

西安理工大学

硕士学位论文

高性能计算集群技术应用研究

姓名：王勇超

申请学位级别：硕士

专业：计算机应用技术

指导教师：张璟

20070301

论文题目：高性能计算集群技术应用研究

学科专业：计算机应用技术

研 究 生：王勇超

签名：王勇超

指导教师：张 璟 教授

签名：张璟

摘 要

高性能计算集群是以提高科学计算能力为目的计算机集群技术,已成为计算机领域的重要研究方向。研究高性能计算集群在科学研究和工程计算等领域应用,对于推动科技创新、经济发展、社会进步、国防安全,促进我国高性能计算应用水平的提升,具有很实际的意义。

作者研究了高性能计算、集群系统构建、基于 PC 集群的高性能计算系统开发的关键技术,基于普通 PC 机、以太网网络、Linux 操作系统,SSH,应用 MPI 等技术构建了一个由 16 个节点构成的高性能计算集群系统。利用 High Performance Linpack 基准测试方法对该集群系统进行了性能测试。在此基础上,探讨并实现了基于高性能计算集群系统的枚举排序算法,为进一步深入研究基于 PC 的高性能计算集群系统大规模应用奠定了良好基础。

本文详细介绍了作者的研究开发工作。

关键词：高性能计算；PC 集群；并行计算；Linux；MPI

Title: The Research and Realization on HPC Cluster Technology

Major: Computer Application Technology

Author: WANG YongChao

Signature: Wang Yongchao

Supervisor: Prof. ZHANG Jing

Signature: Zhang Jing

ABSTRACT

High performance computing cluster has their target on promoting scientific computing capabilities and has become a very important branch of computer science. The study on high performance cluster for scientific research and implementations is highly helpful to make progresses in technology, economy, national security, and the popularity of high performance clusters in our nation.

This paper introduces the key technologies in the implementation of high performance computers based on PC, such as high performance computing and cluster building, etc. and suggests a 16 node cluster build up by PCs, which also uses Linux OS, SSH, MPI technologies. The whole cluster was tested under High Performance Linpack testing rules. And based on the testing results, this paper discusses an enumerate-sorting algorithm used in this cluster system, which would further the study on implementation of large scale high performance clusters build up by PCs.

Key Words: High performance computing; PC Cluster;; Paralleled computing; Linux; MPI

独创性声明

秉承祖国优良道德传统和学校的严谨学风郑重申明：本人所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的成果。尽我所知，除特别加以标注和致谢的地方外，论文中不包含其他人的研究成果。与我一同工作的同志对本文所论述的工作和成果的任何贡献均已在论文中作了明确的说明并已致谢。

本论文及其相关资料若有不实之处，由本人承担一切相关责任

论文作者签名：王勇超 2007年 3月 20日

学位论文使用授权声明

本人王勇超在导师的指导下创作完成毕业论文。本人已通过论文的答辩，并已经在西安理工大学申请博士/硕士学位。本人作为学位论文著作权拥有者，同意授权西安理工大学拥有学位论文的部分使用权，即：1) 已获学位的研究生按学校规定提交印刷版和电子版学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，可以将学位论文的全部或部分内容编入有关数据库进行检索；2) 为教学和科研目的，学校可以将公开的学位论文或解密后的学位论文作为资料在图书馆、资料室等场所或在校园网上供校内师生阅读、浏览。

本人学位论文全部或部分的公布（包括刊登）授权西安理工大学研究生部办理。

（保密的学位论文在解密后，适用本授权说明）

论文作者签名：王勇超 导师签名：张琛 2007年 3月 20日

1. 前言

1.1. 课题研究背景及意义

从 1946 年 2 月世界上第一台计算机 ENIAC 诞生至今短短 60 年, 计算机技术已发生了五次飞跃。制造计算机的元器件从最初的电子管、晶体管发展到集成电路, 集成电路又从中小规模发展到大规模、超大规模。当时 ENIAC 的运算速度是每秒 5000 次加法运算, 而现在计算机的运算速度最快可以达到每秒 280.6 万亿次浮点运算^[1], 是 ENIAC 的 56 亿倍。最初的计算机仅仅应用在科学计算领域, 现在计算机的应用已经深入到了社会的每一个角落, 科研、工业、商业、农业、服务业、政府、军队、家庭、教育、娱乐等等, 从几乎无处不在, 它已经深深的改变了人类传统的工作、学习和生活方式, 计算机已经成为社会正常运转不可缺少的一部分。

虽然计算机的性能已经很高, 并且还在不断提高, 但是, 依然不能满足人类对计算性能需求的增长速度。在石油勘探、地震预测预报、气候模拟及大范围天气预报、新型武器设计和核武器系统的研究模拟、地质研究、天体和地球科学、虚拟现实系统等诸多领域都需要大量的计算来解决。但是限于处理器制造工艺, 单颗处理器性能可提高的空间已越来越小, 在这种情况下, 通过其他途径获得高性能计算能力成为一个重要的研究领域。于是, 除了继续提高单机计算性能外, 并行处理作为一种可以有效提高计算能力的手段越来越受到科技工作者的重视和青睐, 高性能并行计算也迅速成为一个非常热门的研究领域。

虽然目前具有高性能并行计算能力的商业化的超级计算机发展迅猛, 并达到了很高的水平, 但是其价格昂贵, 一般的企业、单位难以承受。随着个人计算机和互连网络设备价格的下降和性能的提高, 基于 PC 的低成本计算机集群技术迅速发展, 成为高性能计算的重要分支。

在这种情况下, 研究基于 PC 的高性能计算集群技术及其在科学研究和工程计算等领域的应用, 对于促进我国高性能计算技术的应用推广, 对于推动我国科技创新、经济发展、社会进步、国防安全, 具有深远的历史意义, 也具有很好的学术意义。

1.2. 国内外研究现状

20 世纪 50 年代末、60 年代初, 美国空军制造了一个用真空管电子数字计算机构建的计算机集群, 用以监视苏联核攻击的早期预警系统。随着计算机制造技术的飞速发展, 硬件设备逐步实现商品化和标准化, PC 机的性能越来越高, 价格越来越便宜, 同时免费开源的 Linux 操作系统内核逐步成熟稳定, 性能日趋完善, 集群系统逐步发展起来, 成为最流行的高性能计算平台, 在高性能计算机中占有的比重也越来越大。

1994 年美国航空航天局 (NASA) 的 Goddard 航天中心构建了第一台 Beowulf 式的计算机集群 Wiglaf。由 16 个 66MHz 的 486 处理器 PC 机组成, 采用 10Mb/s 以太网集线器, 速度达到每秒 7200 万次浮点运算^[2,3]。

1996 年, Hyglac 和 Loki 在美国加州理工学院和美国能源部 Los Alamos 国家实验室建成, 采用 16 个 Pentium Pro 处理器和 100Mb/s 以太网交换机, 成为第一套速度首次超过每秒 20 亿次浮点运算的集群系统。

1998 年, 采用 UltraSparc 芯片的工作站和 Myrinet 交换机的 NOW-2 速度达到每秒 480 亿次浮点运算, 成为第一套进入世界超级计算机 500 强的集群系统, 排名 113 位。此后集群系统开始了飞速发展。

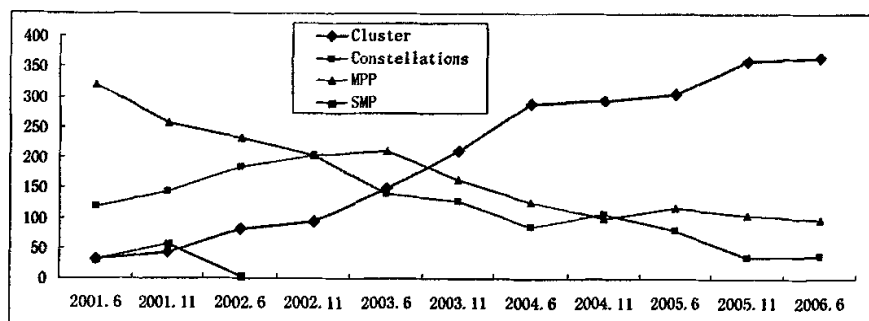


图 1-1 Top500 中计算机体系结构的变化曲线

Figure1-1 The Architecture in Top500

图 1-1 是自 2001 年 6 月至 2006 年 6 月五年中全球运算速度最快的 500 台超级计算机所采用体系结构的变化曲线。从表中可以明显看出, 到 2006 年 6 月, 有 364 台采用了集群体系, 占据了约百分之七十的份额, 而五年前, 这一数字仅仅为 32 台。毫无疑问, 集群已经成为超级计算机的发展趋势。

2004 年 6 月安装在上海超级计算中心的曙光 4000A^[4] 超级计算机 Top500 排名第 10, 是我国厂商制造的超级计算机系统至今取得的最好成绩。该系统就是一套集群系统, 它使用了 2560 颗 AMD Opteron 2.2G CPU 和 Myrinet 网络, 最高速度达到 11264 GFlops。IBM 公司制造的 Barcelona Supercomputing Center 是目前运行速度最快的集群系统, 它使用了 10240 颗 PowerPC 970 处理器和 Myrinet 网络, 最高速度达到 62630GFlops。

目前应用最广泛的集群计算技术可分为三大类: 高可用性集群技术、高性能计算集群技术、高可扩展性集群技术。高可用性集群 (High Availability Cluster, 简称 HA Cluster) 是指以减少服务中断 (宕机) 时间为目的的服务器集群技术。高性能计算集群 (High Performance Computing Cluster, 简称 HPC Cluster), 是指以提高科学计算能力为目的的计算机集群技术。高可扩展性集群技术就是带均衡策略 (算法) 的服务器集群。

目前, 计算机集群技术标准化趋势日益明显, 在安全性、智能化、可用性、可管理性、易用性、可扩展性等方面的标准正在形成。以高性能计算普及为目标、以集群标准化技术为支撑, 以 PC 计算机为主要计算节点的普及高性能计算机系统将会在大量出现, 推动高

性能计算集群技术的发展与普及应用进程。

1.3.本文主要任务

曙光 4000A 等超级计算机虽然也是集群系统，但是它所使用的硬件设备和软件系统大多都是专用的或定制的，并不是我们平时常见的普通计算机硬件设备，价格昂贵，不是一般的院所能够轻易拥有的。本课题的主要任务如下：

1. 研究集群的相关技术，如：并行技术、集群系统技术、并行计算、MPI 等等。
2. 研究集群系统的构建方法，利用普通 PC 机和以太网，在 Linux 环境下构建一个具有多个节点的高性能计算集群系统。
3. 对构建的集群系统进行性能测试。
4. 在构建的集群系统上探索并行程序开发。

2. 相关技术

2.1. 并行计算

2.1.1. 并行计算原理

并行计算的基本思想是用多个处理器来协同求解同一问题,即将被求解的问题分解成若干个部分,各部分均由一个独立的处理机来并行计算。理想情况下, N 台计算机应该能够提供 N 倍的计算能力,不论当前计算机的速度如何,都可以期望被求解的问题将在 $1/N$ 的时间内完成。但是这仅仅只是理想状态,实际上,被求解的问题在通常情况下都不可能分解为完全独立的若干个部分,而且不可避免要进行必要的数据交换和同步。尽管如此,并行计算仍然可以使整个计算机系统的性能得到实质性的改进,而改进的程度取决于待求解问题自身的并行程度^[5-8]。

并行计算的优点是具有巨大的数值计算和数据处理能力,能够被广泛地应用于国民经济、国防建设和科技发展中具有深远影响的重大课题,如石油勘探、地震预测和预报、气候模拟和大范围天气预报、新型武器设计、核武器系统的研究模拟、航空航天飞行器、卫星图像处理、天体和地球科学、实时电影动画系统及虚拟现实系统等等。

并行计算系统既可以是专门设计的、含有多个处理器的超级计算机,也可以是若干台以某种方式互连的独立计算机构成的集群。我们将要构建的集群系统就是通过以太网方式互连的多台独立的 PC 机。

2.1.2. 并行计算机分类

按照 Flynn 分类法,计算机可以分为四类,分别是单指令流单数据流(SISD)、单指令流多数据流(SIMD)、多指令流单数据流(MISD)和多指令流多数据流(MIMD),其中单指令流多数据流(SIMD)和多指令流多数据流(MIMD)属于并行计算机^[6]。

- 单指令流单数据流(SISD)

这种类型的计算机属于串行计算机,常见的计算机都属于这种类型。它有两个重要特征:串行的和确定的。

SISD 型计算机的 CPU 主要采用复杂指令系统(CISC)和精简指令系统(RISC)这两种指令系统。CISC 指令条数多达 300 多条,使用复杂的寻址方式。RISC 指令条数一般不到 100 条,且指令定长,寻址方式也只有 3 到 5 种,几乎所有的指令都基于寄存器操作,且均在单周期内执行。

- 单指令流多数据流(SIMD)

这是一种非常重要的计算机体系结构。与 SISD 不同, SIMD 型计算机虽然也是只有一个指令流,但是却可以有处理单元,可以同时多个数据流进行操作,当然,在同一时刻仍然是只有一条指令在执行。这种类型计算机的特点是:同步的和确定的。对于数据并行类的问题性能优异。

SIMD 型计算机主要有阵列处理机和流水线处理机两种。阵列处理机由功能简单但数量庞大的处理机构成,数据以某种方式流经各个处理机,由它们进行处理。主要是利用资源重复的方法开拓并行性。

- 多指令流单数据流(MISD)

这种类型的计算机很少,只有一些特殊用途的计算机可以勉强称得上是 MISD 型的,如:操作一个单信号流的多频率滤波器,或者破译一个单代码信息的多密码系统。

- 多指令流多数据流(MIMD)

这是理想的并行计算机结构,每个处理机在各自的数据流上执行各自的指令,与其他处理机完全独立。这种类型的主要特征是:

- ✓ 指令流可以同步或异步的执行;
- ✓ 指令流的执行具有确定性和不确定性;
- ✓ 适合回路、块或子程序级的并行;
- ✓ 可以按照多指令或单程序的模式运行。

多处理机和多计算机都属于这种类型,都是由一些可编程的且可以各自执行各自程序的处理机构成。

根据内存分布和存储方式的不同, MIMD 型计算机又可以分为共享存储、分布式共享存储和分布式存储三类。

- ✓ 共享存储: 又称对称多处理机 (SMP)。SMP 系统是在并行技术发展初期最常见的多处理机计算机体系。SMP 系统中所有处理机的地位都是一样的,它们连接在一起,通过某种中央开关机构访问全局共享存储器。这也正是这种类型计算机的瓶颈所在,因此 SMP 系统的 CPU 数量不能过多,一般不超过 100 个,否则处理机和存储系统之间的瓶颈将导致性能急剧下降。

- ✓ 分布式共享存储: 又称非一致性内存访问 (NUMA)。这种类型的计算机系统中每个处理机都有各自的局存,将各个局存组合起来形成一个共享的地址空间。由于不需要集中的开关机构,所以扩展性比共享存储好很多,可以连接大量的处理机,编程也比较方便。

- ✓ 分布式存储: 这种系统中每个处理机都是一个单独的节点,拥有自己的局部存储器,节点之间通过高速网络互联。每个处理机可以访问自己的内存,对于其他节点则不能直接访问,需要通过消息传递的方式进行。分布式存储主要有两种形式:

- 大规模并行 (MPP) 系统: 这是一种常见的并行系统,它解决了 SMP 系

统中处理机和存储系统之间的瓶颈,可以最大限度的增加处理机的数量。

- 集群(Cluster)系统:利用高速网络将一些高性能工作站或高档PC机按照某种结构连接起来,在并行环境的支持下,统一调度、协调处理,实现并行处理。其特点如下:
 - ◆ 每个节点都是一个独立完整的计算机;
 - ◆ 节点之间通过消息传递的机制进行联系;
 - ◆ 可以最大限度的增加节点数量;
 - ◆ 节点机依赖于现有的成熟的技术;
 - ◆ 通信网络的性能(数据传输速度、带宽、延迟)是影响集群整体性能的关键。

据近几年世界超级计算机 Top500 排名的情况,在 Top500 中, SMP 系统从 2002 年 6 月以后就消失了, MPP 系统也逐步淡出, Cluster 系统从五年前的仅仅 32 台突飞猛进,到 2006 年 11 月,已经占据了超过 70% 的份额。可见, Cluster 系统具有明显的优势,代表了今后并行计算机的发展方向。

2.1.3.并行编程模型

为了简化并行程序设计,需要采用合适的并行编程模型。目前两种最重要的并行编程模型是数据并行和消息传递^[9-10]。数据并行编程模型的编程级别较高,编程相对简单,但只适用于解决数据并行问题;消息传递编程模型的编程级别较低,编程相对复杂,但却有着更加广泛的应用范围。

数据并行指的是将相同的操作同时作用于不同的数据,从而提高问题求解速度。数据并行很早就被应用于向量计算机。长期的实践表明数据并行可以高效地解决大部分科学与工程计算问题。数据并行模型是一种较高层次上的并行计算模型,它提供给程序员一个全局的地址空间。通常这类模型所采用的语言本身就提供有并行执行的语义,因而程序员只需要简单地指明执行什么样的并行操作和并行操作的对象,就实现了数据并行的编程。

数据并行模型虽然可以解决许多科学与工程计算问题,但对于非数值计算类问题,如果仍通过数据并行的方式来进行,则难以取得较高的效率。目前,数据并行模型面临的主要问题是如何实现高效的编译。只有具备了高效的编译器后,数据并行程序才可以在共享内存和分布式内存的并行计算机上取得高效率,才可能提高并行程序设计的效率和可移植性,从而进一步推广数据并行程序设计技术。

在消息传递模型中,各个并行执行的任务之间通过传递消息来交换信息、协调步伐、控制执行。消息传递一般是基于分布式内存的,但同样也适应于共享内存的并行计算机。消息传递模型为程序员提供了更加灵活的控制手段和表达形式,一些用数据并行模型很难表达的并行算法,采用消息传递模型则可以很容易地实现。机动灵活和控制手段的多样化,

是消息传递模型能提供很高执行效率的重要原因。

消息传递模型为程序员提供了尽可能大的灵活性,但同时也将各个并行任务之间复杂的信息交换及协调控制交给了程序员,从而在一定程度上加重了程序员的负担。尽管如此,消息传递模型的基本通信模式仍然是简单和清楚的,目前大量并行程序采用的都是消息传递并行编程模型。

2.2.MPI 消息传递并行编程模型

由于消息传递并行编程模型的广泛应用,目前已经出现了许多基于该模型的并行编程语言,其中最流行的是 PVM (Parallel Virtual Machine) 和 MPI (Message Process Interface)。

2.2.1.PVM

PVM 是第一个将工作站集群作为并行计算平台并被广泛接受的并行编程语言。它由美国的 Oak Ridge 国家实验室、Tennessee 大学、Emorg 大学、CMU 大学等联合开发而成,能够将异构的 Unix 计算机通过异构网络连接成一个“虚拟”的并行计算系统,为其上运行的应用程序提供分布式并行计算环境^[6,11]。

PVM 是一种基于局域网的并行计算环境。它通过将多个异构的计算机有机地组织起来,形成一个易于管理、易于编程、并且具有很好的可扩展性的并行计算环境。目前 PVM 支持 C 和 Fortran 语言。程序员首先参照消息传递模型编写好并行程序,然后将编译后的程序以任务为单位在网络中特定类型的计算机上运行。PVM 能够在虚拟机中自动加载任务并运行,并且还提供了任务间相互通信和同步的手段。由于所有的计算任务都被分配到合适的计算节点上,多个节点并行运算,从而实现了任务一级的并行。

2.2.2.MPI

MPI 是为开发基于消息传递模型的并行程序而制定的工业标准,其目的是为了提高并行程序的可移植性和易用性。参与 MPI 标准制定的人员来自欧美 40 多个组织,大部分主要的并行计算机制造商、大学研究所、政府实验室、工业组织等都投入到 MPI 标准化工作。有了统一的并行编程语言标准,并行计算环境下的应用软件及软件工具就都能够实现透明的移植,各个厂商就可以依据标准提供独具特色和优势的软件实现和软件支持,从而提高了并行处理的能力^[6,12,13]。

MPI 是一种基于消息传递模型的并行编程接口,目前已经发展成为消息传递模型的代表和事实上的工业标准,而不是一门具体的语言。迄今为止,所有的并行计算机制造商都提供对 MPI 的支持,因而从理论上说任何一个正确的 MPI 程序可以不加修改地在所有

并行计算机上运行。

MPI 有下列特点:

(1) MPI 只是一个支持并行计算的程序库,并不是一个并行操作系统。

(2) MPI 标准虽然也可以支持异构集群,但是它并没有强制性地要求每个厂商都提供支持异构的实现。

(3) MPI 的语义非常精确。作为一个标准,它也必须有完备而精确的语义。因此,在 MPI 程序库中能够找到各种各样的发送和接收语义。

(4) 作为一种消息传递界面标准, MPI 没有严格要求底层的通信协议。它只是对于应用程序的通信程序库。至于下层的硬件,协议完全由厂商自己决定。这就为取得高效的通信效率提供了广阔的空间。近几年来,高性能的通信硬件不断地被投入使用,新颖的底层通信协议被设计出来。由于 MPI 的模块化和与底层协议无关性,使得 MPI 可以很快在这些底层协议中实现,从而取得很好的性能指标。

1993 年 2 月, MPI 1.0 发布, 1997 年 7 月, MPI 2.0 发布, 被称为 MPI-2, 原来的版本被称为 MPI-1。MPI-2 在动态进程管理、远程存储访问和并行 I/O 等等很多方面进行了重大扩充和改进。

2.3.网络互联技术

目前常用的高速互联网络技术有快速以太网、千兆以太网、IEEE SCI、Myrinet 和 InfiniBand 等几种^[14]。

快速以太网: 快速以太网是运行于 UTP 或光缆上的 100Mb/S 的高速局域网的总称。由于 TCP/IP 运行时对 CPU 的占用较多, 并且理论上的传输速度和延迟都比较差, 相对于其他几种网络互联技术是性能最差的一种。

千兆以太网 (Giganet): 千兆以太网是用于 Linux 平台的虚拟接口 (VI) 体系结构卡的第一家供应商, 提供 cLAN 卡和交换机。VI 体系结构是独立于平台的软件和硬件系统, 它由 Intel 开发, 用于创建群集。它使用自己的网络通信协议在服务器之间直接交换数据, 而不是使用 IP, 并且它并不打算成为 WAN 可路由的系统。Giganet 产品当前可以在节点之间提供 1 Gbps 单向通信, 理论最小延迟为 7 微秒, 实测延迟为 50-60 微秒左右, 并且运行时对 CPU 的占用也比较大。

IEEE SCI: IEEE 标准 SCI 的延迟更少 (理论值 1.46 微秒, 实测值 3-4 微秒), 并且其单向速度可达到 10Gb / 秒, 与 InfiniBand 4X 的理论值一样。SCI 是基于环拓的网络系统, 不像以太网是星形拓扑。这将使在较大规模的节点之间通信速度更快。更有用的是环面拓扑网络, 它在节点之间有许多环形结构。两维环面可以用 n 乘 m 的网格表示, 其中在每一行和每一列都有一个环形网络。三维环面也类似, 可以用三维立体节点网格表示, 每一层上有一个环形网络。密集超级计算并行系统使用环面拓扑网络, 为成百上千个节点

之间的通信提供相对最快的路径。

Myrinet: Myrinet 提供网卡和交换机,其单向互连速度最高可达到 1.28 Gbps。网卡有两种形式,铜线型和光纤型。铜线型 LAN 可以在 10 英尺距离内以全速进行通信,而在长达 60 英尺距离内以半速进行操作。光纤型 Myrinet 可以在 6.25 英里长的单模光纤或者 340 英尺长的多模光纤上全速运行。Myrinet 只提供直接点到点、基于集线器或基于交换机的网络配置,但在可以连接到一起的交换光纤数量方面没有限制。添加交换光纤只会增加节点间的延迟。两个直接连接的节点之间的平均延迟是 5 到 18 微秒,比以太网快。由于 Myrinet 板上集成可编程微处理器,能满足一些研究者的特定需要。

InfiniBand: InfiniBand 是由 InfiniBand 协会开发的体系结构技术,它是一种用于实现基于通道的交换式技术的通用 I/O 规范。IB 的理论带宽达 30Gbit/S。InfiniBand 技术通过连接主机信道适配器(HCA)、目标信道适配器(TCA)和网络交换机 3 个基本组件而发挥作用。

目前使用最广泛的依然是价格最便宜的快速以太网,千兆以太网应用范围也正在不断扩大。其他几种网络互联技术的应用范围还比较小,主要应用在对网络性能要求较高的地方。

2.4. 集群系统

2.4.1. 集群系统原理

将多台同构或异构的计算机连接起来协同完成特定的任务就构成了集群系统,集群系统就是一种并行处理系统^[15~20]。

集群系统主要分为三种:高可用性集群、高性能集群和负载均衡集群。

高可用性集群:主要功能就是提供不间断的服务。它适用必须一天二十四小时不停运转的计算机环境,如 web 服务器、工业控制器、ATM、远程通讯转接器、医学与军事监测仪以及股票处理机等。

高性能集群:通过将多台机器连接起来同时处理复杂的计算问题。它应用在需要大规模科学计算的环境中,如天气预报、石油勘探、分子模拟、基因测序等。高性能集群上运行的应用程序一般使用并行算法,根据一定规则把一个大的问题分为许多小的子问题,在集群内不同节点上进行计算,这些小问题的处理结果,经过处理后合并为原问题的最终结果。由于这些小问题的计算一般是可以并行完成的,从而可以缩短问题的处理时间。

负载均衡集群:使负载可以在计算集群集中尽可能平均地分摊处理。该负载可能是需要均衡的应用程序处理负载或网络流量负载。运行同一组应用程序时,每个节点都可以处理一部分负载,并且可以在节点之间动态分配负载,以实现平衡。负载网络流量时将流量发送给在其它节点上运行的网络服务器应用。

2.4.2. 集群系统特点

集群系统一般都采用成熟的、容易获得的计算机技术和通信技术以及硬件设备。这也使得集群系统具有价格便宜、扩展性好等特点^[15]。

首先，集群系统造价便宜。构造系统所使用的全部硬件均为通用的商品化产品，造价便宜。构建系统所用的软件，如 MPICH 编程环境、测试工具软件等，可以从网上免费下载，甚至操作系统也可以使用网上下载免费 Linux 版本。

其次，集群系统具有很好的可扩展性。可以根据需要方便地向集群中加入或删除工作节点，可以通过简单的增加新的节点来扩充集群的处理能力或增加新的功能。

最后，集群系统的每一个节点都是一台独立的计算机，在集群没有任务进行处理的时候每一个节点都可以单独使用，这样可以大大提高设备的利用率。

2.4.3. 集群系统分类

Beowulf 是最具代表的 Linux 高性能集群。起初，Beowulf 只是一个著名的科学计算集群系统，现在 Beowulf 已经成为一类广为接受的高性能集群的类型。在 Beowulf 集群的基础上又衍生出了很多集群系统。不同于 Beowulf 的集群系统有 COW 和 Mosix 这两类著名的集群系统^[19]。

Beowulf 集群：Beowulf 是一种能够将多台计算机用于并行计算的体系结构。通常 Beowulf 系统由通过以太网或其他网络连接的多个计算节点和管理节点构成。管理节点控制整个集群系统，同时为计算节点提供文件服务和对外的网络连接。它使用的是常见的硬件设备，如普通 PC、以太网卡和集线器。它很少使用特别定制的硬件和特殊的设备。Beowulf 集群的软件也是随处可见的，比如 Linux、PVM 和 MPI。

COW 集群：与 Beowulf 一样，COW (Cluster Of Workstation) 也是由最常见的硬件设备和软件系统构建而成。通常也是由一个控制节点和多个计算节点构成。

COW 和 Beowulf 的主要区别在于：

COW 中的计算节点主要都是闲置的计算资源，如办公室中的桌面工作站，它们就是普通的 PC，采用普通的局域网进行连接。因为这些计算节点白天会作为工作站使用，所以主要的集群计算发生在晚上和周末等空闲时间。而 Beowulf 中的计算节点都是专职用于并行计算，并且进行了性能优化。Beowulf 采用高速网 (InfiniBand, SCI, Myrinet) 上的消息传递 (PVM 或 MPI) 进行进程间通信 (IPC)。

因为 COW 中的计算节点主要的目的是桌面应用，所以它们都具有显示器、键盘和鼠标等外设。而 Beowulf 的计算节点通常没有这些外设，对这些计算节点的访问通常是在管理节点上通过网络或串口线实现的。

Mosix 集群：是一种非常特别的集群，它致力于在 Linux 系统上实现集群系统的单

一系统映像 SSI(Single System Image)。Mosix 集群将网络上运行 Linux 的计算机连接成一个集群系统。系统自动均衡节点间的负载。因为 Mosix 是在 Linux 系统内核中实现的集群，所以用户态的应用程序不需要任何修改就可以在 Mosix 集群上运行。对于他来说，Mosix 集群就是运行 Linux 的一台 PC。通常用户不会注意到 Linux 和 Mosix 的差别。

3. 集群系统构建

一个完整的计算机系统由硬件和软件系统两大部分组成, 集群系统也是如此。集群系统的硬件环境相对简单, 软件环境就复杂多了, 主要包括操作系统、集群管理系统、并行程序开发环境等。下面分别进行介绍。

3.1. 硬件环境

简单的说, 构建集群系统就是组建一个由若干台独立计算机组成的局域网。也就是说, 我们需要若干台独立节点计算机和网络互联设备两大部分硬件设备。目前市场上的硬件种类繁多, 不同厂家不同型号的产品令人眼花缭乱。当然, 不同厂家不同品牌不同型号的产品在功能、性能和价格方面也会有很大差别。这些商品化的硬件产品都可以用来构建集群, 只是使用不同产品所构建的集群系统功能、性能和价格不同。

3.1.1. 节点计算机

整个集群系统由若干个节点计算机构成, 每一个节点计算机在集群中的地位和作用是不一样的, 根据功能可以分为六类:

◇ 用户节点(User Node): 是外部世界访问集群系统的网关。用户通常登录到这个节点上编译并运行作业。用户节点是外部访问集群系统强大计算或存储能力的唯一入口, 是整个系统的关键点。

◇ 控制节点(Control Node): 主要承担两种任务: 为计算节点提供基本的网络服务, 如 DHCP、DNS 和 NFS。调度计算节点上的作业, 通常集群的作业调度程序(如 PBS)应该运行在这个节点上。通常控制节点是计算网络中的关键点, 如果它失效, 所有的计算节点都会失效, 所以控制节点也应该有硬件冗余保护。

◇ 管理节点(Management Node): 是集群系统各种管理措施的控制节点。它管理网络的控制点, 监控集群中各个节点和网络的运行状况。通常的集群的管理软件也运行在这个节点上。

◇ 存储节点(Storage Node): 是集群系统的数据存储器和数据服务器。如果集群系统的应用运行需要大量的数据, 就需要存储节点。

◇ 安装节点(Installation Node): 提供安装集群系统的各种软件, 包括操作系统、各种运行库、管理软件和应用。它还必须开放文件服务, 如 FTP 或 NFS。

◇ 计算节点: 是整个集群系统的计算核心, 它的功能就是执行计算。可根据需要和预算来决定采用什么样的计算节点配置。

每台计算机所扮演的节点类型由集群的实际需求和计算机的配置决定。虽然从体系

结构上说有六种类型的节点,但是,根据实际情况,用户节点、控制节点、管理节点、存储节点和安装节点可以是同一台计算机,这台计算机就是整个集群的主节点(Master Node)。其他节点仅仅承担计算任务,是集群的计算节点。集群就是由多个计算节点和一个主节点构成。当然,在大型集群系统中部署这些节点是比较复杂的问题,通常要综合应用需求,拓扑结构和预算等因素决定。由于本课题组建的集群规模较小,所以用户节点、控制节点、管理节点、存储节点和安装节点都由与外网连接的一台计算机担当,其他节点只作为单纯的计算节点。

集群的节点计算机既可以是服务器或工作站也可以是普通的 PC 机。本课题所构建集群系统使用的是普通的 PC 机。为了使集群的性能更好,节点机的 CPU 的主频越高越好,内存容量越大越好, I/O 系统速度越快越好。

本课题所构建集群系统的节点计算机配置如下:

主节点配置: Dell PC 机, INTEL P4 2.0G CPU, 1024MB 内存, 40G 硬盘。

子节点配置: Dell PC 机, INTEL P4 2.0G CPU, 512MB 内存, 40G 硬盘。

3.1.2.网络互联

因为大多数高性能科学计算任务都是通信密集型的,尽可能的缩短节点间通信延迟和提高吞吐量是一个核心问题,因此网络是集群最关键的部分,构建集群应尽量采用高传输速度、高带宽、低延迟的网络。网络互联中还有一种重要设备就是交换机和集线器。交换机和集线器的区别主要表现在带宽上。交换机每个端口都可以随时进行双向全带宽传输,而集线器同一时间则只能有一个端口进行全带宽接收或发送,如果多个端口同时传输数据,则只能分享带宽。

由于经费原因,本课题采用了价格低廉的普通以太网交换机来组建集群网络。集群系统内共有 16 台 PC 机,利用超五类双绞线通过网卡和交换机把 16 台 PC 机组成一个局域网,其网络结构如图 3-1 所示。

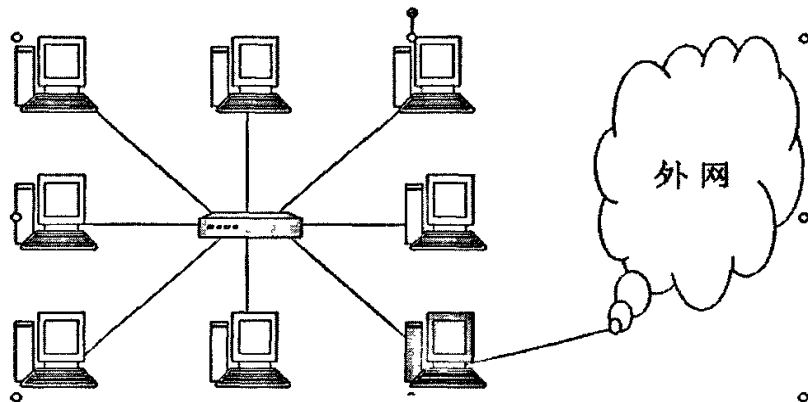


图 3-1 集群网络结构示意图

Figure3-1 The Network Structure of Cluster

3.2.操作系统

目前常用的操作系统有 Windows，Unix 两大类。Windows 是在个人计算机中应用最普遍的操作系统软件，而 Unix 运行稳定、安全性也比较好，在服务器操作系统领域占统治地位。

表 3-1 Top500 中的操作系统
Table3-1 The OS in Top500

操作系统	数量
Linux	326
AIX	43
CNK/SLES 9	27
HP Unix (HP-UX)	27
SuSE Linux Enterprise Server 9	25
UNICOS	8
RedHat Enterprise 4	7
SLES10 + SGI ProPack 5	5
Solaris	5
Redhat Linux	4
MacOS X	3
Super-UX	3
SUSE Linux	3
SuSE Linux Enterprise Server 8	3
Tru64 UNIX	3
UNICOS/SUSE Linux	3
SUSE Linux Enterprise Server 10	2
UNICOS/Linux	2
Redhat Enterprise 3	1

Linux 是从 Unix 发展来的一个开源操作系统，从 1991 年 10 月 5 日第一次发布以来，借助于 Internet 和众多爱好者的共同努力，发展迅猛。Linux 已成为目前构建高性能计算集群的最佳选择。表 3-1 显示了 2006 年 11 月超级计算机 TOP 500 中各种操作系统的使用情况^[1]，其中有 376 台采用各类 Linux 操作系统，占总数的 75.2%，而五年前仅有 44 台，占总数的 8.8%。

Linux 操作系统继承了 UNIX 操作系统超过二十五年的经验、源代码以及技术支持。

Linux 实际上指的只是一个内核，它是提供命令行或者程序与计算机硬件之间接口软件的核心部分。Linux 操作系统内核管理内存、硬盘、CPU、文件、进程等。在这个内核基础上，众多的厂商和组织发布了很多不同版本的 Linux，其中 Redhat 与 Suse Linux 的系统管理工具比较出色，Debian Linux 系统在功能的可裁剪性方面较好。本课题采用的操作系统是 RedHat Linux 9.0。采用 Linux 操作系统可以最大限度的节约成本。

3.3.网络配置

由于集群系统是多节点协同工作，构成集群的节点必须通过网络联结在一起，所有节点的操作系统安装完毕之后，首先进行网络配置。在构建硬件环境时，所有节点已通过以太网卡、交换机和网线组成了局域网络，下一步要做的工作是软件设置。

节点之间的通信采用计算机间最常用的 TCP/IP 协议。使用 TCP/IP 的好处是它对硬件设备没有特殊要求、传输数据的可靠性较高、是一种通信协议可以自由使用网络通讯协议。

进行网络配置的第一步是确保所有节点计算机都已安装上 TCP/IP 协议。大多数情况下，在安装操作系统时系统会默认安装 TCP/IP 协议。

TCP/IP 协议安装完毕之后，要为所有节点统一分配 IP 地址。在本课题构建的集群系统中节点只有主节点和子节点两种角色。从 16 个节点中选择一个性能最高的节点作为主节点，主节点担当用户节点、控制节点、管理节点、存储节点和安装节点多种角色，而且还作为普通计算节点参与计算。除主节点之外的其余 15 个节点都是子节点，仅作为计算节点使用。

由于角色的不同，在进行网络配置时，主节点和子节点的配置也稍有不同。为了方便，将所有节点放在一个域里面。其具体网络配置如图 3-2 所示。



图 3-2a 节点主机名、域的配置

Figure3-2a configure HostName and Domain

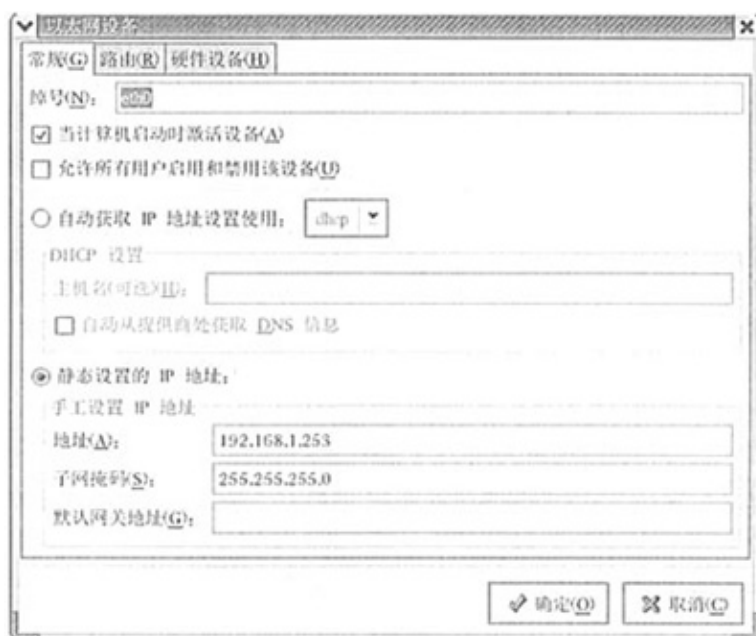


图 3-2b 节点 IP 配置

Figure 3-2b configure IP of node



图 3-2c 节点主机表配置

Figure 3-2b configure HostName Talbe

节点机的主机配置需要输入集群内全部节点的 IP、名称、别名等网络配置信息，比较繁琐。其实，这些信息都保存在配置文件/etc/hosts 中，这些信息对于所有的节点都是相同的。而且，集群节点每增加一个节点都得将新增节点的这些信息加入进来，如果其中任何一个节点的配置发生了变化，就得更改所有节点上的配置。所以，在这里可以先不做这项工作，等到后面单一登录系统构建完毕之后，通过远程操作将做好的一份配置文件分发复制到所有子节点，而且可以保证所有节点配置的一致性和准确性。

为了与外部网络进行联系，还要为主节点分配外网 IP 地址。主节点是集群系统与外部网络进行联系的唯一接口，也是用户与集群系统的交互接口，用户通过主节点提交任务和进行管理。集群系统对一般用户就是一个整体，集群内部所有节点对用户都是透明的在用户看来，他只是在和一台超级计算机进行交互。

虽然主节点需要分配外网 IP 地址，但主节点并不需要安装两块网卡，只需重新添加网络连接即可，一块网卡完全可以绑定多个 IP 地址。在本课题构建的集群当中，主节点的内网 IP 为“192.168.1.253”，外网 IP 为“202.200.121.18”，除了 IP 地址之外与机房内集群节点之外的其他普通计算机配置没什么不同。其配置如图 3-3 所示。其中，eth0 为内网网络设备配置，eth0:1 为内网网络设备配置。



图 3-3a 主节点网络配置

Figure 3-3a The Network Setting

在内网配置完成之后，继续新建网络，分配外网 IP 地址“202.200.121.18”。

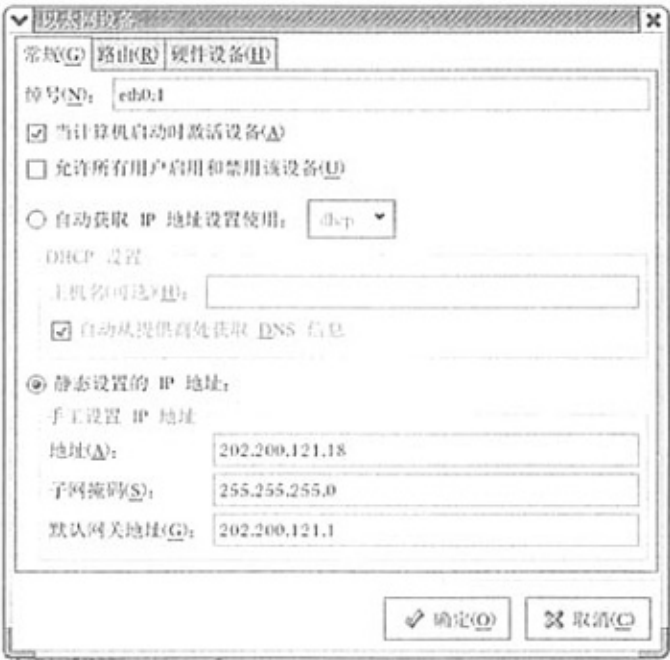


图 3-3b 主节点外网 IP 配置

Figure3-3b Extrareticulum IP Setting Of Main Node

一些网络管理员为了方便管理，将 IP 地址和网卡的 MAC 地址进行了绑定，这时候，还需要为外网用的 IP 地址指定硬件设备并关联 MAC 地址，如图 3-4 所示。



图 3-4 网卡 MAC 关联

Figure3-4 Associate MAC

当然，如果条件允许，为外网单独安装一块网卡会更好一些。

最后，还需要对系统的安全级别进行调整。默认情况下，计算机的安全级别设为中级，且禁止了 SSH 等服务的网络访问，而这些服务在后面构建单一登录系统和单一文件系统等工作中是必不可少的，所以必须要对它进行调整。要允许 SSH 等必要服务的网络访问，或者由于在集群内部都是信任连接可以不考虑安全问题干脆就直接将安全级别设为无防火墙，如图 3-5 所示。



图 3-5 防火墙设置
Figure3-5 Firewall Setting

3.4.单一登录构建

对于普通用户来说，由多个节点构成的集群系统是一个整体，集群系统的物理构造对用户是透明的，用户提交任务的时只须登录主节点提交一次任务即可，不需要为了将任务提交给其它节点而重复登录。

对于管理员来说，可能会更关心其中某一个节点的运行状态，经常需要登录到某个节点执行一些特定的操作。

在构建并行环境的时候，节点间大量频繁的通信会不停的需要登录信息，节点数量一多，用户名和密码的管理也是一项复杂的工作。

为了解决上述问题，需要为集群系统构建单一登录。所谓的单一登录最直接的外在表现就是，无论是用户提交任务还是管理员进行管理操作，登录集群系统只须要输入一次用户名和密码就可以操控集群系统中的任意一个节点。

在集群系统中，构建单一登录可以通过远程通信协议来完成。当前用于远程通信的协议很多，其中比较出名的是 RSH 和 SSH(Secure Shell)。RSH 是通过网络将要执行的命令分发给目的机，目的机在接收命令后根据用户的情况和当前的环境判断是否执行发送来的命令。RSH 在安全方面存在缺陷，RSH 在发送命令时使用的是明文的方式，这样一来命令在网络中传输很容易被截获，并且有可能被修改。

SSH 是 RSH 的一种改进，SSH 是通过 SSL 的加密方式来传输数据，从而避免了数据被截获和修改的可能。SSH 还可以实现远程登录，任何节点都可以通过 SSH 登录到其他节点进行权限允许内的操作。有了 SSH，我们只需要登录主节点，就可以控制集群内的所有节点，从而实现单一登录。

利用 SSH 构建单一登录的主要操作有以下几个步骤。

1. 以 root 用户登录到主节点计算机，进入 root 目录。
2. 更改/etc/hosts 文件，将所有节点计算机的 IP 地址、主机名称、别名等信息写入 hosts 文件。格式如下所示：

127.0.0.1	localhost.localdomain	localhost
192.168.1.253	node0.xaut	node0
192.168.1.1	node01.xaut	node01
192.168.1.2	node02.xaut	node02
.....		
192.168.1.15	node15.xaut	node15

更改其他节点上的/etc/hosts 文件，可以将 node0 上的 hosts 文件复制过去覆盖原来文件：

```
#scp 192.168.1.253: /etc/hosts /etc/hosts 192.168.1.1: /etc/hosts /etc/hosts
#scp 192.168.1.253: /etc/hosts /etc/hosts 192.168.1.2: /etc/hosts /etc/hosts
.....
#scp 192.168.1.253: /etc/hosts /etc/hosts 192.168.1.15: /etc/hosts /etc/hosts
```

3. 在 node0 生成 SSH 密钥对。

```
#ssh-keygen -t rsa 一路回车即可产生.ssh 文件夹
#ls -a 查看是否有.ssh 文件夹
```

4. 生成 authorized_keys 文件，将.ssh 文件夹下的 id_rsa.pub 复制为 authorized_keys。

```
#cp id_rsa.pub authorized_keys
```

5. 建立 node0 本身的信任连接。

```
#ssh node0 一路回车，遇提示时输入 yes，若需输入用户名密码则输入
```


6. 以 root 用户登录到子节点 node01, 进入 root 目录。

```
#ssh-keygen -t rsa 生成.ssh 文件夹
```

生成.ssh 文件夹后, 用 node0 上的.ssh 文件夹内文件覆盖本地生成的文件:

```
#scp 192.168.1.253: /root/.ssh/* /root/.ssh
```

建立到 node0 的信任连接:

```
#ssh node0 一路回车, 遇提示时输入 yes, 若需输入用户名密码则输入
```

按照同样的步骤和方法设置 node02, node03...node15。

7. 以 root 用户登录到主节点, 建立到所有节点的信任连接:

利用 SSH 依次登录所有节点, 一路回车, 遇提示时输入 yes, 若需输入用户名密码则输入。

```
#ssh node0
```

```
#ssh node1
```

```
#ssh node2
```

```
...
```

```
#ssh node15
```

执行一遍之后, 在主节点上利用 SSH 登录任一节点都不再需要输入用户名和密码。

8. 用主节点的.ssh 文件夹覆盖子节点的.ssh 文件夹, 可以在主节点上执行下面命令:

```
#scp /root/.ssh/* 192.168.1.01: /root/.ssh
```

```
#scp /root/.ssh/* 192.168.1.02: /root/.ssh
```

```
.....
```

```
#scp /root/.ssh/* 192.168.1.15: /root/.ssh
```

至此, 可以在任一节点上通过 SSH 远程登录其它节点而不需要输入用户名和密码。单一登录系统构建成功。

3.5.并行环境构建

通过第 2 章的介绍, 我们知道目前两种最重要的并行编程模型是数据并行和消息传递。数据并行编程模型的编程级别较高, 编程相对简单, 但只适用于解决数据并行问题; 消息传递编程模型的编程级别较低, 编程相对复杂, 但却有着更加广泛的应用范围。

基于消息传递并行编程模型的并行编程语言主要有 PVM 和 MPI 两种。PVM 能够在虚拟机中自动加载任务并运行, 并且还提供了任务间相互通信和同步的手段。MPI 的效率比 PVM 高很多, 且 MPI 使得并行程序具备更好的可移植性和易用性, 所以已成为开发基于消息传递模型的并行程序的工业标准。

为了使集群系统的应用范围更为广泛, 可采用消息传递编程模型; 为了更好的性能可

选择 MPI。但是，因为 MPI 只是一个并行编程语言标准，所以要编写基于 MPI 的并行程序，还必须借助某一 MPI 具体实现。MPI 有很多具体实现，如 MPICH，LAM，CHIMP 和 MPI/Pro 等，其中由美国 Argonne 国家实验室和 MSU 研制的 MPICH 是 Linux 平台下最重要的一种 MPI 实现^[22]。

MPICH 是一个与 MPI 规范同步发展的版本。每当 MPI 标准推出新的版本时，MPICH 就会有相应的实现版本，现在已经从 MPI-1 发展到了 MPI-2，对应的 MPICH2^[23]也已经推出。本课题采用的就是 MPICH2 1.0.3。

可以从网上 (<http://www-unix.mcs.anl.gov/mpi/mpich2/index.htm#download>) 免费下载 MPICH2 的源码。MPICH2 的源码下载下来之后，首先以 root 用户登录主节点，建立安装目录，如 /usr/MPICH2-install，将下载文件解压缩，通过 configure 脚本完成初始配置，然后使用 make 编译，编译成功之后 make install 安装。



图 3-6 节点文件配置

Figure3-6 File “mpd.hosts” Setting

安装之后，还需进行配置。

首先在 root 目录下编辑 .bashrc 文件修改环境变量，即将 MPICH2 的安装路径加入到文件中，如：PATH="\$PATH:/usr/MPICH2-install/bin"。

然后修改 /etc/mpd.conf 文件，修改内容为：secretword=myword，并设置文件读取权限和修改时间：

```
#touch /etc/mpd.conf
```

```
#chmod 600 /etc/mpd.conf
```

最后在 root 目录下建立节点文件 mpd.hosts，内容为所有节点的主机名，如图 3-6 所

示。



```
root@node0:
文件(E)  编辑(E)  查看(V)  终端(T)  转到(G)  帮助(H)

[root@node0 root]# which mpd
/usr/mpich-install/bin/mpd
[root@node0 root]# which mpicc
/usr/bin/mpicc
[root@node0 root]# which mpiexec
/usr/mpich-install/bin/mpiexec
[root@node0 root]# which mpirun
/usr/bin/mpirun
[root@node0 root]#
```


图 3-7a 环境变量测试

Figure3-7a Environment Variables Test

安装配置完成之后，主节点的并行环境就构建好了。接下来可以对其进行测试，如图 3-7 所示。

图 3-7a 主要是测试并行环境所必须的一些命令，包括并行环境启动、并程序的编译、执行等等。如果安装配置没有问题，则在使用 which 命令之后可以显示正确的路径，否则说明安装配置有问题。

图 3-7b 主要是测试本地并行环境。首先启动并行环境，然后查看节点运行情况，图中表示节点“node0”已经启动了。最后终止并行环境。



```
root@node0:
文件(E)  编辑(E)  查看(V)  终端(T)  转到(G)  帮助(H)

[root@node0 root]# mpd &
[1] 14382
[root@node0 root]# mpdtrace
node0
[root@node0 root]# mpdallexit
[1]+  Done                  mpd
[root@node0 root]#
```

图 3-7b 测试本地并行环境

Figure3-7b Local Parallel Environment Test

图 3-7c 是通过运行 MPICH 附带的一个求解 π 的例子程序来测试并行环境。如图所示

表示，从 mpd.hosts 启动一个节点 node0.xaut 计算并行程序 cpi，耗时 0.001293 秒，计算 π 的结果为 3.141592654424341。可以不管具体的计算数据，只要程序正常运行就证明主节点上的并行环境已经建成。

```

mpd@node0
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
[root@node0 root]# mpd &
[1] 14421
[root@node0 root]# mpdboot -n 1 -f mpd.hosts
[1]+  Done                  mpd
[root@node0 root]# mpdtrace
node0
[root@node0 root]# mpiexec -n 1 /usr/mpich-install/examples/cpi
Process 0 of 1 is on node0.xaut
pi is approximately 3.1415926544231341. Error is 0.000000008333410
wall clock time = 0.001239
[root@node0 root]# mpdallexit
[root@node0 root]#

```

图 3-7c 求解 π 算例测试

Figure3-7c Compute π

其他子节点的并行环境不需要安装和配置，在构建单一文件系统时通过 NFS 共享主节点的 MPICH2 安装文件夹即可。

3.6.单一文件系统构建

由于集群系统是多节点协同工作，大量相同的软件和数据要安装在所有的节点上，同时大量的数据共享也是必不可少的。如上一节介绍，若构建集群并行环境，在主节点安装并行环境 MPICH2 时，还必须在所有节点上安装 MPICH2，集群系统有多少个节点就得安装多少遍。为了解决这种问题，就必须建立单一文件系统。NFS (NetWork File System) 网络文件系统是集群系统中解决这个问题一个非常有效方法。

NFS 是一种使用比较广泛的网络文件系统。通过 NFS 服务可以把需要共享的文件、文件夹或者分区共享给网络上的其他计算机，需要访问这些数据的计算机使用 mount 命令把共享的资源加载到自己的系统上，然后就可以像使用本地文件系统一样进行对其访问和操作。

本课题中，将主节点作为 NFS 服务器。

NFS 的配置主要分三步，以下是操作过程：

1、安装 NFS。在系统安装时如果选择了网络服务安装的选项一般情况下会默认安装 NFS。如果 NFS 服务没有启动，则需要手动启动它。NFS 服务只需要在服务器上开启，客户节点只使用 mount 就可以，不需要其它的配置。

开启 NFS 服务进程：

```
# service nfs start
```

让 NFS 服务在系统启动时自动启动

```
# chkconfig --level 345 nfs on
```

2、配置 NFS

可以通过“系统设置—服务器设置—NFS 服务器”服务器设置工具配置，如图 3-8 所示。



图 3-8a NFS 服务器端配置
Figure3-8a NFS Service Setting



图 3-8b 添加 NFS 共享目录
Figure3-8b NFS Share Directory Setting



图 3-8c NFS 共享目录选项配置
Figure3-8c Option of NFS Share Directory



图 3-8d NFS 共享目录用户访问配置
Figure3-8d Access Option of NFS Share Directory

将主节点的/home/clu 和/usr/MPICH2-install 两个文件夹共享给所有节点。clu 文件夹存放一些共享信息，如可执行程序等等。mpich-install 文件夹是并行编程环境 MPICH 的安装文件夹，通过将其共享，在主节点安装好 MPICH 之后其它所有节点就都不需要再重复安装了。*.xaut 表示将其以读写同步的方式共享给域 xaut 内的所有计算机，主节点还有其它需要共享的数据可以按照上面的格式继续在下面添加。

因为 NFS 就是读取/etc/exports 文件并按照规则共享文件的，所以也可以直接修改 /etc/exports 文件。NFS 配置文件的格式如下：

共享的目录	可以接受的 IP 范围	访问规则
-------	-------------	------

本文的配置信息如下所示:

```
/home/clu/          *.xaut(rw,sync,no_root_squash)
/usr/MPICH2-install/ *.xaut(rw,sync,no_root_squash)
```

3、配置子节点的 NFS 服务

使用 mount 命令把共享的资源加载到自己的系统上, 如在以 root 用户登录节点 node01, 建立文件夹/home/clu 和/usr/mpich-install。虽然使用 mount 命令可以临时挂载主节点的 NFS 共享文件夹, 但是为了使计算机每次启动时自动挂载, 我们需要修改/etc/fstab 文件, 将以下内容添加的最后:

```
node0:/home/clu          /home/clu          nfs
node0:/usr/MPICH2-install /usr/MPICH2-install nfs
```

对于单纯的文件共享, 子节点的文件夹没有必要一定和主节点一致, 但是有时候程序执行与路径有关, 为了便于管理和操作, 我们将子节点文件夹和主节点文件夹设为同样的路径和名称。

重新启动计算机, 会发现这两个文件夹内已经有了内容, 就是主节点对应文件夹内的内容。这样节点 node01 就设置完毕了。为了使其它节点也可以共享主节点的数据, 也要一一设置。由于前面我们已经构建了单一登录, 所以我们不必一一登录其他节点, 这些操作可以在一个节点上进行, 使用 SSH 远程登录到其他节点, 新建文件夹, 复制节点 node01 的/etc/fstab 覆盖本地文件即可。

至此, 就完成了 NFS 的全部配置, 集群系统的单一文件系统构建成功。

3.7.单一作业管理系统构建

完成前面六项工作之后, 集群系统实际上就已经可以进行并行计算了。用户可在主节点上提交作业, 然后由主节点分配任务给各个子节点进行并行计算。当有多个作业需要计算的时候, 必须逐个提交, 一个任务完成之后才能提交下一个任务, 否则由于任务交叉分时进行, 互相抢占资源会造成效率急剧下降。集群上运行的一般都是大型并行程序, 需要运行很长时间, 且这个时间通常是无法准确预知的, 让用户一直守候在跟前等待并不现实, 这些工作需要由作业管理系统来完成。

作业管理是系统运行中很重要的一项工作, 特别是像集群这样的多处理器系统。OpenPBS 是一种常用的作业管理软件, 最初由美国航空航天局 (NASA) 开发, 后来逐步发展完善, 现在已经是并行计算机任务管理中最先考虑的选择。

可从网上免费下载 OpenPBS, 也可选择提供技术服务支持的付费商业化版本 PBS pro, 目前使用更多的还是免费版 OpenPBS。

下载 OpenPBS 源码之后, 首先以 root 用户登录, 建立安装目录, 解压缩, 通过 configure 脚本完成初始配置, 然后使用 make 编译, 编译成功之后通过 make install 安装。

安装完成后进行进一步配置。由于角色不同，主节点和子节点配置有所不同。在主节点上，修改节点文件/usr/spool/PBS/server_priv/nodes，明确其需要控制的节点，即加入子节点的主机名，如：

```
node01
node02
.....
node15
```

在子节点上，修改配置文件/usr/spool/PBS/mom_priv/config，使其接受主节点的控制。修改如下：

```
$logevent 0x1ff
$clienthost node0
```

安装配置完 OpenPBS 后，单一作业系统构建完毕，用户即可通过 OpenPBS 进行作业提交，之后作业的调度等工作便由 OpenPBS 来自动完成，用户还可以通过它来删除和查看任务。

3.8.其他可控管理

对于集群来说，由于节点计算机数量较多，安装和使用过程中都有很多重复的工作，平时看来一个非常简单的操作就可能令人头痛。前面通过构建单一登录和单一文件系统已经解决了大量的软件安装和配置，数据共享等方面的问题，但是仍有很多问题没有解决，例如，需要将所有节点机重新启动，十几台计算机挨个重启一遍还算可以接受，如果集群规模一大，有成百上千个节点机，这个工作将成为一件非常可怕的事情。

在 Linux 下，所有操作都可通过命令行来执行，我使用 shell 脚本编写了一些可控管理程序，解决了节点机重启和关机、系统启动选择、图形文本模式切换等功能，大大增强了集群系统的可控性。

shell 脚本可将简单的程序和命令有机结合起来，完成更复杂的任务，实际上就是一种批处理命令。shell 有自己的语法结构和命令，可以使用变量、通配符和正则表达式。下面以重新启动集群所有子节点的脚本为例进行说明：

在 Linux 下重启计算机的命令是 reboot，重新启动本机只需要在命令行下执行命令“reboot”即可。如果要在主节点上进行操作使某个子节点重启，可以远程登录到这个节点，然后执行“reboot”命令就可以了。reboot 命令的连接存放在/sbin目录下，所以也可以执行“/sbin/reboot”。在前面已经构建了单一登录，可以采用 SSH 来在远程计算机上执行命令。如重新启动节点 node01：#ssh node01 /sbin/reboot

如果需要重新启动所有的节点实际上就是依次执行下列命令：

```
#ssh node01 /sbin/reboot
```



```
#ssh node02 /sbin/reboot
```

```
.....
```

```
#ssh node15 /sbin/reboot
```

由于是重复执行同样操作，只是节点名不同，执行次数也是事先知道的，所以可以使用 For 循环来实现命令的重复重复执行。而每次执行的节点名称是不同的，而且并节点编号不一定连续，所有使用 in 来标示所有需要操作的节点编号。完整的代码如下所示：

```
#!/bin/sh
for i in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
do
    sleep 1
    echo " Reboot node$i ....."
    ssh node$i /sbin/reboot
done
```

第一行“#!/bin/sh”必不可少。表明此脚本使用/bin/sh 来解释执行。第二行开始 for 循环，循环次数由声明的一个变量“i”来决定，“i”的取值范围是 01、02、03、04、05、06、07、08、09、10、11、12、13、14 和 15。第三行到第七行是循环体。第四行完成的功能只是停顿 1 秒钟。第五行输出提示信息，表示正在重启那个节点。注意“\$”不可少，它是导出变量值的操作符，表明后面的“i”不是单纯的字符了，而是变量。第六行具体执行重启操作。通过 SSH 执行远程计算机上的命令。

通过 shell 脚本可实现很多类似功能，大大增强集群系统的可操作性和便捷性。

3.9. 集群规模扩展

集群系统构建完毕之后，集群系统的规模可以通过简单增加和删除节点机来实现。缩小规模比较容易，只需要去掉节点机，在里面去掉相应的注册项目即可。增加新的节点则相对比较麻烦，需要调整很多设置。具体要做的工作有新增节点的系统安装和网络配置，重构单一登录、将新增节点纳入单一文件系统，配置新增节点并行计算环境，修改其他可控管理程序等几个方面。

下面以新增一个节点到前面所构建的集群系统为例进行说明。

前面所构建集群已有 16 个节点，分别是“node0, node01, node02.....node15”，则新增节点命名为“node16”，按照顺序为其分配 IP 地址为“192.168.1.16”。接下来进行集群扩展操作。

1. 为新增节点的系统安装和网络配置。

- 首先为新增的节点机安装操作系统，安装过程与其他节点机的安装过程相同，计算机名为“node16”
- 操作系统安装完毕之后进行网络配置，安装 TCP/IP 协议，分配 IP 地址，IP 地址为“192.168.1.16”。

这一步要达到的目的是新增节点与其他节点可以互相访问。

2. 重构单一登录。

➤ 修改主节点/etc/hosts 文件，将新增节点信息加入。

#vi /etc/hosts 打开 hosts 文件，添加如下内容：

```
192.168.1.16    node16.xaut    node16
```

➤ 重新配置 SSH

这一步完成之后要达到目的是，在集群的新增节点和原有节点之间可以不需要输入用户名和密码就可以通过 SSH 远程登录并执行操作。

3. 将节点机纳入单一文件系统。

按照构建单一文件系统操作中的第三步配置节点 node16 的 NFS 服务。即：新建挂载文件夹，修改/etc/fstab 文件。

这一步达到的目的要使新增节点可以将 NFS 服务器上的共享文件开机后自动加载为本地文件。

4. 修改并行计算环境配置。

修改主节点 mpd.hosts。将新增节点的 IP、主机名、域名、别名等信息按照前面介绍的格式添加到 mpd.hosts 文件的最后即可。

5. 修改其他可控管理程序。

主要是修改 shell 脚本中循环变量的取值范围，将节点 node16 添加进来。

通过以上操作，就将新增节点 node16 加入到集群系统中了。

4. 集群系统性能测试

Linpack^[9]是当今世界上使用最广泛的计算机计算性能测试基准，它通过对稠密线性代数方程组求解能力的测试，评价高性能计算机系统的浮点运算性能，测试结果按每秒浮点运算次数（Flop/S）表示。

HPL(high performance Linpack)是第 1 个标准的公开版本：并行 Linpack，目前广泛用于 Top500 测试。这一测试主要针对分布式存储大规模并行计算系统而设计，用户可以设置任意大小的问题空间，使用任意个数的 CPU，利用基于高斯消去的各种优化方法寻求最佳的测试结果。

4.1.HPL 安装

HPL 的源码可以从网上免费下载（<http://www.netlib.org/benchmark/hpl/hpl.tgz>），下载后需要根据自己的实际情况进行设置修改，重新编译，然后安装。此外，LinPack 在执行过程中需要一个基本线性代数子程序(BLAS)，一个线性方程组的数学库。虽然 Linux 在安装时会自动安装一些数学库，但并不能满足 HPL 测试的要求，因此需要下载新的 BLAS 包。下面介绍安装具体步骤。

以 root 用户登录主节点，建立工作目录，如：/home/clu/hpl。

下载一个已经编译好的 BLAS 包（http://www.netlib.org/blas/archives/blas_linux.tgz），下载之后解压缩，得到 blas_linux.a 文件，将这个文件复制到 HPL 安装目录下，如图 4-1 所示。

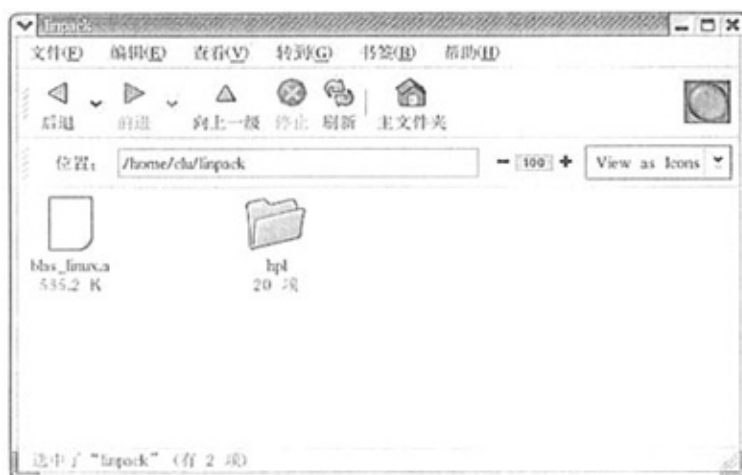


图 4-1 将 blas_linux.a 文件复制到 HPL 安装目录下

Figure4-1 Setting of File “blas_linux.a”

解压下载得到的 HPL 源码，并进入解压目录，创建适合于自己的 makefile 文件。HPL 提供了很多模版以供选择，都放在 setup 文件夹下。选择的时候主要是看计算机的硬件配置，如本课题集群中的节点计算机采用的都是 Intel P4 CPU，所以选用比较接近的 Linux_PII_FBLAS，在其基础上进行适当的修改。将其复制到解压目录下，重新命名为 make.linux，然后对其进行修改。需要修改五处，都是路径的设置，具体设置如下：

- 设置 hpl 的路径：TOPdir = /home/clu/hpl
- 设置 mpich 的路径：MPdir = /usr/MPICH2-install
- 设置 blas 库文件的位置：LAlib = /home/clu/hpl /blas_linux.a
- 设置 C 并行编译器的位置：CC = /usr/MPICH2-install /bin/mpicc
- 设置 Fortran 并行编译器的位置：LINKER = /usr/MPICH2-install /bin/mpif77

路径修改完毕之后把文件中的 Linux_PII_FBLAS 改为自己的名字，如：Linux_Xaut。

Makefile 文件配置好之后编译安装即可。安装完毕之后在 hpl/bin/Linux_Xaut 目录下会有 xhpl 和 HPL.dat 两个文件，如图 4-2 所示。xhpl 是测试程序本身，HPL.dat 是控制 hpl 运行的参数文件。

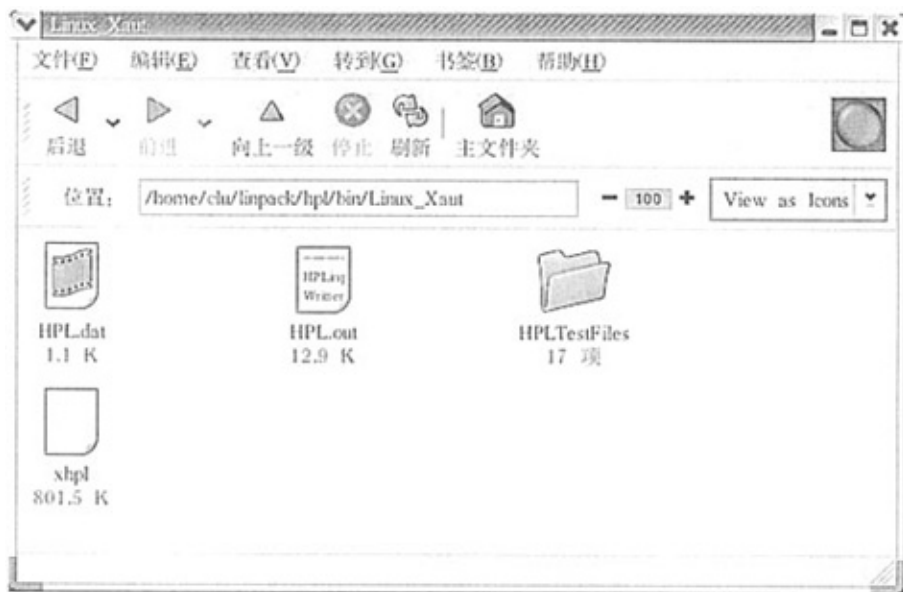


图 4-2 HPL 测试过程主要使用的几个文件

Figure4-2 The Most Important Files of HPL

HPL 测试并非像某些人想象的那样容易——简单的根据一个程序执行的结果就能得出结论。HPL.dat 参数文件配置的不同，测试结果会有很大不同，我们必须通过不断的修改 HPL.dat 参数文件里面的多项参数进行多次的测量，才能得到最佳配置参数，从而得到最佳测试结果。下一节将对 HPL.dat 参数文件内的参数进行详细介绍。

以上工作完成后，主节点就安装好了，调试通过后，要通过单一文件系统 NFS 共享

给其他节点。

4.2.HPL 参数

HPL 测试中参数的设置对测试结果影响很大,进行 HPL 测试的难点也在于 HPL 参数的设置。HPL 的参数在 HPL.dat 文件中设置,这个文件共有 31 行,可以设置的参数有很多,但没有必要全部重新设置,多数参数可以采用默认值。对测试结果影响较大的参数主要有问题空间的大小 (Ns)、LU 分解数据块大小 (NBs),处理器网络组成方式 (P, Q),这几个参数也是在测试中需要频繁调整的。下面简单介绍 HPL.dat 文件中的重点部分。

第 3 行: HPL.out output file name (if any)。

这一行参数决定将输出信息的保存到那个文件中,默认为 HPL.out,路径就是 XHPL 程序所在的目录,如: hpl/bin/Linux_Xaut。

第 4 行: 6 device out (6=stdout,7=stderr,file)。

指定输出结果的输出设备,有三个选项: 6, 7, 其他。6 表示把计算出的结果输出到标准输出设备(显示器); 7 表示输出错误信息;其他的数字或者是“file”表示输出到一个文件中,就是在前面第三行定义的那个文件(如 HPL.out)。

第 5 行: 3 # of problems sizes (N)。

指定问题的个数,大于 0 小于等于 20 的整数。每个问题的大小和详细的信息在下面进行详细设置。

第 6 行: 3000 6000 10000 Ns

指定每个问题的大小,问题个数是上一行设定的数字。前 n 个数据会被使用,其中 n 表示上一行中我们指定的问题个数。默认值表示求解 3 个问题,问题大小的值分别为 3000、6000、10000,如果上一行为 2,则表示求解 2 个问题,问题大小分别前两个值 3000、6000,第三个值 10000 则被忽略。但是如果上一行设定问题数量为大于 3 的数,则程序无法正常运行,因为此行给定的问题大小描述不够,所以,这一行的问题大小描述数量必须等于或大于上一行设定的问题数量。

第 7 行: 5 # of NBs

指定采取多少种分块对问题矩阵进行分解,大于 0 小于等于 20 的整数。具体分块大小在下面具体设定。第一个整型数据将被使用,后面的数据将被忽略。默认值 5 表示将使用 5 个大小不同的分块,分块具体的大小将会在后面的行中进行描述。

第 8 行: 80 100 120 140 160 NBs

确定每个分块的大小。前 n 个数据有效,其中 n 是上一行定义的分块个数,如默认值 5 那么程序只读取前 5 个数据,后面的数据将被忽略。若定义的分块个数小于 n,则程序不能正常运行,即分块个数要大于等于 n。默认值: 80 100 120 140 160 NBs 表示将使用 5 次不同分块,其中大小分别为 80、100、120、140、160。

第 9 行: 0 PMAP process mapping (0=Row-,1=Column-major)指定 MPI 进程如何的分配到节点上。这个参数对于集群系统可以不用调整, 使用默认值即可。

第 10 行: 2 # of process grids (P x Q)

指定计算网格的个数, 大于 0 小于等于 20 的整数。默认值 2 表示将使用 2 个不同的计算网格。

第 11、12 行: 12 Ps

68 Qs

确定每个计算网格的行数据和列数据。这两行的前 n 列是上一行网格定义的具体说明。同样的, 列数应该大于等于 n 。例如在默认的情况下使用 2 个计算格, 那么这两行的前 2 个数据被分别看作行和列。表示 xhpl 使用两个计算网格, 分别是 1-6 和 2-8。在这里, 至少需要 16 个节点 ($P \times Q$ 的最大值), $P \times Q$ 为计算所使用的节点数, 所以其最大值不能超过集群可用节点总数。

从第 13 行开始之后的参数都采用默认值, 一般不修改。在本课题的测试当中, 主要调整的参数主要是问题空间的大小 (N_s)、LU 分解数据块大小 (NBs) 和处理器网络组成方式 (P, Q)。在测试时, 测试结果输出到文件 HPL.out 文件中, 不断调整参数设置进行测试, 然后对测试结果进行分析。下面一节将详细介绍本课题进行的测试。

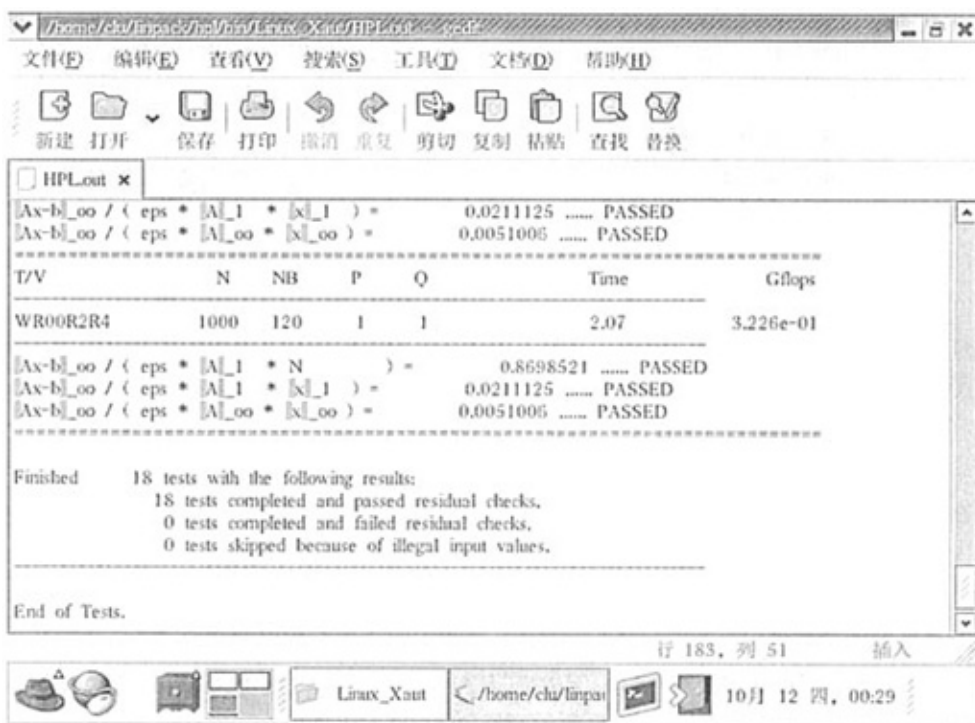


图 4-3 HPL 测试输出结果
Figure4-3 Output File of HPL Test

4.3.HPL 测试与分析

HPL 测试十分繁琐，仅仅调整最关键的 N、NB、P、Q 这几个参数，就需要做千百次测试，而对于每一种组合情况都会进行 18 次测试，测试结果如图 4-3 所示。我们需要做的就是不断的调整输入参数，输出结果进行分析，再测试，再分析，不断重复测试与分析，以找出各个参数对测试结果的影响，以便使用最佳的输入参数对集群系统进行测试。

4.3.1.Ns 对 HPL 测试的影响

问题空间 (Ns) 即集群求解问题的规模，其最大取值与可用内存大小有关。HPL 测试程序计算使用的是 64 位精度的矩阵，所以设 N 为问题空间，M 为内存总量(单位 MB)，则 N 与 M 之间存在如下关系：

$$N^2 \times 64 = M \times 1024 \times 1024 \times 8$$

即：

$$N = \sqrt{M \times 1024 \times 1024 \times 8 / 64} \approx 362\sqrt{M}$$

根据上面公式可以计算出问题空间的最大值，大于这个值，计算时内存不足系统将会使用交换分区，从而导致计算速度迅速降低。在实际当中，由于系统本身要占用一部分内存，所以计算时应将系统占用的内存部分去掉，同时由于计算本身也需要占用部分内存，所以一般取可用内存的 80%。

假设集群共有 P 个计算节点，所有节点的可用内存均为 m (单位 MB)，则问题空间的取值如下：

$$N = 362\sqrt{m \times 0.8 \times P}$$

在其他参数不变的情况下，不断调整问题空间大小，查看问题空间对测试结果的影响。表 4-1 为问题空间对测试结果影响测试的参数，根据实际测试结果得到图 4-4 所示。

表 4-1 问题空间测试参数
Table4-1 The Parameter of N Test

NBs	P	Q
120	4	4

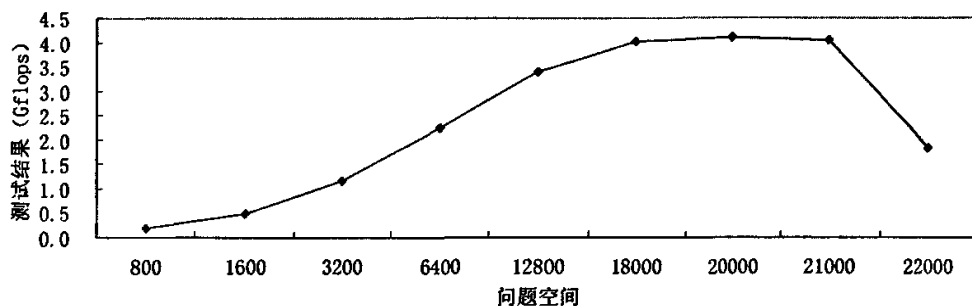


图 4-4 测试问题空间对测试结果的影响

Figure4-4 Test The Affect of N

从图 4-4 中曲线可以明显看出,问题空间对测试结果的影响很大。同样的配置,随着问题空间的增大,测试结果迅速增高,达到一定高度后又迅速降低。问题空间为 800 时,实测结果仅为 0.1941GFlops,不但不及问题空间为 20000 时的 5%,而且远低于单机测试结果最好值的 0.37GFlops,16 个节点反而不如 1 个节点的计算性能。原因是问题空间太小,网络通信等额外开销远远超出了节点增加带来的性能提高。

4.3.2.NBs 对 HPL 测试的影响

LU 分解数据块大小 NBs 为矩阵分解为小数据矩阵的大小。NBs 的最佳取值没有明确的计算依据,更主要的依靠根据具体环境下通过多次实验不断摸索尝试。网络结构、问题空间、集群计算节点数量等诸多因素对 NBs 的取值都有影响。

首先,在单节点下用不同的问题空间下测试 NBs 对测试结果的影响。测试结果如图 4-5 所示。

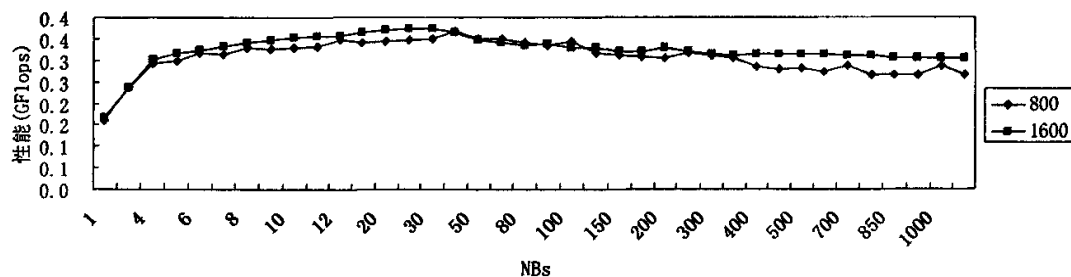


图 4-5 测试 NBs 对测试结果的影响

Figure4-5 Test The Affect of NBs

从图 4-5 中可以看出,在问题空间分别为 800 和 1600 时的两条测试曲线都显示: NBs 从 1 到 5 时测试结果是迅速增高,从 5 开始,变化就趋于平缓,表明在单节点下, NBs 对测试结果的影响并不是很明显。

分别在 1 个节点, 4 个节点, 9 个节点和 16 个节点下测试 NBs 对测试结果的影响,

测试结果如图 4-6 所示。

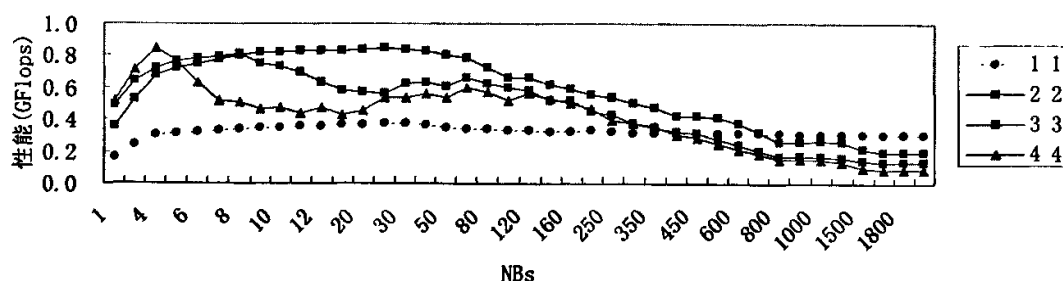


图 4-6 在不同节点下测试分块大小对性能影响测试

Figure4-6 Test The Affect of NBs Wwith Different Scale of Cluster

从图中曲线可以看出，当节点数大于 1 时，NBs 对测试结果的影响比较明显，原因是 NBs 取值影响负载均衡。NBs 取值太小，虽然可以更好的实现负载均衡，但不能充分的发挥节点 CPU 的处理性能，且计算过程需要不断的进行网络通信，这时候网络带宽和延迟对测试结果将产生很大的影响。NBs 取值过大，导致负载不均衡，根据木桶原理，最终的测试结果反而降低。极端情况下，NBs 等于或大于 N_s ，负载极度不均衡，这种情况下的测试结果甚至远远低于单个节点的测试结果。

4.3.3.PQ 对 HPL 测试的影响

PQ 决定处理器网络组成方式。P 代表水平方向处理器个数，Q 代表垂直方向处理器个数，P 和 Q 共同组成一个二维处理器网络。对于 PQ 的取值，已有较为明确的说法，一般认为处理器网络应采用近似正方形的结构，同时 P 尽可能大于等于 Q^[8]。

在问题空间为 3600，采用 4 个节点分别采用 1*4，2*2，4*1 进行测试，测试结果如图 4-7 所示，证明此观点正确。

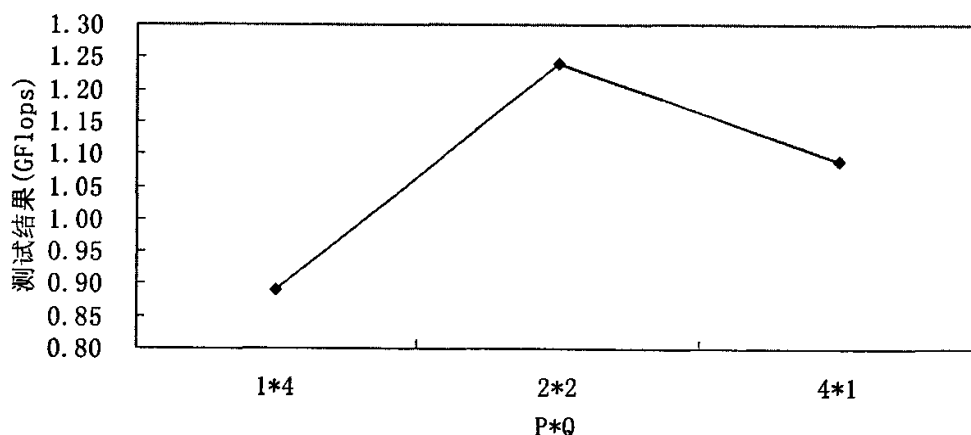


图 4-7 同一问题不同网络结构测试结果

Figure4-7 Test The Affect of PQ

4.3.4.节点数对 HPL 测试的影响

我们最关心的是集群系统的整体性能如何。理论上说增加节点数会提高整体计算性能，实际情况如何？我们对问题空间分别为 3600 和 7200 的两个问题分别用 1，2，4，6，9，12，15，16 个节点进行求解，图 4-8 和图 4-9 分别是求解该问题的计算耗时对比曲线和计算性能对比曲线。

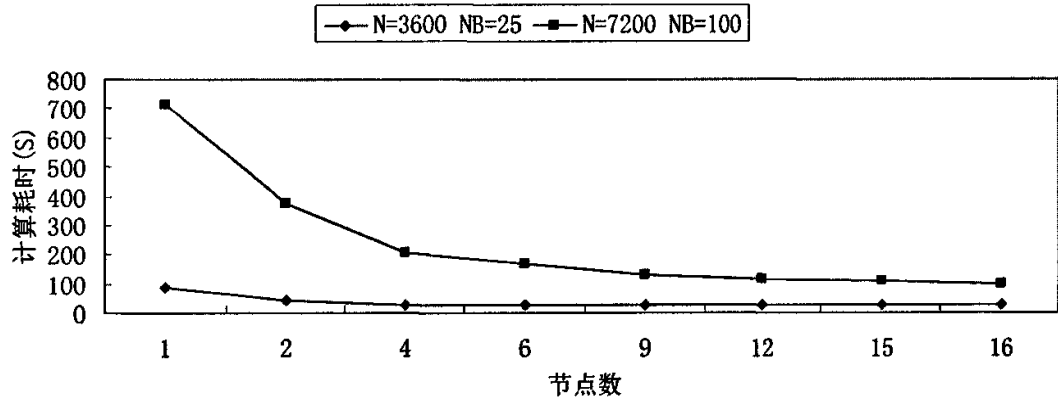


图 4-8 同一问题下不同节点数对计算耗时的测试结果
Figure4-8 Elapsed Time With Different Scale of Cluster

通过图 4-8 可以很明显的看出，对同一问题随节点数的增加，计算耗时逐渐减少。问题空间越大，体现的越明显。问题空间为 7200 时，单机计算耗时 712.36 秒，而 16 个节点仅需 95.53 秒。

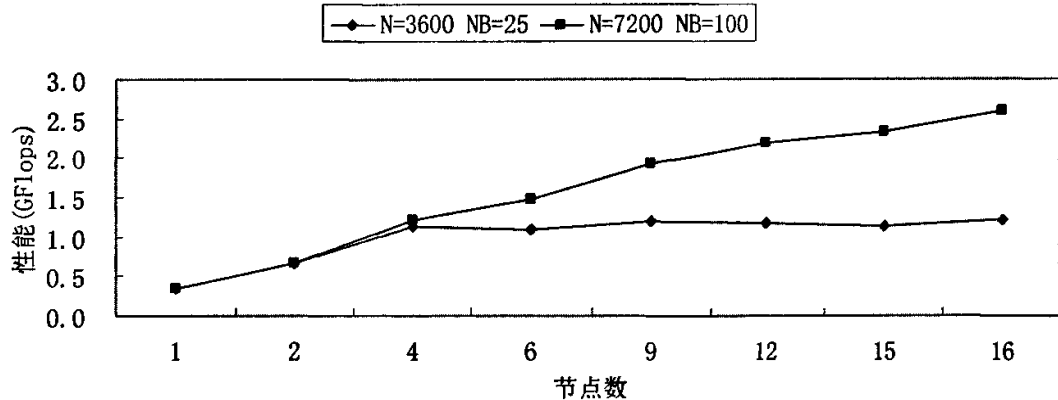


图 4-9 同一问题下不同节点数对计算性能的测试结果
Figure4-8 HPL Test With Different Scale of Cluster

通过图 4-9 可以看出，随着计算节点的增加，若问题空间太小，网络通信等额外开销不能抵消计算性能提高带来的好处，则整体计算性能测试值不会提高，反而有可能降低。且在未超过问题空间最大值的情况下，问题空间越大，测试性能越高。当问题空间为 7200 时，16 个节点的实测性能能达到 2.89GFlops，为 1 个节点实测性能 0.35GFlops 的 7.43 倍。

继续增大问题空间，进行多次测试，发现当问题空间达到 20000 时，实测结果为 4.10GFlops，超出单节点测试最高值 0.37GFlops 的 11 倍还多。由于额外开销的增加并没有达到理想状态下的 16 倍，但是多节点并行计算的巨大威力已经显示了出来。

通过 HPL 测试，说明集群系统的构建是成功的，通过多节点间的协同并行处理，明显的提高了计算性能，且随着节点数的增加，集群的整体计算性能基本呈线性增加，影响集群整体性能的关键在于单个节点的性能以及集群系统所采用的网络互连技术(主要是网络带宽和通信延迟)，而这些都由所采用硬件设备的特性决定。

在硬件设备一定的情况下，只能从并行算法以及任务分配、作业调度方面着手提高集群性能。在本课题中下一步所要做的工作就是在此集群系统上进行并行程序的开发，实现串行算法的并行化及并行程序设计中的任务合理分配，第 5 章内容将对这方面的工作进行介绍。

5. 并行排序算法实现

经过前面几章介绍的工作，集群系统构建完成且经过了 HPL 测试，显示了集群系统相对单机的高性能计算能力，接下来的工作就是在已经构建的集群系统上进行实际应用，做一些实际的程序开发。本课题对计算机领域熟知的排序问题进行了并行化实现，使其能够在高性能集群中运行。

所谓排序，就是将原本杂乱无章的数据元素按照一定顺序依某种关键字进行排列。可以这样来定义：若给定的文件含有 n 个记录 $\{R_1, R_2, \dots, R_n\}$ ，它们的关键字分别为 $\{K_1, K_2, \dots, K_n\}$ ，要把这 n 个记录重新排列成为 $\{R_{i1}, R_{i2}, \dots, R_{in}\}$ ，使得 $\{K_{i1} \geq K_{i2} \geq \dots \geq K_{in}\}$ （或 $\{K_{i1} \leq K_{i2} \leq \dots \leq K_{in}\}$ ）。

排序算法有很多，如枚举排序、插入排序、冒泡排序、快速排序、希尔排序、归并排序、堆排序等等。枚举排序算法若使用串行算法，其时间复杂度为 $O(n^2)$ ，性能与其他同类算法相比较低。但是枚举排序算法设计思想简单，易于实现，若在集群系统中进行并行化，其性能将大大提高。下面介绍在集群系统中枚举排序算法的并行化实现。

5.1. 枚举排序算法原理

枚举排序（Enumeration Sort）也称为秩排序（Rank Sort）。假设要求按关键字从小到大排序，则枚举排序算法的基本思想就是：对每一个待排序的元素统计小于它的所有元素的个数，由此得到该元素在最终排序完成之后的序列中的位置，得到了每一个元素的位置之后，也就得到了最终的排序序列^[27]。

假设有一个整数数组 ArrayInt，它含有 n 个元素，分别是 ArrayInt[0]…ArrayInt[n-1]。要求将其从小到大排列，即令任意元素 ArrayInt[k] ($0 < k < n-1$)，存在如下不等式：ArrayInt[k-1] \leq ArrayInt[k] \leq ArrayInt[k+1]。

按照枚举排序算法，假设最终排序后得到的数组为 ArrayIntB[]，首先将 ArrayInt[0] 与 ArrayInt[1]…ArrayInt[n-1] 一一比较，得到比 ArrayInt[0] 小的元素的个数，假设为 k ，则 ArrayIntB[k]=ArrayInt[0]；同样，将 ArrayInt[1] 与 a[0], a[2]…a[n-1] 一一比较，得到比 ArrayInt[1] 小的元素的个数 m ，ArrayIntB[m]=ArrayInt[1]；依此类推，经过 $n(n-1)$ 此比较操作，可以得到最终的排序结果 ArrayIntB[]，由此我们也可以明显看出枚举排序算法的时间复杂度为 $O(n^2)$ 。

5.2. 枚举排序的串行算法

根据枚举排序算法原理，我们可以很快写出它的 C 语言串行算法实现。

我们采用 3 个数组分别存放待排序的输入序列、秩序列和排序结果的输出序列。实际编写程序时,采用指针来实现,以提高性能和灵活性。假设只考虑整数序列的排序,我们定义三个整型指针 IN、Rank、OUT。

```
int *IN, *Rank, *OUT;    /* IN:待排序序列    Rank:秩序列    OUT:输出序列 */
```

设待排序序列的长度为 DataSize, 即: 待排序序列含有 DataSize 个元素。为三个序列分配内存空间:

```
IN=(int *)malloc(DataSize*sizeof(int));    /* 为待排序序列分配内存空间 */
OUT=(int *)malloc(DataSize*sizeof(int));    /* 为输出序列分配内存空间 */
Rank=(int *)malloc(DataSize*sizeof(int));    /* 为秩序列分配内存空间 */
```

接下来,我们就可以进行排序操作了。对 IN 序列中的每一个元素,将它和序列中的所有元素比较一遍,统计出比它小个元素的个数,存放在 Rank 序列中。

```
for(i=0;i<DataSize;i++)    /* 计算每一个元素的秩,存在秩序列中 */
{
    k=0;
    for(j=0;j<DataSize;j++)    /* 统计IN[i]的秩 */
    {
        if(IN[i]>IN[j]) || (IN[i]==IN[j] && i>j)
            k=k+1;
    }
    Rank[i]=k; /* 将IN[i]的秩存入秩序列 */
}
```

最后,根据 Rank 序列得到排序结果,存放在输出序列中:

```
for(i=0;i<DataSize;i++)
    OUTrank[i]=IN[i];
```

到此为止,利用枚举排序算法对输入序列进行从小到大排序就完成了。

5.3.枚举排序的并行算法

要完成一个求解问题的并行算实现一般遵循以下步骤:

1. 考虑待求解问题是否具备并行性。求解问题具备并行性才有可能有并行算法,如果不具备并行性就不可能实现并行计算。
2. 编写解决问题的串行算法。
3. 将串行算法并行化。

枚举排序算法具有很好的并行性。枚举排序的主要工作就是求解待排序序列中所有元素的秩,而求解任何一个元素秩的操作都是完全独立的,和求解其他元素秩的操作没有任何关联。换句话说,求解序列中全部元素秩的操作可以在同一时刻同时进行,且互不干扰,这使得枚举排序算法具备非常理想的并行性。

枚举算法的串行程序我们在上一节已经完成了,下面的工作就是对它进行并行化。对照枚举排序的串行算法,并行算法需要在初始化、任务分配、子任务并行执行、子任务汇

总等几个方面做工作。下面对这几方面进行介绍。

5.3.1.初始化

1. 传送参数

在 C 语言编写的程序中,主程序 main 接受参数 argc 和 argv。要让所有的节点得到输入的参数,就得通过 MPI_INIT 函数的参数 argc 和 argv 传送给每一个参与计算的节点。

```
MPI_Init(&argc,&argv);
```

2. 节点统一编号

由于参与计算的节点数目较多,节点进行协同工作不可避免进行信息通讯和数据交换,为了便于区分,所有参与计算的节点要统一编号。使用 MPI_COMM_RANK 函数的让每一个参与计算的节点确定自己 ID, 并自己的 ID 号存放在变量 MyID 中。

```
MPI_Comm_rank(MPI_COMM_WORLD,&MyID);
```

通过 MPI_COMM_RANK 函数获得的 ID 从 0 开始依次编号。如在 16 个节点的集群系统中,主节点为 0,其余子节点分别为 1, 2, 3.....15。

3. 获得节点总数

使用 MPI_COMM_SIZE 函数使每一个节点都获得参与计算的节点总数。

```
MPI_Comm_size(MPI_COMM_WORLD,&SumID);
```

4. 获得求解问题

主节点通过 MPI_BCAST 函数将待排序序列信息(长度和每一个元素的值)广播给其他所有节点。

```
MPI_Bcast(&DataSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
MPI_Bcast(&IN, DataSize, MPI_INT, 0, MPI_COMM_WORLD);
```

由于广播序列长度的工作只有主节点才做,所以这次语句只在主节点上执行,其他节点不执行,对于集群系统来说,就是只在编号为 0 的节点上执行,即:满足 MyID 为 0 的条件时执行。

5.3.2.任务分配

主节点一项重要的任务就是要根据求解问题和参与集群节点情况为每一个节点分配任务。

任务分配是否合理直接影响了集群系统求解问题的效率,直接体现在求解问题所需时间上。由于集群系统是多节点协同工作,根据木桶原理,总工作时间取决于耗费时间最长的节点所需时间,所以,为了最大的提高性能,缩短工作时间,必须使耗时最长的节点所用的时间尽可能的短。任务分配时就是解决负载均衡问题,根据每个节点各自的实际情况

为其分配任务。如果忽略节点之间的差异，认为所有计算节点的性能相同，则要求任务分配尽量的平均。

若平均的分配任务，则一般情况下，每个节点的任务数=任务总数/节点数。这是一种理想状态下的任务分配，任务正好平均分配给每个节点，但多数情况下，并不是这样，任务数不能被节点数整除，存在多余的任务而不能够平均分配。问题在于，多余的任务交给谁来完成。

一般采用的并行算法是将多余的任务交给最后一个节点。假设节点数为 N ，为每个节点分配 k 个任务后剩余 m 个，则按照这样的分配方法，则将多余的 m 个任务分配给最后一个节点。如此以来，最后一个节点需要完成 $k+m$ 个任务，其他节点需要完成 k 个任务，则各个节点之间的任务差为： m 。

在枚举排序中，则意味着最后一个节点要比其他节点多求解 m 个元素的秩。假设待排序序列有 P 个元素，则求解一个元素的秩需要进行 P 次比较运算，那么，多求解 m 个元素的秩就要多做 Pm 次比较运算。

最糟糕的情况下，差一个不能够平均分配， $m=N-1$ ，最后一个节点要比其他的节点多进行 $P(N-1)$ 次比较运算。

本课题在考虑任务分配时，采用了一种改进的分配策略，不是将多余的任务交给某一个节点而是平均分配给了一部分节点，即：将多余的 m 个任务分配给 m 个节点，每个节点再分配 1 个任务。这样一来，各个节点之间的任务差最大为 1 个，任务分配更加平均。

设 $MyLength$ 为节点分配到的任务数，总任务数 $DataSize$ ，节点总数 $SumID$ ，则首先进行第一次分配：

$$MyLength = DataSize / SumID;$$

然后根据是否有多余的任务来决定是否进行二次分配。

$$m = DataSize \% SumID;$$

如果 m 等于 0，则表示没有多余任务，否则表示还有多余的任务没有分配完，需要进行二次分配。二次分配时将多余的 m 个任务分配给前 m 个节点，也就是编号从 0 到 $m-1$ 的节点。

对于枚举排序来说，一个节点分配了 $MyLength$ 个任务就是说它需要求解 $MyLength$ 个元素的秩，但是并不知道是哪 $MyLength$ 个元素。为了使每个节点明确自己的任务，按照节点 ID 从第一个元素开始分配，给每个节点分配 $MyLength$ 个元素。需要注意的是， $MyLength$ 的值并不是相同的，若前 m 个节点的 $MyLength=k$ ，则从第 m 个节点开始， $MyLength=k-1$ ，所以，设每个节点负责从第 s 个元素开始到 t 个元素为止这 $MyLength$ 个元素，则可以通过以下代码实现：

```
if(MyID < m)
    s = MyLength * MyID;
else
    s = MyLength * MyID + SumID - m;
```

```
t=s+MyLength;
```

5.3.3.子任务求解

经过初始化和任务分配之后，每个节点都明确了自己任务，接下来就可以进行任务求解了，计算第 s 个元素开始到 t 个元素为止这 $MyLength$ 个元素的秩，并将结果相应的存入长度为 $MyLength$ 的秩序列中。

```
for(i=s;i<t;i++) /* 计算每一个元素的秩，存在秩序列中 */
{
    k=0;
    for(j=0;j<DataSize;j++)
    {
        if(IN[i]>IN[j])
            k=k+1;
        else if(IN[i]==IN[j] && i>j)
            k=k+1;
    }
    Rank[i-s]=k;
    if(MyID==0)
        printf("%ld^%d\t",i,t);
}
```

5.3.4.子任务汇总

所有节点完成各自任务之后，由主节点进行汇总。对于枚举算法就是将每个节点求解出来的秩序列合并，得到完整的秩序列。

子节点通过 `MPI_SEND` 函数向主节点发送自己求解得到个 `Rank` 序列，同时主节点通过 `MPI_RECV` 函数逐个收集子节点的 `Rank` 序列，根据子节点的 `ID` 和 `Rank` 序列长度将所有的 `Rank` 序列依次拼接起来，最终得到完整的秩序列。

这里需要注意的是拼接次序和位置必须准确无误，否则最终结果必定不正确。完整的秩序列得到之后，枚举排序并行算法的并行化也就完成了。

在集群系统上使用 16 个节点运行并行程序，对 30 万个随机生成的整数进行排序，所需时间仅为单机运行串行程序耗费时间的约 10 分之一，效率提高近 10 倍。

6. 结束语

在本课题实施中, 作者对高性能计算集群的构建和应用做了一些研究和实验, 利用普通 PC 在 Linux 环境下构建了基于 MPICH2 的集群系统, 进行了高性能 Linpack 测试, 在此集群系统中进行了枚举排序算法的并行化应用。实验结果表明这种构建高性能计算集群系统的方法切实可行。HPL 基准测试证明, 影响这种集群系统性能的主要参数问题空间的大小 (N_s)、LU 分解数据块大小 (NB_s), 处理器网格组成方式 (P, Q), 影响集群系统整体性能的主要因素是计算节点的中央处理器 (CPU)、内存以及网络架构 (主要是网络带宽和延迟)。枚举排序算法的并行化实现证明, 高性能计算集群不但可以节省计算时间, 提高计算精度, 而且可以解决单机无法完成的超大规模求解问题, 是低成本解决超级计算能力需求的有效途径。

由于研究时间较短, 作者构建的集群系统还很不完善, 节点机性能都还没有充分的发掘出来, 节点间的负载均衡问题还需改进, 集群系统的管理也很不方便, 这些都是下一步研究的内容。

7. 致谢

首先，我要衷心的感谢我的导师张璟教授。两年多来，张老师在工作、学习和生活上都给予了我悉心的指导和亲切的关怀。张老师学识渊博，在科研方面认真严谨，一丝不苟，在教学方面严格负责而又耐心细致，使我受益匪浅。

其次，要感谢网络工程系的张彤副教授、孙钦东副教授、王彬老师、张志强老师、宋昕老师，软件系的张云峰老师，计算机科学系的王海晟老师，网络实验室的吴孟斌老师和何文娟老师。张彤老师在这两年的工作、学习和生活方面都给予了亲切的关心和莫大的帮助，在我编写算法程序的过程中遇到的一些问题也提出了很有帮助的建议；由于在做论文之前没有接触过 Linux，在我学习 Linux 的过程中，孙钦东老师、张云峰老师和王海晟老师给予了很多指导；王彬老师和张志强老师在我进行集群构建的时候提出了许多很好的建议和意见；宋昕老师在外文资料的查阅方面给予了很多帮助。感谢网络实验室为我提供了构建集群所需的全部硬件设备，由于这些设备都是平时学生上机需要使用的，为了不和正常教学活动发生冲突，经常在非工作时间使用实验室，吴孟斌老师和何文娟老师在整个实验期间为我提供了很多的便利条件，在此深表谢意。

还要特别感谢我的妻子，一直以来她默默的承担了家里的一切事务，无微不至的照顾我的生活，使我可以专心于学习和工作。没有她的充分的理解和支持，我甚至无法顺利完成研究生的学业。

最后谨向所有关心、支持和帮助过我的老师、同学、朋友们致以最衷心的感谢！

8. 参考文献

- 【1】 <http://www.top500.org/>
- 【2】 <http://www.beowulf.org/>
- 【3】 <http://www.beowulf-underground.org/>
- 【4】 曙光天潮 TC4000A 技术白皮书
- 【5】 陈国良. 并行计算—结构、算法、编程[M]. 北京: 高等教育出版社, 2004
- 【6】 孙世新, 卢光辉等. 并行算法及其应用[M]. 北京: 机械工业出版社, 2005
- 【7】 Ananth Grama. 并行计算导论[M]. 张武等译. 北京: 机械工业出版社, 2005
- 【8】 李代平等著. 分布式并行计算技术[M]. 北京: 冶金工业出版社, 2004
- 【9】 都志辉. 高性能计算并行编程技术-MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001
- 【10】 罗省贤, 何大可等. 基于 MPI 的网络并行计算环境及应用[M]. 成都: 西南交通大学出版社, 2001
- 【11】 <http://www.csm.ornl.gov/pvm/>
- 【12】 <http://www-unix.mcs.anl.gov/mpi/>
- 【13】 <http://www.mpi-forum.org/>
- 【14】 http://www.ccw.com.cn/server/jssc/htm2004/20040715_10QY6.htm
- 【15】 车静光. 微机集群组建、优化和管理[M]. 北京: 机械工业出版社, 2004
- 【16】 屈钢, 邓健青, 韩云路. Linux 集群技术研究[J]. 计算机应用研究. 2005.5
- 【17】 (美)Rajkumar Buyya. 高性能集群计算、结构与系统(第一卷)[M]. 郑纬民等译. 北京: 电子工业出版社, 2001
- 【18】 (美)Rajkumar Buyya. 高性能集群计算、编程与应用(第二卷)[M]. 郑纬民等译. 北京: 电子工业出版社, 2001
- 【19】 (美)Alex Vrenios. Linux 集群体系结构[M]. 马朝晖等译. 北京: 机械工业出版社, 2003
- 【20】 <http://www.pccluster.net>
- 【21】 <http://www.redcluster.net>
- 【22】 李艳. 集群环境下的并行编程模型和开发工具. 红色集群-李艳专栏.
http://www.redcluster.net/download/material/200409/parallel_program_develop.doc
- 【23】 <http://www-unix.mcs.anl.gov/mpi/mpich2/>
- 【24】 <http://www.netlib.org/linpack/>
- 【25】 <http://www.OpenPBS.org/>
- 【26】 <http://www.pbspro.com/>
- 【27】 陈国良. 并行算法实践[M]. 北京: 高等教育出版社, 2004
- 【28】 马昕炜. Linux 服务器配置手册[M]. 北京: 科学出版社, 2005

- 【29】 Christopher Negus(美). Linux 宝典[M]. 徐小青等译. 北京: 电子工业出版社, 2005
- 【30】 迟学斌. 2003 年高性能计算培训班材料. <http://lssc.cc.ac.cn/training2003>
- 【31】 孟庆平、刘淘英、李恪. 机群系统管理的研究和实现[J]. 计算机科学. 2003
- 【32】 陈熠, 孟丹. 大规模机群监控系统的研究与实现[J]. 国家智能中心, 2004
- 【33】 (英) Neil Matthew. Linux 高级编程[M]. 叶小虎, 龙浩等译. 北京: 机械工业出版社, 2002
- 【34】 (美) Barry Wilkinson, Michael Allen. 并行程序设计[M]. 陆鑫达等译. 北京: 机械工业出版社, 2002

9. 附录

9.1.可控管理部分脚本（部分）

可控管理的脚本根据执行模式的不同大体可以分为两大类，一类是完全靠执行 shell 命令来完成操作，另一类是没有相应的 shell 命令，只能够通过修改配置文件来实现。对于前者，比较简单，只需要编写 shell 脚本，直接调用 shell 命令即可。对于后者，由于只能通过修改配置文件来实现，所以只能先修改好配置文件，然后通过 shell 命令将子节点上的对应文件替换掉。可以针对各种需要事先编写好相应的配置文件，需要的时候直接用其替换，也能大大减轻工作量。

下面给出四种操作的脚本实现，前两种是通过直接调用 shell 命令实现，后两种是通过替换配置文件实现。

● 重新启动

```
#!/bin/sh
for i in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
do
    sleep 1
    echo " Reboot node$i ....."
    ssh node$i /sbin/reboot
done
```

● 关机

```
#!/bin/sh
for i in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
do
    sleep 1
    echo " Halt node$i ....."
    ssh node$i /sbin/halt
done
```

● 更新 SSH 登录配置

```
#!/bin/sh
for i in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
do
    sleep 1
    echo " Initing node$i SSH....."
    scp node0:/root/.ssh/* node$i:/root/.ssh
done
```

● 设置双系统开机默认选择

```
#!/bin/sh
for i in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
do
    sleep 1
    #echo "Initing node$i boot with Liinux....."
    #scp node0:/home/clu/config/grub_linux.conf node$i:/boot/grub/grub.conf
done
```

9.2.枚举排序串行程序源码（部分）

枚举排序的串行程序比较简单，在单个节点上执行，不需要进行任务的分配汇总等工作，下面是部分核心代码。

```
IN=(int *)malloc(DataSize*sizeof(int));    /*为待排序序列分配内存空间 */
OUT=(int *)malloc(DataSize*sizeof(int)); /*为输出序列分配内存空间 */
srand(unsigned)time(NULL));
for(i=0;i< DataSize;i++)
{
    IN[i]=((int)rand())%10000; /* 生成随机数 */
}
time (&start);
for(i=0;i< DataSize;i++)
{
    k=0;
    for(j=0;j<Max;j++)
    {
        if(IN [i]> IN [j])
            k=k+1;
        else if(IN [i]== IN [j] && i>j)
            k=k+1;
    }
    OUT[k]= IN [i];
}
time (&end);
dif = difftime (end,start);
```

```

printf("\n");
for(i=0;i<DataSize;i++)    /* 输出排序结果 */
{
    printf("%d\t",OUT[i]);
}
printf("\nUse %.4lf seconds to Rand Store.\n", dif);
getch();
}

```

9.3.枚举排序并行程序源码（部分）

● 初始化

这部分代码完成初始化工作，包括确定参与计算的节点总数以及各节点处理器的编号、广播初始化参数，输入序列初始化等。

```

MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&MyID);    /*每个处理器确定各自 ID*/
MPI_Comm_size(MPI_COMM_WORLD,&SumID);    /*每个处理器确定总处理器个数*/

if(MyID==0)
{
    printf("\nPlease input the DataSize length of the Array:");
    scanf("%ld",&DataSize);

    printf("\nPlease the Seed:");
    scanf("%ld",&Seed);

    time (&start);
    printf("\nRand Numbers:\n");
}

IN=(int *)malloc(DataSize*sizeof(int));/* 为待排序序列分配内存空间 */

if(MyID==0)
{
    srand(Seed); /* 生成随机数种子 */
    fp=fopen(argv[1],"w+");
    for(i=0;i<DataSize;i++)
    {
        IN[i]=((int)rand())%10000; /* 生成随机数 */
    }
}

```

```

        printf("%d\t",i,IN[i]);
        fprintf(fp,"%d\t",IN[i]); /* 将生成的随机数输出到待排序序列存放文件 */
    }
    fclose(fp);

    printf("\nBroadCast to clusters:\n");
}

```

MPI_Bcast(&DataSize, 1, MPI_INT, 0, MPI_COMM_WORLD); /*主节点将待排序序列长度广播给其他所有节点*/

● 划分任务

主节点根据参与作业任务总量和计算的节点总数情况为子节点分配任务、各节点根据自己的作业任务初始化秩序列和输出序列。

```

/*在各个处理器间划分任务*/
MyLength=DataSize/SumID; /*每个处理器确定各自要排序的序列长度*/
m=DataSize%SumID;
if(m>0)
{
    m=(MyLength+1)*SumID-DataSize;
    if(MyID<SumID-m)
        MyLength=MyLength+1;
}

```

MPI_Bcast(&IN, DataSize, MPI_INT, 0, MPI_COMM_WORLD); /*主节点将待排序序列广播给其他所有节点*/

```

if(MyID==0)
{
    OUT=(int *)malloc(DataSize*sizeof(int)); /* 主节点为输出序列分配内存空间 */
    Rank=(int *)malloc(DataSize*sizeof(int)); /* 主节点为秩序列分配内存空间 */

    printf("\nStart Sorting.....\n");
}
else
{
    Rank=(int *)malloc(MyLength*sizeof(int)); /* 子节点为秩序列分配内存空间 */
}

```

● 执行任务

各节点明确了自己的作业任务，完成了初始化工作，开始执行。

```

if(MyID<SumID-m)
    s=MyLength*MyID;
else
    s=MyLength*MyID+SumID-m;

```



```

t=s+MyLength;

for(i=s;i<t;i++)    /* 计算每一个元素的秩,存在秩序列中 */
{
    k=0;
    for(j=0;j<DataSize;j++)
    {
        if(IN[i]>IN[j])
            k=k+1;
        else if(IN[i]==IN[j] && i>j)
            k=k+1;
    }
    Rank[i-s]=k;
    if(MyID==0)
        printf("%ld^%d\\t",i,t);
}

```

● 任务汇总

所有节点都完成了各自的作业任务之后，主节点从子节点获取作业任务的计算结果，并汇总处理，得到最终的计算结果。

```

/*从各个处理器收集已排序好的数据*/
if(MyID==0)
{
    printf("\n OK \n He bing.....\n");
    for(i=1;i<SumID;i++)
    {
        if(i<SumID-m)
            MPI_Recv(Rank+MyLength*MyID,MyLength,MPI_INT,i,0,MPI_COMM_WORLD,&status);
        else
            MPI_Recv(Rank+MyLength*MyID+SumID-m,MyLength-1,MPI_INT,i,0,MPI_COMM_WORLD,&status);
    }
}
else
    MPI_Send(Rank,MyLength,MPI_INT,0,0,MPI_COMM_WORLD);

if(MyID==0)
{
    printf("\n OK \n Saving.....");

    fp2=fopen(argv[2],"w+");
    for(i=0;i<DataSize;i++)    /* 按照计算所得秩,将每一个元素写入输出序列 */
    {
        OUT[Rank[i]]=IN[i];
    }
}

```

```
    }

    for(i=0;i<DataSize;i++)    /* 写入到输出序列存放文件 */
    {
        fprintf(fp2,"%d\t",OUT[i]);
    }

    time (&end);
    dif = difftime (end,start);
    printf ("\nUse %.4lf seconds to Rand Store.\n", dif);
    fclose(fp2);

}

MPI_Finalize();
```

10. 在校学习期间所发表的论文

王勇超 张璟 李军怀. 基于 MPICH2 的高性能计算集群系统研究. 2006 年西安理工大学研究生学术年会.