
上海超级计算中心
Shanghai Supercomputer Center
总第26期
1
2009年
高性能计算
发展与应用
Development & Application
of High Performance Computing

目 录

云计算及其关键技术

● 邓倩妮 上海交通大学计算机系 上海 200040
● 陈 全 上海交通大学计算机系 上海 200040

摘要：

论文对新兴的计算模型——云计算进行了简要的介绍。论文给出了云计算的定义，介绍了云计算的发展背景和应用场景，分析了云计算和网格计算以及传统超级计算的区别，总结了云计算的关键技术：存储技术、数据管理技术以及编程模型。

关键词：云计算; 数据存储; 数据管理; 编程模型

1. 云计算产生背景及定义

1.1 云计算的定义

云计算(Cloud Computing)是一种新近提出的计算模式。维基百科给云计算下的定义：云计算将IT相关的能力以服务的方式提供给用户，允许用户在不了解提供服务的技术、没有相关知识以及设备操作能力的情况下，通过Internet获取需要服务[1]。

中国云计算网将云定义为：云计算是分布式计算（Distributed Computing）、并行计算（Parallel Computing）和网格计算（Grid Computing）的发展，或者说是这些科学概念的商业实现[2]。

Forrester Research 的分析师 James Staten 定义云为:“云计算是一个具备高度扩展性和管理性并能够胜任终端用户应用软件计算基础架构的系统池”。

虽然目前云计算没有统一的定义，结合上述定义，可以总结出云计算的一些本质特征，即分布式计算和存储特性，高扩展性，用户友好性，良好的管理性。云计算技术具有以下特点：

(1) 云计算系统提供的是服务。服务的实现机制对用户透明，用户无需了解云计算的具体机制，就可以获得需要的服务。

(2) 用冗余方式提供可靠性。云计算系统由大量商用计算机组成机群向用户提供数据处理服务。随着计算机数量的增加，系统出现错误的概率大大增加。在没有专用的硬件可靠性部件的支持下，采用软件的方式，即数据冗余和分布式存储来保证数据的可靠性。

(3) 高可用性。通过集成海量存储和高性能的计算能力，云能提供一定满意度的服务质量。云计算系统可以自动检测失效节点，并将失效节点排除，不影响系统的正常运行。

(4) 高层次的编程模型。云计算系统提供高级别的编程模型。用户通过简单学习，就可以编写自己的云计算程序，在“云”系统上执行，满足自己的需求。现在云计算系统主要采用Map-Reduce模型。

(5) 经济性。组建一个采用大量的商业机组成的机群相对于同样性能的超级计算机花费的资金要少很多。

1.2 云计算的应用场景

云计算有着广泛的应用前景。如表1所示：

表1 云计算的应用领域

领域	应用场景
科研	地震监测
	海洋信息监控
	天文信息计算处理
医学	DNA信息分析
	海量病历存储分析
	医疗影像处理
网络安全	病毒库存储
	垃圾邮件屏蔽
图形和图像处理	动画素材存储分析
	高仿真动画制作
	海量图片检索
互联网	Email服务
	在线实时翻译
	网络检索服务

云计算在天文学[7]、医学等各个领域有着广泛的应用前景。

趋势科技和瑞星等安全厂商纷纷提出了“安全云”计划。如今，每天有2万多种新的病毒和木马产生，传统的通过更新用户病毒库的防毒模式，受到了严峻的挑战，用户端的病毒库将过于庞大。趋势科技和瑞星的“安全云”将病毒资料库放在“云”端，与客户端通过网络相连，当“云”在网络上发现不安全链接时，可以直接形成判断，阻止其进入用户机器，从根本上保护机器的安全。

据趋势科技大中华区执行总裁张伟钦介绍，趋势科技已投入了大量资金，在全球数个地方建设了新型数据中心。同时，趋势科技还花费了1000多万美元，租借了34000多台服务器，构建了一个服务遍及全球的“安全云”。目前趋势科技已将公司中低端的部分产品线放到“云安全”计划中，而高端的大部分产品线，仍在准备过程中。

谷歌提供的Gmail、Google Earth、Google Analytics等服务都基于其云计算服务器运行[8]。谷歌基于云计算提供的翻译服务具有现今最好的性能[9]。对互联网和美国人生活的一项研究显示，大约70%的在线用户使用以上“云计算”服务。

1.3 云计算的发展

目前，亚马逊，微软，谷歌，IBM，Intel等公司纷纷提出了“云计划”。例如亚马逊的AWS（Amazon Web Services)[3]、IBM和谷歌联合进行的“蓝云”计划等。这对云计算的商业价值给予了巨大的肯定。同时学术界也纷纷对云计算进行深层次的研究。例如谷歌同华盛顿大学以及清华大学合作，启动云计算学术合作计划(Academic Cloud Computing Initiative)，推动云计算的普及，加紧对云计算的研究。美国卡耐基梅陇大学等提出对数据密集型的超级计算（DISC：Data Intensive SuperComputing）进行研究，本质上也是对云计算相关技术开展研究。

IDC的调查显示，未来五年云计算服务将急速增长，预期2012年市场规模可达420亿美元。目前企业导入云计算已逐渐普及，并且有逐年成长趋势。估计在2012年，企业投入在云计算服务的支出将占整体IT成本的25%，甚至在2013年提高至IT总支出的三分之一。

由此可见，在各大公司以及学术界的共同推动下，云计算技术将会持续发展。

1.4 云计算与其他超级计算的区别

1.4.1 云计算与网格计算的区别

Ian Foster 将网格定义为：支持在动态变化的分布式虚拟组织（Virtual Organizations）间共享资源，协同解决问题的系统[4]。所谓虚拟组织就是一些个人、组织或资源的动态组合。

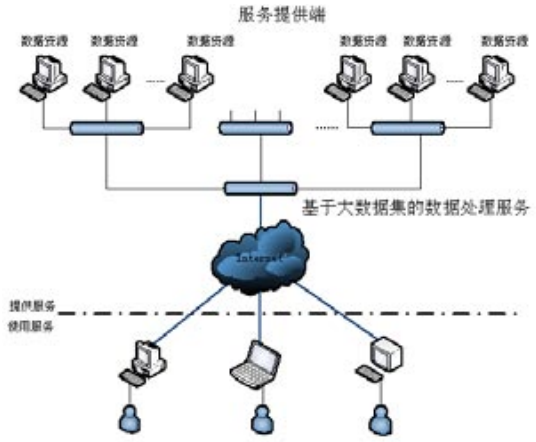


图1“云”系统的结构

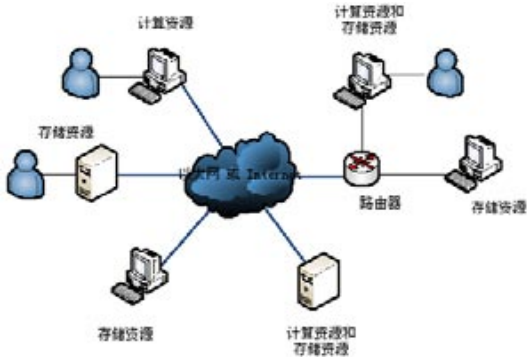


图2 网格的结构

图1和图2分别为云及网格的结构示意图。图1 显示，云计算是一种生产者—消费者模型，云计算系统采用以太网等快速网络将若干机群连接在一起，用户通过因特网获取云计算系统提供的各种数据处理服务。图2显示，网格系统是一种资源共享模型，资源提供者亦可以成为资源消费者，网格侧重研究的是如何将分散的资源组合成动态虚拟组织。

云计算和网格计算的一个重要区别在于资源调度模式。云计算采用机群来存储和管理数据资源，运行的任务以数据为中心。即调度计算任务到数据存储节点运行。而网格计算，则以计算为中心。计算资源和存储资源分布在因特网的各个角落，不强调任务所需的计算和存储资源同处一地。由于网络带宽的限制，网格计算中的数据传输时间占总运行时间的很大一部分。

1.4.2 云计算系统与传统超级计算机的区别

超级计算机拥有强大的处理能力，特别是计算能力。2008年11月17日，最新一期的Top500[6]榜单发布。冠军“RoadRunner”是IBM为美国 Los Alamos 国家实验室建造的计算机系统。它的运算速度达到了

1.026 Petaflop/s。RoadRuner超级计算机包含12960个IBM PowerXcell 8i处理器以及6948个分布于刀片服务器上的AMD Opteron芯片刀片服务器安装在288个IBM BladCener机架上。RoadRuner拥有80TB的内存，外存使用1.5PB容量的Panasas存储，外存通过10Gb/秒以太网进行连接。耗资超过1亿美元。

TOP500对超级计算机的排名方式可以看出，传统的超级计算机注重运算速度和任务的吞吐率。以运算速度为核心进行计算机的研究和开发。而云计算则以数据为中心，同时兼顾系统的运算速度。传统的超级计算机耗资巨大，远超云计算系统。例如，趋势科技花费1000多万美元租用34000多台服务器，构建自身的“安全云”系统。

1.5 云计算的关键技术

云计算是一种新型的超级计算方式，以数据为中心，是一种数据密集型的超级计算[10]。在数据存储、数据管理、编程模式等方面具有自身独特的技术。

1.5.1 数据存储技术

为保证高可用、高可靠和经济性，云计算采用分布式存储的方式来存储数据，采用冗余存储的方式来保证存储数据的可靠性，即为同一份数据存储多个副本。

另外，云计算系统需要同时满足大量用户的需求，并行地为大量用户提供服务。因此，云计算的数据存储技术必须具有高吞吐率和高传输率的特点。

云计算的数据存储技术主要有谷歌的非开源的GFS（Google File System）[11]和 Hadoop 开发团队开发的GFS的开源实现HDFS（Hadoop Distributed File System）[12][13]。大部分IT厂商，包括yahoo、Intel的“云”计划采用的都是HDFS的数据存储技术。

未来的发展将集中在超大规模的数据存储、数据加密和安全性保证、以及继续提高I/O速率等方面。

1.5.2 数据管理技术

云计算系统对大数据集进行处理、分析向用户提供高效的服务。因此，数据管理技术必须能够高效的管理大数据集。其次，如何在规模巨大的数据中找到特定的数据，也是云计算数据管理技术所必须解决的问题。

云计算的特点是对海量的数据存储、读取后进行大量的分析，数据的读操作频率远大于数据的更新频率，云中的数据管理是一种读优化的数据管理。因此，云系统的数据管理往往采用数据库领域中列存储的数据管理模式。将表按列划分后存储。

云计算的数据管理技术最著名的是谷歌的BigTable[14]数据管理技术，同时Hadoop开发团队正在开发类似BigTable的开源数据管理模块。

由于采用列存储的方式管理数据，如何提高数据的更新速率以及进一步提高随机读速率是未来的数据管理技术必须解决的问题。

1.5.3 编程模式

为了使用户能更轻松的享受云计算带来的服务，让用户能利用该编程模型编写简单的程序来实现特定的目的，云计算上的编程模型必须十分简单。必须保证后台复杂的并行执行和任务调度向用户和编程人员透明。

云计算采用类似MAP-Reduce[15]的编程模式。现在所有IT厂商提出的“云”计划中采用的编程模型，都是基于MAP-Reduce的思想开发的编程工具。

MAP-Reduce不仅仅是一种编程模型，同时也是一种高效的任务调度模型。Map-Reduce这种编程模型并不仅适用于云计算，在多核和多处理器、cell processor、以及异构机群上同样有良好的性能[16, 17, 18]。

该编程模式仅适用于编写任务内部松耦合、能够高度并行化的程序。如何改进该编程模式，使程序员得能够轻松的编写紧耦合的程序，运行时能高效的调度和执行任务，是Map-Reduce编程模型未来的发展方向。

2. 数据存储技术

为了满足云计算的分布式存储方式、同时保证数据可靠性和高吞吐率以及高传输率的需求。目前各IT厂商多采用GFS或HDFS的数据存储技术。

以GFS为例。GFS是一个管理大型分布式数据密集型计算的可扩展的分布式文件系统。它使用廉价的商用硬件搭建系统并向大量用户提供容错的高性能的服务。

GFS和普通的分布式文件系统有以下区别，如表2所示：

表2 GFS与传统分布式文件系统的区别

	GFS	传统分布式文件系统
组件失败管理	不作为Exception处理	作为Exception处理
文件大小	少量大文件	大量小文件
数据写方式	在文件末尾附加数据	修改现存数据
数据流和控制流	数据流和控制流分开	数据流和控制流结合

GFS系统由一个Master和大量块服务器构成。Master存放文件系统的所有的元数据，包括名字空

间、存取控制、文件分块信息、文件块的位置信息等。GFS中的文件切分为64MB的块进行存储。

在GFS文件系统中，采用冗余存储的方式来保证数据的可靠性。每份数据在系统中保存3个以上的备份。为了保证数据的一致性，对于数据的所有修改需要在所有的备份上进行，并用版本号的方式来确保所有备份处于一致的状态。

客户端不通过Master读取数据，避免了大量读操作使Master成为系统瓶颈。客户端从Master获取目标数据块的位置信息后，直接和块服务器交互进行读操作。

GFS的写操作将写操作控制信号和数据流分开，如图3[11]所示：

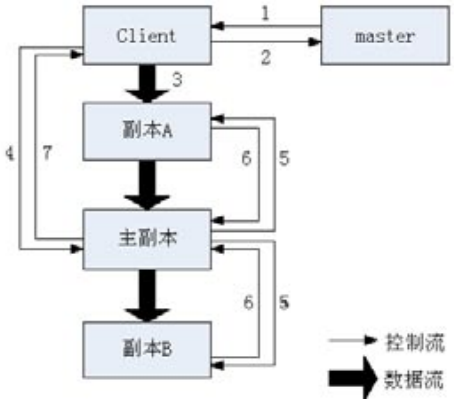


图3 写控制信号和写数据流

即，客户端在获取Master的写授权后，将数据传输给所有的数据副本，在所有的数据副本都收到修改的数据后，客户端才发出写请求控制信号。在所有的数据副本更新完数据后，由主副本向客户端发出写操作完成控制信号。具体请见[11]。

3. 数据管理技术

为了满足云计算的大规模数据集管理，高效的数据定位需求。谷歌采用BigTable的数据管理技术。在各大IT厂商的支持下，Hadoop开发团队正在开发其开源版本。

以BigTable为例。BigTable数据管理方式设计者——Google给出了如下定义：“ BigTable是一种为了管理结构化数据而设计的分布式存储系统，这些数据可以扩展到非常大的规模，例如在数千台商用服务器上的达到PB(Petabytes)规模的数据。 ”

BigTable对数据读操作进行优化，采用列存储的方式，提高数据读取效率。BigTable管理的数据的存储结构为：<row: string, column: string, time: int64> ->string。BigTable的基本元素是：行，列，记录板和时间戳。其中，记录板是一段行的集合体。

BigTable中的数据项按照行关键字的字典序排列，每行动态地划分到记录板中。每个节点管理大约100个记录板。时间戳是一个64位的整数，表示数据的不同版本。

BigTable在执行时需要三个主要的组件： 链接到每个客户端的库，一个主服务器，多个记录板服务器。主服务器用于分配记录板到记录板服务器以及负载平衡，垃圾回收等。记录板服务器用于直接管理一组记录板，处理读写请求等。

为保证数据结构的高可扩展性，BigTable采用三级的层次化的方式来存储位置信息，如图4[14]所示。

其中第一级的Chubby file中包含Root Tablet的位置，Root Tablet包含所有METADATA tablets的位置信息，每个METADATA tablets包含许多User Table的位置信息。具体见[14]。

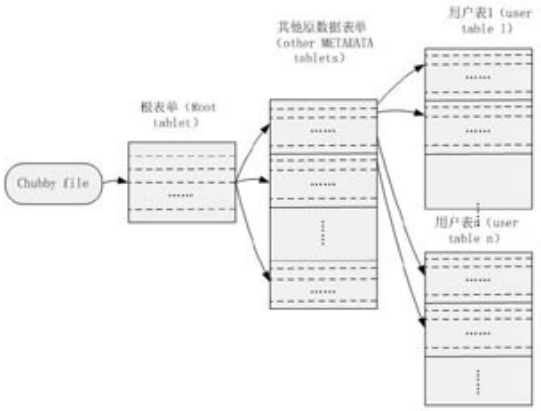


图4 BigTable中存储记录板位置信息的结构

4. 编程模型技术

当前各IT厂商提出的“ 云 ”计划的编程工具均基于Map-Reduce的编程模型。Map-Reduce是一种处理和产生大规模数据集的编程模型，程序员在Map函数中指定对各分块数据的处理过程，在Reduce函数中指定如何对分块数据处理的中间结果进行归约。用户只需要指定map和reduce函数来编写分布式的并行程序。当在机群上运行Map-Reduce程序时，程序员不需要关心如何将输入的数据分块、分配和调度，同时系统还将处理机群内节点失败以及节点见通信的管理等。图5给出了一个Map-Reduce程序的具体执行过程。

从图5可以看出，执行一个Map-Reduce程序需要五个步骤：输入文件、将文件分配给多个worker并行地执行、写中间文件（本地写）、多个Reduce workers同时运行、输出最终结果。本地写中间文件在减少了对网络带宽的压力同时减少了写中间文件的时间耗费。执行Reduce时，根据从Master获得的中间文件位置信息，将Reduce命令发送给中间文件所在节点执行，进一步减少了传送中间文件对带宽的需求。

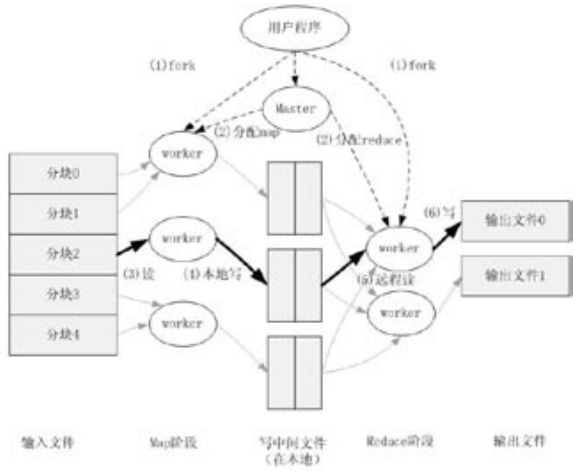


图5 Map-Reduce程序的具体执行过程

Map-Reduce模型具有很强的容错性，当worker节点出现错误时，只需要将该worker节点屏蔽在系统外等待修复，并将该worker上执行的程序迁移到其他worker上重新执行同时将该迁移信息通过Master发送给需要该节点处理结果的节点。Map-Reduce使用检查点的方式来处理Master出错失败的问题，当Master出现错误时，可以根据最近的一个检查点重新选择一个节点作为Master并由此检查点位置继续运行。

5. 结语

综上所述，云计算是一种新型的计算模式。它的最主要特征是系统拥有大规模数据集、基于该数据集，向用户提供服务。它使用大量的普通商用机来构建系统，通过冗余存储的方式确保整个系统的可靠性和可用性。与传统超级计算机在底层编程不同，数据密集计算的云系统上使用基于Map-Reduce的高级编程模式。这使得编程人员可以不用考虑底层的并行化方式，专心与程序的逻辑实现。普通用户经过简单的学习，可以编写出满足自身需要的简单程序。

越来越多的IT厂商提出了自己的“云”计划，并投入大量资金推动云计算的发展。这恰恰为云计算提供了良好的发展机遇。虽然现在的云计算并不能完美地解决所有的问题，但是在不久的将来，一定会有越来越多的云计算系统投入实用，云计算系统也会不断地被完善，并推动其他科学技术的发展。

参考文献：

[1] 维基百科http://en.wikipedia.org/wiki/Cloud_computing

[2] 中国云计算网。 http://www.cloudcomputing-china.cn/Article/ShowArticle.asp?ArticleID=1

[3] Jinesh Varia. Cloud architectures- Amazon web services [EB/OL]. ACM Monthly Tech Talk , http://acmbangalore.org/events/monthly-talk/may-2008--cloud-architectures---amazon-web-services.html, May, 2008

[4] IAN FOSTER; CARL KESSELMAN; STEVEN TUECKE. The anatomy of the grid enabling scalable virtual organizations. International Journal of High Performance Computing Applications. August 2001, 15(3): 200-222

[5] FRAN Berman, GEOFFREY Fox, TONY Hey. The grid: past, present, and future [A]. Grid Computing: Making the Global Infrastructure a Reality [C]. John Wiley & Sons, Ltd, 2003. 9-50.

[6] Top 500 supercomputing sites. http://www.top500.org/

[7] ALEXANDER S. Szalay, PETER Kunszt, ANI Thakar, JIM Gray, DON Slutz, ROBERT J. Brunner. Designing and mining multi-terabyte astronomy archives: The Sloan Digital Sky Survey [A]. SIGMOD International Conference on Management of Data Proceedings of the 2000 ACM SIGMOD international conference on Management of data. ACM, 2000. 29(2): 451-462

[8] Luiz Andr é Barroso, Jeffrey Dean, Urs H-lzle. Web search for a planet: The Google cluster architecture [J]. IEEE Micro, Mar/Apr, 2003, 23(2): 22 - 28.

[9] Google tops translation ranking[N]. News@Nature, http://www.nature.com/news/2006/061106/full/news061106-6.html, Nov. 6, 2006.

[10] RANDAL E.Bryant. Data-Intensive supercomputing: the case for DISC[R]. CMU Technical Report CMU-CS-07-128. May 10, 2007.

[11] SANJAY GHEMAWAT; HOWARD GOBIOFF; PSHUN-TAK LEUNG. The Google file system. Proceedings of the nineteenth ACM symposium on Operating systems principles. Oct. 2003

[12] Hadoop. http://hadoop.apache.org/

[13] Yahoo! Hadoop Tutorial. http://public.yahoo.com/gogate/hadoop-tutorial/start-tutorial.html

[14] Fay Chang, Jeffrey Dean, Sanjay Ghemawat et al. BigTable: a distributed storage system for structured data [A]. Operating Systems Design and Implementation, 2006.

未来需要云计算

● 姚继锋 上海超级计算中心 上海 201203 jfyao@ssc.net.cn

改变未来的云计算

云计算无疑是过去一年中最热门的词汇之一。在计算机技术的发展史上，除了云计算，大概还没有第二个因为一个响亮的名字而迅速引起广泛关注的技术。云计算到底是什么？为什么需要云计算？是商业炒作还是未来愿景？是现有技术的简单组合还是充满挑战和未知？…… 类似的讨论早已在媒体热火朝天，但莫衷一是。

所谓“云”，是指在各种技术架构图中常用一个云团来表示的互联网；所谓云计算，即是基于互联网的计算。由此可见，云计算并不是一个新事物（这也是它被一些人诟病为广告宣传语的原因），24年前 SUN公司就提出了“The Network is the Computer”，并作为企业战略奋斗至今，相较于云计算，这句话更有力量。

云计算不是一个技术名词，很难给出一个确切的定义，它作为一种新的计算形态，直接对应的是传统的桌面计算，即随着PC的发展和普及在过去20多年间人们使用计算机最主要的方式：每个人拥有自己的硬件、软件，本地保存数据和进行处理。互联网只是让人们能更方便的去获得信息，但计算和处理主要还是基于本地的PC进行。但如果云计算仅仅是指通过互联网利用远端的计算能力进行处理，那么现有的提供一些特定计算功能的网站是不是云计算，例如提供公历/农历的转换或者简单的图像处理？答案自然是否定的。云计算的特质是面向海量的数据和复杂的计算，这是被很多人忽略的一点。

任何成功的新事物，它无非是提供两个功能：将原来坏的事情，变得不那么坏（做减法）；或者将原来好的事情，变得更好（做加法）。也可以换一个角度，是来缓解或者解决当前面临的问题或困境；或者启迪、开发新的功能、需求或价值。云计算一个明显的优势是可以降低应用计算的成本。利用云计算，用户可以避免本地建设、运维不菲的计算系统，通过支付低廉的服务费用，即可完成同样的计算或处理过程。类似这样的优势，是云计算的缓解困境之道，但对云计算而言，主要的意义应该在后者，即它的出现和存在，是为了触发、满足一些以前未有的需求。

计算机的出现是为了满足人们对获取信息、处理信息的需求。纵观数十年计算机技术的发展，有着一条清晰的主线：获得性能更好、处理能力更强的计算机（这是做加法。另外一条主线是获得更方便、更好用、更安全、更低廉的计算机，这是做减法）。搜索引擎、音频视频、3D动画和游戏、手机、电子导航……。所有这些在十年、二十年前你难以想象的新事物某种程度上都得归功于计算能力的不断提高，并已经成为日常生活中不可或缺的一部分。展望五年、十年或者二十年后，还会有哪些新事物出现？不是科幻作家，这个问题或许有些困难，但无疑人们能获得的信息会更多、更好、更便捷。那么导致这些未来新事物出现的推力会有哪些？云计算毫无疑问应该是答案之一。

在未来，云计算存在的形态将会是一个个如同Google、Amazon这样的运营和服务中心，可以简单的将它视为数据中心+计算中心+界面/接口。通过界面或者接口，普通用户将可以利用以往只能为少数人所拥有的庞大的数据和处理能力，获得自己所需的信息。云计算对未来最大的意义在于：如果你现在拥有前所未有的数据和计算能力，你能创造什么？

Google推动云计算的一个举措是让高校学生利用现有的API进行编程。如果你是其中之一，是一名未来的程序员，想象一下，编写一小段代码，后台运作的是成千上万台的服务器，徜徉的是浩瀚的数据海洋，这是什么样的感觉？同样，如果你是一名科研工作者，平常只是利用桌面的PC进行模拟演算，而现在给你的是世界上最快的、性能是单台PC数万倍的超级计算机，那又会是什么感觉？

云计算不会很大的改变你现在已有的大部分计算，它的存在主要不是为了替代，而是为了创造。现在使用PC版的Word来编写文档，在线的文档编辑工具（如Google Docs）不会改变这一现状，如果有云计算版本的Word，那它不仅仅是有多人协同这样的简单功能，而可能是当你敲下一行标题或者输入一列关键字，在页面上会弹出成百上千个你可以作为

参考和范本的文档，而这些文档，是从数十亿篇已有的文档中为你精心挑选和准备的。甚至系统会通过某种复杂的算法自动的为你生成一篇文章。这才是云计算可能会做的。

理论、实验和计算，这是人类进行创新的三条途径。云计算会使庞大的计算力为更多的人群所利用，它必将很大的加速技术改变人类生活的进程。创造者可以是科学家、工程师、或者程序员，也可能是任何一个有奇思妙想的普通人，只要他有一个终端，有一根网线，能方便的去操控数据、处理数据。

谁来进行云计算

Google、百度、新浪、腾讯、盛大等众多已经有着丰富数据资源或计算资源的互联网企业将会走在云计算浪潮的前列，除了安全性、带宽、软硬件资源管理等技术因素外，他们面临的最大挑战是尽快寻找到或者创造出新的基于云计算的用户需求。

传统的数据中心和超级计算中心，因其资源的优势，将很有可能走在云计算浪潮的前列。事实上，传统的超级计算中心已经完全符合云计算的特征和描述。例如笔者所在的上海超级计算中心，作为国内首家也是唯一一家面向公众开发的公共计算服务平台，已经通过网络为各个应用领域的用户提供计算服务。对这类数据或计算中心，云计算时代面临的主要挑战同样是拓宽或者寻找新的服务领域和服务内容。

对于个人或者中小型企业，云计算通常充满着机遇。他们可以通过和云计算服务中心的合作，托管运行自己的服务。一方面可以将现有的单机难以运行的应用移植到云计算中心，以服务的方式为用户进行数据处理或者计算；另外一方面，同样可以利用云计算服务中心开放的API，自行开发应用为用户提供服务，Google Map API即是这样的例子。个人或者中小型企业与大型的云计算服务中心的紧密合作，这将很可能是未来云计算的主要形态，也是创新的主要动力，毕竟，人民的智慧才是无穷的。

最后，政府也将在云计算浪潮中扮演重要的角色，需要运作大型的云计算中心来完成对各个领域大量数据的管理、整合和处理。事实上，已经有多个地方政府着手建立拥有海量存储和庞大计算能力的信息处理中心。想一想美剧反恐24小时中的场景，哪个政府不希望拥有那样的信息处理能力呢？

云计算的技术挑战

工欲善其事，必先利其器。云计算的前景虽然美好，然而还有不少的技术障碍亟需解决，主要包括高可靠的系统技术、可扩展的并行计算技术、海量数据的挖掘技术和数据安全技术。

1. 高可靠的系统技术

支撑云计算的是大规模的集群计算系统，当系统规模增大后，可靠性和稳定性就成为最大的挑战之一。需要通过有效的系统配置、监控、管理、调度、虚拟化等技术，实现一个强大的、动态的、自治的计算存储资源池，提供云计算所需要的大容量计算力。

系统级的容错技术是系统技术方面的一个难点。大量服务器进行同一个计算时，单节点故障不应影响应用的正常运行。对类似数据检索这样计算节点间无通讯的应用，这一点比较容易实现。但对那些有大量通讯的紧耦合类应用，当前业内仍无有效的系统级容错方案。目前主要还是依赖应用层面的检查点和重启技术，一方面增加了开发的难度和工作量，另外一方面对运行性能也有一定的影响。

2. 可扩展的并行计算技术

并行计算技术是云计算的核心技术，也是最具挑战性的技术之一。多核处理器的出现增加了并行的层次性，使得并行程序的开发比以往更难。而当前业内并无有效的并行计算解决方案，无论是编程模型、开发语言还是开发工具，距离开发者的期望都有很大的差距。自动的并行化解决方案在过去的30年间已经被证明基本是死胡同，但传统的手工式的并行程序开发方式又难以为普通的程序员所掌握。Intel、微软、SUN、Cray等业内巨头正投入大量人力物力进行相关的研究，但真正成熟的产品在短期内很难出现。

可扩展性是云计算时代并行计算的主要考量点之一，应用性能必须能随着用户的请求、系统规模的增大有效的扩展。当前目前大部分并行应用在超过一千个的处理器（核）上都难以获得有效的加速性能，未来的许多并行应用必须能有效扩展到成千上万个处理器上。这对开发者是巨大的挑战。

3. 海量数据的挖掘技术

云计算面对的是TB乃至PB级的海量数据，如何从数据中获取有效的信息，这将是决定云计算应用成败的关键。除了利用并行计算技术加速数据处理的速度外，还需要新的思路、方法和算法来完成更准确、快捷、强大的数据挖掘。

除了海量数据的挖掘，海量数据的存储和管理也将是一个巨大的挑战。在云计算时代，数据库将面临严重的危机，要么将集群数据库有效扩展到成

千上万个节点，要么它就会被类似于Google文件系统这样的新技术所替代。“数据库已死”，这个断言将成为可能。

4. 数据安全技术

将原本保存在本地、为自己所掌控的数据交给一个外部的云计算服务中心，这样一个改变并不容易。网络技术的发展，使得带宽将不会成为主要障碍，安全性依旧是最重要的顾虑。然而，如同早已习惯将钱存在银行一样，未来的数据银行必将会出现，只是时间的早晚问题。技术其实不是最主要的障碍，制度、法规、诚信、习惯、观念，这些非技术的因素将决定云计算的受欢迎程度。

要闻集锦

InfiniBand有望实现远距离传输

据http://gcn.com网站2008年12月23日消息报道，美国能源部（DOE）下属的橡树岭国家实验室的研究人员证实，InfiniBand可以通过数千英里长的专用网络、以超过高速TCP/IP连接的速率来传送大型数据集。

在测试中，研究人员在8600英里的光路两端的两台计算机间实现了7.34GB/s的平均传输速率。相反，使用TCP高速版本（HTCP，超文本高速缓冲协议）的最高传输速率也仅为1.79GB/s。

橡树岭国家实验室的研究人员Nageswara Rao在最近的SC08会议上发表了名为“10GigE广域网性能分析与InfiniBand技术”的文章。DOE实验室发现他们需要的是远距离传输大型文件。在未来的几个月里，欧盟的大强子对撞机将要启动，届时将横跨大西洋把PB字节的数据传送到DOE实验室和美国的学术机构。

Rao称，以高速广域网络传递大型数据是非常困难的，其中还包括来自存储器网络的包转换和TCP/IP调谐的复杂任务。以数千英里的距离传递信息并保持端对端的传输速率仍是个十分复杂的问题。

尽管InfiniBand互连广泛应用于高性能计算机系统，但很少进行远距离通信。相反，InfiniBand主要用于在每个端点通过10GigE或其它协议把通信量转换为TCP/IP包，并在另一端把通信量转换回InfiniBand。但一些厂商，如Obsidian Research和Network Equipment Technologies已经开始利用广域设备的InfiniBand，使通信量在整个传送过程中都以InfiniBand来传递。

与一些利用10GigE TCP/IP专用形式相比，橡树岭国家实验室的官员希望了解到远距离InfiniBand的连接性的优势。

通过使用DOE试验的线路交换测试台网络UltraScienceNet，研究人员设置了一个10GB的光路，来回的传递距离达到8600英里。在每个端点，设置了一组Obsidian Research的Longbow XR InfiniBand开关，通过广域网来运行InfiniBand。该网络是一个双OC-192同步光学网络，可支持9.6GB/s的吞吐量。

研究人员发现，通过专用网络远距离传送大型文件时，InfiniBand工作良好。对于短距离来说，HTCP在0.2英里的距离内，其传送量为9.21GB/s，而InfiniBand为7.48GB/s。但随着距离增加，HTCP的性能降低。相反，随着距离的增加，InfiniBand的吞吐量始终保持良好。

然而Rao表示，HTCP在传送更多通信量时更具竞争力。这一点不足为奇，因为TCP/IP是专为分时网络设计的，是一种在多个端点间传递通信量的网络。充分利用专用网络的TCP/IP要承担相当多的工作，并不能获得最佳效果。

因此，研究人员发现InfiniBand可能成为广域网数据传递的替代解决方案。美国国防部和能源部的高性能网络计划也支持这一研究。

（肖 湄）

伯克利云计算白皮书（节选）

● Michael Armbrust， Armando Fox，Rean Griffith，Anthony D. Joseph，Randy H. Katz，
Andrew Konwinski，Gunho Lee，David A. Patterson，Ariel Rabkin，Ion Stoica，Matei Zaharia
加州大学伯克利分校 电子工程和计算机科学系

● 卢大勇[译] 上海超级计算中心 上海 201203 dylu@ssc.net.cn
● 陆 琪[译] 惠普公司 上海 201203
● 姚继锋[校] 上海超级计算中心 上海 201203

译者按：

“云计算”一词自出现以来，在产业界和学术界就掀起了波澜，众说纷纭，莫衷一是。在一些人眼里（如Google、IBM），云计算是未来的方向、潮流和必然，他们迫不及待的拥抱云计算；而在另外一些人眼里（如Oracle公司总裁Larry Ellison、GNU发起人Richard Stallman），云计算只是又一个商业炒作的概念，毫无新意，甚至蠢不可及。在过去的一年中，已经有大量的关于云计算的文章见诸博客、报纸、杂志和严肃的学术刊物。但本文无疑是迄今为止关于云计算最重要的文章之一。

文章的作者是分布式计算领域最有影响力的研究团队之一，领衔者David A. Patterson是 计算机界的权威，曾担任伯克利计算机系主任和美国ACM主席，本文是他们长达六个多月工作的结果，详细解答了什么是云计算、和以前的模型如SaaS有什么不同、为什么现在是云计算发展的最佳时机、云计算将创造什么新的机遇、有哪些挑战以及如何应对等一系列问题。

文章以内部技术报告方式发表于2009年2月10日，原文标题为“Above the Coulds: A Berkeley View of Cloud Computing”。全文可从如下地址下载：

http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf。

原文篇幅超过20页，本文为其节选，翻译全文可从上海超级计算中心网站下载。

1. 云计算：一个即将实现的古老梦想

云计算是计算作为基础设施这一长久以来梦想的新称谓，它在最近正快速变为商业现实。但到底什么是云计算？什么时候运用云计算会有效？这些问题仍然没有得到明确的解决。

我们这篇论文的目的，就是要明确一些术语，提供简单的公式来量化比较云计算和传统计算，并明确阐述云计算的发展的最大技术和非技术挑战，及相应存在的机遇。我们将尝试回答以下问题：

1. 什么是云计算，它与以前的模型如SaaS有什么不同？

2. 为什么现在正是云计算要发展的时候，而以前的尝试都是失败的？

3. 成为云计算供应商需要什么条件？为什么公司需要考虑成为一个云计算的供应商？

4. 云计算将创造什么新的机遇？

5. 如何对现有的云计算产品进行分类，他们在技术和商业上各有什么不同的挑战？

6. 云计算可能创造什么新的经济模型？服务运营者改如何决定将服务转移到云中还是保留在私有数据中心里？

7. 云计算成功的10大挑战是什么？相应有什么解决方案？

8. 将来的应用软件、基础软件和硬件都需要为适应云计算作哪些设计上的改变？

2. 什么是云计算？

云计算包含互联网上的应用服务及在数据中心提供这些服务的软硬件设施。互联网上的应用服务一直被称作软件即服务（Software as a Service, SaaS），所以我们使用这个术语。而数据中心的软硬件设施就是我们称作的云（Cloud）。

当云以即用即付的方式提供给公众的时候，我们称其为公共云，这里出售的是效用计算。当前典型得效用计算有Amazon Web Services、Google AppEngine和微软的Azure。不对公众开放的企业或组织内部数据中心的资源称作私有云。因此云计算就是SaaS和效用计算，但通常不包括私有云。在本文中，除非会引发歧义，否则我们将使用云计算这个术语。图1表示了云计算各层中的用户和供应商，我们将使用这些术语使我们的论点更将清晰。

SaaS对于最终用户和供应商的好处已经被广泛认识了。服务提供商只需要关注软件的安装、维护和版本的集中控制；最终用户可以在任何时间、任何地点访问服务，更容易共享数据和劳动，并安全的将数据存储在基础系统中。云计算不会改变这一切，而且还能为更多的应用服务供应商提供选择，因为他们可以在没有数据中心的情况下将他们的产品以SaaS方式发布。正如半导体代工的出现使芯片公司可以不拥有生产线而专注于芯片设计与销售一样，云计算使企业能不建立和提供数据中心就可以方便的发布SaaS服务。如同SaaS供应商减轻了传统的软件使用者的许多困难一样，云计算供应商将可以极大的帮助SaaS供应商。

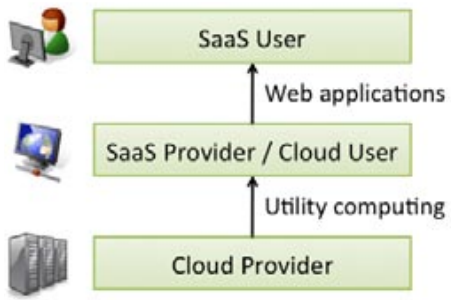


图1 云计算的用户和供应商

从硬件上看，云计算在三方面突破了传统。

1. 云计算能所需应变的提供使似乎无限的计算资源，云计算终端用户无需再为计算力准备计划或预算。

2. 云用户（SaaS服务供应商）可以根据需要，逐步追加硬件资源，而不需要预先给出承诺。

3. 云计算提供其用户短期使用资源的灵活性（例如：按小时购买处理器或按天购买存储）。当不再需要这些资源的时候，用户可以方便的释放这些资源。

我们认为这三点都是云计算可能对技术和经济造成的重要变革。事实上，我们注意到，过去在效用计算上的努力都失败了，正是因为遗漏了这三大特性中的一到两点。举例来说，2000-2001年的英特尔计算服务要求以合同方式被长期使用，而不是以小时为单位的购买方式。

云计算对于其用户（SaaS供应商）的吸引力已经非常明了了，那么谁将成为云计算的供应商呢？他们为什么要这样做？首先，实现所谓复用性和大宗采购这样的规模经济需要建立超大型数据中心。建造、提供、启动这样一个数据中心需要数百万美元的投资。另一个重要的条件是，这些公司还必须开发出可扩展的基础软件（如MapReduce、Google的文件系统、BigTable和Dynamo）以及配备专业的运维人员，以确护数据中心免受物理或电子攻击。

因此，成为云计算供应商的必要非充分条件是：必须已经拥有非常大型的数据中心、大规模的基础软件和运维数据中心的高级人才。在此前提下，以下因素也可能影响一个公司是否能成为云计算的供应商：

1. 能挣很多的钱。一个足够大的公司仍然可以利用规模经济，以低于中等规模公司的成本提供很好的服务，同时获得可观的利润。

2. 利用已有的投资。在现有体系中增加云计算服务，可以新增一种收入方式。理想情况下，追加的成本并不高，而且能分摊前期数据中心的巨大投资。

3. 捍卫特许经营权。随着传统服务器和公司应用转入云计算，拥有特许经营权的供应商将希望为他们自己的应用提供云。

4. 占据技术要塞。拥有足够数据中心和软件资源的公司都希望在云计算时代真正来临之前建立自己的立足点。Google AppEngine提供了另一种迁移到云环境的方式。它的吸引力在于它提供大量自动化的可扩展性和负载均衡的功能，这样开发人员不再需要在开发中考虑这些问题了。

5. 利用客户关系。IT服务企业，如IBM全球服务通过他们提供的服务，拥有广泛的客户关系。提供一个品牌的云计算，客户可以不用担心迁移过程，从而维持双方的投资和客户关系。

6. 成为一个平台。正如我们所见到的那样，Facebook提倡的应用程序插件方式是非常适合云计算的。Facebook应用插件的提供商Joyent，同时也是一个云计算供应商。但是，Facebook的动机是让他们的社交网络应用变成一个新的开发平台。

3. 云的大风暴：为什么是现在？

尽管我们认为建造和运营超大规模商用数据中心是云计算可能的前提，但是技术发展的趋势和新的商业模式是使之现在能成为现实的关键。一旦云计算开始腾飞，以前无法理解的应用机遇和使用模型都将被发现。

3.1 新技术趋势和商业模式

随着Web 2.0的出现，“高接触、高利润、高承诺”的服务提供方式正在朝“低接触、低利润、低承诺”的自助式服务转变。Amazon Web Services根据这一观点，在2006年提供无合同的即用即付计算服务：所有用户需要的只是一张信用卡。第二个创新点就是出售硬件级的虚拟机时，允许客户互不影响地选择他们各自需要的软件，通过共享硬件降低成本。

3.2 新的应用机遇

虽然我们还没有看到云计算从根本上创造新型的应用，但是我们认为，几个重要类型的现有应用将变得更加引人注目，并在将来推动云计算的发展。Jim Gray在2003年调研技术发展趋势的时候得出这样的结论：经济必要性要求我们将数据存储在应用程序的附近，因为广域网的成本（保持相对昂贵）比其他所有IT硬件设备的成本都要降低得更慢。虽然自Gray分析之后，硬件成本已经发生了改变，但是他的“盈亏平衡点”的观点并没有改变过。

移动互动应用程序。Tim O’Reilly认为“将来是属于那些能对人或者其他传感器提供实时响应的服务”。这种服务是非常适合云的，因为他们不仅要求高可用性，而且通常需要大型数据中心妥善存储大量数据。更突出的例子就是那些需要综合两个或多个源信息或服务提供综合服务的应用，如mushups。

并行批处理程序。云计算为批处理和数据分析提供了独特的机遇，TB级的数据分析将可以在数小时内完成。如果应用程序中的数据具有足够的并行性，用户可以利用云提供的成本特性，即相同成本下同时使用大量机器能在短时间内完成少量机器需要长时间才能完成的工作。

分析需求的兴起。计算密集型批处理的典型案例就是商业分析。虽然大型数据库工业起初主要针对事务处理，但是这种需求已经不再增长了。现在越来越多的计算资源用于理解客户、供应链、购买习惯及排名分析等数据分析问题上了。因此，网数据库处理的资源平衡点正在从事务转向商业分析。

计算密集型桌面应用程序的扩展。最新版本的数学软件包Matlab和Mathematica可以通过云计算进行复杂的评估计算。其他的桌面应用程序可能同样可以无缝地扩展到云。

地域局限的应用程序。一些应用程序可能本身非常适合云的伸缩性和并行性，但是却因为数据传输成本或者数据传输延迟的根本性限制而不得不放弃使用云计算。除非广域网数据传输的成本（或者延迟）降低，否则这样的应用程序就不适用于云。

4. 效用计算分类

任何应用都需要计算模块，存储模块和通讯模块（这里假设应用是简单分布的）。对于获得弹性和无限计算能力（需要将资源进行虚拟化）可复用性是必要的（如何实现复用与共享是被程序员所隐藏的）。我们的观点是，不同的效用计算将由展现给开发者的抽象层次的不同和各种计算存储资源管理层次的不同这两者所决定的。

Amazon EC2是一个典型案例。一个EC2实例看上去像一台物理硬件，用户可以控制几乎除核心以上的所有软件堆栈。另外一个典型案例是面向特定领域的应用平台，像Google AppEngine和Force.com（SalesForce商业软件开发平台）。微软的Azure是一个介于灵活性（如同EC2那样）和编程方便性（如同Goolge AppEngine一样）两者之间的案例。

在云计算领域是否存在一种模式最终将取代其他模式？我们可以参照编程语言和框架的来进行说明。类似于C和汇编语言这样的底层语言能很好的控制物理机器的通信，但如果开发人员写的是Web应用程序，那么套接字的管理，分发请求等等用这类语言，即使采用好的库来写也是繁重而乏味的。另一方面，上层框架，例如Ruby on Rails使得这些机械性的工作对于程序员透明化，但这只能用于那些符合这个框架的应用来使用，并且这些应用只能使用框架所提供的东西；任何框架不提供的东西需要深入框架内进行修改，这通常会非常困难。另外，就像上层语言被底层语言实现一样，具备高可管理性云计算平台可以建立在低可管理性平台的基础上。举例来说，AppEngine能够建立在Azure和EC2的顶部；Azure能够建立在EC2的顶部。

5. 云计算经济学

5.1 弹性/伸缩性：转嫁风险

虽然云计算的经济吸引力常常被描述成“将资金支出转变为运营支出”，我们相信“即买即用”这一说法更准确的描述了购买者所能看到的经济方面的好处。通过云计算购买的计算时间可以用不同的方式实现（100台机器算1个小时或者1台机器算100个小时）。此外，由于不需资金来提前进行计算能力的部署，这部分资金可以投入到利润空间较大的核心业务中去。

伸缩性关键的一点是云计算可以细粒度的添加和移除资源（例如以服务器或者处理器为单位），并且时间的计算位是分钟而不是周，这将能使计算

资源跟计算负载可以很好的匹配。而现实中数据中心的服务器利用率大概仅在5%到20%左右。这听上去低得吓人，但考虑到许多服务的峰值工作量比平均值要高2-10倍，就容易理解了。很少有用户部署系统低于峰值需求，这就必然导致了非峰值时间资源的浪费。负载的波动性越强，导致的浪费就越多。

云计算将错误估计负载的风险从服务运营商转移到了云提供商。云提供商可能需要收取额外的费用以承担这个风险。

5.2 成本比较：应该转向云计算吗？

如果从经济角度考虑，是该将现有的基于数据中心的服务转向云计算，还是继续驻留在数据中心？

我们统计了2003年和2008年计算资源成本，给出了2003年和2008年1美元所能购买资源的比较结果，并且给出了在2008年的价格基础上，与AWS上使用1美元价值的资源其实际成本的对比。乍一看，2008年1美元的硬件采购将比租用1美元相同硬件的要有优势。然而这个简单分析遗漏了许多重要的因素。

每个资源单独支付。大多数应用所使用的计算、存储和网络带宽都不是相同的；一些应用主要使用CPU，另外的主要占用网络资源等等，并且这些应用都不能充分用满一个单一资源。直接购买云计算能够将应用与各种类型的资源隔离开来，能够降低不能充分利用资源的浪费。

电力、制冷和场地成本。电力、制冷和放置硬件的场地成本，这些因素在我们的分析中并未考虑。据粗略估计：在分摊放置硬件的建筑物的使用周期成本后，将会使CPU、存储和带宽的成本增加一倍。

运维成本。当前用于运维硬件的成本是十分低的——重启系统非常简单（可以按IP地址或者分组控制电源开关），而且经过简单培训的维护人员就可以进行服务器或者机柜级的损坏组件更换。一方面，由于效用计算是采用虚拟机而不是物理机，从云用户的观点来看，这些工作是云供应商应该承担的。

6. 云计算10大问题与机遇

本章中，我们将云计算发展所面临的问题进行排序，给出一个列表。其中每一个问题对应一个机遇——即我们对于克服相应问题的想法，包括一些直接的产品开发到主要的研究项目。表1中给出了10大问题与机遇。

表格1 云计算发展的10大挑战及相应的机会

	问题	机会
1	服务的可用性	选用多个云计算提供商；利用弹性来防范DDOS攻击
2	数据丢失	标准化的API；使用兼容的软硬件以进行波动计算
3	数据安全性和可审计性	采用加密技术，VLANs 和防火墙；跨地域的数据存储
4	数据传输瓶颈	快递硬盘；数据备份/获取；更加低的广域网路由开销；更高带宽的LAN交换机
5	性能不可预知性	改进虚拟机支持；闪存；支持HPC应用的虚拟集群
6	可伸缩的存储	发明可伸缩的存储
7	大规模分布式系统中的错误	发明基于分布式虚拟机的调试工具
8	快速伸缩	基于机器学习的计算自动伸缩；使用快照以节约资源
9	声誉和法律危机	采用特定的服务进行保护
10	软件许可	使用即用即付许可；批量销售

问题1：服务的可用性

现今，绝大多数互联网服务提供商都会利用多个网络提供商来使自己避免在一个单独公司出现故障的情况下导致从互联网上消失，我们相信唯一合乎情理的提供极高可用性服务的解决方案就是多云计算提供商。

可用性相关的另一个问题是DDoS（Distributed Denial of Service，分布式拒绝服务）攻击。由于弹性，云计算将攻击目标从SaaS提供商转移到能够马上吸引攻击并且具有DDoS攻击保护能力的效用计算提供商。

问题2：数据丢失

虽然现在的软件集已经在跨平台性方面改进了很多，但是从本质上说云计算的API仍是私有的，或者说当前没有建立起统一的标准。因此，用户很难将他们的数据和程序从一个站点迁移到另一个。这也是很多用户不愿采用云计算的原因。

显然，将云计算API标准化是上面问题的解决方案，这样SaaS开发人员能够在多个云计算提供商处发布服务和数据。某个公司的失误或故障将不会影响到用户数据的每个拷贝。

问题3：数据安全性和可审计性

过去一段时间中我们曾听到多次这样的话：“我们企业的那些敏感数据将永远不会放到云中。”当前的云从本质上来说是提供了公共（而不是私有）网络，因此会遭受更多的攻击。可审计性也很重要，按照《萨班斯法案》和《健康保险携带和责任法案》等相关法律规定，企业提交到云中的数据必须满足审计需求。

我们相信建立同现有的各类内部IT环境同样安全的云计算环境是不存在任何根本问题的，灵活的运用加密存储、虚拟局域网、网络中间件（例如防火墙、包过滤）等技术就能迅速的解决当前已经碰到的一些问题。

问题4：数据传输瓶颈

当前应用发展越来越趋向于数据密集型。如果应用能够被拆分交由不同的云去处理，这将导致复杂的数据存放和传输。

快递硬盘是克服这样高成本的互联网传输成本问题的一种机遇。Jim Gray发现，传输大量数据最便宜的方式是用次晨达的快递方式来快递硬盘，甚至整个计算机。虽然没有硬盘厂商和计算机厂商的保证，我们认为以快递硬盘这种方式来进行数据传输是可靠的。

第二个机遇是发现在云中保存数据的吸引人之处。由于一旦数据存放在云中，数据传输将不在是一个瓶颈，这将会催生其他一些利用云计算能力的新服务。

第三个更加激进的机遇是试图快速降低广域网带宽的成本。除了广域网带宽成为一个瓶颈外，云内网络技术也可能是一个性能瓶颈。

问题5：性能不可预知性

我们的经验表明，在云计算中多虚拟机能够很好的共享CPU和内存，但是I/O的共享却有明显的问题。

一个机遇是改进体系结构和操作系统以获得更有效率的虚拟中断和I/O通道。另外一个可能是闪存能够降低I/O冲突。闪存相比硬盘能够支持更多的单位时间内的I/O操作，因此，有着随机I/O访问冲突的多虚拟机能够很好的协同工作，而不会出现使用机械硬盘时常见的互相干扰。

最后一个不可预期的问题是运行某些批处理程序时多个虚拟机的调度问题，这在高性能计算领域尤其明显。克服这一问题的机遇就在于为云计算提供一个有效的资源调度和管理工具。

问题6：可伸缩的存储

目前针对这个问题有许多不同的尝试，从提供丰富的查询和存储API，提供性能保证，到由存储系统支持数据结构，都各不相同。存在的机遇是创建一个存储系统，不仅具备上述功能，而且提供向上和向下的伸缩性支持，同时在可扩展性、数据持久性以及高可用性等数据管理方面满足程序员需求。

问题7：大规模分布式系统中的错误

云计算中一个很大的挑战是从大规模分布式系统中去消除错误。一个经常出现的问题是这些bug不会在稍小规模的配置中重现，因此调试必须在生产环境中进行。

云计算中的虚拟机可能会成为一个机遇，它可能使程序员获得在传统分布式系统上难以获得的一些有价值的信息。

问题8：快速伸缩

即用即付无疑很适合存储和网络带宽，这两者都可以用使用字节数来衡量。由于使用虚拟机，计算稍稍有些不同。这里的机遇是在不违背使用协议的前提下能够实现根据负载自动并快速的调整计算规模以能够最大的节省费用。

问题9：声誉和法律危机

一个用户的恶意操作会影响到整个云的声誉。创建类似于信任邮件服务那样的声誉保护服务将可能会成为一个机遇。

另外一个问题是法律责任的转移——当出现问题是，云计算提供商将会希望由用户去承担相应法律责任，而不要将责任转嫁给他们。

问题10：软件许可

当前软件许可证通常限定在运行软件的机器上。用户购买软件并按年支付维护费用。许多云计算提供商从一开始就倾向于开源软件，部分正是因为商业软件许可证模式并不适合效用计算。

首要的机遇要么开源持续流行要么商业软件公司改变他们的许可证结构，让其更加适合云计算。还有一个办法是鼓励软件公司制定销售政策来向云计算提供产品。将软件公司销售部门的一些反对派转到支持云计算者的阵营中来。

7. 总结与云计算的展望

计算作为一项服务功能是人们长久以来的梦想。使用的弹性符合了通过互联网向用户直接提供服务的商业需求，因为，相比20年前，工作量的增长和收缩变得更加快速。过去要花好几年时间来增加业务和发展几百万用户，现在只要一个月的时间就可以做到。

从云供应商的观点出发，利用商品化的计算、存储和网络低成本的建立大型数据中心使得以低于许多中等规模的数据中心的价格“即用即付”的销售资源成为可能，并且利用资源在大量用户间的复

用来获取利润。从云用户的观点出发，云计算可以令一个初创的软件公司象初创的芯片厂商拥有为之服务的代工厂一样拥有自己的数据中心。除了初创公司，许多老牌公司或机构同样充分利用了云计算的伸缩性。

虽然，云计算提供商可能碰到上文提及的问题，我们相信经过长期的运行，这些提供商将会成功的完成这些挑战并建立一套可以让其他提供商效仿的运营模式，也许正是通过成功的把握我们提及的解决这些的问题的那些机遇来实现。

因此，开发人员需要明智的来设计下一代的系统，以适应云计算。一般来说，重点应该放在成百上千个虚拟机上运行应用的横向可扩展性，而不是考虑单个系统的使用效率。这隐含了如下几点：

应用软件，将来的应用软件将会同时在客户端和云中运行。在云中的运行部分需要具备快速的向上和向下伸缩的能力，这是对软件的新需求。客户端部分需要在和云断开的情况下仍然可用，这和现在的Web2.0技术不同。同时，这样的软件必须采用支付使用许可证的模式来满足云计算。

基础软件，将来的基础软件不仅能在物理机上运行，也需要能够在虚拟机上运行。此外，基础软件需要在一开始就建立自己的记账系统。

硬件系统，将来的硬件系统需要被设计成按某个容器（至少12个机柜）而不是按单个服务器或者单机柜进行扩展，因为这个容器将是云计算时代采购硬件的最小单位。运维的成本将与购置成本变得一样重要，当内存、磁盘和网络闲置的时候，可将其调至节能状态。处理器要求在虚拟机环境下能很好的工作，闪存应该被加入到存储结构中，局域网交换机和广域网路由器都需要在带宽和成本上进行改善

最后，我们对云计算的将来充满信心，并急切的想知道5年后云计算到底是什么样子：

随着时间的推移技术和价格的改变：更高层的虚拟云的结算单位是什么？不同资源价格将会变成什么样子？明显的，单个芯片的核数将会不断增长，每2到4年增加一倍。闪存将很有可能新增为传统的存储层次中速度较快的一层；那么它的结算单位将会是什么？作为当前发展最慢的技术，技术或者商业的创新是否能加速网络带宽价格的下落？

虚拟化层：云计算是由EC2这样的底层硬件虚拟机主导，还是由类似微软的中间层语言Azure，或者像Google AppEngine这样的上层应用框架主导？我们将会有更多的虚拟层次来满足不同的应用吗？一些提供增值服务的公司会在效用计算大潮中生存吗？还是所有成功的服务都被云提供商整合？如果他们不认同单一的虚拟层，多家公司会信奉一套通用标准吗？这会导致价格低限的出现使得云计算提供商失去吸引力吗？或者他们会区分服务的质量来保证利润吗？

并行图形绘制技术综述

韩伟杰 李晓梅 张文

装备指挥技术学院信息装备系 北京 101416 visc_hwj@126.com

摘要：

并行绘制技术可以有效提高图形绘制的效率，满足大规模数据集的绘制需求。图形绘制过程以流水线的方式组织，其内在的可并行性构成并行图形绘制的基础。本文首先介绍图形绘制的流水线过程，并对其内在的可并行性进行分析；然后介绍并行绘制的各种实现方式和面临的主要问题；最后讨论并行绘制的发展趋势。

关键词：图形绘制, 流水线并行, 数据并行, 作业并行, 拼接合成

1. 引言

随着科学研究和工程设计的深入发展，计算的规模和数据处理的规模也越来越大，如气象、生物技术、核技术、油藏模拟等领域以及卫星数据的处理，其计算和处理数据的规模已经达到TB甚至PB量级。如何快速、准确地表现海量数据，实现对大规模数据的可视化，对传统的绘制方法提出了挑战。并行绘制是解决这一挑战问题的有效方法之一。所谓并行绘制，就是指利用多个图形硬件或图形绘制流水线，使用它们累加的绘制能力来完成绘制任务[1]。并行绘制可以利用有限的绘制条件交付高端图形工作站所能提供的绘制能力，为科研人员完成大规模数据集绘制提供了实现途径。

本文首先分析了图形绘制的流水线过程，并对其内在的可并行性进行分析，在此基础上引出并行绘制的各种实现方式，并对实现并行绘制面临的主要问题进行了分析，最后总结归纳了并行图形绘制系统的主要实现方法，并讨论了并行图形绘制技术未来的发展趋势。

2. 图形绘制流水线及其可并行性分析

图形绘制一般以流水线作业的方式分阶段实现图形计算，它主要用来描述图形数据处理的过程，将整个图形处理划分为多个流水阶段。图形绘制流水线的组织方式决定了图形绘制过程存在着可并行性。

2.1 图形绘制流水线

图形绘制流水线由两个阶段组成：几何处理和光栅化，如图1所示。几何处理阶段将一个三角形从三维坐标（物体空间）映射到二维坐标系统（图像空间）。光栅化阶段将处理后的三角形转化成像素显示到计算机屏幕上。

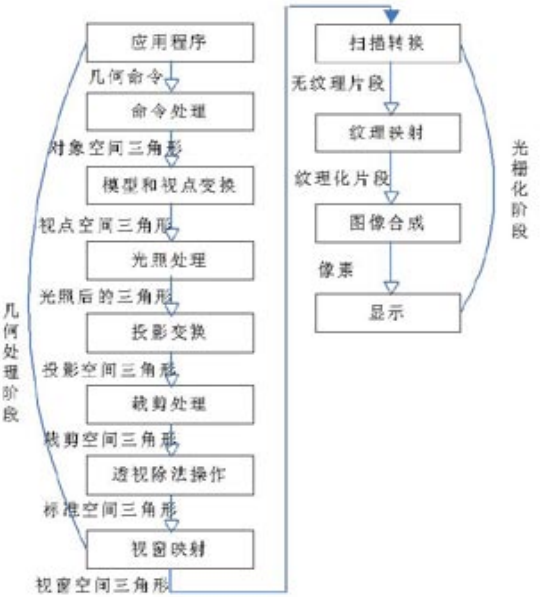


图1 图形绘制流水线

2.2 图形绘制流水线可并行性分析

图形绘制流水线的阶段化结构使其计算步骤独立，非常适合模块化实现。此外，绘制计算的基本处理单元为三角形面片，面片之间的计算无需相互引用，也无顺序性要求，其计算过程存在数据弱相

关性。所以，从并行计算理论来看[2]，图形绘制是一种流水线式的结构，具有模块化和数据弱相关的特点，比较适合并行处理，不过还需要进行计算任务的划分和计算结果的合成，并用某种方式加以组合。

图形绘制模块化的特点使其非常适合功能并行。此外，绘制系统由一系列顺序相连的处理单元组合，每个处理单元执行一个功能，并将结果输出到下一单元，这样的结构适合流水线并行。流水线并行技术已经得到了充分的发展，成为当代微处理机性能提高的关键技术[3]。一条绘制流水线可以处理一个顺序的图元序列，而多条绘制流水线的组合还可以构成其它类型的并行方式。

3. 并行绘制方法

从图形绘制过程的流水线结构和模块化特点角度考虑，并行绘制方法可以分为流水线并行、数据并行和作业并行。

另外，当多条绘制流水线共同工作生成一幅 $M \times N$ 的图像时，有两种实现途径：（1）流水线各自生成一幅 $M \times N$ 图像，但它仅包含部分图元的深度图像，最后合成得到最终图像。（2）流水线各自生成部分子图像，最后进行拼接合成最终图像。在这种方式下，需要判定图元的归属，并确定图元在流水线之间的传输发生在流水线的哪个阶段。根据实现方法的不同可将并行绘制分为sort-first（前分布拼接合成）、sort-middle（中分布拼接合成）和sort-last（后分布拼接合成）三种结构方式。

3.1 流水线并行、数据并行和作业并行

流水线并行[4,5]是指绘制流水线本身并行执行。图形绘制由一系列顺序相连的处理模块组成，每个处理模块执行一个功能，并将结果输出到下一模块，这样的系统称为绘制流水线。当绘制流水线的各个模块同时并行执行时，其理论加速比等于流水线所包含的模块数目。流水线并行存在两个重要的局限性，一是流水线的整体加速比受限于流水线中最慢的阶段，所以在设计功能模块时要避免出现这种瓶颈。另外，并行效果还受限于流水线所包含的阶段数目。

数据并行[4,5]是将数据划分成子数据流，在一些相同的处理模块上对这些子数据流进行处理。这种方式的并行效果不受绘制流水线阶段数的限制，但受制于相同处理模块的数目和系统内部的通信带宽。由于多边形绘制算法的数据相关性较弱，所以数据并行方式比较适合，并行度很高。此外，数据并行方式具有良好的可扩展性，可以构建大规模模块的绘制系统。

作业并行[5]是指在一个绘制流水线中，某些独立模块之间并行执行。作业并行适合于流水线中有多个独立分支的情况。此时，可以使用多个进程（或线程）分别处理这些独立分支。作业并行的缺点是其并行度受到流水线中独立分支个数的限制，而且由于各个作业处理操作的差异，难以做到负载平衡。

3.2 Sort分类方式

Sort-first[6]方式是在几何处理阶段对几何图元进行重新分布。该方法将输出图像划分为一些不相交的区域，如小矩形或连续的多条扫描线，每条绘制流水线负责一个或多个区域。图元在进入流水线之前先进行计算以确定其覆盖的区域，此计算被称为“预变换”，通常是计算出图元在屏幕上的外包围盒并进行比较。之后，图元进入相应的流水线，一个图元有可能覆盖多个区域而进入多条流水线。图元分布完成之后，各条流水线单独工作，它们输出的子图像拼接成为最终图像。此方式的优点是流水线相互独立，通信带宽较小。但图元在屏幕上的分布不均匀及图元复杂程度的差异容易导致系统负载不平衡。

Sort-middle[1,6]方式是在绘制流水线的中间重新分布几何图元。该方法对图像的划分和拼接过程与sort-first相同，不同之处是图元乱序进入流水线的几何处理单元，经过几何变换之后，图元被转换成2D屏幕坐标，然后根据屏幕坐标被分布到正确的光栅化处理单元。该方法在几何处理和光栅化之间进行图元分布，其阶段化的处理方式有利于模块化实现。

Sort-last[7]方式是在绘制流水线的末端重新分布几何图元。在这种方式下，图元乱序进入多条流水线，每条流水线独立完成几何处理和扫描转换，生成一幅包含部分图元的完整图像，最后做深度合成。此方式有利于实现负载平衡，但其深度合成操作容易成为瓶颈。

4. 并行绘制面临的主要问题

基于以上对并行绘制实现过程的分析，可知其需要解决的主要问题包括：负载平衡、图元分配、可扩展性和图像合成等。

4.1 负载平衡

负载平衡是并行算法实现高性能计算的重要技术[8]。一般而言，任务划分的粒度越细就越能保证系统的负载平衡，但必须在任务划分的粒度和消息传递的数量之间做出折中，过细的任务划分会导致较大的任务划分开销，同时也会带来更多的消息传

递，占用太多的网络资源[9]。

负载平衡策略主要是针对具体的类型而言，一般分为三种类型：静态调度策略、动态调度策略和自适应调度策略[10]。静态调度策略给每个处理器分配一个固定的任务，这种策略在每次绘制之前没有任务分配开销，对于场景对象分布不均匀的情况会造成负载严重失衡。动态负载调度策略在每帧绘制之前分配任务，由此可以优化处理器的负载。自适应的任务分配算法在运行过程中动态地调整任务分配。上述三种策略都可归纳为“基于几何数据分析”的方法，其它如Whitman自顶向下分解算法[11]和MADH算法[12]都是基于上述思想提出的。此外，浙江大学提出了基于时空转换的负载平衡策略[13]，直接以服务器的绘制工作时间作为负载的度量，将时间值转换为空间值，再以空间值来控制对绘制服务器的任务分配。

4.2 图元分配

图元分配计算过程要使用包围盒，一般有三种实现策略[14]。其一是对每个三角形不做几何处理，直接对三角形求包围盒，根据包围盒将该三角形分配给目标图形处理器。其二是首先将三角形变换到二维空间，然后对变换到二维空间的三角形求包围盒，最后根据屏幕划分情况将其分配到目标区域。其三是对一组三角形直接求包围盒，然后将该包围盒变换到透视投影空间，最后将这一组三角形分配给目标图形处理器。其中，第三种策略所做的目标区域查询量最小，同时成组发送也可以降低传递的消息量，减少网络数据打包的开销，比较适合于并行实现。

此外，Princeton大学提出了一种图元分配策略[15]，该方法通过硬盘加载的方式实现模型对象的分布，并通过消息机制实现同步，保证绘制命令的正确执行，同时也实现了绘制命令对象的分布。但该方法存在封装性较差、特定应用程序需重写等问题。此外，浙江大学提出了基于保留模式并行绘制的图元对象分布策略[13,16]，通过对象的全局唯一标识和函数调用检查机制实现对象的远程调用，并通过状态和数据缓存提高系统效率，实现了多机绘制的同步。

4.3 可扩展性

并行图形绘制系统的可扩展性主要体现在三个方面：系统节点数目的可扩展性、系统节点部件的可扩展性和问题规模的可扩展性[9]。要实现节点数目的可扩展，就必须考虑减少节点之间消息传递的数量和降低消息传递的时延，例如采用树状的通信结构，增加树的层次降低系统消息传递的开销。系统节点部件的可扩展性主要表现在，如3D图形加速显示卡的更新和升级换代、网络带宽和CPU计算能力的提升等。问题规模的可扩展性主要是指当问题规模增大时，算法的计算时间复杂度是否为线性变化等。从中可以看出，网络通信带宽是影响可扩展性的重要因素。

如何缓解网络带宽瓶颈问题，解决方法主要致力于减少需进行归属判断的三角形数量，其方法大致分为三类：1）整个模型数据被完整拷贝到全部或部分绘制节点，以减少传输数据的频率[17,18]；2）采用渐进网格模型，以减少实际需调入的信息量[19]；3）对模型进行压缩，在不损失信息的条件下减少传输量[20]。此外，浙江大学[21]提出构建一个三角形条带数据压缩框架，即视点连贯性的分片条带压缩(VSPC)方法，该方法可以进一步优化数据压缩性能。

4.4 图像合成

并行图形绘制的图像合成步骤容易成为系统瓶颈，原因主要有两个方面[6,7]。一是合成模块每次只能合成两幅图像，当节点数目增多时，需要进行多次合成操作才能产生最终图像。二是需要将多幅图像从绘制节点传输到合成节点，由此会给网络或总线造成很大的传输负担。针对问题一，一般采用二叉树合成算法、二分交换合成算法[22]和并行流水线合成算法[23]解决。针对问题二，相对于传输整幅图像，可以只传输活跃像素以减小传输量，或者使用游程编码对图像进行压缩并基于压缩图像进行合成[24]，也可以有效减小图像数据量，但必须考虑算法的开销。

针对二分交换合成方法不适用于分布式环境的问题，清华大学[25]提出了基于虚拟机的并行绘制方法，该方法采用异步二分思想，在分布式环境下在$O(\log n)$时间内完成对n幅局部图像的并行合并，并在虚拟机环境中进行实现。

5. 并行图形绘制系统的实现

并行图形绘制系统的实现主要有以下两种方式：

（1）基于高端多处理器和高性能图形工作站的实现

并行图形绘制系统传统上多采用高端多处理器和多流水线的高性能并行工作站实现，如SGI采用sort-middle方法实现的 Infinite Reality 系统[26]，通过顶点总线对像素片段生成器进行广播，每秒可绘制700万个光照、纹理、反走样的三角面片。此外，UNC采用sort-last方法实现的PixelFlow系统[27]，采用全图像合成方法实现了可伸缩的并行图像处理，并

采用像素流结构实现真实感图像的绘制。

但是，这些系统由于其设备价格昂贵，并且难以扩展和升级，由此严重阻碍了它们的普及和推广。

（2）基于PC集群的实现

随着高性能微机图形卡的出现，基于PC集群构建并行图形绘制系统已经成为并行绘制的发展趋势[28]。

洛杉矶Alamos国家实验室[5]采用流水线并行、数据并行和作业并行方法，在集群环境下实现了对海洋模拟数据的并行绘制。此外，装备指挥技术学院并行计算实验室[29]在并行绘制方面也进行了相关探索，在集群环境下基于VTK实现了对人体头部CT扫描数据的并行绘制。

Stanford大学的WireGL是第一个基于sort-first结构实现的并行图形绘制系统，它第一次对并行绘制的体系结构、运行机制等关键技术问题提出解决方案。Chromium是WireGL的后续版本，明确提出了“流处理”的概念，可以实现sort-first和sort-last两种结构。浙江大学的AnyGL系统在WireGL的基础上进行了多方面的扩展，实现了一个混合sort-first和sort-last结构的并行图形绘制系统。此外，浙江大学的D3DPR并行图形绘制系统基于Direct3D9实现并行绘制，可以提供具有更强真实感和沉浸感的虚拟环境。

Princeton大学的Display Wall[15]是第一个利用多个投影仪拼接投影显示墙实现大视野高分辨率显示效果的并行图形绘制系统，该系统采用sort-first结构。此外，浙江大学也实现了一个多屏幕拼接显示并行图形绘制系统MSPR[17]，系统提供了一套OpenGL应用程序接口。

基于集群构建并行图形绘制系统，其优点主要体现在以下几点：

（1）将无法在单机上完成的绘制任务分布到集群环境中的多个PC节点上，满足目前三维图形应用领域中场景数据日益膨胀的绘制需求；

（2）驱动多台投影仪构建多屏幕拼接投影显示墙，营造大视野、高分辨率的视觉效果。

（3）基于PC集群的并行图形绘制系统具有高性价比、可扩展性好、升级方便和适用范围广等优点。

6. 并行图形绘制的发展趋势

并行图形绘制技术可以有效地提高图形绘制的效率，该领域的研究已经取得了多方面的成果。并行图形绘制以后的发展趋势主要包括以下几个方面：

（1）基于集群构建并行图形绘制系统是重要研究方向，有着广阔的发展空间。当前应用领域比较成熟的并行绘制系统皆是基于集群环境实现[15]。

基于PC集群的并行图形绘制系统面临的主要问题是网络带宽容易成为瓶颈[14]，如何降低网络传输数据量、缓解网络带宽压力是未来研究的一个重点。

（2）前分布拼接合成、中分布拼接合成和后分布拼接合成是实现并行绘制的三种重要方式，基于这三种方式可以构建相应的并行图形绘制系统。但这些系统往往是基于其中的一种[6,7]或者是混合使用其中两种方式进行实现[14]，系统结构存在着一定的局限性。因此，如何有效地混合使用这三种方式实现并行绘制是以后研究的一个重要方向。

（3）并行绘制面临的主要问题包括负载平衡、粒度划分和几何指令流的压缩等。粒度划分越细就越容易实现负载平衡，但由此也会增加网络流量。另外，几何指令流的压缩对系统性能和可扩展性有着重要影响，并且会影响到系统的负载平衡。所以，如何实现这些问题的折中和优化也是一个挑战[20]。

（4）网格技术能够通过动态的资源组织满足数据存储和计算的要求，实现自治和动态的资源管理以及数据采集、存储和计算的分布。网格技术的出现可以有效解决计算资源缺乏的问题，为并行绘制提供共享地理分布的数据源、分析设备和可视化设备等，面向网格环境的并行绘制将会成为并行绘制的一个重要研究方向。

7. 结束语

并行绘制可以提高大规模数据场绘制的效率。图形绘制过程以流水线的方式组织，其内在的可并行性构成并行绘制的基础；按照绘制流水线的不同组织方式，并行绘制可分为流水线并行、数据并行和作业并行，按照图元分布的不同可分为sort-first、sort-middle和sort-last型结构；并行绘制需要解决的主要问题包括负载平衡、图元分配、可扩展性和图像合成等；并行绘制系统主要基于高性能图形工作站和集群环境实现。

参考文献：

[1] Steven Molnar, Michael Cox, David Ellsworth et al. A Sorting Classification of Parallel Rendering [J]. IEEE Computer Graphics and Applications. 1994, 14(4):23:32.

[2] 李晓梅, 莫则尧, 罗晓广等. 可扩展并行算法的设计与分析[M]. 北京: 国防工业出版社, 2000.

[3] 李晓梅, 吴建平. 数值并行算法与软件[M]. 北京: 科学出版社, 2007.
[4] NASA. Parallel Rendering [R]. NASA Contractor Report 195080 ICASE Report No. 95-31. 1995.
[5] Ahrens J, Charles Law, Will Schroeder, et al. A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Dataset [R]. Los Alamos National Laboratory, 2000.
[6] Bethel E. Wes, Humphreys Grey, Paul Brian. Sort-First Distributed Memory Parallel Visualization and Rendering [C]. Proceedings of IEEE Symposium on Parallel and Large Data Visualization and Graphics, Washington: IEEE Press, 2003:7-16.
[7] James Ahrens, James Painter. Efficient Sort-Last Rendering Using Compression-Based Image Composition [J]. Second Eurographics Workshop on Parallel Graphics and Visualization, Eurographics, 1998:145-151.
[8]李晓梅, 黄朝晖等. 并行与分布式可视化技术及应用[M]. 北京: 国防工业出版社, 2001.
[9] Hong feng Yu, Kwan-Liu Ma. A Study of I/O Methods for Parallel Visualization of Large-Scale Data [J]. Parallel Computing. 2005(31):167-183.
[10] Clematis, Agostino, Gianuzzi. Load Balancing and Computing Strategies in Pipeline Optimization for Parallel Visualization of 3D Irregular Meshes [J]. EuroPVM/MPI. 2005(3666):457-466.
[11] Scott Whitman. Dynamic Load Balancing for Parallel Polygon Rendering [J]. IEEE Computer Graphics and Applications. 1994, 14(4):41-48.
[12] Muller C. The Sort-First Rendering Architecture for High-Performance Graphics [C]. Proceedings of the 1995 Symposium on Interactive 3D Graphics. 1995. 75-82.
[13] 金哲凡. 保留模式图形并行绘制研究[D]. 杭州: 浙江大学, 2003.
[14] Samanta, Funkhouser, Kai Li, et al. Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs [J]. SIGGRAPH/Eurographics Workshop on Graphic Hardware. August 2000.
[15] Kai Li, Han Chen. Early Experiences and Challenges in Building and Using a Scalable Display Wall System [J]. IEEE Computer Graphics and Applications. 2000, 20(4):671-680.
[16] 沈兵虎, 潘瑞芳, 金哲凡. 基于保留模式并行绘制的图形对象分布策略[J]. 计算机工程, 2007, 33(5):275-277.
[17] 彭浩宇, 金哲凡, 石教英. 基于保留模式的in-the-core并行超大数据量图形绘制[J]. 软件学报, 2004, 15(增刊):223-231.
[18] Samanta R, Zheng J, Funkhouser T, et al. Load balancing for multi-projector rendering systems [C]. Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware, Los Angeles, 1999:107-116.
[19] Samanta R, Funkhouser T, Li K. Parallel rendering with k-way replication [C]. Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics, San Diego, 2001:75-84.
[20] 金哲凡, 杨建, 石教英. 分布式并行绘制系统中几何指令流压缩的研究与实现[J]. 计算机辅助设计与图形学学报, 2002, 14(9):824-828.
[21] 秦爱红, 熊华, 石教英. 基于集群机并行绘制的三角形条带压缩[J]. 计算机辅助设计与图形学学报, 2007, 19(7):898-906.
[22] Takeuchi, Fumihiko, Hagihara. An Improved Binary-Swap Compositing for Sort-Last Parallel Rendering on Distributed Memory Multiprocessors [J]. Parallel Computing, 2003(29):1745-1762.
[23] 彭浩宇, 金哲凡, 秦爱红等. 复式并行流水线在基于PC集群机的并行绘制中的应用[J]. 计算机辅助设计与图形学学报, 2006, 18(10):1581-1586.
[24] Qin Aihong, Xiong hua, Peng Haoyu, et al. Cluster Parallel Rendering based on Encoded Mesh [J]. Journal of Zhejiang University, 2006, 7(7):1124-1133.
[25] 邓俊辉, 唐泽圣. 基于虚拟机的并行体绘制[J]. 软件学报, 2000, 11(8):1087-1093.
[26] Montrym S, Baum Daniel, Dignam David, et al. InfiniteReality: A real-time graphics system [A]. In: Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, Los Angles, California, 1997, 293-302.
[27] Steven Molnar, John Eyles, John Poulton. PixelFlow: High speed rendering using image composition [J]. Computer Graphics, 1992, 26(2):231-240.
[28] Stanimire Tomov, Robert Bennett, Michael McGuigan, et al. Application of Interactive Parallel Visualization for Commodity-based Clusters Using Visualization APIs [J]. Computer & Graphics, 2004(28):273-278.
[29] 韩伟杰, 张文, 李晓梅. 基于PC机群的并行可视化服务器设计与实现[J]. 系统仿真学报, 2007, 19(S2):394-414.1.

大规模数据密集型系统中的去重查询优化

● 宋怀明　安明远　王洋　孙凝晖
中国科学院计算技术研究所计算机系统结构重点实验室 北京 100190 shm@ncic.ac.cn
● 袁春阳
国家计算机网络应急技术处理协调中心 北京 100029

摘要：

在大规模数据密集型系统中，海量数据分布存储在多节点，给去重查询提出了新的挑战．针对去重查询中可能出现的不同情况，提出了一种有效地数据分布策略和并行处理方法：即散列(hash)和直方图相结合的数据分布策略，以及异步式并行查询引擎，对多节点的去重查询进行优化．异步式并行查询引擎充分发掘了海量数据处理中流水级的并行，消除了多节点同步等待的开销，能够尽早地返回用户结果，降低去重查询的响应时间．在真实系统DBroker上的实验表明，数据分布策略能极大地改善相关属性的去重查询性能，而异步式并行查询引擎能够充分发掘并行性，对不相关属性的去重查询具有明显的性能提升．

关键词：去重，消除重复，海量数据，异步查询，并行查询引擎

1. 引言

近年来，大规模数据密集型系统越来越多的应用在了网络安全监控、金融数据分析、电信数据处理、传感器网络等领域。从上个世纪90年代开始，数据量的增长速度大约每年可以翻一番，远远超过了摩尔定律指出的硬件性能的增长速度。在大规模数据密集型系统中，海量数据通常采用shared-nothing的结构分散存储在多个数据节点，这种方式易于构架和实现，对于提高系统的并行性和扩展性具有明显的优势，但另一方面数据的分散存储也给全局的数据去重查询提出了新的挑战，主要表现在以下几个方面：

- 节点间大量数据交换对网络通讯的压力。
- 大量的中间结果的存储和管理。
- 多节点并行带来的执行倾斜和同步等待的问题。

在大规模数据密集型系统中，一方面需要将海量的数据进行划分和分散存储，以提高数据存取的带宽和计算的并行度；另一方面这种数据划分和并行计算也大大增加了查询执行控制的复杂度。在去重查询处理中，通常采用hash重分区的策略，对扫描的数据重新划分，这种对数据重划分的算法在机群环境下，导致节点之间的大量的数据交换，给网络的通讯带来了巨大的压力，此外查询处理的中间结果也需要在节点间进行传输。查询语句执行计划中涉及到多个步骤，每个步骤均需要多个节点的协同工作，这也给查询计划各步骤之间的流水级的并行和访问控制提出了新的挑战。在查询处理过程中，通常采用的是目前比较成熟的2阶段处理方法[9,12]，即前处理和后处理两个阶段：前处理在各数据库节点独立的查询，后处理对前处理的结果进行合并。在这一过程中，前处理不可避免的会出现执行时间不一致的情况，如果等到所有节点都完成再启动后处理的结果的合并，则可能导致大量系统资源的空闲等待。

本文针对上面提出的3个方面的问题展开了研究，提出了一种有效的数据分布策略，即散列和直方图相结合的数据分布策略，以此来减少去重查询过程中节点之间的数据交换。同时也设计实现了针对海量数据处理的并行查询引擎，充分发掘并利用机群环境下的多机流水并行，提高了系统查询处理的运行效率，并通过异步方式的改进，消除多个并行节点之间的同步等待，提高流水处理的效率，在真实系统DBroker上进行的实验表明，这些方法均达到了较好的效果。

本文的组织结构如下：第2节对数据划分和去重查询的相关研究做一简单介绍；第3节提出了针对shared-nothing结构的海量数据的去重提出了一系列

的优化方法，包括hash和直方图相结合的数据划分策略，并行查询引擎的设计以及异步方式的改进；第4节在实际系统DBroker进行了相关的实验，证明划分策略以及并行查询引擎在海量数据去重查询处理中的有效性；第5节是对全文的一个简单总结。

2. 相关研究

去重(duplication elimination)是数据库中的常见查询操作，目前一般采用的消除重复的算法主要有排序合并算法[1,3,5]和散列(hash)合并算法[2,5]，这些算法在单机数据库中已经发展比较完善。而在机群结构的大规模数据密集型系统中，数据的网络传输和通信开销增加了去重查询处理的难度。文献[4]对多数据源的数据合并问题展开了研究，并提出了一种基于聚类的有效的数据合并方法。焦[12]和王[9]的博士论中均提出并实现了一种通用的2阶段的查询处理方法，这种2阶段的方法能支持大部分查询的处理，但是对于中间结果巨大的去重查询来说，效果并不理想。

在大规模数据密集型系统中，由于数据量巨大，数据一般按照某种策略分布在多个数据库节点。常见的数据分布策略包括范围划分(range)，轮询划分(round-robin)和采用散列函数(hash)的方式[10,11]。数据的划分原则不但要考虑系统常用的查询模式，而且还需要做到各节点数据量的均衡。特别地，针对去重查询，如果查询中去重的属性列和数据划分的属性列相同，则可以大大减小查询时的通信开销。我们将去重属性和被划分的数据属性相同的情况称为相关属性的去重，否则称为不相关属性的去重。一般来说，对属性列的range划分和hash划分对于去重查询均具有较好的效果，本文采用hash分区和等深直方图相结合对数据进行划分，hash的方法大大减少了相关属性去重时各节点之间的通信，而等深直方图的方式能基本保证各节点数据量的均衡。

采用shared-nothing结构的多机数据库系统，无论采用哪种数据分布方式，也无论数据量是多么均衡，都不可避免的会出现各节点执行时间不一致的情况。其主要原因可归于三点：

(1) 节点性能的差异。由于硬件缺陷或者不稳定导致部分节点的执行性能低下，在多节点的机群环境下，不可避免的会出现个别节点的硬件发生老化或者缺陷的情况。

(2) 任务自身的差异。查询任务可能在不同的节点产生的中间结果的数据量不同，带来任务本身的计算量和存取开销在各节点的差异。文献[7,8]均描述了由于各阶段任务划分差异导致执行时间不一致的现象。针对这些中间结果不平衡的情况，文献[8,13]均提出了一种任务重划分的解决办法，但是在大规模数据密集型系统中，任务的重划分会引入大量网络的数据的交换，在中间结果巨大的情况下，这种方法并不合适。

(3) 任务调度的差异。多任务同时运行时，由于各节点调度顺序的差异，导致每个任务在各节点的执行完成时间的不一致，王[9]对这种由于各节点调度顺序的不同带来的执行时间的差异展开了相关的研究。文献[14,15,16]等提出了gang scheduling的调度方式，将一些需要同步进程或线程同时调度执行，能减少同步等待时间，但是如果调度的请求中含有I/O或通信等操作，这些措施的调度效果仍然不太理想，而海量数据处理中I/O却是系统开销的主要方面，因此这些方法依然无法避免调度带来的执行时间的差异。

多节点的执行时间存在差异是不可避免的，焦[12]提出了一种同步区的技术，在2阶段查询处理过程中，后处理需要等到所有数据库节点返回结果再进行结果的合并，同步区的技术放弃某些执行时间过长的节点，以提高整个查询的响应速度。而本文提出的异步式的并行查询引擎没有同步区的概念，也不用等待所有节点都完成前处理才启动最终结果的合并，这种方式在部分节点先返回结果时即可首先启动部分合并的计算，旨在减小执行时间的差异带来的影响。同时异步的执行方式能够尽早的返回用户的部分查询结果，大大减小了用户请求的响应时间，在海量数据处理过程中，极大的提高了查询执行的效率。

针对大规模数据密集型系统中的去重查询的特点，本文提出了组合式的优化方法：将hash分区和直方图相结合的数据划分策略，减少节点间的通讯开销；异步式的并行查询引擎尽快启动后处理的结果合并。这两种方式的结合，对于相关属性的去重和不相关属性均有较大的性能提升，特别是在中间结果巨大的去重查询中，效果更为明显。

3. 系统设计和实现

3.1 数据划分

消除重复常用的方法有排序方法和散列方法，对于数据量特别大的情况，一般采用散列函数的方法，将数据分成多个散列桶(bucket)，然后在各桶内用同样的方法消除重复，这是一个递归的过程。如果已知用户的去重列，则可以在数据写入系统时对这个去重的属性值进行散列划分，以节省查询时划分的开销。

本文采用的散列划分的函数是取模的运算，假设数据分布的节点数为n，取一个相对较大的数m(m>>n),将记录的R的属性值col_a与m取模Mod (R.col_a,

m), 则可能得到 0~m-1, 这m种可能结果，对每个散列的结果的记录数进行统计，然后按照等深直方图的方法，将这每个结果分成n份，分成的结果如下：

$S=\{S_1(0,x_1), S_2(x_1,x_2), \cdots, S_n(x_{n-1},m-1)\}$

其中$0\leq x_1\leq x_2\cdots\leq x_{n-1}\leq m-1$，$S_i(i=1\sim n)$表示分成的散列集合，每个集合包括一组数值，一个集合对应一个数据库节点，表示通过散列得到的结果如果在该范围，则将记录存放在对应的节点上。其中m值远大于节点数的原因在于细化散列的结果，避免生成偏斜的等深直方图。采用hash的方式对数据进行划分，保证了去重查询中节点之间记录的无关性，各节点返回的结果就是最终结果的一部分，无需再进行合并，可以有效的降低查询的响应时间；而采用等深直方图的方式，则能基本保证各数据库节点的数据量均衡。由于数据的划分方法在记录写入各节点时必须是明确的，所以该方法需要建立在对历史数据统计分析的基础上。

3.2 并行查询引擎

在shared-nothing结构的大规模数据密集型系统中，hash的方式能有效减小节点之间的通信，提高了相关属性的去重查询的性能，但是对于非相关属性的去重查询，hash的方式并不能带来真正的好处。本文引入了并行查询引擎的设计，旨在减小查询的响应时间，同时也能提高海量数据的去重查询性能。并行查询引擎充分发掘多节点之间的流水级并行，其结构和处理流程分别如图1和图2所示。

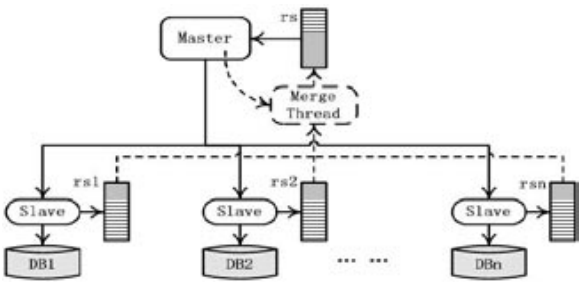


图1 并行查询引擎结构图

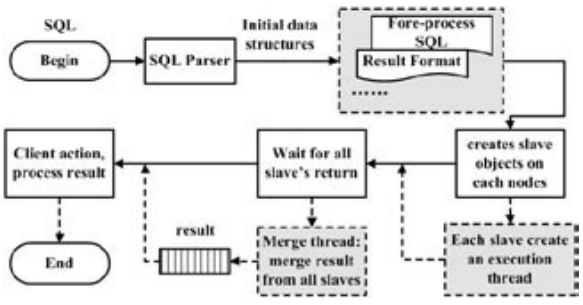


图2 并行查询引擎处理流程

并行查询引擎包括一个master线程和一组slave线程，其中master是查询的控制线程，完成查询语句的分析和下发，并对最终结果进行合并；slave线程则在各数据库节点运行前处理查询语句，并将结果传送给master。其处理流程主要是：master接收到用户的查询请求，首先对查询语句进行分析和分解，分成前处理语句和后处理的合并计划，然后将前处理语句分发给各个slave，每个slave在一个数据库节点执行查询，此时master处于等待状态，当所有slave完成并返回结果时，master根据合并计划启动多个线程对slave返回的结果进行合并，并将合并后的结果返回给客户端。

并行查询引擎具有良好的流水并行性，各slave返回结果和master的结果合并可以同时进行，对海量数据的去重查询具有较好的效果，这是由于海量数据的去重查询会产生大量的中间结果，流水的处理方式减低了处理过程中内存的需求。一个简单的去重查询的例子如下：在网络安全监控系统中，对各出入口上满足某种条件的事件的源地址的去重数量统计select count(distinct src_ip), inout_id from event_base where … group by inout_id; 只需要将发送给各数据库的前处理语句加上对出入口属性的排序，即前处理语句改写成select src_ip, inout_id from event_base where … group by inout_id, src_ip order by inout_id. 在各节点的查询中，数据库内部的算法从 hash group变成了sort group，算法效率上并没有太大的差异，但是这种改变却能够使得后处理的结果合并能够流水的进行并同时输出结果。Master不需要保存大量的中间结果，节约了运行时的存储资源，这种流水的处理方式大大提高了去重查询的效率。

3.3 异步方式的改进

在shared-nothing的大规模数据库系统中，各节点执行时间的差异不可避免，而同步的等待方式则会带来大量的系统资源的浪费。异步的方式正是针对各节点执行时间差异进行的改进，防止流水线在启动或者运行过程中过多的同步等待。Master不必等到所有的节点都返回结果再进行最终结果的合并，同时在流水的结果合并中，也不必等到所有节点的相关数据都返回才启动合并计算，master线程只要有部分节点返回数据就可以首先启动合并计算，这种方式叫异步的结果集合并。

异步的方式对slave返回和master合并结果进行了改进，采用hash的方式对slave返回的结果进行重划分，这样每个slave可以独立的将返回的结果放入hash分区结构，而不必关心其他slave的状态。Master在进行最终结果的合并时，只要有合并的计算

能够进行则进行计算，而不必同步等待所有的相关结果均返回。由于各slave返回的数据是有序的，因此对于每个分组，只要每个slave都完成了该分组数据的返回，则可以尽快输出该分组合并后的结果，释放相应的空间。

表1 异步并行查询引擎的算法简要描述

```
Slave线程读取结果并分发
Fetch_deliver(){
    …
    while ( !rs->eod()) {              // 未到结尾
      hash_map.add  (rs.record()); //  结果放入hash队列
      cur_top ();    // 修改当前结果标志
      rs->next();    // 取下一条结果
    }
}
Master线程合并最终结果
Merge(){
    …
    while (! hash_map.empty() ||
      ! rsv->eod()){   // 当hash_map不为空或结果未取完
      cur = get_top(); // 获取当前的结果集顶端
      count_dinstinct(cur);  //  计算所有的小于顶端的分组
                             // 的去重统计结果
    }
}
```

算法的简要描述如表1所示，其中每个slave线程顺序读取结果集并写入hash_map，该结构是一个多维的hash表，数据写入时自动的完成hash划分，slave随后检查或修改当前分组的信息；master线程则循环的将排序在当前分组之前的所有的数据从hash_map中读取并合并，将结果尽快输出，直到所有的结果均处理完成．在整个过程中，master线程和多个slave线程是并行运行，相互独立的.

在机群环境下，去重查询的任务首先被分配到所有数据库节点，不可避免的出现各节点执行时间不一致的情况，此外在读取各节点的执行结果时，也不可避免的出现各节点不同步的情况．异步的方式是对并行查询引擎后处理中读取结果和结果合并进行的优化，消除了节点之间的同步等待，提高了系统资源的利用率和查询处理的效率.

4. 实验结果

4.1 实验环境介绍

实验是在真实系统DBroker上进行的，该系统是构架在Dawning 4000L机群服务器上的大规模数据密集型的应用系统。DBroker底层的数据库采用的shared-nothing的方式，其节点配置为（4*2.2G CPU, 32GB Memory, 4TB Raid5, 1Gb Ethernet, Linux AS4, Oracle 10.2），查询服务器的节点配置为（4*2.2G CPU, 32GB Memory, 10Gb Ethernet, Linux AS4）。实验中共采用了16个数据库节点和1个查询服务器节点，并行查询引擎运行在查询服务器节点。测试的数据集采用了863-917国家网络安全监测平台的仿真数据，测试语句形式为select count(distinct src_ip), count(distinct dst_ip), inout_id, dev_id from event_base where … group by inout_id, dev_id; 该类查询表示对源IP和目的IP地址按照出入口和设备号进行去重计数统计。

4.2 性能对比

图3是在不同数据量下对各种数据分布策略执行时间的一个比较，其中round-robin的方式为轮询的数据分布，hash-1是指采用hash分区的不相关属性的去重查询，hash-2是指相关属性的去重。

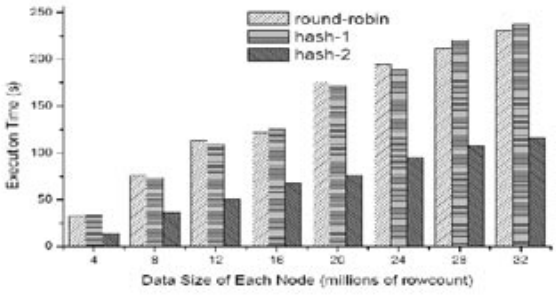


图3 不同分布策略执行时间的对比

测试中均采用异步并行查询引擎进行查询处理，测试的节点数为16，每个节点数据量为从400万增加到3200万条记录，从图中可以看出，hash分布和等深直方图相结合的方式，能够大大减低相关属性的去重的响应时间，而对于不相关属性的去重和轮询方式的差别不明显。对于相关属性的去重，hash方式的数据分布不需要进行后处理，因此对查询性能的提升是非常明显的，另一方面这种分布策略也大大减小了处理过程中网络的通信量。

图4和图5的测试保持每个节点的数据量不变(1200万条记录)，节点数从2增加到16的情况下，并行查询引擎和2阶段查询方法的执行时间的对比。其中图4是执行总时间的对比，图5是每个阶段执行时间的比较，图中pqe表示并行查询引擎的方法，pqe-fore和pqe-post分别表示前后处理的时间，2step表示

传统的2阶段的处理方式，2step-fore和2step-post分别表示前后处理的时间。

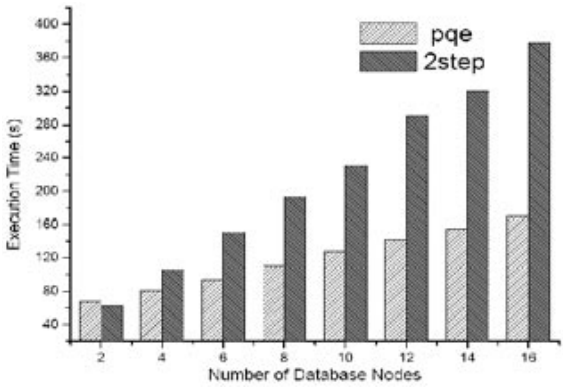


图4 不同节点数下的执行时间对比

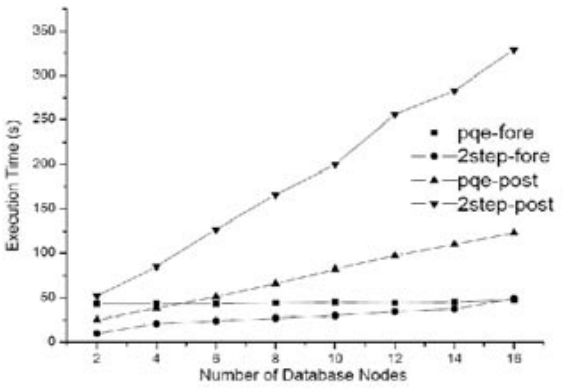


图5 不同节点数下各阶段的执行时间对比

从测试结果可以看出，无论从总的执行时间，还是从首条结果返回时间来看，并行查询引擎的效果都要比2阶段的执行方法更好。并行查询引擎的方式在前处理执行完成即可流水地读取各节点的结果和进行结果的合并，同时流水地输出合并的结果，而2step的方式只有全部执行完成才能返回结果，更加大了两者性能的差距。此外在每个节点数据量一定的情况下，节点数的增加对于并行查询引擎来说，其前处理的时间是相对固定的，即首条结果输出的时间相对稳定，而2阶段的方式其前处理的时间是随着节点数增加而增加的，因此并行查询引擎具有更好的节点数量的可扩展性。

图6和图7的测试保持数据库节点数不变（16节点），每个节点的数据量从400万条记录逐步增加到3200万条记录的情况下，两种方法执行时间的变化情况。从图6的测试结果可以看出，并行查询引擎的性能无论在小数据量还是大数据量的情况下，均优于2阶段的查询方法。而从图7的各阶段的执行时间来看，在数据量较小的时候，两种方式前处理的时间差异不大，而在数据量较大的情况下，并行查询引擎的性能更好，因为这种方法前处理不需要将查询的结果全都写入查询服务器节点，而是在后处理中流水的读取并进行结果的合并。

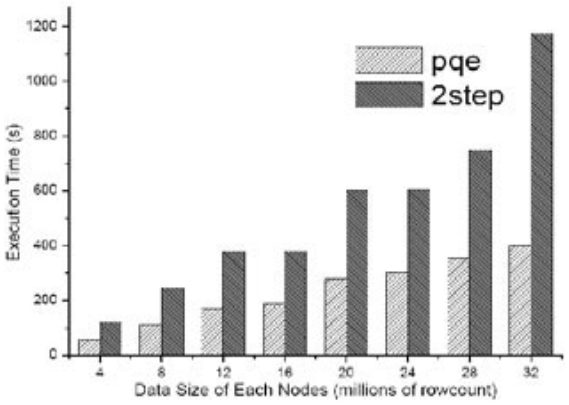


图6 不同数据量下的执行时间对比

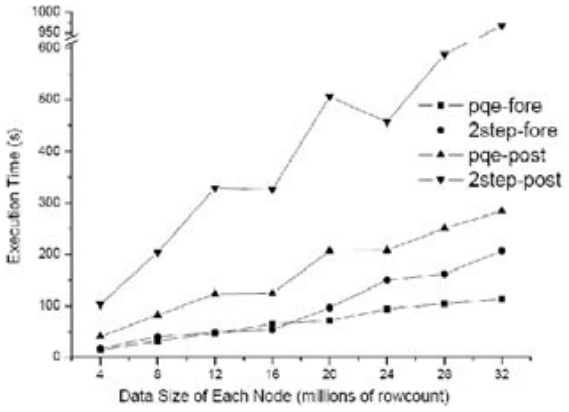


图7 不同数据量下各阶段的执行时间对比

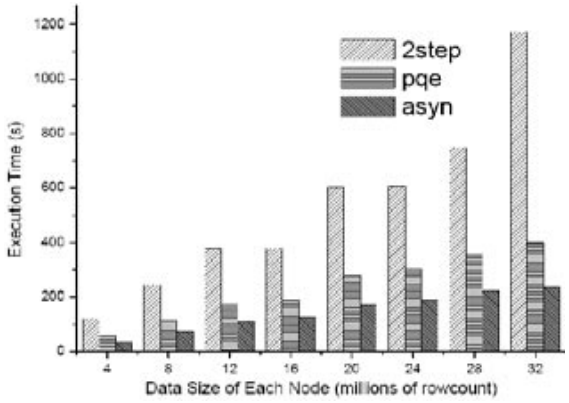


图8 不同数据量下各方法执行时间对比

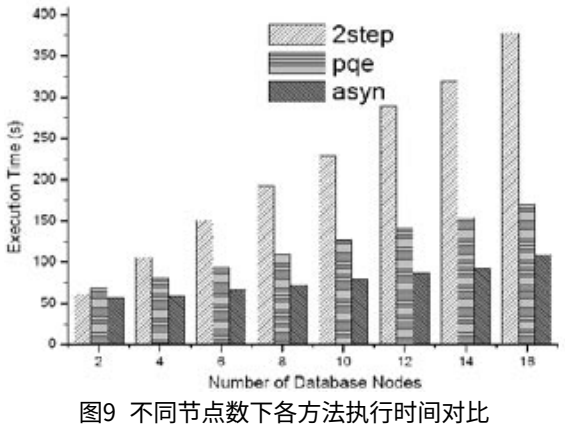


图9 不同节点数下各方法执行时间对比

图8和图9是测试异步方式的改进对于并行查询引擎的影响，其中asyn表示异步方式的并行查询引擎．图8的测试保持节点数不变(16节点)，每个节点的记录数从400万逐步增加到3200万；图9保持每个节点的数据量不变(1200万记录数)，数据库节点数从2个逐步增加到16个．从测试结果可以看出，异步的方式进一步提高了并行查询引擎的性能，在节点数逐步增加的情况下，相比同步的方式，查询性能提升效果更加显著．这是因为随着节点数的增加，各节点之间的同步等待开销所占的比例增大，而异步的方式各节点不必同步的读取结果集，避免了节点间相互等待的开销，因此具有更好的扩展性。

4.3 讨论

采用hash的方式对数据进行划分，保证了相同属性值的数据存储在同一个节点，在相关属性的去重查询中，只需在各节点进行去重查询，而不必对前处理结果再进行合并，节省了处理步骤，对于性能的提升是明显的。而等深直方图的方式则是在统计的基础上保证各节点数据量均衡的一个有效手段。Hash和直方图相结合的分布方式需要具有数据属性值统计的先验知识，对于统计特性动态变化的系统，还必须增加分布策略动态调整的机制，这将在一定程度上增加了系统的复杂性。

并行查询引擎是机群环境下的大规模数据库系统的查询中间件，将2阶段的查询处理分解成多机协同的流水处理过程，并充分挖掘流水线各步骤间的并行性，较好的达到对海量数据去重查询的性能要求。异步方式的改进更加提高了查询结果合并的速度，减小了节点之间同步等待的开销，提高流水线的资源利用率，和同步的方法相比，具有明显的性能提升。这种基于中间件的结构将查询处理分解成前后处理2个步骤，前处理过程能够较好的利用现有数据库的查询处理机制，而不必考虑其内部具体的算法，简化了系统的设计和实现。

但另一方面，基于中间件的实现也存在一定的局限性。由于各节点采用了关系数据库进行数据存取和查询处理，使得无法在更细粒度的层面对处理过程进行优化，只能借助于数据库内部已有的一些优化的手段，这也在一定程度上限制某些具体的优化方法的实现。

5. 结论和进一步研究

海量数据的去重已成为大规模数据密集型系统的一个关键应用。在基于shared-nothing结构的大规模数据密集型系统中，如何减少去重查询中节点间的通讯开销和提高多节点的计算并行度成为提高查询性能的主要切入点。散列和等深直方图相结合的数据分布策略，在减小节点间通讯量的同时，也能保证节点间的数据量均衡，大大提高了相关属性去重查询的性能。并行查询引擎的设计通过流水线的方式降低了查询请求的响应时间，而异步的方式在此基础上进一步消除了节点间的同步等待的开销，对于非相关属性的去重查询，具有较好的效果。实验表明，这些方法能够大幅提高查询的性能。此外，这种数据分布策略和异步式的并行查询引擎对于shared-nothing结构上的其它类型的查询，如分组查询(group)和连接查询(join)，同样具有较好的效果。

并行查询引擎的流水线的处理方式能够支持对分组属性排序结果的直接输出，但是对于去重属性的聚集值的排序，却不能直接输出结果，需要对流水线的输出结果进一步进行处理。在下一步的工作中，将对去重属性聚集值的排序问题展开研究，以充分发挥机群环境的计算并行度。

参考文献：

[1] Dina Bitton, David J. Dewitt. Duplication Record Elimination in Large Data Files. ACM Transaction on Database Systems, Vol.8, No.2, pages: 255-265, June 1983

[2] Goetz Graefe, Richard L. Cole. Fast Algorithms for Universal Quantification in Large Databases. ACM Transactions on Database Systems, Vol.20 No.2, pages:187-236, June, 1995

[3] Xiaoyu Wang, Mitch Cherniack. Avoid Sorting and Grouping In Processing Queries. In: Proc of the 29 VLDB conference. Berlin, Germany, 2003

[4] Mauricio A. Hernandez, Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. ACM SIGMOD Record, Vol.24 Issue 2, May 1995, 127-138

[5] J. Claussen, A. Kemper, D. Kossmann, C. Wiesner. Exploiting early sorting and early partitioning for decision support query processing. The VLDB Journal(2000)9: 190-213, 2000

[6] Alfons Kemper, Christian Wiesner. HyperQueries: Dynamic Distributed Query Processing on the Internet. Proc. of the 27th

VLDB Conference, September 2001
[7] T. Mayr, P. Bonnet, J. Gehrke, P. Seshadri. Leveraging Non-Uniform Resources for Parallel Query Processing. Proc. of CCGRID ’03, page 120, 2003
[8] Lionel Brunie, Harald Kosch. Control strategies for complex elational query processing in shared nothing systems. SIGMOD Record, Vol.25, No.3, September 1996
[9] Wang Yong. Workload and Query Scheduling of Event Stream Analysis: [PhD thesis]. Beijing, Institute of Computing Technology, Chinese Academy of Sciences, 2008（王勇. 事件流应用的负载特征分析和查询调度[博士论文]. 北京,中科院计算所,2008）
[10] Manish Mehta, David DeWitt. Data placement in shared-nothing parallel database systems. The VLDB Journal(1997)6:53-72, 1997
[11] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. Communications of ACM, Vol.35, Issue 6, June 1992, pages: 85-98.
[12] Jiao Limei, On Design Research of Event-Stream Oriented Cluster Database[Ph.D Thesis], Beijing, Institute of Computing Technology, Chinese Academy of Sciences, 2007 (焦丽梅. 面向事件流机群数据库系统的设计研究[博士论文]. 北京,中科院计算所, 2007)
[13] Ambuj Shatdal and Jeffrey F. Naughton. Adaptive parallel aggregation algorithms. ACM SIGMOD Record 1995, Vol.24, Issue 2, pages: 104-114, May 1995.
[14] D.G. Feitelson and L. Rudolph. Gang Scheduling Performance Benefits for Fine-Grain Synchronization. Parallel and Distributed Computing, Vol.16, No.4, pages: 306-318, 1992.
[15] Dror G. Feitelson, Packing Schemes for Gang Scheduling, Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, pages.89-110, April 16, 1996.
[16] Yair Wiseman and Dror G. Feitelson, Paired Gang Scheduling, IEEE Transactions on Parallel and Distributed Systems, Vol.14, No.6, p.581-592, June 2003.

要闻集锦

AMD计划用GPU制造千万亿次超级计算机

据www.hpcwire.com网站2009年1月8日消息报道，美国AMD公司在近日举行的2009国际消费电子展上表示，该公司计划在2009年底使用超过1000块Radeon HD 4870显卡来组建一部超级计算机：Fusion Render Cloud，它将带来每秒钟1千万亿次（petaflops）的浮点运算能力和100万条线程的处理能力。AMD表示，这台超级计算机将主要运用于云计算领域，还将给游戏玩家和3D动画工作人员带来强有力的在线服务。

AMD公司CEO Dirk Meyer表示：“在当前的经济形势下，计算机用户需要的不再是‘更大’或‘更快’的技术，而是可提供更高价值、更强大的交互功能和接入能力的技术。今天，AMD展示了一系列基于我们的Fusion平台的先进技术。它们可带来更丰富、更具交互性的视觉体验。Fusion平台的目标是帮助AMD充分利用其在图形处理和微处理器技术领域的核心优势，在视觉革命中发挥更大的作用。而这一革命，无论在家中、工作还是娱乐中，都将能够从根本上改变消费者的计算体验。”

卢斯卡影业有限公司IT运营总监 Kevin Clark 表示：“作为引领数字特效领域发展潮流的电影公司，卢斯卡影业一直对如何更有效地利用平台功能非常感兴趣。我们在5年前选择了AMD，因为我们与他们的合作非常愉快，而且他们的平台有着明显的优势。我们期待能够一如既往的密切合作，提高技术及制片方面的标准。”

（李 苏）

基于共享内存的机群服务检查点机制研究

- 梁 毅 北京工业大学计算机学院 北京 100022 yliang@bjut.edu.cn
- 王 磊 中国科学院计算技术研究所 北京 100190
- 樊建平 中国科学院深圳先进技术研究院 深圳 518054
- 詹剑锋 中国科学院计算技术研究所 北京 100190

摘要：

针对既有基于稳定存储的机群服务检查点存在的系统成本高、恢复时间长的问题，提出了一种基于共享内存的机群服务检查点机制，设计了一套面向基于共享内存的检查点信息主-备存储模式的的检查点信息管理协议，确保机群服务检查点信息一致性；设计了一套基于单向逻辑环的检查点组管理协议，确保检查点逻辑备份环中检查点进程的成员视图一致性。性能试验结果表明，该检查点机制具有较好的检查点信息读写性能，组管理协议系统开销小，较好的满足了机群服务检查点需求。

关键字：机群服务，检查点，共享内存，组管理

机群服务，如机群作业调度、系统管理等多为有状态服务[1]，对于这类服务，实现其高可用不仅需要确保服务进程在失效后快速重启，还要需要实现服务状态的快速可靠恢复。机群服务检查点机制是实现服务状态快速可靠恢复的重要途径之一。机群服务检查点机制设计的核心目标是1）检查点信息的高效读写，提高机群服务的可用性；2）检查点信息的一致性，确保机群服务失效后能够根据检查点信息可靠地恢复服务状态。所谓检查点信息一致性是指在任一时刻机群服务所获取的检查点信息的更新版本应不小于机群服务所感知的检查点信息更新版本[1,2]。

既有面向机群服务的检查点机制通常基于稳定存储介质，通过数据库、文件系统来实现机群服务检查点信息的维护。该类方案着眼于检查点信息的可靠存储，但是上述方案存在以下缺点：1）系统成本高，为了保证数据库、文件系统的高可用，需要采用双机高可用环境和RAID设备，会增加系统的成本；3）恢复时间较长，都需要对于磁盘等稳定存储介质的读取，读取的开销较大[7,14,15]。

本文针对上述缺点，提出了一种基于共享内存的机群服务检查点机制，该机制通过基于共享内存的检查点信息读写提高了检查点的执行效率；并基于单向逻辑环的检查点备份结构实现检查点信息的维护，从而降低了共享内存作为非稳定存储介质进行检查点信息维护带来的风险。本文针对基于共享内存的机群服务检查点机制设计的关键问题展开研究，主要贡献为：

1）对机群服务检查点信息模型进行了定义，并设计了一套检查点信息管理协议，该协议面向基于共享内存的检查点信息主-备多存储的模式，保证了机群服务检查点信息的一致性；

2）将维护基于单向逻辑环的检查点备份结构抽象为组管理问题，设计了一套基于单向逻辑环的检查点组管理协议，该协议保证了逻辑备份环中检查点进程的成员视图一致性。

本文的组织结构如下：第一节给出了相关工作；第二节提出了基于共享内存的机群服务检查点机制的基本思想和关键问题；第三、四节分别针对关键问题对检查点信息管理协议和检查点进程组管理协议进行了详细阐述和分析；第五节给出了实验分析和结果。第六节给出了本文结论。

1. 相关工作

检查点按应用领域分为：科学计算领域和非科学计算领域，科学计算领域多为针对并行程序的检查点，非科学计算领域主要针对商业计算应用或是机群服务；科学计算的检查点主要关注恢复时多节

点上的多进程间全局一致状态获取效率和检查点记录开销[5]；非科学计算的检查点主要关注检查点恢复时信息更新的一致性[2]。检查点按纪录介质分为：稳定存储检查点和无盘（非稳定存储）检查点，检查点按实现层次和对于用户是否透明分为：系统级检查点和应用级检查点。

对于科学计算应用，检查点在不同介质和不同层次上都有相关实现。[3]是一个基于数据库实现的应用级检查点,采用关系数据库系统来代替以前采用文件的方式来存储机群系统的检查点,便于数据的索引与归一化，[4]是一个基于内存的系统级检查点，它将Score机群系统的检查点由基于磁盘读写改进为基于内存的读写，解决了基于磁盘读写的速度瓶颈问题。

对于非科学计算应用，由于非科学计算应用的程序大多采用多进程/线程技术，要实现系统级检查点非常困难，因此非科学计算通常基于稳定存储介质，通过数据库、文件系统等实现应用级检查点[2，6]。[7]是一个机群的商业应用管理工具，它可以管理机群中部署的商业应用，通过资源分配、负载平衡、失效处理等手段来保证应用的服务质量。它的检查点通过一个共享数据库实现。[13]是一个符合CCA架构的应用框架，他为符合CCA标准的应用提供多种基础服务，它的检查点是基于XML格式的文件实现。[14]是在大规模数据中心中实现一个基于模型方法的服务自动部署工具。它的检查点通过日志方式的记录在文件系统中。

2. 基于共享内存的机群服务检查点机制设计思路

根据机群服务检查点机制设计的核心目标，基于共享内存的机群服务检查点机制的主要设计思路是：

1）基于共享内存的检查点信息读写

在每个机群服务所在节点上启动一个检查点进程，维护一块或多块机群服务和检查点进程都可以访问的共享内存；在此节点上运行的服务将其关键信息保存在该共享内存中，并在机群服务进程失效重启后，能够迅速通过检查点进程读出该服务的检查点信息，恢复服务状态[8]。

2）基于单向逻辑环的检查点备份结构

将机群内所有检查点进程组成一个单向逻辑环结构，在该逻辑环中，检查点进程按照顺时针方向，向其前继结点备份检查点信息，当检查点进程失效重新启动后，检查点进程可根据在逻辑环中的备份信息，恢复失效前的所维护的服务检查点信息，从而支持服务失效重启。

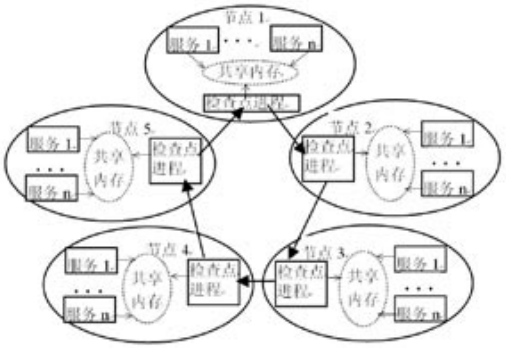


图1 基于5个节点的机群服务检查点逻辑结构

图1给出了基于5个节点的基于共享内存的机群服务检查点逻辑结构图。节点上的检查点进程根据节点间箭头的顺序进行检查点信息的单向备份。检查点进程1将检查点信息备份到检查点进程2,2将检查点信息备份到3,以此类推。

基于共享内存的机群服务检查点机制设计的核心思想是通过基于共享内存访问提高检查点的执行效率，并通过检查点间的备份结构降低内存作为非稳定存储带来的检查点信息维护可靠性风险。根据机群可靠性理论，短时间内出现两次或更多次故障的概率近似为零[1]。因此，相邻节点单向备份的方式较好的确保了在任一时刻，系统中至少存在机群检查点信息的一个副本。

由上述设计思路可知，与基于稳定存储的检查点机制不同，基于共享内存的检查点机制将同一机群服务的检查点信息存放在服务所在节点（主节点）和其备份节点上，并基于逻辑备份环确保检查点信息的可靠维护。因此，基于共享内存的机群服务检查点设计的关键问题包括：

1）如何面向基于共享内存的检查点信息主-备多存储的模式，保证了机群服务检查点信息的一致性。

2）如何有效实现检查点进程单向逻辑备份环的维护，从而确保检查点信息的可靠备份和恢复。

针对上述关键问题本文分别设计了检查点信息管理协议和基于单向逻辑环的检查点组管理协议。

3. 检查点信息管理协议

本节针对基于共享内存的机群检查点机制设计的关键问题1），首先给出了基于共享内存的机群服务检查点信息的组织模型，并基于该模型，设计了基于主备一致的检查点信息管理协议，确保检查点信息一致性[16]。

3.1 机群服务检查点信息组织模型

任一检查点共享内存可存放运行于本节点的多

个机群服务的检查点信息，并可按照检查点环形备份结构，存放其前继结点的检查点信息。在检查点共享内存中，检查点信息按“块”组织，一个机群服务的检查点信息可存放于一个连续的共享内存块中。对于任一机群服务，其检查点信息可用以下模型描述：

CSckinfo = (CSid, SMbckentr, size, ver)；

其中，Csid为该机群服务id；SMbckentr表示存放该服务检查点信息的共享内存块的访问入口，size为该共享内存块的大小，ver为该服务检查点信息的更新版本（初值为0）。

值得注意的是，对于任一机群服务其Csid是全局唯一的，且在其失效重起后，该Csid将保持不变。对于任一节点的检查点共享内存将在检查点进程启动时被重置清空。

3.2 检查点信息管理协议

为了确保检查点信息一致性，该协议需对任一检查点信息发生更新的情况进行处理。检查点信息更新包括检查点数据更新和检查点存储位置的更新。存储位置更新包括检查点信息备份节点变更、机群服务所在节点（主节点）变更。数据更新则包括对检查点数据的各类修改，如内存块创建/删除、检查点数据更新写入等。检查点信息管理协议应涵盖对上述情况的处理。

检查点信息备份节点变更协议

检查点信息备份节点变更的时机包括检查点环形备份结构中有节点加入、退出或失效，导致检查点进程间的备份关系发生改变。处理协议如下：

(1) 机群服务所在节点（主节点）的检查点进程根据环形备份结构，向其新的备份节点发送服务检查点信息及检查点描述信息备份请求；若发送请求不成功，则操作失败；若发送成功，则继续。

(2) 新的备份节点的检查点进程接收备份请求，若备份节点已存在该机群服务检查点信息，则首先释放该服务所属的共享内存块，删除该服务的检查点描述信息；然后根据备份请求，在该节点的检查点共享内存中分配相应的内存块，存放备份检查点信息，并根据接收的检查点描述模型，在备份节点创建该服务的检查点描述信息，然后向主节点的检查点进程返回备份成功；

(3) 主节点的检查点进程接收备份节点的返回信息，若备份不成功，则检查点备份节点变更处理失败，否则继续；

(4) 主节点的检查点进程根据环形备份结构判断，若原备份节点未失效，则向原备份节点发送删除该服务检查点信息的请求；否则备份节点变更处理成功；

(5) 原备份节点接受删除请求，释放该服务所属的共享内存块，删除该服务的检查点描述信息；然后向主节点的检查点进程返回成功信息；

(6) 主节点检查点进程接收原备份节点返回后，备份节点变更处理成功。

机群服务所在节点变更协议

机群服务所在节点变更的时机是机群服务所在节点失效后选择新的节点进行服务恢复或是新的机群服务启动。当发生节点变更时，新选择的节点上的检查点进程将发起“检查点信息迁移”操作，描述如下：

(1) 新选择的机群服务所在节点（主节点）的检查点进程根据检查点环形备份结构，向该备份环中所有检查点进程广播机群服务id(CSid)，查找该服务所属的检查点内存块；若广播不成功，则操作失败，否则继续；

(2) 环形备份环中的各检查点进程接收查找请求，根据服务id，查找本节点的检查点共享内存，若存在该服务所属的内存块集,则返回该服务的检查点描述信息；否则返回查找结果为空消息；

(3) 主节点的检查点进程接收各检查点进程的返回消息，若所有返回消息均为空消息，则该服务为新增机群服务，主节点检查点进程为该服务创建检查点描述信息，设置更新版本为0；

(4) 若所返回的消息存在非空消息，则选取其返回的服务检查点描述信息更新版本ver最大的检查点进程，获取其上的该服务检查点信息；在主节点的检查点共享内存中，分配内存块，复制相应的检查点信息；并复制该服务检查点描述信息（内存块访问入口信息将根据本节点共享内存分配重新设置），若操作不成功，则信息迁移失败；否则继续；

(5) 主节点检查点进程发起“检查点信息备份节点变更”处理，若变更处理失败，则信息迁移失败，否则继续；

(6) 主节点向返回机群服务检查点信息的检查点进程，发送删除该机群服务所属检查点内存块信息的请求；

(7) 返回机群服务检查点信息的检查点进程释放该服务所属的内存块，并删除其描述信息；

(8) 主节点接收删除请求返回后，数据迁移处理成功。

检查点数据更新协议

(1) 机群服务向所在节点（主节点）的检查点进程发送更新的检查点信息及相关描述，若发送不成

功，则操作失败；若发送成功，则继续。

⑵ 主节点的检查点进程根据服务 id (CSid) 在该节点的检查点描述信息中查找属于该服务的描述信息，若未查找到，则执行“检查点信息迁移”操作；若执行不成功，则更新失败，否则继续；

⑶ 主节点检查点进程从本节点检查点描述信息中获取该服务检查点信息的更新版本号ver_h，并根据环形备份结构，将ver_h与需更新的检查点信息及其相关描述向其备份节点检查点进程发送，若发送失败，则更新失败；

⑷ 备份节点检查点进程接收请求，根据更新信息，对备份节点上盖服务的检查点信息进行更新，然后对备份节点上该服务的检查点描述信息进行相应更新，并将该节点上的服务检查点信息的更新版本号ver_b设置为$\max\{ver_b', ver_h\}+1$，其中，ver_b'为更新前该节点上服务检查点信息的更新版本号。检查点向主节点检查点进程发送更新成功信息，并将verb返回；

⑸ 主节点检查点进程若收到备份节点执行不成功返回，则数据更新失败，若接收到执行成功返回，则根据更新信息，对主节点上盖服务的检查点信息进行更新，然后对主节点上该服务的检查点描述信息进行相应更新，并设置检查点信息的更新版本号为verb；

⑹ 主节点检查点进程向机群服务返回更新成功。

检查点信息读取协议

⑴ 机群服务向所在节点（主节点）的检查点进程发送检查点信息读取请求，若发送不成功，则读取失败；若发送成功，则继续。

⑵ 主节点的检查点进程根据服务id (CSid)在该节点的检查点描述信息中查找属于该服务的描述信息，若未查找到，则执行“检查点信息迁移”操作；若执行不成功，则读取失败，否则继续；

⑶ 主节点的检查点进程根据服务id (CSid)在该节点的检查点描述信息中查找属于该服务的描述信息，若未查找到，则读取失败；若查找到，则根据该服务检查点信息的共享内存块访问入口读取检查点信息，并向服务返回该检查点信息。

3.3 协议分析

结论：检查点信息管理协议满足检查点信息更新一致性。

证明：令

$$ver_i(CS\_w), ver_i(CS\_r), ver_i(Local), ver_i(Backup)$$

分别为t_i时刻机群服务所感知的检查点信息更新版本、机群服务所读取的检查点信息更新版本、机群服务所在节点（主节点）的检查点进程所维护的检查点信息更新版本、备份检查点进程所维护的检查点信息更新版本。则满足检查点信息一致性实质上是对于任意时刻t_i,满足

$$ver_i(CS\_w) \le ver_i(CS\_r)$$

同时，由检查点数据读取协议描述可知：对于任意时刻t_i,满足

$$ver_i(CS\_w) = ver_i(Local)$$

以下用归纳法证明：

（1）首先由协议描述可知，在服务检查点信息的初始状态（即未进行任何变更操作）不妨设为t_0时刻，满足

$$ver_{t_0}(CS\_w) = ver_{t_0}(Local) = ver_{t_0}(Backup) = 0$$

因此可知，t_0时刻满足

$$ver_{t_0}(CS\_w) = ver_{t_0}(CS\_r) = 0$$

（2）不妨令任意两个相邻的时刻t_{i-1}与t_i间能且仅能发生一次检查点信息的变更操作。

设$t=t_{i-1}$时刻满足

$$ver_{t_{i-1}}(CS\_w) \le ver_{t_{i-1}}(CS\_r)$$

则

1）若在t_{i-1}与t_i时刻间未发生任何的检查点信息变更操作，则各更新版本均未发生变化，因此满足：

$$ver_{t_i}(CS\_w) \le ver_{t_i}(CS\_r)$$

2）若在t_{i-1}与t_i时刻间发生了检查点数据变更操作，由协议描述可知，对于任意检查点数据变更操作均满足备份节点的信息更新先于主节点的信息更新；主节点的信息更新先于服务所感知的信息更新，可得

$$ver_{t_i}(CS\_w) \le ver_{t_i}(Local) \le ver_{t_i}(Backup)$$

因此满足

$$ver_{t_i}(CS\_w) \le ver_{t_i}(CS\_r)$$

3）若在t_{i-1}与t_i时刻间发生了检查点信息存储位置变更操作（备份节点变更、主节点变更）由协议描述可知，该变更不由机群服务发起，因此，

$$ver_{t_i}(CS\_w) = ver_{t_{i-1}}(CS\_w)$$

若执行备份节点变更操作，则有

$$ver_{t_i}(Local) = ver_{t_i}(Backup) = ver_{t_{i-1}}(Local)$$

因此满足

$$ver_{t_i}(CS\_w) \le ver_{t_i}(CS\_r)$$

若执行主节点变更操作，则有

$$ver_{t_i}(Local) = ver_{t_i}(Backup) = ver_{t_{i-1}}(Backup)$$

总结上述分析可知,任意检查点信息变更后均有

$$ver_{t_{i-1}}(Local) \le ver_{t_{i-1}}(Backup)$$

因此满足

$$ver_{t_i}(CS\_w) \le ver_{t_i}(CS\_r)$$

总结1)，2)，3)可知，在t_i时刻可以满足

则由(1)(2)可知，对于任意时刻t_i,满足

$$ver_i(CS\_w) \le ver_i(CS\_r)$$

因此，该协议满足检查点信息一致性。

证毕

4. 基于单向逻辑环的检查点组管理协议

基于单向逻辑环的检查点备份结构是基于共享内存的机群服务检查点机制中实现检查点信息可靠维护的关键。只有在组成逻辑环的各检查点进程间维护关于逻辑环组成成员状态及成员间逻辑顺序关系的一致视图，才能确保检查点进程在执行检查点信息存储位置更新及数据更新等操作时，能够正确感知相关检查点进程的位置和状态。

在组成逻辑环的各检查点进程间维护一致视图的问题实质上可以被抽象为分布式系统中组成员管理问题。

定义1 组成员管理（Group membership） 组成员管理指基于同一应用的一组进程组成一个组（group）。在这样的一个组中，当成员加入、退出或失效时，每个成员间要始终保持统一、一致的成员状态视图[8,9,10]。

组成员管理在分布式系统中广泛存在，针对不同的应用具有不同的实现机制，其关键是组结构的定义和组管理协议的设计[8,9,10,11]。

4.1 检查点逻辑环中的组结构定义

在组结构的设计中，将一个检查点单向逻辑环中所有检查点进程依照单向逻辑环定义的逻辑顺序关系，即顺时针方向，组成一个组。在任一检查点组中，定义一个组长（Leader）成员和一个副手（Prince）成员,其余组成员均为一般成员。Leader成员负责根据成员变更事件更新并广播成员状态视图，Prince成员将在Leader失效时，接管Leader成员的功能。在任一检查点组中，有且仅有一个leader和一个prince存在，且Leader为Prince的前继节点。

对于检查点组成员间维护的一致状态视图的定义为View = (vid, groupset);

其中，vid为视图版本号；groupset表示组内有效成员集及有效成员间逻辑顺序关系，其描述形式为groupset = (GM, f, s);

其中，GM为组内有效成员集，可表示为：

$GM=(gm_1, gm_2, \cdots, gm_n)$

其中，$gm_i = (mid_i, ip_i)$，mid_i为检查点组成员的id，ip_i为检查点组成员的ip地址;

f: GM→GM，s: GM→GM, 分别定义组内有效成员的前继结点和后续结点，可表示如下：

$$f(gm_i) = \begin{cases} gm_{i-1}, & 1 < i < n \\ gm_n, & i = 1 \end{cases}$$

$$s(gm_i) = \begin{cases} gm_{i+1}, & 1 < i < n \\ gm_1, & i = n \end{cases}$$

值得注意的是，对于任一组成员其id是全局唯一的，且在其失效重起后，该id将保持不变。

在该检查点组中，成员间通讯消息形式为(vid, info)，其中vid为消息发送方的持有的最新视图版本号，info为传送的消息。

在检查点组中，任一成员依照其获得的成员状态视图，定期从其前继结点接收心跳信号(Heartbeat)，以监控前继结点的状态，从而形成环形心跳侦测环。环形心跳侦测环是实现组管理的基础。图2给出了图1给出的5个结点的检查点逻辑环的组结构，其中虚线表示成员间接收心跳信号的关系，实线表示成员间备份检查点信息的关系，成员1为Leader， 成员5为Prince。

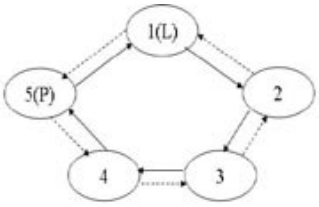


图2 5个检查点进程构成的组结构

4.2 检查点组管理协议

基于检查点组中不同的成员角色，我们设计了一套组管理协议包括组成员加入、组成员退出和组成员失效处理协议，以实现组成员一致视图的维护。

组员失效处理协议

(1) 若组成员的前继结点一段时间收不到该成员的心跳信号，若该成员为非leader成员，则其前继结点向Leader报告该组成员失效；若该成员为Leader成员，则其前继结点，即Prince成员，将自身升级为新的Leader；

(2) 若失效成员为非Leader成员，则Leader判断失效报告中的版本号是否与视图版本号一致，若不一致则不予以处理；否则，根据报告对失效成员进行侦测，若发现其失效，则更新组成员状态视图，将该成员从状态视图中剔除，将视图版本号加1，并将新的成员状态视图向组成员发送；

(3) 若失效成员为Leader成员，则新的Leader将失效成员从状态视图中剔除，将视图版本号加1，并将新的成员状态视图向组成员发送；

(4) 接收到视图信息的组成员判断新的视图版本号是否大于既有视图版本号，若判断成立，则更新

成员状态视图，并执行如下操作：

a) 根据原有成员状态视图，判断失效成员是否为自己的前继结点，如果是，则根据更新的成员视图，重新设置自己的检查点信息的备份成员和所接收的心跳信号的发送成员；判断失效成员是否为自己的后续结点，如果是，则根据更新的成员视图，重新设置自己所备份的检查点信息的所属成员和所发送心跳信号的接收成员；

b) 如果失效成员原为Prince，则根据原有成员状态视图，判断失效成员是否为自己的前继结点，如果是，则将自身的角色修改为Prince；

c) 向Leader发送回应信息；

(5) 如果Leader接收到所有的成员的回应消息，则失效处理成功；否则Leader对没有进行回应的成员进行失效处理。

组成员加入协议

在组成员加入协议中，我们区分两类加入行为：1）重新加入，即由检查点进程失效引起的加入行为；2）新加入，即由检查点进程所在节点时效而引起的加入行为，或者一个全新的检查点进程的加入行为。

(1) 加入成员向Leader发加入组的请求，若发送请求不成功，则操作失败；若发送成功，则执行以下处理;

(2) Leader判断申请为重新加入申请，还是新加入申请，若为新加入申请，则为该申请成员分配组员编号，其值为已分配的组员编号的最大值加1；若为重新加入申请，则判断该加入成员的编号是否已存在于当前成员状态视图中，若存在则不予以处理；

(3) Leader更新组成员状态视图，将视图版本号加1，并将新的成员状态视图向组成员发送。

(4) 接收到视图信息的组成员判断新的视图版本号是否大于既有视图版本号，若判断成立，则更新成员状态视图，并执行如下操作：

a) 检查自身的编号，确定其后续结点和前继节点的编号。如果加入成员为Leader的后续结点，则原来作为Prince的成员取消自己的Prince身份，新加入成员将自己设为Prince；

b) 判断加入的成员是否为自己的前继结点，如果是，则将加入成员设置为自己检查点信息的备份对象和所接收的心跳信号的发送成员；

c) 判断加入德成员是否为自己的后续结点，如果是，则将加入成员设置为自己所备份的检查点信息的所属成员和所发送心跳信号的接收成员；

d) 向Leader发送回应信息。

(5) 若Leader接收到所有成员的回应信息，则更新成功，否则，将未收到回应的成员按照“成员失效”协议处理。

(6) 若新加入成员长期无法接收到Leader发送的状态视图更新信息，则申请加入失败。

组成员退出协议

(1) 退出成员向Leader发退出组的请求，若发送请求不成功，则操作失败；若发送成功则继续。

(2) Leader判断退出请求的版本号是否与视图版本号一致，若不一致则不予以处理；否则继续如下处理。

(3) Leader根据成员退出消息更新成员状态视图，并将新的成员状态视图向组成员及退出成员发送。

(4) 接收到视图信息的组成员判断新的视图版本号是否大于既有视图版本号，若判断成立，则更新成员状态视图，并执行如下操作：

a) 根据原有成员状态视图，判断退出成员是否为自己的前继结点，如果是，则根据更新的成员视图，重新设置自己的检查点信息的备份成员和所接收的心跳信号的发送成员；判断退出成员是否为自己的后续结点，如果是，则根据更新的成员视图，重新设置自己所备份的检查点信息的所属成员和所发送心跳信号的接收成员；

b) 如果退出成员原为Prince，则根据原有成员状态视图，判断退出成员是否为自己的前继结点，如果是，则将自身的角色修改为Prince；

c) 向Leader发送回应信息；

(5) 退出成员收到Leader发送的消息后，退出。

4.3 协议分析

根据文献[11,12]，为确保成员间一致的状态视图，组成员管理协议应满足如下性质：

1）**成员变更确定性** 对于任何组成员变更的事件，包括成员加入、成员失效和成员退出，最终都将被感知,并形成相应的成员状态视图。

2）**视图顺序性** 对于任意两个成员变更事件e_i，e_j，其被感知的时刻分别为t_i，t_j，且满足$t_i \leq t_j$；其对应的成员状态视图分别为$view_i$和$view_j$，，对于任一组成员获得$view_i$和$view_j$的时刻分别为t'_i,t'_j，则$t'_i \leq t'_j$；

3）**稳态一致性** 当组达到稳态时，所有组成员的状态视图是一致的。其中，稳态是指组中所有成员均运行良好，成员间通讯正常，且无成员变更事件的状态。

以下我们阐述该协议对确定性、顺序性及一致

性的确保。

1）成员变更确定性

本协议的确定性是通过协议设计中所覆盖的组成员变更事件实现的。本协议覆盖了所有成员变更事件，即包括成员加入、失效和正常退出。对于成员加入和正常退出，将通过成员主动发起请求，Leader接收请求实现事件感知。而对于成员失效事件，可通过相邻节点的定期监控被发现，并最终通过可靠通讯向Leader报告来实现事件感知，或由Leader在进行视图更新广播过程中发现。综上所述，本协议对任何组成员变更事件实现了感知，故确保了确定性。

2）视图顺序性

本协议的顺序性是通过Leader集中协调处理和成员状态视图的版本管理来确保的。由协议描述可知，对于任一成员变更事件，是由Leader集中处理形成更新视图，升级视图版本号，并向各组成员广播，这就确保了成员变更事件的串行处理，即对于任意两个成员变更事件e_i，e_j，其被感知的时刻分别为t_i，t_j，若$t_i \leq t_j$，则$view_i$必先于$view_j$，形成，同时通过可靠广播，先于$view_j$传递给所有组成员。因此，本协议确保了顺序性。

3）稳态一致性

协议的一致性的确保是基于协议的确定性和顺序性的。可用反证法分析。若组达到稳态，至少存在两个组成员membera和memberb其状态视图不一致，分别为$view_i$和$view_j$，令e_i和e_j分别为$view_i$和$view_j$所对应的成员变更事件，且e_i先于e_j被感知，则根据协议顺序性可知，任一成员应在获得$view_i$后，获得视图$view_j$。在网络良好的前提下，若未获得视图$view_j$，其原因必为该组成员发生故障或Leader发生故障，这与组处于稳态的定义相悖。所以可知，该协议满足当组达到稳态时，所有组成员的状态视图是一致的。

同时，由于机群紧耦合计算环境中通常构建多套高速网络，网络可靠性高，延迟小。因此，与既有组管理协议比较，该协议的设计基于组成员通信可靠高效的前提，对于成员状态视图的更新采用由Leader成员集中控制的方式。这使得对于任一成员状态视图的更新同步仅需Leader与组成员通过一轮通信即可完成。因此，与传统的基于多轮协商的组管理协议相比，该协议具有简洁、高效的特点。

5. 实验分析

我们在基于曙光4000A的机群操作系统phoenix中实现了该检查点机制原型[13]。以集群操作系统核心服务(kernel)的形式为其上层的其他机群服务提供检查点功能，并以动态联接库的实现方式提供编程接口。我们对该原型进行了性能试验。

5.1 测试环境

基于曙光4000A，我们对协议进行了性能测试。测试环境共含64个节点，每个节点主要配置为2个AMD 64位Opteron CPU，内存为2GB，节点运行Linux操作系统。节点间通过两套网络连接，分别为千兆以太网和百兆以太网。在每个节点上部署一个检查点进程，所有检查点进程构成一个检查点进程组。

5.2 检查点信息读取与更新性能

我们选取检查点信息的大小分别为10kB， 20KB, 40KB, 80KB和100KB，进行读取与更新性能测试。

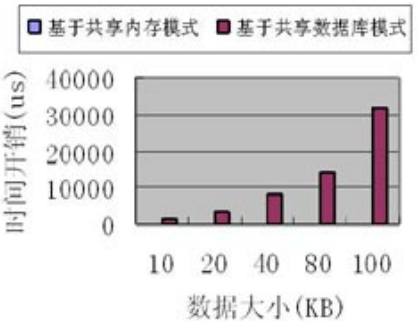


图3 检查点读取性能比较

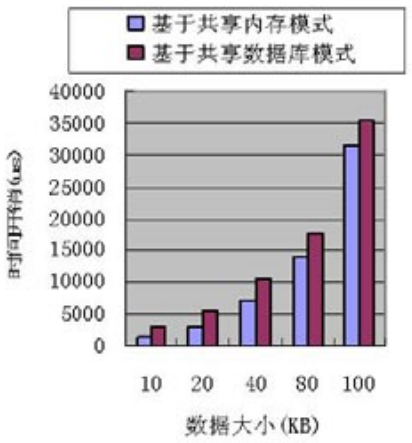


图4 检查点写入性能比较

由图3，图4可见，在检查点信息的更新性能上，基于数据库和基于共享内存的方式差别不大，这主要是因为这两种方式进行写操作时的主要开销都是在网络通讯上；在检查点信息的读性能上，基于数据库方式的开销远远大于基于共享内存方式，这主要是因为我们设计的基于共享内存的检查点机制在检查点读取时，只对本地的共享内存进行操作。因此可知，机群服务失效恢复后，采用基于共享内存的方式读取检查点能获得很好的性能。

5.3 检查点组管理协议性能

对于检查点组管理协议性能我们测试成员状态视图更新开销，成员状态视图更新操作是组管理协议对成员加入、失效和退出事件处理的主要开销。本节给出了在不同规模下，成员状态视图更新的时间开销。

表1 成员状态视图更新开销

机群规模(节点数)	成员加入更新开销(秒)	成员退出更新开销(秒)	成员失效更新开销(秒)
4	0.00203	0.0019	0.00205
8	0.00411	0.00412	0.00412
16	0.00831	0.00831	0.00833
64	0.0347	0.03468	0.0347

由表1可知，在相同机群规模下，成员加入、退出和失效所导致的状态视图更新由于均仅需一轮Leader成员与组成员之间的可靠通信即可完成，因此，时间开销基本相同。随着机群规模的扩大，由于Leader成员需通知的组成员随之增加，因此时间开销增大，但由于仅需一轮通信即可完成，因此开销较小，在节点个数为64的规模下，最大时间开销仅为0.0347秒。

6. 结论及下一步工作

不同于传统的基于共享数据库实现的检查点，本文提出了一种基于共享内存的机群服务检查点机制。该机制通过基于共享内存的检查点信息读写提高了检查点的执行效率；并基于单向逻辑环的检查点备份结构实现检查点信息的可靠维护。本文主要贡献包括1）面向基于共享内存的检查点信息主-备多存储的模式，设计了一套确保机群服务检查点信息的一致性的检查点信息管理协议； 2）设计了一套基于单向逻辑环的检查点组管理协议，该协议保证了逻辑备份环中检查点进程的成员视图一致性。基于曙光4000A对该检查点机制进行了原型实现及性能测试。实验表明与基于共享数据库的检查点机制比较，该检查点机制的检查点信息读取和更新性能最大分别提高了82倍和2倍；对检查点组管理协议的主要开销—成员状态视图更新开销的测试表明，在不同的机群规模下，该更新操作的时间开销较小，最大仅为0.0347秒，因此该检查点机制较好的满足了机群服务检查点需求。在下一步工作中，我们将对共享内存中检查点信息的组织进行进一步细化，以提高检查点读写效率。

参考文献：

[1] Building Secure and Reliable Network Applications.Kenneth P.Birman.Manning Publications Co.1996

[2] Why Do Computers Stop and What Can Be Done About It? Gray, Jim。HP Labs Technical Reports TR-85.7 June 1995。

[3] 基于数据库的机群检查点的研究与实现。武剑锋、戈弋、李三立。小型微型计算机系统。第23卷第3期2002年3月

[4] A Faster Checkpointing and Recovery Algorithm with a Hierarchical Storage Approach. Wen GAO Mingyu CHEN.Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA ’05)

[5] 一种降低并行程序检查点开销的方法。周小成，孙凝晖。计算机工程2007.6。

[6] S. D. Gribble, A Design Framework and a Scalable Storage Platform to Simplify Internet Service Construction. PhD dissertation, UC Berkeley, Fall 2000.

[7] 江滢. 曙光4000A基础件应用管理详细设计报告. 中国科学院计算技术研究所国家智能计算机研究与开发中心技术报告TR-DWARE. 2006.10

[8] George Coulouris;Jean Dollimore;Tim Kindberg, Distributed Systems: Concepts and Design (4th Edition), Addison Wesley, 2008.1

[9] Sergio Mena, André Schiper, A Step Towards a New Generation of Group Communication Systems, ACM/IFIP/USENIX International Middleware Conference On Middleware 2003, Rio de Janeiro, Brazil, 2003

[10] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev.Moshe: A group membership service for WANs. ACM Transactions on Computer Systems, 20(3):1 - 48, 2002

[11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. ACM Comp. Surveys, 33(4):1-43, Dec. 2001

[12]Bidyut Gupta,Shahram Rahimi, A Novel Recovery Approach for Cluster Federations, Advances in Grid and Pervasive Computing, 4459(2): 519-530, 2007

[13] Jianfeng Zhan， Ninghui Sun, Fire Phoenix Cluster Operating System Kernel and its Evaluation, IEEE International Conference on Cluster Computing 2005, Boston, USA, 2005

一种改进的OpenMP Guided调度策略研究

● 刘胜飞 中国科学院软件研究所并行计算实验室 北京 100190 bigbigfei@gmail.com
● 张云泉 中国科学院软件研究所并行计算实验室 北京 100190 zyq@mail.rdcps.ac.cn

摘要：

在科学计算中，循环结构是最重要的并行对象之一。在考虑负载平衡、调度开销等多方面因素的基础上，OpenMP标准制定了静态调度、动态调度、指数调度和运行时调度等不同策略。针对指数调度策略不适合递减型循环结构的缺点，本文提出一种改进的new_guided指数调度策略，并在OMPi编译器上加以实现。new_guided调度策略的主要思想是对前半部分的循环采用静态调度，后半部分的循环采用指数调度。此外，本文针对不同的循环结构，在多核处理器上对不同的调度策略进行了评测。测试结果表明，在一般情况下，OpenMP默认的static策略的调度性能最差；对于规则的循环结构和递增的循环结构，dynamic策略、guided策略和new_guided策略的性能差别不大；对于递减型的循环结构，dynamic策略和new_guided策略的性能相当，要优于guided调度策略；对于某些极不规则的随机循环结构，dynamic策略明显优于其它策略，new_guided策略的性能介于dynamic和guided之间。

关键词：OpenMP，负载平衡，静态调度，动态调度，指数调度，OMPi

1. 引言

OpenMP是一种支持Fortran，C/C++的共享存储并行编程标准。OpenMP基于fork /join的并行执行模型，将程序划分为并行区和串行区。不同的处理器之间通过共享变量完成数据交换。与消息传递相比，OpenMP具有可移植和易编程性等特点。因此，OpenMP在SMP（symmetric multi-processing）和多核体系结构的并行编程中得到了广泛的应用。

OpenMP一般面向循环结构进行并行化。循环语句在数值运算和图像处理中是最常见的并行结构之一。在并行处理的过程中，调度策略直接影响着系统资源的利用率和并行效率。OpenMP规范中的调度策略包括：静态调度（static scheduling）、动态调度(dynamic scheduling)、指数调度(guided scheduling)和运行时调度(runtime scheduling)。本文对OpenMP的指数调度策略进行改进，在OMPi编译器中实现了改进后的算法，并对OpenMP的循环调度策略进行了测试和分析。本文组织如下：第1节为引言；第2节介绍循环调度的相关研究工作；第3节介绍了OpenMP的调度策略；第4节介绍OMPi编译器；第5节介绍改进后的guided调度策略；第6节为实验结果与分析；第7节总结全文。

2. 相关工作

负载平衡一直是高性能计算领域研究的一个热点。在循环调度方面，学术界提出了很多循环调度的算法。除了OpenMP规范中的三种调度策略外，还包括梯式调度（trapzoid scheduling）[1]、亲和调度（affinity scheduling）[2]、基于耦合的调度（affinity-based scheduling）[3]等。不同的调度策略的效率跟目标系统的体系结构紧密相关。例如梯式调度在分布式存储系统中的效率明显高于指数调度[1]，但是在共享存储系统中却又低于指数调度[4]。文献[5]考虑处理器之间的性能差异，提出了一种改进的梯式调度策略，主要适用于分布式系统。

文献[6]和本文的研究都是基于文献[7]提出的α调度策略。但是，本文的工作与文献[6]和[7]不同。文献[6]和[7]的算法都是采用Master/Slave的并行模型实现，适用于分布式存储系统。本文的研究是基于现有的OpenMP编译器，可以更加方便直接地应用在共享存储系统的并行编程中。

在对OpenMP的调度策略进行评测方面，论文[8]通过不平衡迭代次数和调度次数对规则的循环结构进行了理论分析。本文采用实际的应用程序，在多核处理器上对不同的调度策略进行测试，并结合

论文[8]的观点对实验结果进行了分析。

3. 循环调度策略

3.1 循环分类

如果各个循环体之间没有依赖关系，那么每个循环体可以被作为一个独立的任务被调度执行。对于不同的循环体，它们的计算量并不一定相同。一般按循环体计算量之间的差异，可以将循环结构分为4种类型[1]，如图1所示。

规则循环结构是指每次迭代具有相同的计算量，如图1(a)所示。递增/递减循环是指随着循环变量的增加，每次迭代的计算量会逐步增加/减少，如图1(b)和图1(c)所示。随机循环是指每次迭代的计算量与循环变量之间没有线性的关系，例如图1(d)中的条件循环。后面三种循环结构统称为非规则的循环结构。对于规则循环结构的调度，不同处理器的负载完全取决于所采用的调度策略。对于非规则循环结构，不同处理器的负载除了于调度策略有关之外，还与每次迭代的计算量相关。

```
//规则循环
DOALL k = 1 To N
//loop body
X[k] = X[k] + A
END DOALL
```
(a)

```
//递增循环
DOALL k = 1 TO N
Serial DO i =1 TO k
//loop body
End Serial DO
END DOALL
```
(b)

```
//递减循环
DOALL k = 1 TO N
Serial DO i=1 TO N-k
//loop body
End Serial DO
END DOALL
```
(c)

```
//随机循环
DOALL k = 1 TO N
IF(condition) THEN
COMPUTE(1)
ELSE
COMPUTE(100)
ENDIF
END DOALL
```
(d)

图1 不同类型的循环结构

3.2 OpenMP调度策略

为了实现较好的负载平衡而取得最优的性能，就必须对循环进行调度与分块。OpenMP标准提供了四种不同调度策略：static调度、dynamic调度、guided调度和runtime调度。其中，runtime调度是指在程序运行时，通过环境变量指定其余3中调度策略中的一种，因此本文仅讨论前3种调度策略。

static调度是指将循环迭代划分成相等大小的块。当循环迭代次数不能整除线程数与块大小的乘积时，尽可能地划分成相等大小的块。在默认情况下，没有指定迭代块的大小，静态调度会尽可能为每个线程分配数量相同的迭代块。如果指定chunksize参数的值，静态调度将会为每个线程依次分配chunksize的迭代次数，直到所有迭代块都分配完毕。当parallel for编译指导语句没有带schedule子句时，大部分系统中都会默认采用不带参数的static调度方式。

dynamic调度一般采用一个内部的任务队列，当某个线程可用时，为其分配由chunksize参数指定的迭代次数。当线程完成当前分配的迭代块以后，将从任务队列头部取出下一组迭代。在默认情况下，块大小为1，此时每个迭代逐次地分配到各个线程，调度开销也会变得很大。

guided调度是一种采用指导性的启发式自调度方法。开始时每个线程会分配到较大的迭代块，之后分配到的迭代块会逐渐递减。迭代块的大小会按指数级下降到指定的chunksize大小。因此，guided调度又被称为指数调度[8]。可选的chunksize参数可以指定所使用的迭代块的最小值，默认为1。每次调度的迭代块数是由$NC_k=LC_k/(p*NP)$计算的，其中LC_k表示剩余的未调度的循环迭代次数，NP是线程个数，NC_k表示第k个迭代块的大小，p是一个比例因子，推荐值为1或者2。在OMPi编译器中，p的值为1。

当总迭代块数为1000，线程数为4时，表1给出了不同调度策略的分块大小。

表1 不同调度策略划分的迭代块大小

调度策略	每次调度的迭代块大小
static调度(默认)	250，250，250，250
dynamic调度(chunksize=50)	50，50，50，50，50，50，……
guided调度(chunksize=50)	250，188，141，106，79，59，50，50，50，27

3.3 不同调度策略的分析

为了实现较好的负载平衡而获取最优的性能，就必须保证每个处理器尽可能地处于忙碌状态，同时将调度开销、上下文切换开销和同步开销降到最低。论文[8]定义了不平衡迭代次数和调度次数来衡量不同调度策略的整体性能。不平衡迭代次数是指执行任务最多的处理器所执行的迭代次数与执行最少处理器所执行的迭代次数之差，用来衡量任务负载的不均衡。调度次数则可以用来近似地衡量因调度产生的额外开销。在实际的多线程程序中，往往很难达到这二者都最小的理想情况。

论文[8]针对规则的循环结构，分析了不同调度策略的额外开销和负载均衡。但是，对于非规则的循环结构，每次迭代的任务量并不相同，因此无法

采用不平衡的迭代次数对不同调度策略进行分析。此外，在不同的体系结构中实现各种调度策略的开销也并不相同，导致因调度产生的额外开销不能简单地采用调度次数来衡量。

在多核处理器上，循环调度所关注的问题不仅仅是负载的不平衡和额外的调度开销这两个基本因素。由于任务调度将决定循环迭代在各个线程上的分布情况，因此对各个处理单元访问存储系统的行为也会产生影响。比如采用静态调度策略，能够尽可能地降低当多个处理单元同时访问同一片存储区域时发生访存冲突的几率[9]。由于循环一般是顺序访存的，所以将循环分割为较大的块就可以减少重叠访存的几率，提高处理器cache的使用效率。但是较大的分块对于负载平衡却可能是不利的。反过来也是如此：有利于负载平衡的策略也可能对访存的性能不利。

4. OMPi编译器

OMPi[10]是由希腊Ioannina大学开发的一款OpenMP编译器。OMPi的最新版本是1.0.0，支持OpenMP2.5标准。OMPi实际上是一个C语言的代码转换器，可以运行在任何支持pthreads的平台上。在编译过程中，OMPi首先将带有OpenMP制导语句的C代码转换为采用多线程库实现的C代码。在默认情况下，转换后的代码采用pthreads多线程库实现。转换完成后，OMPi使用系统自带的编译器（本文的测试采用的是gcc编译器）对转换后的代码进行编译和连接。

图2给出了一个OMPi转换的例子。其中，ort_get_guided_chunk函数每次调用时都会计算本次分配的迭代块大小，并把分配到的迭代块的开始位置保存在from_变量中，把迭代块的结束位置保存在to_变量中。

5. 一种改进的guided调度策略

本文在OMPi编译器中实现了一种static调度和guided调度相结合的方法。对于前面α%的循环迭代，采用static调度策略平分给所有线程；而对于后面1-α%的循环迭代采用guided调度策略。我们将这种新的调度策略记为new_guided调度策略。

```
//原始代码
#pragma omp parallel for schedule(guided,50)
for(i = 0; i < 1000;  i++)
{
a[i] = i;
}
```

⇩

```
//OMPi转换后的代码
//初始化操作
……..
1   step_ = (1);
2   ort_entering_for(0,0,0,1000,50,step_, &gdopt_);
3   while(ort_get_guided_chunk (0,1000,step_,50,
4               &from_,&to_,(int*)0, gdopt_) ) {
5         if (from_ < (1000) && to_ == (1000))
6                   is_last_iter_ = 1;
7         for (i = from_; i < to_; i++) {
8                   {
9                               a[i] = i;
10                  }
11      } /* for */
12    //结束操作
……
```

图2 OMPi代码转换示例

OMPi编译器具有很好的代码结构，可以很方便地进行扩展。OMPi采用一个结构体ort_gdopt_t保存循环执行时的状态。为了实现new_guided调度策略，我们对ort_gdopt_t扩充了两个变量first_chunksize和is_first。其中，first_chunksize表示第一次调度时每个线程执行的迭代块大小，可以通过总的迭代次数乘以α%后再除以线程数得到。is_first为ture则表示本次执行的是第一个迭代块，采用static调度策略；否则采用guided调度策略。扩充后的ort_gdopt_t定义如图3所示。

```
typedef struct {
    volatile int  *data;
    // Denotes the current lb of the loop
    volatile void  *lock; // Lock to access *data
    int    nth;       // # siblings
    void   *me;       // my info node
    int    first_chunksize;
    //每个线程的第一块迭代块大小
    bool   is_first;
    //判断当前迭代块是否为第一块
} ort_gdopt_t;
```

图3 结构体ort_gdopt_t的定义

这样，对图2中的ort_get_guided_chunk函数做少量的扩充便可以实现改进后的调度策略。上一节提到，每次guided调度都是通过ort_get_guided_chunk函数实现的。它的基本框架如图4所示，其中第8行和第12行之间的代码为新加的，用来判断采用static调度还是guided调度。

我们对α的取值进行了测试。选取α=40%，50%，60%，70%，80%，90%进行比较，发现当α=50%时，new_guided调度策略相对于OMPi的

guided调度可以取得最大性能提升。在这种情况下，分别采用static和guided策略调度了所有循环迭代的一半。以表1中的数据为例，此时new_guided策略每次调度的迭代块大小依次为125、125、125、125、125、94、71、53、50、50、50、7。

```
1  int ort_get_guided_chunk(int chunksize,
   ort_gdopt_t *t, …/*other parameters*/) {
2    //chunksize的调用值为OpenMP制导语句中指定的最小迭代块大小
3  …… //省去初始化操作和错误检测
4  othr_set_spin_lock( (othr_spin_lock_t *) t->lock);
5      //采用guided调度策略计算本次调度的迭代块大小iters，省去具体代码。
6  if (iters > chunksize)
   //程序中指定的迭代块大小为chunksize
7    chunksize = iters;
8  if(t->is_first)
9  {  //每个线程的第一次调度采用static策略
10    chunksize = t->first_ chunksize;
11    t->is_first = false;
12  }
13   (*t->data) += step * chunksize;
   //更新下一次调度的迭代块的开始位置
14  othr_unset_spin_lock((othr_spin_lock_t *) t->lock);
15  …… //结束操作
16 }
```

图4 修改后的ort_get_guided_chunk函数

6. 实验结果及分析

6.1 实验环境

我们采用两个不同的多核处理器平台进行实验：(1)曙光天阔服务器S4800A1为4路 AMD Opteron Processor 870双核，主频为2.0GHz，8个核共享16GB内存，操作系统为Turbo Linux 3.4.3-9.2，gcc版本为3.4.3，优化选项为-O3。(2)IBM刀片机群中的1片HS21刀片，CPU为2路Intel Xeon 5150 2.66 GHz 双核处理器，内存为4GB，操作系统为Red Hat Linux 3.4.4-2，gcc版本为3.4.4，优化选项为-O3。在两个平台下，OpenMP编译器都采用OMPi(0.9.0版)。

测试程序选取了图1中所示的四种循环结构分别测试。Mandelbrot Set(MS)程序用于计算复平面上的曼德布洛特集合，窗口大小为18000*18000，来自论文[5]的研究项目。MS程序是一个随机的循环结构，图5给出了MS每次迭代的计算量分布情况。Compute Pots(CP)程序用于计算3维空间中的80000个点的某种场势。CP程序是一个递增的循环结构，来自2006年英特尔多核平台编码优化大赛。Adjoint Convolution(AC)程序用于计算两个长度为n2的向量的卷积，是一个递减循环结构，向量长度为800*800。论文[2,11-14]中的研究都采用了AC作为测试程序。Matrix Multiplication(MM)程序是一个用于计算浮点矩阵乘法的程序。MM程序是一个规则循环，矩阵规模为3000*3000。

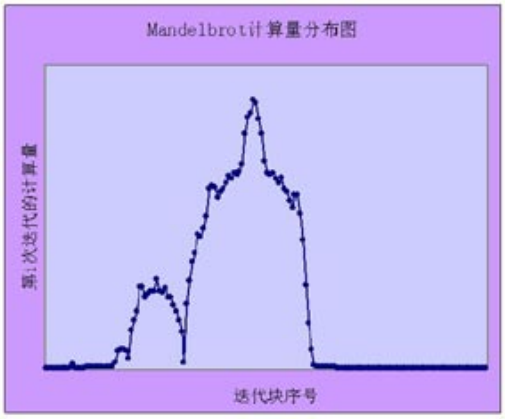


图5 Mandelbrot Set计算量分布情况

在测试过程中，计时操作是针对整个循环结构的，记录每个循环结构从开始计算到所有线程退出循环体所花费的时间。如图2所示的原始代码，我们在第1条语句的前面和第5条语句的后面分别插入gettimeofday()函数来获得循环执行的开始时间和结束时间，然后相减得到循环的计算时间。

6.2 在曙光服务器上的测试结果

对于每个循环结构，我们采用不同的调度策略进行测试。在测试过程中采用1至8个线程，测试结果如表2-表5所示。其中，调度策略中的10和50是指OpenMP制导语句中对应的chunksize参数。例如，(dynamic,10)表示每次分配的迭代次数为10；(guided,10)则表示除了最后一次迭代，每次分配的最小迭代次数为10。

表2 Mandelbrot Set在曙光机上的测试结果(单位：秒)

调度策略	线程数							
	1	2	3	4	5	6	7	8
static	272	197	220	247	172	194	135	121
static,10	272	136	91	68	55	91	78	68
dynamic,10	272	136	136	91	54	54	45	39
dynamic,50	272	136	91	68	54	45	45	40
guided,10	272	197	195	131	121	97	81	76
guided,50	272	197	195	238	121	98	80	77
new_guided,10	272	173	148	120	100	85	75	67
new_guided,50	272	173	148	120	100	85	75	67

表3 Compute Pots在曙光机上的测试结果(单位：秒)

调度策略	线程数							
	1	2	3	4	5	6	7	8
static	425	313	368	183	151	138	203	98
static,10	414	215	144	108	172	143	62	107
dynamic,10	415	211	142	107	107	71	71	61
dynamic,50	414	211	213	143	107	72	61	62
guided,10	415	211	142	107	110	89	74	62
guided,50	415	211	142	107	86	71	73	62
new_guided,10	414	210	141	106	85	71	61	53
new_guided,50	414	211	141	106	85	71	61	53

表4 Adjoint Convolution在曙光机上的测试结果(单位：秒)

调度策略	线程数							
	1	2	3	4	5	6	7	8
static	182	147	168	110	119	113	116	113
static,10	178	105	127	112	122	117	124	124
dynamic,10	182	119	106	108	112	115	115	117
dynamic,50	183	106	110	110	110	118	115	117
guided,10	183	148	125	118	109	116	114	114
guided,50	183	147	118	116	114	116	117	121
new_guided,10	181	126	110	110	111	114	116	116
new_guided,50	182	106	113	109	113	104	115	116

表5 Matrix Multiplication在曙光机上的测试结果(单位：秒)

调度策略	线程数							
	1	2	3	4	5	6	7	8
static	160	78	56	44	37	55	49	45
static,10	153	78	55	42	65	55	43	43
dynamic,10	152	80	53	43	37	33	33	31
dynamic,50	152	79	56	55	45	38	33	34
guided,10	152	79	55	44	47	37	35	36
guided,50	152	77	55	43	37	41	38	32
new_guided,10	136	74	54	46	47	47	49	49
new_guided,50	137	74	54	46	49	49	49	49

6.3 在IBM刀片上的测试结果

对于每个循环结构，我们采用和曙光服务器相同的调度策略进行测试。测试过程中采用1至4个线程，测试结果如表6-表9所示。

表6 Mandelbrot Set在IBM刀片上的测试结果(单位：秒)

调度策略	线程数			
	1	2	3	4
static	208.71	151.21	169.35	133.33
static,10	210.31	105.23	70.27	52.63
dynamic,10	207.97	104.04	69.36	52.04
dynamic,50	207.92	104.05	69.42	52.04
guided,10	207.94	150.96	149.54	100.77
guided,50	207.94	150.92	149.44	100.96
new_guided,10	208.58	133.24	113.66	92.46
new_guided,50	208.65	133.23	113.74	92.53

表7 Compute Pots在IBM刀片上的测试结果(单位：秒)

调度策略	线程数			
	1	2	3	4
static	282.81	211.79	156.93	123.61
static,10	284.45	141.54	94.23	70.65
dynamic,10	286.28	141.39	94.17	70.63
dynamic,50	286.31	141.45	94.28	70.76
guided,10	286.11	141.48	94.21	70.64
guided,50	286.3	141.32	94.16	70.75
new_guided,10	285.74	143.19	95.57	71.80
new_guided,50	285.73	143.10	95.73	71.89

表8 Adjoint Convolution在IBM刀片上的测试结果(单位：秒)

调度策略	线程数			
	1	2	3	4
static	154.29	115.86	85.99	68.02
static,10	154.80	76.98	51.59	38.57
dynamic,10	154.40	77.21	51.35	38.49
dynamic,50	154.49	76.99	51.39	38.49
guided,10	154.42	116.05	85.93	67.80
guided,50	154.69	116.36	86.47	68.00
new_guided,10	153.92	77.37	51.67	38.94
new_guided,50	154.29	77.54	51.66	39.04

表9 Matrix Multiplication在IBM刀片上的测试结果(单位：秒)

调度策略	线程数			
	1	2	3	4
static	120.48	60.00	40.53	30.49
static,10	120.35	59.99	40.57	30.33
dynamic,10	120.40	59.99	40.58	30.36
dynamic,50	120.37	60.72	40.57	30.36
guided,10	120.33	60.12	40.62	30.38
guided,50	120.37	60.72	40.53	31.12
new_guided,10	120.34	60.76	40.53	30.49
new_guided,50	120.36	60.72	40.89	31.07

6.4 实验结果分析

从表2 - 表9可以看出，不同的调度策略和chunksize参数在曙光服务器和IBM刀片上对性能的影响基本相同。对于四个测试程序，默认的static调度性能都是最差的。对于dynamic、guided和new_guided调度，chunksize参数对调度性能的影响比较小。

MS程序是一个不规则的循环结构。从表2和表6可以看出，dynamic调度和（static，10）调度可以获得较好的性能，其次是new_guided调度策略。如图3所示，MS程序的迭代可以分为前面、中间和后面三个部分。其中，在前面部分和后面部分的迭代中，每次迭代的计算量都很小，而中间部分的迭代的计算量都特别大。对于这样的循环结构，guided和默认的static调度都会导致获得中间那部分迭代的线程工作量非常大，而其它线程的工作量非常小，从而导致严重的负载不平衡。dynamic调度和（static，10）调度会将计算量大的迭代和计算量小的迭代均匀地分配给各个线程，因此可以获得较好的调度性能。这两个调度策略的区别在于（static，10）在进入循环前就将所有迭代分配完毕，而dynamic调度则是在一个线程执行完当前迭代块后再给它分配下一个迭代块。

论文[8]的研究指出：“如果计算负载是随迭代变量增大而减小的，则应避免使用指数调度，因为它的前几个子块大小都很大，可能导致负载不均衡。”从表8的测试结果可以看出，递减结构的AC程序采用（static，10）、dynamic和new_guided调度时能够获得最佳性能，而采用guided调度策略时性能较差。这一点与论文[8]的结论一致。对于递减循环结构，计算负载随着迭代变量增大而减小，而guided策略划分的迭代块大小也是由大变小，且呈指数下降。因此，对于递减循环结构采用guided调度会导致前面的迭代块计算量非常大，而后面的迭代块计算量非常小，从而引起负载不平衡。在曙光服务器上，当线程数大于3时，随着线程数的增加，AC程序的执行时间没有继续减少。这是由于AC程序访问存储单元时具有一定的不规则性，当线程进一步增多时，存储访问的局部性被破坏了。

论文[8]采用调度次数来衡量因调度产生的额外开销，认为对递增结构的循环，“应避免使用静态调度策略和分块大小较大的动态调度策略。”但是，在实际的编译器中，不同调度策略进行一次调度的开销往往并不相同。编译器在为线程分配迭代块时，一般首先锁住所有线程共享的任务队列，然后在临界区（critical section）计算需要分配的迭代块大小，并更新任务队列，最后解锁。dynamic调度采用的是OpenMP制导语句中的chunksize参数来直接指定迭代块大小，而guided调度则需要在每次调度时计算迭代块大小。也就是说，dynamic调度对应的临界区代码量少于guided调度的临界区代码量。例如在OMPi编译中，dynamic调度对应的临界区代码为2行，而guided调度对应的临界区代码为8行。因此，执行一次dynamic调度的开销要少于执行一次guided调度的开销。但是，guided调度每次分配的迭代块多于dynamic，因此，对于同一个程序，采用guided策略的调度次数要少于策略。对于表1中的例子，采用dynamic策略需要进行20次调度，而采用guided策略只需要10次调度。如果综合考虑一次调度的额外开销和调度次数，dynamic调度和guided的总额外开销近乎相似。从表3、表5、表7、表9中的测试结果可以看出，对递增结构的CP程序和规则的MM程序采用dynamic、guided和new_guided调度时的性能差不多。

综合表2 - 表9的测试结果可以看出，在OpenMP程序中采用dynamic策略和（static，10）策略的调度性能比较好。new_guided策略对guided策略进行了改进，调度性能介于guided和dynamic之间。由于static调度在编译期就决定了各个处理器的迭代块大小，因此无法根据程序运行时处理器的负载动态调整。此外，当调度非规则循环结构或者处理器性能不一致时，static调度很容易出现负载不平衡。如在表3、表4、表5中，（static，10）策略的调度性能就比dynamic和guided策略的调度性能差。综合前面的分析，我们可以认为，在大部分的情况下，dynamic策略都能获得最好的调度性能。

8. 总结及以后的工作

随着多核处理器的问世，多线程程序设计已经成为一个不可回避的问题。在采用OpenMP编程时，调度策略和参数的选择对一个OpenMP程序的性能具有一定的影响，选择恰当的调度策略和参数往往跟具体的运行环境和循环结构有关。

现有的研究工作（如文献[1]和[8]）都表明，对于递减型的循环结构，采用guided调度策略时存在严重的负载不均衡问题。本文借鉴论文[7]的α调度策略，对循环结构的前50%的迭代采用static调度，后面50%的迭代采用guided调度，从而改进了OpenMP标准所指定的guided调度策略。此外，针对不同的循环结构，我们在不同的多核处理器上对不同的调度策略进行测试和分析。本文的进一步工作可以研究α调度策略在机群OpenMP系统中的应用。

参考文献：

[1] T H Tzen, L M Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers. IEEE Trans on Parallel and Distributed Systems, 1993, 4(1): 87-98

[2] E P Markatos, T J LeBlanc. Using processor affinity in loop scheduling on shared memory multiprocessors. IEEE Trans on Parallel and Distributed Systems, 1994, 5(4): 379-400

[3] W Shi, Z Tang, W Hu. A more practical loop scheduling for home-based software DSMs. The ACM-SIGARCH Workshop on Scheduling Algorithms for Parallel and Distributed Computing from Theory to Practice, Greece, 1999

[4] Yun Zhang, Mihai Burcea, Victor Cheng, Ron Ho and Michael Voss, An Adaptive OpenMP Loop Scheduler for Hyperthreaded SMPs. International Conference on Parallel and Distributed Systems, San Francisco, 2004

[5] T. Chronopoulos, R. Andonie, M. Benche and D. Grosu, A Class of Loop Self-Scheduling for Heterogeneous Clusters. Proceedings of the 2001 IEEE International Conference on Cluster Computing, 2001

[6] Wen-Chung Shih, Chao-Tung Yang, Ping-I Chen, Shian-Shyong Tseng. A Hybrid Parallel Loop Scheduling Scheme on Heterogeneous PC Clusters. Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies, 2005

[7] Chao-Tung Yang, Shun-Chyi Chang. A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters. Journal of Information Science and Engineering, 2004, 20(2), pp: 263-273

[8] Shameem Akhter, Jason Roberts. 多核程序设计技术.李宝峰等译. 北京:电子工业出版社, 2007. pp: 147-151

[9] V.V. Dimakopoulos, E. Leontiadis and G. Tzoumas, A portable C compiler for OpenMP V.2.0. In: Proceedings of the 5th European Workshop on OpenMP, Aachen, Germany, 2003, pp: 5-11

[10] 赖建新, 胡长军等. OpenMP任务调度开销及负载均衡分析. 计算机工程, 2006, 32(18): 58-60

[11] S. F. Hummel, E. Schonberg, L. E. Flynn. Factoring: a method scheme for scheduling parallel loops. Communications of the ACM, 1992, 35(8): 90-101

[12] Wang, Yi-Min, Wang, Hsiao-Hsi, Chang, Ruei-Chuan. Hierarchical Loop Scheduling for Clustered NUMA Machines. The Journal of Systems and Software, 2000, vol. 55:33-44

[13] Chao-Tung Yang, Shian-Shyong Tseng, Ming-Chang Hsiao, and Shih-Hung Kao, A portable parallelizing compiler with loop partitioning, Proceedings of the NSC ROC (Part A), 1999, 23(6) :751-765

[14] Rizos Sakellariou. A Compile-Time Partitioning Strategy for Non-Rectangular Loop Nests. Proceedings of the 11th International Symposium on Parallel Processing, April, 1997: 633-637

浅析Viva软件编程

● 杜晓梅 江南计算技术研究所 无锡 214083
● 肖华云 江南计算技术研究所 无锡 214083

1. 引言

Star Bridge Systems公司的Viva开发软件是一种高级、图形的、面向对象的编程工具，它使FPGA编程更简便，功能也更强大。Viva为复杂的超级计算机领域和嵌入式系统应用简化了硬件设计，为采用数值法编写可重用和可扩展的程序库提供了一个平台。

针对高性能计算市场，Viva开始冒险尝试可重构高性能计算的趋势是使用FPGA开发工具，基于C语言来编程。使用C是一种自然反应和趋势，因为它已成为大部分编程人员熟悉和便于使用的开发语言。

2. Viva对于FPGA编程的作用

FPGA曾作为一个中间阶段被硬件设计工程师用于设计、验证并测试电路，最后的目标是可以在一个ASIC（专用集成电路）中使用，例如一个定制DSP或其他专用芯片。ASIC是快速且高度专用的，因此它的效率非常高。其投入市场的成本很高，因此通常用于大规模市场应用。在过去的二十年里，基于文本的硬件设计语言（HDL）如VHDL和Verilog已用于设计或编程这种定制电路。FPGA比处理器具有更低的时钟速度，所以不会再作为处理元件本身的功能使用。

过去几年中，FPGA一直在赶超处理器，并且已经超越了摩尔定律的预测，它比微处理器更密集、更快，也更便宜。事实上，现在的大部分定制电路设计可以在一个FPGA上执行，而不用经过把一个定制的ASIC投入市场这一长期又昂贵的过程。这些FPGA最终能否用于加速甚至是替换传统的微处理器来运行高度并行、计算速度更快的应用呢？在20世纪90年代中期，Star Bridge的创始人兼CTO Kent Gilson非常相信这一点，并且开始探索建立FPGA的理想开发环境。这种远见使Viva编程环境得以建立，是专门为开发目前的高性能FPGA而建立的。

尽管基于C的FPGA编程环境目前可以帮助应用编程人员通过制定简单的重定向功能调用在基于FPGA的硬件中放置这些核，但他们不能真正为建立最优核和进行复杂设计而设计成并行硬件设计语言。C和C++从未真正设计为可以在可重构FPGA硬件中进行并行编程，也从未设计成结合硬件设计语言（如VHDL）来完成任务，这样做使编程人员非常难以应付且面临挑战。这些解决方案足以把核和简单的单芯片设计应用到一个单独的FPGA中，但为了在一个深度扩展（多个FPGA设计和多个FPGA板对板设计）的环境中获得最佳性能和更大的并行性应用，这些工具的能力将需要大大改进。

Star Bridge的Viva FPGA设计环境是要帮助工程师快速地设计、建立样机、测试、排错、模拟、优化并在FPGA中运行其程序。这些设计可以在Linux或Windows环境中通过对C的目标可重构硬件的API调用很容易地被植入到目标环境中。

3. Viva软件编程

Viva算法编辑器允许任何数值法的最纯粹的描述。它是面向对象的、图形的、递归表示机理，“独立实现”的含义是指利用的算法是不用考虑最终目标运行环境的物理结构而设计的。利用传统编程语言，使用的算法必须被分解，以便通过一个语言编译器来遵守串行语言辞典的限制。这些语言本身效仿了硬件的很多限制，它们迫使这种表述限制作为固定的数据类型和串行执行表述，以便可以适应基于执行系统的微处理器。

3.1 Viva设计原理

在过去的几年里，在目前的更复杂的FPGA上，曾做过很多尝试以适应系统和应用级硬件编程逐渐增加的需求。一种方法就是扩展标准硬件设计语言如Verilog的抽象级别。在EDA领域中，这是比较有用的，但它不适合传统的应用编程人员为HPC建立应用程序。另一个更加普遍的方法（尤其是在可重构高性能计算世界中）就是利用对硬件编程并行性的模板来扩展C++的功能性。由于这些方法更加接近于标准的C，因此在FPGA中开发的功率和效率会有损失。

Viva的设计源于可扩展的FPGA编程，它使得人们可以在面向对象的绘图环境中利用所有自然并行性和递归表述来表示出算法。通过把功能性对象拖

到一个板上，并利用表示数据流的传送器把它们连接起来，可以非常容易地模拟算法，获得所有固有的并行性。在Viva中，由于默认设计表达是并行的，串行执行的表达属于异常情况。对象是多态的并且可以用于不同的数据类型、数据率和数据精度，这样，就提供了代码重新使用的优势。这使得设计者可以在时间（执行所必须的时钟周期的数量）和空间（该操作可使用的硬件资源的数量）之间协商设计需求。在Viva中，需要有关于表达门级别的设计的自由，或建立可以代表系统级别抽象的更高级别的对象。Viva安装了数学程序库和很多其他便于算法开发进程的对象。

Viva最终将为基于这种表达的可重构硬件和目标系统的容量与能力建立一个定制结构。考虑到特别设计和数据流的需求，可以对所需的原型、对象和设计方法进行混合和匹配。Viva允许“按需结构”并创造最佳的硬件内算法的实现。

Viva的另一种能力就是从算法设计者获得并提取目标硬件系统。Viva采用了Java的虚拟机的概念，它使写在一台计算机上的代码便于在其它计算机上执行，这就达到了一个更高级别的抽象。这种新的抽象是根据一个组成该系统的行为能力的系统描述、这些行为所消耗的资源以及这些资源的成本所构成的。系统描述被分级建立并组合，以便确定整个目标系统或用户用以编程的执行环境。

在系统编辑器中，Viva能够描述数据处理候选分区的物理部件。它可以是来自FPGA、存储器、总线、I/O接口、微处理器、ASIC，以及通信和行为系统的任何信息。物理元件、它们的行为、计时和成本都被Viva捕获到。在一个Viva系统描述中提到的所有可用资源都成为候选分区，用于计算数据处理。这些都可以通过算法设计者来分配，或通过Viva本身来分配、分区和协商。因此，任何系统类型都包括CPU、FPGA和通信系统，任何排列或组合都可以在Viva中被模拟，使得用户程序可以在任何计算基底上透明执行，Viva可以在不用解释执行层开销的情况下完成。

Viva中的系统描述存在于x86系统中，用于PC机上模拟设计。它们不仅在超级计算机的Star Bridge系列中出现，还出现在Honeywell可重构空间计算机、XESS XSB-300板、XESS XSV板、Smiths-航天PMC板和SGI可重构硬件系统中，主要的可重构HPC厂商对此也取得一些进展。

Viva具有在不改变代码的情况下在不同的执行环境中运用算法资源代码的能力。如果该算法被正确设计，那么在一个不同的系统描述中交换和重新编译满足了在一个不同的目标系统上执行的需求。Viva使用系统描述作为综合操作的上下文，这样就导致了在系统描述中分解的编译程序。这为以后开发的硬件验证代码是十分有好处的。

3.2 Viva结构及特点

Viva是一种大规模数百万门设计的停止一次（One-stop）设计方案，采用了OOP的高级图解捕获类型设计范例。它具有部分综合选项的快速、有效的综合，以及有效的基于窗口选项的建模和应用模拟。对于不同FPGA板具有自适应性，并且可以使用FPGA厂商工具来放置并布线，无需费时、费力或改变设计就可以把在Viva中创造的设计移植到其他硬件结构中。

3.2.1 Viva总体结构

Viva的总体结构包括三个方面：用户界面；排错；CoreLib和系统描述。下面，分别简要介绍这三个部分。

（1）用户界面

Viva的用户界面包括工具栏、项目/程序库编辑器、消息窗口、I2ADL编辑器、数据集编辑器、系统编辑器和资源编辑器（如图1所示）。

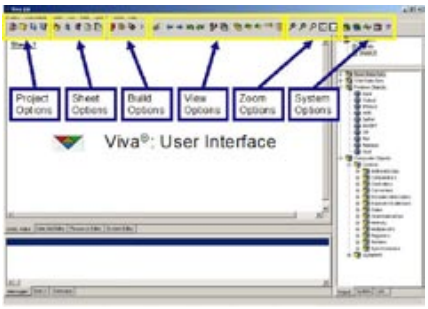


图1 Viva的用户界面

（2）排错

在Viva中，进行排错的过程为：通过输出对象与传送器的连接和指定输出对象作为用户陷阱检测内部信号；采用x86系统描述来模拟可以检验其功能性；消息窗口说明了错误/警告和对象发生错误的位置；在Viva文件夹内已记录错误消息。图2是一个排错例子，其目的是在Viva中证明排错技术。

在Viva中，全局和陷阱两种属性可以推动调试技术。全局属性用于建立一个输出，该输出不用视为对象占用面积的一部分。另外，它也可用于在一个对象的输入和另一个对象的输出间建立隐式连接。陷阱属性随着一条消息而显现出来。当数据被传播到该输出上，这条消息就会显示出来。

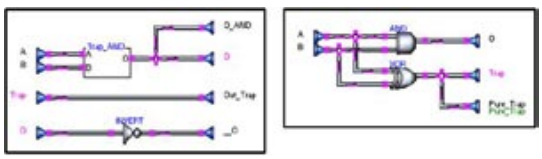


图2 排错示例

（3）CoreLib和系统描述

CoreLib和系统描述（如图3所示）的特点为：CoreLib设计程序库包括大量来自原语的对象；程序库模块包括数学函数、逻辑门、移位器、寄存器、脉冲生成器、计数器、反馈和循环控制对象、开始－完成－繁忙－等待（GDBW）界面和对象、存储器对象和与其他芯片进行芯片间通信的总线对象；在应用开始时默认负载。

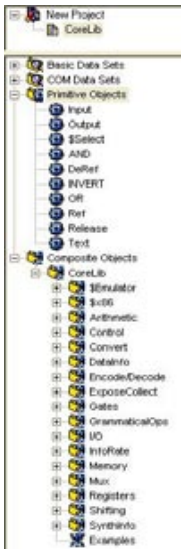


图3 CoreLib和系统描述界面示意图

3.2.2 Viva主要特点

Viva具有如下主要特点：数据流中心编程模型；并行元件面向对象语言；递归算法/拓扑；无限操作员超载；数据集多态性（多重精确操作员）；信息率多态性（多速率操作员）；上下文敏感操作员合成（系统目标和使用上下文）；强类型；数据集分解/分解操作员；恒定折叠与用户可扩展算法；动态数据集生成；定时驱动分区/同合成；局部和全局标签分辨率；操作系统核；事件驱动实时混合模式（串行和并行）执行核；多处理分层线程生成和寿命管理操作员；动态默认界面生成和执行；处理间通信和处理器部分存储器管理；包括迟边界动态对象的完全COM（公用对象模型）执行实现；通过进程和机器边界的生成和界面配置；应用分配的卓越的可执行生成；动态可重构支持。

3.3 Viva设计流程

Viva的大致设计周期及设计流程为（如图4、图5所示）：

首先，在基于Viva窗口、面向对象的图形开发环境中创建一个设计好的模拟验证。在这个设计阶段，用户只需要有一台计算机即可。Viva在目标硬件中运行设计之前就可以使用户在计算机上表达、测试并验证设计；

然后，要在Viva中装载适当的“系统描述”，它会告诉Viva你所针对的所有硬件。该系统描述为Viva解释了物理元件、行为通信系统和在目标系统上使用不同硬件元件的相关开销。一旦这种系统描述被负载，你就需要在真实的目标系统中为实现重新编译并建立已证实的设计。利用Viva，你可以在执行期间利用到硬件中的可见度来对设计进行测试和排错；

最后，一旦用户的设计完成，Viva就会为FPGA生成可负载位流。这种位流基本可以在任何操作系统环境中被负载并运行。如果目标是一台Star Bridge超级计算机，API允许在Linux环境中从C负载并运行代码。通过API和系统描述，Star Bridge已建立了代码以便在VME、Windows和Linux中运行。

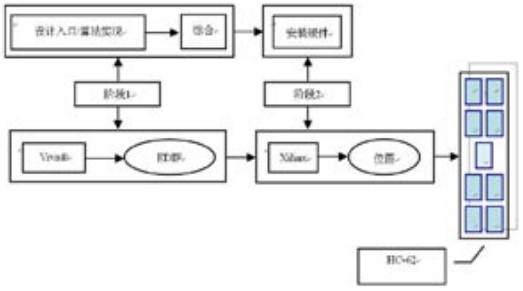


图4 Viva中的设计周期

移植到Viva的步骤为：首先，进行算法分析，而无需进行优化；接着，进行设计方面的考虑，包括并行化（内部和多管）和硬件效率（I/O、存储器和数据宽度）；最后，进行编码、测试和修改。

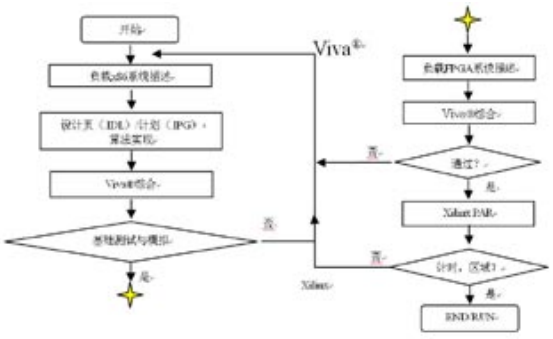


图5 Viva中的设计流程

4. 结束语

Viva是Star Bridge System公司（成立于1998年，位于犹他州盐湖城）开发的面向高性能可重构计算的开发环境。目前，它主要用于Star Bridge制造的超级计算机，但Star Bridge公司日前已宣布，该公司正在与Cray和SGI公司商谈合作事宜，力图尽快地将Viva移植到Cray XD1超级计算机及SGI Altix服务器上，以进一步促进可重构超级计算的软件编程。

2008年上海超级计算中心基础科学用户研究进展

王 涛 [编][译] 上海超级计算中心 北京 100190 twang@ssc.net.cn

编者按：

2008年上海超级计算中心为超过200家的基础科学研究组提供了高性能计算服务与支持。在整个2008年度，用户共发表了114篇被科学引文索引（SCI）收录的论文，领域遍及物理、化学、天文、生物等基础科学方向，这些论文的研究成果均为上海超级计算中心计算平台支持产生。本文在上述的114篇论文里，精选了在国际顶级刊物美国化学会志（Journal of American Chemical Society）和物理评论快报（Physics Review Letters）上发表的7篇重要成果进行介绍，以方便广大读者进一步了解高性能计算在基础科学领域内的应用。这些成果均为相关领域内的重要进展，在国际上引起了广泛关注。

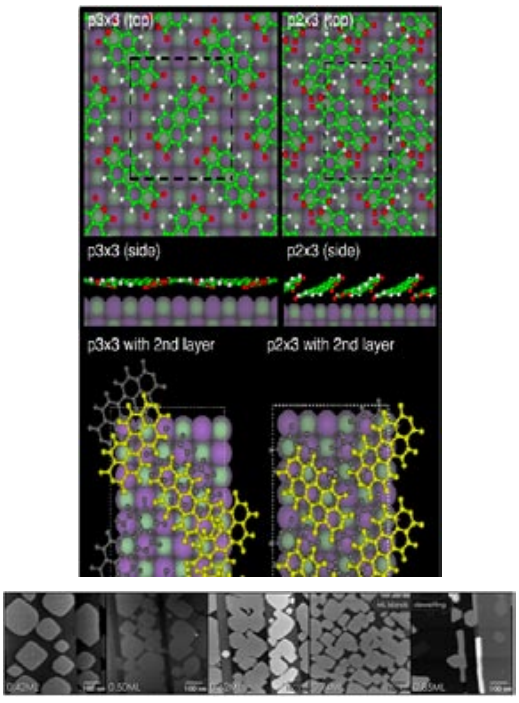
论文题目：分子体系中应力导致的去浸润：PTCDA在NaCl表面的两种生长模式

项目来源：国家自然科学基金、国家科技部、中国科学院

用户来源：中科院物理所、加拿大McGill大学物理系

论文来源：物理评论快报第100卷第18期，论文号：186104，2008年5月9日出版

本项工作在有机功能分子在典型绝缘体表面生长的研究中取得了重要进展。功能分子在固体表面上的构型是影响其电子结构的重要因素之一。国际上的前期研究工作重点主要在金属表面上有机功能分子的组装生长。而在作为绝缘层的绝缘体表面上有机分子生长机制的研究，对于实用器件的构筑意义重大。本项工作通过利用高分辨原子力显微镜与第一性原理计算相结合的方法，对PTCDA这一有机半导体原型分子在NaCl绝缘体表面的生长模式进行了研究。研究发现PTCDA的第一层生长模式为浸润生长，其中分子平面与NaCl表面平行，且分子与表面的相互作用力为主导。与通常在金属表面的多层膜生长模式不同的是，在NaCl表面多层膜生长过程为去浸润生长，其中分子间的相互作用为主导。这使得原为浸润的第一分子产生了去浸润，即原与表面平行的PTCDA分子平面由于与第二层分子较强的相互作用而产生了倾斜（与第二层分子平行），而不再与表面大面积接触。这一转变来源于多种相互作用及其产生应力之间的相互平衡。通过理论计算与实验结合该研究澄清了一种有机功能分子在典型绝缘体表面生长的重要一步，即去浸润生长以及这种生长机制的原因。该项研究结果发现了一种新的分子多层膜生长模式，为组装实用化分子器件的研究奠定了良好基础。



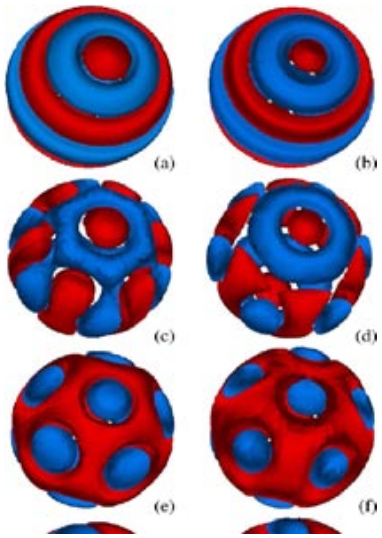
论文题目：中度薄球壳中的水热对流

项目来源：国家自然科学基金、中国科学院

用户来源：上海天文台、英国Exeter大学数学科学系、美国地球物理与行星物理研究所地球与空间科学部

论文来源：物理评论快报第101卷第2期，论文号：028501，2008年7月11日出版

本项工作在球壳型孔隙媒介中的水热对流解析以及它的直接三维数值模拟研究中，取得了重要进展。水在一个具有中心重力场和内热的球壳型孔隙媒介中的对流是太阳系天体演变过程中非常可能发生的现象。在碳质球粒陨石父体的早期热过程中，父体中心的冰首先溶化，释放出来的岩石将沉入中心以组成岩心。这时，液态水可能不只是简单的填充父体中的孔隙，而是以对流的方式循环，这种现象与在地壳的地热系统中观察到的现象类似。这种通过岩型球壳的水热对流在许多外行星的冰质卫星中也可能曾经发生过或者正在发生。这些卫星都是有一个水-冰壳层来环绕岩心。通过对球形壳层孔隙媒介中水热对流机制的理解，可以帮助我们明白这些天体的演变过程。另外，许多天文物理的研究对象也是用球壳形几何和球形对称的方式来进行研究，这类数学问题和本项研究工作中的问题有很大的相似性。因此本项研究工作对许多天文物理中的重要问题有很大的参考价值。目前，人们对球壳形孔隙媒介中的水热对流机制的理解非常有限，主要原因是球形对称体系和球形对称边界条件的对流问题在数学上非常复杂。已经有一些研究人员提出过一些解决方案，但是通过非线性分析得到球壳型孔隙媒介中的水热对流解析解以及它的直接三维数值模拟的验证，还没有被尝试过。在本项工作中，作者首次对球形对流的现实物理问题进行了数学分析，得到了对流流动的结构。此外，本项工作也首次得到了水热球形对流解析解和直接数值模拟相一致的结果。作者们的数学分析和数值模拟发现在微行星体和冰质卫星中，水通过球壳型孔隙媒介的循环模式有很多类型。



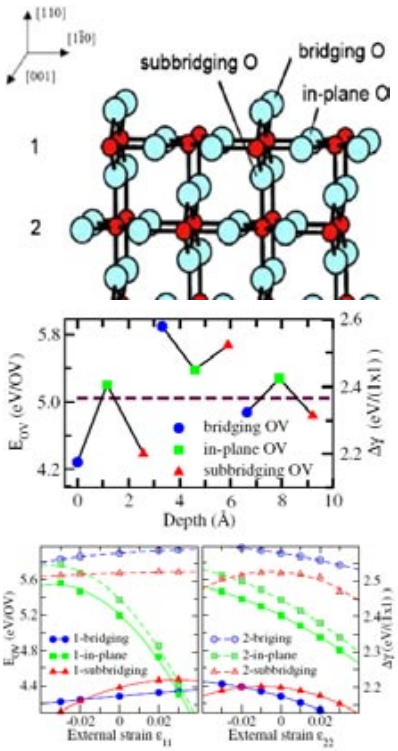
论文题目：金红石相TiO2（110）表面上的氧空位与外部应力的相互关系

项目来源：国家自然科学基金、国家科技部

用户来源：南京大学物理系

论文来源：物理评论快报第101卷第11期，论文号：116102，2008年9月12日出版

本项工作通过第一性原理计算，研究了二氧化钛晶体表面氧空位的形成能与外应力的关系。氧空位是过渡金属氧化物表面最普遍的缺陷，在光催化和环境净化等应用的物理过程中起着重要的作用。如果表面氧空位的位置和分布能够人为控制和调节，人们就可以根据不同的应用目的来调控表面的性质。该工作通过第一性原理计算发现，金红石的TiO2（110）面上不同位置的氧空位的形成能依赖于外应力。氧空位形成能对外应力的依赖随着氧空位的具体位置不同而变化，因而可以通过外应力来控制TiO2（110）表面氧空位的分布。这个工作为表面化学、环境保护、催化等领域的进一步研究提供了非常重要的信息。



论文题目：锌指半胱氨酸（2）-组氨酸（2）的金属耦合折叠

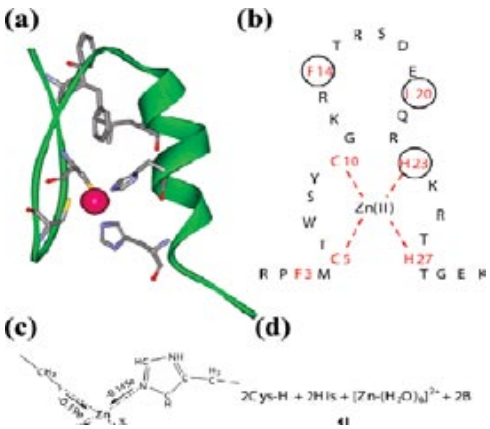
项目来源：国家自然科学基金、国家重点基础研究发展计划（973）、中国博士后科学基金

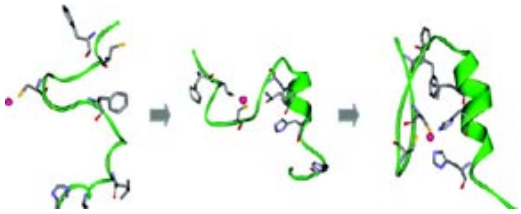
用户来源：南京大学物理系

论文来源：美国化学会志第130卷第3期，页号：

892-900，2008年1月23日出版

本项工作是蛋白质折叠动力学研究的重要进展。在该项工作中，作者利用大规模计算机模拟首次理论研究了金属辅助因子诱导的蛋白质折叠问题。蛋白质折叠是生物学中心法则中至今仍未解决的一个重要环节。以往的研究主要集中在能够自发折叠的蛋白质体系。但在生物体内，蛋白质的正确折叠通常需要辅助因子的帮助。金属离子是一类重要的辅助因子，真核生物中百分之三十以上的蛋白含有金属离子。然而，由于缺少一个合理的描述金属离子与蛋白质相互作用的理论模型，金属离子辅助的蛋白质折叠问题目前还仅仅局限于实验研究。在该工作中，作者们发展了一套描述金属离子与蛋白质相互作用的理论模型，在该模型中通过将量子化学的计算结果整合到经典的分子动力学中，从而能够描述金属离子与配位原子间的电荷转移，以及金属离子诱导的去质子化等效应，同时还考虑了金属离子诱导的极化效应。这一模型的建立使得基于计算机模拟研究金属离子耦合的蛋白质动力学过程成为可能。在该项工作中，基于所建立的理论模型研究了典型的金属蛋白“锌指蛋白”的折叠过程，给出了金属离子与蛋白质结合的详细路径，以及金属离子的结合在蛋白质折叠中的作用。发现锌离子不仅可以稳定锌指蛋白的天然态结构，并且参与整个蛋白质折叠过程，调控蛋白质二级结构的折叠顺序与相对稳定性。这种金属离子在蛋白质二级结构折叠中的作用是由疏水核的形成所介导。同时，计算结果还表明锌指蛋白折叠过程中存在错误配位与配体交换现象。该工作从原子的层次揭示了锌离子在锌指蛋白折叠中的作用以及锌离子的结合与蛋白质折叠相耦合的物理机制，对金属辅助因子诱导的蛋白质折叠的一般规律有了深入的认识。所建立的理论模型可以用于金属离子诱导的蛋白质折叠、聚集以及其它功能运动等常规分子动力学模型无法描述的蛋白质动力学过程。





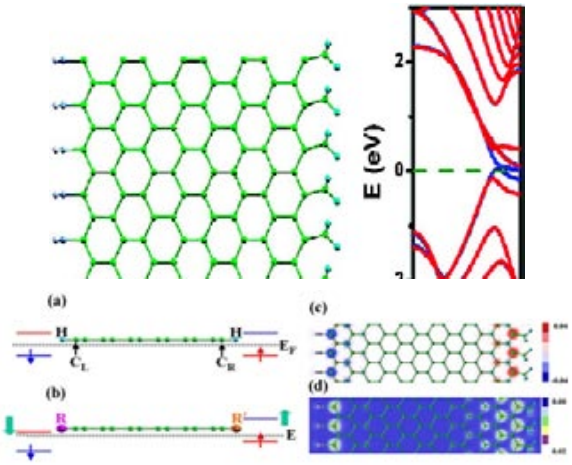
论文题目：边缘修饰的锯齿型石墨烯纳米带的半极金属性

项目来源：国家自然科学基金、国家重点基础研究发展计划（973）

用户来源：中国科技大学微尺度物质科学国家实验室

论文来源：美国化学会志第130卷第13期，页号：4224-4225，2008年4月2日出版

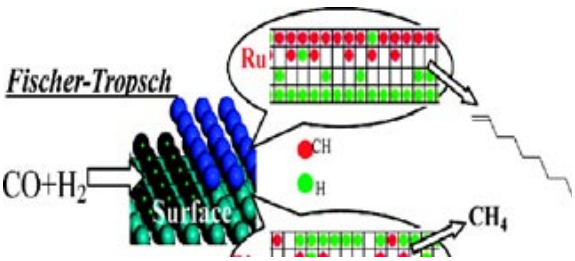
本项工作在石墨烯纳米带的半极金属性研究中取得了重要进展。基于理论计算研究，作者们提出一种新的方法使石墨烯纳米带成为半极金属。石墨烯材料因为其特殊的性质以及在下一代电子学器件应用中的巨大潜能，成为了国际上的研究热点。而一个自旋通道为金属另一个自旋通道为半导体的半极金属是自旋电子学器件中的一种重要材料。加州伯克利分校的Louie研究组最先提出了利用施加横向电场来调节锯齿型石墨烯纳米带电子结构，从而达到实现半极金属的目的。该研究组在早期的研究中指出，通过施加横向电场的方法，因为要求的临界电场强度高，不利于实际应用。在此基础上，作者们设计了一种基于化学修饰的方法来获取半极金属的纳米带。他们通过在石墨烯纳米带的两边修饰不同的功能团：NO2和CH3，利用不同功能团对边界态的影响，成功地实现了在一维单层石墨烯纳米带上获得半极金属性质的目的。进一步的研究显示，通过降低边界修饰功能团的浓度，可以提高体系的稳定性，从而获得更易实现的半极金属材料。



图(a-b)：理论设计的原理。计算得到的石墨烯纳米带的(c)自旋密度和(d)电荷密度。

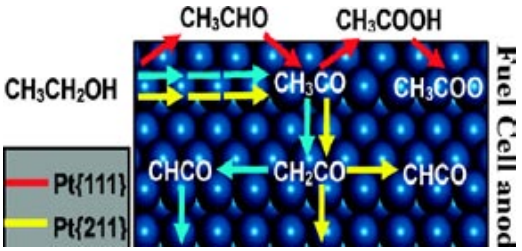
论文题目：通过第一性原理的统计力学研究了解钌和铑上Fischer-Tropsch反应选择性开关
项目来源：国家自然科学基金、浦江人才计划、上海市科委基金
用户来源：复旦大学化学系
论文来源：美国化学会志第130卷第25期，页号：7929-7937，2008年6月25日出版

本项工作在Fischer-Tropsch反应催化机理的研究中取得了重要进展。Fischer-Tropsch反应可以将一氧化碳和氢气的合成气转化为具有较高分子量的碳氢化合物，这样就可以通过这个反应路径，利用天然气或煤来制造燃料或具有更高价值的有机化合物。因此Fischer-Tropsch反应在化学工业中具有独特的地位和很高的经济价值，一直是多相催化研究中的热点问题。如何才能最大的将一氧化碳和氢气的合成气转化为长链碳氢化合物一直悬而未决。在实验上，人们已经发现了VIII族金属可以作为该反应的催化剂，但是该反应的催化机理的关键因素仍然不清楚。例如钌可以催化产生达到C20的高分子量的碳氢化合物，而铑和镍却主要产生不需要的甲烷。这种选择性的关键因素是Fischer-Tropsch反应技术的核心。在本项工作中，作者建立了一个二维晶体模型，使用蒙特卡罗模拟和密度泛函计算对Fischer-Tropsch反应在钌和铑上的动力学进行了原子级别的研究，检查了Fischer-Tropsch反应的关键基本步骤。理论计算表明，在富碳条件下，分解一氧化碳的能力是好的Fischer-Tropsch反应催化剂的关键因素；而CR + C这种以前认为在钌上的碳链增长机理也是其他过渡金属催化剂的碳链增长机理。此外，CH的势能面和CH2+H、CH3+H加氢反应能垒是Fischer-Tropsch反应过程选择性的另外两个关键因素。



论文题目：铂基乙醇氧化的详细机理和结构敏感性：解决复杂反应网络的新的过渡态寻找方法
项目来源：国家自然科学基金、浦江人才计划、上海市科委基金
用户来源：复旦大学化学系
论文来源：美国化学会志第130卷第33期，页号：10996-11004，2008年8月20日出版

本项工作在乙醇铂基氧化的反应机理上取得了重要进展。乙醇在铂上的氧化是典型的多步骤多选择性多相催化过程。彻底理解这种基础反应将极大的有益于催化剂的设计，可用于直接乙醇燃料电池的开发和生物质产生的含氧化合物的降解。在本项工作中，作者们仔细研究了在不同铂表面，包括密堆积型表面Pt{111}、台阶型表面Pt{211}、开放型表面Pt{100}，乙醇氧化反应网络。他们采用了高效的反应路径搜索方法，这种方法利用周期性密度泛函理论集成了新的过渡态搜索技术。这种新技术可以通过简单而有效的优化局部极小值得到大多数表面反应的过渡态和鞍点。作者们发现，铂上乙醇氧化的选择性极大的依赖于表面结构。这种依赖性是由两个关键性的结构敏感性的反应步骤带来的：（1）乙醇的初始脱氢；（2）乙酰基氧化。在开放性表面，乙醇倾向于通过强烈吸收中间体（CH2CO或CHCO）来断裂C-C键，从而完全氧化成二氧化碳；但是在Pt{111}面，乙醇仅仅部分氧化成乙醛和乙酸。本项工作指出，开放型表面Pt{100}是在低覆盖率下完全氧化乙醇的最好表面，这一点也揭示了最近发现的铂四六麻黄纳米晶具有显著催化性能的原因。这种结构选择性的物理原因可以用热力学和动力学进行合理解释。此外，本项工作还发现决定乙醇氧化的两个基本因素为：（1）表面金属原子与不饱和含碳碎片的成键能力；（2）羟基在上表面位点相对于其它位点的稳定性。



流固耦合问题并行求解的研究

● 李政 金先龙 曹源
上海交通大学高性能计算中心 上海 200030
● 丁峻宏
上海超级计算中心 上海 201203

摘要：
流固耦合问题广泛存在于各种工程领域，本文根据流固耦合问题的计算特点与模型的结构特点，在递归坐标二分分区方法的基础上提出了基于耦合计算的负载均衡二分分区方法。采用两种分区方法对车载储液容器与空投储液容器进行并行求解，结果表明，基于耦合计算的负载均衡二分分区方法具有更好的加速比和并行效率。

关键词：流固耦合，并行计算，区域分解

1. 引言

流固耦合问题广泛存在于核动力、航空等领域。对于该问题的研究主要以实验为基础，但随着计算机性能的提高与计算方法的改进，数值模拟发挥了越来越大的作用。目前，解决流固耦合问题的数值计算方法包括差分法、有限元、边界元等方法，各种方法都需要在时间域与空间域离散进行求解，由于流体控制方程对流项的存在以及需保证流固耦合计算的稳定性和计算精度都要求采用较小的时间步长，其计算是十分耗时的。此外，随着科学研究和工程技术的不断发展，有限元模型规模也逐渐增大，尽管目前计算机的计算速度与精度不断提高，但传统的串行机已无法满足这些实际工程计算问题的需要，对流固耦合这类问题采用多CPU的并行计算是十分必要的[1,2]。

目前，针对结构问题或者是流体问题的并行计算研究很多，而针对流固耦合分析的并行处理还处于起步阶段，文献亦不多。Johan等[3,4]在分布存储器的并行多处理器系统上研究了流固耦合问题在各处理器间信息传递问题，采用了RSB的分区策略，使得流固耦合的并行计算得以实现。Ray等[5]在共享存储器的并行多处理器系统上采用METIS程序包对网格进行分区，研究了流固耦合问题的并行计算。Vinay等[6]将流体域与结构域的单元均匀的分配到各个处理器上来加快流固耦合问题的并行计算。

论文针对车载储液容器与空投储液容器计算特点，在Dawning4000A计算平台上，提出了基于耦合计算的负载均衡二分方法，对流固耦合模型进行合适的分区，保证了各节点计算量的均衡，从粗粒度上实现了流固耦合问题的并行计算，并研究了算法的可扩展性，极大地提高了算法的并行度。

2. 流固耦合问题的并行分析

基于区域分解的并行有限元方法是当前并行计算最活跃的研究和应用领域之一，其主要研究内容是网格的划分和基于其上的数据分割。有限元网格的分区方法，对并行计算的效率高低有着决定性的影响[7-9]。目前主要的区域分解算法主要有模拟退火法、贪婪方法、递归谱二分法、递归坐标二分法(Recursive coordinate bisection algorithm，简称RCB)等。而针对网格划分算法的软件包主要有Chaco、JOSTLE、METIS、PAEMETIS、PARTY、SCOTCH和S-HARP等，它们从静态划分、动态划分、并行划分等角度提出了自己的算法，并都有一定的科学应用价值。

递归坐标二分法是较常用的区域分解方法，也是研究所采用的显式动力学软件LS-DYNA所默认的方法，该方法属于几何模型分区方式，是由Berger和Bokhari于1987年提出来的[10]。在该方法中假设所用处理器的个数是2的次方（如果不是，则分区失败），所以又可以称为二元区域分解法。该方法概念简单，分区过程快速、不费时等特点。但是，尽管递

归坐标二分方法可以使得各个分区的单元数量大致相等。但是由于该方法只考虑有限元模型的几何信息而没有考虑模型的载荷分布和计算类型特点，分区的质量一般，尤其对于储液容器这类结构复杂，具有鲜明计算特点的流固耦合问题并不是高质量的分区方案。

研究针对流固耦合问题的特点，在递归坐标二分方法的基础上，结合上海曙光4000A巨型机的体系结构平台，提出了基于耦合计算的负载均衡二分方法（Interaction load balanced bisection, 简称ILBB），该方法应满足如下分区规则：

（1）分区内流体-结构耦合计算负载尽可能相等。

（2）分区内接触计算负载尽可能相等。

（3）保证各分区单元数基本相等。

（4）减小分区间公共节点数，即尽可能减少分区间通信。

（5）每个子域尽可能快的收敛。

为了评价并行计算效果，采用了并行计算效率η与加速比Sp的概念，其公式定义如下

$$S_p = \frac{t_s}{t_p} \quad (1)$$

$$\eta = \frac{S_p}{N} \times 100\% \quad (2)$$

其中，t_p并行计算执行时间，t_s为采用一个处理器上执行时间

3. 车载储液容器的并行计算

车载柔性储液结构以其适应性强、机动灵活等特点广泛应用于各个行业。储液结构与容器内液体间的作用问题为典型的流固耦合问题，本研究以某型号的车载储液容器为例，该系统模拟包括储液容器、法兰、拉带和货箱底板四部分，模型几何尺寸与材料参数均采用实际数据，同时考虑材料变形与空气的影响，建立空气网格，有限元模型如图1所示，单元总数目为32，880，节点数目为36，437。储液容器与货箱底板采用采用了Lagrangian描述的壳单元，而储液容器内部的流体采用了多物质ALE描述的实体单元。结构与流体间采用了基于罚函数的耦合方式，而储液容器的接触部分（包括储液容器与法兰、储液容器与拉带、储液容器与货箱底板）采用了罚接触方式。

论文基于多物质的ALE有限元法，采用动力学软件LS-DYNA完成了车载储液容器装卸过程动态响应计算，主要模拟了货箱底板翻转30度的过程。该模拟中，通过拉带将储液容器固定于货箱底板之上，然后给货箱底板施加以翻转角速度曲线，除货箱底板后部可以转动外，其它位置均自由。

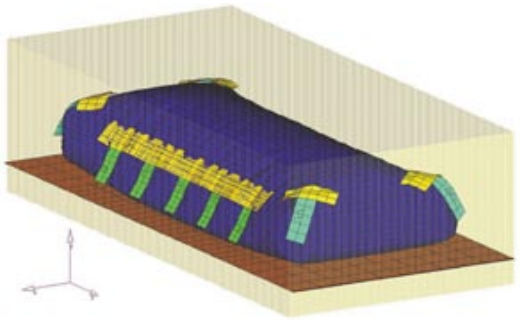


图1 车载储液容器有限元模型

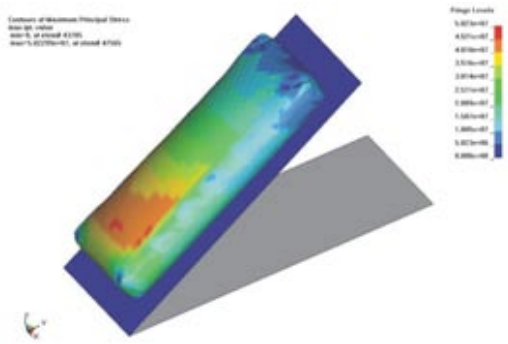
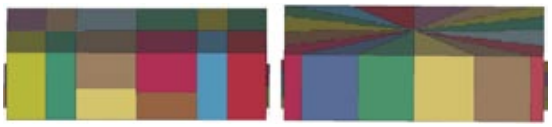


图2最大主应力云图

图2为储液容器的应力分布云图，可见，在储液容器翻转30度后，储液容器内部的水体堆积于后部，导致该处产生应力集中，这与实际物理试验是相符合的。



(a) RCB (b) ILBB

图3 两种分区方法16分区结果

研究采用RCB与ILBB两种分区方法对模型进行分区，并通过1，2，4，8，16个CPU进行计算，图6为两种分区方法得到的16分区拓扑结构图，可见两种分区方法得到的拓扑结构是不同的。

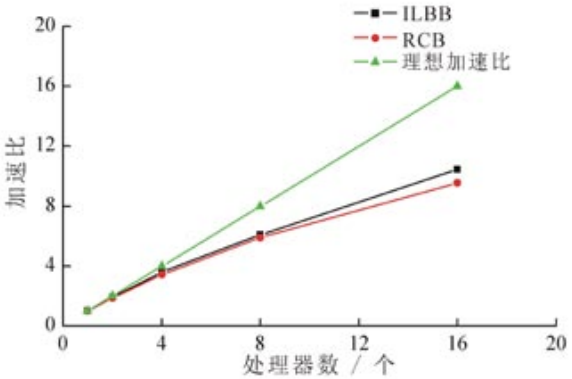


图4 两种方法下的加速比比较

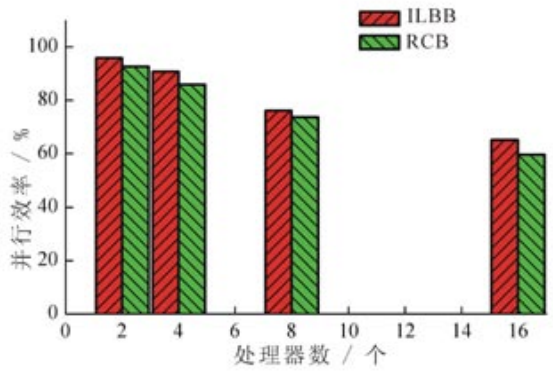


图5 两种方法下的并行效率比较

图4与图5比较了两种分区方法下的加速比与并行效率。可见，ILBB方法的加速比与并行效率都比RCB要高，相对于RCB法，ILBB法能节约较多的计算时间，提高计算效率。这主要是因为RCB分区保证了分区内单元数量基本相等，但未考虑流体与结构间耦合计算的影响；而ILBB法在保证单元数量基本相等的同时使得各分区内参与耦合计算的单元数量基本相等，这就使得各分区内耦合特性相似，负载基本相当。同时可见两种方法下，计算并行效率都随着CPU数目的增多而下降，这主要是由于CPU间的通信量增大造成的。

4. 空投储液容器的并行计算

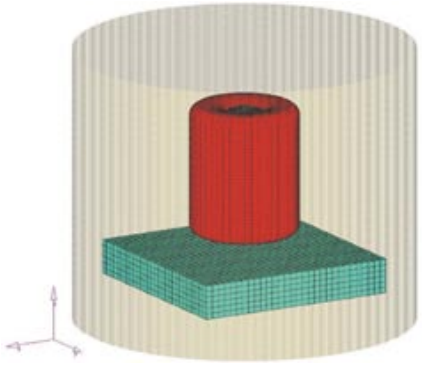


图6 储液容器有限元模型

空投储液容器因其方便灵活、不易破损的特点可用作储运或紧急救援设备。某型号空投储液容器主要由罐体、罐口、增阻装置等组成，因碰撞瞬间增阻装置作用可忽略，模型只考虑了罐体与罐口两部分。根据实际尺寸建立有限元模型（如图6），单元总数228,033，节点总数238,484。

论文同样基于多物质的ALE有限元法，采用通用动力学软件LS-DYNA完成了空投柔性储液容器动态响应计算。该模拟中，空投高度为100m，装水量为80%。为缩短模拟计算时间，假定空投储液容器以某一初始速度，自0.05m处落下。在忽略风阻系数的情况下，可设储液容器的初始速度为44.26m/s。

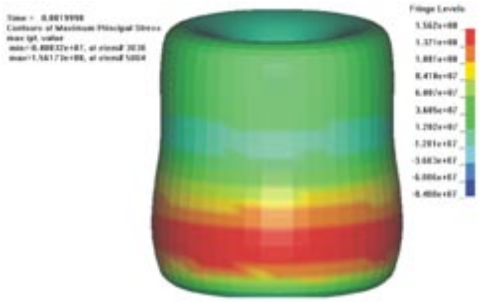
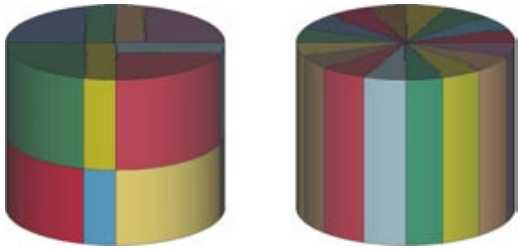


图7 罐体应力云图

图7为空投储液容器的应力分布云图，可见，在储液容器与地面碰撞后，由于水体的向下冲击作用使得罐体下部向外碰撞，从而产生了应力集中现象。



(a) RCB (b) ILBB

图8 两种分区方法16分区结果

研究同样采用RCB与ILBB两种分区方法对模型进行分区，并非别通过1，2，4，8，16个CPU进行计算，图8为两种分区方法得到的16分区拓扑结构图，可见其拓扑结构是不同的。

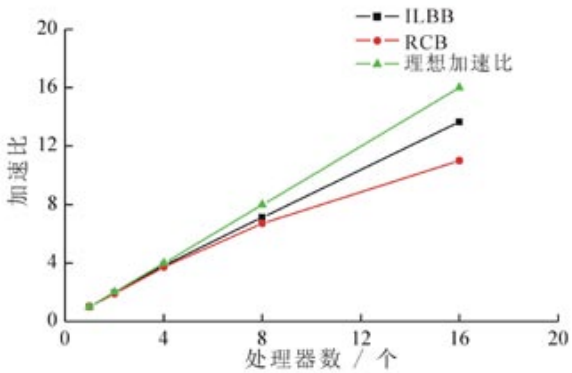


图9 两种方法下的加速比比较

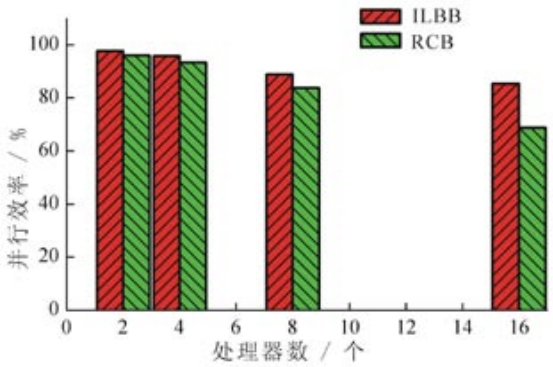


图10 两种方法下的并行效率比较

图9与图10比较了两种分区方法下的加速比与并行效率。可见，ILBB方法的加速比与并行效率都比RCB要高。这也是因为ILBB法考虑了模型的计算特点，而RCB只是仅仅保证分区内节点的相等。

同时通过与车载储液容器并行结果相比较可见，ILBB分区方法对不同模型加速比提高的效果并不相同，显然该方法对空投储液容器加速比提高的效果好于对车载储液容器加速比提高的效果，这主要是因为空投储液容器的结构相对于车载储液容器的结构要规则一些，而ILBB本身为粗粒度的分区方法，所以造成了对空投储液容器的分区效果更好一些。同时也说明了ILBB分区还不完善，需要进一步的加以改进。

5. 结论

针对流固耦合问题的计算特点，提出了基于耦合计算的负载均衡二分分区方法，通过与坐标递归对分方法相比较，证实提出的方法具有更高的加速比与并行效率。但是该方法为粗粒度的分区方法，对于不同的计算模型，其提高的效果并不一致。

参考文献：

[1] 岳宝增, 刘延柱, 王照琳. 求解液体大幅晃动问题的数值方法评述. 上海交通大学学报, 1999, 33(6): 760 - 763.

[2] M. Ainsworth, S. J. Sherwin. Domain preconditioners for p and hp finite element approximation of Stokes equations. Computer Methods in Applied Mechanics and Engineering, 1999, 1785: 243 - 266.

[3] Z. Johan, K. K. Mathur, S. L. Johnson, et al. An efficient communications strategy for finite element methods on the Connection Machine CM - 5 system. Computer Methods in Applied Mechanics and Engineering, 1994, 113: 363 - 387.

[4] Z. Johan, K .K. Mathur, S. L. Johnson, et al. Scalability of finite element applications on distributed - memory parallel computers. Computer Methods in Applied Mechanics and Engineering, 1994, 119: 61 - 72.

[5] S. E. Ray, G. P. Wren, T. E. Tezduyar. Parallel implementations of a finite element formulation for fluid - structure interactions in interior flows. Parallel Computing, 1997, 23: 1279 - 1292.

[6] K. Vinay, E. Tayfun. A parallel 3D computational method for fluid - structure interactions in parachute systems. Computer Methods in Applied Mechanics and Engineering, 2000, 190: 321 - 332.

[7] 曹银峰, 李光耀, 钟志华. 汽车碰撞过程并行有限元仿真技术.机械工程学报, 2005, 41(2): 153 - 157.

[8] V. E. Taylor, B. Nour - Omid. A study of the factorization fill - in for a parallel implementation of the finite element method. International Journal for Numerical Methods in Engineering, 1994, 37: 3809 - 3823.

[9] H. Lama. Review of domain decomposition for the implementation of FEM on massively parallel computers. IEEE Antennas and Propagation Magazine, 1995, 37(1): 93 - 98

[10] M. J. Berger, S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocess - ors. IEEE Transaction Computers, 1987, 36:570 - 580

主机系统信息安全的分析及研究

● 薛 刚 上海超级计算中心 上海 201203 gxue@ssc.net.cn

摘要：

本文主要阐述了目前主机系统上普遍存在的安全局限性问题，结合上海超级计算中心技术支持部多年管理维护主机的实际经验，进一步分析主机系统的信息安全需求以及实现主机系统信息安全必须具备的一些功能。

前言

近年来，随着高性能计算应用的发展，以及集群技术的日趋成熟，集群主机系统的使用已经越来越普遍。通常对于企业来说，集群主机系统上的业务运行是企业的核心所在，因此如何确保集群主机系统的安全已经越来越受到关注。

从架构来看，集群主机系统由多台服务器构成，每台服务器上都安装了操作系统。集群主机系统内的任何操作都是基于这些服务器上的操作系统来完成的，可以说操作系统是这些服务器，乃至集群主机系统硬件之上最底层、最基础的东西。一旦操作系统无法保证信息安全，出现漏洞，那么黑客将很容易能够攻击集群系统里的数据文件，甚至控制整个集群系统。但从目前常用的操作系统来看，无论是LINUX、UNIX、还是WINDOWS，每种操作系统都存在或多或少的系统漏洞，这对于运行关键业务的系统而言是无法接受的。

针对开放操作系统安全性的弱点，目前常见解决方法是：在集群主机的操作系统上，增加一个安全保护层（主机信息安全层）。主机信息安全并不是简单取代操作系统的安全管理功能，而是通过从操作系统的核心层截取访问控制权限，通过对访问控制列表的匹配实现对访问权限的验证，从而加强操作系统安全性，以确保系统资源及应用服务达到企业级高安全级别，可以抵御来自内部或外部的各种攻击行为。

1. 操作系统存在的安全问题

从目前来看，集群主机系统一般使用LINUX、UNIX、WINDOWS等操作系统。这些操作系统都是开放式商用操作系统，一般来说安全级别是C2级，其设计主要考虑开放性，对安全性没有给予特别的重视，在实际应用中存在的安全问题，已经难以满足现代复杂网络环境下关键商业应用的需要。

对于开放系统的安全问题，以LINUX系统为例，操作系统本身提供了一定的用户认证和存取控制安全措施。LINUX系统对用户认证的管理主要靠/etc/passwd、/etc/shadow等用户库配合登录程序完成的，对系统文件和目录的访问控制是靠目录和文件的属性完成的。

这些安全机制存在着一些致命的问题。首先，系统管理员root和操作系统内核不受这套安全机制的限制，可以任意操作任何资源。这就为系统安全留下了很大的隐患：实际上，绝大多数的系统入侵企图都是以获取系统管理员root的权限为最终目的的，一旦获取了root的权限，就可以完全控制整个服务器。

其次，LINUX操作系统对资源的保护比较粗糙，不够细致。它仅保护了文件和目录，对进程、网络连接、终端等都没有提供足够的保护；对文件和目录的保护也仅有读、写、执行三种权限；而且对文件的权限控制由文件属主控制。这些都带来许多安全隐患。

又例如su命令存在安全问题，入侵者从别的相同系统中拷贝su命令，或者自己编写程序调用setuid系统调用，就可以轻易饶过安全保护软件。因此真正的服务器保护方案应该在操作系统级别实现，否则就可以被授权和审计绕过。

2. 信息安全系统与主机系统的关系

主机信息安全系统应该在不改变系统执行文件的情况下成为操作系统有效的一部分，在不重写操作系统内核的情况下实现安全控制功能。主机信息安全通过实时检测系统调用，收集访问请求信息进

行权限认证，决定是否允许访问的通过。以LINUX系统为例，在操作系统中有一个系统调用表，包含指向每个系统调用的内存地址的指针。应用程序对资源的访问、对硬件设备的使用、进程间的通讯都是通过系统调用接口在操作系统内核中实现的。主机系统信息安全应保存了该表中与安全有关的系统调用的指针，并把这些系统调用重定向到相应代码。

当操作系统完成初始化之后，主机系统信息安全的相应服务应该立刻启动并与系统调用表内设置挂钩，在进行用户请求的系统服务执行之前把控制传递给主机系统的信息安全系统，由其决定是否应该将服务授予用户。

例如，某个用户可能会试图访问被主机系统的信息安全所保护的资源。该访问请求向内核生成一个系统调用并打开该资源。当主机系统的信息安全截听该系统调用并决定是否应该授予访问权。如果被授予许可，那么信息安全模块就会将控制传递给正常系统服务；如果信息安全模块拒绝了许可，那么它就会将标准许可-拒绝错误代码返回给启动系统调用的程序，系统调用随之停止。信息安全数据库中的访问规则和策略应该允许管理员自定义。

3. 主机系统信息安全需求分析

由于主机系统信息安全涉及的各方面功能和技术繁多、涉及面太广，下面仅以访问控制管理功能为例，分析主机系统的访问控制信息安全技术。

访问控制管理是主机信息安全需求中最为复杂的一项功能。通过加强主机系统的访问控制，可以大大提高主机系统的完整性与保密性。

3.1 防止堆栈溢出

堆栈溢出(又称缓冲区溢出)攻击是最常用的黑客技术之一。这种攻击可能是由于某些编程语言、开放源代码程序的本质决定的。由于LINUX系统本身以及其上的许多应用程序都是用C语言编写的，C语言不检查缓冲区的边界。在某些情况下，如果用户输入的数据长度超过应用程序给定的缓冲区，就会覆盖其他数据区，包括用户堆栈。这称作“堆栈溢出或缓冲区溢出”。一般情况下，覆盖其他数据区的数据是没有意义的，最多造成应用程序错误，但是，如果输入的数据是经过“黑客”精心设计的，覆盖堆栈的数据恰恰是黑客的入侵程序代码，黑客就获取了程序的控制权。如果该程序恰好是以root运行的，黑客就获得了root权限，然后他就可以编译黑客程序、留下入侵后门等，实施进一步地攻击。

攻击者通过寻找本身有这种弱点的包含setuid的程序实现堆栈溢出攻击。这种程序可以由任何用户运行，一旦程序运行，就以root权限运行在系统中。攻击者需要知道程序运行时保存的信息在系统中的位置，根据这些信息，攻击者运行程序，输入比系统能够接受的更多的数据。例如，假设系统缓冲区只有20Byte，攻击者输入30byte，就象在桶中装入太多的水，水会溢出流到地上一样，剩余的数据就会进入特定的系统空间。经过精心设计，攻击者就可获得root权限。这种弱点一旦被发现并贴在Internet上供他人使用，这种攻击方式就会一直存在，而且适用于其他类似系统，直到弱点被消除。这种攻击依赖于系统软件的弱点，而且数据溢出后必须进入堆栈的同一位置。80年代著名的“蠕虫”病毒就是利用LINUX系统的login程序的弱点，使用“堆栈溢出”发起攻击的。

3.2 用户认证登陆

用户认证和授权是给合法用户相应权限，禁止非法用户访问的第一步。以LINUX系统为例，操作系统对用户的认证是通过用户名和口令完成的，用户相关信息记录在/etc/passwd和/etc/shadow等文件中。在用户通过终端登录、使用rlogin、telnet远程登录或者使用rsh、ftp等协议访问系统时，这些软件都要利用这些文件对用户的合法性进行认证。对用户的认证是通过这些应用程序完成的，与操作系统内核无关。实际上，内核只通过进程uid和gid区分不同的用户，并不能识别什么是登录过程。

有些安全控制软件是使用新的登录守护程序(如telnetd、login等)代替原有程序实现对用户认证的增强的。这种实现方式需要针对不同的登录过程开发新的版本，往往无法覆盖所有的登录过程。而且，一旦原有程序被拷贝回来，就绕过了安全控制软件。

所以主机系统的信息安全模块必须可以识别不同登录过程中使用的系统调用序列来拦截用户登录过程，例如系统缺省带有的所有登录过程，login、telent、tlogin、ftp等，以完成用户认证控制的。

此外，主机系统的信息安全模块还应提供以下几种关于用户登陆的限制，避免因用户的口令质量不高，就是其口令被窃取，“黑客”也很难侵入系统。

1) 限制用户登录的终端

这样，特定用户就只可以从特定终端或特定客户机登录系统，即使用户口令被窃取，黑客也无法利用该用户从其他终端登录，大大增加了入侵的难度；

2) 限制用户登录的时间

只有在规定的时间段之外，即使有合法的帐户和口令，也无法登录系统。

3) 限制用户登录的次数

通过限制用户的登录次数，既可以限制用户不能从两台以上终端同时登录(在一台终端上可以同时登录多次)，也可以限制用户总共仅能同时登录一次。

4) 限制错误口令输入次数

当超过规定的次数，用户的帐户自动锁死。以后即使提供了正确口令，也无法登录系统。这样就防止了“黑客”猜测口令、进行攻击的可能性。

3.3 口令质量控制

对系统的安全威胁有许多种，但是威胁最大的是普遍存在的口令选择不当所构成的威胁。实际上，当前所有的缓冲区溢出、输入确认攻击和数据驱动的攻击等带来的威胁总和，仍然无法与易于猜出的口令所构成的威胁相比。

最常见的口令有四类：没有口令、用户名作口令(不管是同样或不同的用户)、公司名称和字典中的单词(如口令password或夏天summer)。如果采取艺术方式来推断正确的口令，研究用户的姓名、背景资料及在公司中担任的职务常常可以推断出明显的口令。例如，如果目标是CFO，则口令很可能是money、bill、finance或stocks。一旦获得用户名，对攻击者来说，最大的困难就解决了。实际上，90%以上的时间，管理员只消试试Password一词(或者根本就不用试任何口令)，就能进入网络。

因此，草率选择的口令无疑是最大的安全隐患。如果用户能将他们的口令设置成的较为复杂，就能消除很大安全隐患。但是，仅靠行政规章制度来规范口令的质量，往往达不到预期的效果，因为人的习惯和惰性往往使用户倾向于使用简单的单词作为口令。所以管理员需要规范口令质量，从而消除最大的安全隐患。主机系统信息安全必须实现如下口令规则：

- 口令的最大长度和最小长度
- 禁止使用用户名、某些特定词做口令
- 规定口令的组成，如至少几个字母、几个数字、几个特殊字符等
- 口令的使用周期，多长时间用户必须改变口令
- 口令的历史，在多长时间内口令不能重复，新旧口令之间必须有多大差别等

3.4 文件和目录的权限控制

1) 以控制列表的方式控制文件的权限

主机系统的信息安全模块应该可以采取访问控制列表的方式确定用户对文件的权限，因此每一个不同的用户都可以对文件有不同的权限，而不仅仅限于属主、同组者和其他三种情况，以增加控制的灵活性。

LINUX系统自主存取控制是以”Owner/Group/Others”形式进行的，对每一个文件或目录的访问者分为文件的主人用户(Owner)、同组用户(Group)和其他用户(Others)三类，这三类用户对该文件可以有读、写、执行三种访问权限。文件的属主、所属用户组以及访问权限都作为文件的属性保存在文件的描述之中。操作系统内核在访问文件时，把访问进程(用户)的uid和gid与文件的相应属性匹配，确定该进程对文件的存取权限。

2) 扩展文件的访问权限

LINUX系统对文件的读、写、执行三种权限并不能真正反映复杂业务系统的需要。例如，某些业务人员的职责是进行日常交易，反映在业务系统中可能是更新(读、写)某些文件；但文件的建立和维护应该是由系统维护人员进行的，业务人员不应删除这些文件。但是在LINUX系统中，对文件有写的权限，就能删除该文件。如果该业务人员由于误操作或心存不轨，删除了这些文件，就会对业务造成不可弥补的损失。如果主机系统的信息安全模块可以实现将文件的访问权限扩展为读、写、执行、删除、改模式、改属主等多种，对目录的访问权限扩展为进入、搜索和删除，就可以大大增加了控制的细致灵活程度。

3) 对关键文件的额外的保护

主机系统的信息安全模块还应提供了对关键文件的额外的保护措施。通常一些系统或应用的配置文件是很少变化的，“黑客”入侵以后要留下“后门”或进行进一步破坏，很可能要修改一些系统配置文件。因此，这些文件的变化很可能预示着有黑客入侵或系统的损坏。但是，操作系统本身没有一种机制监视这些文件的变化，管理人员有不可能经常察看这些文件，确定其变化情况。

3.5 保护特权程序

在LINUX系统中，有些情况下某用户需要访问他无权访问的文件。例如，passwd(属主为root，所有用户都有执行权限)程序是修改用户口令的程序，任何用户都可以执行它修改自己的口令。但是，修改口令需要更新(写)口令文件/etc/passwd(属主为root，其他用户没有写权限)。当passwd程序运行时，该进程的uid是该用户，所以它没有修改/etc/passwd文件的权限，也就无法实现更改口令的任务。也就是说，用户对/etc/passwd无写权和他应该能够更改该文件中他的口令字段之间存在着矛盾。LINUX系统是采用特权

程序的方式解决这一矛盾的，passwd文件属性中有一个s(setuid)标志(用ls -l命令可以看到)，表示该程序为特权程序。特权程序在执行时，进程的有效uid被设置为程序的属主。也就是说，无论passwd程序由哪个用户运行，该进程的有效uid都是root，所以就哪个修改/etc/passwd文件，从而实现修改口令的功能了。

因此，特权程序的特点是可以由任何用户运行，一旦程序运行，就以root权限运行在系统中。如果这种程序存在问题，就有可能使一般用户获得root权限，对系统安全构成威胁。

所以主机系统的信息安全模块必须对特权程序提供额外的保护。管理员可以定义哪些特权程序可以执行，哪些用户可以使用哪些特权程序。这样，只有安全管理员确认的特权程序才可以运行，其他不被信任的特权程序不能执行，就减少了特权程序带来的威胁。

3.6 保护进程

LINUX系统中没有对进程的保护措施。进程的属主和root可以用kill命令杀死正在运行的进程。实际上，一些关键的进程如数据库守护进程、应用程序进程等应该一直运行，不应该被杀死。

所以主机系统的信息安全模块要提供对进程的保护，以截取发向进程的sigkill、sigstop和sigterm信号。被保护的进程可以正常或异常退出，但是不能被非授权的用户(包括root)杀死。这就保护了误操作造成的关键进程的异常中止，保障了系统的可靠性。

3.7 对su命令的保护

LINUX系统中有一个su命令，使用该命令可以把用户临时变为另一个用户，以执行该用户的任务。root在su成其他用户时不需输入口令。实际上，su命令是通过setuid系统调用完成的。主机信息安全系统通过拦截setuid系统调用，完成对用户间临时切换的控制。

主机系统的信息安全模块应该提供把uid和gid定义为资源的功能，这样就可以规定哪些用户可以临时变为哪些用户。即使是root，如果没有得到授权，也不能su到其他用户。这就加强了系统的安全性。

3.8 保护网络连接

主机系统的信息安全模块必须可以控制本计算机提供的网络服务的服务对象，根据客户机的情况允许或禁止某些计算机访问本服务器的某些应用，同时还应保护向外的连接。

3.9 取消“超级用户”

在LINUX系统中，root用户具有非常特殊的地位。操作系统内核识别uid＝0、gid=1的超级用户，使其不受资源权限限制，以最大权限访问所有资源。但是在实际的商用模型中，系统管理员的权限不应该包括窥探和篡改业务数据。所以，商用需求和技术实现之间就存在这样的矛盾。

更为严重的是，由于root帐户的特殊性，某人一旦获得了root权限，就可以改变审计记录，抹去他的活动记录。对于攻击者来说，root变成了一个有效的目标，一旦得到root权限，就可为所欲为。攻击者可以通过各种途径成为root，如猜口令，利用拙劣的系统配置，利用缓冲区溢出等等。一旦突破了root，他们就可以完全控制系统，影响可用性，改变内容，删除数据，开始进攻其他系统，留下后门等等。

而且，在现在的实际生产环境中，为了使软件易于安装和实现，许多应用软件都需要以root用户运行，许多提供Web，mail等网络服务的程序也需要以root运行。这就造成了root的滥用，这些软件中只要有一个存在安全漏洞，就会被黑客利用，完全控制计算机。

所以主机信息的安全系统模块必须提供超级用户无法访问未经授权的文件。这样既可以防止一个单位的敏感数据(如财务、人事等)，也可以防止由于超级用户误操作而给系统带来的损失。同时，主机系统信息安全模块还应该提供把不同的权限分配给不同的角色功能，从而分散了root的权力，如定义：安全管理员、审计员和口令管理员三个角色。安全管理员制定、实施安全策略、建立、修改和维护用户属性；但是不能修改用户的口令，也不能改变策略的审计属性；这两部分工作分别有口令管理员和审计员负责。这三个角色互相制约，任何一个人的操作都可以得到其他人的监督，共同完成安全管理功能。这样，即便是操作系统的超级用户，也无法超越授权访问资源，从而解决了上述商用需求和技术实现之间的矛盾。而且，即便“黑客”攻破了root，也无法突破主机信息安全系统的安全策略，访问敏感资源。

3.10 提升普通用户权限级别

在LINUX系统中，有些系统功能是只能由root用户完成的。例如mount一个光盘这样涉及硬件的操作，只能以root的身份才能进行。但是在现在的应用环境下，这样的任务往往需要其他用户也能完成。如系统管理员不在的时候，业务要求其他人也能够完成该操作，以保证系统正常运行。通常情况下，遇到这种情况，系统管理员只好把root口令告诉其他人，让他们以root登录，才能完成这样的操作。

root口令的随意散播就给系统的安全造成了极大的隐患。

主机系统的信息安全模块应该提供一个安全的方法代替root授权。管理员定义哪些原本只能由root执行的任务现在可以有哪些用户执行。例如，定义用户smith可以mount光盘。例如用户smith通过该实用程序提交mount光盘的任务，该程序在经过主机信息安全系统的授权后，就会以root的身份帮助smith完成该任务，从而避免root口令的扩散。

4. 主机系统信息安全必须具备的其他功能

4.1 主机安全策略集中管理、部署功能

主机系统的信息安全必须提供安全策略集中管理、部署功能，管理员只需要一台服务器，就能够实现集群主机中的所有服务器安全管理以及安全策略的部署，这样基于策略的安全事件管理方法可以大大提高效率并降低安全运维管理成本。

管理员可以登录到安全策略模型数据库上，针对主机运行的各种作业具体情况和各类用户使用集群架构主机的特点，制订相应的安全访问策略，以实现主机系统安全策略的统一规范定制并快速部署到相应的集群节点上，从而简化安全管理工作，提高安全策略实施可靠性。

4.2 主机日志审计功能

日常审计功能简单说就是增强的日志功能。管理员可以看到该服务器上所有用户的与安全相关的系统调用请求。包括每个用户运行了哪个命令，该命令访问了哪些数据，产生了上面结果等都会在该实时跟踪窗口中现实出来。

通过日志审计功能，系统管理员可能及时了解用户对各种重要资源的访问行为，可以分别针对资源、人员、时间进行日志审计设置。通过主机信息安全系统的日志审计功能，管理员可以审计记录用户登录注销、授权决策事件、执行文件等事件，允许跟踪对受保护资源建立的授权访问并监控具有管理性质的活动，并将试图进入系统但违例的情况，也可以都记录下来。

主机信息安全系统的日志审计功能还拥有自我保护功能，以确保审查数据的完整，防止删除或修改。

4.3 主机事件管理以及主动报警功能

通常，为了实现违规事件的及时告警，需要一个外部的事件服务器来收集主机安全保护的告警事件，该事件服务器还可以用于收集其它安全系统送来的安全事件以进行集中的告警处理。

通过主机系统信息安全对实时安全事件告警的功能，系统可以对各种安全事件进行实时告警通知，使系统管理员能够及时发现各种违规事件，并进行及时处理，将影响降到最低程度。告警信息可以实时显示在监控页面上，或通过邮件系统或者短信网关进行接口及时告警。

小结

从目前来看，主机系统信息安全还有很长的路要走，包括对异构平台的统一管理支持、对系统资源开销的控制、对审计记录的智能分类统计等。由于对主机系统的攻击方法和技术不断变化和更新，因此对于主机系统的信息安全的研究也是需要不断研究和加强，以尽最大可能提供一个完善、受保护的主机系统。

LAMMPS介绍及其在“魔方”上的性能测试和比较

- 寇大治 上海超级计算中心 上海 201203 dzkou@ssc.net.cn
- 刘 源 中国科学技术大学高分子科学与工程系 合肥 230026 wendao@mail.ustc.edu.cn
- 王奉超 中国科学院力学研究所 北京 100190 wangfc@lnm.imech.ac.cn

摘要:

本文介绍了分子动力学软件LAMMPS的一些基本情况以及一些使用和编译的经验和方法。同时针对上海超级计算中心魔方超级计算平台，应用LAMMPS做了一系列的测试，并将测试结果与世界上其他几个高性能计算平台（包括Spirit、HPCx、Blue Gene Light和Red Storm）做了对比和分析。

1. 简介

LAMMPS即Large-scale Atomic/Molecular Massively Parallel Simulator，可以翻译为大规模原子分子并行模拟器，主要用于分子动力学相关的一些计算和模拟工作，一般来讲，分子动力学所涉及到的领域，LAMMPS代码也都涉及到了。LAMMPS由美国Sandia国家实验室开发[1]，以GPL licence发布，即开放源代码且可以免费获取使用，这意味着使用者可以根据自己的需要自行修改源代码。LAMMPS可以支持包括气态，液态或者固态相形态下、各种系综下、百万级的原子分子体系，并提供支持多种势函数。且LAMMPS有良好的并行扩展性。

LAMMPS更新很快，几乎每周都有源代码的修改。按照日期发布，每次发布两个版本，一个普通版，一个upgrade版，相当于一个stable版本和一个testing版本。LAMMPS的帮助文档一直在逐渐的增大，最近已经增加到近600页了，快速及时的更新和良好完备的帮助文档是一个软件健康发展的标志。这两方面也正说明LAMMPS是当下一款非常好的分子动力学软件包，甚至可以说LAMMPS已经是一套分子动力学的软件系统体系了。LAMMPS包含了大量的Commands，在manual中详细讲述了各命令的功能、语法格式、参数意义、注意事项、默认值等内容。

2. LAMMPS的使用

lammps做分子动力学模拟时，需要一个输入文件（input script），也就是in文件，以及关于体系的原子坐标之类的信息的文件（data file）。lammps在执行计算的时候，从这个in文件中读入命令，所以对LAMMPS的使用最主要的就是对in文件的编写和使用。下面介绍一些关于in文件的事项

- 每一非空行都被认为是一条命令（大小写敏感，但极少有命令或参数大写的）。
- in文件中各命令的顺序可能会对计算产生影响，但大部分情况下不会有影响。
- 每行后的“&”表示续行（类似fortran）。
- “#”表示注释（类似bash）。
- 每行命令中的不同字段由空格或者制表符分隔开来，每个字段可以由字母、数字、下划线、或标点符号构成。
- 每行命令中第一个字段表示命令名，之后的字段都是相关的参数。
- 很多命令都是在需要修改默认值的情况下才特别设置的。

in文件整体来看分为4个部分

1. Initialization

这一部分包含了关于计算体系最基本的信息，例如：

units: 单位系统(units style)，lammps现在提供包括lj、real、metal、si和cgs几种单位系统。

dimension: 定义了两维或者三维模拟（默认是三维）。

boundary: 定义了分子动力学体系使用的边界条件，例如周期性边界条件或者自由边界条件等。

atom_style: 定义模拟体系中的原子属性，这一命令与力场设置的参数中的原子类型（atom type）不同。

pair_style: 相互作用力场类型，例如范德化势或

者硬球势等。

bond_style: 键合相互作用势类型。

angle_style: 键角作用势类型。

dihedral_style: 二面角作用势类型。

improper_style: 混合作用势类型。

其他还有一些参数设置，例如newton, processors, boundary, atom_modify等。

2. Atom definition

lammps提供3种定义原子方式：

通过read_data或read_restart命令从data或restart文件读入，这些文件可以包含分子拓扑结构信息，这一方法在续算上也很有用。

按照晶格的方式创建原子，这种方式不包含分子拓扑信息，可能会用到例如如下的一些命令：lattice, region, create_box, create_atoms。

对已经设置好的原子可以用replicate命令复制后生成一个更大规模的计算体系。

3. Settings

原子或分子的拓扑信息定义好后，就需要制定一系列的设置，例如力场系数、模拟参数、输出选项等。

力场系数可以通过例如这样的一些命令来定义：pair_coeff, bond_coeff, angle_coeff, dihedral_coeff, improper_coeff, kspace_style, dielectric, special_bonds等。实际上力场系数也可以在关于体系的原子坐标之类的信息的文件（data file）中制定，这样具体参考read_data命令的相关介绍。

各种模拟参数可以由例如如下这样一些命令来设置：neighbor, neigh_modify, group, timestep, reset_timestep, run_style, min_style, min_modify等。模拟过程中通过compute, compute_modify, variable等一些命令来制定。而输出选项可以由thermo, dump, restart等一些命令来设置。

4. Run a simulation

通常run命令被设置在in文件的最后，使用run命令来开始一个分子动力学模拟的过程。另外，使用minimize命令来实施能量最小化计算。使用temper命令来进行复制品交换采样模拟。

其他一些重要的命令被分类列于如下：

Initialization: atom_modify, atom_style, boundary, dimension, newton, processors, units

Atom definition: create_atoms, create_box, lattice, read_data, read_restart, region, replicate

Force fields: angle_coeff, angle_style, bond_coeff, bond_style, dielectric, dihedral_coeff, dihedral_style, improper_coeff, improper_style, kspace_modify, kspace_style, pair_coeff, pair_modify, pair_style, pair_write, special_bonds

Settings: communicate, dipole, group, mass, min_modify, min_style, neigh_modify, neighbor, reset_timestep, run_style, set, shape, timestep, velocity

Fixes: fix, fix_modify, unfix

Computes: compute, compute_modify, uncompute

Output: dump, dump_modify, restart, thermo, thermo_modify, thermo_style, undump, write_restart

Actions: delete_atoms, delete_bonds, displace_atoms, displace_box, minimize, run, temper

Miscellaneous: clear, echo, if, include, jump, label, log, next, print, shell, variable

关于LAMMPS计算前后的处理问题，计算前的原子初始形态文件的生成，由read_data读入一个data文件，这个文件包括体系中各个原子的xyz坐标等等相关参数，或者由其他软件生成并修改后符合LAMMPS的输入文件格式生成。而计算后的输出，因为LAMMPS不支持图形输出，需要借助第三方可视化软件实现，例如VMD。LAMMPS的输出文件主要可以分为三种：一种是log.lammps，这里面记录了整个计算过程屏幕上显示的所有信息，可由thermo、thermo_modify等命令控制；另一种是输出应力、能量、原子位置、速度等等信息，由dump命令控制输出文件；第三种是断点续算的restart文件输出信息，由write_restart命令控制。

3. LAMMPS的编译和安装

通过LAMMPS官方网站http://lammps.sandia.gov/download.html下载源代码，本文以lammps-9Jan09（即2009年1月9日发布版本）为例介绍相应的移植工作。官方网站的帮助文档中也给出了详尽的编译指南，可以在如下的网址中找到：http://lammps.sandia.gov/doc/Section_start.html。并行环境即使用上海超级计算中心魔方超级计算平台mvapich1.1。另外，LAMMPS的编译和使用需要FFTW2的支持，这里使用的是fftw-2.1.5。

首先编译安装fftw，然后进入LAMMPS目录，其中lib目录中有两个特殊的势函数（meam和poems），编译完整版本的LAMMPS，要先编译这两个势函数，对于这两个势函数的编译需要修改相应的Makefile文件，设置相应的编译器。这之后进入src目录，对于编译完整的LAMMPS，需要先make yes-all来准备所有需要编译的文件，LAMMPS的编译和CPMD有些类似，完成这一步之后就需要修改src/MAKE文件夹内的Makefile文件，主要需要加上对fftw头文件和库文件的链接，如果完整编译还需要加入对MEAM势和对POEMS势的头文件和库文件的链接，另外设置好相

应的编译器就可以完成编译了。

4. 测试结果与分析

本节讨论LAMMPS在上海超级计算中心魔方超级计算平台上的一些测试情况。魔方超级计算机系统的计算节点采用Quad-Core AMD Opteron™-2347 /Barcelona处理器，每一个计算节点为四路四核心，单核心频率1.9GHz，L2 2M，L3 2M，bus speed 1000MHz，单节点采用4GB*16根，共计 64GB ddr2 667MHz内存，计算网络采用InfiniBand，管理网络采用千兆以太网。魔方的软件环境包括，操作系统为SUSE Linux Enterprise Server 10 (x86_64)，VERSION = 10，PATCHLEVEL = 2 (kernel: 2.6.16.60-0.27-smp),并行环境采用mvapich1.1。

LAMMPS的benchmark主要有两种：Fixed-size和Scaled-size，Fixed-size也就是固定问题规模，一般以32000个原子为算例，而Scaled-size则是指问题并行运算在P个进程时，问题本身的规模也扩大P倍，问题规模的扩大这里特指体系的扩大，也就是原子数目的扩大，因为问题本身都是在三维空间内求解的，扩大研究的原子数目就通过扩大计算盒子的体积来实现，这样最终选择的并行规模常常就为13=1、23=8、43=64、83=512、163=4096等。本文的benchmark 主要针对Lennard-Jones 溶液、Polymer chain 熔体和EAM metallic 固体三种代表较强的算例体系测试的，这三种体系都是微正则系综，32000个原子，100个分子动力学步长，不同的是原子排布方式和密度等情况，例如Lennard-Jones 溶液体系采用fcc 0.8442的面心立方格点。

同时本文还给出“魔方”平台与其他一些平台的对照曲线，这些平台的基本情况如表1所列，魔方平台的部分数据取自测试报告，其他平台的数据取自LAMMPS官方网站：

表1、各测试对照平台的基本情况

Nickname	Magic Cube	Spirit	HPCx	Blue Gene Light	Red Storm
Vendor/ Machine	DAWNING	HP	IBM p690+	IBM	Cray XT3
Processors	~25000	512	512	65536	10000
Site	SSC	SNL	Daresbury	Daresbury	SNL
CPU	1.9G Opteron	3.4G Xeons	1.7G Power4+	0.7G PowerPC 440	2G Opteron
Interconnect	InfiniBand	Myrinet	custom	custom	custom
Bandwidth	1656	230	1450	150	1100
Latency	2.5	9	6	3	7

表2 Lennard-Jones 溶液

Processors:	1	8	64	512
Fixed CPU secs:	7.005	0.9596	0.1512	0.05588
Fixed || Efficiency:	100%	91.2%	72.4%	24.5%
Scaled CPU secs:	7.005	8.315	8.979	9.212
Scaled || Efficiency:	100%	84.2%	78.0%	76.0%

图1到图6分别给出了三种不同测试体系Fixed-size和Scaled-size在五个不同平台上的对比曲线，表2-4给出了魔方平台的测试数据，而其它4个平台的测试数据获取自LAMMPS官方网站（http://lammps.sandia.gov/bench.html），本文没有重复给出。

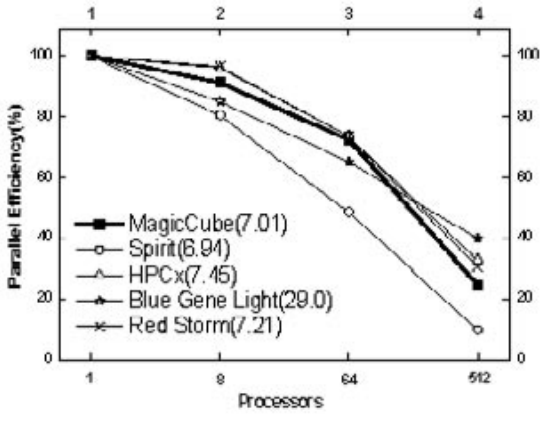


图1 Fixed-size Lennard-Jones 溶液

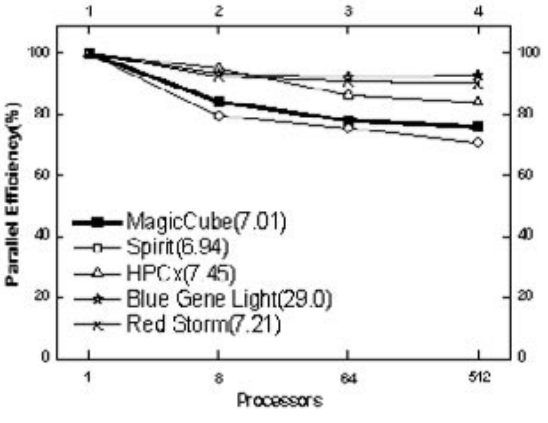


图2 Scaled-size Lennard-Jones 溶液

表3 Polymer chain 熔体

Processors:	1	8	64	512
Fixed CPU secs:	3.684	0.4334	0.07937	0.03531
Fixed || Efficiency:	100%	106.3%	72.5%	20.4%
Scaled CPU secs:	3.684	4.711	5.251	6.571
Scaled || Efficiency:	100%	78.2%	70.2%	56.1%

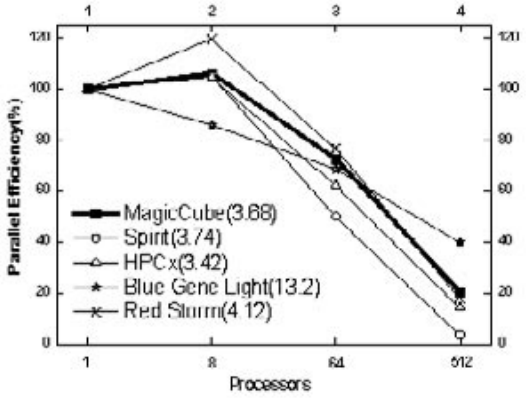


图3 Fixed - size Polymer chain 熔体

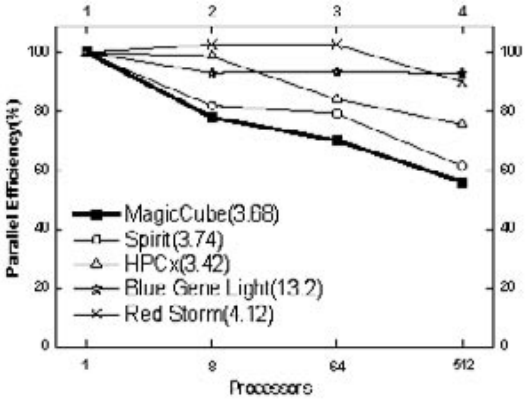


图4 Scaled - size Polymer chain 熔体

表4 EAM metallic 固体

Processors:	1	8	64	512
Fixed CPU secs:	17.24	2.431	0.3519	0.08987
Fixed || Efficiency:	100%	88.6%	76.5%	37.5%
Scaled CPU secs:	17.24	20.51	21.43	22.15
Scaled || Efficiency:	100%	84.1%	80.4%	77.8%

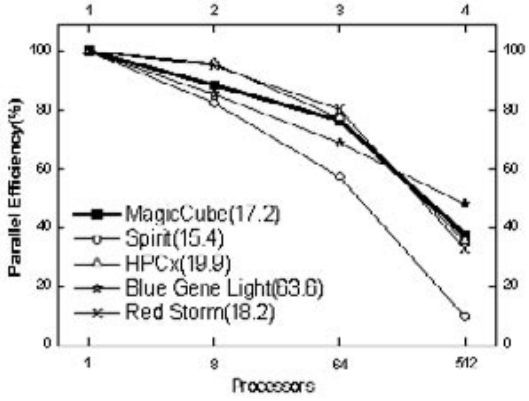


图5 FixFixed - size EAM metallic 固体

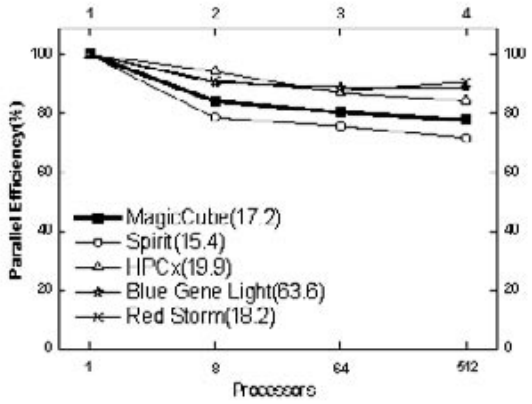


图6 Scaled - size EAM metallic 固体

由图1到图6可以看出几个平台的性能差异并不是非常大，综合来看，只有Spirit平台表现稍差，而就绝对性能来看Spirit是唯一一个超过魔方的平台，这可能和Spirit很高的处理器主频有关，而剩余的几个平台中，魔方的主频虽然不是最高的，但因为AMD Opteron Barcelona 2347处理器是新近推出的，表现出了较好的性能。而对于Spirit平台，虽然有很好的绝对性能，但作为一个超级计算平台的综合和整体来看，其较差的网络互联性能导致其综合较差的情况。Blue Gene Light因为处理器很老，主频也很低，所以绝对性能表现的很差，但其网络性能很好，就表现出很好的并行效率，而其主要的优势在于其庞大的规模。最后魔方，Red Storm和HPCx平台综合来看差异不大，Red Storm平台采用较为独特的网络互联体系结构可能占有一定的优势，而HPCx平台也以良好的网络性能取胜。

5. 结论

本文对分子动力学软件LAMMPS的基本情况以及使用编译方法等做了介绍，并针对上海超级计算中心魔方超级计算平台做了LAMMPS一些典型体系的性能测试，测试结果与LAMMPS官方网站给出的几个主要的超级计算平台的测试结果给出了比较和分析，这些平台包括Spirit、HPCx、Blue Gene Light和Red Storm，测试结果显示魔方、HPCx和Red Storm均表现出了良好的性能，且三个平台比较相近，Spirit表现出较好的绝对性能，但并行效率较差，而Blue Gene Light并行效率较好，但绝对性能较差。

参考文献：

[1] LAMMPS: http://lammps.sandia.gov

火灾动态模拟器FDS软件介绍

● 李　萍　上海超级计算中心 上海 201203 pli@ssc.net.cn

摘要：

FDS（Fire Dynamics Simulator）作为研究火灾中烟气传播规律以及火灾预防研究的开源代码，在科学研究和工程实践中得到日益广泛的应用，本文简要介绍了该软件的特点、安装平台、编译、使用方法以及注意事项，在文章末尾给出了几个典型的应用实例。

1. 简介

FDS（Fire Dynamics Simulator）是美国国家标准研究所（NIST：National Institute of Standards and Technology）建筑火灾研究实验室（Building and Fire Research Laboratory）开发的模拟火灾中流体运动的计算流体动力学软件。该软件采用数值方法求解受火灾浮力驱动的低马赫数流动的NS方程（粘性流体NavisStokes），重点计算火灾中的烟气和热传递过程。由于FDS是开放的源码，在推广使用的同时，根据使用者反馈的信息持续不断地完善程序。因此，在火灾科学领域得到了广泛应用。其源码可以从www.fire.nist.gov/fds/下载并学习。

该软件发展到现在已有25年的历史，在九十年代中期，LES（large-eddy simulation）、NIST-LES、LES3D、 IFS（Industrial Fire Simulator）和ALOFT(A Large Outdoor Fire Plume Trajectory)等代码统一被整理发展成为FDS，从2000年开始对外发布，2001年12月发布第二版，2002年12月发布了第三版，2004年8月发布了第四版，2005年发布了第五版，当前版本为5.2。

该程序源码包括25个独立的Fortran文件，每个都是模型相关的程序，比如：质量方程、动量方程、能量方程、压力求解、灭火洒水等。该软件就有很大的开放性，其源码放在特定的ftp上，即使做了小的改动，也可以在ftp上发现新文件；除此之外，专门的讨论区便于使用者交流经验与发现问题。

Smokeview是用于展示FDS模拟结果的可视化程序。

2. 软件特点

FDS自2000年公开发布以来受到了普遍的关注，据统计，该模型大约一半应用于烟气控制系统和喷头、探测器的激活启动的研究设计，另一半应用于居民和工业建筑火灾后的重建和修复设计。通过一系列的发展，FDS致力于解决火灾保护工程中的实际消防问题，与此同时，也为火灾动力学和燃烧的理论研究提供工具。

1．流体动力学模型：FDS数值求解热驱动下低速流动的N-S方程。其核心算法为显式预估校正方案，时间和空间采用二阶精度，湍流采用Smagorinsky形式的大涡模拟（LES，Large Eddy Simulation），在足够细的网格下能实施直接模拟（DNS, Direct Numerical Simulation），缺省状况下使用LES。采用拉格朗日粒子法追踪洒水和燃料喷雾模型。

2．燃烧模型：对于大多数应用，FDS采用混合物燃烧模型。该模型假设燃烧混合控制，燃料和氧气反应速度无限快。主要反应物和生成物的质量分数通过＂状态关系＂从混合物分数中得到，通过简单分析和测量的结合得到经验表达式。

3．辐射输运：辐射热传递通过求解非扩散气体的辐射输运方程得到，在有些特殊情况下采用宽带模型。与对流输运方程一样，此方程求解也采用有限体积法。此方法使用约100个离散的角，有限体积解法需要15%的计算机CPU运行时间，对于解决复杂的热辐射传导问题这个代价是适度的。水滴可以吸收热量辐射，在包含水幕喷雾的情况下是很重要的，在所有设自动喷水灭火系统的情况下都很有用。吸收系数通过Mie理论得到。

4．几何：FDS基于直线性网格求解控制方程。所以在直接建模时，要注意所建实体区域为矩形以适应背景网格。

5．多重网格：多网格用来描述计算中需使用多个矩形网格的。当计算区域的划分不可能只用一种矩形网格完成时可以设置多个矩形网格。

6．边界条件：所有固体表面都指定热量边界条件和燃料燃烧信息。通常，燃料属性储存在数据库中用名称调用。表面之间的热和质量用经验公式计算，但DNS模拟时热和质量的传导可以通过计算直接得到。

7．FDS模型除了输出各种原始数据外，还提供了多个图形输出模式，有助于直观地观察数据，如“截面文件”、“等值面”、“电热偶”及“边界条件”等。截面文件为彩色的切片，或贯穿整个控制体的断面，通过这个断面可以直观地观察气体温度的动态变化。

8．FDS5.0新增特点：FDS5在处理固体边界以及气相燃烧方面有着重要的改变。主要体现在：采用了多步燃烧能够模拟局部火焰的熄灭、CO的生成，更准确地计算热释放率；可以模拟多层材料的固体结构；更加灵活地处理洒水器、热探测器以及烟雾探测器等设备的启动以及洒水模拟启动后对火灾发展的影响；提高了多重网格能力，增加了处理背压与大气压不相同状况的能力，提高了运用MPI的并行处理能力。

3. 安装编译

3.1 软件推荐安装编译

硬件需求：FDS需要较快的CPU和充足的RAM，推荐最小配置为1GHz CPU和512MB RAM，当然配置越高越好，CPU速度决定计算需要多长时间，RAM决定可以计算多少网格，同时需要较大的硬盘空间存储输出数据，一般来说，单个计算结果文件要超过1GB，较快的网络有利于减少数据传输的延迟。

计算机操作系统(os)和软件要求：当前的FDS和Smokeview可以在Microsoft Windows，Mac OS X及各种类型的Unix/Linux安装并运行。对于MS windows系统，可以从官方网站上下载到已经编译好的可执行文件；而Linux、Unix和Mac系统推荐下载源码在本机上进行编译（Fortran 90和C编译器）进行编译后再使用，这样可以避免直接下载得可执行文件由于信息库的不兼容性造成不可用。需要说明的是，Somkeview作为后处理软件在不支持图形化的机器上是不能运行的。若需进行并行计算，需要安装相应的MPI。

如果有旧版本在该机器上成功编译并执行过，新版不用编译便可执行，若是第一次执行，则需要编译。

表1列出了目前版本包含的所有源码，源码主要是有25个Fortan 90的文件，加上用于监控输出文件的C代码 isob.c。串行版本的FDS采用main.f90编译，并行版本需要 main_mpi.f90编译，编译时从可以从网站上下载Makefile文件，编译时需要按照下表列出的顺序，对于Unix/Linux用户来说，不同平台Makefiles可以用于协助编译。

3.2 曙光4000A上的安装编译

硬件：Cluster机群，单节点四路 AMD Opteron 850, 8G RAM

操作系统：TurboLinux 8.0 64 Bit Server Edition

编译器：PGI6.0.8

通讯库： MPICH1.2.6 GM2.1.2

软件安装与编译：

1. 下载FDS的源代码（目前下载到的为：FDS5_RCB_Source_Archive.zip）以及makefile文件。

2. 拷贝下载的文件到用户自己的安装目录（~/fds），解压文件，在提示符下输入：mkdir ~/fds，通过ftp上传下载的文件到该目录下，在fds目录下建立文件夹FDS5Source，解压源码文件到 ~/fds/FDS5Source目录下。

3. 建立执行文件夹~/fds/FDS5，拷贝makefile文件到此目录下，修改makefile文件中VPATH 的值，在此处为：VPATH=../FDSSource。

4. 编译，登陆到相应的计算节点，分别编译完成串行以及并行，make target。编译完成后，生成相应的执行文件， fds5_intel（串行） fds5_mpi_intel（并行），文件名称与makefile中的输入以及所使用的编译器相匹配。

表1 FDS源码列表

File Name	Description
isob.c	C Routine for computing isosurfaces and 3D smoke
prec.f90	Specification of numerical precision
smvv.f90	Interfaces for C routines used for Smokeview output
devc.f90	Derived type definitions and constants for devices'
type.f90	Derived type definitions
mesh.f90	Arrays and constants associated with each mesh
cons.f90	Global arrays and constants
func.f90	Global functions and subroutines
irad.f90	Functions needed for radiation solver, including RadCal
ieva.f90	Support routines for evac.f90
evac.f90	Egress computations (future capability)
pois.f90	Poisson (pressure) solver
radi.f90	Radiation solver
part.f90	Lagrangian particle transport and sprinkler activation
ctrl.f90	Definitions and routines for control functions
dump.f90	Output data dumps into files
read.f90	Read input parameters
mass.f90	Mass equation(s) and thermal boundary conditions
wall.f90	Wall boundary conditions
fire.f90	Combustion routines
pres.f90	Spatial discretization of pressure (Poisson) equation
divg.f90	Compute the flow divergence
init.f90	Initialize variables and Poisson solver
velo.f90	Momentum equations
vege.f90	Experimental vegetation model
main.f90 or main_mpi.f90	Main programs, serial and parallel versions

4. 使用步骤

FDS软件包含FDS和Smokeview两部分，FDS是主体用于模拟计算，Smokeview是用于查看FDS计算结果的可视化软件。其计算流程如图1所示，具体使用的步骤为：

4.1 建立输入文件

在使用FDS5.0进行计算时，用户需要准备一个输入文件，其扩展名为.fds，在以前版本中其扩展名为.data，它提供了要考虑描述情景必要的说明。FDS输入文件用来指定工程名、计算区域的大小、网格的大小、计算时间、周围环境情况、建筑物的几何特性、材料属性、燃烧特性、固体边界条件、探测器设置、烟气特性等，以及要预期输出的计算结果，其中工程名、计算区域、网格和计算时间是最基本的设置。输入文件中每行都是以“&”开头紧接着名单群（如开头、表格、时间等等），接着是一个空格或者逗号，用来划分群组中参数列，每行以“/”结尾，下面给出简单的例子：

&HEAD CHID=’sample’, TITLE=’A Sample Input File’/

&GRID IBAR= 24, JBAR= 24, KBAR= 48/

&PDIM XBAT0=-.30, XBAR=0.30, YBAR0=-.30, YBAR=0.30, ZBAR=1.2/

&TIME TWFIN=10./

&MISC RADIATION= .FALSE./

&SURF ID=’burner’, HRRPUA=1000./

&OBST XB=-.20, 0.20, -.20, 0.20, 0.00, 0.05, SURF_IDS=’burner’/

&VENT CB=’CBAT’, SURF_ID=’OPEN’/

&VENT CB=’ZBAR’, SURF_ID=’OPEN’/

&SLCF PBY=0., QUANTITY=’TEMPERATURE’/

&BNDF QUANTITY=’HEAT_FLUX’/

在建立一个输入文件时，推荐的做法为在简单例子基础上修改。

当计算结束后，可以输出点数据、面数据、物体表面数据、等值数据和静态数据等，每种数据都包含有温度、速度矢量、压力、组分体积分数、碳黑密度、可见度、减光系数等等。并可以利用FDS后处理软件SmokeView来动态显示计算结果。

4.2 运行FDS

以版本5为例进行说明，FDS既可以单机运行（执行命令为fds5.exe），也可以mpi的并行（执行命令为fds5_mpi.exe），不管单机运行还是并行运行输入文件是一样的。单机运行在命令提示符输入fds5.exe job_name.fds，并行运算时使用得命令为mpirun -np n fds5_mpi.exe job_name.fds，其中，n为计算使用的cpu数目；job_name.fds为输入文件的名称，“jobname”代表可以确定模拟的任意特征，与计算相关的所有结果输出文件名都具备这一共有名称，除了输入文件以外，还有一些包含模拟输入参数的外部文件，其包含表述材料的参数、喷头信息等，把这些文件放在一个特定文件夹下。

由于FDS计算采用大涡模拟，计算时间较长，FDS可采用STOP文件对中途体计算运算后续算。具体做法是FDS输出目录里新建*.stop文件，当FDS监测到该文件时即建立一个PL3D文件然后停止，修改后在MIC语句里加入RESTART=.TRUE，就可以在原来停止的地方继续算了。

可以通过*.out文件监控计算的进程，在计算过程中也可能非正常结束，原因可能是数值不稳定、计算机RAM不足、因为计算机系统故障或FDS程序故障，此时会发送一个错误报告，以有助于修改输入文件或修复其他故障。

输出文件的形式：CHID.out加上各种结果文件（采用dump.f）。其中CHID.out中包含输入参数、各种变量的说明、CPU使用情况以及每100步迭代的诊断信息。

4.3 在曙光4000A上提交作业

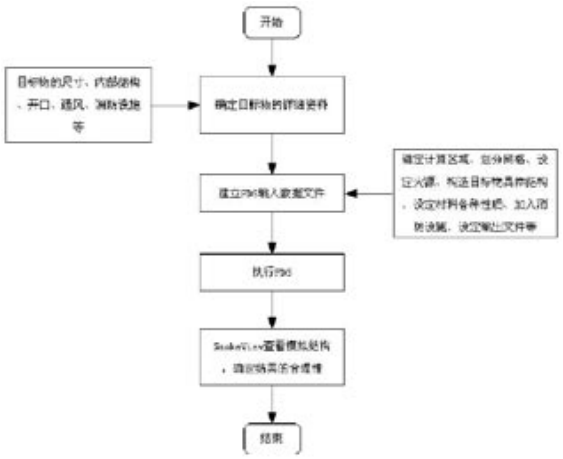


图1 FDS 计算流程图

上海超级计算中心的曙光4000A采用Platform公司的Lsf软件进行作业调度管理。提交FDS的计算任务，除输入文件之外，需要撰写一个Lsf所需的脚本，除最后一行外与其他软件的作业脚本基本相同。下面给出一个典型的实例（jobname.lsf）：

APP_NAME=esource

NP=8

NP_PER_NODE=4

RUN=”fds_mpi.exe jobname.fds”

其中，第一行用于指定所用队列（需要有该队列的权限），第二行指定了使用CPU的数目，第四行指定了运行的计算内容。

5. FDS应用限制

和其它计算流体力学软件相比，FDS有以下问题：

1.FDS建模不灵活，只能直接创建立方体模型，其它形状的模型只能用立方体模型来近似，因此会加大计算误差。

2.FDS只能划分矩形网格，而在某些情况下，其他形状的网格计算更准确。

3.FDS只能在命令行状态下运行。

另外，根据FDS所采用的计算方法及其自身性质，也可以看出：FDS是以时间平均方程求解，因此流场的湍流解析会有所出入；.FDS大都以小型火灾的实验数据发展完成，因此，对大型的火灾模拟可能误差较大；FDS仅能模拟感温探测器，对其他探测器如感烟探测器则无法有效模拟。

但，FDS作为火灾科学界比较有影响的软件，将会在各个方面不断发展和完善，必能更好的为火灾科研人员服务。

6. FDS应用成果简介

作为经过实践验证的专业火灾模拟工具，FDS不仅在高校、研究等被广泛使用，而且在企业单位也被广泛使用。目前，国外应用及成果已经非常丰富，国内的使用和研究在不断发展壮大中，清华大学、中国科技大学、上海交通大学、消防研究所等。正如软件所专注，其应用领域主要为：隧道（公路、铁路）、机场、剧院、核电厂及大型厂房等的火灾及预防的数值模拟。

要闻集锦

Lawrence Livermore国家实验室将部署20P蓝色基因/Q超级计算机

美国能源部（DOE）的两台千万亿次超级计算机Roadrunner和Jaguar 将会被一台计算能力为其十倍新的机器所取代。2009年2月3日，IBM公司和美国国家核安全局（NNSA）宣布，2011 Lawrence Livermore国家实验室将会安装计算能力达到20Petaflop的超级计算机系统，该系统将为国家核武器研究提供计算能力。

该系统基于IBM公司蓝色基因工程，今年第一季度，IBM将首先部署500T的超级计算机“ Dawn ”，接着在2011年将会部署20P下一代蓝色基因超级计算机“ Sequoia ”。该系统预期2012年上线。新系统将会取代Lawrence Livermore国家实验室目前现有的100T 计算能力的ASC Purple超级计算机和600T蓝色基因/L超级计算机，用于运行基于ASC 程序的武器方针程序。

今年部署的Dawn超级计算机主要用来进行武器代码的移植和调试。一旦蓝色基因/Q上线，那么这些代码将会在新的Sequoia系统上正式运行。Dawn现在正在Lawrence Livermore进行安装，目前已有一半左右的节点安装完成，实验室根据4月份的系统承受日程，计划将在余下的几个月内完成剩余部分的安装。

由于Dawn系统不像蓝色基因/L，蓝色基因/P和蓝色基因/Q系统都支持节点级缓存一致（这可以允许在集群系统上使用SMP类型的代码），因此，使用Dawn作为Sequoia的一个里程碑是可能而且契合实际的，特别是将武器代码的每个核的MPI任务分别进行映射的确是一个非常大的挑战，但将SMP消息传递模块整合到各个节点和跨节点的分布式并行更加实际。

Sequoia的超级计算机不仅仅在计算能力上要超出现有千万亿次超级计算机系统十倍，而且在能耗方便也有着非常大的改进。IBM Deep Computing 副总裁Dave Turek表示，Sequoia的功率大约600万瓦，其能效比为3000Mflops/watt*。这相当于蓝色基因/P系统（能效比440Mflops/watt*）的7倍，而且这也比现在Top1的Roadrunner超级计算机（其能效比为587Mflops/watt*）高。而且，计算能力为1.6Petaflop的超级计算机其功率大约为850万瓦（能效比为188Mflops/watt*）。

2011年上半年，如果Sequoia进行部署，那么Law-

要闻集锦

rence Livermore实验室的空间将会是成为额外开销，因为，现在已经有ASC Purple和Blue Gene/L部署在此，但真正的问题应该是供电。虽然新的Sequoia比现在的2台机器节能很多，但是实验室正计划将其的供电能力从1250万瓦增加到3000万瓦。

目前，IBM还未公布蓝色基因/Q的底层架构。但可以猜测Sequoia将会由98,304个节点共160万个核组成。无论采用的是单路16核处理器还是双路8核甚至4路4核处理器，每个蓝色基因/Q结点有16个核。Sequoia总内存将会达到1.6PB，因此每个节点的内存容量为16GB。这相比现有的蓝色基因/P技术的4核4GB内存有很大的提高。

Sequoia的第二个任务是支持基础科学研究，科学家们需要现在蓝色基因/L系统20到50倍的计算能力。除了为ASC Purple提供武器代码的迁移，Sequoia额外的计算能力也将达到千万亿次水平，Seager说：“这次计算能力的提升将是实验室历史上最大规模的一次跳跃。”

（卢大勇）

2009年高性能计算的五大趋势

www.wwpi.com网站2009年2月19日发表专家文章，阐述了2009年在全球经济形势低迷的大环境下，高性能计算发展呈现的五大趋势，全文内容整理如下。

1. 高性能计算的应用领域继续扩大

公司的首席信息官们（CIO）已经开始认真考虑在计算金融、工程、风险分析、商业分析以及医疗保健领域采用高性能计算。因为各大公司都被迫降低成本，所以高性能计算变得更有必要。除了高效的刀片式计算/存储平台，Windows HPC Server将替代Linux或Unix推动高性能计算应用并向各大公司提供超级计算机性能。

令人难以置信的是，Platform/Waters公司在进行调查分析后预测，金融部门的HPC采购量将在2009年里有所增加，而且Tabor Research公司的Addison Snell最近也重申了他对于HPC的谨慎乐观态度。

2. 切实倡导绿色IT

美国环保署（EPA）在2007年8月2日递交美国国会的服务器和数据中心能效报告中，根据当前趋势预测，从2007年到2011年数据中心的能源消耗将增加一倍。这其中或许有夸张的成分，但也能促使许多相关机构采取服务器整合或虚拟化等措施来节能降耗，把环境保护作为一项工作重点。

在经济消沉时期更应该提倡绿色IT，因为绿色技术既有益于环境也能帮助降低成本。IT经理可以通过提高能效、降低能源成本来节省资金的投入。根据市场研究公司IDC的调查，2005年全球各大公司在电力和冷却服务器上共花费了261亿美元。据美国能源部最近的能耗调查，这比特拉华州与佛罗里达州以及德克萨斯州之间17个州的所有商业大楼的电力消耗还要多。超过60%的电力都被用来冷却数据中心内的设备，这完全是浪费能源，所以，能够提供更强计算能力且解决了散热问题的节能HPC平台有着重大的意义。

另一个值得关注的绿色IT措施是创新的计算资源租赁。IT经理们发现，可以通过租赁计算资源来使用新的高效系统而无需增加成本支出，在今后，这种通过租赁而非购买的方式获取高性能计算资源的措施会越来越普遍。

3. 云计算/软件将真正成为一种服务/设备

2009年将有许多企业进入云计算领域，因为更多机构希望通过使用云计算服务来降低运营成本，同时增加资本资产。云计算服务有望使企业通过裁减部分或全部IT设备并从互联网获得应用程序或软件基础设施来节省资金和资源。VMware以及其它供应商现在要做的就是建立创建云计算环境所需的基础设施和应用软件。

运行这些应用程序所需的性能来自于可以运行多

要闻集锦

种商用Linux版本和微软Windows操作系统的灵活的计算平台。基于刀片的混合服务器和存储器系统非常适合处理云计算所需的大量数据，然后在维持适当成本的情况下以必需的速度进行传输。

4. 传统数据中心逐渐失去优势

当高科技的成本不断降低时，房地产和能源的价格则会飞涨，厂商们只有不断创新，才能继续发展。集装箱式机箱将成为最终的模块化数据中心，可以部署在任何能源需求较低的地方。像纽约这样人口稠密的地区已经基本没有能源和空间再部署新的设施。在输出相同的情况下，集装箱式机箱的占地面积比传统设备少70%，并且可避免修建永久性设备。集装箱式机箱已交付并准备投入使用，它将被完全应用并配置在HPC平台上。

目前，人们迫切需要更多的计算或存储设备，以至于没有多少时间来规划和建立新的数据中心，更不用说培训人员和配置设备了。集装箱式机箱是一个很好的解决方案，与建立传统的数据中心相比，它只需很少的时间就可以增加所需的计算和存储能力。这种较高的“即插即用”能力使CIO能够减少IT上的预算，进一步节省资金。另外还可以通过将机箱安放在便宜的或是可再生的能源（如风力发电厂或太阳能电池阵列）附近，以达到节省资金的目的。

5. 进入新的存储时代

日益普遍的Web 2.0技术使思想和内容更加易于表达，媒体创作和传播的权力转移到了“观众”手里，促使存储需求日益增长。今天的用户不再满足于作为内容的使用者，他们自己也创建内容。在相机、手机以及其它设备上创建的数据已经越来越普遍，人们普遍希望能够永久保存这些数据。此外，医疗、金融以及电子邮件记录等企业对存储的需求也在不断增加。

为了支持这种增长，人们不断增加处理器数量来提高HPC服务器的性能，所产生的数据量和传输量也远远高于前几年。较大的HPC站点每周甚至每天都能产生数万亿字节的数据，并因此要求数千万亿字节的存储能力。

内容寻址存储（CAS）是一项突破性的新技术，它解决了非结构性内容存档的问题。但是CAS系统并非只能用于存档。实际上，除了长期内容存档的功能外，CAS软件还能为云存储架构提供所需的性能。

尽管固态硬盘驱动器（SSD）还需要进一步改善，但它的时代即将来临。未来五年在绿色IT的推动下，SSD将成为主要存储介质。它将比传统硬盘驱动器有着更优秀的性能功耗比。此外，不同于传统硬盘，SSD可以迅速转换到低功率模式，然后很快恢复到高性能模式，中间没有等待磁头旋回的延迟。这种较低的平均能耗降低了使用SSD的数据中心对冷却系统的要求。

除了能耗优势外，固态存储器还将几个硬盘的运行性能封装在一起作为一个单独的硬盘，为现有的数据中心提供了实现更强的处理能力的机遇。

（金 溪）

高性能计算
发展与应用

（内部刊物）

Development & Application
of High Performance Computing

主　办：上海超级计算中心
协　办：江南计算技术研究所
　　上海科学技术情报研究所

编辑部：上海张江高科技园区蚌埠路585号
邮　编：201203
电　话：50801266
传　真：50801265
http://www.ssc.net.cn
E-mail:news@ssc.net.cn