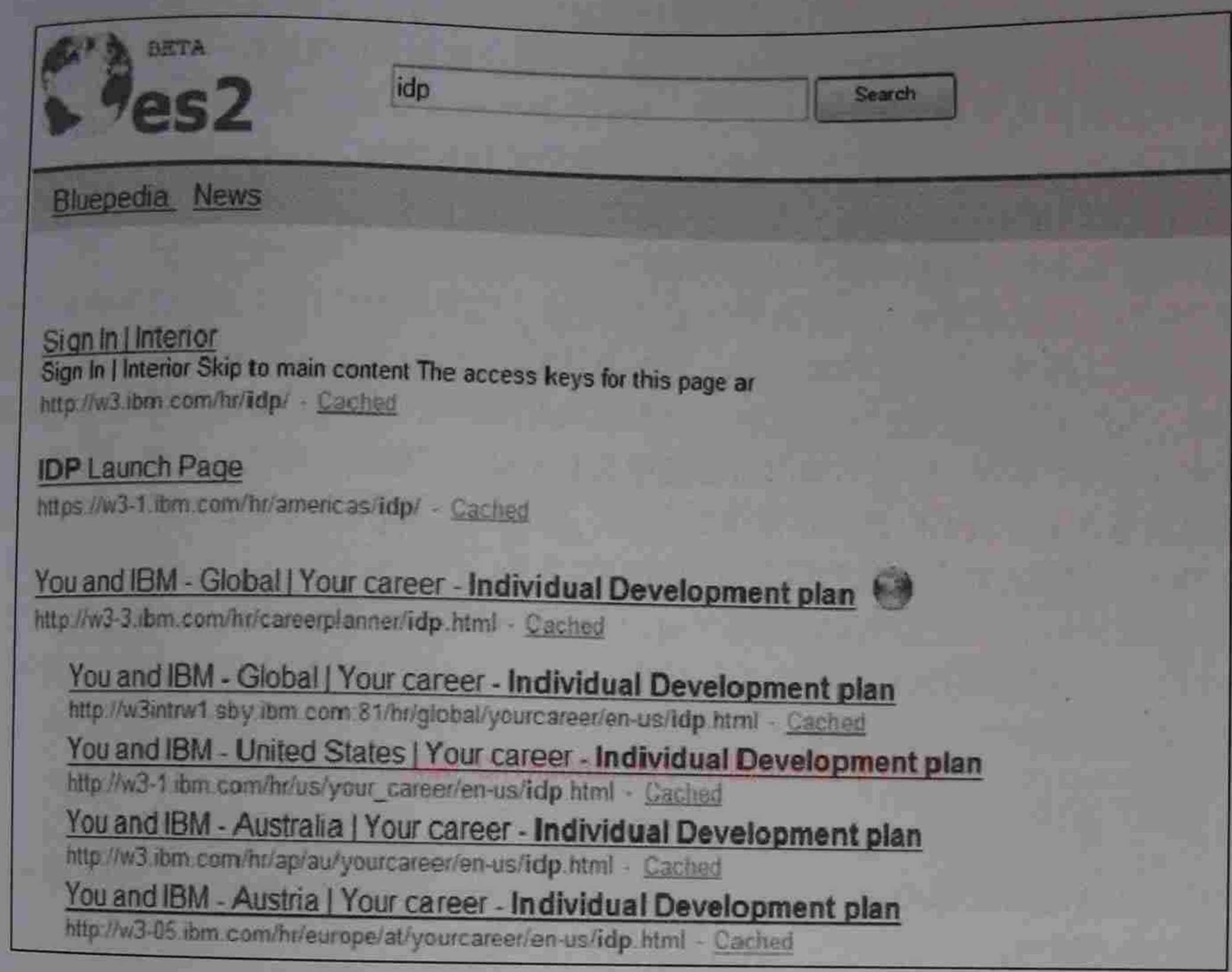


图12-7　ES2中的说明性查询结果

(1) 图12-7中的第一个结果在标题中并不包含单词idp，甚至在网页内容中也没有。ES2是通过从URL http://w3.ibm.com/hr/idp/中提取关键词idp，从而关联到这个页面。图12-7的第二个结果，除了在URL中有idp之外，我们还通过采用一个正则表达式明确地查找以“Launch Page”、“Portal”、“Main Page”等短语结尾的标题，从标题中提取出idp。在ES2中，我们使用几百个这种精心设计的模式，将其应用到Web页面中的URL、标题、META头信息以及各种其他特性，以检测和关联导航页的索引项。这种分析被称为本地分析，12.4.3节将介绍我们是如何在Hadoop集群上并行执行它的。

(2) 我们当前在IBM 内部网抓取了接近500个在URL中包含*idp*的页面，以及超过1000个在标题中包含idp或mdividual development plan的页面。要把结果缩小到图12-7中所示的两个特定的URL，我们需要使用一套复杂的分析算法，它们属于全局分析过程的一部分。12.4.3节将介绍如何使用Hadoop框架在ES2上实现这种分析。

(3) 注意到图12-7中第三项结果中所有的结果页都组织在一起，文本Individual Development Plan在标题中被突出显示，来标识它是与查询词idp相匹配的。若要完成这个匹配，在离线分析过

www.ChinaDBA.net 中国DBA超级论坛

程中还要经历以下两个步骤：①将短语Individual Development Plan提取出来，用于对标题进行本地分析；②在索引过程中，提取的短语被认为是idp的首字母缩写词的扩展，导致idp一词也被添加到索引。一般情况下，我们采用的这个处理过程被称为变体生成过程，即提取出查询词的多个变体，通过本地分析生成并添加到索引中。ES2采用的是一套变体生成策略——从简单的N-Gram断词提取到较为复杂的算法。由于篇幅关系，我们不再描述这些算法的详细信息。

(4) 最后，为了让结果可以根据查询用户的地理位置进行定制，并支持图12-7中显示的结果分组类型，我们在内部网中标记每个页面的特定地理位置（国家、地区和IBM机构）。在ES2的本地分析过程中，这种标记方法将有助于规则驱动分类器提取多种页面特性。

图12-7中的示例显示出离线分析和变体生成过程在ES2中的重要作用。ES2收集的每个页面都经历了多个逻辑工作流，每个工作流包括一个本地分析阶段、一个全局分析阶段和一个适当的变体生成策略。工作流的输出是带有一组索引词的输入页面的某个子集。依赖于特定的提取模式和变体生成规则，两个不同工作流的输出将相应地具有不同的“精度”特征。例如，如果一个工作流仔细地提取包含某个人名以及名字变体的标题，而另一个工作流仅生成页面标题所有可能的N-Gram，则前者很有可能会生成比后者更为精确的结果。

由不同精度特性的多个工作流输出所创建的索引结构只是故事的一半。为了充分利用这种索引，ES2采用了一种复杂的运行时查询处理策略。对ES2运行时组件的讨论已超出了案例研究的范围。我们将注意力集中在离线分析工作流及其在Hadoop的实现上。

www.ChinaDBA.net 中国DBA超级论坛

12.4.1 ES2 系统结构

我们假定读者对Hadoop和Nutch有所了解[①]。Nutch 是在Hadoop MapReduce平台上实现的用于Web爬行的开源爬虫。

ES2还使用了Jaql[②]，一个为JSON（一种流行的半结构化数据模型）设计的数据流语言。Jaql提供了一种类似Unix管道的语法，将半结构化JSON数据的多个处理阶段连接在一起。ES2工作流涉及多个算法，它们用于在将数据插入索引之前的本地分析、全局分析和变体生成。没有足够的数据管理支持，这个复杂的多阶段工作流很快变得无法处理。要解决此问题，ES2使用JSON来表示其数据，用Jaql指定工作流 （参见图12-8）。

图12-8显示了ES2的系统结构。ES2有6个组成部分：爬虫、本地分析、全局分析、变体生成和索引、背景挖掘和搜索运行时。ES2 使用Nutch的增强版（0.9版）——一个基于Hadoop平台的可扩展的开源爬虫。此外，ES2还从IBM的社会书签服务（称为Dogear）收集信息。与delicious.com非常类似，Dogear包含由IBM社区所收藏的各种URL，以及与每个URL相关联的标签集合。与URL关联的标签包含与页面相关的有价值的线索，而ES2使用此信息建立其索引。所有阶段都使用通用分布式文件系统HDFS用于输入和输出。本地分析处理每个页面来提取有关页面的特征，并在HDFS中将结果存储为JSON对象。ES2使用Jaql将每个页面推过剩余的管道，根据需要在每个阶

① http://nutch.apache.org/。

② http://code.google.com/p/jaql/。

段转换数据。Jaql查询被用于汇集不同的本地分析结果并调用全局分析。Jaql还被用于调用变体生成，以及使用本地和全局分析的输出对工作流进行索引。索引会定期复制到一组不同的服务于用户查询的机器。

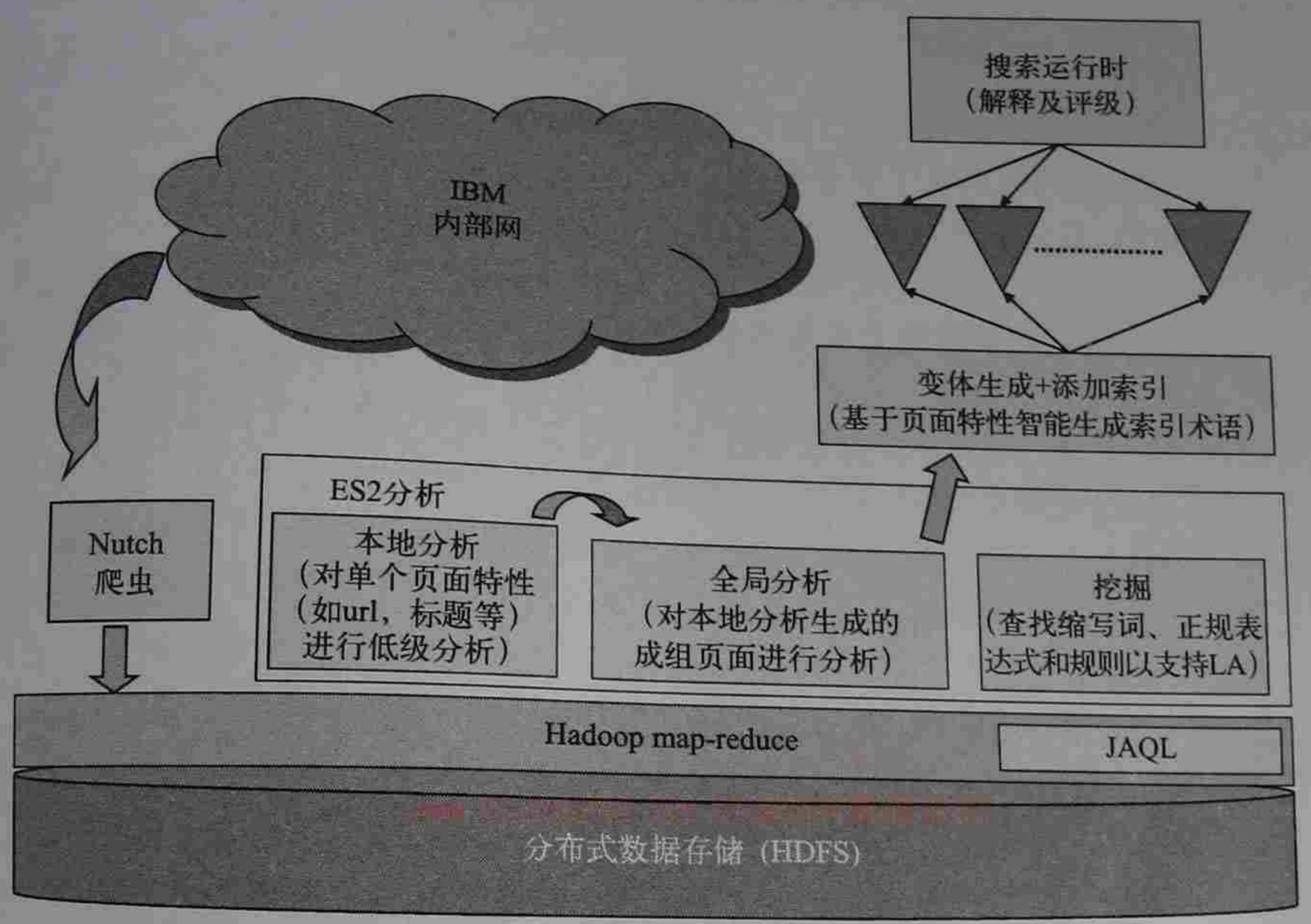


图12-8 ES2系统结构

虽然不属于主要工作流，ES2仍会定期执行几个挖掘和分类任务。此示例包含自动生成首字母缩写词库、正则表达式库[6]与地理分类规则的算法。

12.4.2 ES2 爬虫

ES2 使用Nutch 0.9。Nutch中的主要数据结构为CrawlDB：它是一个键值集合，其中键为Nutch所知的URL，而值为URL的状态。状态包含与URL有关的元数据，如发现时间、是否已经被获取等。Nutch被设计为一个包含3个MapReduce作业的序列。

- 生成——在此阶段中，通过扫描（来自CrawlDB的）输入键/值对，为已经被发现但未提取的URL生成一个提取列表。生成此列表的通用方法是基于适当的评分机制选择未提取URL中的前k个（k为Nutch中的配置参数）。
- 提取——在此阶段，在输入提取列表中与URL相关联的页被提取并解析。输出包括 URL和页面解析后的结果。
- 更新——此阶段通过解析提取阶段的页面内容来收集所有已发现的URL，并将它们合并到CrawlDB。在每个生成—提取—更新周期提取的页被称为段（segment）。

www.ChinaDBA.net 中国DBA超级论坛

我们遇到的第一个问题是抓取速度。Nutch的抓取速度每秒不到3页——远远低于集群可用的网络带宽。更为严重的问题是，在对8000万个网页采样抓取之后，所找到页面的质量出奇的低。在本节中，我们确定这两个问题的根本原因，并描述如何增强Nutch，以使它能够适应IBM内部网。

性能方面的修改

Nutch的设计目的在于做Web抓取。当把它用在抓取IBM内部网时，我们会观察到多个性能瓶颈。我们发现出现瓶颈的原因在于企业内部网包含的主机远少于Web，而Nutch中的一些设计假定有大量的独立主机。我们描述这个问题的两个表现方式以及在ES2中面向企业对Nutch设计进行调整的方法。

提取阶段主要的性能瓶颈被称为*长尾问题*（long tail problem），它显示出如下特性。在提取阶段的初期，抓取速度相对很高（通常为每秒几十个页面）。但它很快就减小到每秒不到一个页面，一直持续到该段完成。略为分析就能知道这种特性很大程度上取决于在提取列表中URL个数最大的那个主机。通过观察Nutch中两个控制提取速度的参数就能明白这一点：一个参数是Nutch在提取列表中可以同步抓取的独立主机个数，另一个参数是Nutch对同一个主机发起连续请求之前等待的时间。长尾问题的直观解决办法是限制提取列表中特定主机的URL个数。遗憾的是，这还不够，因为主机服务器可能并不相同，从不同主机获取相同数量页面所需的时间可能有很大差异。我们增加一个time-shutoff参数，在一定时间之后将fetcher终止，从而在工程上解决这个问题。虽然这会提前结束提取阶段（在该段中所获取的页面总数也较少），但是通过避免拖尾阶段，我们可以维持较高的平均抓取速度。在实践中，我们观察到通过适当地设置这个参数，可以将平均爬取速度提高接近3倍。理想情况下，当前的提取速率应该确定这个参数，遗憾的是，这需要将多个map任务的信息集中在一起，现在用Hadoop还不能轻松地做到这一点。

fetcher中的主存数据结构导致另一个性能瓶颈。fetcher工作时，首先创建一组序列，其中每个序列为特定主机存储URL——我们把这种数据结构成为FetchQueues。被分配给FetchQueues的固定数量的内存在各个队列之间共享。fetcher读取从其输入获得的URL，并将它们插入到FetchQueues中，直到它耗尽所分配的内存。分配给FetchQueues中每个队列的工作者线程同时从不同主机上读取页面，直到它们的队列为空。因为输入中的URL由主机进行排序（这是在生成阶段人为造成的），而fetcher在URL上所耗费的分配给FetchQueues的内存来自非常少的主机，这带来了瓶颈。这种设计适合在Web上抓取大量的主机，因为在提取列表中的每个主机只有几个URL。而在企业中，主机最多也不过几千个。结果是，几个工作者线程积极地从FetchQueues中获取内容，导致资源严重地浪费。通过将FetchQueues替换为总大小不受限制的基于磁盘的数据结构，我们解决了这一问题。这允许fetcher使用输入中的所有URL来填充 FetchQueues，从而保持最大的活跃工作者线程个数。这个简单的变化可以将提取速率提高几倍。

12.4.3　ES2 分析

ES2的复杂性与能力大多在于其分析。在本节中，我们简要说明不同的算法，特别注意将这些算法映射到Hadoop上时所做的设计选择。

www.ChinaDBA.net 中国DBA超级论坛

1. 本地分析

在本地分析中，每页单独进行分析以提取线索，这些线索有助于决定该页面是否为候选导航页（candidate navigational page）。在ES2中使用了5种不同的本地分析算法，即TitleHomePage、PersonalHomePage、URLHomePage、AnchorHome和NavLink。这些算法使用基于正则表达式模式、词典和信息提取工具[7]的规则来确定候选导航页。例如，使用这样的正则表达式，“\ A\ W*(.+)\s<Home>”（Java 正则表达式语法），PersonalHomePage算法可以检测出标题为“G. J. Chaitin's Home”的页面是G. J. Chaitin 的主页。该算法输出特征名称（“个人主页”），并将一个值与此特征（“G. J. Chaitin”）相关联。下一节将描述重定向对本地分析的影响，并讨论解决方案。

2. 重定向解决方案

IBM内部网的很多网站将重定向用于更新、负载平衡、升级和内部重组处理。遗憾的是，重定向会导致本地分析算法出现问题。例如，URLHomePage使用URL文本来检测候选导航页面。重定向之后，目标URL可能就不再包含与原始URL相同的特征。以网址 http://w3.can.ibm.com/hr/erbp为例。本地分析算法可以通过URL中的线索正确识别这个URL是Employee Referral Bonus Program（ERBP）的主页。但这个URL重定向到位于 http://w3-03.ibm.com/hr/hrc.nsf/3f31db8c0ff0ac90852568f7006d51ea/ac3f2f04ba60a6d585256d05004cef97?OpenDocument的Lotus Domino服务器后，Lotus Domino数据库提供Employee Referral Bonus Program的信息服务。源URL中的线索在目标中就不再可用，本地分析算法将不再将该页面识别为导航页面。要防止出现这种情况，ES2解析所有重定向，收集那些会重定向到目标页面的URL集合，并为合适的URL提供本地分析。

若要跟踪重定向，我们修改Nutch来标记每个由源URL重定向的目标页。考虑图12-9。爬虫遵循从页面A到页面B，以及从页面B到页面C的重定向。我们通过用源URL（A）来标记页面B和页面C来跟踪这些重定向。这个标记以元数据字段的形式被存储在段文件中。段文件是一个键/值集合，其中键是页的URL，值是页的内容（连同其他的元数据字段）。

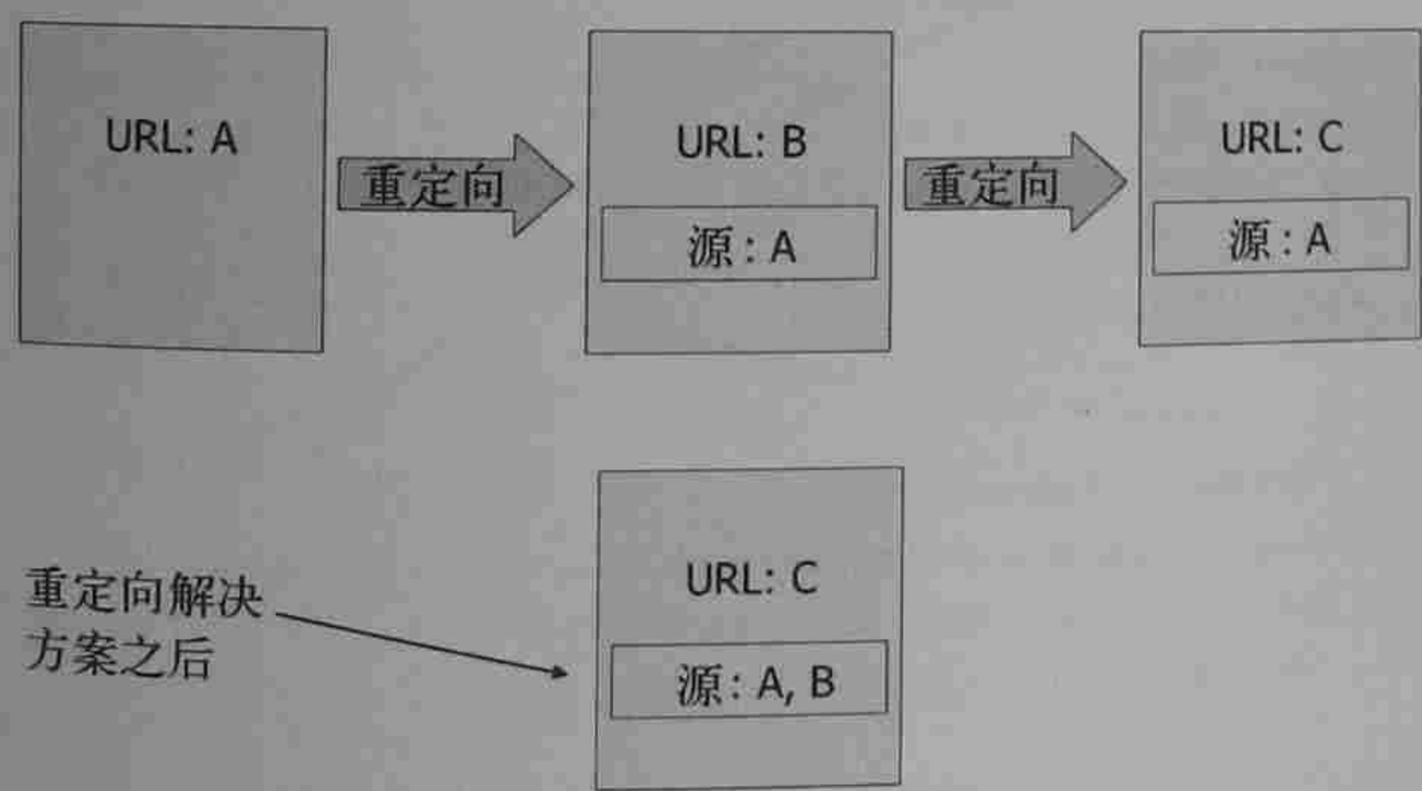


图12-9 重定向解决方案

代码清单12-2（称为ResolveSimple）概述了Map和Reduce函数，这两个函数用于解决段上的重定向和调用本地分析的功能。map阶段输出源URL和页面内容。reduce阶段将所有源URL相同的页面放到单个组中。在图12-9所示的示例中，页面A、B和C的共同源URL为A。这个组（C）中的目标页面以及其他的URL（A和B）于是也被传递到本地分析中。

www.ChinaDBA.net 中国DBA超级论坛

代码清单12-2　ResolveSimple

```
Map (Key: URL, Value: PageData)
if PageData.SourceURL exists then
   Ouput [PageData.SourceURL, PageData]
else
   Output [URL, Pagedata]
end if
End

Reduce (Key: URL, Values: Pageset)
Let URLset = Set of all URLs in Pageset
Let page = Target of redirection in Pageset
result = LocalAnalysis(page, URLset)
output [page.URL, result]
End
```

3. Hadoop实现

在ResolveSimple这个例子中，在化简程序里调用本地分析。这就要求Hadoop将页面的内容从map阶段传递到reduce阶段。这涉及在网络上对大量数据进行排序和移动。要避免出现这种情况，我们修改ResolveSimple（见代码清单12-2）并将重定向解决方案和本地分析分开，以便在map阶段运行本地分析算法。这让本地分析的计算与数据可以在相同的位置，从而使得性能得到显著的提升。

在代码清单12-3中，我们给出了改进的算法，称为Resolve2Step。在该算法的map阶段，我们仅传递元数据，而不传递（占用大部分数据量的）页面内容。在ResolveSimple的reduce阶段，我们输出一个两列的表：第一列为这组页面中目标页的URL，而第二列是当页面提交给本地分析时与之关联的一组URL。

代码清单12-3　Resolve2Step

```
1: Resolve Redirections
Map (Key: URL, Value: Page)
if PageData.SourceURL exists then
   Ouput [PageData.SourceURL, PageData.metadata]
else
   Output [URL, Pagedata.metadata]
end if
End
Reduce (Key: URL, Values: Pageset)
Let URLset = Set of all URLs in Pageset
Let page = Target of redirections in Pageset
output [page.URL, URLset]
End

2: Run Local Analysis
Map (Key: URL, Value: Page)
Load resolveTable from output of previous step if needed
Let URLSet = resolveTable[URL]
result = LocalAnalysis(page, URLset)
output [page.URL, result]
End
```

www.ChinaDBA.net 中国DBA超级论坛

如果一个URL引起重定向，我们不要在此表中为它添加一个条目。表12-4显示了此类表的一个示例。在图12-9中显示的重定向链形成表12-4中的第一行。在本地分析的后续仅有map任务的作业中，map任务将重定向表读入内存。对于典型的段而言，这个表相当小，很容易装入内存。对于输入段中的每个URL，如果mapper发现一个非空的项，就会查找这个表。它将这些URL传递给本地分析。通过在map阶段调用本地分析，Resolve2Step 避免了像ResolveSimple一样在网络上将页面内容传递给reducer。在8个节点的集群上，对大概400 000个页面的本地分析执行这两个算法。ResolveSimple花了22分钟才完成，Resolve2Step只花了7分钟。

表12-4 Resolve2Step中的Resolution表

URL	源
C	{A,B}
…	…
X	{Y}
…	…

为了了解Resolve2Step是如何扩展的，我们在同一段（400 000页）上运行此算法，并将集群大小从1个服务器改为8个服务器。时间显示在图12-10中。加速比图显示，在曲线开头，我们得到线性扩展，之后增加节点的好处就减弱了。这是因为输入仅包含一个有40万页的段。Hadoop无法在更细粒度上有效地划分该任务。我们将在下一节中看到，对于较大的输入数据集，Hadoop可以有效地划分任务并提供线性扩展。

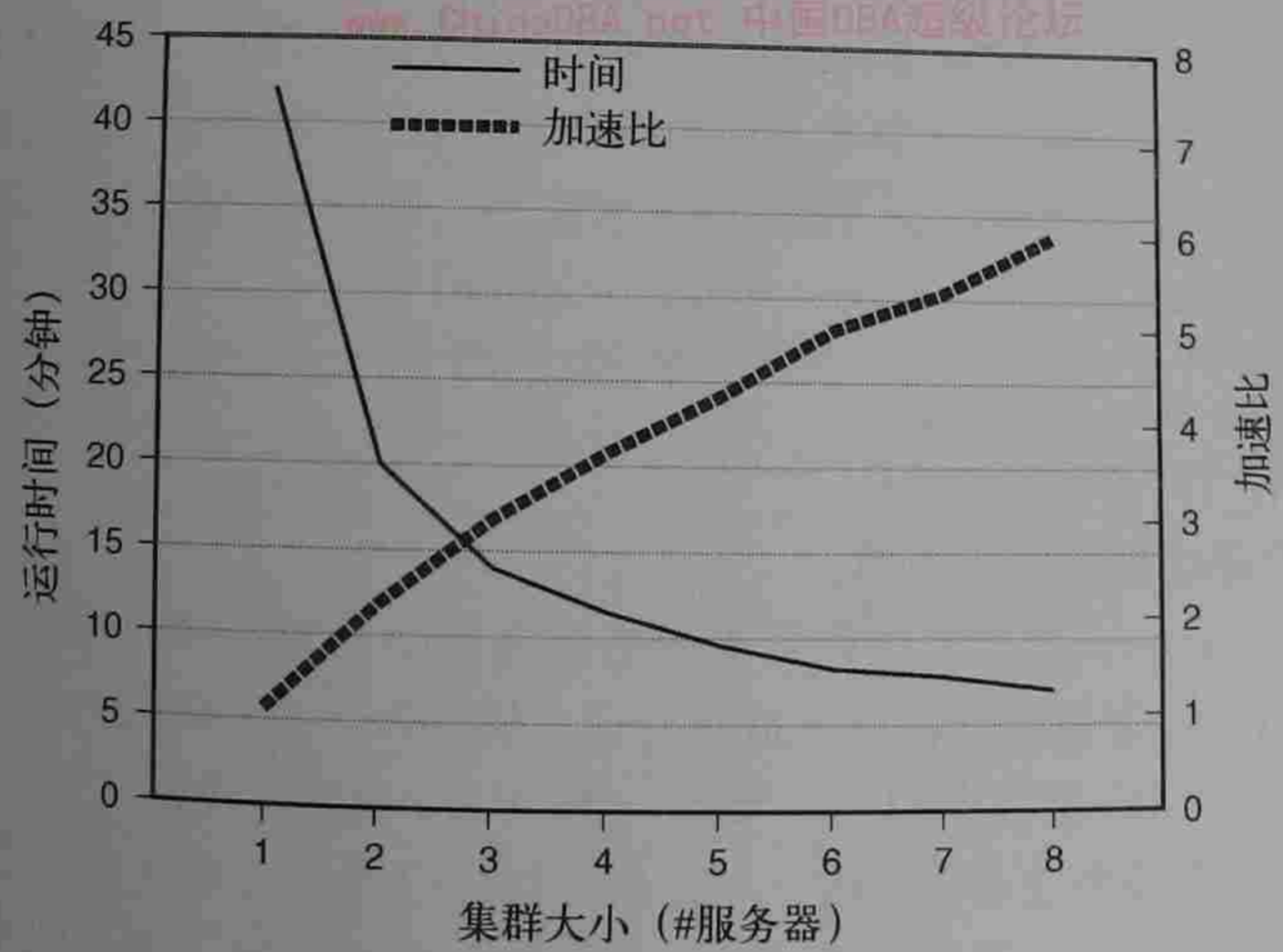


图12-10 对于一个包含40万个页面的段，随着集群规模的增加，Resolve2Step算法的性能线性增长

4. 全局分析

上一节中所述的本地分析任务通过从每个页面提取相关特征来确定候选导航页。但如12.4.1节所述，相同的导航特征可能与多个页相关联。考虑到网页作者在很多Web页面中使用了相同的

www.ChinaDBA.net 中国DBA超级论坛

标题。例如，“G. J. Chaitin home page”是G. J. Chaitin网站上很多页面的标题。在个人主页上做的本地分析会将所有这些页面都列为候选。ES2使用全局分析确定合适的页面子集作为导航页面。参考文献[5]介绍了两种算法：site root analysis和anchor text analysis。我们简要回顾这些算法，并描述如何使用Jaql在大型数据集上实现这些算法。

5. 全局分析算法

每个全局分析任务的输入为一组网页及其在本地分析中发现的对应特征。代码清单12-4显示了一个JSON记录的示例，对应于页面“G. J. Chaitin home page”。由本地分析生成的字段在名为“LA”的记录中，而全局分析的字段放在“GA”中。

代码清单12-4　全局分析和Jaql查询的JSON输出示例

```
[...,
{docid: 1879495641814943578,
 url: "http://w3.watson.ibm.com/~chaitin/index.html",
 title: "G J Chaitin Home Page",
 ...
 LA: {
    personalHomepage: {name: G J Chaitin, begin: 0, end: 11},
    geography: {countries: "USA", ...}
    ...},
 GA: {
    personalHomepageSiteRootAnalysis: {marked: true, ...},
    ...}
}, ...]
$alldocs  = file "laDocs.json";
$results  = file "phpGADocs.json";
$alldocs
-> filter not isnull($.LA.personalHomepage.name)
-> partition by $t = $.LA.personalHomepage.name
   |- SiteRootAnalysis($t, $) -|
-> write $results;
```

全局分析的两个算法具体如下所示。

- *site root analysis* ——两个算法都用于分组待选页面，并识别一组典型的页面。给定一个待选页面集合，它首先根据特征值进行分区，例如PersonalHomePage。为每个组构建一个页面集合，每个URL是其中的一个节点，将URL A和B关联为父与子。如果A是B的最长前缀 （较短的前缀有更早的祖先）。这个集合可以通过一些复杂的逻辑来消减，其中涉及来自于其他本地分析算法中的输入，其中的细节超出了案例研究的范畴。site root analysis不仅用于PersonalHomePage的输出，也用于TitleHomePage的输出（例如，标题为“Working at Almaden Research Center”或“IT Help Central”的页面）。
- *anchor text analysis*——该算法通过为每个页面检查指向它的所有页面来搜集其锚点文本。聚集后的锚点文本被处理用于为该URL提取一组代表性的项目。该算法的更多细节见参考文献[5]。

www.ChinaDBA.net 中国DBA超级论坛

6. Hadoop实现

首先，在全局分析中，合并步骤将对从Dogear收集的URL的主要爬虫和标签进行本地分析的结果归并在一起。紧跟着在去重步骤消除重复的网页。然后每个全局分析任务都会做一些标准数据操作（例如，分区、过滤及连接）以及一些与任务特定的用户自定义函数，如URL群的生成和裁剪。Jaql用于从宏观上指定这些任务，并使用Hadoop并行执行它们。

考虑在代码清单12-4中Jaql查询用于对PersonalHomePage数据进行全局分析。前两行指定了输入文件和输出文件。输入假定为一个JSON数组——这里是数组的记录——每个记录代表一个页面和与之关联的本地分析结果。第三行是Jaql管道的开始：页的流向从输入文件开始，由`$allDocs`指代，到后面的运算符。管道运算符之间的连接由`->`表示。在输入之后，“filter”运算符会在其谓词计算结果为`true`时生成一个值。在该示例中，只有那些有本地分析（LA）字段、PersonalHomePage字段和非空名称的页面会输出到下一个运算符。`$`是指管道中代表当前值的变量。过滤后的页面根据名字来分区。对于每个分区，使用用户定义函数`SiteRootAnalysis`进行求值。该函数将分区字段*$t*（*name*的变量）以及在分区（$）中的所有页面作为输入。最后，标注的页面被写入输出文件*$results*中。

Jaql查询结果显示在代码清单12-4中，这个结果通过转换为MapReduce作业并将作业提交到Hadoop中计算后得到。在此示例中，map阶段过滤页面并提取分区键。reduce阶段对每个分区用`SiteRootAnalysis`函数进行计算，并将输出结果写入一个文件。一般情况下，Jaql自动将一个管道定义的集合转换为一个MapReduce作业的有向无环图。

在图12-11和图12-12中，我们展示了使用Jaql和Hadoop对1600万个文档集合进行全局分析时的扩展性。图12-11详细显示了每个阶段的运行时间，涉及从本地分析到全局分析的整个过程。图12-12显示随着服务器被添加到集群中，合并、去重和全局分析的处理性能获得了成比例的提高。

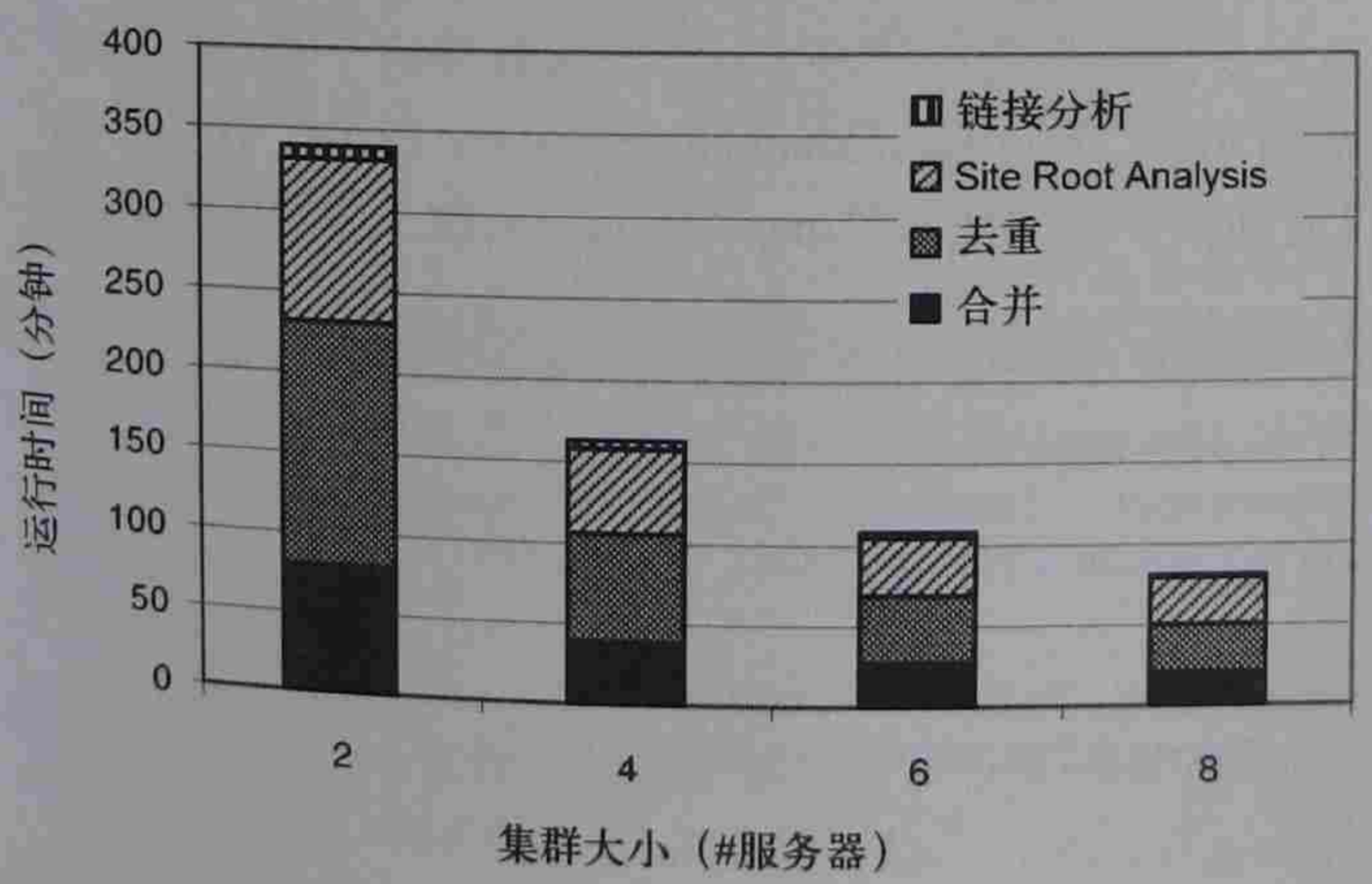


图12-11 全局分析时间

www.ChinaDBA.net 中国DBA超级论坛

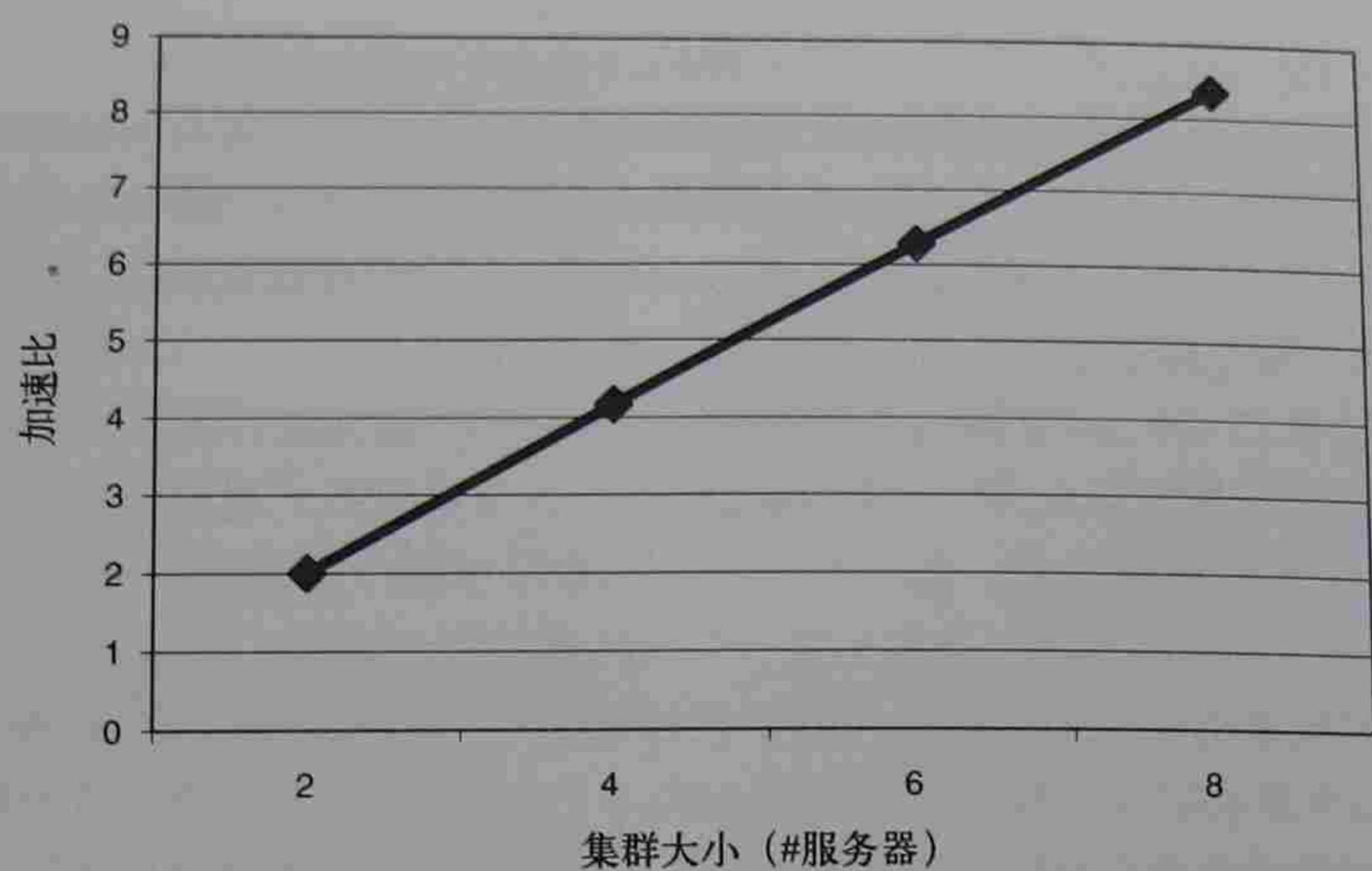


图12-12　全局分析加速比

7. 挖掘任务

ES2在后台挖掘任务中自动使用抓取的数据生成首字母缩写词库、正则表达式模式和地理分类规则。我们知道本地分析算法使用了这些资源。当更新这些资源之后，本地分析会周期地在所有页面上重新运行。我们用一个例子来简要说明ES2下所使用的首字母缩写词挖掘算法和地理分类算法。

首字母缩写词挖掘是得益于Hadoop并行实现的计算密集型任务。ES2中所使用的算法是改编自参考文献[8]。它的工作原理是检查每个页面，首先要确定形式为“longForm(shortForm)”或“shortForm(longForm)”的文本。在确定首字母缩写词和它们的候选longForm之后，map函数输出[shortForm,longForm]作为一个键/值对。reduce函数将为给定的shortForm搜索所有可能的longForm，并在输出之前按频率进行排序。reduce函数将近乎相同的longForm合并在一起，如“Individual Development Plan”和“Individual Development Plans”作为“idp”的longForm。你在代码清单12-5中可以看到用于挖掘任务的map和reduce函数。这项任务很容易在集群上并行实现。

代码清单12-5　缩写词挖掘

```
Map (Key: URL, Value: PageText)
Identify all (shortForm, longForm) pairs in the text
For each instance, output [shortForm, longForm]
End

Reduce (Key: shortForm, Values: longForms)
Canonicalize longForms that differ slightly
Compute frequency of each longForm
Output longForms in sorted order
End
```

图12-13显示了当节点数从2增加到8时，这个算法在1000万份文档的样本的运行时间。可以看出，即使在2个节点时，总体任务也可以在25分钟内完成。但我们看到该任务不会随集群规模

www.ChinaDBA.net 中国DBA超级论坛

线性地扩展。我们猜测这是因为输入的数据分散为几段，而Hadoop会将这个作业在map阶段拆分成许多任务，进而引起一个较大而且固定的开销，它与集群的大小无关。我们正在思考用一些方法来克服这些性能问题。

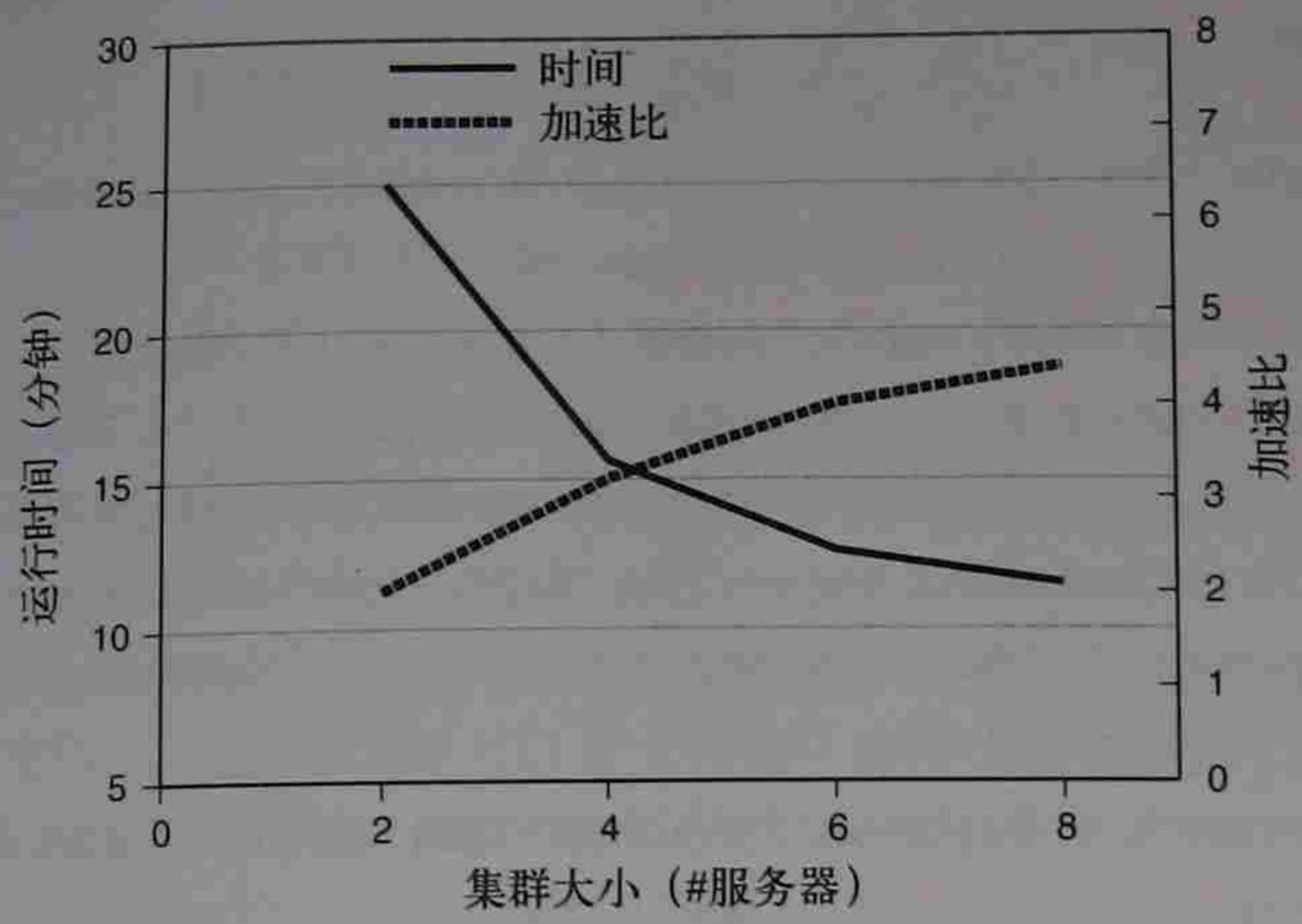


图12-13　缩写词挖掘可随集群规模获得线性加速

地理分类任务的目标是将内部网上的每个页面用与该页面最相关的国家、地区和IBM机构进行标记。许多因素导致这项任务特别烦琐。例如，许多IBM内部的新业务流程和Web应用是先在美国部署，然后再扩展到世界各地的其他国家和地区。负责后续部署中内容开发的网站管理员通常将美国的页面作为原始版本，将其编辑为相应国家的页面。但在做这种定制化的时候，很多管理员并没有相应地修改传达地域、语言和其他特定地理信息的HTML meta header[①]。因此直接从meta header中提取信息的简单分类器很容易出错。在ES2中，我们目前采用一种复杂的规则驱动的分类器，其中包含一套手工创建的强大的规则，它们建立在页面特征的一个小集合上（例如，标题中出现的国家名、URL中的国家代码等）。虽然规则已经通过手工调整来获得高精度，但效果——分类器能够指定非空地理标签的页面的数量——依然非常有限。改进需要比当前分类器使用更多的页面中的特征。但手工地为更多的特征建立准确的规则集合是非常辛苦的。我们正在开发一种可扩展的挖掘算法，自动从这些新特征中“诱导出”附加的分类规则，假定现在已经可以获得高质量的规则集合。因为每个页面都有几百个特征，使用Hadoop这类平台对于扩展我们的挖掘算法到上百万个页面上是非常关键的。

12.4.4　小结

我们描述了ES2的体系结构——它是一种可扩展的企业搜索系统，由IBM使用开源组件开发，如Nutch、Hadoop、Lucene和Jaql。我们还针对企业信息抓取，概述了Nutch所需作的改变。我们将参考文献[5]中的本地分析算法和全局分析算法映射到Hadoop上。在实现一个包括抓取、本地

① 注意meta header是给浏览器和爬虫准备的，当页面被展示时是看不见的。

www.ChinaDBA.net 中国DBA超级论坛

分析、全局分析和索引的复杂工作流时，我们发现JSON是一种有效的数据格式，而Jaql是一个非常强大的工具。总之，我们相信Hadoop、Nutch、Lucene和Jaql组成了一套强大的工具集，使用它们可以构建出像EC2这样复杂、可扩展的系统。

12.4.5 参考文献

[1] Fagin, R., R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. "Searching the workplace web." In *WWW*, pages 366–375, 2003.

[2] Broder, A. "A taxonomy of web search." *SIGIR Forum*, 36(2):3–10, 2002.

[3] Hawking, D. "Challenges in enterprise search." In *ADC*, pages 15–24, 2004.

[4] Fontura, M., E. J. Shekita, J. Y. Zien, S. Rajagopalan, and A. Neumann. "High performance index build algorithms for intranet search engines." In *VLDB*, pages 1158–1169, 2004.

[5] Zhu, H., S. Raghavan, S. Vaithyanathan, and A. Löser. "Navigating the intranet with high precision." In *WWW*, pages 491–500, 2007.

[6] Li, Y., R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. "Regular expression learning for information extraction." In *EMNLP*, 2008.

[7] Reiss, F., S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. "An algebraic approach to rule-based information extraction." In *ICDE*, pages 933–942, 2008.

[8] Schwartz, A. S., and M. A. Hearst. "A simple algorithm for identifying abbreviation definitions in biomedical text." In *Pacific Symposium on Biocomputing*, pages 451–462, 2003.

www.ChinaDBA.net 中国DBA超级论坛

附录 A

HDFS文件命令

A

本附录列出了用于管理文件的HDFS命令。它们的形式为hadoop fs -*cmd* <args>，其中cmd为特定的文件命令，<args>是一个数目可变的参数。

表A-1为命令的用法。方括号（[]）中的参数是可选的，省略号（...）意味着这些可选参数是可重复的。FILE为文件名，而PATH既可以是文件名，也可以是目录名。SRC和DST分别特指源和目的路径名。LOCALSRC和LOCALDST进一步要求位于本地文件系统之上。

表A-1　命令及其用法

命　令	用法与描述
cat	hadoop fs -cat FILE [FILE ...] 显示文件内容。若要读取压缩文件，应该使用text命令
chgrp	hadoop fs -chgrp [-R] GROUP PATH [PATH ...] 变更文件和目录的群组。选项-R递归地执行变更。用户必须为文件的所有者或超级用户。关于HDFS文件权限系统的更多背景信息，请参见8.3节
chmod	hadoop fs -chmod [-R] MODE[,MODE ...] PATH [PATH ...] 变更文件和目录的访问权限。类似于Unix的对应命令，MODE可以为一个3位八进制数，或{augo}+/-{rwxX}。选项-R递归地执行变更。用户必须为文件的所有者或超级用户。关于HDFS文件权限系统的更多背景信息，请参见8.3节
chown	hadoop fs -chown [-R] [OWNER][:[GROUP]] PATH [PATH...] 变更文件和目录的所有者。选项-R递归地执行变更。用户必须为超级用户。关于HDFS文件权限系统的更多背景信息，请参见8.3节
copyFromLocal	hadoop fs -copyFromLocal LOCALSRC [LOCALSRC ...] DST 等同于put（从本地文件系统中复制文件）
copyToLocal	hadoop fs -copyToLocal [-ignorecrc] [-crc] SRC [SRC ...] LOCALDST 等同于get（将文件复制到本地文件系统中）
count	hadoop fs -count [-q] PATH [PATH ...] 显示由PATH确定的子目录个数、文件个数、使用字节个数，以及所有的文件/目录名。选项-q显示额度信息
cp	hadoop fs -cp SRC [SRC ...] DST 将文件从源复制到目的。如果指定了多个源文件，目的端必须为一个目录
du	hadoop fs -du PATH [PATH ...] 显示文件大小。如果PATH是一个目录，会显示该目录中每个文件的大小。文件名用完整的URI协议前缀表示。请注意虽然du反映了磁盘使用情况，但不能望文生义，因为真实的磁盘使用情况依赖于块大小和副本系数

www.ChinaDBA.net 中国DBA超级论坛

（续）

命 令	用法与描述
dus	hadoop fs -dus PATH [PATH ...] 类似于du，但是当作用于目录时，dus会显示文件大小之和，而非逐个显示每个文件的大小
expunge	hadoop fs -expunge 清空回收站。如果打开回收站属性，当文件被删除时，它首先会移动到临时目录.Trash/中。只有超过用户设置的延迟之后，文件才会被从.Trash/目录中永久删除。而expunge命令强制删除.Trash/目录中的所有文件。请注意只要文件仍在.Trash/目录中，它就可以被恢复到原始目录中
get	hadoop fs -get [-ignorecrc] [-crc] SRC [SRC ...] LOCALDST 将文件复制到本地文件系统。如果指定了多个源文件，本地目的端必须为一个目录。如果LOCALDST被置为-，文件被复制到stdout HDFS计算每个文件中每个块的校验。一个文件的校验被独立存储在一个隐藏文件中。如果从HDFS读取一个文件，在这个隐藏文件中的校验被用于验证文件的完整性。对于get命令，选项-crc会复制这个校验文件。选项-ignorecrc则会在复制时跳过校验检查
getmerge	hadoop fs -getmerge SRC [SRC ...] LOCALDST [addnl] 获取由SRC指定的所有文件，将它们合并为单个文件，并写入到本地文件系统中的LOCALDST。选项adnl将在每个文件的结尾加入一个换行符
help	hadoop fs -help [CMD] 显示命令CMD的用法信息。如果不输入CMD，它会显示所有命令的用法信息
ls	hadoop fs -ls PATH [PATH ...] 列出文件和目录。每个入口点会显示文件名、权限、所有者、组、大小和修改时间。文件入口点还会显示它们的副本系数
lsr	hadoop fs -lsr PATH [PATH ...] ls的递归版本
mkdir	hadoop fs -mkdir PATH [PATH ...] 创建目录。会创建路径中所有缺失的父目录（类似Unix的mkdir -p）
moveFromLocal	hadoop fs -moveFromLocal LOCALSRC [LOCALSRC ...] DST 类似于put，只是本地的源在成功复制到HDFS上之后会被删除
moveToLocal	hadoop fs -moveToLocal [-crc] SRC [SRC ...] LOCALDST 显示一条“not implemented yet”消息
mv	hadoop fs -mv SRC [SRC ...] DST 将文件从源移动到目的。如果指定多个源文件，目的端必须为一个目录。不允许跨文件系统的移动
put	hadoop fs -put LOCALSRC [LOCALSRC ...] DST 从本地文件系统中复制文件或目录到目标文件系统。如果LOCALSRC被置为-，则输入为stdin且DST必须为文件
rm	hadoop fs -rm PATH [PATH ...] 删除文件和空目录
rmr	hadoop fs -rmr PATH [PATH ...] rm的递归版本
setrep	hadoop fs -setrep [-R] [-w] REP PATH [PATH ...] 改变文件的目标副本系数，放入REP中。选项-R将递归地改变PATH指定目录中所有文件的目标副本系数。副本系数需要用一定的时间才能达到目标值。选项-w将等待副本系数以与目标值相匹配

www.ChinaDBA.net 中国DBA超级论坛

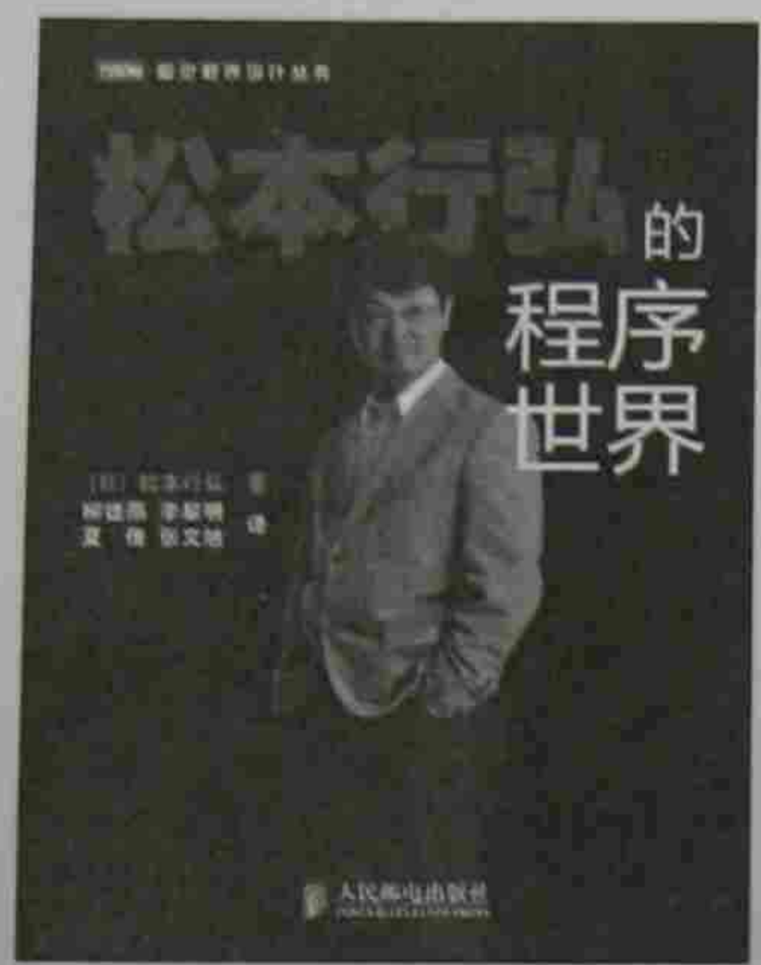
（续）

命　令	用法与描述
stat	hadoop fs -stat [FORMAT] PATH [PATH ...] 显示文件中的“统计”信息。FORMAT字符串完全被打印出来，但会按以下设定的格式进行替换 %b 以数据块为单位的文件大小 %F 根据文件类型为字符串“directory”或“regular file” %n 文件名 %o 块大小 %r 副本 %y 以yyyy-MM-dd HH:mm:ss格式显示的UTC时间 %Y 自1970年1月1日（UTC）起计的毫秒数
tail	hadoop fs -tail [-f] FILE 显示FILE中最后1KB数据
test	hadoop fs -test -[ezd] PATH 对PATH进行如下类型的检查。 -e　PATH是否存在。如果PATH存在，返回0。 -z 文件是否为空。如果长度为0，返回0。 -d 是否为目录。如果PATH为目录，返回0
text	hadoop fs -text FILE [FILE ...] 显示文件的文本内容。当文件为文本文件时，等同于cat。文件为压缩格式（gzip以及Hadoop的二进制序列文件格式）时，会先解压缩
touchz	hadoop fs -touchz FILE [FILE ...] 创建长度为0的文件。如果文件已经存在且长度非0，则报错

www.ChinaDBA.net 中国DBA超级论坛

TURING 图灵教育

图灵最新重点图书



松本行弘的程序世界
书号：978-7-115-25507-5
作者：まつもとゆきひろ（松本行弘）
译者：柳德燕 李黎明 夏倩 张文旭
定价：75.00元

“Ruby之父”松本行弘教你大师级的程序思考方法！

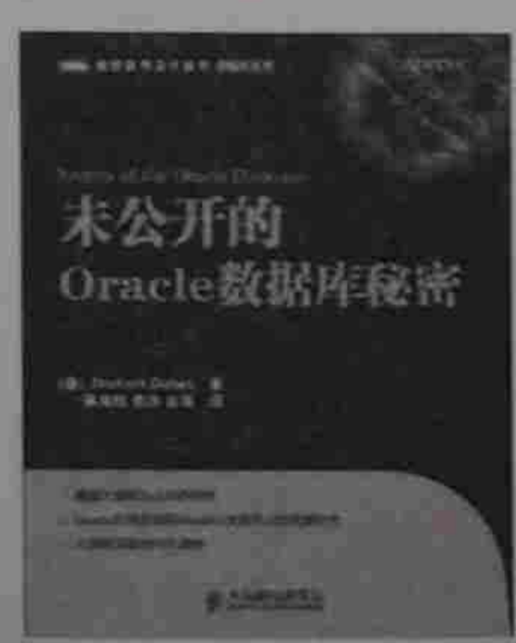
松本行弘（matz，Yukihiro Matsumoto）1965年出生。岛取县米子市出身。筑波大学第三学院资讯学系毕业。于1993年开始开发面向对象的编程语言Ruby，于1995年发表。目前为网络应用通讯研究所（NaCI）研究员，并兼任乐天技术研究所的研究员。以Matz的昵称为人所知。

本书是探索程序设计思想和方法的经典之作。作者从全局的角度，利用大量的程序示例及图表，深刻阐述了Ruby编程语言的设计理念，并以独特的视角对与编程相关的各种技术进行了考察。阅读本书不仅可以深入了解编程世界各个要素之间的关系，而且能够学到大师级的程序思考方法。

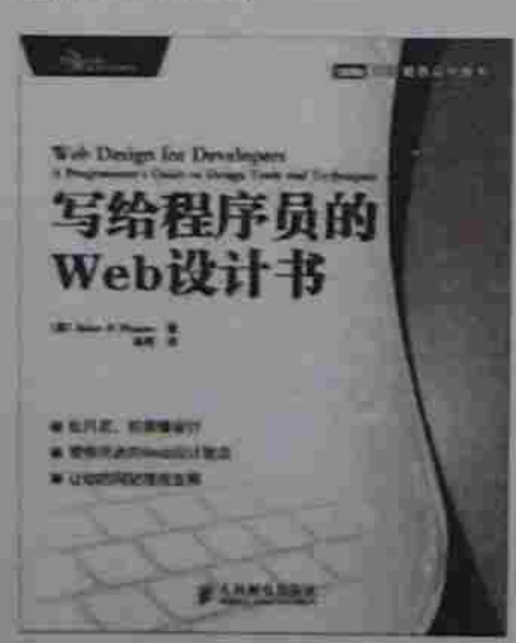
软件项目成功之道
书号：978-7-115-25965-3
定价：39.00元

软件研发之道：微软开发团队的经验法则
书号：978-7-115-25260-9
定价：45.00元

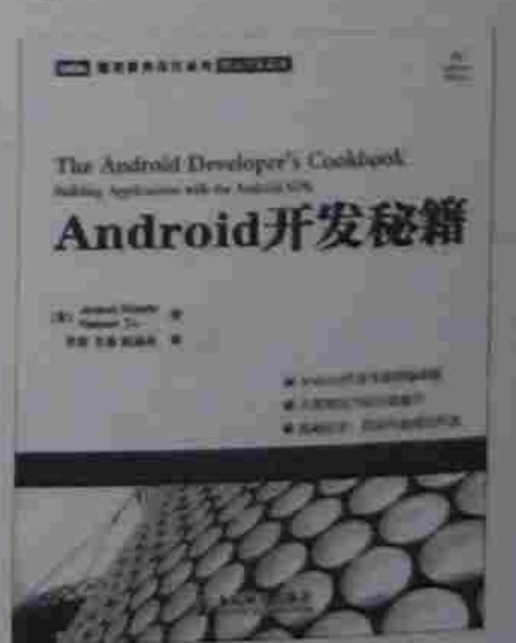
浮现式设计：专业软件开发的演进本质
书号：978-7-115-25978-3
定价：59.00元



未公开的Oracle数据库秘密
书号：978-7-115-26016-1
定价：69.00元



写给程序员的web设计书
书号：978-7-115-25911-0
定价：59.00元



Android开发秘籍
书号：978-7-115-25718-5
定价：49.00元

www.ChinaDBA.net 中国DBA超级论坛

Hadoop in Action

Hadoop实战

"本书是初学者的指路明灯，是高级用户的洞察力之源。"

——Philipp K. Janert, Principal Value公司

"为你全面阐释Hadoop的内容、成因和运行机理。"

——Paul Stusiak, Falcon 技术公司

"将Hadoop阐释清楚的最佳图书！"

——Rick Wagner, Acxiom公司

"全面覆盖Hadoop！他书无而本书有。"

——John S. Griffin, Overstock.com

"本书是对Hadoop和MapReduce的极佳介绍。"

——Kenneth DeLong, BabyCenter公司

Apache Hadoop是一个NoSQL应用程序框架，在分布式集群中运行，它适合于处理大数据集。如果需要从数据中分析信息，那么Hadoop是你的最佳选择。

www.ChinaDBA.net 中国DBA超级论坛

本书是一本深受读者好评的专著，旨在教会你如何以MapReduce方式编写程序，其中包含MapReduce编程中的最佳实践及设计模式。书中内容由浅入深，以几个简单的例子开始，继而转向Hadoop在较为复杂的数据分析中的应用。此外，还介绍了Streaming API 及Pig和Hive等工具。

MANNING

图灵社区：www.ituring.com.cn
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/数据库/Hadoop

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-26448-0

ISBN 978-7-115-26448-0

定价：59.00元

www.ChinaDBA.net 中国DBA超级论坛