

# 深入理解大数据

## 大数据处理与编程实践

UNDERSTANDING BIG DATA  
BIG DATA PROCESSING AND PROGRAMMING

主编：黄宜华（南京大学） 副主编：苗凯翔（英特尔公司）



- 学术界与业界完美结合的结晶。从原理剖析到系统化算法设计与编程实践
- 多年系统性教学实践和成果总结。一系列业界产品增强功能深度技术创新
- 一系列大数据算法、优秀课程设计以及来自科研课题及业界应用的实践案例



机械工业出版社  
China Machine Press

计算机类专业系列能力培养系列

# 深入理解大数据： 大数据处理与编程实践

Understanding Big Data  
Big Data Processing and Programming

主 编：黄宜华（南京大学）  
副主编：苗凯翔（英特尔公司）

HZ BOOKS  
华章图书



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

深入理解大数据：大数据处理与编程实践 / 黄宜华主编. —北京：机械工业出版社，2014.7  
(计算机类专业系统能力培养系列教材)

ISBN 978-7-111-47325-1

I. ①深… II. ①黄… III. ①数据管理—高等学校—教材 IV. ①TP274

中国版本图书馆 CIP 数据核字 (2014) 第 145263 号

本书从 Hadoop MapReduce 并行计算技术与系统的基本原理剖析着手，在系统介绍基本工作原理、编程模型、编程框架和接口的基础上，着重系统化地介绍 MapReduce 并行算法设计与编程技术，较为全面地介绍了基本 MapReduce 算法设计、高级 MapReduce 编程技术以及一系列较为复杂的机器学习和数据挖掘并行化算法，并引入来自 Intel Hadoop 系统的一系列增强功能以及深度技术剖析；最后，为了提高读者的算法设计与编程实战能力，本书较为详细地介绍了一系列综合性和实战性大数据处理和算法设计问题，这些问题来自课程同学参加的全国性大数据大赛中的获奖算法、课程中的优秀课程设计以及来自本团队的科研课题及业界实际的大数据应用实战案例。书中第 8 章和第 10 章的所有算法均有完整实现代码可供下载学习。

本书是国内第一本基于多年课堂教学实践总结和撰写而成的大数据处理和并行编程技术书籍，因此，本书非常适合高等院校作为 MapReduce 大数据并行处理技术课程教材使用，同时也很适合于高等院校学生作为自学 MapReduce 并行处理技术的参考书。与此同时，由于本书包含了很多来自业界实际产品的深度技术内容、并包括了丰富的算法设计和编程实战案例，因此，本书也很适合作为 IT 和其他应用行业专业技术人员进行大数据处理应用开发和编程实现时的参考手册。



## 深入理解大数据：大数据处理与编程实践

主 编：黄宜华（南京大学） 副主编：苗凯翔（英特尔公司）

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：姚 蕾

责任校对：董纪丽

印 刷：

版 次：2014 年 8 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：32.5

书 号：ISBN 978-7-111-47325-1

定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东



## 推荐序一

(中国工程院院士、中国计算机学会大数据专家委员会主任 李国杰<sup>①</sup>)

数据是与自然资源、人力资源一样重要的战略资源，掌控数据资源的能力是国家数字主权的体现。大数据研究和应用已成为产业升级与新产业崛起的重要推动力量，如果落后就意味着失守战略性新兴产业的制高点。近年来，大数据浪潮席卷全球，引起世界各国的高度关注，美国等发达国家出台了发展大数据的国家计划，全世界著名 IT 企业都在积极推动大数据技术的研发和应用，国内外很多高校和研究机构都在从事大数据技术和数据科学的研究。

学术界已总结了大数据的许多特点，包括体量巨大、速度极快、模态多样、潜在价值大等。对于处理大数据的技术人员，首先面对的困难是过去熟悉的处理系统和软件对付不了大数据，需要学会使用大数据处理和分析平台，进一步的需求是掌握大数据并行处理的算法和程序设计的方法。

Google 公司是大数据处理的先驱，其三大核心技术 MapReduce、GFS 和 BigTable 奠定了大数据分布式处理的基础。MapReduce 是一种分布式运算技术，也是简化的分布式编程模式。在 Google 公司三大核心技术基础上，Apache 社区开发的开源软件 Hadoop 是实现 MapReduce 计算模型的分布式并行编程框架。Hadoop 还提供一个分布式文件系统 (HDFS) 及分布式数据库 (HBase)，将数据部署到各个计算节点上。Hadoop 的独特之处在于它的编程模型简单，用户可以很快地编写和测试分布式系统。2008 年以来，Hadoop 逐渐被互联网企业广泛接纳，这一开源的生态环境已成为大数据处理的主流和事实标准。

一般而言，大数据处理有三种模式：离线计算、在线处理和流计算。Hadoop 是目前使用较广泛的离线计算应用框架，在线处理与流计算尚未形成广泛使用的开源生态环境。大数据处理平台还在不断发展之中，2013 年出现的 Spark 在全面兼容 Hadoop 的基础上，通过更多的利

---

① 李国杰院士，中国计算机学会大数据专家委员会主任，是我国计算机界的老一辈科学家，在并行处理、计算机体系结构、人工智能、组合优化等方面成果卓著，荣获过多项国家级奖励，领导中科院计算所和曙光公司为发展我国高性能计算机产业、研制龙芯高性能通用 CPU 芯片做出了重要贡献，对国内计算机科技、教育和产业的发展也提出过有影响的政策建议。

用内存处理大幅提高了系统性能。Spark 等新框架的出现并不是取代 Hadoop，而是扩大了大数据技术的生态环境，促使生态环境向良性化和完整化发展。

目前国内外大数据技术人才十分短缺。据麦肯锡公司预计，美国到 2018 年大数据分析技术人才缺口将达 19 万人。中国巨大的人口基数会带来更为巨量的数据，未来几年国内也将需要数十万以上的大数据技术人才。技术市场大规模的人才需求对高校大数据技术人才培养提出了很大的挑战。

作为国内最早从事大数据技术教学与研究的教师之一，南京大学黄宜华教授 2010 年就在 Google 公司资助下开设了“MapReduce 大规模数据并行处理技术”研究生课程，并组织成立了南京大学 PASA 大数据技术实验室，开展了一系列大数据技术研究工作。在多年课程教学和科研工作基础上，以理论联系实际的方式，结合学术界的教学科研成果与来自业界的系统研发经验，他组织撰写了这本专业技术教材——《深入理解大数据》，着重介绍目前主流的 Hadoop MapReduce 大数据处理与编程技术。

与市场上现有的一些大数据处理和编程书籍相比，该书有较大的特色。该书较为全面地介绍了大数据处理相关的基本概念和原理，着重介绍了 Hadoop MapReduce 大数据处理系统的组成结构、工作原理和编程模型。在此基础上，该书由浅入深、循序渐进，重点介绍和分析了基于 MapReduce 的各种大数据并行处理算法和程序设计的思想方法，并辅以经过完整实现和验证的各种算法代码的分析介绍，内容涵盖了常用的基本算法以及较为复杂的机器学习和数据挖掘算法的设计与实现。该书还通过一系列来自全国性大赛获奖算法、部分优秀课程设计、部分科研成果、以及业界实际的大数据应用编程实战案例，较为深入地阐述了相关的大数据并行处理和编程技术。

作为国内第一本经过多年课堂教学实践总结而成的大数据并行处理和编程技术书籍，该书很适合高等院校作为 MapReduce 大数据并行处理技术课程的教材，同时也很适合于作为大数据处理应用开发和编程专业技术人员的参考手册。

此外，我很高兴地看到，该书已纳入了教育部计算机类专业教学指导委员会制定的计算机类专业系统能力培养计划，作为“计算机类专业系统能力培养系列教材”。从计算技术的角度看，大数据处理是一种涉及到几乎所有计算机技术层面的综合性计算技术，涉及到计算机软硬件技术的方方面面，因此，大数据处理是一门综合性、最能体现计算机系统能力培养的课程。为此，把大数据处理纳入计算机类专业系统能力培养课程体系第三层次的核心课程，作为一门起到一定“收官”作用的综合性课程，这是在计算机系统能力培养方面的一个很好的尝试。

中国工程院院士

中国计算机学会大数据专家委员会主任

李国杰

2014 年 7 月，于北京

## 推荐序二

### 大数据处理技术——信息时代的金钥匙

可以毫不夸张地说，我们现在正处在信息爆炸的时代！随着移动互联网和物联网的迅猛发展，数据正在以前所未有的规模急剧增长。海量数据的收集、存储、处理、分析以及由此而产生的信息服务正在成为全球信息技术发展的主流。如果说大数据是信息时代的“石油”，那么大数据处理就是信息时代对这些数据“石油”的开采、运输、加工和提炼过程。可以预见，我们未来的生活将会像我们依赖石油化工产品一样依赖丰富多彩的大数据分析应用和信息服务。

在这一轮大数据的发展和变革中，中国以其过去三十几年信息化和近几年移动互联网和物联网发展所积累的基础，正面临着前所未有的良好发展机遇。与西方发达国家相比，中国在大数据技术发展方面有不少自身独特的优势：中国巨大的人口基数孕育着巨大的技术市场，同时也造就巨大的数据资源。因此，在这一轮新的变革中，一方面，我们面临着很多技术挑战，需要努力学习国外先进的技术和经验，需要去不断探索和发现很多新的数据分析应用商业模式；但另一方面，上述的一些独特优势，加上大数据领域目前还处于初期阶段，这些因素使得我们有很好的机会和一定的优势与其他发达市场一道，探索如何通过大数据技术来进一步提高数据信息分析应用的技术水平与服务质量，并以此改善我们的生活。因此，在大数据的研究与应用方面，中国和西方发达国家可以齐头并进、互相借鉴。

我很欣慰地看到，作为国内最早从事大数据技术研究和教学的团队之一，南京大学黄宜华教授和他的大数据实验室同仁们在大数据技术领域已经进行了多年系统深入的研究工作，取得了卓有成效的研究成果。我和黄教授相识于两年多前的中国大数据技术大会上，黄教授的学识和为人令人钦佩。此后我们在大数据研究方面展开了建设性的合作。

英特尔作为一家全球领先的计算技术公司，长期以来始终以计算技术的创新为己任。在大数据处理技术方面，我们也竭尽全力发挥出我们在软硬件平台的组合优势引领大数据技术的全

面发展和推广。让人欣喜的是，英特尔中国团队是英特尔 Hadoop 系统开发的主力军。这也为我们与黄教授的合作创造了得天独厚的条件。

这本《深入理解大数据》的力作正是我们双方在大数据领域共同努力的结晶，是以学术界和业界完美结合的方式，在融合了学术界系统化的研究教学工作和业界深度的系统和应用研发工作基础上，成功打造出的一本大数据技术佳作。

本书在总结多年的技术研发和教学内容的基础上，深入浅出地概括了大数据的基本概念和技术内容，然后重点介绍了主流大数据处理系统 Hadoop 的基本原理和架构；在此基础上逐章详细介绍了 Hadoop 平台下大数据分布存储、并行化计算和算法设计等一系列重要技术及其编程方法，尤其是详细介绍了大量实际的大数据处理算法设计和编程实现方法。相信这是一本适合软件技术人员和 IT 行业管理人员理解和掌握大数据技术的不可多得的技术书籍，也是一本适合于在校大学生和研究生学习和掌握大数据处理和编程技术的好教材。

在未来十年里，我们将看到数以十亿计的移动设备、可穿戴设备和智能终端设备融入我们生活的方方面面，沉浸式计算体验（Immersive Computing Experience）将成为我们生活的常态。随之应运而生的海量数据，大数据的存储和处理将分布于数据生命周期的各个阶段，因此，我们需要更多、更便捷的大数据处理方法和能力。显然，这并非一个算法、一个软件或一个高性能处理器所能单独完成的事情。我们需要以开放和软硬件结合的综合性体系架构，来实现大规模的大数据分析处理系统的部署和使用。基于英特尔架构优化的 Hadoop 平台是一个很好的开端。

大数据技术带来的变革方兴未艾，我们正在打造开启信息技术新时代的金钥匙。我也衷心地希望这本书能成为读者打开大数据技术之门的钥匙！

英特尔亚太研发有限公司总经理

何京翔





## 推荐序三

据预测，到 2020 年，全球包含 PC、平板电脑、智能手机等联网设备将超过 300 亿台。实际上，随着物联网技术与可穿戴设备的飞速发展，终端设备会远远大于这个数量。随之而来各种应用也会爆炸式增长。大量应用会产生巨量的数据，数据内容的种类也会非常多样化，比如大量的普通数据、医疗影像数据以及越来越多城市摄像头所录下的视频数据等。

根据国际分析机构 IDC 的统计，全球不同设备产生的数据量，到 2020 年预计将会突破 40ZB。如此海量、持续、细粒度、多样化的数据，让各个行业都看到了数据的巨大潜在价值，这将大力推动大数据技术和应用的发展，为当今和未来的科技和经济发展以及社会的生产和生活带来重大影响。

目前，全球的大数据市场规模很大，并保持了 30% 以上的年增长率。在中国，据 2012 年的统计，中国占据全球数据总量的 13%；而据预计，随着中国的不断发展，作为全球第二大经济体，中国将拥有全球最高的终端设备出货量以及全球最高的物联网的用户数，并且我们的增长速度也将超过全球。到 2020 年，中国的整个数据量将超过 8ZB，也就是说，数据增长率将是 2012 年的 23 倍。虽然中国的大数据解决方案才刚刚起步，但是预计在未来五年内中国大数据市场将会保持 50% 的增长率。大数据市场在未来几年将拥有整体 IT 市场 4 倍的增长速度、服务器市场 5 倍的增长速度，并且将远远高于云计算市场的增长速度。大数据市场在中国的 IT 行业已经变得越来越重要。

近年来，大数据的各种应用层出不穷。在政府行业的“平安城市”项目中，很多摄像头在收集视频数据，通过这些视频数据的管理和分析，可有效预防犯罪、保障社会公共安全。此外，互联网舆情分析、地震预测、气象分析、人口信息综合分析等，也都是政府民生相关的大数据应用领域。

在金融行业，很多行业用户使用大数据解决方案，对海量结构化交易数据进行实时入库处理，并提供并发查询，进行金融欺诈分析监测以预防金融犯罪；还可以通过数据分析挖掘以进

行精准的金融营销，通过对金融分析发现更多的投资组合、避免投资风险等。

电信行业也开始使用大数据解决方案，提供对上亿电话通联详单数据的快速查询和分析，并可以通过对电信用户数据的分析，提供基于 LBS（基于位置的服务）的数据分析服务，进行电信产品和服务的精准营销、精准广告投放和促销等。

零售行业也开始使用大数据解决方案，通过对用户的交易数据进行关联性分析挖掘以决定商品在货架上的摆放位置，在方便用户购物的同时、提高用户的购买量。

交通领域也逐步采用智能交通管理方案，通过对道路的交通状况进行分析预测，实现智能化道路交通管理和分流，对违章进行自动检测和处理，完成套牌车辆检测、区域分析以及其他异常行为的检测、分析和预警。其他还有像铁路、航空等交通行业的客票和货运处理、物流管理等，都将成为典型的大数据应用领域。

在医疗行业，对于医疗影像（如 X 光片、CT 片等）、就诊、用药、手术、住院状况等医疗数据的信息化管理要求越来越高。同时目前医疗行业也开始关注如何通过对医疗大数据进行融合分析，为医生提供辅助诊断、医疗方案推荐、药物疗效分析、病因分析、专家治疗经验共享等基于大数据的智能化诊疗服务。

大数据广泛的应用前景代表了 IT 行业的未来。近年来，大数据的巨大应用需求推动了大数据处理技术取得了长足的进步和发展。但是，大数据的 4V 特性（大体量、多样性、时效性、以及精确性）决定了大数据处理仍然面临着巨大的技术困难和挑战，因此，我们还需要大力推动大数据技术的研发和应用。这就需要培养更多熟练掌握大数据处理技术的专业人才。而今天的人才市场上还极为缺乏这种熟练掌握大数据技术的专业人才。

为此，需要让更多的专业技术人员学习和掌握大数据技术，这正是我们编写这本 Hadoop 大数据处理技术书籍的主要动机和目的。本书希望通过对目前最为主流、最广为业界接受使用的 Hadoop 大数据处理和编程技术的深入介绍，对 IT 专业技术人员与学生学习和掌握大数据技术起到较大的帮助作用！

英特尔中国大数据首席技术官

苗凯翔博士

2014 年 3 月 20 日，于上海

# 丛书序言

## ——计算机专业学生系统能力培养和系统课程设置的研究

未来的 5 ~ 10 年是中国实现工业化与信息化融合,利用信息技术与装备提高资源利用率、改造传统产业、优化经济结构、提高技术创新能力与现代化管理水平的关键时期,而实现这一目标,对于高效利用计算系统的其他传统专业的专业人员需要了解和掌握计算思维,对于负责研发多种计算系统的计算机专业的专业人员则需要具备系统级的设计、实现和应用能力。

### 1. 计算技术发展特点分析

进入本世纪以来,计算技术正在发生重要发展和变化,在上世纪个人机普及和 Internet 快速发展基础上,计算技术从初期的科学计算与信息处理进入了以移动互联、物物相联、云计算与大数据计算为主要特征的新型网络时代,在这一发展过程中,计算技术也呈现出以下新的系统形态和技术特征。

#### (1) 四类新型计算系统

1) **嵌入式计算系统** 在移动互联网、物联网、智能家电、三网融合等行业技术与产业发展中,嵌入式计算系统有着举足轻重和广泛的作用。例如,移动互联网中的移动智能终端、物联网中的汇聚节点、“三网融合”后的电视机顶盒等是复杂而新型的嵌入式计算系统;除此之外,新一代武器装备,工业化与信息化融合战略实施所推动的工业智能装备,其核心也是嵌入式计算系统。因此,嵌入式计算将成为新型计算系统的主要形态之一。在当今网络时代,嵌入式计算系统也日益呈现网络化的开放特点。

2) **移动计算系统** 在移动互联网、物联网、智能家电以及新型装备中,均以移动通信网络为基础,在此基础上,移动计算成为关键技术。移动计算技术将使计算机或其他信息智能终端设备在无线环境下实现数据传输及资源共享,其核心技术涉及支持高性能、低功耗、无线连接和轻松移动的移动处理机及其软件技术。

**3) 并行计算系统** 随着半导体工艺技术的飞速进步和体系结构的不断发展,多核/众核处理机硬件日趋普及,使得昔日高端的并行计算呈现出普适化的发展趋势;多核技术就是在处理器上拥有两个或更多一样功能的处理器核心,即将数个物理处理器核心整合在一个内核中,数个处理器核心在共享芯片组存储界面的同时,可以完全独立地完成各自操作,从而能在平衡功耗的基础上极大地提高 CPU 性能;其对计算系统微体系结构、系统软件与编程环境均有很大影响;同时,云计算也是建立在廉价服务器组成的大规模集群并行计算基础之上。因此,并行计算将成为各类计算系统的基础技术。

**4) 基于服务的计算系统** 无论是云计算还是其他现代网络化应用软件系统,均以服务计算为核心技术。服务计算是指面向服务的体系结构(SOA)和面向服务的计算(SOC)技术,它是标识分布式系统和软件集成领域技术进步的一个里程碑。服务作为一种自治、开放以及与平台无关的网络化构件可使分布式应用具有更好的复用性、灵活性和可增长性。基于服务组织计算资源所具有的松耦合特征使得遵从 SOA 的企业 IT 架构不仅可以有效保护企业投资、促进遗留系统的复用,而且可以支持企业随需应变的敏捷性和先进的软件外包管理模式。Web 服务技术是当前 SOA 的主流实现方式,其已经形成了规范的服务定义、服务组合以及服务访问。

## (2) “四化”主要特征

**1) 网络化** 在当今网络时代,各类计算系统无不呈现出网络化发展趋势,除了云计算系统、企业服务计算系统、移动计算系统之外,嵌入式计算系统也在物联时代通过网络化成为开放式系统。即,当今的计算系统必然与网络相关,尽管各种有线网络、无线网络所具有的通信方式、通信能力与通信品质有较大区别,但均使得与其相联的计算系统能力得以充分延伸,更能满足应用需求。网络化对计算系统的开放适应能力、协同工作能力等也提出了更高的要求。

**2) 多媒体化** 无论是传统 Internet 应用服务,还是新兴的移动互联网服务业务,多媒体化是其面向人类、实现服务的主要形态特征之一。多媒体技术是利用计算机对文本、图形、图像、声音、动画、视频等多种信息进行综合处理、建立逻辑关系和人机交互作用的新技术。多媒体技术使计算机可以处理人类生活中最直接、最普遍的信息,从而使得计算机应用领域及功能得到了极大的扩展,使计算机系统的人机交互界面和手段更加友好和方便。多媒体具有计算机综合处理多种媒体信息的集成性、实时性与交互性特点。

**3) 大数据化** 随着物联网、移动互联网、社会化网络的快速发展,半结构化及非结构化的数据呈几何倍增长。数据来源的渠道也逐渐增多,不仅包括了本地的文档、音视频,还包括网络内容和社交媒体;不仅包括 Internet 数据,更包括感知物理世界的的数据。从各种类型的数据中快速获得有价值信息的能力,称为大数据技术。大数据具有体量巨大、类型繁多、价值密度低、处理速度快等特点。大数据时代的来临,给各行各业的数据处理与业务发展带来重要变

革,也对计算系统的新型计算模型、大规模并行处理、分布式数据存储、高效的数据处理机制等提出了新的挑战。

**4) 智能化** 无论是计算系统的结构动态重构,还是软件系统的能力动态演化;无论是传统 Internet 的搜索服务,还是新兴移动互联的位置服务;无论是智能交通应用,还是智能电网应用,无不显现出鲜明的智能化特征。智能化将影响计算系统的体系结构、软件形态、处理算法以及应用界面等。例如,相对于功能手机的智能手机是一种安装了开放式操作系统的手机,可以随意安装和卸载应用软件,具备无线接入互联网、多任务和复制粘贴以及良好用户体验等能力;相对于传统搜索引擎的智能搜索引擎是结合了人工智能技术的新一代搜索引擎,不仅具有传统的快速检索、相关度排序等功能,更具有用户角色登记、用户兴趣自动识别、内容的语义理解、智能信息化过滤和推送等功能,其追求的目标是根据用户的请求从可以获得的网络资源中检索出对用户最有价值的信息。

## **2. 系统能力的主要内涵及培养需求**

### **(1) 主要内涵**

计算机专业学生的系统能力的核心是掌握计算系统内部各软件/硬件部分的关联关系与逻辑层次;了解计算系统呈现的外部特性以及与人 and 物理世界的交互模式;在掌握基本系统原理的基础上,进一步掌握设计、实现计算机硬件、系统软件以及应用系统的综合能力。

### **(2) 培养需求**

要适应“四类计算系统,四化主要特征”的计算技术发展特点,计算机专业人才培养必须“与时俱进”,体现计算技术与信息产业发展对学生系统能力培养的需求。在教育思想上要突现系统观教育理念,在教学内容中体现新型计算系统原理,在实践环节上展现计算系统平台技术。

要深刻理解系统化专业教育思想对计算机专业高等教育过程所带来的影响。系统化教育和系统能力培养要采取系统科学的方法,将计算对象看成一个整体,追求系统的整体优化;要夯实系统理论基础,使学生能够构建出准确描述真实系统的模型,进而能够用于预测系统行为;要强化系统实践,培养学生能够有效地构造正确系统的能力。

从系统观出发,计算机专业的教学应该注意教学生怎样从系统的层面上思考(设计过程、工具、用户和物理环境的交互),讲透原理(基本原则、架构、协议、编译以及仿真等),强化系统性的实践教学培养过程和内容,激发学生的辩证思维能力,帮助他们理解和掌控数字世界。

## **3. 计算机专业系统能力培养课程体系设置总体思路**

为了更好地培养适应新技术发展的、具有系统设计和系统应用能力的计算机专门人才,我们需要建立新的计算机专业本科教学课程体系,特别是设立有关系统级综合性课程,并重新规划计算机系统核心课程的内容,使这些核心课程之间的内容联系更紧密、衔接更顺畅。



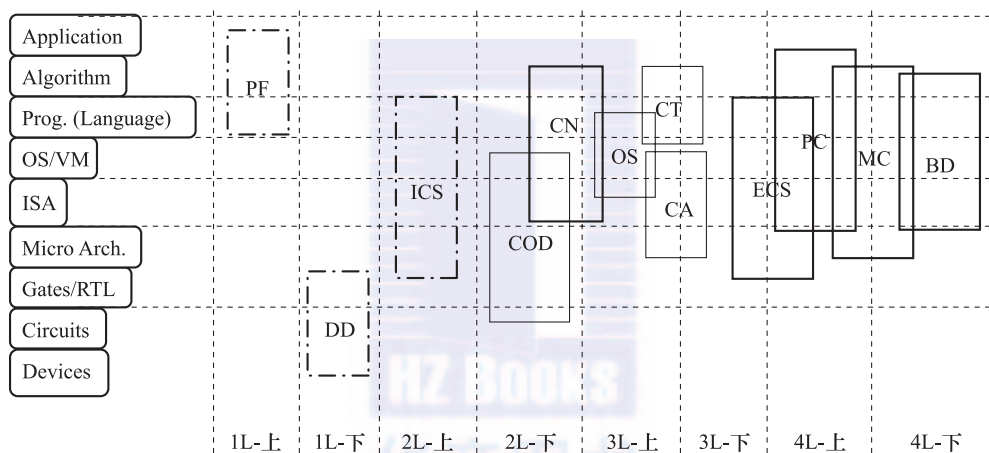
我们建议把课程分成三个层次：计算机系统基础课程、重组内容的核心课程、侧重不同计算系统的若干相关平台应用课程。

第一层次核心课程包括：“程序设计基础（PF）”、“数字逻辑电路（DD）”和“计算机系统基础（ICS）”。

第二层次核心课程包括：“计算机组成与设计（COD）”、“操作系统（OS）”、“编译技术（CT）”和“计算机系统结构（CA）”。

第三层次核心课程包括：“嵌入式计算系统（ECS）”、“计算机网络（CN）”、“移动计算（MC）”、“并行计算（PC）”和“大数据并行处理技术（BD）”。

基于这三个层次的课程体系中相关课程设置方案如下图所示。



图中左边部分是计算机系统的各个抽象层，右边的矩形表示课程，其上下两条边的位置标示了课程内容在系统抽象层中的涵盖范围，矩形的左右两条边的位置标示了课程大约在哪个年级开设。点划线、细实线和粗实线分别表示第一、第二和第三层次核心课程。

从图中可以看出，该课程体系的基本思路是：先讲顶层比较抽象的编程方面的内容；再讲底层有关系统的具体实现基础内容；然后再从两头到中间，把顶层程序设计的内容和底层电路的内容按照程序员视角全部串起来；在此基础上，再按序分别介绍计算机系统硬件、操作系统和编译器的实现细节。至此的所有课程内容主要介绍单处理器系统的相关内容，而计算机体系结构主要介绍各不同并行粒度的体系结构及其相关的操作系统实现技术和编译器实现技术。第三层次的课程没有先后顺序，而且都可以是选修课，课程内容应体现第一和第二层次课程内容的螺旋式上升趋势，也即第三层次课程内容涉及的系统抽象层与第一和第二层次课程涉及的系统抽象层是重叠的，但内容并不是简单重复，应该讲授在特定计算系统中的相应教学内容。例如，对于“嵌入式计算系统（ECS）”课程，虽然它所涉及的系统抽象层与“计算机系统基础

(ICS)”课程涉及的系统抽象层完全一样，但是，这两门课程的教学内容基本上不重叠。前者着重介绍与嵌入式计算系统相关的指令集体系结构设计、操作系统实现和底层硬件设计等内容，而后者着重介绍如何从程序员的角度来理解计算机系统设计及实现中涉及的基础内容。

与传统课程体系设置相比，最大的不同在于新的课程体系中有一门涉及计算机系统各个抽象层面的能够贯穿整个计算机系统设计及实现的基础课程：“计算机系统基础（ICS）”。该课程讲解如何从程序员角度来理解计算机系统，可以使程序员进一步明确程序设计语言中的语句、数据和程序是如何在计算机系统中实现和运行的，让程序员了解不同的程序设计方法为什么会有不同的性能等。

此外，新的课程体系中，强调课程之间的衔接和连贯，主要体现在以下几个方面。

1) “计算机系统基础”课程可以把“程序设计基础”和“数字逻辑电路”之间存在于计算机系统抽象层中的“中间间隔”填补上去并很好地衔接起来，这样，到2L-上结束的时候，学生就可以通过这三门课程清晰地建立单处理器计算机系统的整机概念，构造出完整的计算机系统的基本框架，而具体的计算机系统各个部分的实现细节再通过后续相关课程来细化充实。

2) “数字逻辑电路”、“计算机组成与设计”、“嵌入式计算系统”中的实验内容之间能够很好地衔接，可以规划一套承上启下的基于FPGA开发板的综合实验平台，让学生在一个统一的实验平台上从门电路开始设计基本功能部件，然后再以功能部件为基础设计CPU、存储器和外围接口，最终将CPU、存储器和I/O接口通过总线互连为一个完整的计算机硬件系统。

3) “计算机系统基础”、“计算机组成与设计”、“操作系统”和“编译技术”之间能够很好地衔接。新课程体系中“计算机系统基础”和“计算机组成与设计”两门课程对原来的“计算机系统概论”和“计算机组成原理”的内容进行了重新调整和统筹规划，这两门课程的内容是相互密切关联的。对于“计算机系统基础”与“操作系统”、“编译技术”的关系，因为“计算机系统基础”以Intel x86为模型机进行讲解，所以它为“操作系统”（特别是Linux内核分析）提供了很好的体系结构基础。同时，在“计算机系统基础”课程中为了清楚地解释程序中的文件访问和设备访问等问题，会从程序员角度简单引入一些操作系统中的相关基础知识。此外，在“计算机系统基础”课程中，会讲解高级语言程序如何进行转换、链接以生成可执行代码的问题；“计算机组成与设计”中的流水线处理等也与编译优化相关，而且“计算机组成与设计”以MIPS为模型机进行讲解，而MIPS模拟器可以为“编译技术”的实验提供可验证实验环境，因而“计算机系统基础”和“计算机组成与设计”两门课程都与“编译技术”有密切的关联。“计算机系统基础”、“计算机组成与设计”、“操作系统”和“编译技术”这四门课程构成了一组计算机系统能力培养最基本的核心课程。

从“计算机系统基础”课程的内容和教学目标以及开设时间来看，位于较高抽象层的先行

课（如程序设计基础和数据结构等课程）可以按照原来的内容和方式开设和教学，而作为新的“计算机系统基础”和“计算机组成与设计”先导课的“数字逻辑电路”，则需要对传统的教学内容，特别是实验内容和实验手段方面进行修改和完善。

有了“计算机系统基础”和“计算机组成与设计”课程的基础，作为后续课程的操作系统、编译原理等将更容易被学生从计算机系统整体的角度理解，课程内容方面不需要大的改动，但是操作系统和编译器的实验要以先行课程实现的计算机硬件系统为基础，这样才能形成一致的、完整的计算机系统整体概念。

本研究还对 12 门课程的规划思路、主要教学内容及实验内容进行了研究和阐述，具体内容详见公开发表的研究报告。

#### 4. 关于本研究项目及本系列教材

机械工业出版社华章公司在较早的时间就引进出版了 MIT、UC-Berkeley、CMU 等国际知名院校有关计算机系统课程的多种教材，并推动和组织了计算机系统能力培养相关的研究，对国内计算机系统能力培养起到了积极的促进作用。

本项研究是教育部 2013 ~ 2017 年计算机类专业教学指导委员会“计算机类专业系统能力培养研究”项目之一，研究组成员由国防科技大学王志英、北京航空航天大学马殿富、西北工业大学周兴社、南开大学吴功宜、武汉大学何炎祥、南京大学袁春风、北京大学陈向群、中国科技大学安虹、天津大学张刚、机械工业出版社华章公司温莉芳等组成，研究报告分别发表于中国计算机学会《中国计算机科学技术发展报告》及《计算机教育》杂志。

本系列教材编委会在上述研究的基础上对本套教材的出版工作经过了精心策划，选择了对系统观教育和系统能力培养有研究和实践的教师作为作者，以系统观为核心编写了本系列教材。我们相信本系列教材的出版和使用，将对提高国内高校计算机类专业学生的系统能力和整体水平起到积极的促进作用。

“计算机类专业系统能力培养系列教材”编委会组成如下：

主 任 王志英

副主任 马殿富

委 员 周兴社 吴功宜 何炎祥 袁春风 陈向群 安 虹 温莉芳

秘 书 姚 蕾

此外，本系列教材的出版得到赛灵思电子科技有限公司和英特尔有限公司的支持。

“计算机类专业系统能力培养系列教材”编委会

2014 年 5 月



# 前言

2012 年以来，大数据（Big Data）技术在全世界范围内迅猛发展，在全球学术界、工业界和各国政府得到了高度关注和重视，掀起了一场可与 20 世纪 90 年代的信息高速公路相提并论的发展热潮。

大数据技术如此重要，已经被我国政府提升到国家重大发展战略的高度。2014 年我国政府工作报告中指出：“设立新兴产业创业创新平台，在新一代移动通信、集成电路、大数据、先进制造、新能源、新材料等方面赶超先进，引领未来产业发展”。由此可见，大数据已经被我国政府列为推动国家科技创新和引领经济结构优化升级、赶超国际先进水平、引领国家未来产业发展的战略性计划。两会期间，CCTV 中央电视台的新闻报道开创性地引入了大数据新闻报道手段，以大数据说话，高频率使用大数据报道两会重大新闻，引起了全国民众的普遍关注和兴趣。

大数据也同样成为各发达国家政府高度关注的战略性高科技技术和产业。2012 年 3 月，美国总统奥巴马签署并发布了一个“大数据研究发展创新计划”（Big Data R&D Initiative），投资 2 亿美元启动大数据技术和工具研发，这是继 1993 年美国宣布“信息高速公路”计划后的又一次重大科技发展部署。美国政府认为大数据是“未来的新石油”，将大数据研究上升为国家意志，认为大数据将对未来的科技与经济发展带来重大影响，一个国家拥有数据的规模和运用数据的能力将成为综合国力的重要组成部分，对数据的占有和控制也将成为国家间和企业间新的争夺焦点。在随后的近两年里，英国、法国、德国、日本等发达国家政府都纷纷推出了相应的大数据发展战略计划。

《大数据时代》一书的作者、英国牛津大学教授、被誉为“大数据时代预言家”的维克托·迈尔-舍恩伯格认为：“大数据开启了一次重大的时代转型”，认为大数据将带来巨大的变革，改变我们的生活、工作和思维方式，改变我们的商业模式，影响我们的经济、政治、科技和社会等各个层面。他认为，大数据将成为企业的核心竞争力，成为一种商业资本，成为企业的重要资产。

大数据技术最大的推动力来自于行业应用需求。过去几年来，随着计算机和信息技术的迅猛发展和普及应用，行业应用系统的规模迅速扩大，行业应用所产生的数据量呈爆炸性增长。动辄达到 PB 级规模的行业 / 企业大数据已经远远超出了现有传统的计算技术和信息系统的处

理能力。另一方面，人们发现，大数据在带来巨大技术挑战的同时，也带来巨大的商业价值，带来巨大的技术创新与商业机遇。大数据巨大的应用需求和隐含的深度价值极大地推动了大数据技术的快速发展，促进了大数据所涉及到的各个技术层面和系统平台方面的长足发展。

在大数据处理的众多技术和系统中，起到开创性作用、最为主流的当数 Google 公司在 2003 年发明的 MapReduce 技术以及随后在 2007 年由开源组织 Apache 推出的开源的 Hadoop MapReduce 技术和系统。目前，Hadoop 已经成为全世界最为成功和最广为接受使用的主流大数据处理技术平台，在国内外几乎所有知名 IT 和互联网企业中都得到推广使用，成为了事实上的大数据处理工业标准。

除了知名 IT 和互联网企业外，大数据技术的迅猛发展与行业应用需求的快速增长，也推动了其他各个行业对相关大数据处理与应用技术的高度关注。近年来，国内外越来越多的典型行业开始制定和启动了行业大数据处理应用开发计划，期冀使用大数据处理技术管理和分析企业大数据。然而，目前国内外的实际状况是，由于 MapReduce 等相关大数据处理技术发展较新，除了知名 IT 和互联网企业能娴熟运用外，目前大多数应用行业普遍不熟悉这方面的技术，即使很多中小规模的软件公司也不掌握这方面的开发技术，大多刚刚开始关注和学习这方面的技术。同时，由于国内外绝大多数高校对大数据技术的关注较晚，有关大数据方面的课程教学和人才培养工作未能跟上技术市场的变化和 demand。这些因素使得目前技术市场上大数据技术人才严重短缺。

幸运的是，本团队早在 2010 年开始即开始关注大数据处理技术，并开展了系统的大数据技术教学和研究工作。2009 年底，Google 中国公司大学合作部在清华大学举行了 MapReduce 海量数据处理技术培训班。培训班结束后，在 Google 中国公司大学合作部精品课程计划资助下，由本人负责在南京大学建设了 MapReduce 大规模数据并行处理技术课程，并自 2011 年开始为南京大学计算机系研究生开设了该课程，使我们成为国内最早系统性从事 MapReduce 大规模数据并行处理技术教学的院校之一。课程开设后取得非常好的教学效果。课程同学组队参加了 2012 年由中国云计算产业联盟主办的首届“中国云计算·移动互联网创新大赛”，夺得 4 个大数据赛题全部 17 个奖项中的 8 项大奖，获得奖金 20 万元；在 2013 年由中国计算机学会主办的“第一届中国大数据技术创新大赛”上夺得大赛唯一的一项一等奖，获得奖金 10 万元。本书的很多章节内容正是在总结所开设课程内容、上述大赛获奖算法以及部分同学的优秀课程设计内容基础上组织形成。

为了满足专业技术人员学习 MapReduce 相关技术的需求，近几年来，国内陆续推出了一些有关 Hadoop MapReduce 的编程技术书籍，为计算机专业人员学习和掌握 Hadoop 编程技术提供了很有价值的学习资料。然而，目前出版的这些书籍，大多是参照 Hadoop 官方技术文档和资料整理编写而成，主要集中在对 Hadoop 编程接口以及简单编程示例的介绍，对 MapReduce 技术背后系统的工作原理、编程模型、设计思想以及编程和算法设计方法介绍不够深入和系统，使得这些编程手册性的技术书籍不太适宜作为高校课程教学或者初学者自学时的教材使用。

根据近 5 年来我们开展 Hadoop MapReduce 大数据并行处理技术课程教学中所发现的问题和总结出的经验,相对而言,Hadoop MapReduce 的基本工作原理、编程接口和简单示例程序都比较容易学习和理解。但是,学习者和程序员普遍感到困惑的是,针对稍微复杂一些的实际的大数据处理和算法设计问题(如设计实现一个机器学习和数据分析算法),由于 MapReduce 并行程序设计与传统的程序设计技术方法有较大的不同,如何依据数据本身的特点以及 MapReduce 并行程序设计思想和并行算法设计方法,对这些实际问题分析并理清其 MapReduce 程序或算法设计思路、并最终完成编程实现,对此大家普遍感到有一定的困难和障碍。另一方面,在我们接触到的一些对 Hadoop 已有一定编程经验、希望对开源 Hadoop 系统的优化增强功能和深度技术做进一步了解的技术人员中,会感到现有的书籍资料中大多找不到这方面的技术内容。

为此,我们在总结多年来 MapReduce 并行处理技术课程教学经验和成果的基础上,与业界著名企业 Intel 公司的大数据技术和产品开发团队和资深工程师联合,以学术界的教学成果与业界高水平系统研发经验完美结合,在理论联系实际的基础上,在基础理论原理、实际算法设计方法以及业界深度技术三个层面上,精心组织材料编写完成了本书。

全书从 Hadoop MapReduce 技术与系统的基本原理剖析着手,在系统介绍基础理论原理、设计思想和编程模型的基础上,介绍编程框架与接口,然后着重系统化地介绍 MapReduce 并行算法设计与编程技术,由浅入深,循序渐进,较为全面地介绍和覆盖了基本 MapReduce 算法设计、高级 MapReduce 编程技术以及一系列较为复杂的机器学习和数据挖掘并行化算法,并介绍了来自 Intel Hadoop 系统产品的一系列增强功能及其深度的技术剖析;最后,为了给读者进一步介绍一些综合性和实战性的算法设计和编程案例,本书收集了一系列实战性的大数据处理和算法设计问题,这些问题来自本课程同学参加的全国性大数据大赛中的获奖算法、本课程中的优秀课程设计以及来自本团队的科研课题及业界实际的大数据应用实战案例。

与市场上现有的一些同类书籍相比,本书主要有三大特点:第一个特点是,对 MapReduce 的基本工作原理和编程模型等基础理论原理有较为系统深入的阐述,让读者对 MapReduce 技术有一个理论上的深入理解,为后期学习和合理运用算法设计方法打下一个较为坚实的理论基础;第二个特点是,对于那些看懂了基本原理和接口、却苦于难以下手去具体设计和编程实现实际大处理处理算法问题的读者来说,本书会重点给他们讲解 MapReduce 并行程序和算法设计思路和方法,重点介绍和展示如何根据大数据问题本身的特点和 MapReduce 并行程序设计特点,将一个大数据问题或算法转化为 MapReduce 并行化算法设计思路和实现方法,然后再辅以详细的程序实现代码加以分析介绍;第三个特点是,对于那些有较好基础、希望了解更深入技术的读者来说,本书引入了开源技术书籍资料中所没有的来自业界产品的增强功能和深度技术内容。

本书共分 11 章,分为两大部分,其中第一部分包括第 1 ~ 7 章,主要介绍 Hadoop 系统的相关技术内容;第二部分包括第 8 ~ 11 章,主要介绍 MapReduce 的编程和算法设计。

第1章 大数据处理技术简介，简要介绍大数据并行处理技术的基本概念和技术内容，MapReduce 并行计算技术设计思想和功能特点，以及 Hadoop 系统的基本组成和构架。

第2章 Hadoop 系统的安装和操作管理，介绍 Hadoop 系统的安装和操作管理方法。

第3章 大数据存储——分布式文件系统 HDFS，介绍分布式文件系统 Hadoop HDFS 的基本组成和工作原理、HDFS 文件系统操作命令、HDFS 的基本编程接口和编程示例。

第4章 Hadoop MapReduce 并行编程框架，介绍 Hadoop MapReduce 并行编程模型、框架、基本构架和工作过程以及 MapReduce 编程接口。

第5章 分布式数据库 HBase，介绍分布式数据库 HBase 的基本功能特点和组成结构、数据模型、HBase 的安装与操作、HBase 的编程接口和编程示例，并深度介绍 HBase 的读写操作特性和 HBase 的一些高级功能。

第6章 分布式数据仓库 Hive，介绍分布式数据仓库 Hive 的基本功能特点和结构组成、数据模型、Hive 的安装和操作、Hive 的查询语言 HiveQL 以及 Hive JDBC 编程技术。

第7章 Intel Hadoop 系统优化与功能增强，介绍 Hadoop Intel 系统优化与功能增强以及 Intel Hadoop 系统的安装管理，然后详细介绍和解读 Intel Hadoop 系统 HDFS 的高级功能扩展、HBase 的高级功能扩展与编程示例以及 Hive 的高级功能扩展与编程示例。

第8章 MapReduce 基础算法程序设计，介绍 MapReduce 的基础算法设计，包括 WordCount、矩阵乘法、关系代数运算、单词共现算法、文档倒排索引、PageRank 网页排名算法以及专利文献分析算法设计方法和编程实现。本章所有算法均有完整实现代码供下载学习。

第9章 MapReduce 高级程序设计技术，介绍 MapReduce 高级程序设计技术，包括复合键值对的使用、用户定制数据类型、用户定制输入输出格式、用户定制 Partitioner 和 Combiner、组合式 MapReduce 计算作业、多数据源的连接、全局参数 / 数据文件的传递与使用以及关系数据库的连接与访问技术。

第10章 MapReduce 数据挖掘基础算法，介绍基于 MapReduce 的机器学习和数据挖掘并行化算法设计方法和编程实现，包括 K-Means 聚类算法、最近邻分类算法、朴素贝叶斯分类算法、决策树分类算法、频繁项集挖掘算法以及隐马尔科夫模型和最大期望算法。本章所有算法均有完整实现代码供下载学习。

第11章 大数据处理算法设计与应用编程案例，介绍大数据处理算法与基本应用编程案例的算法设计和编程实现，包括基于 MapReduce 的搜索引擎算法、基于 MapReduce 的大规模短文本多分类算法、基于 MapReduce 的大规模基因序列比对算法、基于 MapReduce 的大规模城市路径规划算法、基于 MapReduce 的大规模重复文档检测算法、基于内容的并行化图像搜索算法与引擎、基于 MapReduce 的大规模微博传播分析、基于关联规则挖掘的图书推荐算法、以及基于 Hadoop 的城市智能交通综合应用案例。



本书由南京大学计算机系 PASA 大数据实验室黄宜华教授担任主编、并负责全书内容的组织和编审，Intel 公司大数据软件部首席工程师苗凯翔担任副主编。其余作者主要来自南京大学计算机系 PASA 大数据实验室以及 Intel 公司大数据软件部；此外，北京神州立诚科技有限公司也参加了部分章节的编写。

本书第 1 章由黄宜华、周珊编写完成，第 2 章由仇红剑编写完成，第 3 章由赵頔编写完成，第 4 章由黄宜华、沈仪和赵博编写完成，第 5 章由姜伟华、杜竟成编写完成，第 6 章由萧少聪、韩小姣编写完成，第 7 章由陈建忠、王星宇、王毅、Manoj Shanmugasundaram 编写完成，第 8 章由唐云、金磊编写完成，第 9 章由黄宜华编写完成，第 10 章由金磊、赵頔、仇红剑、顾荣编写完成，第 11 章由顾荣、赵博、韦永壮、笱庆、陈虎、李相臣、彭岳、王刚、姬浩、张同宝、陈新江编写完成。

衷心感谢 Intel 公司大数据软件部参编工程师在本书写作过程中的共同努力和辛苦付出！特别感谢 Intel 公司 Evelyn Yan（颜历）、Kally Wang（王星宇）、Yale Wang（王毅）在本书写作过程中所做的大量组织协调工作！也感谢本团队所在南京大学计算机系 PASA 大数据实验室所有参编作者的辛苦努力和付出！感谢机械工业出版社华章分社在本书编写和出版过程中的大力支持和帮助！

Google 中国公司大学合作部在过去的几年中为我们开设大数据技术课程给予了大力的支持和帮助，在此，谨向 Google 中国公司表示衷心的感谢！同时也要衷心感谢清华大学郑纬民教授和陈康副教授，他们为 2009 年 Google 公司的技术培训提供了全部课件并进行了主讲，使我们获益匪浅，该课件也为后期本课程的建设提供了良好的基础。此外，也要衷心感谢南京大学计算机软件新技术国家重点实验室，早在 2010 年即投入 100 万元资助购建了一个科研教学专用的 MapReduce 大数据并行处理集群，为几年来本团队的教学科研和本系其他诸多课题组的研究工作提供了良好的计算设施和条件。

本书是国内第一本经过多年课堂教学实践后撰写而成的大数据处理和并行编程技术书籍，因此，本书非常适合于高等院校作为 MapReduce 大数据并行处理技术课程教材使用，同时也很适合于高等院校学生作为自学 MapReduce 并行处理技术的参考书。与此同时，由于本书包含很多来自业界实际产品的深度技术内容，并包括了丰富的算法设计和编程实战案例，因此本书也非常适合于作为 IT 和其他应用行业专业技术人员从事大数据处理应用开发和编程工作时的参考手册。

书中第 8 章和第 10 章全部算法设计的程序代码都经过本团队完整编程实现并运行通过，源码可在与本书配套的南京大学 PASA 大数据实验室（PASA：Parallel Algorithms, Systems, and Applications）网站上下载：<http://pasa-bigdata.nju.edu.cn/links.html>。

由于作者水平有限，书中难免会有不准确甚至错误之处。不当之处敬请读者批评指正，并将反馈意见发送到邮箱：[feedback\\_bigdata@163.com](mailto:feedback_bigdata@163.com)，以便我们再版时修正错误。

南京大学计算机科学与技术系 PASA 大数据实验室

黄宜华

2014 年 3 月 18 日，于南京

# 目 录

推荐序一	
推荐序二	
推荐序三	
丛书序言	
前 言	

## 第一部分 Hadoop 系统

第 1 章 大数据处理技术简介	2
1.1 并行计算技术简介	2
1.1.1 并行计算的基本概念	2
1.1.2 并行计算技术的分类	6
1.1.3 并行计算的主要技术问题	10
1.2 大数据处理技术简介	13
1.2.1 大数据的发展背景和研究意义	13
1.2.2 大数据的技术特点	16
1.2.3 大数据研究的主要目标、基本原则和基本途径	17
1.2.4 大数据计算模式和系统	18
1.2.5 大数据计算模式的发展趋势	21
1.2.6 大数据的主要技术层面和技术内容	22
1.3 MapReduce 并行计算技术简介	25
1.3.1 MapReduce 的基本概念和由来	25
1.3.2 MapReduce 的基本设计思想	26

1.3.3 MapReduce 的主要功能和技术特征	28
1.4 Hadoop 系统简介	30
1.4.1 Hadoop 的概述与发展历史	30
1.4.2 Hadoop 系统分布式存储与并行计算构架	31
1.4.3 Hadoop 平台的基本组成与生态系统	33
1.4.4 Hadoop 的应用现状和发展趋势	37
<b>第 2 章 Hadoop 系统的安装与操作管理</b>	<b>39</b>
2.1 Hadoop 系统安装方法简介	39
2.2 单机和单机伪分布式 Hadoop 系统安装基本步骤	39
2.2.1 安装和配置 JDK	40
2.2.2 创建 Hadoop 用户	40
2.2.3 下载安装 Hadoop	40
2.2.4 配置 SSH	41
2.2.5 配置 Hadoop 环境	42
2.2.6 Hadoop 的运行	43
2.2.7 运行测试程序	43
2.2.8 查看集群状态	44
2.3 集群分布式 Hadoop 系统安装基本步骤	44
2.3.1 安装和配置 JDK	44
2.3.2 创建 Hadoop 用户	45
2.3.3 下载安装 Hadoop	45
2.3.4 配置 SSH	45
2.3.5 配置 Hadoop 环境	46
2.3.6 Hadoop 的运行	48
2.3.7 运行测试程序	48
2.3.8 查看集群状态	49
2.4 Hadoop MapReduce 程序开发过程	49
2.5 集群远程作业提交与执行	53
2.5.1 集群远程作业提交和执行过程	53
2.5.2 查看作业执行结果和集群状态	53
<b>第 3 章 大数据存储——分布式文件系统 HDFS</b>	<b>56</b>
3.1 HDFS 的基本特征与构架	56

3.1.1	HDFS 的基本特征	57
3.1.2	HDFS 的基本框架与工作过程	57
3.2	HDFS 可靠性设计	60
3.2.1	HDFS 数据块多副本存储设计	60
3.2.2	HDFS 可靠性的设计实现	61
3.3	HDFS 文件存储组织与读写	63
3.3.1	文件数据的存储组织	63
3.3.2	数据的读写过程	65
3.4	HDFS 文件系统操作命令	68
3.4.1	HDFS 启动与关闭	68
3.4.2	HDFS 文件操作命令格式与注意事项	69
3.4.3	HDFS 文件操作命令	69
3.4.4	高级操作命令和工具	77
3.5	HDFS 基本编程接口与示例	83
3.5.1	HDFS 编程基础知识	83
3.5.2	HDFS 基本文件操作 API	84
3.5.3	HDFS 基本编程实例	87
<b>第 4 章</b>	<b>Hadoop MapReduce 并行编程框架</b>	<b>91</b>
4.1	MapReduce 基本编程模型和框架	91
4.1.1	MapReduce 并行编程抽象模型	91
4.1.2	MapReduce 的完整编程模型和框架	93
4.2	Hadoop MapReduce 基本构架与工作过程	96
4.2.1	Hadoop 系统构架和 MapReduce 程序执行过程	96
4.2.2	Hadoop MapReduce 执行框架和作业执行流程	98
4.2.3	Hadoop MapReduce 作业调度过程和调度方法	102
4.2.4	MapReduce 执行框架的组件和执行流程	106
4.3	Hadoop MapReduce 主要组件与编程接口	107
4.3.1	数据输入格式 InputFormat	107
4.3.2	输入数据分块 InputSplit	109
4.3.3	数据记录读入 RecordReader	110
4.3.4	Mapper 类	112
4.3.5	Combiner	114



4.3.6	Partitioner	115
4.3.7	Sort	116
4.3.8	Reducer 类	119
4.3.9	数据输出格式 OutputFormat	120
4.3.10	数据记录输出 RecordWriter	122
<b>第 5 章 分布式数据库 HBase</b>		<b>123</b>
5.1	HBase 简介	123
5.1.1	为什么需要 NoSQL 数据库	123
5.1.2	HBase 的作用和功能特点	125
5.2	HBase 的数据模型	126
5.2.1	HBase 的基本数据模型	126
5.2.2	HBase 的查询模式	128
5.2.3	HBase 表设计	129
5.3	HBase 的基本构架与数据存储管理方法	132
5.3.1	HBase 在 Hadoop 生态中的位置和关系	132
5.3.2	HBase 的基本组成结构	133
5.3.3	HBase Region	133
5.3.4	Region Server	135
5.3.5	HBase 的总体组成结构	138
5.3.6	HBase 的寻址和定位	139
5.3.7	HBase 节点的上下线管理	142
5.4	HBase 安装与操作	145
5.4.1	安装一个单机版的 HBase	145
5.4.2	HBase Shell 操作命令	146
5.4.3	基于集群的 HBase 安装和配置	149
5.5	HBase 的编程接口和编程示例	152
5.5.1	表创建编程接口与示例	152
5.5.2	表数据更新编程接口与示例	153
5.5.3	数据读取编程接口与示例	155
5.5.4	HBase MapReduce 支持和编程示例	157
5.6	HBase 的读写操作和特性	161
5.6.1	HBase 的数据写入	161

5.6.2	HBase 的数据读取	171
5.7	其他 HBase 功能	173
5.7.1	Coprocessor	173
5.7.2	批量数据导入 Bulk Load	176
<b>第 6 章</b>	<b>分布式数据仓库 Hive</b>	<b>179</b>
6.1	Hive 的作用与结构组成	179
6.2	Hive 的数据模型	181
6.2.1	Hive 的数据存储模型	181
6.2.2	Hive 的元数据存储管理	182
6.2.3	Hive 的数据类型	183
6.3	Hive 的安装	184
6.3.1	下载 Hive 安装包	184
6.3.2	配置环境变量	184
6.3.3	创建 Hive 数据文件目录	185
6.3.4	修改 Hive 配置文件	185
6.4	Hive 查询语言——HiveQL	188
6.4.1	DDL 语句	188
6.4.2	DML 语句	189
6.4.3	SELECT 查询语句	190
6.4.4	数据表操作语句示例	190
6.4.5	分区的使用	192
6.4.6	桶的使用	193
6.4.7	子查询	194
6.4.8	Hive 的优化和高级功能	194
6.5	Hive JDBC 编程接口与程序设计	196
<b>第 7 章</b>	<b>Intel Hadoop 系统优化与功能增强</b>	<b>200</b>
7.1	Intel Hadoop 系统简介	200
7.1.1	Intel Hadoop 系统的主要优化和增强功能	200
7.1.2	Intel Hadoop 的系统构成与组件	201
7.2	Intel Hadoop 系统的安装和管理	202
7.3	Intel Hadoop HDFS 的优化和功能扩展	202

7.3.1	HDFS 的高可用性	203
7.3.2	Intel Hadoop 系统高可用性配置服务	204
7.3.3	Intel Hadoop 系统高可用性配置服务操作	206
7.3.4	自适应数据块副本调整策略	208
7.4	Intel Hadoop HBase 的功能扩展和编程示例	211
7.4.1	HBase 大对象存储 (LOB)	211
7.4.2	加盐表	212
7.4.3	HBase 跨数据中心大表	213
7.5	Intel Hadoop Hive 的功能扩展和编程示例	216
7.5.1	开源 Hive 的不足	216
7.5.2	Intel Hadoop “Hive over HBase” 优化设计	216
7.5.3	Hive over HBase 的架构	216

## 第二部分 MapReduce 的编程和算法设计

第 8 章	MapReduce 基础算法程序设计	220
8.1	WordCount	220
8.1.1	WordCount 算法编程实现	220
8.2	矩阵乘法	223
8.2.1	矩阵乘法原理和实现思路	223
8.2.2	矩阵乘法的 MapReduce 程序实现	224
8.3	关系代数运算	227
8.3.1	选择操作	227
8.3.2	投影操作	228
8.3.3	交运算	229
8.3.4	差运算	230
8.3.5	自然连接	231
8.4	单词共现算法	233
8.4.1	单词共现算法的基本设计	233
8.4.2	单词共现算法的实现	234
8.4.3	单词共现算法实现中的细节问题	235
8.5	文档倒排索引	237

8.5.1	简单的文档倒排索引	237
8.5.2	带词频等属性的文档倒排索引	239
8.6	PageRank 网页排名算法	242
8.6.1	PageRank 的简化模型	243
8.6.2	PageRank 的随机浏览模型	244
8.6.3	PageRank 的 MapReduce 实现	245
8.7	专利文献分析算法	249
8.7.1	构建专利被引用列表	250
8.7.2	专利被引用次数统计	251
8.7.3	专利被引用次数直方图统计	252
8.7.4	按照年份或国家统计专利数	254
<b>第 9 章</b>	<b>MapReduce 高级程序设计技术</b>	<b>256</b>
9.1	简介	256
9.2	复合键值对的使用	257
9.2.1	把小的键值对合并成大的键值对	257
9.2.2	巧用复合键让系统完成排序	259
9.3	用户定制数据类型	262
9.3.1	Hadoop 内置的数据类型	263
9.3.2	用户自定义数据类型的实现	263
9.4	用户定制数据输入输出格式	264
9.4.1	Hadoop 内置的数据输入格式与 RecordReader	265
9.4.2	用户定制数据输入格式与 RecordReader	265
9.4.3	Hadoop 内置的数据输出格式与 RecordWriter	269
9.4.4	用户定制数据输出格式与 RecordWriter	269
9.4.5	通过定制数据输出格式实现多集合文件输出	270
9.5	用户定制 Partitioner 和 Combiner	271
9.5.1	用户定制 Partitioner	272
9.5.2	用户定制 Combiner	273
9.6	组合式 MapReduce 计算作业	274
9.6.1	迭代 MapReduce 计算任务	274
9.6.2	顺序组合式 MapReduce 作业的执行	275
9.6.3	具有复杂依赖关系的组合式 MapReduce 作业的执行	275

9.6.4	MapReduce 前处理和后处理步骤的链式执行	276
9.7	多数据源的连接	278
9.7.1	基本问题数据示例	279
9.7.2	用 DataJoin 类实现 Reduce 端连接	279
9.7.3	用全局文件复制方法实现 Map 端连接	285
9.7.4	带 Map 端过滤的 Reduce 端连接	287
9.7.5	多数据源连接解决方法的限制	288
9.8	全局参数 / 数据文件的传递与使用	288
9.8.1	全局作业参数的传递	288
9.8.2	查询全局的 MapReduce 作业属性	290
9.8.3	全局数据文件的传递	291
9.9	关系数据库的连接与访问	292
9.9.1	从数据库中输入数据	292
9.9.2	向数据库中输出计算结果	292
<b>第 10 章</b>	<b>MapReduce 数据挖掘基础算法</b>	<b>295</b>
10.1	K-Means 聚类算法	295
10.1.1	K-Means 聚类算法简介	295
10.1.2	基于 MapReduce 的 K-Means 算法的设计实现	297
10.2	KNN 最近邻分类算法	300
10.2.1	KNN 最近邻分类算法简介	300
10.2.2	基于 MapReduce 的 KNN 算法的设计实现	301
10.3	朴素贝叶斯分类算法	303
10.3.1	朴素贝叶斯分类算法简介	303
10.3.2	朴素贝叶斯分类并行化算法的设计	304
10.3.3	朴素贝叶斯分类并行化算法的实现	306
10.4	决策树分类算法	310
10.4.1	决策树分类算法简介	310
10.4.2	决策树并行化算法的设计	313
10.4.3	决策树并行化算法的实现	317
10.5	频繁项集挖掘算法	327
10.5.1	频繁项集挖掘问题描述	327
10.5.2	Apriori 频繁项集挖掘算法简介	328

10.5.3	Apriori 频繁项集挖掘并行化算法的设计	329
10.5.4	Apriori 频繁项集挖掘并行化算法的实现	331
10.5.5	基于子集求取的频繁项集挖掘算法的设计	335
10.5.6	基于子集求取的频繁项集挖掘并行化算法的实现	336
10.6	隐马尔科夫模型和最大期望算法	340
10.6.1	隐马尔科夫模型的基本描述	340
10.6.2	隐马尔科夫模型问题的解决方法	341
10.6.3	最大期望算法概述	345
10.6.4	并行化隐马尔科夫算法设计	345
10.6.5	隐马尔科夫算法的并行化实现	348
<b>第 11 章 大数据处理算法设计与应用编程案例</b>		<b>352</b>
11.1	基于 MapReduce 的搜索引擎算法	352
11.1.1	搜索引擎工作原理简介	353
11.1.2	基于 MapReduce 的文档预处理	354
11.1.3	基于 MapReduce 的文档倒排索引构建	356
11.1.4	建立 Web 信息查询服务	363
11.2	基于 MapReduce 的大规模短文本多分类算法	365
11.2.1	短文本多分类算法工作原理简介	365
11.2.2	并行化分类训练算法设计实现	366
11.2.3	并行化分类预测算法设计实现	369
11.3	基于 MapReduce 的大规模基因序列比对算法	371
11.3.1	基因序列比对算法简介	371
11.3.2	并行化 BLAST 算法的设计与实现	373
11.4	基于 MapReduce 的大规模城市路径规划算法	379
11.4.1	问题背景和要求	379
11.4.2	数据输入	380
11.4.3	程序设计要求	384
11.4.4	算法设计总体框架和处理过程	385
11.4.5	并行化算法的设计与实现	386
11.5	基于 MapReduce 的大规模重复文档检测算法	396
11.5.1	重复文档检测问题描述	396
11.5.2	重复文档检测方法和算法设计	397

11.5.3	重复文档检测并行化算法设计实现	401
11.6	基于内容的并行化图像检索算法与引擎	404
11.6.1	基于内容的图像检索问题概述	404
11.6.2	图像检索方法和算法设计思路	405
11.6.3	并行化图像检索算法实现	407
11.7	基于 MapReduce 的大规模微博传播分析	412
11.7.1	微博分析问题背景与并行化处理过程	413
11.7.2	并行化微博数据获取算法的设计实现	414
11.7.3	并行化微博数据分析算法的设计实现	416
11.8	基于关联规则挖掘的图书推荐算法	422
11.8.1	图书推荐和关联规则挖掘简介	422
11.8.2	图书频繁项集挖掘算法设计与数据获取	423
11.8.3	图书关联规则挖掘并行化算法实现	425
11.9	基于 Hadoop 的城市智能交通综合应用案例	432
11.9.1	应用案例概述	432
11.9.2	案例一：交通事件检测	433
11.9.3	案例二：交通流统计分析功能	435
11.9.4	案例三：道路旅行时间分析	435
11.9.5	案例四：HBase 实时查询	436
11.9.6	案例五：HBase Endpoint 快速统计	437
11.9.7	案例六：利用 Hive 高速统计	439

## 附 录

附录 A	OpenMP 并行程序设计简介	442
附录 B	MPI 并行程序设计简介	448
附录 C	英特尔 Apache Hadoop* 系统安装手册	457
参考文献		486

第一部分

# Hadoop 系统



- 第 1 章 大数据处理技术简介
  - 第 2 章 Hadoop 系统的安装与操作管理
  - 第 3 章 大数据存储——分布式文件系统 HDFS
  - 第 4 章 Hadoop MapReduce 并行编程框架
  - 第 5 章 分布式数据库 HBase
  - 第 6 章 分布式数据仓库 Hive
  - 第 7 章 Intel Hadoop 系统优化与功能增强
-



# 大数据处理技术简介

近年来，大数据技术在全世界迅猛发展，引起了全世界的广泛关注，掀起了一个全球性的发展浪潮。大数据技术发展的主要推动力来自并行计算硬件和软件技术的发展，以及近年来行业大数据处理需求的迅猛增长。其中，大数据处理技术最直接的推动因素，当数 Google 公司发明的 MapReduce 大规模数据分布存储和并行计算技术，以及 Apache 社区推出的开源 Hadoop MapReduce 并行计算系统的普及使用。为此，本书将重点介绍目前成为大数据处理主流技术和平台 Hadoop MapReduce 并行处理和编程技术。

本章将简要介绍大数据处理相关的基本概念、技术及发展状况。大数据处理的核心技术是分布存储和并行计算，因此，本章首先简要介绍并行计算的基本概念和技术；在此基础上，将简要介绍 MapReduce 的基本概念、功能和技术特点；最后本章将进一步简要介绍开源 Hadoop 系统的基本功能特点和组成。

## 1.1 并行计算技术简介

### 1.1.1 并行计算的基本概念

随着信息技术的快速发展，人们对计算系统的计算能力和数据处理能力的要求日益提高。随着计算问题规模和数据量的不断增大，人们发现，以传统的串行计算方式越来越难以满足实际应用问题对计算能力和计算速度的需求，为此出现了并行计算技术。

并行计算（Parallel Computing）是指同时对多条指令、多个任务或多个数据进行处理的一种计算技术。实现这种计算方式的计算系统称为并行计算系统，它由一组处理单元组成，

这组处理单元通过相互之间的通信与协作，以并行化的方式共同完成复杂的计算任务。实现并行计算的主要目的是，以并行化的计算方法，实现计算速度和计算能力的大幅提升，以解决传统的串行计算所难以完成的计算任务。

现代计算机的发展历程可分为两个明显不同的发展时代：串行计算时代和并行计算时代。并行计算技术是在单处理器计算能力面临发展瓶颈、无法继续取得突破后，才开始走上了快速发展的通道。并行计算时代的到来，使得计算技术获得了突破性的发展，大大提升了计算能力和计算规模。

### 1. 单处理器计算性能提升达到极限

纵观计算机的发展历史，日益提升计算性能是计算技术不断追求的目标和计算技术发展的主要特征之一。自计算机出现以来，提升单处理器计算机系统计算速度的常用技术手段有以下几个方面。

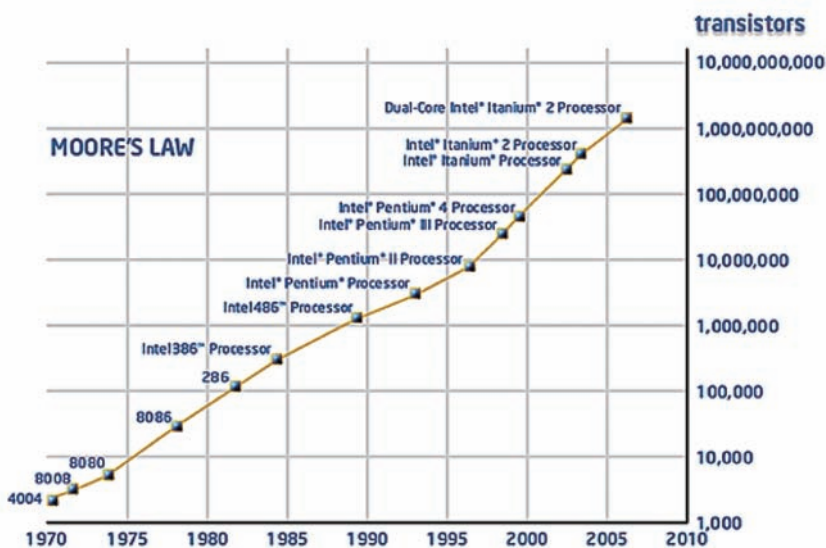


图 1-1 摩尔定律

1) 提升计算机处理器字长。随着计算机技术的发展，单处理器字长也在不断提升，从最初的 4 位发展到如今的 64 位。处理器字长提升的每个发展阶段均有代表性的处理器产品，如 20 世纪 70 年代出现的最早的 4 位 Intel 微处理器 4004，到同时代以 Intel 8008 为代表的 8 位处理器，以及 20 世纪 80 年代 Intel 推出的 16 位字长 80286 处理器，以及后期发展出的 Intel 80386/486/Pentium 系列为主的 32 位处理器等。2000 年以后发展至今，出现了 64 位字长的处理器。目前，32 位和 64 位处理器是市场上主流的处理器的。计算机处理器字长的发展

大幅提升了处理器性能，推动了单处理器计算机的发展。

2) 提高处理器芯片集成度。1965 年，戈登·摩尔 (Gordon Moore) 发现了这样一条规律：半导体厂商能够集成在芯片中的晶体管数量大约每 18 ~ 24 个月翻一番，其计算性能也随着翻一番，这就是众所周知的摩尔定律。在计算技术发展的几十年中，摩尔定律一直引导着计算机产业的发展。

3) 提升处理器的主频。计算机的主频越高，指令执行的时间则越短，计算性能自然会相应提高。因此，在 2004 年以前，处理器设计者一直追求不断提升处理器的主频。计算机主频从 Pentium 开始的 60MHz，曾经最高可达到 4GHz ~ 5GHz。

4) 改进处理器微架构。计算机微处理器架构的改进对于计算性能的提升具有重大的作用。例如，为了使处理器资源得到最充分利用，计算机体系结构设计师引入了指令集并行技术 (Instruction-level Parallelism, ILP)，这是单处理器并行计算的杰出设计思想之一。实现指令级并行最主要的体系结构技术就是流水线技术 (Pipeline)。

在 2004 年以前，以上这些技术极大地提高了微处理器的计算性能，但此后处理器的性能不再像人们预期的那样能够继续提高。人们发现，随着集成度的不断提高以及处理器频率的不断提升，单核处理器的性能提升开始接近极限。首先，芯片的集成度会受到半导体器件制造工艺的限制。目前集成电路已经达到十多个纳米的极小尺度，因此，芯片集成度不可能无限制提高。与此同时，根据芯片的功耗公式  $P=CV^2f$  (其中， $P$  是功耗； $C$  是时钟跳变时门电路电容，与集成度成正比； $V$  是电压， $f$  是主频)，芯片的功耗与集成度和主频成正比，芯片集成度和主频的大幅提高导致了功耗的快速增大，进一步导致了难以克服的处理器散热问题。而流水线体系结构技术也已经发展到了极致，2001 年推出的 Pentium4 (CISC 结构) 已采用了 20 级复杂流水线技术，因此，流水线为主的微体系结构技术也难以有更大提升的空间。

由图 1-2 可以看出，从 2004 年以后，微处理器的主频和计算性能变化逐步趋于平缓，不再随着集成度的提高而提高。如图 1-3a 所示，在 2005 年以前，人们预期可以一直提升处理器主频。但 2004 年 5 月 Intel 处理器 Tejas 和 Jayhawk (4GHz) 因无法解决散热问题最终放弃，标志着升频技术时代的终结。因此，随后人们修改了 2005 年后微处理器主频提升路线图，基本上以较小的幅度提升处理器主频，而代之以多核实现性能提升，如图 1-3b 所示。

## 2. 多核计算技术成为必然发展趋势

2005 年，Intel 公司宣布了微处理器技术的重大战略调整，即从 2005 年开始，放弃过去不断追求单处理器计算性能提升的战略，转向以多核微处理器架构实现计算性能提升。自此

Intel 推出了多核 / 众核构架，微处理器全面转入了多核计算技术时代。多核计算技术的基本思路是：简化单处理器的复杂设计，代之以在单个芯片上设计多个简化的处理器核，以多核 / 众核并行计算提升计算性能。

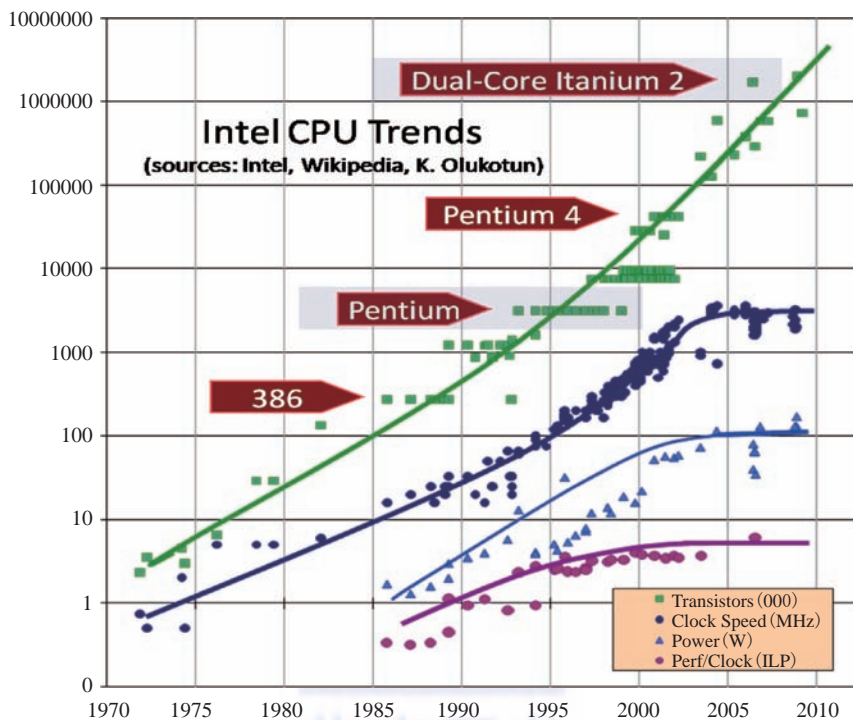


图 1-2 微处理器芯片集成度与主频、功耗、性能的演变趋势图

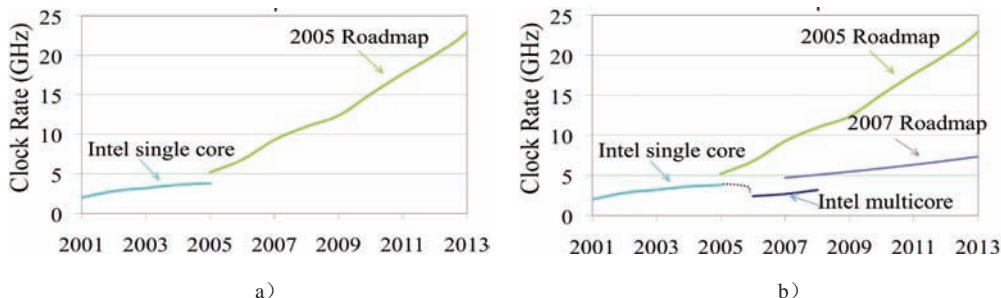


图 1-3 2005 年前后人们预期的主频提升路线图

(注：插图引自 Edward L. Bosworth, The Power Wall, 2010)

自 Intel 在 2006 年推出双核的 Pentium D 处理器以来，已经出现了很多从 4 核到 12 核的多核

处理器产品，如 2007 年 Intel 推出的主要用于个人电脑的 4 核 Core 2 Quad 系列以及 2008 ~ 2010 年推出的 Core i5 和 i7 系列。而 Intel 服务器处理器也陆续推出了 Xeon E5 系列 4-12 核的处理器，以及 Xeon E7 系列 6-10 核处理器。

除了多核处理器产品外，众核处理器也逐步出现。NVIDIA GPU 是一种主要面向图形处理加速的众核处理器，在图形处理领域得到广泛应用。2012 年年底 Intel 公司发布了基于集成众核架构（Intel® MIC Architecture, Intel® Many Integrated Core Architecture）的 Xeon Phi 协处理器，这是一款真正意义上通用性的商用级众核处理器，可支持使用与主机完全一样的通用的 C/C++ 编程方式，用 OpenMP 和 MPI 等并行编程接口完成并行化程序的编写，完成的程序既可在多核主机上运行，也可在众核协处理器上运行。众核计算具有体积小、功耗低、核数多、并行处理能力强等技术特点和优势，将在并行计算领域发挥重要作用。众核处理器的出现进一步推进了并行计算技术的发展，从而使并行计算的性能发挥到极致，更加明确地体现了并行计算技术的发展趋势。

### 3. 大数据时代日益增大的数据规模迫切需要使用并行计算技术

随着计算机和信息技术的不断普及应用，行业应用领域计算系统的规模日益增大，数据规模也急剧增大。

全球著名的互联网企业的数据规模动辄达到数百至数千 PB 量级，而其他的诸如电信、电力、金融、科学计算等典型应用行业和领域，其数据量也高达数百 TB 至数十 PB 的规模。如此巨大的数据量使得传统的计算技术和系统已经无法应对和满足计算需求。巨大的数据量会导致巨大的计算时间开销，使得很多在小规模数据时可以完成的计算任务难以在可接受的时间内完成大规模数据的处理。超大的数据量或计算量，给原有的单处理器和串行计算技术带来巨大挑战，因而迫切需要出现新的技术和手段以应对急剧增长的行业应用需求。

#### 1.1.2 并行计算技术的分类

并行计算技术发展至今，出现了各种不同的技术方法，同时也出现了不同的分类方法，包括按指令和数据处理方式的 Flynn 分类、按存储访问结构的分类、按系统类型的分类、按应用的计算特征的分类、按并行程序设计方式的分类。

##### 1. Flynn 分类法

1966 年，斯坦福大学教授 Michael J. Flynn 提出了经典的计算机结构分类方法，从最抽象的指令和数据处理方式进行分类，通常称为 Flynn 分类。Flynn 分类法是从两种角度进行分类，一是依据计算机在单个时间点能够处理的指令流的数量；二是依据计算机在单个时间



点能够处理的数据流的数量。任何给定的计算机系统均可以依据处理指令和数据的方式进行分类。图 1-4 所示为 Flynn 分类下的几种不同的计算模式。

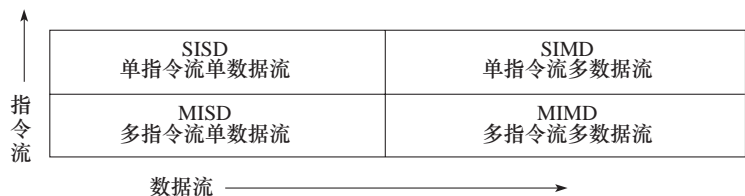


图 1-4 Flynn 分类法

1) 单指令流单数据流 (Single Instruction stream and Single Data stream, SISD)：SISD 是传统串行计算机的处理方式，硬件不支持任何并行方式，所有指令串行执行。在一个时钟周期内，处理器只能处理一个数据流。很多早期计算机均采用这种处理方式，例如最初的 IBM PC 机。

2) 单指令流多数据流 (Single Instruction stream and Multiple Data stream, SIMD)：SIMD 采用一个指令流同时处理多个数据流。最初的阵列处理机或者向量处理机都具备这种处理能力。计算机发展至今，几乎所有计算机都以各种指令集形式实现 SIMD。较为常用的有，Intel 处理器中实现的 MMXTM、SSE (Streaming SIMD Extensions)、SSE2、SSE3、SSE4 以及 AVX (Advanced Vector Extensions) 等向量指令集。这些指令集都能够在单个时钟周期内处理多个存储在寄存器中的数据单元。SIMD 在数字信号处理、图像处理、多媒体信息处理以及各种科学计算领域有较多的应用。

3) 多指令流单数据流 (Multiple Instruction stream and Single Data stream, MISD)：MISD 采用多个指令流处理单个数据流。这种方式实际很少出现，一般只作为一种理论模型，并没有投入到实际生产和应用中。

4) 多指令流多数据流 (Multiple Instruction Stream and Multiple Data Stream, MIMD)：MIMD 能够同时执行多个指令流，这些指令流分别对不同数据流进行处理。这是目前最流行的并行计算处理方式。目前较常用的多核处理器以及 Intel 最新推出的众核处理器都属于 MIMD 的并行计算模式。

2. 按存储访问结构分类

按存储访问结构，可将并行计算分为以下几类：

1) 共享内存访问结构 (Shared Memory Access)：即所有处理器通过总线共享内存的多核处理器，也称为 UMA 结构 (Uniform Memory Access，一致性内存访问结构)。SMP (Symmetric Multi-Processing，对称多处理器系统) 即为典型的内存共享式的多核处理器构架。图 1-5 即为共享内存访问结构示意图。

2) 分布式内存访问结构 (Distributed Memory Access): 图 1-6 所示为分布式内存访问结构的示意图, 其中, 各个分布式处理器使用本地独立的存储器。

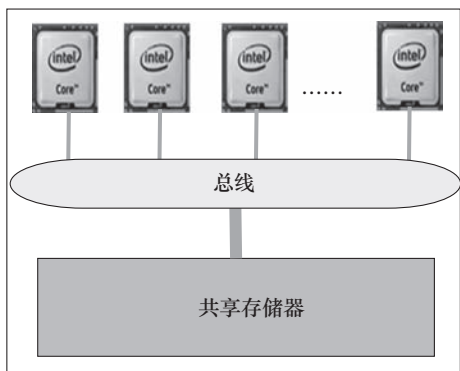


图 1-5 共享内存访问结构

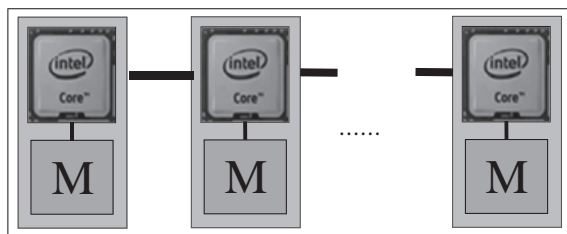


图 1-6 分布式内存访问结构

3) 分布共享式内存访问结构 (Distributed and Shared Memory Access): 是一种混合式的内存访问结构。如图 1-7 所示, 各个处理器分别拥有独立的本地存储器, 同时, 再共享访问一个全局的存储器。

分布式内存访问结构和分布共享式内存访问结构也称为 NUMA 结构 (Non-Uniform Memory Access, 非一致内存访问结构)。在多核情况下, 这种内存访问架构可以充分扩展内存带宽, 减少内存冲突开销, 提高系统扩展性和计算性能。

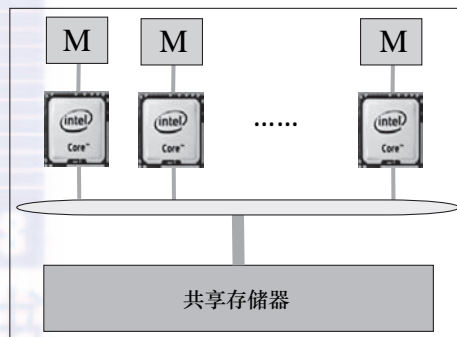


图 1-7 分布共享式内存访问结构

### 3. 按系统类型分类

按并行计算系统类型, 可将并行计算分为以下类型:

1) 多核 / 众核并行计算系统 (Multi-Core/Many-Core) 或芯片级多处理系统 (Chip-level Multiprocessing, CMP)。

2) 对称多处理系统 (Symmetric Multi Processing, SMP), 即多个相同类型处理器通过总线连接、并共享存储器构成的一种并行计算系统。

3) 大规模并行处理系统 (Massive Parallel Processing, MPP), 以专用内联网连接一组处理器形成的一种并行计算系统。

4) 集群 (Cluster), 以网络连接的一组普通商用计算机构成的并行计算系统。

5) 网格 (Grid), 用网络连接远距离分布的一组异构计算机构成的并行计算系统。

#### 4. 按应用的计算特征分类

按应用的计算特征，可将并行计算分为以下类型：

1) 数据密集型并行计算 (Data-Intensive Parallel Computing)，即数据量极大、但计算相对简单的并行计算。

2) 计算密集型并行计算 (Computation-Intensive Parallel Computing)，即数据量相对不大、但计算较为复杂的并行处理。较为传统的高性能计算领域大部分都是这一类型，例如三维建模与渲染、气象预报、生命科学等科学计算。

3) 数据密集与计算密集混合型并行计算，具备数据密集和计算密集双重特征的并行计算，如 3D 电影渲染等。

#### 5. 按并行程序设计方式分类

按并行程序设计方式，可将并行计算分为以下几类：

1) 共享存储变量方式 (Shared Memory Variables)：这种方式通常被称为多线程并行程序设计。多线程并行方式发展至今，应用非常广泛，同时也出现了很多代表性的并行编程接口，包括开源的和一些商业版本的并行编程接口，例如最常用的有 pthread, OpenMP, Intel TBB 等。其中，pthread 是较为低层的多线程编程接口；而 OpenMP 采用了语言扩充的方法，简单易用，不需要修改代码，仅需添加指导性语句，应用较为广泛；而 Intel TBB 是一种很适合用 C++ 代码编程的并行程序设计方法，提供了很多方便易用的并行编程接口。本书的附录 A 将较为详细地介绍 OpenMP 的技术特点和编程方法。共享存储变量方式可能引起数据的不一致性，从而导致数据和资源访问冲突，因此一般都需要引入同步控制机制。

2) 消息传递方式 (Message Passing)：从广义上来讲，对于分布式内存访问结构的系统，为了分发数据实现并行计算、随后收集计算结果，需要在各个计算节点或者计算任务间进行数据通信。这种编程方式有时候可狭义地理解为多进程处理方式。最常用的消息传递方式是 MPI (Message Passing Interface, 消息传递并行编程接口标准)。MPI 广泛应用于科学计算的各个领域，并体现了其高度的可扩展性，能充分利用并行计算系统的硬件资源，发挥其计算性能。具体有关 MPI 技术的介绍，请阅读本书的附录 B：MPI 并行程序设计简介。

3) MapReduce 并行程序设计方式：Google 公司提出的 MapReduce 并行程序设计模型，是目前主流的大数据处理并行程序设计方法，可广泛应用于各个领域的大数据处理，尤其是搜索引擎等互联网行业的大规模数据处理。本书后续重点探讨的数据处理技术正是基于这种方式的并行程序设计技术。

4) 其他新型并行计算和编程方式：由于 MapReduce 设计之初主要致力于大数据的线下批处理，因而其难以满足高实时性和高数据相关性的大数据处理需求。为此，近年来，逐步出现了多种其他类型的大数据计算模式和方法。这些新型计算模式和方法包括：实时流式计



算、迭代计算、图计算以及基于内存的计算等。

### 1.1.3 并行计算的主要技术问题

依赖于所采用的并行计算体系结构，不同类型的并行计算系统在硬件构架、软件构架和并行算法方面会涉及到不同的技术问题，但概括起来，主要包括以下技术问题。

#### 1. 多处理器 / 多节点网络互连技术

对于大型的并行处理系统，网络互连技术对处理器能力影响很大。典型的网络互连结构包括共享总线连接、交叉开关矩阵、环形结构、Mesh 网络结构、互联网络结构等。

#### 2. 存储访问体系结构

存储访问体系结构主要研究不同的存储结构以及在不同存储结构下的特定技术问题，包括共享数据访问与同步控制、数据通信控制和节点计算同步控制、Cache 的一致性、数据访问 / 通信的时间延迟等技术问题。

#### 3. 分布式数据与文件管理

并行计算的一个重要问题是，在大规模集群环境下，如何解决大规模数据的存储和访问管理问题。在大规模集群环境下，解决大数据分布存储管理和访问问题非常关键，尤其是数据密集型并行计算，数据的存储访问对并行计算的性能至关重要。目前比较理想的解决方法是提供分布式数据和文件管理系统，代表性的系统有 Google GFS (Google File System)、Lustre、HDFS (Hadoop Distributed File System) 等。这些分布式文件系统各有特色，适用于不同领域。

#### 4. 并行计算的任务划分和算法设计

并行计算的任务分解和算法设计需要考虑的是如何将大的计算任务分解成子任务，继而分配给各节点或处理器并行处理，最终收集局部结果进行整合。一般有算法分解和数据划分两种并行计算形式，尤其是算法分解，可有多种不同的实现方式。

#### 5. 并行程序设计模型和语言

根据不同的硬件构架，不同的并行计算系统可能需要不同的并行程序设计模型、方法和语言。目前主要的并行程序设计语言和方法包括共享内存式并行程序设计、消息传递式并行程序设计、MapReduce 并行程序设计以及近年来出现的满足不同大数据处理需求的其他并行计算和程序设计方法。而并行程序设计语言通常可以有不同的实现方式，包括：语言级扩充（即使用编译指令在普通的程序设计语言中增加一些并行化编译指令，如 OpenMP 提供

C、C++、Fortran 语言扩充)、并行计算库函数与编程接口(使用函数库提供并行计算编程接口,如 MPI、CUDA 等)以及能提供诸多自动化处理能力的并行计算软件框架(如 Hadoop MapReduce 并行计算框架等)。

## 6. 并行计算软件框架设计和实现

现有的 OpenMP、MPI、CUDA 等并程序序设计方法需要程序员考虑数据存储管理、数据和任务划分、任务的调度执行、数据同步和通信、结果收集、出错恢复处理等几乎所有技术细节,非常繁琐。为了进一步提升并行计算程序的自动化并行处理能力,编程时应该尽量减少程序员对很多系统底层技术细节的考虑,使得编程人员能从底层细节中解放出来,更专注于应用问题本身的计算和算法实现。目前已发展出多种具有自动化并行处理能力的计算软件框架,如 Google MapReduce 和 Hadoop MapReduce 并行计算软件框架,以及近年来出现的以内存计算为基础、能提供多种大数据计算模式的 Spark 系统等。

## 7. 数据访问和通信控制

并行计算目前存在多种存储访问体系结构,包括共享存储访问结构、分布式存储访问结构以及分布共享式存储访问结构。不同存储访问结构下需要考虑不同的数据访问、节点通信以及同步控制等问题。例如,在共享存储访问结构系统中,多个处理器访问共享存储区,可能导致数据访问的不确定性,从而需要引入互斥信号、条件变量等同步机制,保证共享数据访问的正确性,同时需解决可能引起的死锁问题。而对于分布式存储访问结构系统,数据可能需要通过主节点传输到其他计算节点,由于节点间的计算速度不同,为了保证计算的同步,需要考虑计算的同步问题。

## 8. 可靠性与容错性技术

对于大型的并行计算系统,经常发生节点出错或失效。因此,需要考虑和预防由于一个节点失效可能导致的数据丢失、程序终止甚至系统崩溃的问题。这就要求系统考虑良好的可靠性设计和失效检测恢复技术。通常可从两方面进行可靠性设计:一是数据失效恢复,可使用数据备份和恢复机制,当某个磁盘出错或数据损毁时,保证数据不丢失以及数据的正确性;二是系统和任务失效恢复,当某个节点失效时,需要提供良好的失效检测和隔离技术,以保证并行计算任务正常进行。

## 9. 并行计算性能分析与评估

并行计算的性能评估较为常用的方式是通过加速比来体现性能提升。加速比指的是并行程序的并行执行速度相对于其串行程序执行加速了多少倍。这个指标贯穿于整个并行计算技

术，是并行计算技术的核心。从应用角度出发，不论是开发还是使用，都希望一个并行计算程序能达到理想的加速比，即随着处理能力的提升，并行计算程序的执行速度也需要有相应的提升。并行计算性能的度量有以下两个著名的定律：

1) Amdal 定律：在一定的程序可并行化比例下，加速比不能随着处理器数目的增加而无限上升，而是受限于程序的串行化部分的比例，加速比极限是串行比例的倒数，反映了固定负载的加速情况。Amdal 定律的公式是：

$$S = \frac{1}{(1 - P) + \frac{P}{N}}$$

其中， $S$  是加速比， $P$  是程序可并行部分的比例， $N$  是处理器数量。图 1-8 所示是在程序不同的可并行化比例下、在不同处理器数量下的加速比，由图示结果可见，在固定的程序可并行化比例下，加速比提升会有一个上限，处理器数量的增加并不能无限制地带来性能提升。

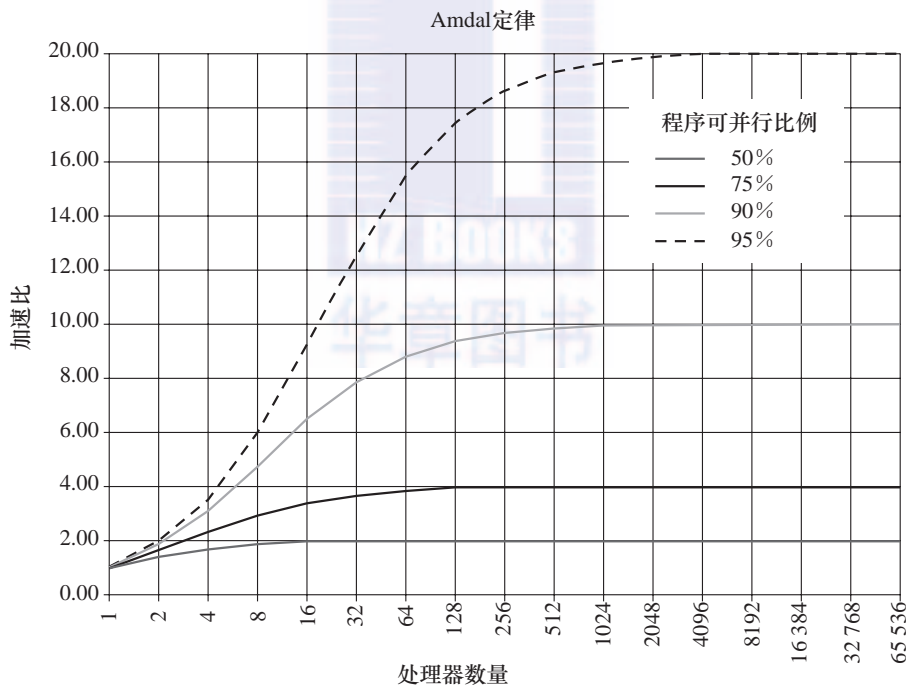


图 1-8 Amdal 定律在程序不同的可并行化比例和不同处理器数量下的加速比

2) Gustafson 定律：在放大系统规模的情况下，加速比可与处理器数量成比例地线性增长，串行比例不再是加速比的瓶颈。这反映了对于增大的计算负载，当系统性能未达到期望值时，可通过增加处理器数量的方法应对（如图 1-9 和 1-10 所示）。

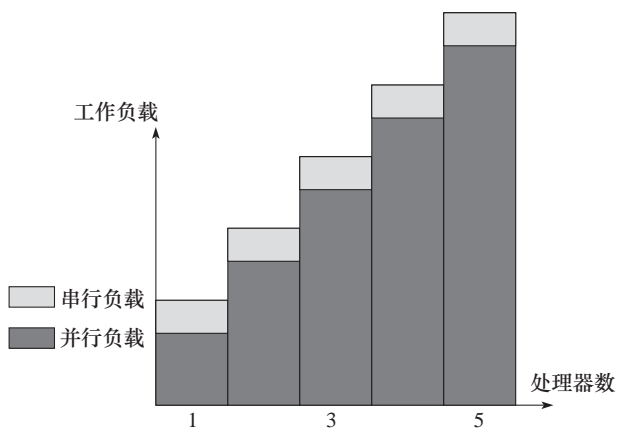


图 1-9 Gustafson 定律：处理器与工作负载

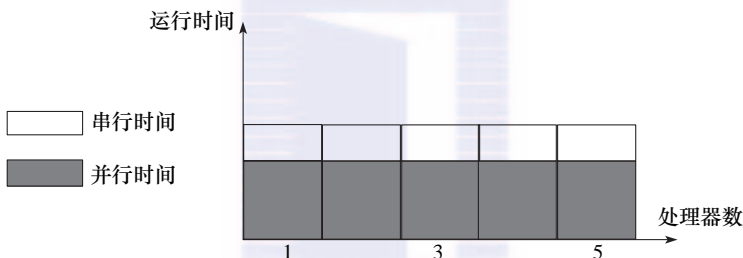


图 1-10 Gustafson 定律：处理器与运行时间

## 1.2 大数据处理技术简介

### 1.2.1 大数据的发展背景和研究意义

近几年来，随着计算机和信息技术的迅猛发展和普及应用，行业应用系统的规模迅速扩大，行业应用所产生的数据呈爆炸性增长。动辄达到数百 TB 甚至数十至数百 PB 规模的行业 / 企业大数据已远远超出了现有传统的计算技术和信息系统的处理能力，因此，寻求有效的大数据处理技术、方法和手段已经成为现实世界的迫切需求。百度目前的总数据量已超过 1000PB，每天需要处理的网页数据达到 10PB ~ 100PB；淘宝累计的交易数据量高达 100PB；Twitter 每天发布超过 2 亿条消息，新浪微博每天发帖量达到 8000 万条；中国移动一个省的电话通联记录数据每月可达 0.5PB ~ 1PB；一个省会城市公安局道路车辆监控数据三年可达 200 亿条、总量 120TB。据世界权威 IT 信息咨询分析公司 IDC 研究报告预测：全世界数据量未来 10 年将从 2009 年的 0.8ZB 增长到 2020 年的 35ZB（1ZB=1000EB=1000000PB），10 年将增长 44 倍，年均增长 40%。

早几年人们把大规模数据称为“海量数据”，但实际上，大数据（Big Data）这个概念早在 2008 年就被提出。2008 年，在 Google 成立 10 周年之际，著名的《自然》杂志出版了一期专刊，专门讨论未来的大数据处理相关的一系列技术问题和挑战，其中就提出了“Big Data”的概念。

随着大数据概念的普及，人们常常会问，多大的数据才叫大数据？其实，关于大数据，难以有一个非常定量的定义。维基百科给出了一个定性的描述：大数据是指无法使用传统和常用的软件技术和工具在一定时间内完成获取、管理和处理的数据集。进一步，当今“大数据”一词的重点其实已经不仅在于数据规模的定义，它更代表着信息技术发展进入了一个新的时代，代表着爆炸性的数据信息给传统的计算技术和信息技术带来的技术挑战和困难，代表着大数据处理所需的新的技术和方法，也代表着大数据分析和应用所带来的新发明、新服务和新的发展机遇。

由于大数据处理需求的迫切性和重要性，近年来大数据技术已经在全球学术界、工业界和各国政府得到高度关注和重视，全球掀起了一个可与 20 世纪 90 年代的信息高速公路相提并论的研究热潮。美国和欧洲一些发达国家政府都从国家科技战略层面提出了一系列的大数据技术研发计划，以推动政府机构、重大行业、学术界和工业界对大数据技术的探索研究和应用。

早在 2010 年 12 月，美国总统办公室下属的科学技术顾问委员会（PCAST）和信息技术顾问委员会（PITAC）向奥巴马和国会提交了一份《规划数字化未来》的战略报告，把大数据收集和使用的提升工作提升到体现国家意志的战略高度。报告列举了 5 个贯穿各个科技领域的共同挑战，而第一个最重大的挑战就是“数据”问题。报告指出：“如何收集、保存、管理、分析、共享正在呈指数增长的数据是我们必须面对的一个重要挑战”。报告建议：“联邦政府的每一个机构和部门，都需要制定一个‘大数据’的战略”。2012 年 3 月，美国总统奥巴马签署并发布了一个“大数据研究发展创新计划”（Big Data R & D Initiative），由美国国家自然基金会（NSF）、卫生健康总署（NIH）、能源部（DOE）、国防部（DOD）等 6 大部门联合，投资 2 亿美元启动大数据技术研发，这是美国政府继 1993 年宣布“信息高速公路”计划后的又一次重大科技发展部署。美国白宫科技政策办公室还专门支持建立了一个大数据技术论坛，鼓励企业和组织机构间的大大数据技术交流与合作。

2012 年 7 月，联合国在纽约发布了一本关于大数据政务的白皮书《大数据促发展：挑战与机遇》，全球大数据的研究和发展进入了前所未有的高潮。这本白皮书总结了各国政府如何利用大数据响应社会需求，指导经济运行，更好地为人民服务，并建议成员国建立“脉搏实验室”（Pulse Labs），挖掘大数据的潜在价值。

由于大数据技术的特点和重要性，目前国内外已经出现了“数据科学”的概念，即数据处理技术将成为一个与计算科学并列的新的科学领域。已故著名图灵奖获得者 Jim Gray 在



2007年的一次演讲中提出,“数据密集型科学发现”(Data-Intensive Scientific Discovery)将成为科学研究的第四范式,科学研究将从实验科学、理论科学、计算科学,发展到目前兴起的数据科学。

为了紧跟全球大数据技术发展的浪潮,我国政府、学术界和工业界对大数据也予以了高度的关注。央视著名“对话”节目2013年4月14日和21日邀请了《大数据时代——生活、工作与思维的大变革》作者维克托·迈尔-舍恩伯格,以及美国大数据存储技术公司LSI总裁阿比分别做客“对话”节目,做了两期大数据专题谈话节目“谁在引爆大数据”、“谁在掘金大数据”,国家央视媒体对大数据的关注和宣传体现了大数据技术已经成为国家和社会普遍关注的焦点。

而国内的学术界和工业界也都迅速行动,广泛开展大数据技术的研究和开发。2013年以来,国家自然科学基金、973计划、核高基、863等重大研究计划都已经把大数据研究列为重大的研究课题。为了推动我国大数据技术的研究发展,2012年中国计算机学会(CCF)发起组织了CCF大数据专家委员会,CCF专家委员会还特别成立了一个“大数据技术发展战略报告”撰写组,并已撰写发布了《2013年中国大数据技术与产业发展白皮书》。

大数据在带来巨大技术挑战的同时,也带来巨大的技术创新与商业机遇。不断积累的大数据包含着很多在小数据量时不具备的深度知识和价值,大数据分析挖掘将能为行业/企业带来巨大的商业价值,实现各种高附加值的增值服务,进一步提升行业/企业的经济效益和社会效益。由于大数据隐含着巨大的深度价值,美国政府认为大数据是“未来的新石油”,对未来的科技与经济发展将带来深远影响。因此,在未来,一个国家拥有数据的规模和运用数据的能力将成为综合国力的重要组成部分,对数据的占有、控制和运用也将成为国家间和企业间新的争夺焦点。

大数据的研究和分析应用具有十分重大的意义和价值。被誉为“大数据时代预言家”的维克托·迈尔-舍恩伯格在其《大数据时代》一书中列举了大量详实的大数据应用案例,并分析预测了大数据的发展现状和未来趋势,提出了很多重要的观点和发展思路。他认为:“大数据开启了一次重大的时代转型”,指出大数据将带来巨大的变革,改变我们的生活、工作和思维方式,改变我们的商业模式,影响我们的经济、政治、科技和社会等各个层面。

由于大数据行业应用需求日益增长,未来越来越多的研究和应用领域将需要使用大数据并行计算技术,大数据技术将渗透到每个涉及到大规模数据和复杂计算的应用领域。不仅如此,以大数据处理为中心的计算技术将对传统计算技术产生革命性的影响,广泛影响计算机体系结构、操作系统、数据库、编译技术、程序设计技术和方法、软件工程技术、多媒体信息处理技术、人工智能以及其他计算机应用技术,并与传统计算技术相互结合产生很多新的研究热点和课题。

大数据给传统的计算技术带来了新的挑战。大数据使得很多在小数据集上有效的传



统的串行化算法在面对大数据处理时难以在可接受的时间内完成计算；同时大数据含有较多噪音、样本稀疏、样本不平衡等特点使得现有的很多机器学习算法有效性降低。因此，微软全球副总裁陆奇博士在 2012 年全国第一届“中国云 / 移动互联网创新大奖赛”颁奖大会主题报告中指出：“大数据使得绝大多数现有的串行化机器学习算法都需要重写”。

大数据技术的发展将给我们研究计算机技术的专业人员带来新的挑战和机遇。目前，国内外 IT 企业对大数据技术人才的需求正快速增长，未来 5 ~ 10 年内业界将需要大量的掌握大数据处理技术的人才。IDC 研究报告指出，“下一个 10 年里，世界范围的服务器数量将增长 10 倍，而企业数据中心管理的数据信息将增长 50 倍，企业数据中心需要处理的数据文件数量将至少增长 75 倍，而世界范围内 IT 专业技术人才的数量仅能增长 1.5 倍。”因此，未来十年里大数据处理和应用需求与能提供的技术人才数量之间将存在一个巨大的差距。目前，由于国内外高校开展大数据技术人才培养的时间不长，技术市场上掌握大数据处理和应用开发技术的人才十分短缺，因而这方面的技术人才十分抢手，供不应求。国内几乎所有著名的 IT 企业，如百度、腾讯、阿里巴巴和淘宝、奇虎 360 等，都大量需要大数据技术人才。

### 1.2.2 大数据的技术特点

大数据具有五个主要的技术特点，人们将其总结为 5V 特征（见图 1-11）：

1) Volume (大体量)：即可从数百 TB 到数十数百 PB、甚至 EB 的规模。

2) Variety (多样性)：即大数据包括各种格式和形态的数据。

3) Velocity (时效性)：即很多大数据需要在一定的时间限度下得到及时处理。

4) Veracity (准确性)：即处理的结果要保证一定的准确性。

5) Value (大价值)：即大数据包含很多深度的价值，大数据分析挖掘和利用将带来巨大的商业价值。

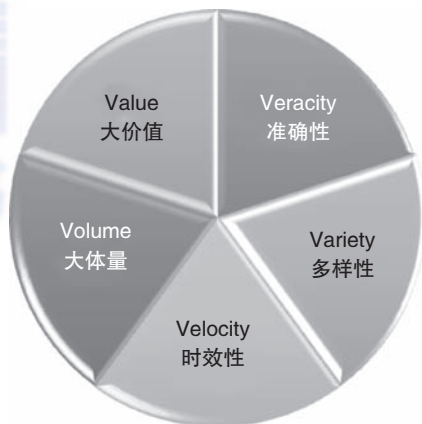


图 1-11 大数据的 5V 特征

传统的数据库系统主要面向结构化数据的存储和处理，但现实世界中的大数据具有各种不同的格式和形态，据统计现实世界中 80% 以上的数据都是文本和媒体等非结构化数据；同时，大数据还具有很多不同的计算特征。我们可以从多个角度分类大数据的类型和计算特征。

- 1) 从数据结构特征角度看，大数据可分为结构化与非结构化 / 半结构化数据。
- 2) 从数据获取处理方式看，大数据可分为批处理与流式计算方式。
- 3) 从数据处理类型看，大数据处理可分为传统的查询分析计算和复杂数据挖掘计算。

4) 从大数据处理响应性能看, 大数据处理可分为实时/准实时与非实时计算, 或者是联机计算与线下计算。前述的流式计算通常属于实时计算, 此外查询分析类计算通常也要求具有高响应性能, 因而也可以归为实时或准实时计算。而批处理计算和复杂数据挖掘计算通常属于非实时或线下计算。

5) 从数据关系角度看, 大数据可分为简单关系数据(如 Web 日志)和复杂关系数据(如社会网络等具有复杂数据关系的图计算)。

6) 从迭代计算角度看, 现实世界的数据处理中有很多计算问题需要大量的迭代计算, 诸如一些机器学习等复杂的计算任务会需要大量的迭代计算, 为此需要提供具有高效的迭代计算能力的大数据处理和计算方法。

7) 从并行计算体系结构特征角度看, 由于需要支持大规模数据的存储和计算, 因此目前绝大多数大数据处理都使用基于集群的分布式存储与并行计算体系结构和硬件平台。MapReduce 是最为成功的分布式存储和并行计算模式。然而, 基于磁盘的数据存储和计算模式使 MapReduce 难以实现高响应性能。为此人们从分布计算体系结构层面上又提出了内存计算的概念和技术方法。

### 1.2.3 大数据研究的主要目标、基本原则和基本途径

#### 1. 大数据研究的主要目标

大数据研究的主要目标是, 以有效的信息技术手段和计算方法, 获取、处理和分析各种应用行业的大数据, 发现和提取数据的深度价值, 为行业提供高附加值的应用和服务。因此, 大数据研究的核心目标是价值发现, 而其技术手段是信息技术和计算方法, 其效益目标是为行业提供高附加值的应用和服务。

#### 2. 大数据研究的基本特点

大数据研究具有以下几方面的主要特点:

- 1) 大数据处理具有很强的行业应用需求特性, 因此大数据技术研究必须紧扣行业应用需求。
- 2) 大数据规模极大, 超过任何传统数据库系统的处理能力。
- 3) 大数据处理技术综合性强, 任何单一层面的计算技术都难以提供理想的解决方案, 需要采用综合性的软硬件技术才能有效处理。
- 4) 大数据处理时, 大多数传统算法都面临失效, 需要重写。

#### 3. 大数据研究的基本原则

大数据研究的基本原则是:

1) 应用需求为导向: 由于大数据问题来自行业应用, 因此大数据的研究需要以行业应用问题和需求为导向, 从行业实际的应用需求和存在的技术难题入手, 研究解决有效的处理技术和解决方案。

2) 领域交叉为桥梁: 由于大数据技术有典型的行业应用特征, 因此大数据技术研究和应用开发需要由计算技术人员、数据分析师、具备专业知识的领域专家相互配合和协同, 促进应用行业、IT 产业与计算技术研究机构的交叉融合, 来提供良好的大数据解决方法。

3) 技术综合为支撑: 与传统的单一层面的计算技术研究和应用不同, 大数据处理是几乎整个计算技术和信息技术的融合, 只有采用技术交叉融合的方法才能提供较为完善的大数据处理方法。

#### 4. 大数据研究的基本途径

大数据处理有以下三个基本的解决途径:

1) 寻找新算法降低计算复杂度。大数据给很多传统的机器学习和数据挖掘计算方法和算法带来挑战。在数据集较小时, 很多在  $O(n)$ 、 $O(n\log n)$ 、 $O(n^2)$  或  $O(n^3)$  等线性或多项式复杂度的机器学习和数据挖掘算法都可以有效工作, 但当数据规模增长到 PB 级尺度时, 这些现有的串行化算法将花费难以接受的时间开销, 使得算法失效。因此, 需要寻找新的复杂度更低的算法。

2) 寻找和采用降低数据尺度的算法。在保证结果精度的前提下, 用数据抽样或者数据尺度无关的近似算法来完成大数据的处理。

3) 分而治之的并行化处理。除上述两种方法外, 目前为止, 大数据处理最为有效和最重要的方法还是采用大数据并行化算法, 在一个大规模的分布式数据存储和并行计算平台上完成大数据并行化处理。

### 1.2.4 大数据计算模式和系统

MapReduce 计算模式的出现有力推动了大数据技术和应用的发展, 使其成为目前大数据处理最成功的主流大数据计算模式。然而, 现实世界中的大数据处理问题复杂多样, 难以有一种单一的计算模式能涵盖所有不同的大数据计算需求。研究和实际应用中发现, 由于 MapReduce 主要适合于进行大数据线下批处理, 在面向低延迟和具有复杂数据关系和复杂计算的大数据问题时有很大的不适应性。因此, 近几年来学术界和业界在不断研究并推出多种不同的大数据计算模式。

所谓大数据计算模式, 是指根据大数据的不同数据特征和计算特征, 从多样性的大数据计算问题和需求中提炼并建立的各种高层抽象 (Abstraction) 和模型 (Model)。传统的并行计算方法主要从体系结构和编程语言的层面定义了一些较为底层的抽象和模型, 但由于大数

据处理问题具有很多高层的数据特征和计算特征，因此大数据处理需要更多地结合其数据特征和计算特性考虑更为高层的计算模式。

根据大数据处理多样性的需求，目前出现了多种典型和重要的大数据计算模式。与这些计算模式相适应，出现了很多对应的大数据计算系统和工具，如表 1-1 所示。

表 1-1 大数据计算模式及其对应的典型系统和工具

大数据计算模式	典型系统和工具
大数据查询分析计算	HBase, Hive, Cassandra, Premel, Impala, Shark, Hana, Redis, 等
批处理计算	MapReduce, Spark, 等
流式计算	Scribe, Flume, Storm, S4, Spark Steaming, 等
迭代计算	HaLoop, iMapReduce, Twister, Spark, 等
图计算	Pregel, Giraph, Trinity, PowerGraph, GraphX, 等
内存计算	Dremel, Hana, Redis, 等

1. 大数据查询分析计算模式与典型系统

由于行业数据规模的增长已大大超过了传统的关系数据库的承载和处理能力，因此，目前需要尽快研究并提供面向大数据存储管理和查询分析的新的技术方法和系统，尤其要解决在数据体量极大时如何能够提供实时或准实时的数据查询分析能力，满足企业日常的管理需求。然而，大数据的查询分析处理具有很大的技术挑战，在数量规模较大时，即使采用分布式数据存储管理和并行化计算方法，仍然难以达到关系数据库处理中小规模数据时那样的秒级响应性能。

大数据查询分析计算的典型系统包括 Hadoop 下的 HBase 和 Hive、Facebook 公司开发的 Cassandra、Google 公司的 Dremel、Cloudera 公司的实时查询引擎 Impala；此外为了实现更高性能的数据查询分析，还出现了不少基于内存的分布式数据存储管理和查询系统，如 Apache Spark 下的数据仓库 Shark、SAP 公司的 Hana、开源的 Redis 等。

2. 批处理计算模式与典型系统

最适合于完成大数据批处理的计算模式是 MapReduce，这是 MapReduce 设计之初的主要任务和目标。MapReduce 是一个单输入、两阶段（Map 和 Reduce）的数据处理过程。首先，MapReduce 对具有简单数据关系、易于划分的大规模数据采用“分而治之”的并行处理思想；然后将大量重复的数据记录处理过程总结成 Map 和 Reduce 两个抽象的操作；最后 MapReduce 提供了一个统一的并行计算框架，把并行计算所涉及到的诸多系统层细节都交给计算框架去完成，以此大大简化了程序员进行并行化程序设计的负担。

MapReduce 的简单易用性使其成为目前大数据处理最成功的主流并行计算模式。在开源

社区的努力下，开源的 Hadoop 系统目前已成为较为成熟的大数据处理平台，并已发展成一个包括众多数据处理工具和环境的完整的生态系统。目前几乎国内外的各个著名 IT 企业都在使用 Hadoop 平台进行企业内大数据的计算处理。此外，Spark 系统也具备批处理计算的能力。

### 3. 流式计算模式与典型系统

流式计算是一种高实时性的计算模式，需要对一定时间窗口内应用系统产生的新数据完成实时的计算处理，避免造成数据堆积和丢失。很多行业的大数据应用，如电信、电力、道路监控等行业应用以及互联网行业的访问日志处理，都同时具有高流量的流式数据和大量积累的历史数据，因而在提供批处理计算模式的同时，系统还需要能具备高实时性的流式计算能力。流式计算的一个特点是数据运动、运算不动，不同的运算节点常常绑定在不同的服务器上。

Facebook 的 Scribe 和 Apache 的 Flume 都提供了一定的机制来构建日志数据处理流图。而更为通用的流式计算系统是 Twitter 公司的 Storm、Yahoo 公司的 S4 以及 Apache Spark Streaming。

### 4. 迭代计算模式与典型系统

为了克服 Hadoop MapReduce 难以支持迭代计算的缺陷，工业界和学术界对 Hadoop MapReduce 进行了不少改进研究。HaLoop 把迭代控制放到 MapReduce 作业执行的框架内部，并通过循环敏感的调度器保证前次迭代的 Reduce 输出和本次迭代的 Map 输入数据在同一台物理机上，以减少迭代间的数据传输开销；iMapReduce 在这个基础上保持 Map 和 Reduce 任务的持久性，规避启动和调度开销；而 Twister 在前两者的基础上进一步引入了可缓存的 Map 和 Reduce 对象，利用内存计算和 pub/sub 网络进行跨节点数据传输。

目前，一个具有快速和灵活的迭代计算能力的典型系统是 Spark，其采用了基于内存的 RDD 数据集模型实现快速的迭代计算。

### 5. 图计算模式与典型系统

社交网络、Web 链接关系图等都包含大量具有复杂关系的图数据，这些图数据规模很大，常常达到数十亿的顶点和上万亿的边数。这样大的数据规模和非常复杂的数据关系，给图数据的存储管理和计算分析带来了很大的技术难题。用 MapReduce 计算模式处理这种具有复杂数据关系的图数据通常不能适应，为此，需要引入图计算模式。

大规模图数据处理首先要解决数据的存储管理问题，通常大规模图数据也需要使用分布式存储方式。但是，由于图数据具有很强的数据关系，分布式存储就带来了一个重要的图划分问题（Graph Partitioning）。根据图数据问题本身的特点，图划分可以使用“边切分”和“顶



点切分”两种方式。在有效的图划分策略下，大规模图数据得以分布存储在不同节点上，并在每个节点上对本地子图进行并行化处理。与任务并行和数据并行的概念类似，由于图数据并行处理的特殊性，人们提出了一个新的“图并行”（Graph Parallel）的概念。事实上，图并行是数据并行的一个特殊形式，需要针对图数据处理的特征考虑一些特殊的数据组织模型和计算方法。

目前已经出现了很多分布式图计算系统，其中较为典型的系统包括 Google 公司的 Pregel、Facebook 对 Pregel 的开源实现 Giraph、微软公司的 Trinity、Spark 下的 GraphX，以及 CMU 的 GraphLab 以及由其衍生出来的目前性能最快的图数据处理系统 PowerGraph。

## 6. 内存计算模式与典型系统

Hadoop MapReduce 为大数据处理提供了一个很好的平台。然而，由于 MapReduce 设计之初是为大数据线下批处理而设计的，随着数据规模的不断扩大，对于很多需要高响应性能的大数据查询分析计算问题，现有的以 Hadoop 为代表的大数据处理平台在计算性能上往往难以满足要求。随着内存价格的不断下降以及服务器可配置的内存容量的不断提高，用内存计算完成高速的大数据处理已经成为大数据计算的一个重要发展趋势。例如，Hana 系统设计者总结了很多实际的商业应用后发现，一个提供 50TB 总内存容量的计算集群将能够满足绝大多数现有的商业系统对大数据的查询分析处理要求，如果一个服务器节点可配置 1TB ~ 2TB 的内存，则需要 25 ~ 50 个服务器节点。目前 Intel Xeon E-7 系列处理器最大可支持高达 1.5TB 的内存，因此，配置一个上述大小规模的内存计算集群是可以做到的。

### 1.2.5 大数据计算模式的发展趋势

近几年来，由于大数据处理和应用需求急剧增长，同时也由于大数据处理的多样性和复杂性，针对以上的典型的大数据计算模式，学术界和工业界不断研究推出新的或改进的计算模式和系统工具平台。目前主要有以下三方面的重要发展趋势和方向：Hadoop 性能提升和功能增强，混合式大数据计算模式，以及基于内存计算的大数据计算模式和技术。

#### 1. Hadoop 性能提升和功能增强

尽管 Hadoop 还存在很多不足，但由于 Hadoop 已发展成为目前最主流的大数据处理平台、并得到广泛的使用，因此，目前人们并不会抛弃 Hadoop 平台，而是试图不断改进和发展现有的平台，增加其对各种不同大数据处理问题的适用性。目前，Hadoop 社区正努力扩展现有的计算模式框架和平台，以便能解决现有版本在计算性能、计算模式、系统构架和处理能力上的诸多不足，这正是目前 Hadoop2.0 新版本“YARN”的努力目标。目前不断有新的计算模式和计算系统出现，预计今后相当长一段时间内，Hadoop 平台将与各种新的计算模式和系统共存，并相互融合，形成新一代的大数据处理系统和平台。



## 2. 混合式计算模式

现实世界大数据应用复杂多样，可能会同时包含不同特征的数据和计算，在这种情况下单一的计算模式多半难以满足整个应用的需求，因此需要考虑不同计算模式的混搭使用。

混合式计算模式可体现在两个层面上。一个层面是传统并行计算所关注的体系结构与低层并行程序设计语言层面计算模式的混合，例如，在体系结构层，可根据大数据应用问题的需要搭建混合式的系统构架，如 MapReduce 集群 + GPU-CUDA 的混合，或者 MapReduce 集群 + 基于 MIC (Intel Xeon Phi 众核协处理系统) 的 OpenMP/MPI 的混合模型。

混合模型的另一个层面是以上所述的大数据处理高层计算模式的混合。比如，一个大数据应用可能同时需要提供流式计算模式以便接收每天产生的大量流式数据，这些数据得到保存后成为历史数据，此时会需要提供基于 SQL 或 NoSQL 的数据查询分析能力以便进行日常的数据查询分析；进一步，为了进行商业智能分析，可能还需要进行基于机器学习的深度数据挖掘分析，此时系统需要能提供线下批处理计算模式以及复杂机器学习算法的迭代计算模式；一些大数据计算任务可能还直接涉及到复杂图计算或者间接转化为图计算问题。因此，很多大数据处理问题都需要有多种混合计算模式的支持。此外，为了提高各种计算模式处理大数据时的计算性能，各种计算模式都在与内存计算模式混合，实现高实时性的大数据查询和计算分析。

混合计算模式之集大成者当属 UC Berkeley AMPLab 研发、现已成为 Apache 开源项目的 Spark 系统，其涵盖了几乎所有典型的大数据计算模式，包括迭代计算、批处理计算、内存计算、流式计算 (Spark Streaming)、数据查询分析计算 (Shark) 以及图计算 (GraphX)，提供了优异的计算性能，同时还保持与 Hadoop 平台的兼容性。

## 3. 内存计算

为了进一步提高大数据处理的性能，目前已经出现一个基本共识，即随着内存成本的不断降低，内存计算将成为最终跨越大数据计算性能障碍、实现高实时高响应计算的一个最有效的技术手段。因此，目前越来越多的研究者和开发者在关注基于内存计算的大数据处理技术，不断推出各种基于内存计算的计算模式和系统。

内存计算是一种在体系结构层面上的解决方法，因此可以适用于各种不同的计算模式，从基本的数据查询分析计算，到批处理计算和流式计算，再到迭代计算和图计算，都可以基于内存计算加以实现，因此我们可以看到各种大数据计算模式下都有基于内存计算实现的系统，比较典型的系统包括 SAP 公司的 Hana 内存数据库、开源的键值对内存数据库 Redis、微软公司的图数据计算系统 Trinity、Apache Spark 等。

### 1.2.6 大数据的主要技术层面和技术内容

大数据是诸多计算技术的融合。从大的方面来分，大数据技术与研究主要分为大数据基

础理论、大数据关键技术和系统、大数据应用以及大数据信息资源库等几个重要方面。

从信息系统的角度来看，大数据处理是一个涉及整个软硬件系统各个层面的综合性信息处理技术。从信息系统角度可将大数据处理分为基础设施层、系统软件层、并行化算法层以及应用层。图 1-12 所示是从信息处理系统角度所看到的大数据技术的主要技术层面和技术内容。

应用层	大数据行业应用/服务层	电信/公安/商业/金融/遥感遥测/勘探/生物医药……
		领域应用/服务需求和计算模型
	应用开发层	分析工具/开发环境和工具/行业应用系统开发
并行化算法层	应用算法层	社会网络，排名与推荐，商业智能，自然语言处理，生物信息媒体分析检索，Web挖掘与检索，大数据分析可视化计算…
	基础算法层	并行化机器学习与数据挖掘算法
系统软件层	并行编程模型与计算框架层	并行计算模型与系统 批处理计算，流式计算，图计算，迭代计算 内存计算，混合式计算，定制式计算……
	大数据存储管理层	大数据查询（SQL，NoSQL NewSQL） 大数据存储（DFS，HBase RDFDB，MemD，RDB） 大数据采集与预处理
基础设施层	并行构架和资源平台层	集群，众核，GPU，混合式构架（如集群+众核，集群+GPU）云计算资源与支撑平台

图 1-12 从信息处理系统角度看大数据的主要技术层面与技术内容

1. 基础设施层

基础设施层主要提供大数据分布存储和并行计算的硬件基础设施和平台。目前大数据处理通用化的硬件设施是基于普通商用服务器的集群，在有特殊的数据处理需要时，这种通用化的集群也可以结合其他类型的并行计算设施一起工作，如基于众核的并行处理系统（如 GPU 或者 Intel 新近推出的 MIC），形成一种混合式的大数据并行处理构架和硬件平台。此外，随着云计算技术的发展，也可以与云计算资源管理和平台结合，在云计算平台上部署大数据基础设施，运用云计算平台中的虚拟化和弹性资源调度技术，为大数据处理提供可伸缩的计算资源和基础设施。

2. 系统软件层

在系统软件层，需要考虑大数据的存储管理和并行化计算系统软件。

### （1）分布式文件系统与数据查询管理系统

大数据处理首先面临的是如何解决大数据的存储管理问题。为了提供巨大的数据存储能力，人们的普遍共识是，利用分布式存储技术和系统提供可扩展的大数据存储能力。

首先需要有一个底层的分布式文件系统，以可扩展的方式支持对大规模数据文件的有效存储管理。但文件系统主要是以文件方式提供一个最基础性的大数据存储方式，其缺少结构化/半结构化数据的存储管理和访问能力，而且其编程接口对于很多应用来说还是太底层了。传统的数据库技术主要适用于规模相对较小的结构化数据的存储管理和查询，当数据规模增大或者要处理很多非结构化或半结构化数据时，传统数据库技术和系统将难以胜任。现实世界中的大数据不仅数据量大，而且具有多样化的形态特征。据统计，现实世界 80% 的数据都是非结构化或半结构化的。因此，系统软件层还需要研究解决大数据的存储管理和查询问题。由于 SQL 不太适用于非结构化/半结构化数据的管理查询，因此，人们提出了一种 NoSQL 的数据管理查询模式。但是，人们发现，最理想的还是能提供统一的数据管理查询方法，能对付各种不同类型的数据的查询管理。为此，人们进一步提出了 NewSQL 的概念和技术。

### （2）大数据并行计算模式和系统

解决了大数据的存储问题后，进一步面临的问题是，如何能快速有效地完成大规模数据的计算。大数据的数据规模之大，使得现有的串行计算方法难以在可接受的时间里快速完成大数据的处理和计算。为了提高大数据处理的效率，需要使用大数据并行计算模型和框架来支撑大数据的计算处理。目前最主流的大数据并行计算和框架是 Hadoop MapReduce 技术。与此同时，近年来人们开始研究并提供不同的大数据计算模型和方法，包括高实时低延迟要求的流式计算，具有复杂数据关系的图计算，面向基本数据管理的查询分析类计算，以及面向复杂数据分析挖掘的迭代和交互计算等。在大多数场景下，由于数据量巨大，大数据处理通常很难达到实时或低延迟响应。为了解决这个问题，近年来，人们提出了内存计算的概念和方法，尽可能利用大内存完成大数据的计算处理，以实现尽可能高的实时或低延迟响应。目前 Spark 已成为一个具有很大发展前景的新的大数据计算系统和平台，正受到工业界和学术界的广泛关注，有望成为与 Hadoop 并存的一种新的计算系统和平台。

## 3. 并行化算法层

基于以上的基础设施层和系统软件层，为了完成大数据的并行化处理，进一步需要考虑的问题是，如何能对各种大数据处理所需要的分析挖掘算法进行并行化设计。

大数据分析挖掘算法大多最终会归结到基础性的机器学习和数据挖掘算法上来。然而，面向大数据处理时，绝大多数现有的串行化机器学习和数据挖掘算法都难以在可接受的时间内有效完成大数据处理，因此，这些已有的机器学习和数据挖掘算法都需要进行并行化的设计和改造。

除此以外,还需要考虑很多更贴近上层具体应用和领域问题的应用层算法,例如,社会网络分析、分析推荐、商业智能分析、Web 搜索与挖掘、媒体分析检索、自然语言理解与分析、语义分析与检索、可视化分析等,虽然这些算法最终大都会归结到底层的机器学习和数据挖掘算法上,但它们本身会涉及到很多高层的特定算法问题,所有这些高层算法本身在面向大数据处理时也需要考虑如何进行并行化算法设计。

#### 4. 应用层

基于上述三个层面,可以构建各种行业或领域的大数据应用系统。大数据应用系统首先需要提供和使用各种大数据应用开发运行环境与工具;进一步,大数据应用开发的一个特别问题是,需要有应用领域的专家归纳行业应用问题和需求、构建行业应用和业务模型,这些模型往往需要专门的领域知识,没有应用行业领域专家的配合,单纯的计算机专业专业技术人员往往会无能为力,难以下手。只有在领域专家清晰构建了应用问题和业务模型后,计算机专业人员才能顺利完成应用系统的设计与开发。行业大数据分析和价值发现会涉及到很多复杂的行业和领域专业知识,这一特征在今天的大数据时代比以往任何时候都更为突出,这就是为什么我们在大数据研究原则中明确提出,大数据的研究应用需要以应用需求为导向、领域交叉为桥梁,从实际行业应用问题和需求出发,由行业和领域专家与计算机技术人员相互配合和协同,以完成大数据行业应用的开发。

### 1.3 MapReduce 并行计算技术简介

#### 1.3.1 MapReduce 的基本概念和由来

##### 1. 什么是 MapReduce

MapReduce 是面向大数据并行处理的计算模型、框架和平台,它隐含了以下三层含义:

1) MapReduce 是一个基于集群的高性能并行计算平台(Cluster Infrastructure)。它允许用市场上普通的商用服务器构成一个包含数十、数百至数千个节点的分布和并行计算集群。

2) MapReduce 是一个并行计算与运行软件框架(Software Framework)。它提供了一个庞大但设计精良的并行计算软件框架,能自动完成计算任务的并行化处理,自动划分计算数据和计算任务,在集群节点上自动分配和执行任务以及收集计算结果,将数据分布存储、数据通信、容错处理等并行计算涉及到的很多系统底层的复杂细节交由系统负责处理,大大减少了软件开发人员的负担。

3) MapReduce 是一个并行程序设计模型与方法(Programming Model & Methodology)。它借助于函数式程序设计语言 Lisp 的设计思想,提供了一种简便的并行程序设计方法,用 Map 和 Reduce 两个函数编程实现基本的并行计算任务,提供了抽象的操作和并行编程接口,



以简单方便地完成大规模数据的编程和计算处理。

## 2. MapReduce 的由来

MapReduce 最早是由 Google 公司研究提出的一种面向大规模数据处理的并行计算模型和方法。Google 公司设计 MapReduce 的初衷主要是为了解决其搜索引擎中大规模网页数据的并行化处理。Google 公司发明了 MapReduce 之后首先用其重新改写了其搜索引擎中的 Web 文档索引处理系统。但由于 MapReduce 可以普遍应用于很多大规模数据的计算问题，因此自发明 MapReduce 以后，Google 公司内部进一步将其广泛应用于很多大规模数据处理问题。到目前为止，Google 公司内有上万个各种不同的算法问题和程序都使用 MapReduce 进行处理。

2003 年和 2004 年，Google 公司在国际会议上分别发表了两篇关于 Google 分布式文件系统和 MapReduce 的论文，公布了 Google 的 GFS 和 MapReduce 的基本原理和主要设计思想。2004 年，开源项目 Lucene（搜索索引程序库）和 Nutch（搜索引擎）的创始人 Doug Cutting 发现 MapReduce 正是其所需要的解决大规模 Web 数据处理的重要技术，因而模仿 Google MapReduce，基于 Java 设计开发了一个称为 Hadoop 的开源 MapReduce 并行计算框架和系统。自此，Hadoop 成为 Apache 开源组织下最重要的项目，自其推出后很快得到了全球学术界和工业界的普遍关注，并得到推广和普及应用。

MapReduce 的推出给大数据并行处理带来了巨大的革命性影响，使其已经成为事实上的大数据处理的工业标准。尽管 MapReduce 还有很多局限性，但人们普遍公认，MapReduce 是到目前为止最为成功、最广为接受和最易于使用的大数据并行处理技术。MapReduce 的发展普及和带来的巨大影响远远超出了发明者和开源社区当初的意料，以至于马里兰大学教授、2010 年出版的《Data-Intensive Text Processing with MapReduce》一书的作者 Jimmy Lin 在书中提出：MapReduce 改变了我们组织大规模计算的方式，它代表了第一个有别于冯·诺依曼结构的计算模型，是在集群规模而非单个机器上组织大规模计算的新的抽象模型上的第一个重大突破，是到目前为止所见到的最为成功的基于大规模计算资源的计算模型。

### 1.3.2 MapReduce 的基本设计思想

面向大规模数据处理，MapReduce 有以下三个层面上的基本设计思想。

#### 1. 对付大数据并行处理：分而治之

一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是采用“分而治之”的策略进行并行化计算。MapReduce 采用了这种“分而治之”的设计思想，对相互间不具有或者有较少数据依赖关系

的大数据，用一定的数据划分方法对数据分片，然后将每个数据分片交由一个节点去处理，最后汇总处理结果。

## 2. 上升到抽象模型：Map 与 Reduce

### (1) Lisp 语言中的 Map 和 Reduce

MapReduce 借鉴了函数式程序设计语言 Lisp 的设计思想。Lisp 是一种列表处理语言。它是一种应用于人工智能处理的符号式语言，由 MIT 的人工智能专家、图灵奖获得者 John McCarthy 于 1958 年设计发明。

Lisp 定义了可对列表元素进行整体处理的各种操作，如：

`(add#(1 2 3 4) #(4 3 2 1))` 将产生结果：`#(5 5 5 5)`

Lisp 中也提供了类似于 Map 和 Reduce 的操作，如：

`(map'vector#+#(1 2 3 4) #(4 3 2 1))`

通过定义加法 map 运算将两个向量相加产生与前述 add 运算同样的结果 `#(5 5 5 5)`。

进一步，Lisp 也可以定义 reduce 操作进行某种归并运算，如：

`(reduce#+#(1 2 3 4))` 通过加法归并产生累加结果 10。

### (2) MapReduce 中的 Map 和 Reduce

MPI 等并行计算方法缺少高层并行编程模型，为了克服这一缺陷，MapReduce 借鉴了 Lisp 函数式语言中的思想，用 Map 和 Reduce 两个函数提供了高层的并行编程抽象模型和接口，程序员只要实现这两个基本接口即可快速完成并行化程序的设计。

与 Lisp 语言可以用来处理列表数据一样，MapReduce 的设计目标是可以对一组顺序组织的数据元素 / 记录进行处理。现实生活中，大数据往往是由一组重复的数据元素 / 记录组成，例如，一个 Web 访问日志文件数据会由大量的重复性的访问日志构成，对这种顺序式数据元素 / 记录的处理通常也是顺序式扫描处理。图 1-13 描述了典型的顺序式大数据处理的过程和特征：

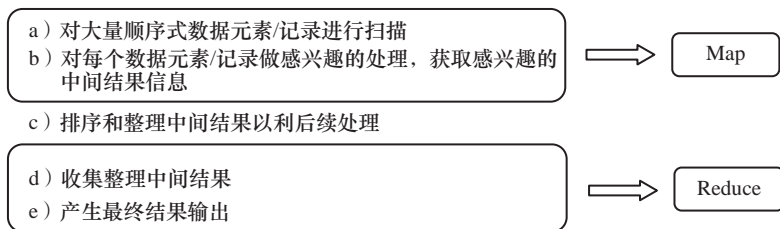


图 1-13 典型的顺序式大数据处理过程和特征

MapReduce 将以上的处理过程抽象为两个基本操作，把上述处理过程中的前两步抽象为 Map 操作，把后两步抽象为 Reduce 操作。于是 Map 操作主要负责对一组数据记录进行某种



重复处理，而 Reduce 操作主要负责对 Map 的中间结果进行某种进一步的结果整理和输出。以这种方式，MapReduce 为大数据处理过程中的主要处理操作提供了一种抽象机制。

### 3. 上升到构架：以统一构架为程序员隐藏系统层细节

MPI 等并行计算方法缺少统一的计算框架支持，程序员需要考虑数据存储、划分、分发、结果收集、错误恢复等诸多细节；为此，MapReduce 设计并提供了统一的计算框架，为程序员隐藏了绝大多数系统层面的处理细节，程序员只需要集中于应用问题和算法本身，而不需要关注其他系统层的处理细节，大大减轻了程序员开发程序的负担。

MapReduce 所提供的统一计算框架的主要目标是，实现自动并行化计算，为程序员隐藏系统层细节。该统一框架可负责自动完成以下系统底层相关的处理：

- 1) 计算任务的自动划分和调度。
- 2) 数据的自动化分布存储和划分。
- 3) 处理数据与计算任务的同步。
- 4) 结果数据的收集整理 (sorting, combining, partitioning, 等)。
- 5) 系统通信、负载平衡、计算性能优化处理。
- 6) 处理系统节点出错检测和失效恢复。

## 1.3.3 MapReduce 的主要功能和技术特征

### 1. MapReduce 的主要功能

MapReduce 通过抽象模型和计算框架把需要做什么 (What need to do) 与具体怎么做 (How to do) 分开了，为程序员提供了一个抽象和高层的编程接口和框架，程序员仅需要关心其应用层的具体计算问题，仅需编写少量的处理应用本身计算问题的程序代码；如何具体完成这个并行计算任务所相关的诸多系统层细节被隐藏起来，交给计算框架去处理：从分布代码的执行，到大到数千、小到数个节点集群的自动调度使用。

MapReduce 提供了以下的主要功能：

1) 数据划分和计算任务调度：系统自动将一个作业 (Job) 待处理的大数据划分为很多个数据块，每个数据块对应于一个计算任务 (Task)，并自动调度计算节点来处理相应的数据块。作业和任务调度功能主要负责分配和调度计算节点 (Map 节点或 Reduce 节点)，同时负责监控这些节点的执行状态，并负责 Map 节点执行的同步控制。

2) 数据 / 代码互定位：为了减少数据通信，一个基本原则是本地化数据处理，即一个计算节点尽可能处理其本地磁盘上所分布存储的数据，这实现了代码向数据的迁移；当无法进行这种本地化数据处理时，再寻找其他可用节点并将数据从网络上传送给该节点 (数据向代

码迁移), 但将尽可能从数据所在的本地机架上寻找可用节点以减少通信延迟。

3) 系统优化: 为了减少数据通信开销, 中间结果数据进入 Reduce 节点前会进行一定的合并处理; 一个 Reduce 节点所处理的数据可能会来自多个 Map 节点, 为了避免 Reduce 计算阶段发生数据相关性, Map 节点输出的中间结果需使用一定的策略进行适当的划分处理, 保证相关性数据发送到同一个 Reduce 节点; 此外, 系统还进行一些计算性能优化处理, 如对最慢的计算任务采用多备份执行、选最快完成者作为结果。

4) 出错检测和恢复: 以低端商用服务器构成的大规模 MapReduce 计算集群中, 节点硬件(主机、磁盘、内存等)出错和软件出错是常态, 因此 MapReduce 需要能检测并隔离出错节点, 并调度分配新的节点接管出错节点的计算任务。同时, 系统还将维护数据存储的可靠性, 用多备份冗余存储机制提高数据存储的可靠性, 并能及时检测和恢复出错的数据。

## 2. MapReduce 的主要技术特征

MapReduce 设计上具有以下主要的技术特征<sup>①</sup>:

### 1) 向“外”横向扩展, 而非向“上”纵向扩展

即 MapReduce 集群的构建完全选用价格便宜、易于扩展的低端商用服务器, 而非价格昂贵、不易扩展的高端服务器。对于大规模数据处理, 由于有大量数据存储需要, 显而易见, 基于低端服务器的集群远比基于高端服务器的集群优越, 这就是为什么 MapReduce 并行计算集群会基于低端服务器实现的原因。

### 2) 失效被认为是常态

MapReduce 集群中使用大量的低端服务器, 因此, 节点硬件失效和软件出错是常态, 因而一个良好设计、具有高容错性的并行计算系统不能因为节点失效而影响计算服务的质量, 任何节点失效都不应当导致结果的不一致或不确定性; 任何一个节点失效时, 其他节点要能够无缝接管失效节点的计算任务; 当失效节点恢复后应能自动无缝加入集群, 而不需要管理人员人工进行系统配置。MapReduce 并行计算软件框架使用了多种有效的错误检测和恢复机制, 如节点自动重启技术, 使集群和计算框架具有对付节点失效的健壮性, 能有效处理失效节点的检测和恢复。

### 3) 把处理向数据迁移

传统高性能计算系统通常有很多处理器节点与一些外存储器节点相连, 如用存储区域网络(Storage Area, SAN Network)连接的磁盘阵列, 因此, 大规模数据处理时外存文件数据 I/O 访问会成为一个制约系统性能的瓶颈。为了减少大规模数据并行计算系统中的数据通信开销, 代之以把数据传送到处理节点(数据向处理器或代码迁移), 应当考虑将处理向数据靠

① 引自马里兰大学 Jimmy Lin 教授所著图书《Data-Intensive Text processing with MapReduce》。

拢和迁移。MapReduce 采用了数据 / 代码互定位的技术方法，计算节点将首先尽量负责计算其本地存储的数据，以发挥数据本地化特点，仅当节点无法处理本地数据时，再采用就近原则寻找其他可用计算节点，并把数据传送到该可用计算节点。

#### 4) 顺序处理数据、避免随机访问数据

大规模数据处理的特点决定了大量的数据记录难以全部存放在内存，而通常只能放在外存中进行处理。由于磁盘的顺序访问要远比随机访问快得多，因此 MapReduce 主要设计为面向顺序式大规模数据的磁盘访问处理。为了实现面向大数据集批处理的高吞吐量的并行处理，MapReduce 可以利用集群中的大量数据存储节点同时访问数据，以此利用分布集群中大量节点上的磁盘集合提供高带宽的数据访问和传输。

#### 5) 为应用开发者隐藏系统层细节

软件工程实践指南中，专业程序员认为之所以写程序困难，是因为程序员需要记住太多的编程细节（从变量名到复杂算法的边界情况处理），这对大脑记忆是一个巨大的认知负担，需要高度集中注意力；而并行程序编写有更多困难，如需要考虑多线程中诸如同步等复杂繁琐的细节。由于并发执行中的不可预测性，程序的调试查错也十分困难；而且，大规模数据处理时程序员需要考虑诸如数据分布存储管理、数据分发、数据通信和同步、计算结果收集等诸多细节问题。MapReduce 提供了一种抽象机制将程序员与系统层细节隔离开来，程序员仅需描述需要计算什么（What to compute），而具体怎么去计算（How to compute）就交由系统的执行框架处理，这样程序员可从系统层细节中解放出来，而致力于其应用本身计算问题的算法设计。

#### 6) 平滑无缝的可扩展性

这里指出的可扩展性主要包括两层意义上的扩展性：数据扩展和系统规模扩展性。理想的软件算法应当能随着数据规模的扩大而表现出持续的有效性，性能上的下降程度应与数据规模扩大的倍数相当；在集群规模上，要求算法的计算性能应能随着节点数的增加保持接近线性程度的增长。绝大多数现有的单机算法都达不到以上理想的要求；把中间结果数据维护在内存中的单机算法在大规模数据处理时很快失效；从单机到基于大规模集群的并行计算从根本上需要完全不同的算法设计。奇妙的是，MapReduce 在很多情形下能实现以上理想的扩展性特征。多项研究发现，对于很多计算问题，基于 MapReduce 的计算性能可随节点数目增长保持近似于线性的增长。

## 1.4 Hadoop 系统简介

### 1.4.1 Hadoop 的概述与发展历史

Hadoop 系统最初的源头来自于 Apache Lucene 项目下的搜索引擎子项目 Nutch，该项目

的负责人是 Doug Cutting。2003 年, Google 公司为了解决其搜索引擎中大规模 Web 网页数据的处理, 研究发明了一套称为 MapReduce 的大规模数据并行处理技术, 并于 2004 年在著名的 OSDI 国际会议上发表了一篇题为“MapReduce:Simplified Data Processing on Large Clusters”的论文, 简要介绍 MapReduce 的基本设计思想。论文发表后, Doug Cutting 受到了很大启发, 他发现 Google MapReduce 所解决的大规模搜索引擎数据处理问题, 正是他同样面临并急需解决的问题。因而, 他尝试依据 Google MapReduce 的设计思想, 模仿 Google MapReduce 框架的设计思路, 用 Java 设计实现出了一套新的 MapReduce 并行处理软件系统, 并将其与 Nutch 分布式文件系统 NDFS 结合, 用以支持 Nutch 搜索引擎的数据处理。2006 年, 他们把 NDFS 和 MapReduce 从 Nutch 项目中分离出来, 成为一套独立的大规模数据处理软件系统, 并使用 Doug Cutting 小儿子当时呀呀学语称呼自己的玩具小象的名字“Hadoop”命名了这个系统。2008 年他们把 Hadoop 贡献出来, 成为 Apache 最大的一个开源项目, 并逐步发展成熟, 成为一个包含了 HDFS、MapReduce、HBase、Hive、Zookeeper 等一系列相关子项目的大数据处理平台和生态系统。

Hadoop 开源项目自最初推出后, 经历了数十个版本的演进。它从最初于 2007 年推出的 Hadoop-0.14.X 测试版, 一直发展到 2011 年 5 月推出了经过 4500 台服务器产品级测试的最早的稳定版 0.20.203.X。到 2011 年 12 月, Hadoop 又在 0.20.205 版基础上发布了 Hadoop1.0.0, 该版本到 2012 年 3 月发展为 Hadoop1.0.1 稳定版。1.0 版继续发展, 到 2013 年 8 月发展为 Hadoop1.2.1 稳定版。

与此同时, 由于 Hadoop1.X 以前版本在 MapReduce 基本构架的设计上存在作业主控节点 (JobTracker) 单点瓶颈、作业执行延迟过长、编程框架不灵活等较多的缺陷和不足, 2011 年 10 月, Hadoop 推出了基于新一代构架的 Hadoop0.23.0 测试版, 该版本系列最终演化为 Hadoop2.0 版本, 即新一代的 Hadoop 系统 YARN。2013 年 10 月 YARN 已经发展出 Hadoop2.2.0 稳定版。

### 1.4.2 Hadoop 系统分布式存储与并行计算构架

图 1-14 展示了 Hadoop 系统的分布式存储和并行计算构架。从硬件体系结构上看, Hadoop 系统是一个运行于普通的商用服务器集群的分布式存储和并行计算系统。集群中将有一个主控节点用来控制和管理整个集群的正常运行, 并协调管理集群中各个从节点完成数据存储和计算任务。每个从节点将同时担任数据存储节点和数据计算节点两种角色, 这样设计的目的是在大数据环境下实现尽可能的本地化计算, 以此提高系统的处理性能。为了能及时检测和发现集群中某个从节点发生故障失效, 主控节点采用心跳机制 (Heartbeat) 定期检测从节点, 如果从节点不能有效回应心跳信息, 则系统认为这个从节点失效。

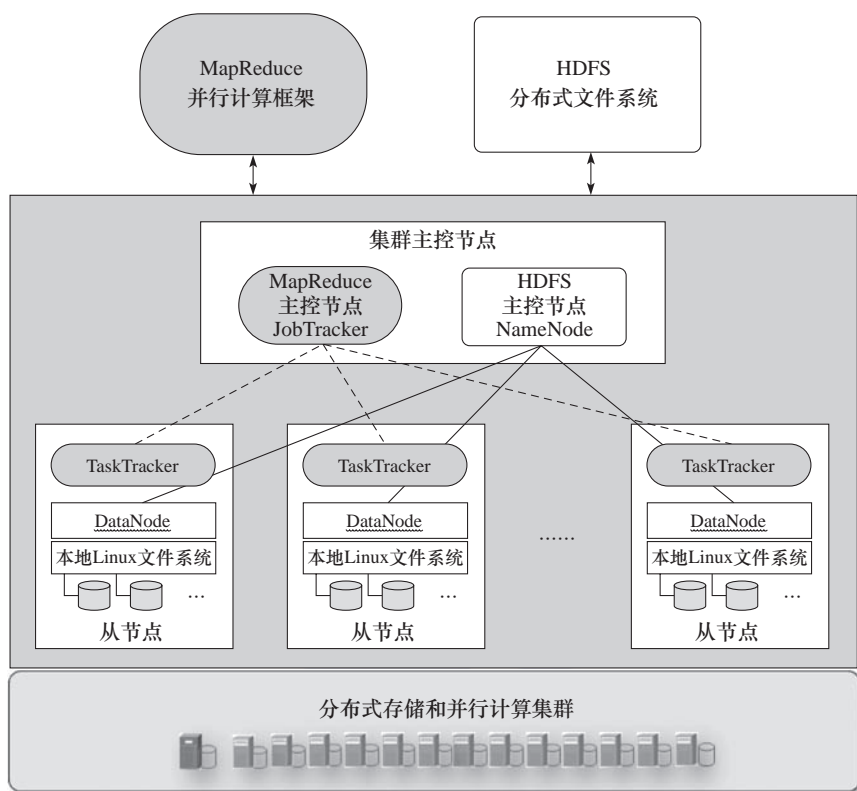


图 1-14 Hadoop 系统分布式存储与并行计算构架

从软件系统角度看，Hadoop 系统包括分布式存储和并行计算两个部分。分布式存储构架中，Hadoop 基于每个从节点上的本地文件系统，构建一个逻辑上整体化的分布式文件系统，以此提供大规模可扩展的分布式数据存储功能，这个分布式文件系统称为 HDFS (Hadoop Distributed File System)，其中，负责控制和管理整个分布式文件系统的主控节点称为 NameNode，而每个具体负责数据存储的从节点称为 DataNode。

进一步，为了能对存储在 HDFS 中的大规模数据进行并行化的计算处理，Hadoop 又提供了一个称为 MapReduce 的并行化计算框架。该框架能有效管理和调度整个集群中的节点来完成并行化程序的执行和数据处理，并能让每个从节点尽可能对本地节点上的数据进行本地化计算，其中，负责管理和调度整个集群进行计算的主控节点称为 JobTracker，而每个负责具体的数据计算的从节点称为 TaskTracker。JobTracker 可以与负责管理数据存储的主控节点 NameNode 设置在同一个物理的主控服务器上，在系统规模较大、各自负载较重时两者也可以分开设置。但数据存储节点 DataNode 与计算节点 TaskTracker 会配对地设置在同一个物理的从节点服务器上。



Hadoop 系统中的其他子系统，例如 HBase、Hive 等，将建立在上述 HDFS 分布式文件系统和 MapReduce 并行化计算框架之上。

### 1.4.3 Hadoop 平台的基本组成与生态系统

Hadoop 系统运行于一个由普通商用服务器组成的计算集群上，该服务器集群在提供大规模分布式数据存储资源的同时，也提供大规模的并行化计算资源。

在大数据处理软件系统上，随着 Apache Hadoop 系统开源化的发展，在最初包含 HDFS、MapReduce、HBase 等基本子系统的基础上，至今 Hadoop 平台已经演进为一个包含很多相关子系统的完整的大数据处理生态系统。图 1-15 展示了 Hadoop 平台的基本组成与生态系统。

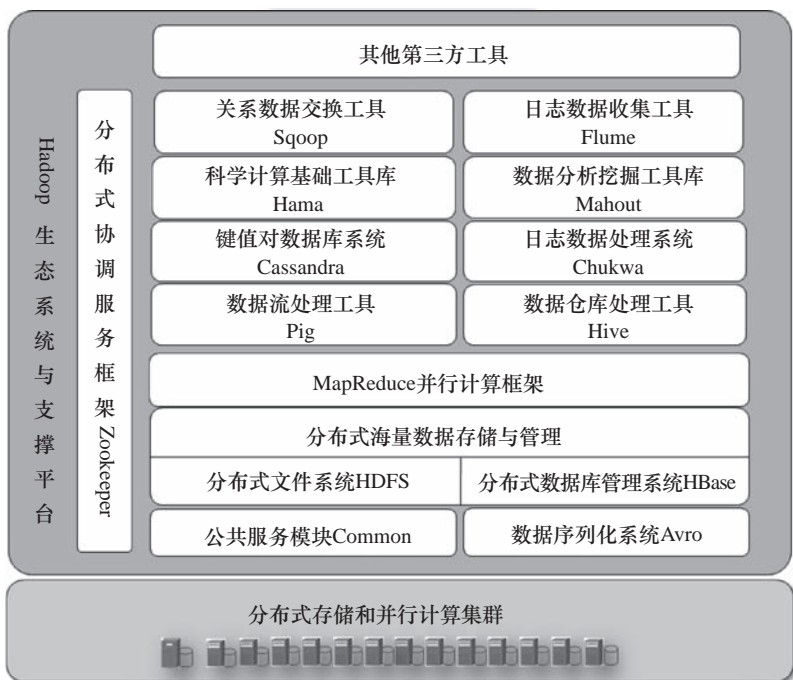


图 1-15 Hadoop 平台的基本组成与生态系统

#### 1. MapReduce 并行计算框架

MapReduce 并行计算框架是一个并行化程序执行系统。它提供了一个包含 Map 和 Reduce 两阶段的并行处理模型和过程，提供一个并行化编程模型和接口，让程序员可以方便快速地编写出大数据并行处理程序。MapReduce 以键值对数据输入方式来处理数据，并能自动完成数据的划分和调度管理。在程序执行时，MapReduce 并行计算框架将负责调度和分配



计算资源，划分和输入输出数据，调度程序的执行，监控程序的执行状态，并负责程序执行时各计算节点的同步以及中间结果的收集整理。MapReduce 框架提供了一组完整的供程序员开发 MapReduce 应用程序的编程接口。

## 2. 分布式文件系统 HDFS

HDFS (Hadoop Distributed File System) 是一个类似于 Google GFS 的开源的分布式文件系统。它提供了一个可扩展、高可靠、高可用的大规模数据分布式存储管理系统，基于物理上分布在各个数据存储节点的本地 Linux 系统的文件系统，为上层应用程序提供了一个逻辑上成为整体的大规模数据存储文件系统。与 GFS 类似，HDFS 采用多副本（默认为 3 个副本）数据冗余存储机制，并提供了有效的数据出错检测和数据恢复机制，大大提高了数据存储的可靠性。

## 3. 分布式数据库管理系统 HBase

为了克服 HDFS 难以管理结构化 / 半结构化海量数据的缺点，Hadoop 提供了一个大规模分布式数据库管理和查询系统 HBase。HBase 是一个建立在 HDFS 之上的分布式数据库，它是一个分布式可扩展的 NoSQL 数据库，提供了对结构化、半结构化甚至非结构化大数据的实时读写和随机访问能力。HBase 提供了一个基于行、列和时间戳的三维数据管理模型，HBase 中每张表的记录数（行数）可以多达几十亿条甚至更多，每条记录可以拥有多达上百万的字段。

## 4. 公共服务模块 Common

Common 是一套为整个 Hadoop 系统提供底层支撑服务和常用工具类库和 API 编程接口，这些底层服务包括 Hadoop 抽象文件系统 FileSystem、远程过程调用 RPC、系统配置工具 Configuration 以及序列化机制。在 0.20 及以前的版本中，Common 包含 HDFS、MapReduce 和其他公共的项目内容；从 0.21 版本开始，HDFS 和 MapReduce 被分离为独立的子项目，其余部分内容构成 Hadoop Common。

## 5. 数据序列化系统 Avro

Avro 是一个数据序列化系统，用于将数据结构或数据对象转换成便于数据存储和网络传输的格式。Avro 提供了丰富的数据结构类型，快速可压缩的二进制数据格式，存储持久性数据的文件集，远程调用 RPC 和简单动态语言集成等功能。

## 6. 分布式协调服务框架 Zookeeper

Zookeeper 是一个分布式协调服务框架，主要用于解决分布式环境中的一致性问题。

Zookeeper 主要用于提供分布式应用中经常需要的系统可靠性维护、数据状态同步、统一命名服务、分布式应用配置项管理等功能。Zookeeper 可用来在分布式环境下维护系统运行管理中的一些数据量不大的重要状态数据，并提供监测数据状态变化的机制，以此配合其他 Hadoop 子系统（如 HBase、Hama 等）或者用户开发的应用系统，解决分布式环境下系统可靠性管理和数据状态维护等问题。

## 7. 分布式数据仓库处理工具 Hive

Hive 是一个建立在 Hadoop 之上的数据仓库，用于管理存储于 HDFS 或 HBase 中的结构化 / 半结构化数据。它最早由 Facebook 开发并用于处理并分析大量的用户及日志数据，2008 年 Facebook 将其贡献给 Apache 成为 Hadoop 开源项目。为了便于熟悉 SQL 的传统数据库使用者使用 Hadoop 系统进行数据查询分析，Hive 允许直接用类似 SQL 的 HiveQL 查询语言作为编程接口编写数据查询分析程序，并提供数据仓库所需要的数据抽取转换、存储管理和查询分析功能，而 HiveQL 语句在底层实现时被转换为相应的 MapReduce 程序加以执行。

## 8. 数据流处理工具 Pig

Pig 是一个用来处理大规模数据集的平台，由 Yahoo! 贡献给 Apache 成为开源项目。它简化了使用 Hadoop 进行数据分析处理的难度，提供一个面向领域的高层抽象语言 Pig Latin，通过该语言，程序员可以将复杂的数据分析任务实现为 Pig 操作上的数据流脚本，这些脚本最终执行时将被系统自动转换为 MapReduce 任务链，在 Hadoop 上加以执行。Yahoo! 有大量的 MapReduce 作业是通过 Pig 实现的。

## 9. 键值对数据库系统 Cassandra

Cassandra 是一套分布式的 K-V 型的数据库系统，最初由 Facebook 开发，用于存储邮箱等比较简单的格式化数据，后 Facebook 将 Cassandra 贡献出来成为 Hadoop 开源项目。Cassandra 以 Amazon 专有的完全分布式 Dynamo 为基础，结合了 Google BigTable 基于列族 (Column Family) 的数据模型，提供了一套高度可扩展、最终一致、分布式的结构化键值存储系统。它结合了 Dynamo 的分布技术和 Google 的 Bigtable 数据模型，更好地满足了海量数据存储的需求。同时，Cassandra 变更垂直扩展为水平扩展，相比其他典型的键值数据存储模型，Cassandra 提供了更为丰富的功能。

## 10. 日志数据处理系统 Chukwa

Chukwa 是一个由 Yahoo! 贡献的开源的数据收集系统，主要用于日志的收集和数据的监控，并与 MapReduce 协同处理数据。Chukwa 是一个基于 Hadoop 的大规模集群监控系统

统，继承了 Hadoop 系统的可靠性，具有良好的适应性和扩展性。它使用 HDFS 来存储数据，使用 MapReduce 来处理数据，同时还提供灵活强大的辅助工具用以分析、显示、监视数据结果。

### 11. 科学计算基础工具库 Hama

Hama 是一个基于 BSP 并行计算模型（Bulk Synchronous Parallel，大同步并行模型）的计算框架，主要提供一套支撑框架和工具，支持大规模科学计算或者具有复杂数据关联性的图计算。Hama 类似 Google 公司开发的 Pregel，Google 利用 Pregel 来实现图遍历（BFS）、最短路径（SSSP）、PageRank 等计算。Hama 可以与 Hadoop 的 HDFS 进行完美的整合，利用 HDFS 对需要运行的任务和数据进行持久化存储。由于 BSP 在并行化计算模型上的灵活性，Hama 框架可在大规模科学计算和图计算方面得到较多应用，完成矩阵计算、排序计算、PageRank、BFS 等不同的大数据计算和处理任务。

### 12. 数据分析挖掘工具库 Mahout

Mahout 来源于 Apache Lucene 子项目，其主要目标是创建并提供经典的机器学习和数据挖掘并行化算法类库，以便减轻需要使用这些算法进行数据分析挖掘的程序的编程负担，不需要自己再去实现这些算法。Mahout 现在已经包含了聚类、分类、推荐引擎、频繁项集挖掘等广泛使用的机器学习和数据挖掘算法。此外，它还提供了包含数据输入输出工具，以及与其他数据存储管理系统进行数据集成的工具和构架。

### 13. 关系数据交换工具 Sqoop

Sqoop 是 SQL-to-Hadoop 的缩写，是一个在关系数据库与 Hadoop 平台间进行快速批量数据交换的工具。它可以将一个关系数据库中的数据批量导入 Hadoop 的 HDFS、HBase、Hive 中，也可以反过来将 Hadoop 平台中的数据导入关系数据库中。Sqoop 充分利用了 Hadoop MapReduce 的并行化优点，整个数据交换过程基于 MapReduce 实现并行化的快速处理。

### 14. 日志数据收集工具 Flume

Flume 是由 Cloudera 开发维护的一个分布式、高可靠、高可用、适合复杂环境下大规模日志数据采集的系统。它将数据从产生、传输、处理、输出的过程抽象为数据流，并允许在数据源中定义数据发送方，从而支持收集基于各种不同传输协议的数据，并提供对日志数据进行简单的数据过滤、格式转换等处理能力。输出时，Flume 可支持将日志数据写往用户定制的输出目标。

#### 1.4.4 Hadoop 的应用现状和发展趋势

Hadoop 因其在大数据处理领域具有广泛的实用性以及良好的易用性，自 2007 年推出后，很快在工业界得到普及应用，同时得到了学术界的广泛关注和研究。在短短的几年中，Hadoop 很快成为到目前为止最为成功、最广泛接受使用的大数据处理主流技术和系统平台，并且成为一种大数据处理事实上的工业标准，得到了工业界大量的进一步开发和改进，并在业界和应用行业尤其是互联网行业得到了广泛的应用。由于在系统性能和功能方面存在不足，Hadoop 在发展过程中进行了不断的改进，自 2007 年推出首个版本以来，目前已经先后推出数十个版本。

但是，由于其最初面向高吞吐率线下批处理的设计目标，以及起初系统构架设计上的诸多先天性的不足，尽管 Hadoop 开源社区一直致力于不断改进和完善系统，但是 Hadoop1.X 以前版本一直存在不少广为诟病的缺陷，包括主控节点单点瓶颈易造成系统拥堵和失效，作业执行响应性能较低难以满足高实时低延迟数据查询分析处理需求，固定的 Map 和 Reduce 两阶段模型框架难以提供灵活的编程能力、难以支持高效的迭代计算、流式计算、图数据计算等不同的计算和编程模式。

为此，在 Hadoop0.20 版本推出之后，Hadoop 开源社区开始设计全新构架的新一代 Hadoop 系统，并于 2011 年 10 月推出了基于新一代构架的 Hadoop0.23.0 测试版，该版本后演化为 Hadoop2.0 版本，即新一代的 Hadoop 系统 YARN。YARN 构架将主控节点的资源管理和作业管理功能分离设置，引入了全局资源管理器（Resource Manager）和针对每个作业的应用主控管理器（Application Master），以此减轻原主控节点的负担，并可基于 Zookeeper 实现资源管理器的失效恢复，以此提高了 Hadoop 系统的高可用性（High Availability, HA）。YARN 还引入了资源容器（Resource Container）的概念，将系统计算资源统一划分和封装为很多资源单元，不再像此前版本那样区分 Map 和 Reduce 计算资源，以此提高计算资源的利用率。此外，YARN 还能容纳 MapReduce 以外的其他多种并行计算模型和框架，提高了 Hadoop 框架并行化编程的灵活性。

与此同时，由于 Hadoop 系统和框架对于不同大数据计算模式支持能力上的不足，在 Hadoop 开源社区之外，人们在不断研究推出可支持不同的大数据计算模式的系统。其中，目前最为关注的当数加州大学伯克利分校 AMP 实验室（Algorithms, Machines, and People Lab）研究开发的 Spark 系统，该系统可广泛支持批处理、内存计算、流式计算、迭代计算、图数据计算等众多计算模式。然而，由于 Hadoop 系统在大规模数据分布存储和批处理能力，以及在系统的可扩展性和易用性上仍然具有不少其他系统所难以具备的优点，并且由于近几年来业界和应用行业在 Hadoop 开发和应用上已有大量的前期投入和上线应用系统，以及

Hadoop 所形成的包含各种丰富的工具软件的完整生态环境，同时也随着 Hadoop 自身向新一代系统的演进和不断改进，在今后相当长一段时间内，Hadoop 系统将继续保持其在大数据处理领域的主流技术和平台的地位，同时其他各种新的系统也将逐步与 Hadoop 系统相互融合和共存。

在开源 Hadoop 系统发展的同时，工业界也有不少公司基于开源的 Hadoop 系统进行一系列的商业化版本开发，他们针对开源系统在系统性能优化、系统可用性和可靠性以及系统功能增强方面进行大量的研究和产品开发工作，形成商业化的发行版。最广为大家熟知的是 Hadoop 系统的创始者组织成立的 Cloudera 公司，研究开发了 Hadoop 商业发行版 CDH，并在美国诸多的行业得到很好的推广应用。Intel 公司自 2009 年以来也研究开发了 Intel 发行版 Hadoop 系统 IDH，在中国诸多大型应用行业得到了良好的推广应用，本书第 7 章将详细介绍 Intel Hadoop 系统在性能优化和功能增强方面的主要技术内容和使用方法。





# Hadoop 系统的安装与操作管理

可以用三种不同的方式安装 Hadoop。本章将分别介绍这几种方法在 Linux 环境下的安装和运行，并介绍基本的 MapReduce 程序开发过程、远程作业提交与执行方法以及如何查看作业执行结果。

## 2.1 Hadoop 系统安装方法简介

Hadoop 可以用三种不同的方式进行安装。第一种方式是单机方式，它允许在一台运行 Linux 或 Windows 下虚拟 Linux 的单机上安装运行 Hadoop 系统。该方式通常适用于程序员先在本机编写和调试程序。第二种方式是单机伪分布方式，它允许在一台运行 Linux 或 Windows 下虚拟 Linux 的单机上，用伪分布方式，以不同的 Java 进程模拟分布运行环境中的 NameNode、DataNode、JobTracker、TaskTracker 等各类节点。第三种方式是集群分布模式，它是在一个真实的集群环境下安装运行 Hadoop 系统，集群的每个节点可以运行 Linux 或 Windows 下的虚拟 Linux。单机和单机伪分布模式下编写调试完成的程序通常不需修改即可在真实的分布式 Hadoop 集群下运行，但通常需要修改配置。

在 Windows 下安装运行 Hadoop，首先需要安装 Cygwin 来模拟 Linux 环境。通常，如果用户需要自己的 Windows 环境单机上安装运行 Hadoop 时可以这样做；但如果是真实的集群环境建议不要用这种方式，因为 Windows 环境下模拟虚拟 Linux 环境运行 Hadoop 会比较复杂，而且运行效率将大为下降。

## 2.2 单机和单机伪分布式 Hadoop 系统安装基本步骤

默认情况下，Hadoop 被配置成一个以非分布式模式运行的独立 Java 进程，适合程序员

在本地做编程和调试工作。Hadoop 也可以在单节点上以伪分布式模式运行，用不同的 Java 进程模拟分布式运行中的各类节点（NameNode、DataNode、JobTracker、TaskTracker 和 Secondary NameNode）。

### 2.2.1 安装和配置 JDK

Hadoop 是以 Java 语言写成，因而需要在本地计算机上预先安装 Java 6 或者更新版本。尽管其他 Java 安装包也声称支持 Hadoop，但使用最广的仍然要数 Sun 的 JDK。

在这里，我们采用的版本为 jdk-6u23-linux-x64。安装步骤如下：

- 1) 将 jdk-6u23-linux-x64-rpm.bin 拷贝到所需要的安装目录下，如 /usr/jdk。
- 2) 执行 ./jdk-6u23-linux-x64-rpm.bin 安装文件。
- 3) 配置 JAVA\_HOME 以及 CLASS\_PATH, vi 进入 /etc/profile，在文件最后加上如下语句：

```
JAVA_HOME=/usr/java/jdk1.6.0
PATH=$JAVA_HOME/bin:$PATH
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME PATH CLASSPATH
```

保存退出，执行以下命令使得配置文件生效。

```
$source /etc/profile
```

- 4) 执行以下命令查看当前版本配置是否生效。

```
$java -version
```

- 5) 查看 CLASSPATH 有无生效，可编写 HelloWorld 类至当前目录，执行以下命令分别进行编译和执行，查看结果是否正确。

```
$javac HelloWorld.java
$java HelloWorld
```

### 2.2.2 创建 Hadoop 用户

为 Hadoop 创建一个专门的用户，例如 hadoop : hadoop-user（用户名：用户组）。可以在安装系统的时候就创建，也可以在安装好之后用如下命令创建：

```
#groupadd hadoop-user
#useradd -g hadoop-user hadoop
#passwd hadoop
```

### 2.2.3 下载安装 Hadoop

从 Apache Hadoop 发布页面（<http://hadoop.apache.org/coases.html>）下载一个稳定的发布包

(通常被打包成一个 gzipped tar 文件), 再解压缩到本地文件系统中。在这里采用的 Hadoop 版本是 Hadoop-1.2.1。

```
$tar -xzvf hadoop-1.2.1.tar.gz
```

## 2.2.4 配置 SSH

为了保证在远程管理 Hadoop 节点以及 Hadoop 节点间用户共享访问时的安全性, Hadoop 系统需要配置和使用 SSH (安全外壳协议)。在单机模式下无需任何守护进程, 因此不需要进行 SSH 设置, 但是在单机伪分布模式和集群分布模式下需要进行 SSH 设置。

Hadoop 需要通过 SSH 来启动 Slave 列表中各台主机的守护进程。但由于 SSH 需要用户密码登录, 因此为了在系统运行中完成节点的免密码登录和访问, 需要将 SSH 配置成免密码登录方式。

配置 SSH 的主要工作是创建一个认证文件, 使得用户以 public key 方式登录, 而不用手工输入密码。配置基本配置步骤如下。

1) 生成密钥对, 执行如下命令:

```
$ssh-keygen -t rsa
```

2) 然后一直按 <Enter> 键, 就会按照默认的选项将生成的密钥对保存在 .ssh/id\_rsa 文件中, 如图 2-1 所示。



图 2-1 将密钥对保存在 .ssh/id\_rsa 文件中

3) 进入 .ssh 目录, 执行如下命令:

```
$cp id_rsa.pub authorized_keys
```

4) 此后执行如下命令:

```
$ssh localhost
```

5) 测试一下能否登录, 是否可实现用 SSH 连接并且不需要输入密码。

## 2.2.5 配置 Hadoop 环境

切换到 Hadoop 的安装路径找到 hadoop-1.2.1 下的 conf/hadoop-env.sh 文件夹, 使用 vi 或文本编辑器打开, 添加如下语句:

```
$export JAVA_HOME=/usr/java/jdk1.6.0
```

Hadoop-1.2.1 的配置文件是 conf/core-site.xml、conf/hdfs-site.xml 和 conf/mapred-site.xml。其中 core-site.xml 是全局配置文件, hdfs-site.xml 是 HDFS 的配置文件, mapred-site.xml 是 MapReduce 的配置文件。以下列出几个示例配置文件。

core-site.xml 的文档内容如下所示:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/tmp/hadoop/hadoop-${user.name}</value>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
<!--注意这里要填写自己的 IP-->
</property>
</configuration>
```

hdfs-site.xml 的文档内容如下所示:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>dfs.namenode.name.dir</name>
<value>/home/hadoop/hadoop_dir/dfs/name</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:///home/hadoop/hadoop_dir/dfs/data</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

mapred-site.xml 的文档内容如下所示：

```
<?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>
    <!--注意这里要填写自己的 IP-->
  </property>
  <property>
    <name>mapreduce.cluster.local.dir</name>
    <value>/home/hadoop/hadoop_dir/mapred/local</value>
  </property>
  <property>
    <name>mapreduce.jobtracker.system.dir</name>
    <value>/home/hadoop/hadoop_dir/mapred/system</value>
  </property>
</configuration>
```

## 2.2.6 Hadoop 的运行

### 1. 格式化 HDFS 文件系统

在初次安装和使用 Hadoop 之前，需要格式化分布式文件系统 HDFS。使用如下命令格式化分布式文件系统：

```
$bin/hadoop namenode -format
```

### 2. 启动 Hadoop 环境

启动 Hadoop 守护进程，命令如下：

```
$bin/start-all.sh
```

成功执行后将在本机上启动 NameNode、DataNode、JobTracker、TaskTracker 和 Secondary NameNode 五个新的 Java 进程。

### 3. 停止 Hadoop 守护进程

最后需要停止 Hadoop 守护进程，命令如下：

```
$bin/stop-all.sh
```

## 2.2.7 运行测试程序

下面用一个程序测试能否运行任务，示例程序是一个 Hadoop 自带的 PI 值的计算。第一



个参数是指要运行的 map 的次数，第二个参数是指每个 map 任务取样的个数。

```
$hadoop jar $HADOOP_HOME/hadoop-examples-1.2.1.jar \
pi2 5
```

### 2.2.8 查看集群状态

当 Hadoop 启动之后，可以用 jps 命令查看一下它是不是正常启动。

```
$jps
4706 JobTracker
4582 SecondaryNameNode
4278 NameNode
4413 DataNode
4853 TaskTracker
4889 Jps
```

如果显示以上的信息，则表示 Hadoop 已正常启动。

## 2.3 集群分布式 Hadoop 系统安装基本步骤

Hadoop 安装时对 HDFS 和 MapReduce 的节点允许用不同的系统配置方式。在 HDFS 看来，节点分别为主控节点 NameNode 和数据存储节点 DataNode，其中 NameNode 只有一个，DataNode 可以有多个。在 MapReduce 看来，节点又可以分为作业主控节点 JobTracker 和任务执行节点 TaskTracker，其中 JobTracker 只有一个，TaskTracker 可以有多个。NameNode 和 JobTracker 可以部署在不同的机器上，也可以部署在同一台机器上，但一般中小规模的集群通常都把 NameNode 和 JobTracker 安装配置在同一个主控服务器节点上。部署 NameNode 和 JobTracker 的机器是 Master（主服务器），其余的机器都是 Slaves（从服务器）。详细的安装和配制过程如下。

### 2.3.1 安装和配置 JDK

集群分布式 Hadoop 系统的安装首先也需要在每台机器上安装 JDK。和单机伪分布式一样，我们采用的版本为 jdk-6u23-linux-x64。在集群中的每台机器上安装 JDK，步骤如下：

- 1) 将 jdk-6u23-linux-x64-rpm.bin 拷贝到所需要的安装目录下，如 /usr/jdk。
- 2) 执行 ./jdk-6u23-linux-x64-rpm.bin 安装文件。
- 3) 配置 JAVA\_HOME 以及 CLASS\_PATH，vi 进入 /etc/profile，在文件最后加上如下语句：

```
JAVA_HOME=/usr/java/jdk1.6.0
PATH=$JAVA_HOME/bin:$PATH
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME PATH CLASSPATH
```

保存退出，执行 source /etc/profile 使得配置文件生效。

4) 执行 `java-version` 查看当前版本配置有没有生效。

5) 执行以下命令分别进行编译和执行, 查看结果是否正确。

```
$javac HelloWorld.java
$java HelloWorld
```

### 2.3.2 创建 Hadoop 用户

在所有机器上建立相同的用户名, 例如名为 “hadoop” 的用户名。这一步使用如下命令实现:

```
#useradd -m hadoop
#passwd hadoop
```

成功建立 Hadoop 用户后, 输入的密码就是该用户的密码。

### 2.3.3 下载安装 Hadoop

和单机伪分布式一样, 从 Apache Hadoop 发布页面 (<http://hadoop.apache.org/coases.html>) 下载一个稳定的发布包, 再解压缩到本地文件系统中。在这里采用的 Hadoop 版本是 Hadoop-1.2.1。

```
$tar -xzf hadoop-1.2.1.tar.gz
```

### 2.3.4 配置 SSH

该配置主要是为了实现在机器之间访问时免密码登录。在所有机器上建立 `.ssh` 目录, 执行如下命令:

```
$mkdir .ssh
```

在 NameNode 上生成密钥对, 执行如下命令:

```
$ssh-keygen -t rsa
```

然后一直按 <Enter> 键, 就会按照默认选项将生成的密钥对保存在 `.ssh/id_rsa` 中。接着执行如下命令:

```
$cd ~/.ssh
$cp id_rsa.pub authorized_keys
$scp authorized_keys datanode1:/home/hadoop/.ssh
$scp authorized_keys datanode2:/home/hadoop/.ssh
```

最后进入所有机器的 `.ssh` 目录, 改变 `authorized_keys` 文件的许可权限:

```
$chmod 644 authorized_keys
```

这时从 NameNode 向其他机器发起 SSH 连接, 只要在第一次登录时需要输入密码, 以后

则不再需要输入密码。

### 2.3.5 配置 Hadoop 环境

要在所有机器上配置 Hadoop，首先在 NameNode 上进行配置，执行如下的解压缩命令：

```
$tar -xzvf /home/hadoop/hadoop-1.2.1.tar.gz
```

Hadoop 的配置文件主要存放在 hadoop 安装目录下的 conf 目录中，主要有以下几个配置文件要修改：

- conf/hadoop-env.sh：Hadoop 环境变量设置。
- conf/core-site.xml：主要完成 NameNode 的 IP 和端口设置。
- conf/hdfs-site.xml：主要完成 HDFS 的数据块副本等参数设置。
- conf/mapred-site.xml：主要完成 JobTracker IP 和端口设置。
- conf/masters：完成 Master 节点 IP 设置。
- conf/slaves：完成 Slaves 节点 IP 设置。

#### 1. 编辑 core-site.xml、hdfs-site.xml 和 mapred-site.xml

core-site.xml 的文档内容如下所示：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/tmp/hadoop/hadoop-${user.name}</value>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://192.168.1.253:9000</value>
    <!--注意这里要填写自己的IP-->
  </property>
</configuration>
```

hdfs-site.xml 的文档内容如下所示：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/home/hadoop/hadoop_dir/dfs/name</value>
  </property>
```

```

<property>
<name>dfs.datanode.data.dir</name>
<value>file:///home/hadoop/hadoop_dir/dfs/data</value>
</property>
<property>
<name>dfs.replication</name>
<!-- 副本数根据集群中 Slave 节点的数目而定，一般小于 Slave 节点数 -->
<value>1</value>
</property>
</configuration>

```

mapred-site.xml 的文档内容如下所示：

```

<?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>192.168.1.253:9001</value>
<!--注意这里要填写自己的 IP-->
</property>
<property>
<name>mapreduce.cluster.local.dir</name>
<value>/home/hadoop/hadoop_dir/mapred/local</value>
</property>
<property>
<name>mapreduce.jobtracker.system.dir</name>
<value>/home/hadoop/hadoop_dir/mapred/system</value>
</property>
</configuration>

```

## 2. 编辑 conf/masters

修改 conf/masters 文件为 Master 的主机名，每个主机名一行，此处即为 NameNode。

## 3. 编辑 conf/slaves

加入所有 Slaves 的主机名，即 datanode1 和 datanode2。

## 4. 把 Hadoop 安装文件复制到其他节点上

要把 Hadoop 安装文件复制到其他节点上，需要执行如下命令：

```

$scp -r hadoop-1.2.1 datanode1:/home/hadoop
$scp -r hadoop-1.2.1 datanode2:/home/hadoop

```

## 5. 编辑所有机器的 conf/hadoop-env.sh

将 JAVA\_HOME 变量设置为各自的 Java 安装的根目录。

至此，Hadoop 已经在集群上部署完毕。如果要新加入或删除节点，仅需修改所有节点的 master 和 slaves 配置文件。

## 2.3.6 Hadoop 的运行

### 1. 格式化 HDFS 文件系统

在初次安装和使用 Hadoop 之前，需要格式化分布式文件系统 HDFS，操作命令如下：

```
$bin/hadoop namenode -format
```

### 2. 启动 Hadoop 环境

启动 Hadoop 守护进程。在 NameNode 上启动 NameNode、JobTracker 和 Secondary NameNode，在 datanode1 和 datanode2 上启动 DataNode 和 TaskTracker，并用如下 jps 命令检测启动情况：

```
$bin/start-all.sh  
$jps
```

NameNode 节点上启动正常结果如下所示：

```
$jps  
14730 SecondaryNameNode  
15099 Jps  
14375 NameNode  
14825 JobTracker
```

用户也可以根据自己的需要来执行如下命令：

1) start-all.sh：启动所有的 Hadoop 守护进程，包括 NameNode、DataNode、JobTracker 和 TaskTracker。

2) stop-all.sh：停止所有的 Hadoop 守护进程。

3) start-mapred.sh：启动 Map/Reduce 守护进程，包括 JobTracker 和 TaskTracker。

4) stop-mapred.sh：停止 Map/Reduce 守护进程。

5) start-dfs.sh：启动 Hadoop DFS 守护进程，包括 NameNode 和 DataNode。

6) stop-dfs.sh：停止 Hadoop DFS 守护进程。

要停止 Hadoop 守护进程，可以使用下面的命令：

```
$bin/stop-all.sh
```

## 2.3.7 运行测试程序

下面用一个程序测试能否运行任务，示例程序是一个 Hadoop 自带的 PI 值的计算。第一



个参数是指要运行的 map 的次数，第二个参数是指每个 map 任务取样的个数。

```
$hadoop jar $HADOOP_HOME/hadoop-examples-0.20.205.0.jar \pi2 5
```

### 2.3.8 查看集群状态

当 Hadoop 启动之后，可以用 jps 命令查看一下它是不是正常启动。在 NameNode 节点上输入 jps 命令：

```
$jps
4706 JobTracker
4582 SecondaryNameNode
4278 NameNode
4889 Jps
```

在 DataNode 节点上输入 jps 命令：

```
$jps
4413 DataNode
4853 TaskTracker
4889 Jps
```

如果显示以上的信息，则 Hadoop 已表示正常启动。

## 2.4 Hadoop MapReduce 程序开发过程

Hadoop MapReduce 程序的开发一般是在程序员本地的单机 Hadoop 系统上进行程序设计与调试，然后上载到 Hadoop 集群上运行。开发环境可以使用 Eclipse，也可以使用其他开发环境，如 IntelliJ<sup>①</sup>。本节仅仅介绍使用 Eclipse 开发 Hadoop 程序的过程。

Eclipse 是一个开源的软件集成开发环境（IDE），可以提供对 Java 应用的编程开发所需要的完整工具平台。Eclipse 官方网站：<http://www.eclipse.org/>。

可以下载 Linux 版本的 Eclipse IDE for Java 开发包，并安装在本地的 Linux 系统中。

### 1. 启动 Eclipse

启动 Eclipse 后，会出现如图 2-2 所示的界面

### 2. 创建 Java Project

创建 Java Project 的界面如图 2-3 所示。

---

① IntelliJ 官方网站：<http://www.jetbrains.com/idea/>

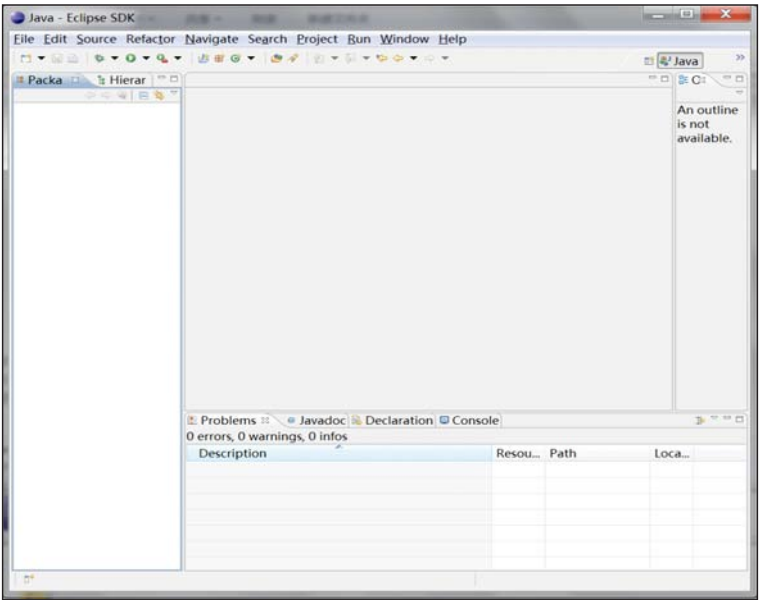


图 2-2 启动 Eclipse

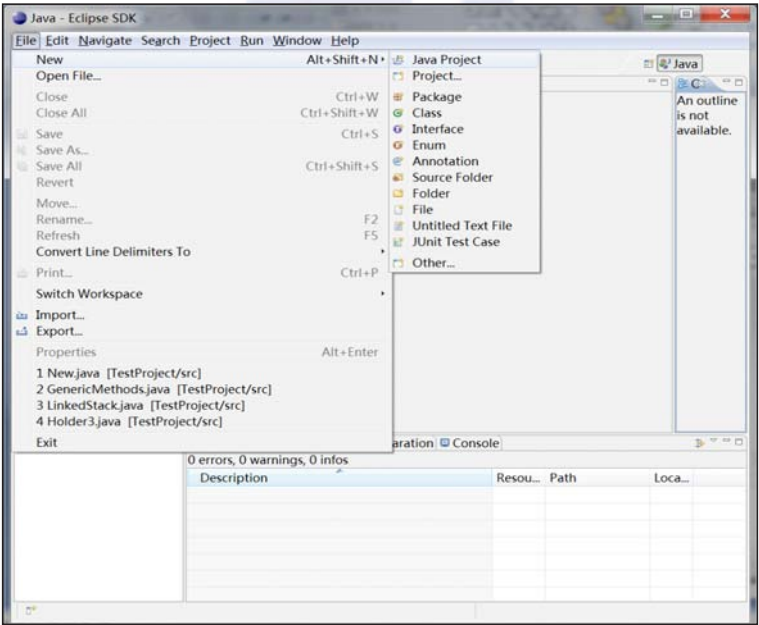


图 2-3 创建 Java Project

### 3. 配置 Java Project

这一步需要加入外部的 jar 文件：hadoop-core-1.2.1.jar 以及 lib 下所有的 jar 包，见图 2-4。

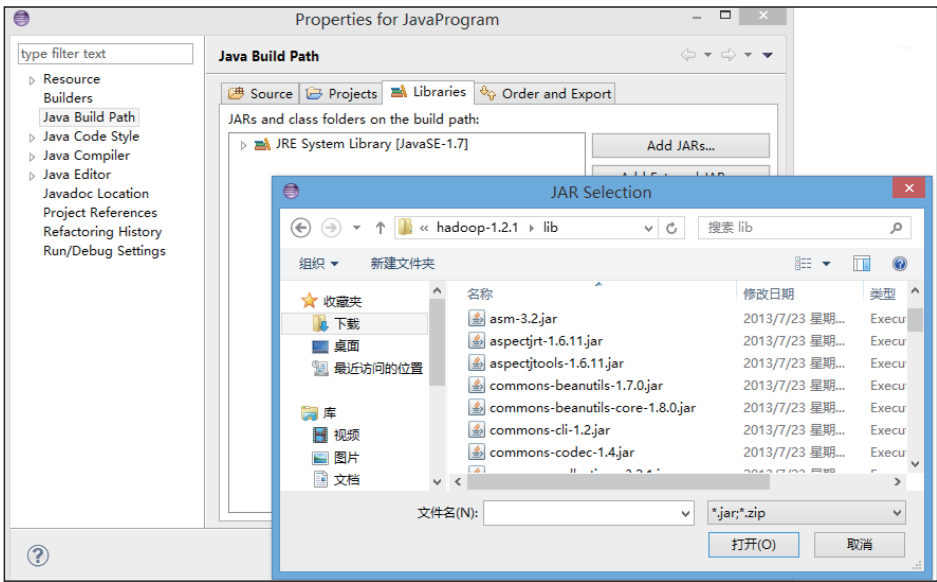


图 2-4 加入相应 jar 包

#### 4. 编写程序代码

编写相应的 MapReduce 程序的代码，见图 2-5。

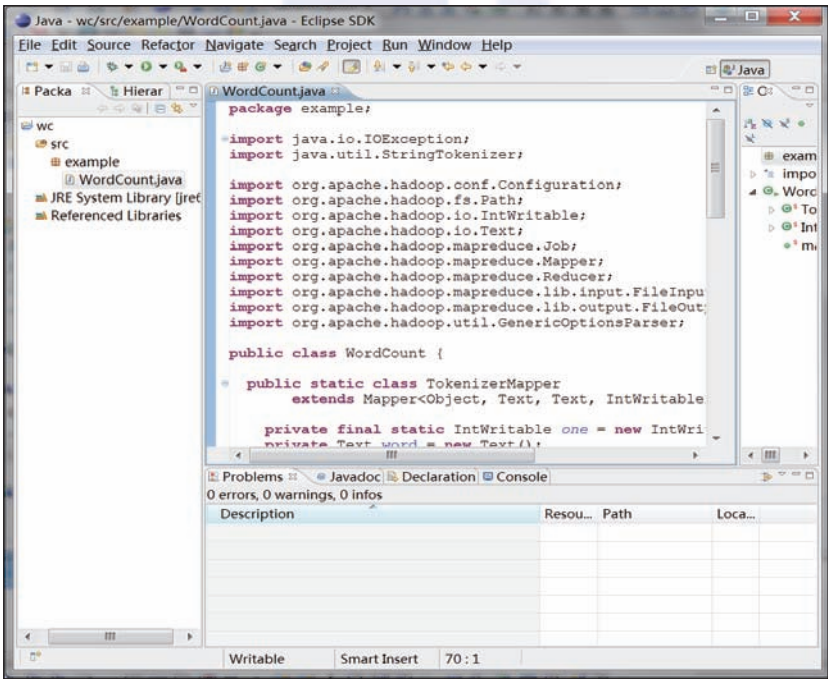


图 2-5 编写程序代码

## 5. 编译源代码

编译 MapReduce 程序。待完成编译时，导出 jar 文件，如图 2-6 所示。

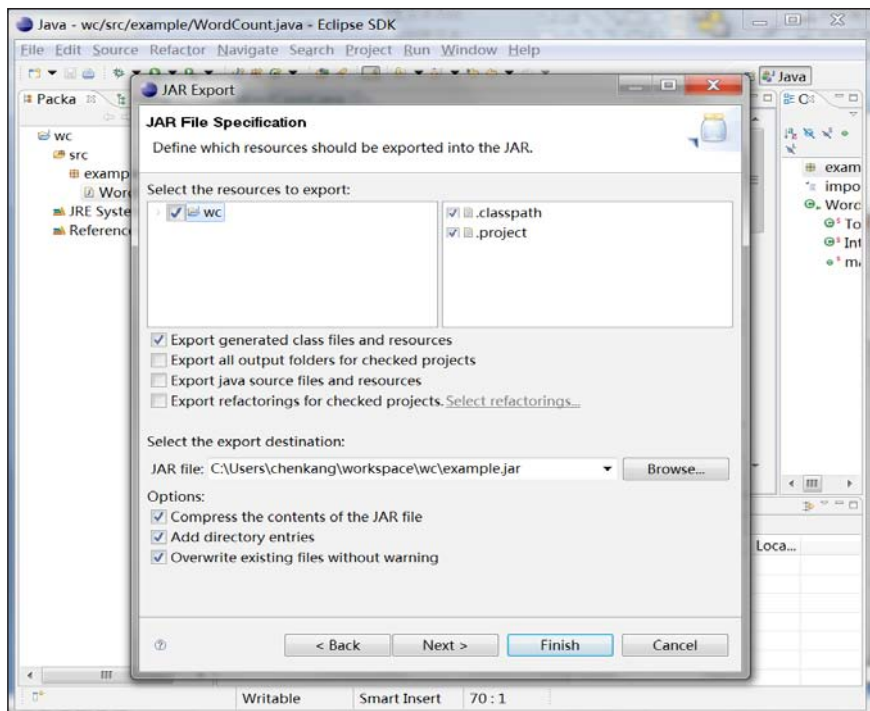


图 2-6 编译源代码

## 6. 本地运行调试

在导出 jar 文件的时候，需要指定一个主类 Main Class，作为默认执行的一个类。将程序复制到本地 Hadoop 系统的执行目录，可以准备一个小的测试数据，即可通过 Hadoop 的安装包进行运行调试。

## 7. 远程作业提交

当需要用集群进行海量数据处理时，在本地程序调试正确运行后，可按照远程作业提交步骤，将作业提交到远程集群上运行。

以 Hadoop MapReduce 计算 PI 值的示例程序为例，运行程序的命令是：

```
$hadoop jar $HADOOP_HOME/hadoop-examples-1.2.1.jar \pi2 5
```

其中，第一个参数是指要运行的 map 的次数；第二个参数是指每个 map 任务取样的个数。

## 2.5 集群远程作业提交与执行

### 2.5.1 集群远程作业提交和执行过程

Hadoop 程序开发与作业提交的基本过程如图 2-7 所示。

具体可分为以下 2 个步骤：

#### (1) 在本地完成程序编写和调试

在自己本地安装了单机分布式或单机伪分布式 Hadoop 系统的机器上，完成程序编写和调试工作。

#### (2) 创建用户账户

为了能访问 Hadoop 集群提交作业，需要为每个程序用户创建一个账户，获取用户名、密码等信息。

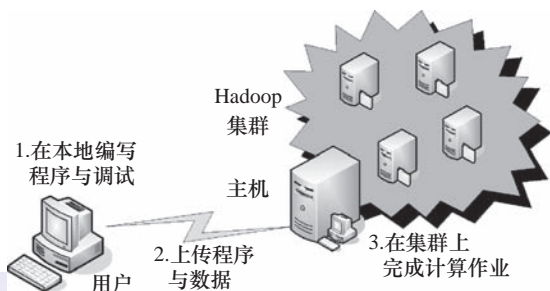


图 2-7 Hadoop 程序开发与作业提交的基本过程

1) 将数据和程序传送到 Hadoop 集群，准备好数据和程序目录，用 scp 命令传送到 Hadoop 平台主机上。

2) 用 SSH 命令远程登录到 Hadoop 集群。

3) 将数据复制到 HDFS 中，进入到程序包所在的目录，用 `hadoop dfs-put` 命令将数据从 Linux 文件系统中复制到 HDFS 中。

4) 用 `hadoop jar` 命令向 Hadoop 提交计算作业。在这里需要注意，如果程序中涉及到 HDFS 的输出目录，这些目录事先不能存在，若存在，需要先删除。

### 2.5.2 查看作业执行结果和集群状态

#### 1. 查看作业运行结果

查阅 HDFS 中的目录，查看计算结果，如图 2-8 所示。也可以将文件从 HDFS 中复制到 Linux 文件系统中查看。

<a href="#">Go to parent directory</a>								
Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
<a href="#">_SUCCESS</a>	file	0 KB	3	64 MB	2013-10-03 13:55	rw-r--r--	hadoop	supergroup
<a href="#">_logs</a>	dir				2013-10-03 13:54	rxr-xr-x	hadoop	supergroup
<a href="#">part-m-00000</a>	file	1.29 MB	3	64 MB	2013-10-03 13:54	rw-r--r--	hadoop	supergroup

图 2-8 HDFS 下的任务计算结果



2. 用 Hadoop 的 Web 界面查看 Hadoop 集群和作业状态

在浏览器中打开 http://NameNode 节点 IP : 50070/ 可以看到集群的基本信息，如图 2-9 所示。

### NameNode 'master:54310'

**Started:** Tue Nov 12 10:00:15 CST 2013  
**Version:** 1.0.4-SNAPSHOT, r5a196ab874ef147afe4aa8403a7b7f73436b3191  
**Compiled:** Wed Dec 19 17:18:25 CST 2012 by labuser01  
**Upgrades:** There are no upgrades in progress.

**Browse the filesystem**  
[Namenode Logs](#)

#### Cluster Summary

5582 files and directories, 9541 blocks = 15123 total. Heap Size is 30.19 MB / 888.94 MB (3%)

Configured Capacity	: 24.18 TB
DFS Used	: 1.14 TB
Non DFS Used	: 7.31 TB
DFS Remaining	: 15.72 TB
DFS Used%	: 4.72 %
DFS Remaining%	: 65.03 %
Live Nodes	: 14
Dead Nodes	: 3
Decommissioning Nodes	: 0
Number of Under-Replicated Blocks	: 0

#### NameNode Storage:

Storage Directory	Type	State
/home/hadoop/hadoop_ins_test/hadoop-1.0.3/hadoop_dir/dfs/name	IMAGE_AND_EDITS	Active

This is [Apache Hadoop](#) release 1.0.4-SNAPSHOT

图 2-9 通过 Hadoop 查看集群的基本信息

相应地，在浏览器中打开 http://NameNode 节点 IP : 50030/ 可以看到集群上的任务执行情况，如图 2-10 所示。

点击一个作业可以查看作业的详细信息，如图 2-11 所示。

### Running Jobs

none

### Completed Jobs

JobId	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201311121000_0001	NORMAL	hadoop	PIEstimator	100.00%	4	4	100.00%	1	1	NA	NA

### Retired Jobs

none

### Local Logs

[Log directory, Job Tracker History](#)

This is [Apache Hadoop](#) release 1.0.4-SNAPSHOT

图 2-10 查看集群上的任务执行情况

Hadoop job_201311121000_0001 on master							
<b>User:</b> hadoop							
<b>Job Name:</b> PiEstimator							
<b>Job File:</b> <a href="hdfs://192.168.1.254:54310/user/hadoop/mapred/staging/hadoop/.staging/job_201311121000_0001/job.xml">hdfs://192.168.1.254:54310/user/hadoop/mapred/staging/hadoop/.staging/job_201311121000_0001/job.xml</a>							
<b>Submit Host:</b> master							
<b>Submit Host Address:</b> 192.168.1.254							
<b>Job-ACLs:</b> All users are allowed							
<b>Job Setup:</b> Pending							
<b>Status:</b> Succeeded							
<b>Started at:</b> Sun Nov 17 16:33:29 CST 2013							
<b>Finished at:</b> Sun Nov 17 16:33:45 CST 2013							
<b>Finished in:</b> 15sec							
<b>Job Cleanup:</b> Pending							
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	4	0	0	4	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

图 2-11 查看作业执行的详细信息

# 大数据存储——分布式文件系统 HDFS

大数据处理面临的第一个问题是，如何有效存储规模巨大的数据？对于大数据处理应用来说，依靠集中式的物理服务器来保存数据是不现实的，容量也好，数据传输速度也好，都会成为瓶颈。要实现大数据的存储，需要使用几十台、几百台甚至更多的分布式服务器节点。为了统一管理这些节点上存储的数据，必须要使用一种特殊的文件系统——分布式文件系统。为了提供可扩展的大数据存储能力，Hadoop 设计提供了一个分布式文件系统 HDFS（Hadoop Distributed File System）。

本章首先简要介绍 HDFS 的基本特征、基本构架、工作过程，以及 HDFS 的可靠性设计和数据存储及访问方法，在此基础上进一步介绍 HDFS 的文件操作命令和 HDFS 的编程接口和编程示例。

## 3.1 HDFS 的基本特征与构架

HDFS 被设计成在普通的商用服务器节点构成的集群上即可运行，它和已有的分布式文件系统有很多相似的地方。但是，HDFS 在某些重要的方面，具有有别于其他系统的独特优点。这个特殊的文件系统具有相当强大的容错能力，保证其在成本低廉的普通商用服务器上也能很好地运行；进一步，HDFS 可以提供很高的数据吞吐能力，这对于那些需要大数据处理的应用来说是一项非常重要的技术特征；另外，HDFS 可以采用流式访问的方式读写数据，在编程方式上，除了 API 的名称不一样以外，通过 HDFS 读写文件和通过本地文件系统读写文件在代码上基本类似，因而非常易于编程使用。

### 3.1.1 HDFS 的基本特征

HDFS 具有下列六种基本特征。

#### (1) 大规模数据分布存储能力

HDFS 以分布存储方式和良好的可扩展性提供了大规模数据的存储能力，可基于大量分布节点上的本地文件系统，构建一个逻辑上具有巨大容量的分布式文件系统，并且整个文件系统的容量可随集群中节点的增加而线性扩展。HDFS 不仅可存储 GB 级到 TB 级别大小的单个文件，还可以支持在一个文件系统中存储高达数千万量级的文件数量。这种分布式文件系统为上层的大数据处理应用程序提供了完全透明的数据存储和访问功能支撑，使得应用程序完全感觉不到其数据在物理上是分布存储在一组不同机器上的。

#### (2) 高并发访问能力

HDFS 以多节点并发访问方式提供很高的数据访问带宽（高数据吞吐率），并且可以把带宽的大小等比例扩展到集群中的全部节点上。

#### (3) 强大的容错能力

在 HDFS 的设计理念中，硬件故障被视作是一个常态。因此，HDFS 的设计思路保证了系统能在经常有节点发生硬件故障的情况下正确检测硬件故障，并且能自动从故障中快速恢复，确保数据不丢失。为此，HDFS 采用多副本数据块形式存储（默认副本数目是 3），按照块的方式随机选择存储节点。

#### (4) 顺序式文件访问

大数据批处理在大多数情况下都是大量简单数据记录的顺序处理。针对这个特性，为了提高大规模数据访问的效率，HDFS 对顺序读进行了优化，支持大量数据的快速顺序读出，代价是对于随机的访问负载较高。

#### (5) 简单的一致性模型（一次写多次读）

HDFS 采用了简单的“一次写多次读”模式访问文件，支持大量数据的一次写入、多次读取；不支持已写入数据的更新操作，但允许在文件尾部添加新的数据。

#### (6) 数据块存储模式

与常规的文件系统不同，HDFS 采用基于大粒度数据块的方式存储文件，默认的块大小是 64MB，这样做的好处是可以减少元数据的数量，并且可以允许将这些数据块通过随机方式选择节点，分布存储在不同的地方。

### 3.1.2 HDFS 的基本框架与工作过程

#### 1. 基本组成结构与文件访问过程

HDFS 是一个建立在一组分布式服务器节点的本地文件系统之上的分布式文件系统。HDFS

采用经典的主 - 从式结构，其基本组成结构如图 3-1 所示。

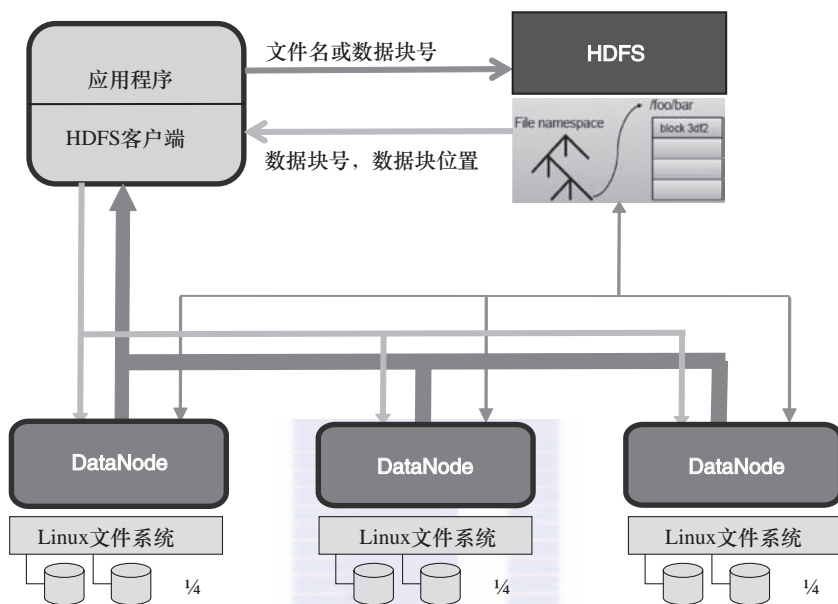


图 3-1 HDFS 的基本组成结构

一个 HDFS 文件系统包括一个主控节点 NameNode 和一组 DataNode 从节点。NameNode 是一个主服务器，用来管理整个文件系统的命名空间和元数据，以及处理来自外界的文件访问请求。NameNode 保存了文件系统的三种元数据：1) 命名空间，即整个分布式文件系统的目录结构；2) 数据块与文件名的映射表；3) 每个数据块副本的位置信息，每一个数据块默认有 3 个副本。

HDFS 对外提供了命名空间，让用户的数据可以存储在文件中，但是在内部，文件可能被分成若干个数据块。DataNode 用来实际存储和管理文件的数据块。文件中的每个数据块默认的大小为 64MB；同时为了防止数据丢失，每个数据块默认有 3 个副本，且 3 个副本会分别复制在不同的节点上，以避免一个节点失效造成一个数据块的彻底丢失。

每个 DataNode 的数据实际上是存储在每个节点的本地 Linux 文件系统中。

在 NameNode 上可以执行文件操作，比如打开、关闭、重命名等；而且 NameNode 也负责向 DataNode 分配数据块并建立数据块和 DataNode 的对应关系。DataNode 负责处理文件系统用户具体的数据读写请求，同时也可以处理 NameNode 对数据块的创建、删除副本的指令。

NameNode 和 DataNode 对应的程序可以运行在廉价的普通商用服务器上。这些机器一般都运行着 GNU/Linux 操作系统。HDFS 由 Java 语言编写，支持 JVM 的机器都可以



运行 NameNode 和 DataNode 对应的程序。虽然一般情况下是 GNU/Linux 系统，但是因为 Java 的可移植性，HDFS 也可以运行在很多其他平台之上。一个典型的 HDFS 部署情况是：NameNode 程序单独运行于一台服务器节点上，其余的服务器节点，每一台运行一个 DataNode 程序。

在一个集群中采用单一的 NameNode 可以大大简化系统的架构。另外，虽然 NameNode 是所有 HDFS 的元数据的唯一所有者，但是，程序访问文件时，实际的文件数据流并不会通过 NameNode 传送，而是从 NameNode 获得所需访问数据块的存储位置信息后，直接去访问对应的 DataNode 获取数据。这样设计有两点好处：一是可以允许一个文件的数据能同时在不同 DataNode 上并发访问，提高数据访问的速度；二是可以大大减少 NameNode 的负担，避免使得 NameNode 成为数据访问瓶颈。

HDFS 的基本文件访问过程是：

- 1) 首先，用户的应用程序通过 HDFS 的客户端程序将文件名发送至 NameNode。
- 2) NameNode 接收到文件名之后，在 HDFS 目录中检索文件名对应的数据块，再根据数据块信息找到保存数据块的 DataNode 地址，将这些地址回送给客户端。
- 3) 客户端接收到这些 DataNode 地址之后，与这些 DataNode 并行地进行数据传输操作，同时将操作结果的相关日志（比如是否成功，修改后的数据块信息等）提交到 NameNode。

## 2. 数据块

为了提高硬盘的效率，文件系统中最小的数据读写单位不是字节，而是一个更大的概念——数据块。但是，数据块的信息对于用户来说是透明的，除非通过特殊的工具，否则很难看到具体的数据块信息。

HDFS 同样也有数据块的概念。但是，与一般文件系统中大小为若干 KB 的数据块不同，HDFS 数据块的默认大小是 64MB，而且在不少实际部署中，HDFS 的数据块甚至会被设置成 128MB 甚至更多，比起文件系统上几个 KB 的数据块，大了几千倍。

将数据块设置成这么大的原因是减少寻址开销的时间。在 HDFS 中，当应用发起数据传输请求时，NameNode 会首先检索文件对应的数据块信息，找到数据块对应的 DataNode；DataNode 则根据数据块信息在自身的存储中寻找相应的文件，进而与应用程序之间交换数据。因为检索的过程都是单机运行，所以要增加数据块大小，这样就可以减少寻址的频度和时间开销。

## 3. 命名空间

HDFS 中的文件命名遵循了传统的“目录 / 子目录 / 文件”格式。通过命令行或者是 API 可以创建目录，并且将文件保存在目录中；也可以对文件进行创建、删除、重命名操作。不

过，HDFS 中不允许使用链接（硬链接和符号链接都不允许）。命名空间由 NameNode 管理，所有对命名空间的改动（包括创建、删除、重命名，或是改变属性等，但是不包括打开、读取、写入数据）都会被 HDFS 记录下来。

HDFS 允许用户配置文件在 HDFS 上保存的副本数量，保存的副本数称作“副本因子”（Replication Factor），这个信息也保存在 NameNode 中。

#### 4. 通信协议

作为一个分布式文件系统，HDFS 中大部分的数据都是通过网络进行传输的。为了保证传输的可靠性，HDFS 采用 TCP 协议作为底层的支撑协议。应用可以向 NameNode 主动发起 TCP 连接。应用和 NameNode 交互的协议称为 Client 协议，NameNode 和 DataNode 交互的协议称为 DataNode 协议（这些协议的具体内容请参考其他资料）。而用户和 DataNode 的交互是通过发起远程过程调用（Remote Procedure Call，RPC）、并由 NameNode 响应来完成的。另外，NameNode 不会主动发起远程过程调用请求。

#### 5. 客户端

严格来讲，客户端并不能算是 HDFS 的一部分，但是客户端是用户和 HDFS 通信最常见也是最方便的渠道，而且部署的 HDFS 都会提供客户端。

客户端为用户提供了一种可以通过与 Linux 中的 Shell 类似的方式访问 HDFS 的数据。客户端支持最常见的操作如（打开、读取、写入等）；而且命令的格式也和 Shell 十分相似，大大方便了程序员和管理员的操作。具体的命令行操作详见 3.4 节。

除了命令行客户端以外，HDFS 还提供了应用程序开发时访问文件系统的客户端编程接口，具体的 HDFS 编程接口详见 3.5 节。

### 3.2 HDFS 可靠性设计

Hadoop 能得到如此广泛的应用，和背后默默支持它的 HDFS 是分不开的。作为一个能在成百上千个节点上运行的文件系统，HDFS 在可靠性设计上做了非常周密的考虑。

#### 3.2.1 HDFS 数据块多副本存储设计

作为一个分布式文件系统，HDFS 采用了在系统中保存多个副本的方式保存数据（以下简称多副本），且同一个数据块的多个副本会存放在不同节点上，如图 3-2 所示。采用这种多副本方式有以下几个优点：1）采用多副本，可以让客户从不同的数据块中读取数据，加快传输速度；2）因为 HDFS 的 DataNode 之间通过网络传输数据，如果采用多个副本可以判断数据传输是否出错；3）多副本可以保证某个 DataNode 失效的情况下，不会丢失数据。

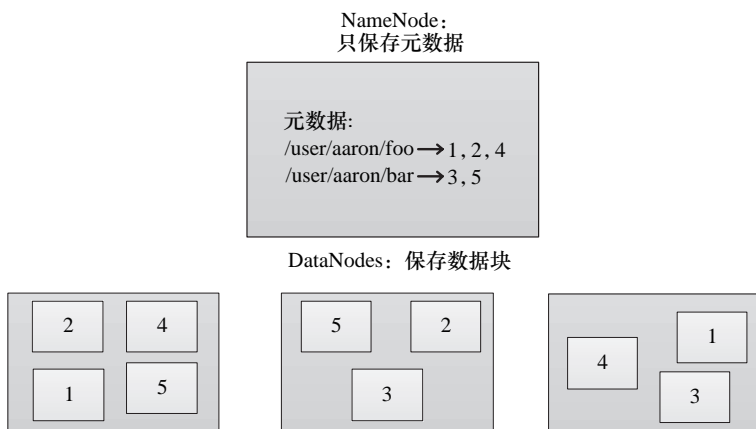


图 3-2 HDFS 数据块多副本存储

HDFS 按照块的方式随机选择存储节点，为了可以判断文件是否出错，副本个数默认为 3（注：如果副本个数为 1 或 2 的话，是不能判断数据对错的）。出于数据传输代价以及错误恢复等多方面的考虑，副本的保存并不是均匀分布在集群之中的，关于副本保存分布和维持 DataNode 负载均衡的更详细的内容，可以参考后面 3.4.4 节关于 balancer 的介绍。

### 3.2.2 HDFS 可靠性的设计实现

#### 1. 安全模式

HDFS 刚刚启动时，NameNode 会进入安全模式（safe mode）。处于安全模式的 NameNode 不能做任何的文件操作，甚至内部的副本创建也是不允许的。NameNode 此时需要和各个 DataNode 通信，获得 DataNode 保存的数据块信息，并对数据块信息进行检查。只有通过了 NameNode 的检查，一个数据块才被认为是安全的。当认为安全的数据块所占的比例达到了某个阈值（可配置），NameNode 才会退出。

#### 2. SecondaryNameNode

Hadoop 中使用 SecondaryNameNode 来备份 NameNode 的元数据，以便在 NameNode 失效时能从 SecondaryNameNode 恢复出 NameNode 上的元数据。SecondaryNameNode 充当 NameNode 的一个副本，它本身并不处理任何请求，因为处理这些请求都是 NameNode 的责任。

NameNode 中保存了整个文件系统的元数据，而 SecondaryNameNode 的作用就是周期性（周期的长短也是可以配置的）保存 NameNode 的元数据。这些元数据中包括文件镜像数据 FsImage 和编辑日志数据 EditLog。FsImage 相当于 HDFS 的检查点，NameNode 启动时会读取 FsImage 的内容到内存，并将其与 EditLog 日志中的所有修改信息合并生成新的 FsImage；在 NameNode 运行过程中，所有关于 HDFS 的修改都将写入 EditLog。这样，如果

NameNode 失效，可以通过 Secondary NameNode 中保存的 FsImage 和 EditLog 数据恢复出 NameNode 最近的状态，尽量减少损失。

### 3. 心跳包 (HeartBeats) 和副本重新创建 (re-replication)

如果 HDFS 运行过程中，一部分 DataNode 因为崩溃或是掉线等原因，离开了 HDFS 系统，怎么办？为了保证 NameNode 和各个 DataNode 的联系，HDFS 采用了心跳包 (Heart beat) 机制。位于整个 HDFS 核心的 NameNode，通过周期性的活动来检查 DataNode 的活性，就像跳动的心脏一样，所以，这里把这些包就叫做心跳包。NameNode 周期性向管理的各个 DataNode 发送心跳包，而收到心跳包的 DataNode 则需要回复。因为心跳包总是定时发送的，所以 NameNode 就把要执行的命令也通过心跳包发送给 DataNode，而 DataNode 收到心跳包，一方面回复 NameNode，另一方面就开始了与用户或者应用的数据传输。

如果侦测到了 DataNode 失效，那么之前保存在这个 DataNode 上的数据就变成不可用的。那么，如果有的副本存储在失效的 DataNode 上，则需要重新创建这个副本，放到另外可用的地方。其他需要创建副本的情况包括数据块校验失败等。

### 4. 数据一致性

一般来讲，DataNode 与应用数据交互的大部分情况都是通过网络进行的，而网络数据传输带来的一大问题就是数据是否能原样到达。为了保证数据的一致性，HDFS 采用了数据校验和 (Checksum) 机制。创建文件时，HDFS 会为此文件生成一个校验和，校验和文件和文件本身保存在同一空间中。传输数据时会将数据与校验和一起传输，应用收到数据后可以进行校验，如果两个校验的结果不同，则文件肯定出错了，这个数据块就变成了无效的。如果判定数据无效，就需要从其他 DataNode 上读取副本。

### 5. 租约

在 Linux 中，为了防止出现多个进程向同一个文件写数据的情况，采用了文件加锁的机制。而在 HDFS 中，同样也需要一种机制来防止同一个文件被多个人写入数据。这种机制就是租约 (Lease)。每当写入文件之前，一个客户端必须要获得 NameNode 发放的一个租约。NameNode 保证同一个文件只会发放一个允许写的租约，那么就可以有效防止出现多人写入的情况。

不过，租约的作用不止于此。如果 NameNode 发放租约之后崩溃了，怎么办？或者如果客户端获得租约之后崩溃了，又怎么办？第一个问题可以通过前面提到的恢复机制解决。而第二个问题，则通过在租约中加入时间限制来解决。每当租约要到期时，客户端需要向 NameNode 申请更新租约，NameNode “审核”之后，重新发放租约。如果客户端不申请，那就说明客户端不需要读写这一文件或者已经崩溃了，NameNode 收回租约即可。

## 6. 回滚

HDFS 与 Hadoop 一样处于发展阶段。而某个升级可能会导致 BUG 或者不兼容的问题，这些问题还可能导致现有的应用运行出错。这一问题可以通过回滚回到旧版本解决。HDFS 安装或者升级时，会将当前的版本信息保存起来，如果升级之后一段时间内运行正常，可以认为这次升级没有问题，重新保存版本信息，否则，根据保存的旧版本信息，将 HDFS 恢复至之前的版本。

## 3.3 HDFS 文件存储组织与读写

作为一个分布式文件系统，HDFS 内部的数据与文件存储机制、读写过程与普通的本地文件系统有较大的差别。下面具体介绍 HDFS 中数据的存储组织和读写过程。

### 3.3.1 文件数据的存储组织

如前所述，HDFS 中最主要的部分就是 NameNode 和 DataNode。NameNode 存储了所有文件元数据、文件与数据块的映射关系，以及文件属性等核心数据，DataNode 则存储了具体的数据块。那么，在 HDFS 中，具体的文件存储组织结构是怎样的呢？

#### 1. NameNode 目录结构

图 3-3 是 NameNode 的目录结构和内容。NameNode 借助本地文件系统来保存数据，保存的文件夹位置由配置选项 `{dfs.name.dir}` 决定（未配置此选项，则为 hadoop 安装目录下的 `/tmp/dfs/name`），所以，这里我们以 `${dfs.name.dir}` 代表 NameNode 节点管理的根目录。目录下的文件和子目录则以 `${dfs.name.dir}/file` 和 `${dfs.name.dir}/subdir` 的格式表示。

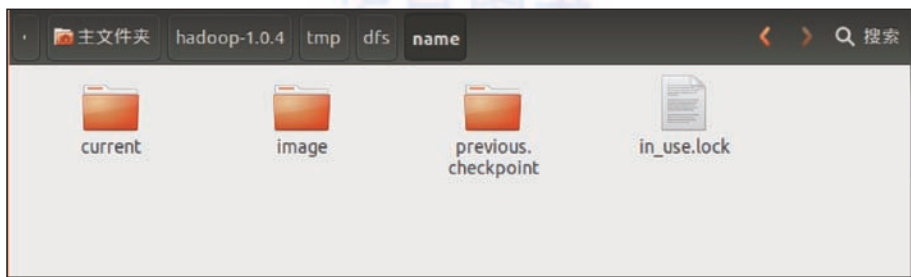


图 3-3 HDFS NameNode 目录结构

在 NameNode 的 `${dfs.name.dir}` 之下有 3 个文件夹和 1 个文件：

1) `current` 目录：主要包含如下的内容和结构：

- a) 文件 `VERSION`：保存了当前运行的 HDFS 版本信息。
- b) `FsImage`：是整个系统的空间镜像文件。
- c) `Edit`：`EditLog` 编辑日志。



d) Fstime: 上一次检查点的时间。

2) previous.checkpoint 目录: 和 current 内容结构一致, 不同之处在于, 此目录保存的是上一次检查点的内容。

3) image 目录: 旧版本 (版本 <0.13) 的 FsImage 存储位置。

4) in\_use.lock : NameNode 锁, 只有在 NameNode 有效 (启动并且能和 DataNode 正常交互) 时存在; 不满足上述情况时, 该文件不存在。这一文件具有“锁”的功能, 可以防止多个 NameNode 共享同一目录 (如果一个机器上只有一个 NameNode, 这也是最常见的情况, 那么这个文件基本不需要)。

## 2. DataNode 目录结构

图 3-4 是 DataNode 的目录结构和内容。DataNode 借助本地文件系统来保存数据, 一般情况下, 保存的文件夹位置由配置选项 {dfs.data.dir} 决定 (未配置此选项, 则为 hadoop 安装目录下的 /tmp/dfs/data)。所以, 这里我们以 \${dfs.data.dir} 代表 DataNode 节点管理的数据目录的根目录, 目录下的文件和子目录则以 \${dfs.data.dir}/file 和 \${dfs.data.dir}/subdir 的格式表示。

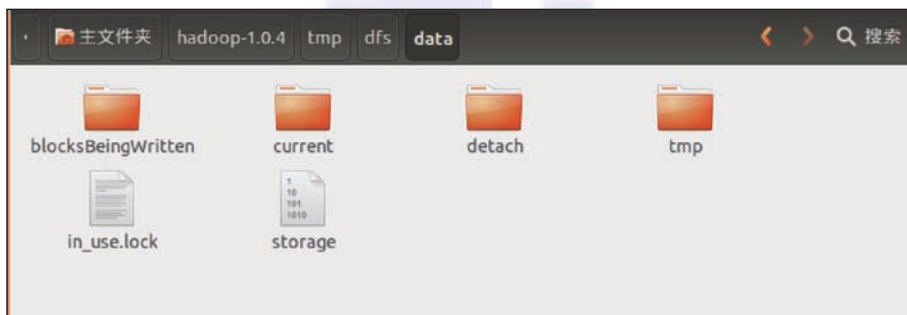


图 3-4 HDFS DataNode 目录结构

一般来说, 在 \${dfs.data.dir} 之下有 4 个子目录和 2 个文件:

1) current 目录: 已经成功写入的数据块, 以及一些系统需要的文件。包括以下内容:

a) 文件 VERSION: 保存了当前运行的 HDFS 版本信息。

b) Blk\_XXXXXX 和 Blk\_XXXXXX.Meta: 分别是数据块和数据块对应的元数据 (比如校验信息等)。

c) subdirXX: 当同一目录下文件数超过一定限制 (比如 64) 时, 会新建一个 subdir 目录, 保存多出来的数据块和元数据; 这样可以保证同一目录下目录 + 文件数不会太多, 可以提高搜索效率。

2) tmp 目录和 blocksBeingWritten 目录: 正在写入的数据块, tmp 目录保存的是用户操作引发的写入操作对应的数据块, blocksBeingWritten 目录是 HDFS 系统内部副本创建时 (当



出现副本错误或者数量不够等情况时)引发的写入操作对应的数据块。

3) detach 目录: 用于 DataNode 升级。

4) storage 文件: 由于旧版本(版本 <0.13)的存储目录是 storage, 因此如果在新版本的 DataNode 中启动旧版的 HDFS, 会因为无法打开 storage 目录而启动失败, 这样可以防止因版本不同带来的风险。

5) in\_use.lock 文件: DataNode 锁, 只有在 DataNode 有效(启动并且能和 NameNode 正常交互)时存在; 不满足上述情况时, 该文件不存在。这一文件具有“锁”的功能, 可以防止多个 DataNode 共享同一目录(如果一个机器上只有一个 DataNode, 这也是最常见的情况, 那么这个文件基本不需要)。

### 3. CheckPointNode 目录结构

图 3-5 是 CheckPointNode 的目录结构和内容。CheckPointNode 和旧版本的 SecondaryNameNode 作用类似, 所以目录结构也十分相近。

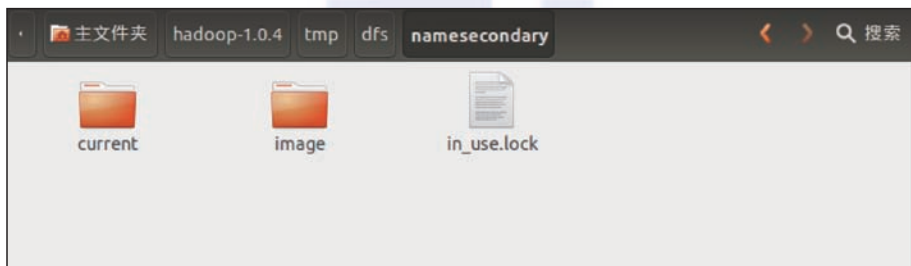


图 3-5 HDFS CheckPointNode 目录结构

CheckPointNode 借助本地文件系统来保存数据, 一般情况下, 保存的文件夹位置由配置选项 {dfs.checkpoint.dir} 决定(未配置此选项, 则为 hadoop 安装目录下的 /tmp/dfs/namesecondary)。所以, 这里我们以 \${dfs.checkpoint.dir} 代表 CheckPointNode 节点管理的数据目录的根目录, 目录下的文件和子目录则以 \${dfs.checkpoint.dir} 和 file, \${dfs.checkpoint.dir}/subdir 的格式表示。

CheckPointNode 目录下的文件和 NameNode 目录下的同名文件作用基本一致, 不同之处在于 CheckPointNode 保存的是自上一个检查点之后的临时镜像和日志。

### 3.3.2 数据的读写过程

数据的读写过程与数据的存储是紧密相关的, 以下介绍 HDFS 数据的读写过程。

#### 1. 数据读取过程

一般的文件读取操作包括 open、read、close 等, 具体可参见 3.5 节的 HDFS 编程接口介绍。这里介绍一下客户端连续调用 open、read、close 时, HDFS 内部的整个执行过程。图 3-6

可以帮助我们更好地理解这个过程。

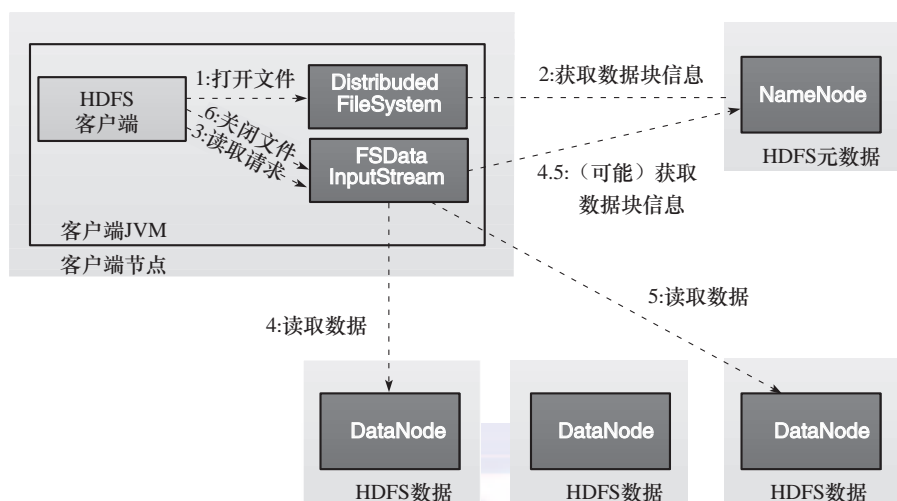


图 3-6 HDFS 数据读取过程

以下是客户端读取数据的过程，其中 1、3、6 步由客户端发起：

客户端首先要获取 FileSystem 的一个实例，这里就是 HDFS 对应的实例。

1) 首先，客户端调用 FileSystem 实例的 open 方法，获得这个文件对应的输入流，在 HDFS 中就是 DFSInputStream。

2) 构造第 1 步中的输入流 DFSInputStream 时，通过 RPC 远程调用 NameNode 可以获得 NameNode 中此文件对应的数据块保存位置，包括这个文件的副本的保存位置（主要是各 DataNode 的地址）。注意，在输入流中会按照网络拓扑结构，根据与客户端距离对 DataNode 进行简单排序。

3 ~ 4) 获得此输入流之后，客户端调用 read 方法读取数据。输入流 DFSInputStream 会根据前面的排序结果，选择最近的 DataNode 建立连接并读取数据。如果客户端和其中一个 DataNode 位于同一机器（比如 MapReduce 过程中的 mapper 和 reducer），那么就会直接从本地读取数据。

5) 如果已到达数据块末端，那么关闭与这个 DataNode 的连接，然后重新查找下一个数据块。

不断执行第 2 ~ 5 步直到数据全部读完，然后调用 close。

6) 客户端调用 close，关闭输入流 DFSInputStream。

另外，如果 DFSInputStream 和 DataNode 通信时出现错误，或者是数据校验出错，那么 DFSInputStream 就会重新选择 DataNode 传输数据。

## 2. 数据写入过程

一般的文件写入操作不外乎 create、write、close 几种，具体可参见 3.5 节的 HDFS 编程接口介绍。这里介绍一下客户端连续调用 create、write、close 时，HDFS 内部的整个执行过程，见图 3-7。

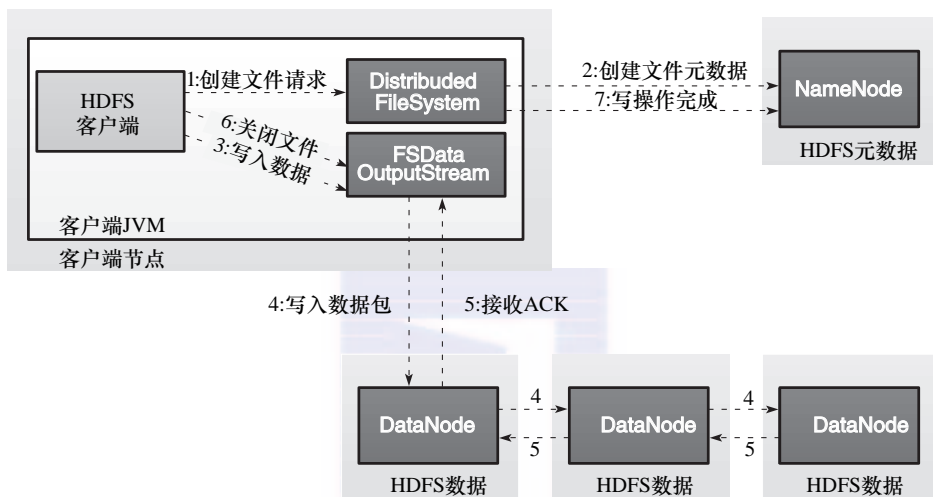


图 3-7 HDFS 数据写入过程

以下是客户端写入数据的过程，其中 1、3、6 步由客户端发起：

客户端首先要获取 FileSystem 的一个实例，这里就是 HDFS 对应的实例。

1 ~ 2) 客户端调用 FileSystem 实例的 create 方法，创建文件。NameNode 通过一些检查，比如文件是否存在，客户端是否拥有创建权限等；通过检查之后，在 NameNode 添加文件信息。注意，因为此时文件没有数据，所以 NameNode 上也没有文件数据块的信息。创建结束之后，HDFS 会返回一个输出流 DFSDDataOutputStream 给客户端。

3) 客户端调用输出流 DFSDDataOutputStream 的 write 方法向 HDFS 中对应的文件写入数据。数据首先会被分包，这些分包会写入一个输出流的内部队列 Data 队列中，接收完数据分包，输出流 DFSDDataOutputStream 会向 NameNode 申请保存文件和副本数据块的若干个 DataNode，这若干个 DataNode 会形成一个数据传输管道。

4) DFSDDataOutputStream 会（根据网络拓扑结构排序）将数据传输给距离上最短的 DataNode，这个 DataNode 接收到数据包之后会传给下一个 DataNode。数据在各 DataNode 之间通过管道流动，而不是全部由输出流分发，这样可以减少传输开销。

5) 因为各 DataNode 位于不同机器上，数据需要通过网络发送，所以，为了保证所有 DataNode 的数据都是准确的，接收到数据的 DataNode 要向发送者发送确认包（ACK

Packet)。对于某个数据块，只有当 DFSDDataOutputStream 收到了所有 DataNode 的正确 ACK，才能确认传输结束。DFSDDataOutputStream 内部专门维护了一个等待 ACK 队列，这一队列保存已经进入管道传输数据、但是并未被完全确认的数据包。

不断执行第 3 ~ 5 步直到数据全部写完，客户端调用 close 关闭文件。

6) 客户端调用 close 方法，DFSDDataInputStream 继续等待直到所有数据写入完毕并被确认，调用 complete 方法通知 NameNode 文件写入完成。

7) NameNode 接收到 complete 消息之后，等待相应数量的副本写入完毕后，告知客户端即可。

在传输数据的过程中，如果发现某个 DataNode 失效（未联通，ACK 超时），那么 HDFS 执行如下操作：

1) 关闭数据传输的管道。

2) 将等待 ACK 队列中的数据放到 Data 队列的头部。

3) 更新正常 DataNode 中所有数据块的版本；当失效的 DataNode 重启之后，之前的数据块会因为版本不对而被清除。

4) 在传输管道中删除失效的 DataNode，重新建立管道并发送数据包。

以上就是 HDFS 中数据读写的大致过程。

### 3.4 HDFS 文件系统操作命令

通过之前章节的学习，相信各位读者对 HDFS 已经有了一个基本的认识。在本小节里，我们来了解一下 HDFS 常用的基本操作命令。

#### 3.4.1 HDFS 启动与关闭

HDFS 和普通的硬盘上的文件系统不一样，是通过 Java 虚拟机运行在整个集群当中的，所以当 Hadoop 程序写好之后，需要启动 HDFS 文件系统，才能运行。

HDFS 启动过程如下：

1) 进入到 NameNode 对应节点的 Hadoop 安装目录下。

2) 执行启动脚本：

```
bin/start-dfs.sh
```

这一脚本会启动 NameNode，然后根据 conf/slaves 中的记录逐个启动 DataNode，最后根据 conf/masters 中记录的 Secondary NameNode 地址启动 SecondaryNameNode。

HDFS 关闭过程如下：

运行以下关闭脚本：

```
bin/stop-dfs.sh
```

这一脚本的运行过程正好是 bin/start-dfs.sh 的逆过程，关闭 Secondary NameNode，然后是每个 DataNode，最后是 NameNode 自身。

### 3.4.2 HDFS 文件操作命令格式与注意事项

HDFS 文件系统提供了相当多的 shell 操作命令，大大方便了程序员和系统管理人员查看、修改 HDFS 上的文件。进一步，HDFS 的操作命令和 Unix/Linux 的命令名称和格式相当一致，因而学习 HDFS 命令的成本也大为缩小。

HDFS 的基本命令格式如下：

```
bin/hadoop dfs-cmd <args>
```

这里 cmd 就是具体的命令，记住 cmd 前面的短线“-”千万不要忽略。

部分命令（如 mkdir 等）需要文件\目录名作为参数，参数一般都是 URI 格式，args 参数的基本格式是：

```
scheme://authority/path
```

scheme 指具体的文件系统，如果是本地文件，那么 scheme 就是 file；如果是 HDFS 上的文件，那么 scheme 就是 hdfs。authority 就是机器的地址和对应的端口。当然，正如 Linux 文件有绝对路径和相对路径一样，这里的 URI 参数也可以做一定程度省略。当对应的设置为 hdfs://namenode:namenodeport 时，如果路径参数为 /parent/child，那么它对应的实际文件为

```
hdfs://namenode:namenodeport/parent/child
```

但是有一点要注意，HDFS 没有所谓当前工作目录的概念。前面说过，HDFS 所有文件元数据都是存在 NameNode 节点上的，具体文件的存放由 NameNode 掌控，某一个文件可能被分拆放到不同的机器上，也可能为了提高效率将路径不同的文件也放到同一台机器上。所以，为 HDFS 提供 cd、pwd 操作，都是不现实的。

### 3.4.3 HDFS 文件操作命令

接下来，我们来了解一下 HDFS 的命令。再提醒一下，文件操作命令的基本格式是：

```
bin/hadoop dfs-cmd <args>
```

#### 1. cat

格式：hadoop dfs-cat URI [URI ...]

作用：将参数所指示的文件的内容输出到 stdout。

示例：

- `hadoop dfs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2`
- `hadoop dfs -cat file:///file3 /user/hadoop/file4`

返回值：成功结束返回 0，出现错误返回 -1。

## 2. chgrp

格式：`hadoop dfs -chgrp [-R] GROUP URI [URI ...]`

作用：改变文件所属的用户组。如果使用 -R 选项，则这一操作对整个目录结构递归执行。使用这一命令的用户必须是文件的所属用户，或者是超级用户。

## 3. chmod

格式：`hadoop dfs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI[URI ...]`

作用：改变文件的权限。如果使用 -R 选项，则这一操作对整个目录结构递归执行。使用这一命令的用户必须是文件的所属用户，或者是超级用户。

## 4. chown

格式：`hadoop dfs -chown [-R] [OWNER][,[GROUP]] URI [URI... ]`

作用：改变文件的所属用户。如果使用 -R 选项，则这一操作对整个目录结构递归执行。使用这一命令的用户必须是文件在命令变更之前的所属用户，或者是超级用户。

## 5. copyFromLocal

格式：`hadoop dfs -copyFromLocal <localsrc> URI`

作用：与 put 命令类似，但是要限定源文件路径为本地文件系统。

## 6. copyToLocal

格式：`hadoop dfs -copyToLocal [-ignorecrc] [-crc] URI`

`<localdst>`

作用：与 get 命令类似，但是要限定目标文件路径为本地文件系统。

## 7. count

格式：`hadoop dfs -count [-q] <paths>`

作用：统计匹配对应路径下的目录数，文件数，字节数（文件大小）。

选项意义：

使用 -count 选项时，输出的列为：

DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE, FILE\_NAME



从左到右分别对应目录下已存在的目录数，文件数，文件大小，文件名  
使用 `-count-q` 选项时，输出的列为：

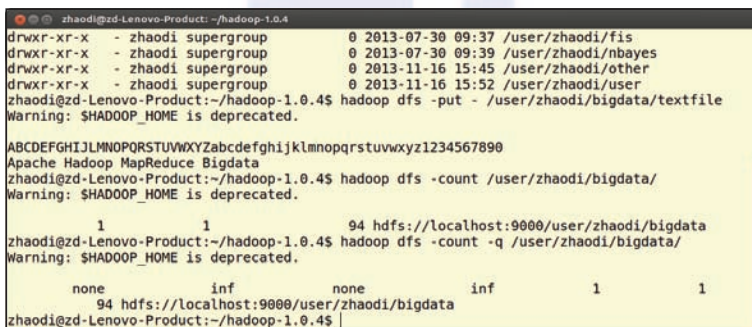
```
QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA,
DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME
```

从左到右的意义是：目录下最大允许文件 + 目录数（不存在上限，则为 `none`），目录下可增加目录 + 文件数（不存在上限，则为 `inf`），目录下最大允许空间（不存在上限，则为 `none`），目录下可用最大空间（不存在上限，则为 `inf`）；后面的几个和 `-count` 选项一致，分别对应目录下已存在的目录数，文件数，文件大小，文件名。

示例：

- `hadoop dfs -count hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2`
- `hadoop dfs -count -q hdfs://nn1.example.com/file1`

返回值：成功结束返回 0，出现错误返回 -1。图 3-8 所示是一个 `count` 选项使用后的结果示例。



```
zhaodi@zd-Lenovo-Product: ~/hadoop-1.0.4
drwxr-xr-x - zhaodi supergroup 0 2013-07-30 09:37 /user/zhaodi/fis
drwxr-xr-x - zhaodi supergroup 0 2013-07-30 09:39 /user/zhaodi/nbayes
drwxr-xr-x - zhaodi supergroup 0 2013-11-16 15:45 /user/zhaodi/other
drwxr-xr-x - zhaodi supergroup 0 2013-11-16 15:52 /user/zhaodi/user
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -put - /user/zhaodi/bigdata/textfile
Warning: $HADOOP_HOME is deprecated.
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890
Apache Hadoop MapReduce Bigdata
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -count /user/zhaodi/bigdata/
Warning: $HADOOP_HOME is deprecated.
      1      1      94 hdfs://localhost:9000/user/zhaodi/bigdata
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -count -q /user/zhaodi/bigdata/
Warning: $HADOOP_HOME is deprecated.
      none      inf      none      inf      1      1
      94 hdfs://localhost:9000/user/zhaodi/bigdata
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$
```

图 3-8 `count` 命令示例

## 8. cp

格式：`hadoop dfs -cp URI [URI ...] <dest>`

作用：将文件拷贝到目标路径中。如果 `<dest>` 为目录的话，可以将多个文件拷贝到该目录下。

示例：

- `hadoop dfs -cp /user/hadoop/file1 /user/hadoop/file2`
- `hadoop dfs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir`

返回值：成功结束返回 0，出现错误返回 -1。

## 9. du

格式：`hadoop dfs -du [-s] [-h] URI [URI ...]`

作用：如果参数为目录，显示该目录下所有目录 + 文件的大小；如果参数为单个文件，则显示文件大小。

选项意义：

-s 指输出所有文件大小的累加和，而不是每个文件的大小。

-h 会将文件大小的数值用方便阅读的形式表示，比如用 64.0M 代替 67108864。

示例：

- `hadoop dfs -du /user/hadoop/dir1 /user/hadoop/file1\`  
`hdfs://nn.example.com/user/hadoop/dir1`

返回值：成功结束返回 0，出现错误返回 -1。

## 10. dus

格式：`hadoop dfs -dus <args>`

作用：显示文件的大小。这个命令等价于 `hadoop dfs -du-s`。

## 11. expunge

格式：`hadoop dfs -expunge`

作用：清空回收站。如需更多有关回收站特性的信息，请参考其他资料和文献。

## 12. get

格式：`hadoop dfs -get [-ignorecrc] [-crc] <src><localdst>`

作用：将文件拷贝到本地文件系统。CRC 校验失败的文件可通过 -ignorecrc 选项拷贝。文件和 CRC 校验和可以通过 -crc 选项拷贝。

示例：

- `hadoop dfs -get /user/hadoop/file localfile`
- `hadoop dfs -get hdfs://nn.example.com/user/hadoop/file localfile`

返回值：成功结束返回 0，出现错误返回 -1。

## 13. getmerge

格式：`hadoop dfs -getmerge <src><localdst> [addnl]`

作用：命令参数为一个源文件目录和一个目的文件。将源文件目录下的所有文件排序后合并到目的文件中。添加 addnl 可以在每个文件后面插入新行。

## 14. ls

格式：`hadoop dfs -ls <args>`

作用：对于一个文件，该命令返回的文件状态以如下格式列出：

```
permissions number_of_replicas userid groupid filesize
modification_date modification_time filename
```

从左到右的意义分别是：文件权限，副本个数，用户 ID，组 ID，文件大小，最近一次修改日期，最近一次修改时间，文件名。

对于一个目录，该命令返回这一目录下的第一层子目录和文件，与 Unix 中 ls 命令的结果类似；结果以如下状态列出：

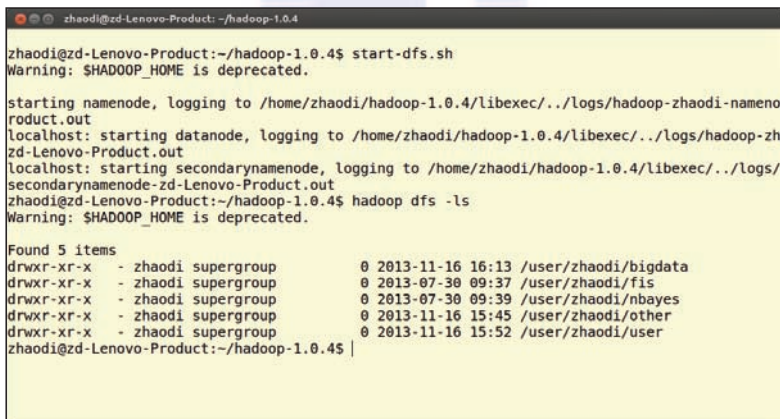
```
permissions userid groupid modification_date modification_time dirname
```

从左到右的意义分别是：文件权限，用户 ID，组 ID，最近一次修改日期，最近一次修改时间，文件名。

示例：

- `hadoop dfs -ls /user/hadoop/file1`

返回值：成功结束返回 0，出现错误返回 -1。图 3-9 所示是一个 ls 命令显示结果示例。



```
zhaodi@zd-Lenovo-Product: ~/hadoop-1.0.4
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ start-dfs.sh
Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /home/zhaodi/hadoop-1.0.4/libexec/../logs/hadoop-zhaodi-namenode.out
localhost: starting datanode, logging to /home/zhaodi/hadoop-1.0.4/libexec/../logs/hadoop-zhaodi-datanode.out
localhost: starting secondarynamenode, logging to /home/zhaodi/hadoop-1.0.4/libexec/../logs/hadoop-zhaodi-secondarynamenode.out
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -ls
Warning: $HADOOP_HOME is deprecated.

Found 5 items
drwxr-xr-x - zhaodi supergroup      0 2013-11-16 16:13 /user/zhaodi/bigdata
drwxr-xr-x - zhaodi supergroup      0 2013-07-30 09:37 /user/zhaodi/fis
drwxr-xr-x - zhaodi supergroup      0 2013-07-30 09:39 /user/zhaodi/nbayes
drwxr-xr-x - zhaodi supergroup      0 2013-11-16 15:45 /user/zhaodi/other
drwxr-xr-x - zhaodi supergroup      0 2013-11-16 15:52 /user/zhaodi/user
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ |
```

图 3-9 HDFS 启动与 ls 命令示例

## 15. lsr

格式：`hadoop dfs -lsr <args>`

作用：在整个目录下递归执行 ls，与 Unix 中的 ls-R 类似。

## 16. mkdir

格式：`hadoop dfs -mkdir <paths>`

作用：以 <paths> 中的 URI 作为参数，创建目录。该命令的行为与 Unix 中 mkdir-p 的

行为十分相似。这一路径上的父目录如果不存在，则创建该父目录。

示例：

- `hadoop dfs -mkdir /user/hadoop/dir1 /user/hadoop/dir2`
- `hadoop dfs -mkdir hdfs://nn1.example.com/user/hadoop/dir  
hdfs://nn2.example.com/user/hadoop/dir`

返回值：成功结束返回 0，出现错误返回 -1。

## 17. moveFromLocal

格式：`hadoop dfs -moveFromLocal <localsrc><dst>`

作用：和 `put` 命令类似，但是源文件 `localsrc` 拷贝之后自身被删除。

## 18. moveToLocal

格式：`hadoop dfs -moveToLocal [-crc] <src><dst>`

作用：输出 “Not implemented yet” 信息，也就是说当前版本中未实现此命令。

## 19. mv

格式：`hadoop dfs -mv URI [URI ...] <dest>`

作用：将文件从源路径移动到目标路径（移动之后源文件删除）。目标路径为目录的情况下，源路径可以有多个。跨文件系统的移动（本地到 HDFS 或者反过来）是不允许的。

示例：

- `hadoop dfs -mv /user/hadoop/file1 /user/hadoop./file2`
- `hadoop dfs -mv hdfs://nn.example.com/file1  
hdfs://nn.example.com/file2 hdfs://nn.example.com/file3  
hdfs://nn.example.com/dir1`

返回值：

成功结束返回 0，出现错误返回 -1。

## 20. put

格式：`hadoop dfs -put <localsrc> ... <dst>`

作用：将单个的源文件 `src` 或者多个源文件 `srcs` 从本地文件系统拷贝到目标文件系统中（`<dst>` 对应的路径）。也可以从标准输入中读取输入，写入目标文件系统中。

示例：

- `hadoop dfs -put localfile /user/hadoop/hadoopfile`
- `hadoop dfs -put localfile1 localfile2 /user/hadoop/hadoopdir`
- `hadoop dfs -put localfile hdfs://nn.example.com/hadoop/hadoopfile`
- `hadoop dfs -put - hdfs://nn.example.com/hadoop/hadoopfile`

最后一个实例中，<localsrc> 为“-”，这就是可以读取标准输入，当用户输入 EOF (Ctrl+C) 时，输入结束，此命令会将这些输入的数据写入 HDFS 的对应目录中。返回值：成功结束返回 0，出现错误返回 -1。图 3-10 所示是一个 put 命令操作结果示例。

```

zhaodi@zd-Lenovo-Product: ~/hadoop-1.0.4
Found 5 items
drwxr-xr-x - zhaodi supergroup 0 2013-11-16 16:13 /user/zhaodi/bigdata
drwxr-xr-x - zhaodi supergroup 0 2013-07-30 09:37 /user/zhaodi/fis
drwxr-xr-x - zhaodi supergroup 0 2013-07-30 09:39 /user/zhaodi/nbays
drwxr-xr-x - zhaodi supergroup 0 2013-11-16 15:45 /user/zhaodi/other
drwxr-xr-x - zhaodi supergroup 0 2013-11-16 15:52 /user/zhaodi/user
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -put - /user/zhaodi/bigdata/textfile
Warning: SHADOOP_HOME is deprecated.
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890
Apache Hadoop MapReduce Bigdata
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$

```

图 3-10 put 命令示例

## 21. rm

格式: `hadoop dfs -rm [-skipTrash] URI [URI ...]`

作用：删除参数指定的文件，参数可以有多个。此命令只删除文件和非空目录。如果指定了 -skipTrash 选项，那么在回收站可用的情况下，该选项将跳过回收站而直接删除文件；否则，在回收站可用时，在 HDFS Shell 中执行此命令，会将文件暂时放到回收站中。这一选项在删除超过容量限制的目录（over-quota directory）中的文件时很有用。需要递归删除时可参考 rmr 命令。

示例：

- `hadoop dfs -rm hdfs://nn.example.com/file /user/hadoop/emptydir`

返回值：成功结束返回 0，出现错误返回 -1。图 3-11 所示是一个 rm 命令操作结果示例。

```

zhaodi@zd-Lenovo-Product: ~/hadoop-1.0.4
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -tail /user/zhaodi/bigdata/textfile
Warning: SHADOOP_HOME is deprecated.
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890
Apache Hadoop MapReduce Bigdata
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ hadoop dfs -rm /user/zhaodi/bigdata/textfile
Warning: SHADOOP_HOME is deprecated.
Deleted hdfs://localhost:9000/user/zhaodi/bigdata/textfile
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$ stop-dfs.sh
Warning: SHADOOP_HOME is deprecated.
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
zhaodi@zd-Lenovo-Product:~/hadoop-1.0.4$

```

图 3-11 tail, rm 命令，以及 HDFS 关闭命令示例

## 22. rmr

格式: `hadoop dfs -rmr [-skipTrash] URI [URI ...]`

作用：删除操作的递归版本，即递归删除所有子目录下的文件。如果指定了 `-skipTrash` 选项，那么在回收站可用的情况下，该选项将跳过回收站而直接删除文件；否则，在回收站可用时，在 HDFS Shell 中执行此命令，会将文件暂时放到回收站中。这一选项在删除超过容量限制的目录（over-quota directory）中的文件时很有用。

示例：

- `hadoop dfs -rmr /user/hadoop/dir`
- `hadoop dfs -rmr hdfs://nn.example.com/user/hadoop/dir`

返回值：成功结束返回 0，出现错误返回 -1。

### 23. setrep

格式：`hadoop dfs -setrep [-R] <path>`

作用：改变一个文件在 HDFS 中的副本个数。使用 `-R` 选项可以对一个目录下的所有目录 + 文件递归执行改变副本个数的操作。

示例：

- `hadoop dfs -setrep -w 3 -R /user/hadoop/dir1`

返回值：成功结束返回 0，出现错误返回 -1。

### 24. stat

格式：`hadoop dfs -stat [format] URI [URI ...]`

作用：返回对应路径的状态信息。可以通过与 C 语言中的 `printf` 类似的格式化字符串定制输出格式，这里支持的格式字符有：

`%b`：文件大小

`%o`：Block 大小

`%n`：文件名

`%r`：副本个数

`%y` 或 `%Y`：最后一次修改日期和时间

默认情况输出最后一次修改日期和时间。

示例：

- `hadoop dfs -stat path`
- `hadoop dfs -stat "%n %b %o %y" path`

返回值：成功结束返回 0，出现错误返回 -1。



## 25. tail

格式: `hadoop dfs -tail [-f] URI`

作用: 在标准输出中显示文件末尾的 1KB 数据。-f 的用法与 Unix 类似, 也就是说当文件尾部添加了新的数据或者做出了修改时, 在标准输出中也会刷新显示。

示例:

- `hadoop dfs -tail pathname`

返回值: 成功结束返回 0, 出现错误返回 -1。

## 26. test

格式: `hadoop dfs -test -[ezd] URI`

作用: 判断文件信息。

选项含义:

- e 检查文件是否存在, 如果存在返回 0。
  - z 检查文件大小是否为 0, 是的话返回 0。
  - d 检查这一路径是否为目录, 是的话返回 0。
- 如果返回 0 则不输出, 否则会输出相应的信息。

示例:

- `hadoop dfs -test -e filename`

## 27. text

格式: `hadoop dfs -text <src>`

作用: 将文本文件或者某些格式的非文本文件通过文本格式输出。允许的格式有 zip 和 TextRecordInputStream。

## 28. touchz

格式: `hadoop dfs -touchz URI [URI ...]`

作用: 创建一个大小为 0 的文件。

示例:

- `hadoop dfs -touchz pathname`

返回值: 成功结束返回 0, 出现错误返回 -1。

### 3.4.4 高级操作命令和工具

本节讲解 HDFS 的一些高级操作功能, 以及通过 web 方式查看 HDFS 信息的方法。

1. archive

在本地文件系统中，如果文件很少用，但又占用很大空间，可以将其压缩起来，以减少空间使用。在 HDFS 中同样也会面临这种问题，一些小文件可能只有几 KB 到几十 KB，但是在 DataNode 中也要单独为其分配一个几十 MB 的数据块，同时还要在 NameNode 中保存数据块的信息。如果小文件很多的话，对于 NameNode 和 DataNode 都会带来很大负担。所以 HDFS 中提供了 archive 功能，将文件压缩起来，减少空间使用。

HDFS 的压缩文件的后缀名是 .har，一个 har 文件中包括文件的元数据（保存在 \_index 和 \_masterindex）以及具体数据（保存在 part-XX）。但是，HDFS 的压缩文件和本地文件系统的压缩文件不同的是：har 文件不能进行二次压缩；另外，har 文件中，原来文件的数据并没有变化，har 文件真正的作用是减少 NameNode 和 DataNode 过多的空间浪费。简单算一笔账，保存 1000 个 10K 的文件，不用 archive 的话，要用 64M × 1000，也就是将近 63G 的空间来保存；用 archive 的话，因为总数据量有 10M（还需要加上这些文件的 \_index 和 \_masterindex，不过很小就是了），只需要一个数据块，也就是 64M 的空间就够了。这样的话，节约的空间相当多；如果有十万百万的文件，那节省的空间会更可观。

将文件压缩成 .har 文件的格式如下：

```
hadoop archive -archiveName name -p <parent><src>*<dest>
```

选项含义如表 3-1：

表 3-1 archive 命令的选项及含义

选 项	含 义
-archiveName	指定压缩文件名
-p	待压缩文件所在父目录
<src>*	待压缩文件路径（相对 <parent>），如果这一部分没有，则是将 <parent> 的所有文件都压缩
<dest>	压缩文件存放路径

示例：

```
hadoop archive -archiveName zoo.har -p /foo/bar /outputdir
```

注意，.har 文件一旦创建之后就不能更改，也不能再次被压缩。如果想给 .har 加文件，只能找到原来的文件，重新创建一个。

访问 har 文件的内容可以通过指定 URL har:///user/data/arch.har 来完成，所以可以通过上节提到的文件操作命令操作 har，比如，显示 har 文件内容可以用：

```
hadoop dfs -ls har:///user/data/arch.har
```

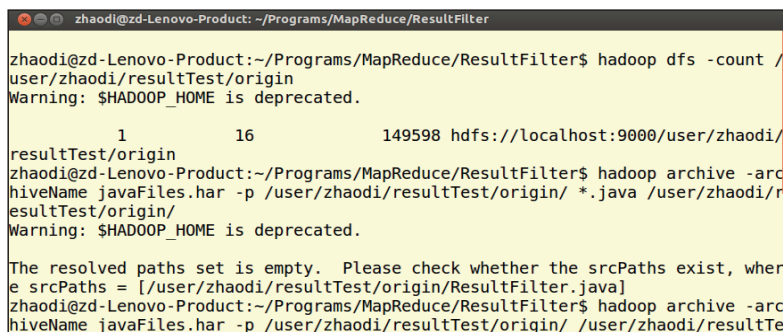
查看全部文件可以用：

```
hadoop dfs -lsr har:///user/data/arch.har
```

也可以作为 MapReduce 作业的输入：

```
hadoopjar MyJob.jar MyJobMain har:///user/data/arch.jar \
/user/data/output/
```

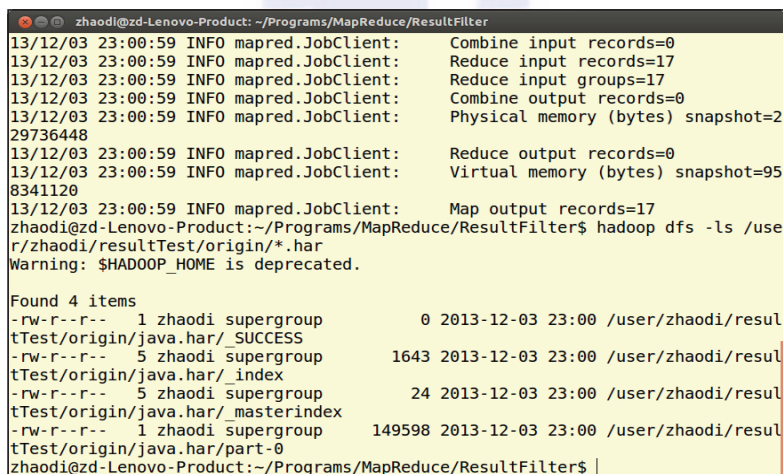
看一下图 2-12，将一个目录 /user/zhaodi/resultTest/origin 压缩成 har 文件。



```
zhaodi@zd-Lenovo-Product: ~/Programs/MapReduce/ResultFilter
zhaodi@zd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop dfs -count /
user/zhaodi/resultTest/origin
Warning: $HADOOP_HOME is deprecated.
      1      16      149598 hdfs://localhost:9000/user/zhaodi/
resultTest/origin
zhaodi@zd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop archive -arc
hiveName javaFiles.har -p /user/zhaodi/resultTest/origin/ *.java /user/zhaodi/r
esultTest/origin/
Warning: $HADOOP_HOME is deprecated.

The resolved paths set is empty. Please check whether the srcPaths exist, wher
e srcPaths = [/user/zhaodi/resultTest/origin/ResultFilter.java]
zhaodi@zd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop archive -arc
hiveName javaFiles.har -p /user/zhaodi/resultTest/origin/ /user/zhaodi/resultTe
```

图 3-12 archive 示例 1



```
zhaodi@zd-Lenovo-Product: ~/Programs/MapReduce/ResultFilter
13/12/03 23:00:59 INFO mapred.JobClient: Combine input records=0
13/12/03 23:00:59 INFO mapred.JobClient: Reduce input records=17
13/12/03 23:00:59 INFO mapred.JobClient: Reduce input groups=17
13/12/03 23:00:59 INFO mapred.JobClient: Combine output records=0
13/12/03 23:00:59 INFO mapred.JobClient: Physical memory (bytes) snapshot=2
29736448
13/12/03 23:00:59 INFO mapred.JobClient: Reduce output records=0
13/12/03 23:00:59 INFO mapred.JobClient: Virtual memory (bytes) snapshot=95
8341120
13/12/03 23:00:59 INFO mapred.JobClient: Map output records=17
zhaodi@zd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop dfs -ls /use
r/zhaodi/resultTest/origin/*.har
Warning: $HADOOP_HOME is deprecated.

Found 4 items
-rw-r--r-- 1 zhaodi supergroup 0 2013-12-03 23:00 /user/zhaodi/resul
tTest/origin/java.har/_SUCCESS
-rw-r--r-- 5 zhaodi supergroup 1643 2013-12-03 23:00 /user/zhaodi/resul
tTest/origin/java.har/_index
-rw-r--r-- 5 zhaodi supergroup 24 2013-12-03 23:00 /user/zhaodi/resul
tTest/origin/java.har/_masterindex
-rw-r--r-- 1 zhaodi supergroup 149598 2013-12-03 23:00 /user/zhaodi/resul
tTest/origin/java.har/part-0
zhaodi@zd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ |
```

图 3-13 archive 示例 2

压缩完毕后，发现 origin 的大小和 har 文件中的 part-XX 的大小一样，之所以一样的原因是 har 压缩文件在文件的数据块占用上做了优化，但是文件本身并未发生变化，只是单纯连接到一起而已。

## 2. balancer

如 3.2.1 节所述，HDFS 并不会将数据块的副本在集群中均匀分布，一个重要原因就是

在已存在的集群中添加和删除 DataNode 被视作正常的情形。保存数据块时，NameNode 会从多个角度考虑 DataNode 的选择，比如：

- 将副本保存到与第一个副本所在 DataNode 所属机架不同的机架上（这里的机架可以认为是若干 DataNode 组成的“局域网”，机架内部的 DataNode 之间的数据传输的代价远小于机架内部 DataNode 和机架外部的数据传输）。
- 在与正写入文件数据的 DataNode 相同的机架上，选择另外的 DataNode 放一个副本。
- 在满足以上条件之后，尽量将副本均匀分布。

在默认的副本因子为 3 的集群中，一般情况下，数据块的存放策略如下：首先，选择一个 DataNode 保存第一个副本；接下来，选择与第一副本所在 DataNode 不同的机架保存第二个副本；最后，和第二个副本相同的机架中，选择另外一个 DataNode 保存第三个副本。

如果管理员发现某些 DataNode 保存数据过多，而某些 DataNode 保存数据相对少，那么可以使用 hadoop 提供的工具 balancer，手动启动内部的均衡过程。

命令如下：

```
hadoop balancer [-threshold <threshold>]
```

-threshold 参数是一个 0 ~ 100 之间的实数，单位为百分比，默认值为 10。这个参数表示：各个 DataNode 的利用率（已用空间 / 可用空间）与整个集群的利用率的差值的绝对值的上限。也就是说，如果每个 DataNode 的利用率和平均利用率相差不大（小于阈值）的话，可以认为这个集群已经“平衡”了。管理员可以通过 Ctrl+C 手动打断 balancer。

另外还有一种运行方式，在终端中输入如下命令：

```
start-balancer.sh[-t <therehold>]
```

可以启动后台守护进程，也能达到同样效果。-t 选项指定阈值。在“平衡”之后，进程退出，手动关闭进程的方式为：

```
stop-balancer.sh
```

### 3. distcp

distcp（distribution copy）用来在两个 HDFS 之间拷贝数据。在 HDFS 之间拷贝数据要考虑很多因素，比如，两个 HDFS 的版本不同怎么办？两个 HDFS 的数据块大小、副本因子各不相同，又该怎么办？不同的数据块分布在不同节点上，如何让传输效率尽量高，等等。

正因如此，HDFS 中专门用 distcp 命令完成跨 HDFS 数据拷贝。从 /src/tools 子目录下的源代码中可以看出，distcp 是一个没有 reducer 的 MapReduce 过程。

distcp 命令格式如下：

```
hadoop distcp [options] <srcurl>*<desturl>
```

<srcurl><desturl> 就是源文件和目标文件的路径，这和 fs 中的 cp 类似。  
Options 选项及含义如表 3-2 所示：

表 3-2 distcp 命令的选项及含义

选 项	含 义
-p[rbugp]	保留之前的属性信息，rbugp 分别表示：r(eplication number 副本数)，b(lockSize 数据块数)，u(ser 所属用户)，g(roup 所属组)，p(ermission 权限)。只用 -p 等价于 -prbugp
-i	忽略传输失败
-log <logdir>	将日志写入到 <logdir> 中
-m <num_maps>	并行传输过程数目最大值
-overwrite	覆盖目标路径已有文件
-update	只有在源路径的文件和目标路径的文件大小不同时才覆盖
-f <urilist_uri>	使用 <urilist_uri> 作为输入
-filelimit <n>	文件数不超过 n
-sizelimit <n>	拷贝数据大小不超过 n?
-delete	删除目标路径中存在但是源路径不存在的文件
-mapredSslConf <f>	为 map 任务配置 SSL 信息的文件

注：不同版本的 HDFS 可以通过 http 协议拷贝，那么命令

```
Hadoop distcp hdfs://dn1:port1/data/file1\  
hdfs://dn2:port2/data/file2
```

可以写成：

```
hadoop distcp hftp://dn1:port1/data/file1\  
hftp://dn2:port2/data/file2
```

后续版本中，HDFS 中增加了 distcp 的增强版本 distcp2。比起 distcp，dsitcp2 多了许多高级功能，如：-bandwidth，允许设置传输带宽；-atomic，允许借助临时目录进行拷贝；-strategy，允许设置拷贝策略；-async，允许异步执行（后台运行传输过程，而命令行可以继续执行命令）。

4. dfsadmin

管理员可以通过 dfsadmin 管理 HDFS。支持的命令选项及含义如表 3-3：

表 3-3 dfsadmin 命令的选项及含义

选 项	含 义
-report	显示文件系统的基本数据
-safemode	维护 HDFS 的安全模式。该命令的参数有：enter，进入安全模式；leave，离开安全模式；get，获知是否开启安全模式；wait 等待离开安全模式。如果手动进入了安全模式，只能通过手动退出
-refreshNodes	更新 DataNode 信息
-finalizeUpgrade	完成升级，将 NameNode 和 DataNode 上的所有的上一个版本信息删除
-help	显示帮助信息

另外，HDFS 还提供了通过 web 查看 HDFS 信息的方式。HDFS 启动之后，会建立 web 服务，在默认情况下，访问 `http://namenode-name : 50070` 即可查看 HDFS 的 Name Node 信息，如图 3-14 所示：

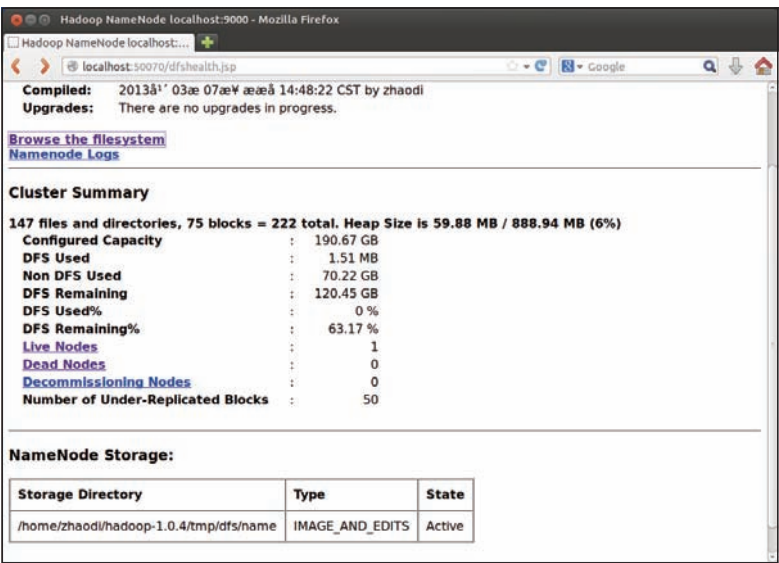


图 3-14 NameNode 的 web 界面

通过 web 界面可以查看 HDFS 的信息，包括总容量、可用容量、DataNodes 的信息、HDFS 运行目录等。

点击“Browse the filesystem”可以查看 HDFS 的目录结构，如图 3-15 所示。  
点击“Live Nodes”可以查看当前有效的 DataNode 的信息，如图 3-16 所示。

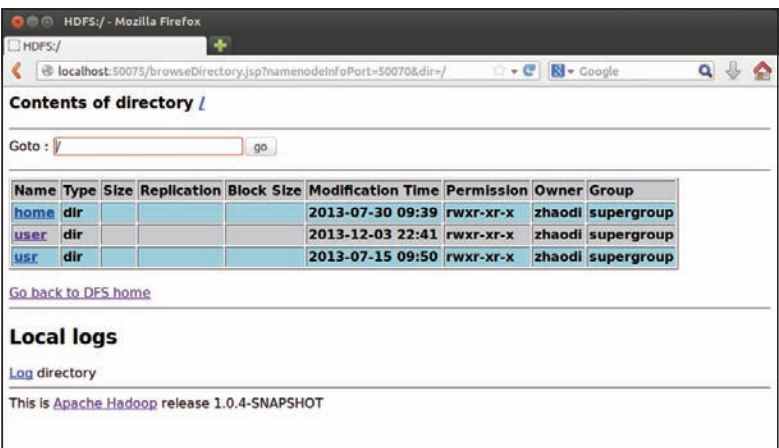


图 3-15 HDFS 的文件目录结构



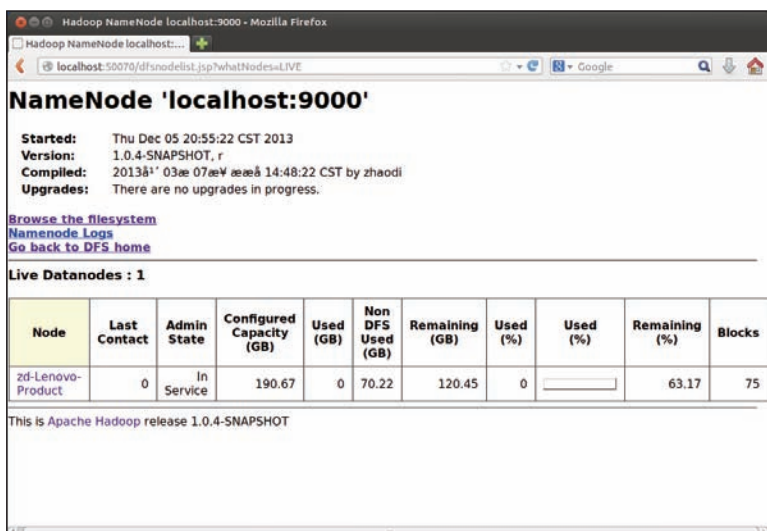


图 3-16 DataNode 的相关信息

## 3.5 HDFS 基本编程接口与示例

除了上一节提到的命令之外，Hadoop 提供了可用于读写、操作文件的 API，这样可以让程序员通过编程实现自己的 HDFS 文件操作。

Hadoop 提供的大部分文件操作 API 都位于 `org.apache.hadoop.fs` 这个包中。基本的文件操作包括打开、读取、写入、关闭等。为了保证能跨文件系统交换数据，Hadoop 的 API 也可以对部分非 HDFS 的文件系统提供支持；也就是说，用这些 API 来操作本地文件系统的文件也是可行的。

### 3.5.1 HDFS 编程基础知识

在 Hadoop 中，基本上所有的文件 API 都来自 `FileSystem` 类。`FileSystem` 是一个用来与文件系统交互的抽象类，可以通过实现 `FileSystem` 的子类来处理具体的文件系统，比如 HDFS 或者其他文件系统。通过 `factory` 方法 `FileSystem.get(Configuration conf)`，可以获得所需的文件系统实例（`factory` 方法是软件开发的一种设计模式，指：基类定义接口，但是由子类实例化之；在这里 `FileSystem` 定义 `get` 接口，但是由 `FileSystem` 的子类（比如 `FilterFileSystem`）实现）。`Configuration` 类比较特殊，这个类通过键值对的方式保存了一些配置参数。这些配置默认情况下来自对应文件系统的资源配置。我们可以通过如下方式获得具体的 `FileSystem` 实例：

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

如果要获得本地文件系统对应的 `FileSystem` 实例，则可以通过 `factory` 方法 `FileSystem.getLocal(Configuration conf)` 实现：

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

Hadoop 中，使用 Path 类的对象来编码目录或者文件的路径，使用后面会提到的 FileStatus 类来存放目录和文件的信息。在 Java 的文件 API 中，文件名都是 String 类型的字符串，在这里则是 Path 类型的对象。

### 3.5.2 HDFS 基本文件操作 API

接下来看一下具体的文件操作。我们按照“创建、打开、获取文件信息、获取目录信息、读取、写入、关闭、删除”的顺序讲解 Hadoop 提供的文件操作的 API。

以下接口的实际内容可以在 Hadoop API 和 Hadoop 源代码中进一步了解。

#### 1. 创建文件

FileSystem.create 方法有很多种定义形式，参数最多的一个是：

```
public abstract FSDataOutputStream create(Path f,
    FsPermission permission,
    boolean overwrite,
    int bufferSize,
    short replication,
    long blockSize,
    Progressable progress)
    Throws IOException
```

那些参数较少的 create 只不过是将其一部分参数用默认值代替，最终还是要调用这个函数。其中各项的含义如下：

f：文件名

overwrite：如果已存在同名文件，overwrite=true 覆盖之，否则抛出错误；默认为 true。

bufferSize：文件缓存大小。默认值：Configuration 中 io.file.buffer.size 的值，如果 Configuration 中未显式设置该值，则是 4096。

replication：创建的副本个数，默认值为 1。

blockSize：文件的 block 大小，默认值：Configuration 中 fs.local.block.size 的值，如果 Configuration 中未显式设置该值，则是 32M。

permission 和 progress 的值与具体文件系统实现有关。

但是大部分情况下，只需要用到最简单的几个版本：

```
public FSDataOutputStream create(Path f);
public FSDataOutputStream create(Path f, boolean overwrite);
public FSDataOutputStream create(Path f, boolean overwrite, int bufferSize);
```

#### 2. 打开文件

FileSystem.open 方法有 2 个，参数最多的一个定义如下：

```
public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException
```

其中各项的含义如下：

f：文件名

bufferSize：文件缓存大小。默认值：Configuration 中 io.file.buffer.size 的值，如果 Configuration 中未显式设置该值，则是 4096。

### 3. 获取文件信息

FileSystem.getFileStatus 方法格式如下：

```
public abstract FileStatus getFileStatus(Path f) throws IOException;
```

这一函数会返回一个 FileStatus 对象。通过阅读源代码可知，FileStatus 保存了文件的很多信息，包括：

path：文件路径

length：文件长度

isDir：是否为目录

block\_replication：数据块副本因子

blockSize：文件长度（数据块数）

modification\_time：最近一次修改时间

access\_time：最近一次访问时间

owner：文件所属用户

group：文件所属组

如果了解文件的这些信息，可以在获得文件的 FileStatus 实例之后，调用相应的 getXXX 方法（比如，FileStatus.getModificationTime（）获得最近修改时间）。

### 4. 获取目录信息

获取目录信息，不仅是目录本身，还有目录之下的文件和子目录信息，如下所述。FileStatus.listStatus 方法格式如下：

```
public FileStatus[] listStatus(Path f) throws IOException;
```

如果 f 是目录，那么将目录之下的每个目录或文件信息保存在 FileStatus 数组中返回。如果 f 是文件，和 getFileStatus 功能一致。

另外，listStatus 还有参数为 Path[] 的版本的接口定义以及参数带路径过滤器 PathFilter 的接口定义，参数为 Path[] 的 listStatus 就是对这个数组中的每个 path 都调用上面的参数为 Path 的 listStatus。参数中的 PathFilter 则是一个接口，实现接口的 accept 方法可以自定义文件过滤规则。

另外，HDFS 还可以通过正则表达式匹配文件名来提取需要的文件，这个方法是：

```
public FileStatus[] globStatus(Path pathPattern) throws IOException;
```

参数 `pathPattern` 中，可以像正则表达式一样，使用通配符来表示匹配规则：

?：表示任意的单个字符。

\*：表示任意长度的任意字符，可以用来表示前缀后缀，比如 `*.java` 表示所有 java 文件。

[abc]：表示匹配 a, b, c 中的单个字符。

[a-b]：表示匹配 a-b 范围之间的单个字符。

[^a]：表示匹配除 a 之外的单个字符。

\c：表示取消特殊字符的转义，比如 `\*` 的结果是 `*` 而不是随意匹配。

{ab, cd}：表示匹配 ab 或者 cd 中的一个串。

{ab, c{de, fh}}：表示匹配 ab 或者 cde 或者 cfh 中的一个串

## 5. 读取

3.3.2 节提到，调用 `open` 打开文件之后，使用了一个 `FSDDataInputStream` 对象来负责数据的读取。通过 `FSDDataInputStream` 进行文件读取时，提供的 API 就是 `FSDDataInputStream.read` 方法：

```
public int read(long position, byte[] buffer, int offset, int length) throws IOException
```

函数的意义是：从文件的指定位置 `position` 开始，读取最多 `length` 字节的数据，保存到 `buffer` 中从 `offset` 个元素开始的空间中；返回值为实际读取的字节数。此函数不改变文件当前 `offset` 值。不过，使用更多的还有一种简化版本：

```
public final int read(byte[] b) throws IOException
```

从文件当前位置读取最多长度为 `b.len` 的数据保存到 `b` 中，返回值为实际读取的字节数。

## 6. 写入

从接口定义可以看出，调用 `create` 创建文件以后，使用了一个 `FSDDataOutputStream` 对象来负责数据的写入。通过 `FSDDataOutputStream` 进行文件写入时，最常用的 API 就是 `write` 方法：

```
public void write(byte[] b, int off, int len) throws IOException
```

函数的意义是：将 `b` 中从 `off` 开始的最多 `len` 个字节的数据写入文件当前位置。返回值为实际写入的字节数。

## 7. 关闭

关闭为打开的逆过程，`FileSystem.close` 定义如下：

```
public void close() throws IOException
```

不需要其他操作而关闭文件。释放所有持有的锁。

## 8. 删除

删除过程 `FileSystem.delete` 定义如下：

```
public abstract boolean delete(Path f,boolean recursive) throws IOException
```

其中各项含义如下：

`f`：待删除文件名。

`recursive`：如果 `recursive` 为 `true`，并且 `f` 是目录，那么会递归删除 `f` 下所有文件；如果 `f` 是文件，`recursive` 为 `true` 还是 `false` 无影响。

另外，类似 Java 中 `File` 的接口 `DeleteOnExit`，如果某些文件需要删除，但是当前不能被删；或者说当时删除代价太大，想留到退出时再删除的话，`FileSystem` 中也提供了一个 `deleteOnExit` 接口：

```
Public Boolean deleteOnExit (Path f) throws IOException
```

标记文件 `f`，当文件系统关闭时才真正删除此文件。但是这个文件 `f` 在文件系统关闭前必须存在。

### 3.5.3 HDFS 基本编程实例

本节介绍使用 HDFS 的 API 编程的简单示例。

下面的程序可以实现如下功能：在输入文件目录下的所有文件中，检索某一特定字符串所出现的行，将这些行的内容输出到本地文件系统的输出文件夹中。这一功能在分析 MapReduce 作业的 Reduce 输出时很有用。

这个程序假定只有第一层目录下的文件才有效，而且，假定文件都是文本文件。当然，如果输入文件夹是 Reduce 结果的输出，那么一般情况下，上述条件都能满足。为了防止单个的输出文件过大，这里还加了一个文件最大行数限制，当文件行数达到最大值时，便关闭此文件，创建另外的文件继续保存。保存的结果文件名为 1, 2, 3, 4, …，以此类推。

如上所述，这个程序可以用来分析 MapReduce 的结果，所以称为 `ResultFilter`。

程序：`Result Filter`

输入参数：此程序接收 4 个命令行输入参数，参数含义如下：

<dfs path>：HDFS 上的路径

<local path>：本地路径

<match str>：待查找的字符串

<single file lines>：结果每个文件的行数

## 程序：ResultFilter

```

import java.util.Scanner;
import java.io.IOException;
import java.io.File;

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 resultFilter
{
    public static void main(String[] args) throws IOException {
        Configuration conf = new Configuration();
        // 以下两句中 ,hdfs 和 local 分别对应 HDFS 实例和本地文件系统实例
        FileSystem hdfs = FileSystem.get(conf);
        FileSystem local = FileSystem.getLocal(conf);

        Path inputDir, localFile;

        FileStatus[] inputFiles;
        FSDataOutputStream out = null;
        FSDataInputStream in = null;
        Scanner scan;
        String str;
        byte[] buf;
        int singleFileLines;
        int numLines, numFiles, i;

        if(args.length!=4)
        {
            // 输入参数数量不够，提示参数格式后终止程序执行
            System.err.println("usage resultFilter <dfs path><local path> " +
                " <match str><single file lines>");
            return;
        }
        inputDir = new Path(args[0]);
        singleFileLines = Integer.parseInt(args[3]);

        try {
            inputFiles = hdfs.listStatus(inputDir); // 获得目录信息
            numLines = 0;
            numFiles = 1; // 输出文件从 1 开始编号
            localFile = new Path(args[1]);
            if(local.exists(localFile)) // 若目标路径存在，则删除之
                local.delete(localFile, true);
            for (i = 0; i<inputFiles.length; i++) {
                if(inputFiles[i].isDir() == true) // 忽略子目录
                    continue;

```



```

System.out.println(inputFiles[i].getPath().getName());
in = hdfs.open(inputFiles[i].getPath());
scan = new Scanner(in);
while (scan.hasNext()) {
    str = scan.nextLine();
    if(str.indexOf(args[2])!=-1)
        continue; // 如果该行没有 match 字符串，则忽略之
    numLines++;
    if(numLines == 1) // 如果是 1, 说明需要新建文件了
    {
        localFile = new Path(args[1] + File.separator + numFiles);
        out = local.create(localFile); // 创建文件
        numFiles++;
    }
    buf = (str+"\n").getBytes();
    out.write(buf, 0, buf.length); // 将字符串写入输出流
    if(numLines == singleFileLines) // 如果已满足相应行数，关闭文件
    {
        out.close();
        numLines = 0; // 行数变为 0, 重新统计
    }
} //end of while
scan.close();
in.close();
} //end of for
if(out != null)
    out.close();
} //end of try
catch (IOException e) {
    e.printStackTrace();
}
} //end of main
} //end of resultFilter

```

程序的编译命令:

```
javac *.java
```

运行命令:

```
hadoop jar resultFilter.jar resultFilter <dfs path>\
    <local path><match str><single file lines>
```

参数和含义如下:

<dfs path>:HDFS 上的路径

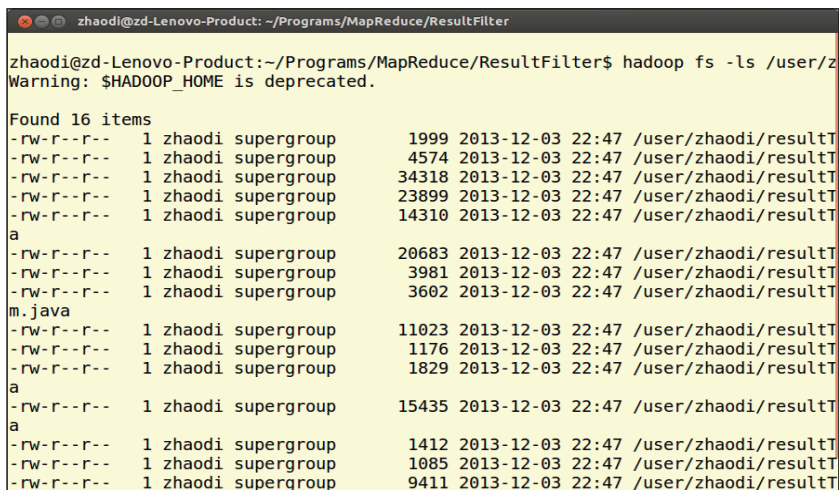
<local path>: 本地路径

<match str>: 待查找的字符串

<single file lines>: 结果的每个文件的行数

上述程序的逻辑很简单，获取该目录下所有文件的信息，对每一个文件，打开文件、循环读取数据、写入目标位置，然后关闭文件，最后关闭输出文件。这里粗体打印的几个函数上面都有介绍，不再赘述。

我们在自己机器上预装的 hadoop-1.0.4 上简单试验了这个程序，在 hadoop 源码中拷贝了几个文件，然后上传到 HDFS 中，文件如下（见图 3-17）：



```

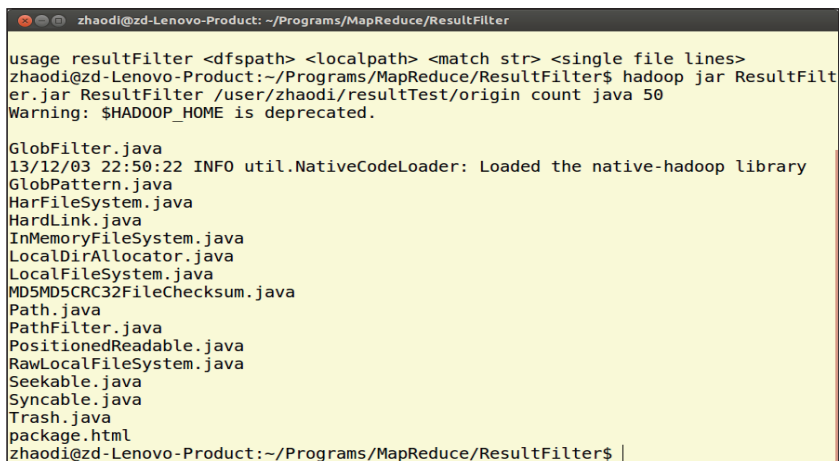
zhaodi@zhd-Lenovo-Product: ~/Programs/MapReduce/ResultFilter
zhaodi@zhd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop fs -ls /user/z
Warning: $HADOOP_HOME is deprecated.

Found 16 items
-rw-r--r-- 1 zhaodi supergroup      1999 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      4574 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup     34318 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup     23899 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup     14310 2013-12-03 22:47 /user/zhaodi/resultT
a
-rw-r--r-- 1 zhaodi supergroup     20683 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      3981 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      3602 2013-12-03 22:47 /user/zhaodi/resultT
m.java
-rw-r--r-- 1 zhaodi supergroup     11023 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      1176 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      1829 2013-12-03 22:47 /user/zhaodi/resultT
a
-rw-r--r-- 1 zhaodi supergroup     15435 2013-12-03 22:47 /user/zhaodi/resultT
a
-rw-r--r-- 1 zhaodi supergroup      1412 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      1085 2013-12-03 22:47 /user/zhaodi/resultT
-rw-r--r-- 1 zhaodi supergroup      9411 2013-12-03 22:47 /user/zhaodi/resultT

```

图 3-17 HDFS 中的内容

然后，编译运行一下该示例程序，显示一下目标文件内容，结果如图 3-18 所示，其中，将出现“java”字符串的每一行都输出到文件中。



```

zhaodi@zhd-Lenovo-Product: ~/Programs/MapReduce/ResultFilter
usage resultFilter <dfspath> <localpath> <match str> <single file lines>
zhaodi@zhd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$ hadoop jar ResultFilt
er.jar ResultFilter /user/zhaodi/resultTest/origin count java 50
Warning: $HADOOP_HOME is deprecated.

GlobFilter.java
13/12/03 22:50:22 INFO util.NativeCodeLoader: Loaded the native-hadoop library
GlobPattern.java
HarFileSystem.java
HardLink.java
InMemoryFileSystem.java
LocalDirAllocator.java
LocalFileSystem.java
MD5MD5CRC32FileChecksum.java
Path.java
PathFilter.java
PositionedReadable.java
RawLocalFileSystem.java
Seekable.java
Syncable.java
Trash.java
package.html
zhaodi@zhd-Lenovo-Product:~/Programs/MapReduce/ResultFilter$

```

图 3-18 运行效果 01

# 相关图书推荐

