大数据与并行计算

赵银亮
西安交通大学
2014.12.10

引言

理论方法	实验方法	科学计算	大数据分析
• 牛顿定律 • 相对论 • 几何学	• 强子对撞机 • 威尔逊云室	• 气象预报 • 石油勘探 • 流体模拟	• 预测 • ?

信息技术

- 大数据分析被认为是科学方法论的第四个范式

2013年3月22日，奥巴马政府宣布投资2亿美元拉动大数据相关产业发展，将“大数据战略”上升为国家意志，奥巴马政府将数据定义为“未来的新石油”

新加坡：数据就是未来流通的货币
日本：2013年6月，安倍内阁正式公布了新IT战略—“创建最尖端IT国家宣言”

报告内容

- 大数据有关概念
- 大数据应用现状
- 并行计算技术简述
- 大数据油田应用探讨

什么是大数据

- 大数据可以被视作一种比率——我们能计算的数据比上我们必须计算的数据。大数据一直存在。
- 大数据成为问题是在技术允许我们收集和存储的数据超过了我们对系统精推细研的能力之后。

大数据印象

- 案例1. 流感预测
  - 2009年甲型H1N1流感爆发前几周，Google在《自然》发文预测：不仅是全美范围的传播，而且具体到特定的地区和州
  - 如何做？将5000万频繁检索词条和美国疾控中心03和08年传播时期的数据进行比较，建立特定检索词的使用频率与流感在时间空间上的传播之间的联系
  - 共处理4.5亿个不同的数学模型，得到的结果与官方数据相关性高达97%，能判断出从哪里传播出来而且有及时性

大数据印象

- 案例2. 坐姿研究与汽车防盗系统
  - 日本先进工业研究所
  - 人坐着的时候他的身形、姿势和质量分布都可以量化和数据化
  - 在汽车座椅下边安装360个压力传感器以测量人对椅子施加压力的方式，从而将屁股特征转化成数据，得到独属于每个乘坐者的精确数据资料，识别正确率98%
  - 根据压力差异识别乘坐者的身份，用于判断是不是车主
  - 还用于分析坐姿与行驶安全之间关系，预防疲劳驾驶等

大数据印象

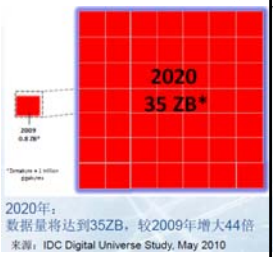
- 案例3：Google翻译系统
  - 谷歌翻译(Google Translate)是目前翻译网页或简短的文字片段使用最多的一个快捷工具，支持该服务的后台核心技术，会在不久的将来被改进为通用翻译器
  - 机器翻译的存在由来已久，但一直远远落后于人工翻译。很多机器翻译软件的开发问题，是如何对不同语言的语法以及词汇进行定义，而这些都不容易解决
  - 通过并行处理大量的可利用的翻译资料，英法语之间的翻译就比旧的通过算法驱动的翻译方法好很多。并行处理的可利用的文本资料库越大，翻译效果就越好
  - 微软、Facebook也做，谷歌和微软用搜索引擎获得海量的数据而Facebook有高达十几亿的实时聊天数据

回头问：什么是数据？

- Data is uninterpreted information.
- Data is any sequence of symbols given meaning by specific acts of interpretation
  - Digital data — is the quantities, characters, or symbols on which operations are performed by a computer stored and recorded on magnetic, optical, or mechanical recording media, and transmitted in the form of electrical signals

大数据的特征：体量大（Volume）

GB TB PB EB ZB YB NB DB
2^{30} 2^{40} 2^{50} 2^{60} 2^{70}



据估计：1B是一粒沙子的话1ZB就是全世界所有海滩的沙子总数

体量大

- 天文学数据
  - 2000年新墨西哥州望远镜几天内收集的数据已超过天文学历史上数据总和，2010年数据量140TB，2016智利大型视场全景巡天望远镜5天内获得同样多数据
- 基因数据
- 地理信息数据
  - 国内2011年1:5万基础地理信息数据库12.3TB，含地形图、航空相片、遥感影像
  - Google数字地球，据2010年李开复估算有近500PB
- 个人位置数据
  - 由GPS芯片、移动基站产生，2009年达到1-3PB

体量大

Facebook共有数据15PB，每天新增60TB

Content Type	Quantity		Comments
Internet	20	Exabytes (10^{18})	1 exabyte = 1,000,000 terabytes
Web Pages	1.5	Trillion (10^{12})	Plus "dark Web"
Tweets	20	Billion (10^{9})	50 million user accounts
Live Posts	2.1	Billion (10^{9})	Forums, discussion boards
Social Members	2.1	Billion (10^{9})	Memberships — top 115 social sites
Social Content Creators	600	Million (10^{6})	People (33% of Internet users)
Facebook Members	500	Million (10^{6})	40% of online hours, top 10 properties
YouTube Visitors	375	Million (10^{6})	As of December 2009
Blogs	70	Million (10^{6})	36,718 listed on Technorati
Formal Periodicals	10s	Thousands (10^{3})	Newspapers, other publications

Gartner Inc.大数据引领估计IT市场：2012 $28b 2013 $34b 2016 $232b

大数据的特征：纷杂(Variety)

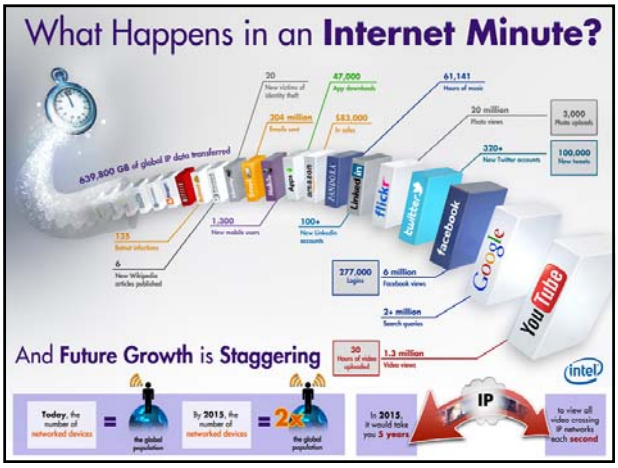
- 各种类型
- 非结构化为主
- 不完全精确
- 不完全正确

纷杂：非结构化数据

- 结构化
  - 包含预定义的数据类型、格式和结构的数据
  - 财务数据、联机分析数据
- 半结构化
  - 具有可识别的模式并可以解析的文本数据文件
  - 自描述和具有自定义模式的XML文档
- 准结构化
  - 具有不规则数据格式的文本数据，通过使用工具可以使之格式化
  - 包含不一致的数据值和格式的网站点击数据
- 非结构化
  - 没有固定结构的数据，通常保存为不同类型的文件
  - 文本文档，PDF文档，图像、声音和视频文件

大数据的特征：快速（Velocity）

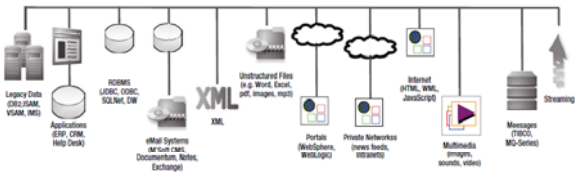
- 源源不断收集，快速积累
  - 互联网每天产生数据
  - 位置数据
  - 社交网络
  - 工业传感器
- 即时得到结果
  - Google搜索



大数据的特征：价值密度低（Value）

- 一段视频可能有用的只有几秒
- 引起的问题：
  - 预计到2020年存储再次成为瓶颈

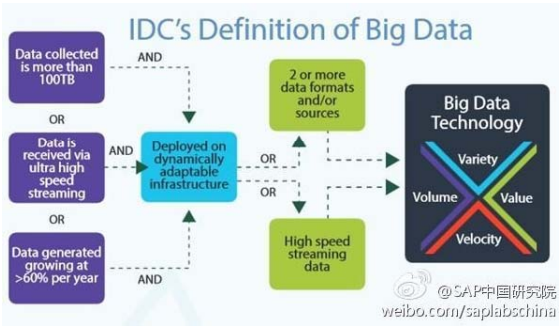
大数据的数据来源



- 大数据来源于以各种目的收集、从各种渠道收集的数据

本有就有	专门收集	遗产数据	共享数据
• 电商 • 社交网络	• 生产数据 • 交易数据	• 企业数据库	• 第三方提供

对大数据的一种量化定义



大数据思维：让数据说话

- 大数据改变了人们思维方式
- 样本=全集
  - 在大数据时代我们可以分析更多的数据，有时候甚至可以处理和某个特别现象相关的所有数据，而不再依赖于随机采样
- 不追求精确追求快速
  - 研究数据如此之多，以至于我们不再热衷于追求精确性
- 重关联，轻因果
  - 不再热衷于寻找因果关系
  - 多问“是什么？”，少问“为什么？”

报告内容

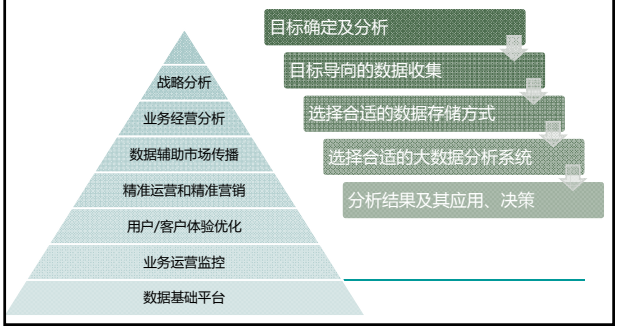
- 大数据有关概念
- 大数据应用现状
- 并行计算技术简述
- 大数据油田应用探讨

大数据能干什么

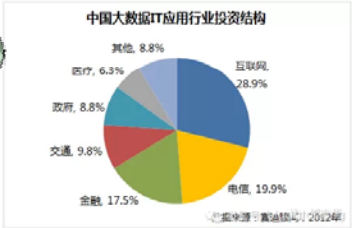
- 大数据技术过程
  - 大数据技术是将非结构化的海量数据在一定时间内完成分析并得出有用的结果，来帮助用户完成决策
  - IDC这样定义大数据技术：大数据技术将被设计用于在成本可承受(economically)的条件下，通过非常快速(velocity)的采集、发现和分析，从大量化(volume)、多类别(variety)的数据中提取价值(value)，将是IT领域新一代的技术与架构的变革。
- 大数据典型应用领域
- 大数据的局限性

大数据技术过程

- 大数据的核心是预测，目标是做决策

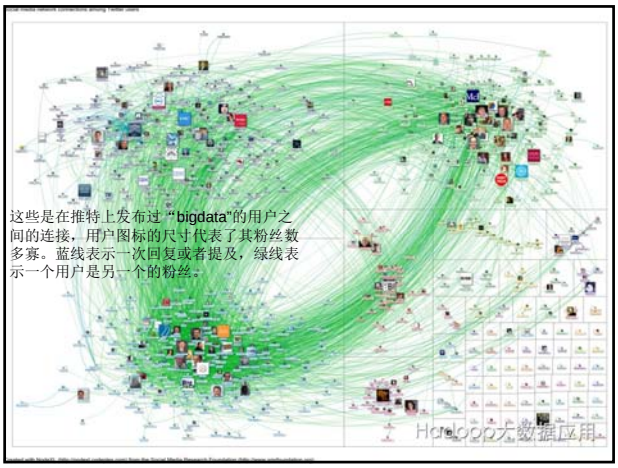


大数据典型应用领域



印第安纳大学Truthy（可信）项目

- 互联网上的政治言论已呈燎原之势
  - 推特圈上每天要出现超过5亿条推文
- 项目目标是从这种每日的信息泛滥中发掘出深层意义
  - 每一天，该项目的计算机过滤多达5千万条推文，试图找出其中蕴含的模式。
- 一个主要的兴趣点是“水军”，靠技术掌握
  - 协调一致的造势运动实际上是由“热衷传播虚假信息的个人和组织”发起的
  - 可辨别由程序反复发布选定信息的假账号
  - 破解2012抹黑米特·罗姆尼在脸谱买粉的造势运动

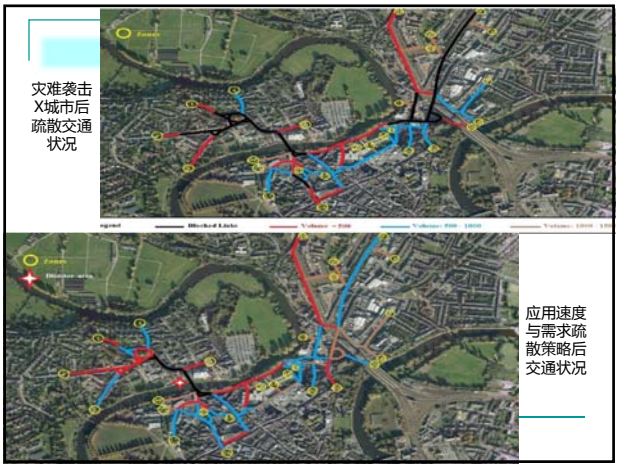


智慧城市灾难应急

- 智慧城市（Smart Cities）依赖一致的无处不在的基础架构，通过有效利用资源给市民提供高质量的生活
- 应急响应系统重要性和用途在显著增加
- 这篇文章提出需求策略和速度策略优化城市疏散行动所需配备，在车辆用量上得到优化

Zubaida Alazawi, et.al. A Smart Disaster Management System for Future Cities, WiMobCity'14, August 11, 2014.

灾难袭击X城市后疏散交通状况



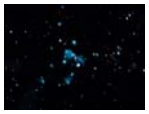
应用速度与需求疏散策略后交通状况

星系动物园项目

- 一台最先进的望远镜扫描整个天空，可能会看到2000亿个这样的恒星世界
- 一系列与宇宙学和星系统计学相关的问题可以通过让许多人做相当简单的分类工作得以解决
- 星系动物园依赖统计学、众多观察者以及处理、检查数据的逻辑来完成星系分类任务
- 最终，有了类别，软件可能会取代志愿者

星系动物园项目

- 绿色豌豆
- 宇宙幽灵
- 尖叫星系
- 天使星系
- 茶壶星系


工业大数据

- 工业大数据是指在工业领域信息化应用中所产生的大数据
- 随着信息化与工业化的深度融合，信息技术渗透到了工业企业产业链的各个环节
  - 条形码、二维码、RFID、工业传感器、工业自动控制系统、工业物联网、ERP、CAD/CAM/CAE/CAI
  - 互联网、移动互联网、物联网
- 工业大数据应用所面临的问题和挑战并不比互联网行业的大数据应用少，某些情况下甚至更为复杂
  - 工业设备所产生数据量远大于企业中计算机和人工产生的数据，多是非结构化，实时性要求也高


工业大数据典型应用：产品创新

- 福特公司第一代福克斯电动车在驾驶和停车时产生大量数据。在行驶中，司机据此持续地更新车辆的加速度、刹车、电池充电和位置信息
- 公司据此了解客户的驾驶习惯，包括如何何时何处充电
- 即使车辆处于静止状态，它也会持续将车辆胎压和电池系统的数据传送给最近的智能电话

工业大数据典型应用：产品故障诊断与预测

- 波音的飞机上，发动机、燃油系统、液压和电力系统等数以百计的变量组成了在航状态，这些数据不到几微秒就被测量和发送一次。737: 10TB/30min
- 不仅得到某个时间点能够分析的工程遥测数据，而且还促进了实时自适应控制、燃油使用、零件故障预测和飞行员通报，能有效实现故障诊断和预测

工业大数据典型应用：产品故障诊断与预测

- 位于美国亚特兰大的GE能源监测和诊断（M&D）中心，收集全球50多个国家上千台GE燃气轮机的数据，每天就能为客户收集10G的数据
- 通过分析来自系统内的传感器振动和温度信号的恒定大数据流，这些大数据分析将为GE公司对燃气轮机故障诊断和预警提供支撑

典型应用：工业物联网生产线

- 现代化工业制造生产线安装有数以千计的小型传感器，来探测温度、压力、热能、振动和噪声。因为每隔几秒就收集一次数据
- 实现设备诊断、用电量分析、能耗分析、质量事故分析（包括违反生产规定、零部件故障）等

在生产工艺改进方面，在生产过程中使用这些大数据，就能**分析整个生产流程**，了解每个环节是如何执行的。一旦有某个流程偏离了标准工艺，就会产生一个报警信号，能更快速地发现错误或者瓶颈所在，也就能更容易解决问题。利用大数据技术，还可以对工业产品的生产过程建立虚拟模型，仿真并优化生产流程，当所有流程和绩效数据都能在系统中重建时，这种透明度将有助于制造商**改进其生产流程**。再如，在能耗分析方面，在设备生产过程中利用传感器集中监控所有的生产流程，能够发现能耗的异常或峰值情形，由此便可在生产过程中**优化能源的消耗**。对所有流程进行分析将会大大降低能耗。

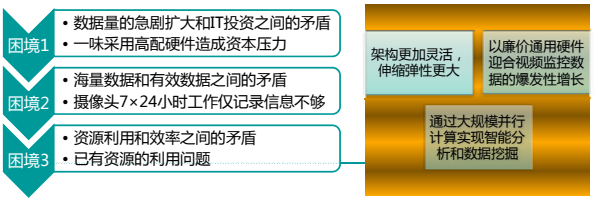
典型应用：工业供应链的分析和优化

- RFID等产品电子标识技术、物联网技术以及移动互联网技术能帮助工业企业获得完整的产品供应链的大数据，利用这些数据进行分析，将带来仓储、配送、销售效率的大幅提升和成本的大幅下降。
- 在海尔供应链的各个环节，客户数据、企业内部数据、供应商数据被汇总到供应链体系中，通过供应链上的大数据采集和分析，海尔公司能够持续进行供应链改进和优化，保证了海尔对客户的敏捷响应

美国较大的OEM供应商超过千家，为制造企业提供超过1万种不同的产品，每家厂商都依靠市场预测和其他不同的变量，如销售数据、市场信息、展会、新闻、竞争对手的数据，甚至天气预报等来销售自己的产品。利用销售数据、产品的传感器数据和出自供应商数据库的数据，**工业制造企业便可准确地预测全球不同区域的需求**。由于可以跟踪库存和销售价格，可以在价格下跌时买进，所以制造企业便可节约大量的成本。如果再利用产品中传感器所产生的数据，知道产品出了什么故障，哪里需要配件，他们还可以预测何处以及何时需要零件。这将会极大地减少库存，优化供应链

典型应用：中大型视频监控

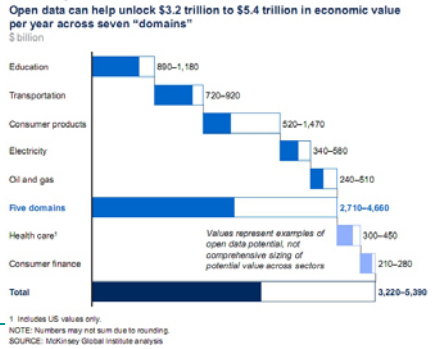
- 大数据视频监控构架带来的价值：让视频监控从人工抽检，进步到高效事前预警、在线和事后分析
  - 智慧城市：精准执法；处理案件；指挥调度；智能交通
  - 实现基于大数据的视频监控云服务，为企业商店餐馆酒店等提供实时监控视频和潜在风险管理，甚至基于视频内容的分析报告，创造新的商业模式。



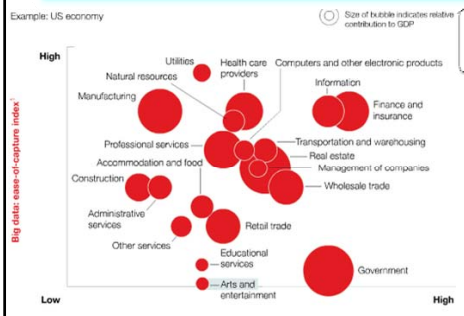
大数据不断增长的商业影响

- 随着我们通过电话、信用卡、电子商务、互联网和电子邮件留下更多的生活痕迹，新商业行为不断出现：
  - 你搜索一条飞往塔斯卡鲁萨的航班，然后便看到网站上出现了塔斯卡鲁萨的宾馆打折信息
  - 你观赏的电影采用了以几十万G数据为基础的计算机图形图像技术
  - 你光顾的商店在对顾客行为进行数据挖掘的基础上获取最大化的利润
  - 用算法预测人们购票需求，航空公司以不可预知的方式调整价格
  - 智能手机的应用识别到你的位置，因此你收到附近餐厅的服务信息

Open Data时代里七大行业潜在的经济价值



各个行业利用大数据价值的难易度以及发展潜力

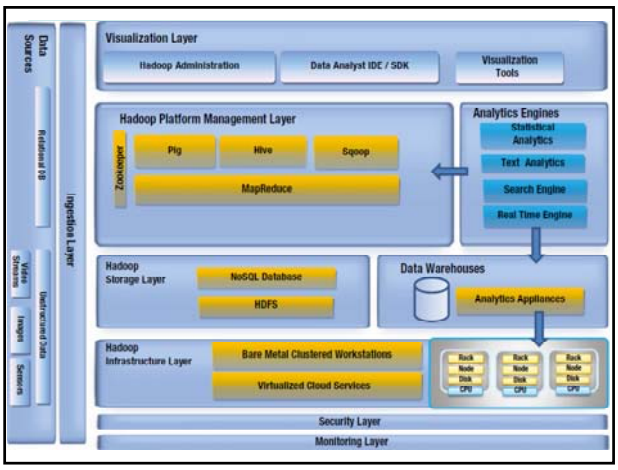


大数据挖掘商业价值的方法

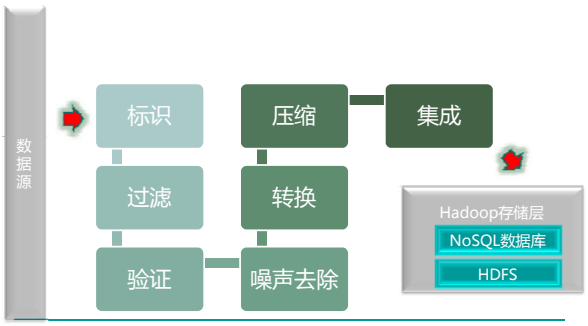
- 客户群体细分，然后为每个群体量定制特别的服务。
- 模拟现实环境，发掘新的需求同时提高投资的回报率
- 加强部门联系，提高整条管理链条和产业链条的效率
- 降低服务成本，发现隐藏线索进行产品和服务的创新

大数据技术架构

- 系统结构
- 技术模型



数据收集层



Hadoop生态系统

HBase是Hadoop数据库，一个分布式、可扩展的大数据存储。

Hive是Hadoop的一个数据仓库系统

Sqoop是一个用来将Hadoop和关系型数据库中的数据相互转移的工具，

Pig是SQL-like语言，是在MapReduce上构建的一种高级查询语言，用于大型数据集分析

Cassandra是一个高性能、可线性扩展、高有效性数据库，可以运行在商用硬件或云基础设施上打造完美的任务关键性数据平台。

Hadoop Common：Hadoop体系最底层的一个模块，为Hadoop各子项目提供各 种工具，如：配置文件和日志操作等

ZooKeeper是一个针对大型分布式系统的可靠协调系统，提供的功能包括：配置维护、名字服 务、 分布式同步、组服务等。

Apache Mahout是个可扩展的机器学习和数据挖掘库，

小结

- 大数据技术是将非结构化的海量数据在一定的时间内完成分析得到有用的结果，来帮助用户完成决策。
- 面对这个由数据堆砌起的复杂世界，或许人类自己会被外界的表象迷惑，甚至迷失自己，但“上帝会说谎，数据却永远不会说谎”。当对海量数据进行检索、比对和分析后，机器可以比你更加了解你的一切，比如你只是在网上检索了几个关键词，搜索引擎服务商们总能在过去海量数据中比对出你此时此刻的潜意识是什么。当无数个你被叠加后，通过大数据便可知道这个世界。

- 随着大规模并行计算、云计算等新技术的崛起，大数据的应用不仅变成可能，而且门槛越来越低，其身影频繁出现在普通商业机构的决策之中，特别是那些快消费品企业。
- 借助大数据，这些企业获得了“商业智能”的能力，使企业的各级决策者可以迅速获得感知市场变化的洞察力，促使他们做出对企业更有利的决策，使得这些企业拥有更强大的创新力。
- 相对于第三产业大数据应用浪潮，第一产业大数据应用相对滞后。

大数据的局限性

- 开放和共享问题
- 隐私和安全问题

大数据的偏见和隐私问题

- “有了足够的数据，数字就可以自己说话” 不是
  - 数据集仍然是人类设计的产物并不能使我们摆脱曲解、隔阂和错误的成见，就像它们存在于个人的感觉和经验中一样
  - 数据源问题：皮尤研究中心获悉，美国上网的成年人中只有16%使用推特网，年轻人和城市人的比例偏多，推特网可能存在多达2000万个虚假账号，此外还有机器人账号
- “大数据将使我们的城市变得更加智能和高效” 一定程度
  - 大数据可以提供帮助改善我们城市的宝贵见识

- 大数据集存在“信号问题”———即某些民众和社区被忽略或未得到充分代表，这被称为数据黑暗地带或阴影区域
- 如果城市开始依靠仅来自智能手机用户的信息，那么这些市民只是一个自我选择样本———它必然导致拥有较少智能手机用户的社区的数据缺失，这样的社区人群通常包括了年老和不那么富有的市民
- 不均衡数据进一步加剧已有的社会不公。2012年“谷歌流感趋势”提示依赖有缺陷的大数据可能给公共服务及公共政策造成影响
- 在网上公开政府部门数据的“开放政府”计划未必会改善政府的透明度和问责，除非存在可以使公众和公共机构保持接触的机制，更不用说促进政府解释数据并以足够的资源作出反应的能力。

- “大数据对不同的社会群体不会厚此薄彼”几乎不是
  - 大数据有可能被用来搞“划红线”，最近，剑桥大学对脸谱网5.8万个“喜欢”标注进行的大数据研究被用来预测用户极其敏感的个人信息，如性取向、种族、宗教和政治观点、性格特征、智力水平、快乐与否、成瘾药物使用、父母婚姻状况、年龄及性别等。“此类容易获得的高度敏感信息可能会被雇主、房东、政府部门、教育机构及私营组织用来对个人实施歧视和惩罚。而人们没有任何抗争的手段。”
  - 警方正在求助于大数据的“预测性警事”模型，希望能够为悬案的侦破提供线索，甚至可以帮助预防未来的犯罪。不过，让警方把工作专注于大数据所发现的特定“热点”，存在着强化警方对声誉不佳的社会群体的怀疑以及使差别化执法成为制度的危险。

- “大数据是匿名的，因此它不会侵犯我们的隐私”大错
  - 蜂窝电话数据看起来也许相当匿名，但是最近对欧洲150万手机用户的数据集进行的研究表明，只需要4项参照因素就足以挨个确认其中95%的人员的身份。
  - 人们在城市中走过的路径存在唯一性，而鉴于利用大量公共数据集可以推断很多信息，这使个人隐私成为“日益严重的担忧”
  - 但是大数据的隐私问题远远超出了常规的身份确认风险的范畴。目前被出售给分析公司的医疗数据有可能被用来追查到你的身份
  - 高度个人化的大数据集将成为黑客或泄露者觊觎的主要目标。

- “大数据是科学的未来”部分正确
  - 大数据为科学提供了新的途径，希格斯玻色子的发现是历史上最大规模网格计算项目的产物。
  - 除非我们认识到并着手解决大数据在反映人类生活方面的某些内在不足，否则我们可能会依据错误的成见作出重大的公共政策和商业决定
  - 新方法将会询问人们做某些事情的原因，而不只是统计某件事情发生的频率。这意味着在信息检索和机器学习之外，还将利用社会学分析和关于人种学的深刻认识。
  - 技术企业很早就意识到社会科学家可以帮助它们更加深刻地认识人们与其产品发生关系的方式和原因

未来会更好

- 我们无法回避这样的事实，即数据绝不是中立的，它很难保持匿名。但是我们可以利用跨越不同领域的专业知识，从而更好地辨别偏见、缺陷和成见，正视隐私和公正将面临的新挑战。
- 回顾人类历史，语言文字出现后，对世界的认识以及积累的知识是以文字记录着，结果使世界越来越好；大数据出现后，对世界的认识以及积累的知识还以大数据记录着，世界同样会更好。

报告内容

- 大数据有关概念
- 大数据应用现状
- 并行计算技术简述
- 大数据油田应用探讨

并行计算技术

- 支撑科学计算的关键技术
  - 提高模拟仿真的速度、精度
  - 代表性的平台是高端并行机（TOP500）
  - 应用在几乎所有学科与计算学科交叉领域，如气象预报、石油勘探等
- 支撑大数据分析的关键技术
  - 提高处理大数据的能力
  - 代表性的平台是大规模分布式并行处理（Hadoop，Spark，云计算）
  - 应用在大数据应用系统的开发、运行、和维护

从并行计算入手理解大数据软件开发

- 提供编程手段来支持开发并行应用程序
  - 通用性
  - 性能

MapReduce | Spark | MPI

OpenMP | CUDA

其中后三种也是科学计算常用的编程语言，说明大数据处理与科学计算不是完全独立的

并行计算技术简介

并行计算技术
- 并行性概念
- 并行机结构模型
- 并行计算模型
- 并行编程模型

并行计算内容概述

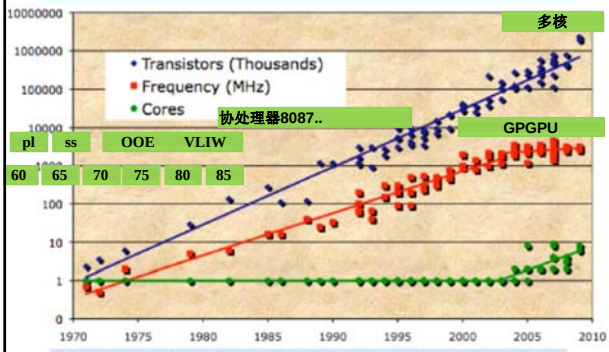
- 处理器进入多核时代，让并行性概念无处不在，从原有专门领域扩展到社会方方面面
- 技术的推动和需求的牵引使得并行机迅速发展，不断为我们想解决而解决不了的问题带来希望
- 开发和维护好的并行应用程序需要哪些能力
  - 掌握并行性有关概念
  - 掌握并行算法的设计与评价方法
  - 善于在并行机模型、并行计算模型和并行编程模型之间求得平衡
  - 理解并行程序开发过程

并行机模型
- 2种SIMD模型
- 5种MIMD模型

并行计算模型
- PRAM
- BSP
- LogP

并行编程模型
- 数据并行
- 共享变量
- 消息传递

处理器进入多核时代，并行性无处不在



The Future of Computing Performance: Game Over or Next Level? The National Academies Press, Washington, DC, 2011.

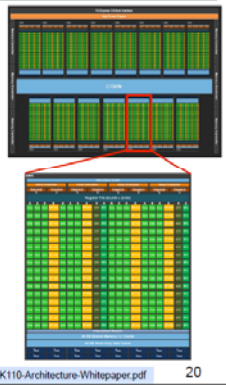
赵银亮 西安交通大学 58

观察

- 处理器发展服从摩尔定律
  - 当价格不变时，集成电路上可容纳的晶体管数目，约每隔18个月便会增加一倍，性能也将提升一倍
- 指令内部并行性开发支持处理器性能提升
- 散热问题催生片上多处理器技术，产生多核处理器
- 通过线程级并行反映多核处理器实际性能
- 图形处理单元已成为核数量巨大的通用计算部件

Recent Processors

- General Purpose Processors
  - Feb 12: Blue Gene/Q
    - 17 cores, 4-way SMT
  - November 11: AMD Interlagos
    - 8 FP cores; 16 integer cores
  - 2011: SPARC T4
    - 8 cores, 8-way fine-grain MT per core
  - Jan 10: IBM Power 7
    - 8 cores, 4-way SMT; 32MB shared cache
  - Nov 08: Intel Nehalem
    - 4 cores, 2-way SMT
- Accelerators
  - Nov 12: Intel Xeon Phi coprocessor
    - ~60 cores
  - May 12: NVIDIA Kepler GK110
    - 15 SMX x 192 CUDA cores (7.1B transistors!)
  - 2008: Tilera TilePro64
    - 64 cores in an 8x8 mesh
- Accelerated Processing Units
  - May 12: AMD Trinity
    - 4 CPU cores; 384 graphics cores
  - Q4 11: Intel Ivy Bridge
    - 4 cores, 2 way SMT; 6-16 GPU execution units



http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf 20

Blue Gene/Q Compute Chip (2012)

- System on a chip
  - processor, memory and network logic
- 360mm², 1.47B transistors
- 16 user + 1 service cores + 1 redundant core
  - all cores are symmetric
  - each 4-way SMT
  - 64-bit Power ISA
  - 1.6GHz
  - 16K L1D, 16K L1I cache
  - L1 prefetch engines
  - each core has 4-way DP FPU
  - peak performance 204.8 GFLOPS @ 55w
- Shared L2 cache: 32MB
  - multi-versioned cache
    - transactional memory, speculative execution, atomic operations
- Dual memory controller
  - 16GB external DDR3 memory
  - 1.3GB/s
  - 2 x 16 byte wide interface (+ECC)
- Chip-to-chip networking
  - router integrated onto chip: support for 5D torus
- External I/O: PCIe Gen2 interface

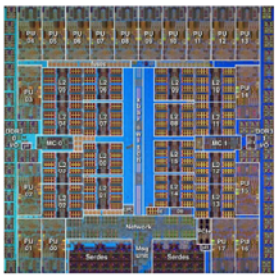


Figure and information credit: Blue Gene/Q compute chip. Ruud Haring. Hot Chips 23. August 2011. http://bit.ly/QWq1ID

21

并行计算的发展动力

- 需求牵引
  - 计算上的建模和模拟是20世纪中科学探索实践最有意义的进展，20多年来，科学计算对各学科都有重要贡献。科学计算特别对于这些问题有用：传统科学理论和实验方法无法解决，实验室研究起来是灾难性的，用传统方法太耗时、太昂贵。

"Sientific Discovery through Advanced Computing", DOE Office of Science, 2000

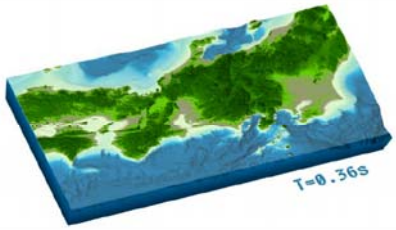
- 技术推动
  - 高性能计算使得大规模计算结点组成计算平台成为可能
  - 处理器、存储、网络技术发展使得高性能、大容量、高带宽计算资源不断刷新指标，并且价格下降

The Need for Speed: Complex Problems

- Science
  - understanding matter from elementary particles to cosmology
  - storm forecasting and climate prediction
  - understanding biochemical processes of living organisms
- Engineering
  - combustion and engine design
  - computational fluid dynamics and airplane design
  - earthquake and structural modeling
  - pollution modeling and remediation planning
  - molecular nanotechnology
- Business
  - computational finance
  - information retrieval
  - data mining
- Defense
  - nuclear weapons stewardship
  - cryptology

Earthquake Simulation

[redacted]

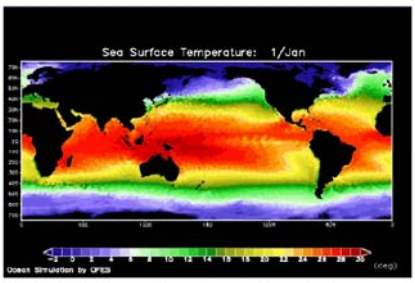


[redacted]

Earthquake Research Institute, University of Tokyo
Tonankai-Tokai Earthquake Scenario
Photo Credit: The Earth Simulator Art Gallery, CD-ROM, March 2004

赵银亮 西安交通大学 64

Ocean Simulation



Ocean Global Circulation Model for the Earth Simulator
Seasonal Variation of Ocean Temperature
Photo Credit: The Earth Simulator Art Gallery, CD-ROM, March 2004

赵银亮 西安交通大学 65

Community Earth System Model (CESM)

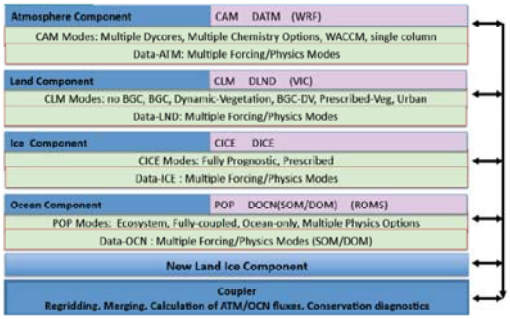


Figure courtesy of M. Vertenstein (NCAR)

27

CESM Execution Configurations

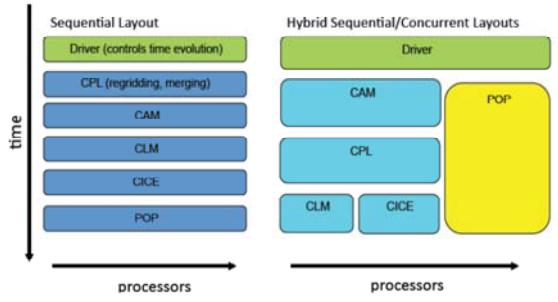
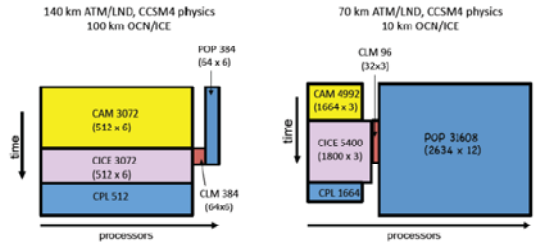


Figure courtesy of M. Vertenstein (NCAR)

28

CESM Simulations on a Cray XT



32.7 SYPD on 3,844 cores
(P processes x T threads)

3.9 SYPD on 37,104 cores
(P processes x T threads)

Performance Limiters: Left is CAM; Right is POP.

Figure courtesy of Pat Worley (ORNL)

29

Fluid-Structure Interactions

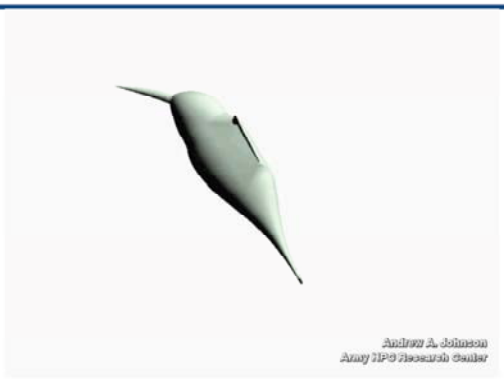
- Simulate …
  - —rotational geometries (e.g. engines, pumps), flapping wings
- Traditionally, such simulations have used a fixed mesh
  - —drawback: solution quality is only as good as initial mesh
- Dynamic mesh computational fluid dynamics
  - —integrate automatic mesh generation within parallel flow solver
    - – nodes added in response to user-specified refinement criteria
    - – nodes deleted when no longer needed
    - – element connectivity changes to maintain minimum energy mesh
  - —mesh changes continuously as geometry + solution changes
- Example: 3D simulation of a hummingbird's flight

[Andrew Johnson, AHPCRC 2005]

31

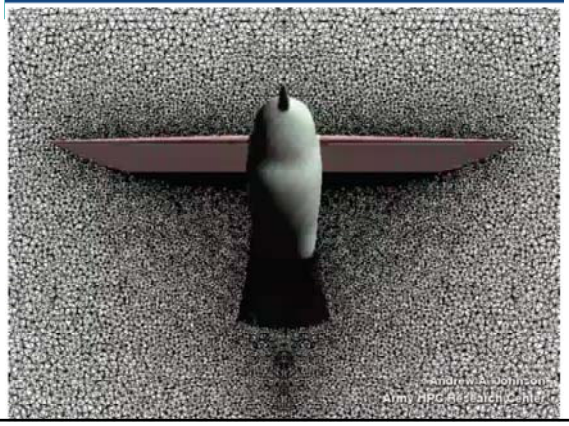
Air Velocity (Front)

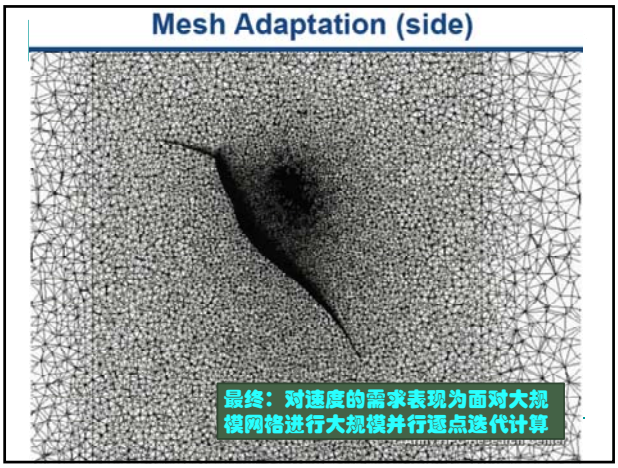
Air Velocity (Side)

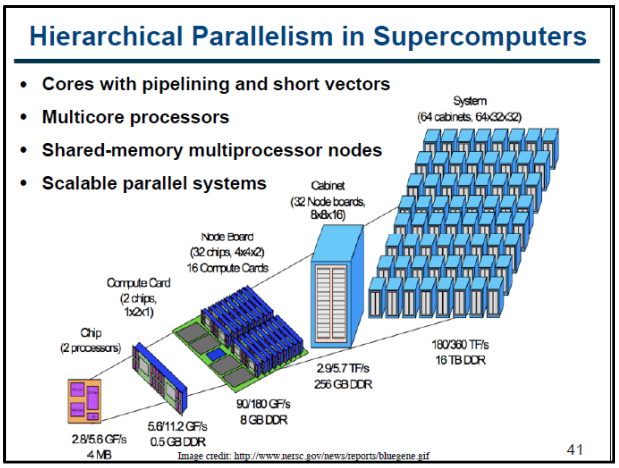


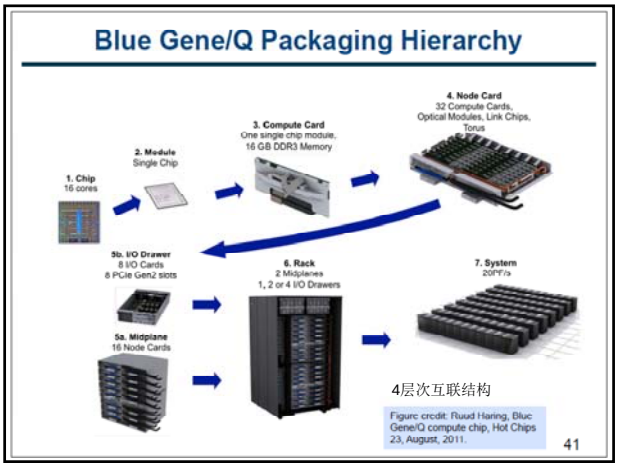
33

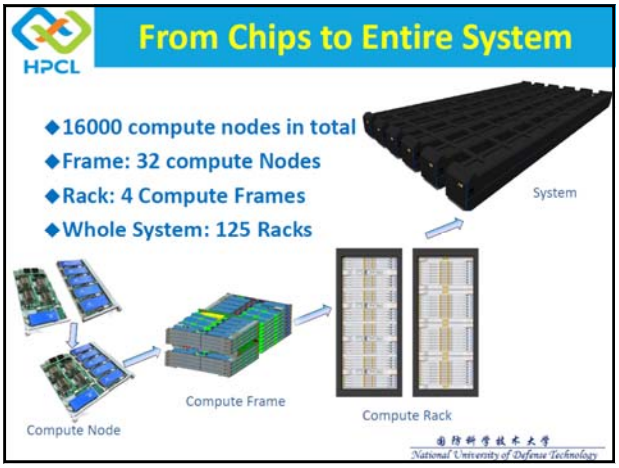
Mesh Adaptation (front)



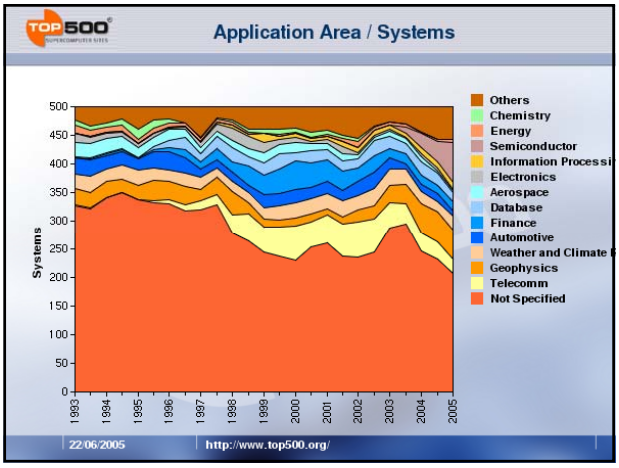

Mesh Adaptation (side)
最终：对速度的需求表现为面对大规模网格进行大规模并行逐点迭代计算


Hierarchical Parallelism in Supercomputers
Cores with pipelining and short vectors
Multicore processors
Shared-memory multiprocessor nodes
Scalable parallel systems
System (64 cabinets, 64x32x32)
Cabinet (32 Node boards, 8x8x16)
Node Board (32 chips, 4x4x2) 16 Compute Cards
Compute Card (2 chips, 1x2x1)
Chip (2 processors)
180/360 TF/s 16 TB DDR
2.9/5.7 TF/s 256 GB DDR
90/180 GF/s 8 GB DDR
5.6/11.2 GF/s 0.5 GB DDR
2.8/5.6 GF/s 4 MB
Image credit: http://www.nersc.gov/news/reports/bluegene.gif
41


Blue Gene/Q Packaging Hierarchy
1. Chip 16 cores
2. Module Single Chip
3. Compute Card One single chip module, 16 GB DDR3 Memory
4. Node Card 32 Compute Cards, Optical Modules, Link Chips, Torus
5a. Midplane 16 Node Cards
5b. I/O Drawer 8 I/O Cards 8 PCIe Gen2 slots
6. Rack 2 Midplanes 1, 2 or 4 I/O Drawers
7. System 20PF/s
4层次互联结构
Figure credit: Ruud Haring, Blue Gene/Q compute chip, Hot Chips 23, August, 2011.
41


HPCL
From Chips to Entire System
16000 compute nodes in total
Frame: 32 compute Nodes
Rack: 4 Compute Frames
Whole System: 125 Racks
System
Compute Node
Compute Frame
Compute Rack
National University of Defense Technology


高端并行机的规模与性能
Cores
Rmax (TFlop/s)
Rpeak (TFlop/s)
Power (kW)
1 National University of Defense Technology China Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT 3120000 33862.7 54902.4 17808
2 DOE/SC/Oak Ridge National Laboratory United States Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc. 560640 17590.0 27112.5 8209
3 DOE/NNSA/LLNL United States Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM 1572864 17173.2 20132.7 7890
4 RIKEN Advanced Institute for Computational Science (AICS) Japan K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu 705024 10510.0 11280.4 12660
5 DOE/SC/Argonne National Laboratory United States Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM 786432 8586.6 10066.3 3945


TOP500
Application Area / Systems
Systems
Others
Chemistry
Energy
Semiconductor
Information Processi
Electronics
Aerospace
Database
Finance
Automotive
Weather and Climate
Geophysics
Telecomm
Not Specified
22/06/2005
http://www.top500.org/

小结

- 并行性不仅是计算机系统结构领域关注点，也成为应用软件领域热点
- 高端并行机规模和能力达到惊人地步
- 科学计算成为支撑着科学进步和技术发展的关键之一

回顾：什么是并行性？

- 并行性(parallelism)问题中具有可以同时进行运算或操作的特性
  - 开发并行性的目的是提高求解问题的效率
- 并行性的体现方式
  - 同时性(simultaneity)指两个或多个事件在同一时刻发生
  - 并发性(concurrency)指两个或多个事件在同一时间间隔内发生
- 体现并行性的层次
  - 指令内部；指令之间；线程级；进程级

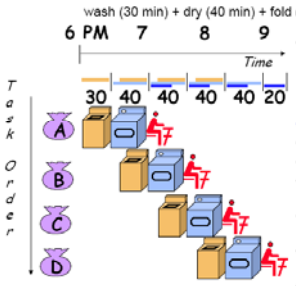
并行性概念

- 开发和利用并行性的途径
  - 时间重叠：在并行性概念中引入时间因素,让多个处理过程在时间上相互错开,轮流重叠地使用同一套硬件设备的各个部分,以加快硬件周转而赢得速度. 如流水线
  - 资源重复：在并行性概念中引入空间因素,通过重复设置硬件资源来提高可靠性或性能. 如双机系统
  - 资源共享：利用软件的方法让多个用户按一定时间顺序轮流地使用同一套资源,以提高其利用率, 可相应提高系统性能

What is Pipelining?

Dave Patterson's Laundry example: 4 people doing laundry

wash (30 min) + dry (40 min) + fold (20 min) = 90 min Latency



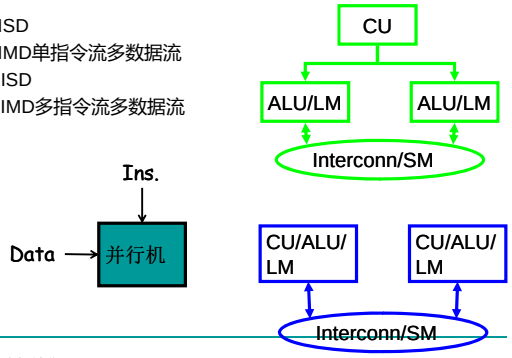
- In this example:
  - Sequential execution takes 4 * 90min = 6 hours
  - Pipelined execution takes 30+4*40+20 = 3.5 hours
- Bandwidth = loads/hour
- BW = 4/6 l/h w/o pipelining
- BW = 4/3.5 l/h w pipelining
- BW <= 1.5 l/h w pipelining, more total loads
- Pipelining helps bandwidth but not latency (90 min)
- Bandwidth limited by slowest pipeline stage
- Potential speedup = Number pipe stages

并行计算技术简介

- 并行计算技术
  - 并行性概念
  - 并行机结构模型
  - 并行计算模型
  - 并行编程模型

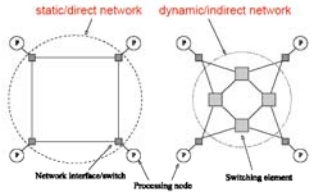
并行计算机分类(Flynn)

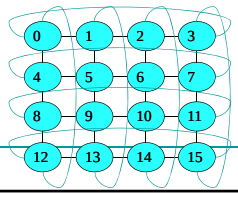
- SISD
- SIMD单指令流多数据流
- MISD
- MIMD多指令流多数据流

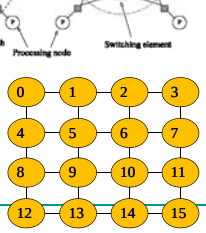


并行机结构模型：互联网络

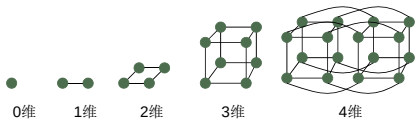
- 网格
- 洗牌交换网络
- 超立方体
- 总线
- 交叉开关



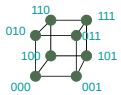




超立方体互联结构



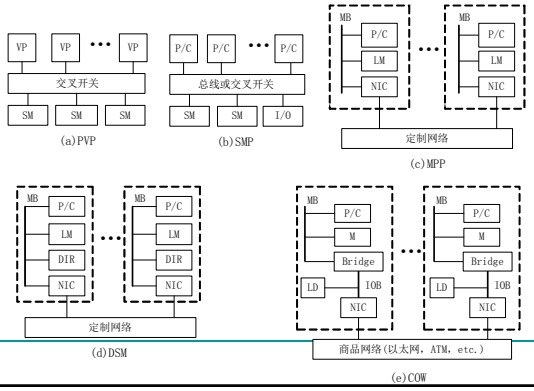
- 结点个数n = 2^d 其中d为维数
- 基于格雷码寻址



$$CC_i(p_{m-1}\cdots p_{i+1}p_ip_{i-1}\cdots p_0)=p_{m-1}\cdots p_{i+1}\overline{p_i}p_{i-1}\cdots p_0,\quad 0\le i<m$$

算法 n=2^d的超立方体上n个数求和
输入：每个结点一个数；输出：每个结点都有Σn
for i=0 to d-1 do
 for each p∈P par do p与$CC_i(p)$交换并各自求和 endfor
endfor

MIMD并行机示意



从应用角度看MIMD并行机结构模型

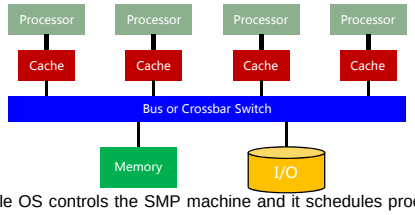
- 对MIMD并行机归类时考虑两个方面:
  - 存储器在物理上是集中式的（共享的）还是分布式的
  - 是单地址空间还是多地址空间

Memory		Address Space	
		Shared	Individual
	Centralized	UMA – SMP (Symmetric Multiprocessor)	N/A
	Distributed	NUMA (Non-Uniform Memory Access)	MPP (Massively Parallel Processors)

88

对称多处理器(SMP)

- Symmetric Multiprocessor (SMP) architecture uses shared system resources that can be accessed equally from all processors

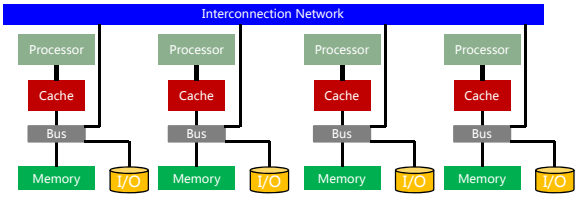


- A single OS controls the SMP machine and it schedules processes and threads on processors so that the load is balanced

89

大规模并行处理器(MPP)

- Massively Parallel Processors (MPP) architecture consists of nodes with each having its own processor, memory and I/O subsystem



- An independent OS runs at each node

90

非均匀存储访问(NUMA/DSM)

- Non-Uniform Memory Access (NUMA) architecture machines are built on a similar hardware model as MPP
- NUMA typically provides a shared address space to applications using a hardware/software directory-based coherence protocol
- The memory latency varies according to whether you access memory directly (local) or through the interconnect (remote). Thus the name non-uniform memory access
- As in an SMP machine, a single OS controls the whole system

91

性能评价

- 通信和计算两个方面
- 并行机、算法、程序三个方面
- 性能指标
  - 加速比（Speedup）串行执行与并行执行时间之比
  - 吞吐率（Throughput）单位时间内完成的并发任务数
  - 效率（Efficiency）加速比与处理器数之比
  - 可扩展性指标（Scalability）随着处理器数目增加性能相应增加的能力

Amdahl定律

- We parallelize our programs in order to run them faster
- How much faster will a parallel program run?
  - Suppose that the sequential execution of a program takes T1 time units and the parallel execution on p processors takes Tp time units
  - Suppose that out of the entire execution of the program, s fraction of it is not parallelizable while 1-s fraction is parallelizable
  - Then the speedup (Amdahl's formula):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$

93

Amdahl定律 ： 例子

- Suppose that 80% of you program can be parallelized and that you use 4 processors to run your parallel version of the program
- The speedup you can get according to Amdahl is:

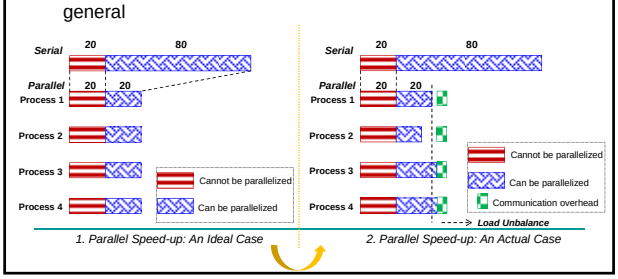
$$\frac{1}{s+\frac{1-s}{p}} = \frac{1}{0.2+\frac{0.8}{4}} = 2.5 \text{ times}$$

- Although you use 4 processors you cannot get a speedup more than 2.5 times (or 40% of the serial running time)

94

理想情形与实际情形

- Amdahl's argument is too simplified to be applied to real cases
- When we run a parallel program, there are a communication overhead and a workload imbalance among processes in general



1. Parallel Speed-up: An Ideal Case 2. Parallel Speed-up: An Actual Case

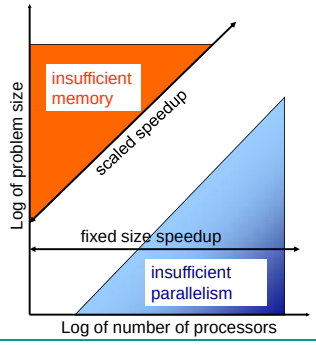
效率原则

- 为有效从并行化中获益应遵从下面原则:
  - 最大化程序中可并行化的部分
  - 对所有进程的负载取得平衡
  - 最小化花费在通信上的时间

Efficiency

- The parallel efficiency of an application
  - Efficiency(p) <= 1
  - For perfect speedup Efficiency (p) = 1
- We will rarely have perfect speedup.
  - Lack of perfect parallelism in the application/algorithm
  - Imperfect load balancing
  - Cost of communication
  - Cost of contention for resources, e.g., memory bus, I/O
  - Synchronization time
- Understanding why an application is not scaling linearly will help finding ways improving the applications performance on parallel computers.

性能极限



可扩放性评测标准

- 并行计算的可扩放性（Scalability）也是主要性能指标
  - 可扩放性最简朴的含意是在确定的应用背景下，计算机系统（或算法或程序等）性能随处理器数的增加而按比例提高的能力
- 影响加速比的因素：处理器数与问题规模
  - 求解问题中的串行分量
  - 并行处理所引起的额外开销（通信、等待、竞争、冗余操作和同步等）
  - 加大的处理器数超过了算法中的并发程度
- 增加问题的规模有利于提高加速的因素：
  - 较大的问题规模可提供较高的并发度；
  - 额外开销的增加可能慢于有效计算的增加；
  - 算法中的串行分量比例不是固定不变的（串行部分所占的比例随着问题规模的增大而缩小）。
- 增加处理器数会增大额外开销和降低处理器利用率，所以对于一个特定的并行系统（算法或程序），它们能否有效利用不断增加的处理器的能力应是受限的，而度量这种能力就是可扩放性这一指标。

可扩放性评测标准（续）

- 可扩放性:调整什么和按什么比例调整
  - 并行计算要调整的是处理数p和问题规模W，
  - 两者可按不同比例进行调整，此比例关系（可能是线性的，多项式的或指数的等）就反映了可扩放的程度。
- 可扩放性研究的主要目的：
  - 确定解决某类问题用何种并行算法与何种并行体系结构的组合，可以有效地利用大量的处理器；
  - 对于运行于某种体系结构的并行机上的某种算法当移植到大规模处理机上后运行的性能；
  - 对固定的问题规模，确定在某类并行机上最优的处理器数与可获得的最大的加速比；
  - 用于指导改进并行算法和并行机体系结构，以使并行算法尽可能地充分利用可扩充的大量处理器
- 目前无一个公认的、标准的和被普遍接受的严格定义和评判它的标准

等效率度量标准

- 令t_e^i 和t_o^i分别是并行系统上第i个处理器的有用计算时间和额外开销时间（包括通信、同步和空闲等待时间等）

$$T_e = \sum_{i=0}^{p-1} t_e^i \qquad T_o = \sum_{i=0}^{p-1} t_o^i$$

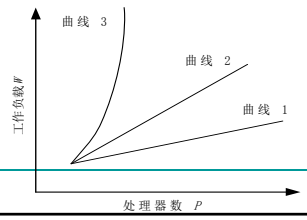
- T_p 是p个处理器系统上并行算法的运行时间，对于任意i显然有 $T_p = t_e^i + t_o^i$，且 $T_e + T_o = pT_p$
- 问题的规模W为最佳串行算法所完成的计算量$W=T_e$

$$S = \frac{T_e}{T_p} = \frac{T_e}{\frac{T_e + T_o}{p}} = \frac{p}{1+\frac{T_o}{T_e}} = \frac{p}{1+\frac{T_o}{W}} \qquad E = \frac{S}{P} = \frac{1}{1+\frac{T_o}{T_e}} = \frac{1}{1+\frac{T_o}{W}}$$

- 如果问题规模W 保持不变，处理器数p增加，开销T_o增大，效率E下降。为了维持效率不变，当处理数p增大时，需要相应地增大问题规模W的值。由此定义等效率函数$f_E(p)$为问题规模W随处理器数p变化的函数

等效率度量标准（续）

- 对于某算法若随着p增加$f_E(p)$增加较小则称其为可扩放的。
- 曲线1表示算法具有很好的扩放性；曲线2表示算法是可扩放的；曲线 3表示算法是不可扩放的。
- 优点：简单可定量计算的、少量的参数计算等效率函数
- 缺点：如果T_o无法计算出（在共享存储并行机中）



并行计算技术简介

- 并行计算技术
  - 并行性概念
  - 并行机结构模型
  - 并行计算模型
  - 并行编程模型

并行算法设计

- 算法复杂性度量
- Brent定理
- PRAM算法
- MPP算法
- PCAM方法学

并行机模型
- 2种SIMD模型
- 5种MIMD模型

并行计算模型
- PRAM
- BSP
- LogP

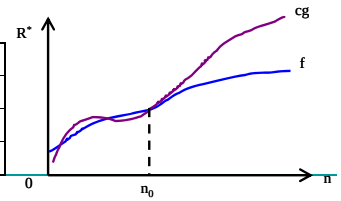
并行编程模型
- 数据并行
- 共享变量
- 消息传递

算法复杂性的渐近表示

$N=\{0,1,2,\ldots\}$ 非负整数集
$N^+=\{1,2,3,\ldots\}$ 自然数集
R 实数集
R^+ 正实数集
$R^*=R^+\cup\{0\}$

$g,f:N^+\to R^*$
f和g之间有如下关系：
$(\exists c\in R^+)(\exists n_0\in N^+)(\forall n\ge n_0)(f(n)\le cg(n))$
称g是f的一个上界，记为$f=O(g)$

f	g	c	n_0
$2n^2$	$3n^2$	>=2/3	1
n^2+2n	n^2	>=3	>=1



算法复杂性度量

- 基本操作
- 问题规模
- 时间复杂度
- 空间复杂度

算法A → 基本操作执行次数 → f(n) → g(n)为f(n)上界 → 算法复杂度为O(g(n))

多项式复杂度与指数复杂度：

$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$

$O(2^n)<O(n!)<O(n^n)$

Brent定理

算法 {
- 抽象描述： 与处理器数和任务分配无关
- 具体描述（实现）： 使用p台处理器

令W(n)是某并行算法A在运行时间T(n)内所执行的运算量，则A使用p台处理器可在t(n)=O(W(n)/p+T(n))时间内执行完毕。

证明：将T(n)分成若干个时间步(并行步)，对应第i个时间步的计算量是$W_i(n)$，在p个处理机时可用不超过$\left\lceil\frac{W_i(n)}{p}\right\rceil$并行步模拟，那么该算法在p个处理机上的执行时间

$$\sum_{i=1}^{T(n)}\left\lceil\frac{W_i(n)}{p}\right\rceil\le\sum_{i=1}^{T(n)}\left(\left\lfloor\frac{W_i(n)}{p}\right\rfloor+1\right)\le\frac{1}{p}\sum_{i=1}^{T(n)}W_i(n)+T(n)\le\frac{1}{p}W(n)+T(n)$$

抽象算法看作T(n)个时间步（也就是并行步，这个总数是正整数，可数），令W_i=抽象算法第i个并行步内的操作数目（运算量），对应的具体算法中所用时间

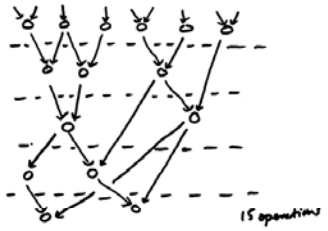
$$\left\lceil\frac{W_i}{p}\right\rceil\le\frac{W_i+p-1}{p}$$

总的时间:

$$\le\sum_{i=1}^{t}\left\lceil\frac{W_i}{p}\right\rceil\le\sum_{i=1}^{t}\frac{W_i+p-1}{p}=\sum_{i=1}^{t}\frac{p}{p}+\sum_{i=1}^{t}\frac{W_i-1}{p}=t+\frac{\sum W_i-\sum 1}{p}=T(n)+\frac{W(n)-T(n)}{p}$$

注意t是并行步的编号，这些编号的个数之和等于T(n)

- 示意



$t = 5$ (6 processors)

To schedule on 2 processors,

1st level	time 3
2nd level	time 2
3rd level	time 1
4th level	time 1
5th level	time 1
	8

Brent

$t_2 \le 5 + \frac{15-5}{2}$

$t_2 \le 10$

So Brent's bound is not tight

例：抽象算法

```
算法. SIMD-SM上的求和
   In: n=2^k, 被求和的数在数组A中。
   Out: S
begin
   (1) for i=1 to n par-do B(i)←A(i) endfor
   (2) for h=1 to logn do
            for i=1 to n/(2^h) par-do
            B(i)←B(2i-1)+B(2i)
            endfor
        endfor
   (3) S←B(1)
end
```

$$W(n) = n + \sum_{j=1}^{\log n} \frac{n}{2^{j-1}} + 1 = O(n) \qquad T(n) = O(\log n)$$

例：具体算法

```
(1) for j=1 to l=n/p do
        B(l(s-1)+j):=A(l(s-1)+j)
    endfor
(2) for h=1 to log n do
    (2.1) if (k-q-h>=0) then
            for j=2^(k-h-q)(s-1)+1 to 2^(k-h-q)s do
            B(j):=B(2j-1)+B(2j)
            endfor
        endif
    (2.2) if(s<=2^(k-h)) then
            B(s):=B(2s-1)+B(2s)
        endif
    endfor
(3) if(s=1)then S:=B(1) endif
end
```

(1) $O(l)=O(n/p)$
(2) 第h次迭代时所用时间$O(n/(2^h p))$
找最长执行时间的p，每次迭代中：段长大于1时，计算量为段长数；段长等于1且段数大于1时，计算量为1。所以总时间：

$$\sum_{j=1}^{\log n} \left\lceil \frac{n}{2^{j-1} p} \right\rceil$$

(1) $O(l)=O(n/p)$
(2) 第h次迭代时所用时间$O(n/(2^h p))$
找最长执行时间的p，每次迭代中：段长大于1时，计算量为段长数；段长等于1且段数大于1时，计算量为1。所以总时间：

$$\sum_{j=1}^{\log n} \left\lceil \frac{n}{2^{j-1} p} \right\rceil$$

(3) $O(1)$

$$\sum_{j=1}^{\log n} \left\lceil \frac{n}{2^{j-1} p} \right\rceil \le \log n + \left\lfloor \sum_{j=1}^{\log n} \frac{n}{2^{j-1} p} \right\rfloor = \log n + \left\lfloor \frac{n}{p} 2 \cdot (1 - \frac{1}{n}) \right\rfloor$$

$$t(n) = O(\frac{n}{p} + \log n)$$

$W(n)=n$, $p=p$, $T(n)=\log n$,所以满足Brent定理。

例：PRAM-CRCW模型求最大值算法

PRAM模型：SIMD，无限大均匀SM，输入输出都在该SM中

Given n elements A[0, n-1], find the maximum.
With n^2 processors, each processor (i,j) compare A[i] and A[j], for $0 \le i, j \le n-1$.

```
FAST-MAX(A):
    n←length[A]
    for i ←0 to n-1, in parallel
        do m[i] ←true
    for i ←0 to n-1 and j ←0 to n-1, in parallel
        do if A[i] < A[j]
            then m[i] ←false
    for i ←0 to n-1, in parallel
        do if m[i] =true
            then max ← A[i]
    return max
```

A[i] \ A[j]	5	6	9	2	9	m
5	F	T	T	F	T	F
6	F	F	T	F	T	F
9	F	F	F	F	F	T
2	T	T	T	F	T	F
9	F	F	F	F	F	T

max=9

The running time is $O(1)$.
Note: there may be multiple maximum values, so their processors Will write to max concurrently. Its $work = n^2 \times O(1) = O(n^2)$.

例：PSRS排序

典型实现为MPI程序用于NORMA结构（MPP结构）如集群上

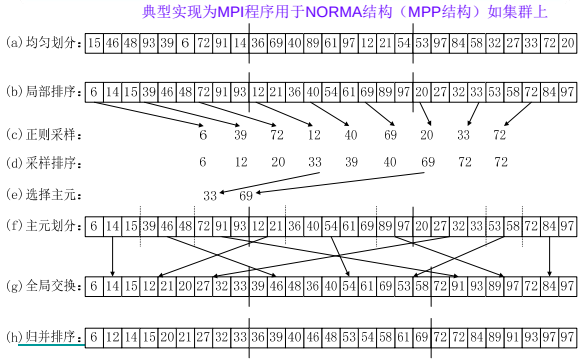


图6.1

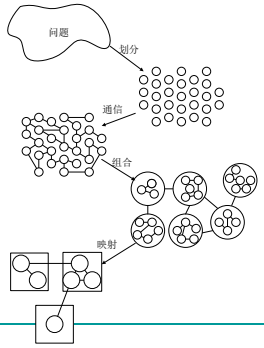
PCAM设计过程

划分：分解成小的任务，开拓并发性；

通信：确定诸任务间的数据交换，监测划分的合理性

组合：依据任务的局部性，组合成更大的任务

映射：将每个任务分配到处理器上，提高算法的性能



并行计算技术简介

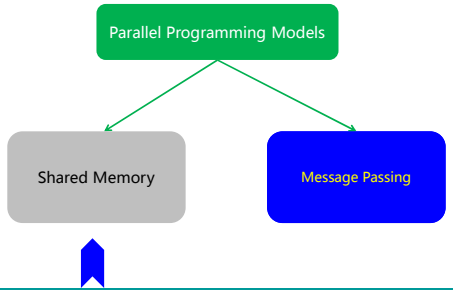
- 并行计算技术
  - 并行性概念
  - 并行机结构模型
  - 并行计算模型
  - 并行编程模型

并行编程模型

- What is a parallel programming model?
  - A programming model is an abstraction provided by the hardware to programmers
  - It determines how easily programmers can specify their algorithms into parallel unit of computations (i.e., tasks) that the hardware understands
  - It determines how efficiently parallel tasks can be executed on the hardware
- Main Goal: utilize all the processors of the underlying architecture (e.g., SMP, MPP, NUMA) and minimize the elapsed time of your program

117

传统编程模型



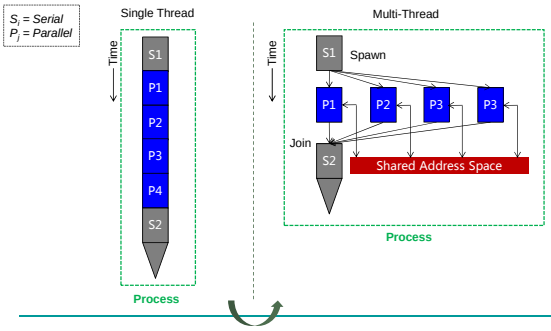
118

Shared Memory Model

- In the shared memory programming model, the abstraction is that parallel tasks can access any location of the memory
- Parallel tasks can communicate through reading and writing common memory locations
- This is similar to threads from a single process which share a single address space
- Multi-threaded programs (e.g., OpenMP programs) are the best fit with shared memory programming model

119

Shared Memory Model



120

Shared Memory Example

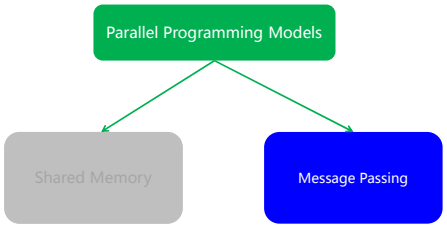
```
for (i=0; i<8; i++)
  a[i] = b[i] + c[i];
sum = 0;
for (i=0; i<8; i++)
  if (a[i] > 0)
    sum = sum + a[i];
Print sum;
```

Sequential

```
begin parallel // spawn a child thread
private int start_iter, end_iter, i;
shared int local_iter=4, sum=0;
shared double sum=0.0, a[], b[], c[];
shared lock_type mylock;

start_iter = getid() * local_iter;
end_iter = start_iter + local_iter;
for (i=start_iter; i<end_iter; i++)
  a[i] = b[i] + c[i];
barrier;

for (i=start_iter; i<end_iter; i++)
  if (a[i] > 0) {
    lock(mylock);
      sum = sum + a[i];
    unlock(mylock);
  }
barrier;    // necessary

end parallel // kill the child thread
Print sum;
```

Parallel

传统编程模型



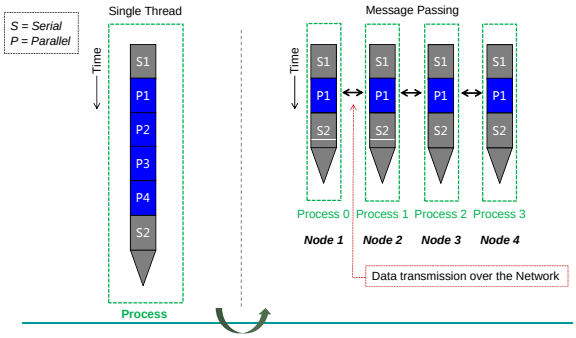
122

Message Passing Model

- In message passing, parallel tasks have their own local memories
- One task cannot access another task's memory
- Hence, to communicate data they have to rely on explicit messages sent to each other
- This is similar to the abstraction of processes which do not share an address space
- MPI programs are the best fit with message passing programming model

123

Message Passing Model



124

Message Passing Example

```
for (i=0; i<8; i++)
  a[i] = b[i] + c[i];
sum = 0;
for (i=0; i<8; i++)
  if (a[i] > 0)
    sum = sum + a[i];
Print sum;
```

Sequential

```
id = getpid();
local_iter = 4;
start_iter = id * local_iter;
end_iter = start_iter + local_iter;

if (id == 0)
  send_msg (P1, b[4..7], c[4..7]);
else
  recv_msg (P0, b[4..7], c[4..7]);

for (i=start_iter; i<end_iter; i++)
  a[i] = b[i] + c[i];

local_sum = 0;
for (i=start_iter; i<end_iter; i++)
  if (a[i] > 0)
    local_sum = local_sum + a[i];
if (id == 0) {
  recv_msg (P1, &local_sum1);
  sum = local_sum + local_sum1;
  Print sum;
}
else
  send_msg (P0, local_sum);
```

Parallel

Shared Memory Vs. Message Passing

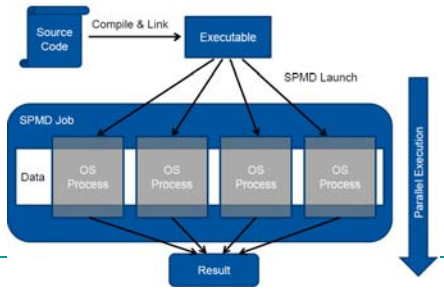
- Comparison between shared memory and message passing programming models:

Aspect	Shared Memory	Message Passing
Communication	Implicit (via loads/stores)	Explicit Messages
Synchronization	Explicit	Implicit (Via Messages)
Hardware Support	Typically Required	None
Development Effort	Lower	Higher
Tuning Effort	Higher	Lower

126

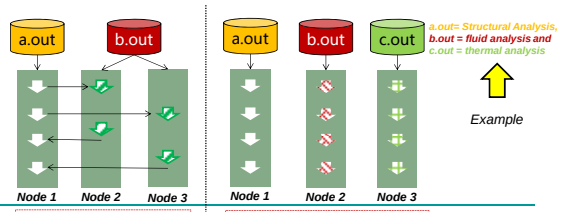
并行编程例子：MPI SPMD/MPMD

- 基于消息传递的多个进程并行执行，是否需要程序级协调
- MPI主要机制：消息传递；笛卡尔拓扑；成组通信等



MPMD

- The MPMD model uses different programs for different processes, but the processes collaborate to solve the same problem
- MPMD has two styles, the master/worker and the coupled analysis



1. MPMD: Master/Slave
2. MPMD: Coupled Analysis

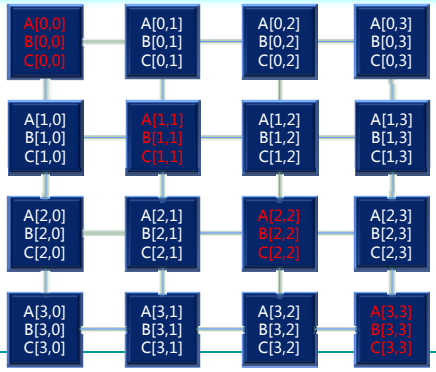
例：Cannon矩阵乘法

- 适用于网格
- 存储有效
- 规则通信
  - 初始化
  - 对齐
  - 循环移位

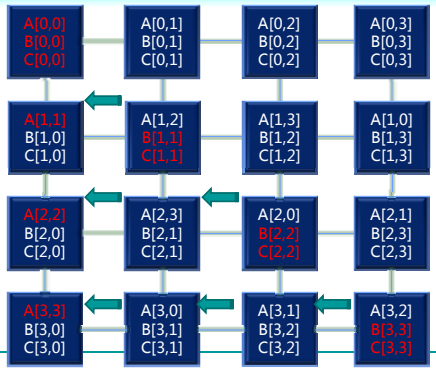
初始化 → 对齐 → 分块乘 → 重复 → 结果

- 移位
- 分块乘

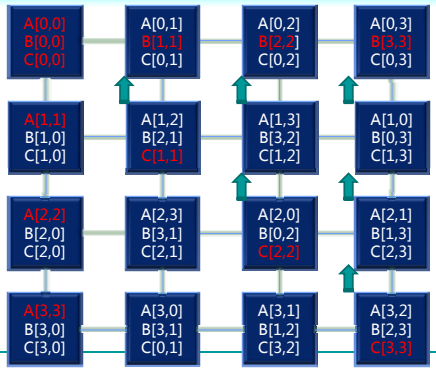
Cannon乘法：初始化



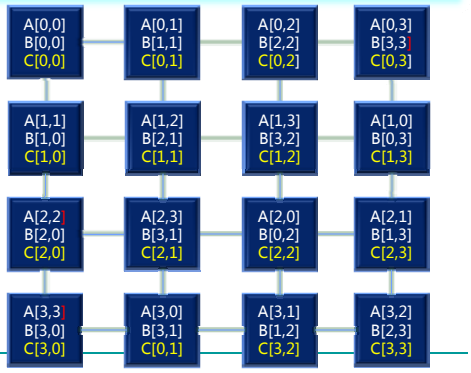
Cannon乘法：对齐A



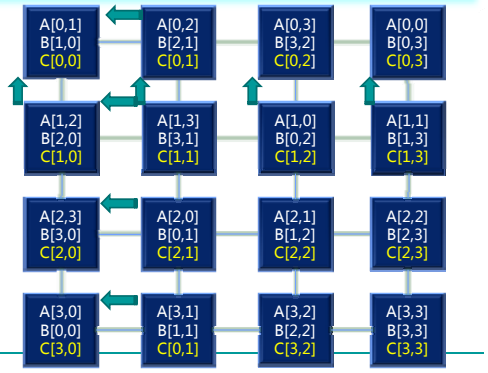
Cannon乘法：对齐B



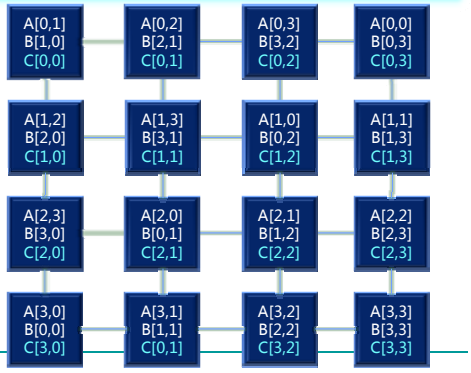
每个结点并行乘分块得到C分块



A和B各自循环移位



各自乘分块并累加到C分块



随后再重复两次，C分块便为最终结果

Cannon矩阵乘法MPI程序

- 参数a, b,和c 分别指向局部存储的分块，分别对应*A*, *B*, 和*C*, 均是n阶方阵.
- 处理器个数为p时分块为sqrt(n)阶方阵，$p=2^t$，t为正整数
- 参数comm 表示调用MatrixMatrixMultiply()的进程标识，通信簇
- 函数接口 `MatrixMatrixMultiply(int n, double *a, double *b, double *c, MPI_Comm comm)`

```
MatrixMatrixMultiply(int n, double *a, double *b,
                     double *c, MPI_Comm comm) {
  int i, nlocal, npes, dims[2], periods[2];
  int myrank, my2drank, mycoords[2];
  int uprank, downrank, leftrank, rightrank, coords[2];
  int shiftsrc, shiftdest;
  MPI_Status status;
  MPI_Comm comm_2d;
  /* Get the communicator related information */
  MPI_Comm_size(comm, &npes);
  MPI_Comm_rank(comm, &myrank);
  /* Set up the Cartesian topology */
  dims[0] = dims[1] = sqrt(npes);
  /* Set the periods for wraparound connections */
  periods[0] = periods[1] = 1;
  /* Create the Cartesian topology, with rank reordering */
  MPI_Cart_create(comm, 2, dims, periods, 1, &comm_2d);
```

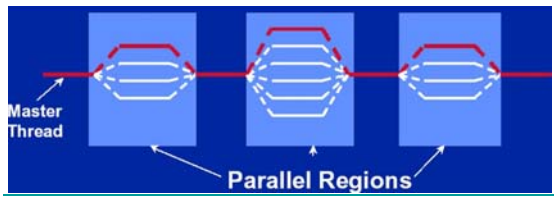
```
  /* Get the rank and coordinates */
  MPI_Comm_rank(comm_2d, &my2drank);
  MPI_Cart_coords(comm_2d, my2drank, 2, mycoords);

  /* Determine the dimension of the local matrix block */
  nlocal = n/dims[0];

  /* Perform the initial matrix alignment. */
  MPI_Cart_shift(comm_2d, 0, -mycoords[0], &shiftsrc
   &shiftdest);
  MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
   shiftdest, 1, shiftsrc, 1, comm_2d, &status);
  MPI_Cart_shift(comm_2d, 1, -mycoords[1], &shiftsrc,
          &shiftdest);
  MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
          shiftdest, 1, shiftsrc, 1, comm_2d, &status);
```

```
  /* Get into the main computation loop */
  for (i=0; i<dims[0]; i++) {
    MatrixMultiply(nlocal, a, b, c); /*c=c+a*b*/

  /* Compute ranks of the up and left shifts */
  MPI_Cart_shift(comm_2d, 0, -1, &rightrank, &leftrank);
  MPI_Cart_shift(comm_2d, 1, -1, &downrank, &uprank);
    /* Shift matrix a left by one */
    MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
        leftrank, 1, rightrank, 1, comm_2d, &status);
    /* Shift matrix b up by one */
    MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
        uprank, 1, downrank, 1, comm_2d, &status);
  }
  /* Restore the original distribution of a and b */
  MPI_Cart_shift(comm_2d, 0, +mycoords[0], &shiftsrc,
   &shiftdest);
  MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
    shiftdest, 1, shiftsrc, 1, comm_2d, &status);
```

```
  MPI_Cart_shift(comm_2d, 1, +mycoords[1], &shiftsrc, &shiftdest);
  MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
      shiftdest, 1, shiftsrc, 1, comm_2d, &status);

  MPI_Comm_free(&comm_2d); /* Free up communicator */
}

/* This function performs a serial matrix-matrix multiplication c = a*b */
MatrixMultiply(int n, double *a, double *b, double *c)
{
  int i, j, k;

  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      for (k=0; k<n; k++)
        c[i*n+j] += a[i*n+k]*b[k*n+j];
}
```

并行编程模型：OpenMP Master-Worker

- 适用于MIMD-SM结构，如多核、GPU
- 线程级并行程序
- 主要机制：循环并行化、线程派生和交互机制



大规模分布式并行编程模型：MapReduce

- MapReduce是一种高度可扩展的编程模型
  - 通过在大量廉价计算结点上并行执行来处理大规模数据（>1TB）
- 1994年从Google提出并流行，今天已经在许多如Apache Hadoop开源项目中实现
- MapReduce成为大数据处理主要手段
  - 高度可扩展性
  - 容错性
  - 简单性
  - 独立于编程语言以及数据存储系统

MapReduce思想来源于LISP（Scheme）

- (map ***f list [list$_2$ list$_3$ …]***)
- (map square '(1 2 3 4))
  - (1 4 9 16)
- (reduce ***f id list***)
- (reduce + 0 '(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 (+ 1 0))))
  - 30
- (reduce + 0 (map square (map – l1 l2))))

MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

144

Problem Scope

- MapReduce is a programming model for data processing
- The power of MapReduce lies in its ability to scale to 100s or 1000s of computers, each with several processor cores
- How large an amount of work?
  - Web-Scale data on the order of 100s of GBs to TBs or PBs
  - It is likely that the input data set will not fit on a single computer's hard drive
  - Hence, a distributed file system (e.g., Google File System-GFS) is typically required

145

Commodity Clusters

- MapReduce is designed to efficiently process large volumes of data by connecting many commodity computers together to work in parallel
- A *theoretical* 1000-CPU machine would cost a very large amount of money, far more than 1000 single-CPU or 250 quad-core machines
- MapReduce ties smaller and more reasonably priced machines together into a single cost-effective *commodity cluster*

146

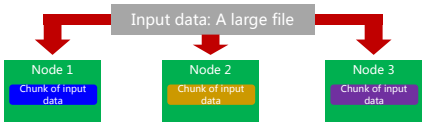
Isolated Tasks

- MapReduce divides the workload into multiple independent tasks and schedule them across cluster nodes
- A work performed by each task is done in isolation from one another
- The amount of communication which can be performed by tasks is mainly limited for scalability reasons
- The communication overhead required to keep the data on the nodes synchronized at all times would prevent the model from performing reliably and efficiently at large scale

147

Data Distribution

- In a MapReduce cluster, data is distributed to all the nodes of the cluster as it is being loaded in
- An underlying distributed file systems (e.g., GFS) splits large data files into chunks which are managed by different nodes in the cluster

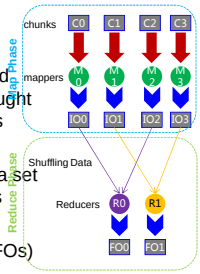


- Even though the file chunks are distributed across several machines, they form a single namesapce

148

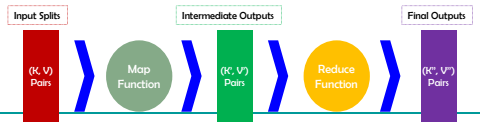
MapReduce: A Bird's-Eye View

- In MapReduce, chunks are processed in isolation by tasks called Mappers
- The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called Reducers
- The process of bringing together IOs into a set of Reducers is known as shuffling process
- The Reducers produce the final outputs (FOs)
- Overall, MapReduce breaks the data flow into two phases, map phase and reduce phase

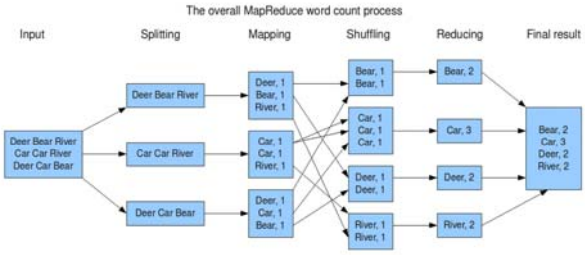


Keys and Values

- The programmer in MapReduce has to specify two functions, the map function and the reduce function that implement the Mapper and the Reducer in a MapReduce program
- In MapReduce data elements are always structured as key-value (i.e., (K, V)) pairs
- The map and reduce functions receive and emit (K, V) pairs



例：统计词频



function map(name, document)
for each word **in** document
emit (word, 1)

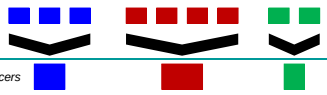
function reduce (word, List partialCounts)
sum = 0
for each pc **in** partialCounts
sum += pc
emit (word, sum)

151

Partitions

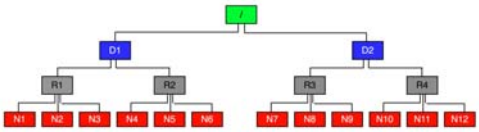
- In MapReduce, intermediate output values are not usually reduced together
- *All values with the same key are presented to a single Reducer together*
- More specifically, a different subset of intermediate key space is assigned to each Reducer : hash(k') mod R
- These subsets are known as *partitions*

Different colors represent different keys (potentially) from different Mappers



Partitions are the input to Reducers

MapReduce的网络拓扑



- MapReduce assumes a tree style network topology
- Nodes are spread over different racks embraced in one or many data centers
- A salient point is that the bandwidth between two nodes is dependent on their relative locations in the network topology
  - For example, nodes that are on the same rack will have higher bandwidth between them as opposed to nodes that are off-rack

MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

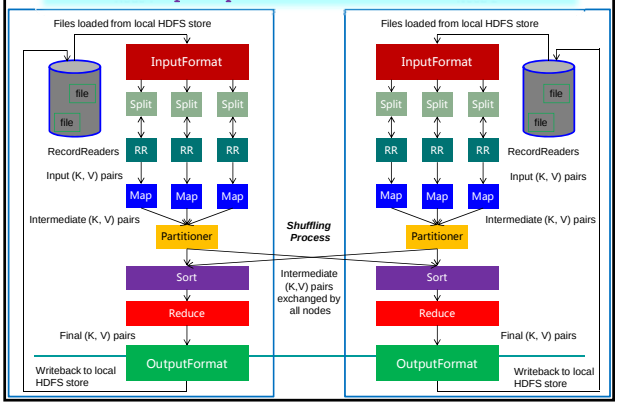
154

Hadoop

- Since its debut on the computing stage, MapReduce has frequently been associated with *Hadoop*
- Hadoop is an open source implementation of MapReduce and is currently enjoying wide popularity
- Hadoop presents MapReduce as an analytics engine and under the hood uses a distributed storage layer referred to as Hadoop Distributed File System (*HDFS*)
- HDFS mimics Google File System (*GFS*)

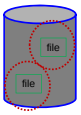
155

Hadoop MapReduce: A Closer Look



Input Files

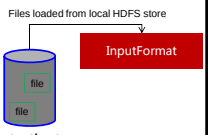
- *Input files* are where the data for a MapReduce task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
  - Line-based log files
  - Binary files
  - Multi-line input records
  - Or something else entirely



157

InputFormat

- How the input files are split up and read is defined by the *InputFormat*
- InputFormat is a class that does the following:
- Selects the files that should be used for input
- Defines the *InputSplits* that break a file
- Provides a factory for *RecordReader* objects that read the file



158

InputFormat Types

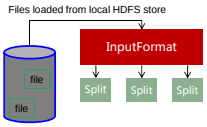
- Several InputFormats are provided with Hadoop:

InputFormat	Description	Key	Value
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into (K, V) pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

159

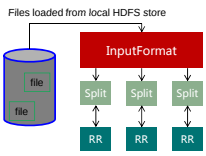
Input Splits

- An *input split* describes a unit of work that comprises a single map task in a MapReduce program
- By default, the InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel
- If the file is very large, this can improve performance significantly through parallelism
- Each map task corresponds to a *single* input split



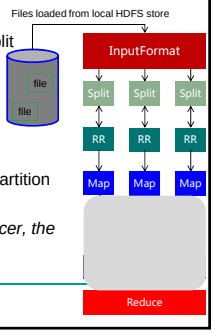
RecordReader

- The input split defines a slice of work but does not describe how to access it
- The *RecordReader* class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer



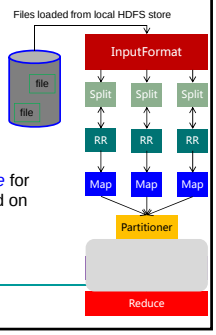
Mapper and Reducer

- The *Mapper* performs the user-defined work of the first phase of the MapReduce program
- A new instance of Mapper is created for each split
- The *Reducer* performs the user-defined work of the second phase of the MapReduce program
- A new instance of Reducer is created for each partition
- *For each key in the partition assigned to a Reducer, the Reducer is called once*



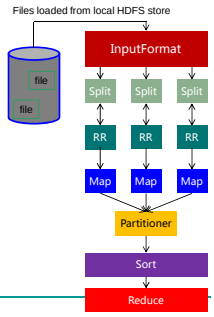
Partitioner

- Each mapper may emit (K, V) pairs to *any* partition
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data
- The *partitioner* class determines which partition a given (K,V) pair will go to
- The default partitioner computes *a hash value* for a given key and assigns it to a partition based on this result



Sort

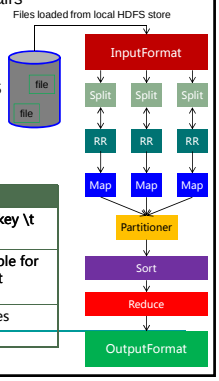
- Each Reducer is responsible for reducing the values associated with (several) intermediate keys
- The set of intermediate keys on a single node is *automatically sorted* by MapReduce before they are presented to the Reducer



OutputFormat

- The *OutputFormat* class defines the way (K,V) pairs produced by Reducers are written to output files
- The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS
- Several OutputFormats are provided by Hadoop:

OutputFormat	Description
TextOutputFormat	Default; writes lines in "key \t value" format
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Generates no output files



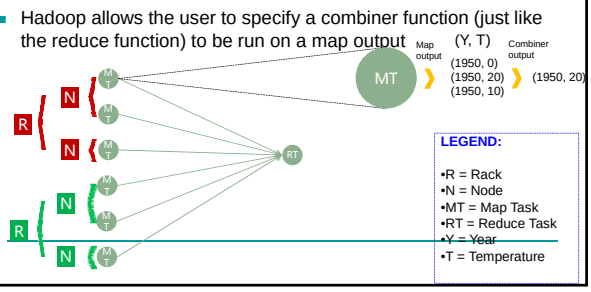
MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

166

Combiner Functions

- MapReduce applications are limited by the bandwidth available on the cluster
- It pays to minimize the data shuffled between map and reduce tasks
- Hadoop allows the user to specify a combiner function (just like the reduce function) to be run on a map output



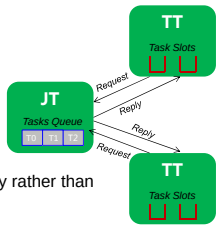
MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

168

Task Scheduling in MapReduce

- MapReduce adopts a *master-slave architecture*
- The master node in MapReduce is referred to as *Job Tracker* (JT)
- Each slave node in MapReduce is referred to as *Task Tracker* (TT)
- MapReduce adopts a *pull scheduling* strategy rather than a *push one*
  - I.e., JT does not push map and reduce tasks to TTs but rather TTs pull them by making pertaining requests



169

Map and Reduce Task Scheduling

- Every TT sends a *heartbeat message* periodically to JT encompassing a request for a map or a reduce task to run

I. Map Task Scheduling:

- JT satisfies requests for map tasks via attempting to schedule mappers in the *vicinity* of their input splits (i.e., it considers locality)

II. Reduce Task Scheduling:

- However, JT simply assigns the next yet-to-run reduce task to a requesting TT regardless of TT's network location and its implied effect on the reducer's shuffle time (i.e., it does not consider locality)

170

Job Scheduling in MapReduce

- In MapReduce, an application is represented as a *job*
- A job encompasses multiple map and reduce tasks
- MapReduce in Hadoop comes with a choice of schedulers:
  - The default is the *FIFO scheduler* which schedules jobs in order of submission
  - There is also a multi-user scheduler called the *Fair scheduler* which aims to give every user a fair share of the cluster capacity over time

171

Fault Tolerance in Hadoop

- MapReduce can guide jobs toward a successful completion even when jobs are run on a large cluster where probability of failures increases
- The primary way that MapReduce achieves fault tolerance is through *restarting tasks*
- If a TT fails to communicate with JT for a period of time (by default, 1 minute in Hadoop), JT will assume that TT in question has crashed
  - If the job is still in the map phase, JT asks another TT to re-execute *all Mappers that previously ran at the failed TT*
  - If the job is in the reduce phase, JT asks another TT to re-execute *all Reducers that were in progress on the failed TT*

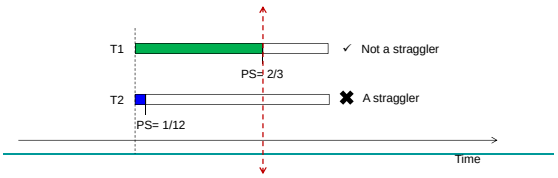
172

Speculative Execution

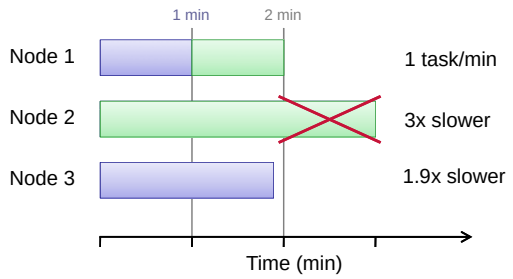
- A MapReduce job is dominated by the slowest task
- MapReduce attempts to locate slow tasks (*stragglers*) and run redundant (*speculative*) tasks that will optimistically commit before the corresponding stragglers
- This process is known as *speculative execution*
- Only one copy of a straggler is allowed to be speculated
- Whichever copy (among the two copies) of a task commits first, it becomes the definitive copy, and the other copy is killed by JT

Locating Stragglers

- How does Hadoop locate stragglers?
  - Hadoop monitors each task progress using a *progress score* between 0 and 1
  - If a task's progress score *is less than* (average – 0.2), and the task has run for at least 1 minute, it is marked as a straggler



Progress Rate Example



MapReduce

- In this part, the following concepts of MapReduce will be described:
  - Basics
  - A close look at MapReduce data flow
  - Additional functionality
  - Scheduling and fault-tolerance in MapReduce
  - Comparison with existing techniques and models

176

什么使得MapReduce如此独特？

- MapReduce is characterized by:
  1. Its simplified programming model which allows the user to quickly write and test distributed systems
  2. Its efficient and automatic distribution of data and workload across machines
  3. Its flat scalability curve. Specifically, after a Mapreduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes

177

与传统编程模型对比

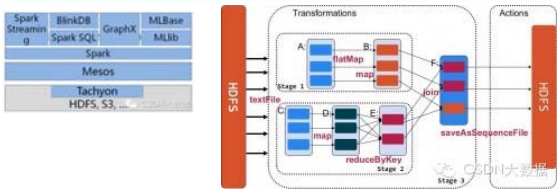
Aspect	Shared Memory	Message Passing	MapReduce
Communication	Implicit (via loads/stores)	Explicit Messages	Limited and Implicit
Synchronization	Explicit	Implicit (via messages)	Immutable (K, V) Pairs
Hardware Support	Typically Required	None	None
Development Effort	Lower	Higher	Lowest
Tuning Effort	Higher	Lower	Lowest

178

Google内部广泛使用的运算环境

- 用以太网交换机连接、由普通PC机组成的大型集群
  - x86架构、运行Linux操作系统、双处理器、2-4GB内存
  - 普通的网络硬件设备，每个机器的带宽为百兆或者千兆，但是远小于网络的平均带宽的一半
  - 集群中包含成百上千的机器，因此，机器故障是常态
  - 存储为廉价的内置IDE硬盘。一个内部分布式文件系统用来管理存储在这些磁盘上的数据。文件系统通过数据复制来在不可靠的硬件上保证数据的可靠性和有效性
  - 用户提交作业给调度系统。每个作业都包含一系列的任务，调度系统将这些任务调度到集群中多台可用的机器上执行

内存计算架构Spark



RDD是Spark的最基本抽象这对于迭代运算比较常见的机器学习算法,交互式数据挖掘来说，效率提升比较大转换（Transformation）和操作（Action）这两大类

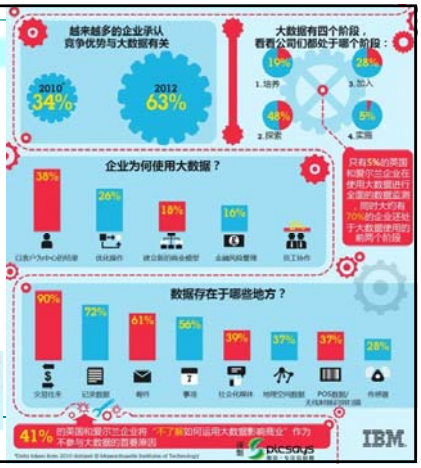
Spark常见存储数据的格式是Key-Value，也就是Hadoop标准的Sequence File，但同时也听说支持类似Parquet这样的列存格式。Key-Value格式的优点在于灵活，上至数据挖掘算法，明细数据查询，下至复杂SQL处理都能承载，缺点也很明显就是存储空间比较浪费，和类似Parquet列存格式相比更是如此，key-Value格式数据一般是原始数据大小的2倍左右，而列存一般是原始数据的1/3到1/4。

实时的流计算平台Storm

报告内容

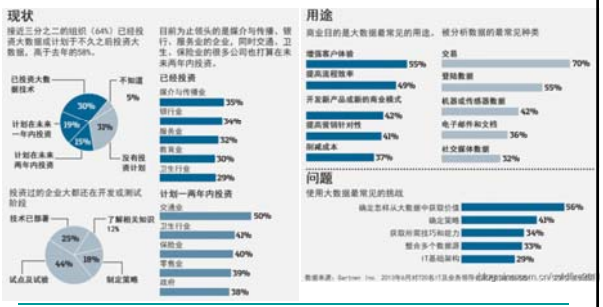
- 大数据有关概念
- 大数据应用现状
- 并行计算技术简述
- 大数据油田应用探讨

- 一份调研报告显示在大数据面前企业的处境



这些结论来自2012年IBM和牛津大学萨伊德商学院一次名为“现实世界中的大数据使用”的研究中关于英国和爱尔兰的结果。研究公司如何使用大数据，从信息的来源到参与的程度，以更好的了解如今的市场。

大多数企业都还在摸索大数据分析方法



大数据在油田应用展望

- 数据收集
- 精准运维
- 油田设备的全寿命管理
- 区位边界管理
- 精准营销
- 物流配送
- 危险品运输车辆监控
- 油气行业的安全监控

大数据在油田应用展望

- 灾害预测
- 灾害检测报警
- 灾害应急决策
- 美国BP公司根据管道承压得知那种原油更有腐蚀性
- 腐蚀检测，腐蚀回路检测

一些资料分析

- 关于如何面对大数据的展望性资料

传统石油公司必须转型成为新型的创新型公司

- 一方面，随着传统化石能源面临枯竭，而且开采成本持续上扬。另一方面，随着时代对环境质量的苛刻，新能源逐渐成为未来选择，而且其成本也在持续下降。
- 随着石油储备的逐步减少，石油石化行业产业链中的勘探、开发难度日益增大，信息化的成熟度已经成为影响行业增长幅度的首要因素。
- 通过对资本、技术、市场、组织进行前瞻性调整，以此谋求成为“在市场竞争中具有优势和持续发展能力的企业”。

卜娜，石油勘探行业怎么用大数据“找油”，中国计算机报，20140815
王晓夏，大数据时代：中国油公司你准备好了吗？能源杂志，2014年8期

一种观点

- 以前较多依赖于企业领袖的洞察力
- 大数据思维启示我们将洞察力更多地构建在对数据的挖掘之中，通过数字来感知企业自身的健康程度和外界市场的实时变化。
- 这一切，要求石油公司必须完成自身的信息化建设。只有信息化才能具备前瞻性的洞察力和灵活的转型能力。

石油行业是信息技术典型应用领域

- 传统上，全球范围内石油产业一直是大规模应用计算机的产业之一，信息技术对石油行业发展起到过极大的推动作用。
  - HPC典型应用于勘探分析、地质资料解释等方面。
- 传统石油公司往往具有全球化、多组织化、流程复杂化等特点，采用ERP系统完成对物流、人流、财流、信息流的集成和管理是必然的选择
  - 遗憾的是，这使得ERP不堪重负，导致企业面对“不上ERP是等死，上ERP是找死”的怪圈。

信息化的挑战

- 即便是诸如埃克森美孚和雪佛龙这样的业界巨头，相比其他行业企业的信息化程度也不能完全称自身已经完成了ERP信息化转型。
- 以埃克森美孚为例，曾在此前一次全球性招标中，一次性投入10亿美金来采购信息化服务。此外，很多巨头每年在信息化建设中投入的比例往往占到公司盈利比例的1%-3%不等。
- 为了了解和模拟出地下数千米的地质构造，通过地震波反射方式来收集海量数据，一般二维数据可达1~2TB，三维数据可高达几百TB甚至PB级，然后进行大量的密集计算和模拟，计算结果出来后还要转换成直观的可视画面，方便专家对数据进行解释，为油气钻井定位提供参考。

大数据应用技术能带来什么

- 逐渐成为业内人士的寄托（智慧能源、智慧油田、智慧管道）
- 资料分析表明关注点仍然是HPC解决石油勘探中的计算问题
- 大数据应用技术为油田发展带来新思维
  - 精准运维（勘探、生产、储运、提炼）
  - 精准营销（已有较多的应用案例借鉴）
  - 意外、损失、事故、灾难预防和应急（十分必要）

一种观点

- 智慧油气田建设将为油气工业带来巨大价值：
- (1)改进油气藏状态监测和数据采集；
- (2)改进地下、地面和企业数据集成和管理；
- (3)改进关键性事件管理和快速响应；
- (4)改进油气勘探与生产状态分析和预测；
- (5)全面优化油气勘探与生产工作流程。

石油石化行业具有数据量大、类型多样、存储格式复杂及数据分散等特点，一些企业经验报道

- 大数据积累是化工生产稳定运行的保障，关注与气化炉运行相关的每一项数据分析是装置长周期运行的重要因素。
- 今年7月19日，“大数据技术在催化裂化装置中的应用”项目在九江石化启动。这是2014年度中国石化股份公司科技开发项目，通过应用大数据技术这一当今数据分析的前沿技术，对催化裂化装置的海量历史数据进行深层分析挖掘，有望快速获取有价值信息，形成可供推广的生产操作指导方案和风险评估技术。
- 金陵石化的技术和操作培训中，突出了巧用装置的大数据寻求优化措施的内容，促进了技术和操作两个层面的生产优化，提升了装置经济效益。
- 内蒙古通辽市龙源绿美化工有限公司气化项目负责人郑万德向记者表示，应用大数据，首先要保证装置稳定运行，其次是降低能耗，比如通过压力、温度、流量等一些参数的提前优化，防止压缩机喘振、压缩机超压等现象产生，对压缩机从起动、到各个负荷之间的参数进行收集，利用数据综合利用系统深度分析后，调整电耗、油耗。

好处

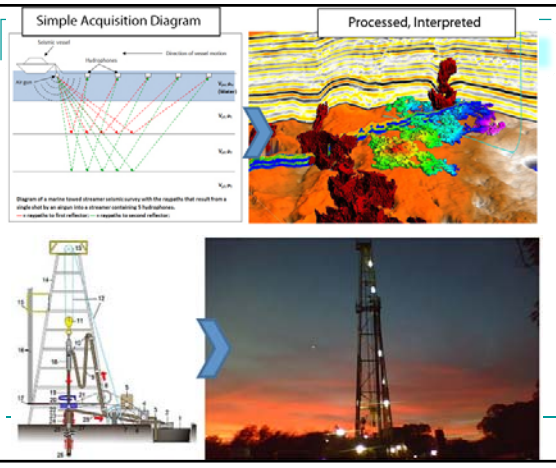
- 提高收益
- 增强安全性
- 提高计算资源利用率
- 更有环境友好性
- 信息资源开放、共享
- 推进行业标准化
- 完善法律法规

资料分析

- 在油气勘探、开发和生产的过程和决策中产生大量数据并每日增长，随着采集、处理和存储解决方案进展，数据量两年翻一番，用途：
  - 对地质结构和地层建模和可视化
  - 描述数量巨大的各井有关行为，如机械性能、油流速、压力（US有约1百万口井）
- 油气企业信息化过程
  - 90年代数据集成
  - 00年代软件集成
  - 近十年：使用全部数据达到更多油气并且风险和环境影响得到减低

Adam Farris.How big data is changing the oil & gas industry. Analytics, Nov./Dec. 2012

- 技术复杂、高风险挑战
  - 相对于天文数字的收入利润率8～9%，面对深地层、海洋和地缘政治区域，降低风险和环境影响很困难
  - 油难找、生产成本高、必须解决钻油潜在的环境和人身安全问题
- 结论
  - 掌握大数据和面对挑战，需要在二者之间架起桥梁，Shell和 Chevron利用风险投资创建一种途径去探索新构想
  - 异花授粉
  - 让现有系统包括在大数据分析中



BP的经验

- First, the benefits. At BP, we have seen significant performance enhancements via real-time drilling technologies. These include BP Well Advisor, a suite of technologies that allow us to monitor well construction in real-time. Our Field of the Future® information system enables continuous remote monitoring of well performance, facility and operational integrity. It is now being designed into all our major projects from the start.

Potential applications in O&G

- Equipment maintenance: using data collected from pumps and wells to adjust repair schedules and prevent or anticipate failure.
- Production optimization using powerful modeling capabilities to anticipate costs and production volumes.
- Price optimization: using scalable compute technologies to determine optimum commodity pricing.
- Safety and compliance: using weather or workforce scheduling data to avoid creating dangerous conditions for workers and mitigating environmental risks.

油气田常用的腐蚀检测技术

- 挂片法：优点可确定腐蚀类型；缺点测量周期长且不能反映腐蚀过程
- 在线电化学探针检测技术：优点可反映腐蚀过程且不必停工适应各种不同介质；缺点？
- 在线电阻探针技术：
- 在线电感探针技术：优点是响应快127um为0.8小时而电阻探针需要70小时，精度可达30nm

英国石油公司在美国的一个炼油厂里，安装了很多无线感应器，通过随时监测管道承压，厂方发现某些原油更具腐蚀性，就可以掌握和采取防止措施。

结论

- 大数据时代是历史性变革时代
- 大数据产业为产业升级和发展带来机遇
- 并行计算技术是大数据支撑技术之一
- 大数据在油田应用才起步，机遇与挑战需要行业内外共同研究

Thanks!