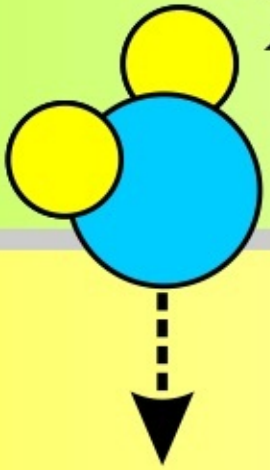
计算化学

集群构建教程



张鋆 编

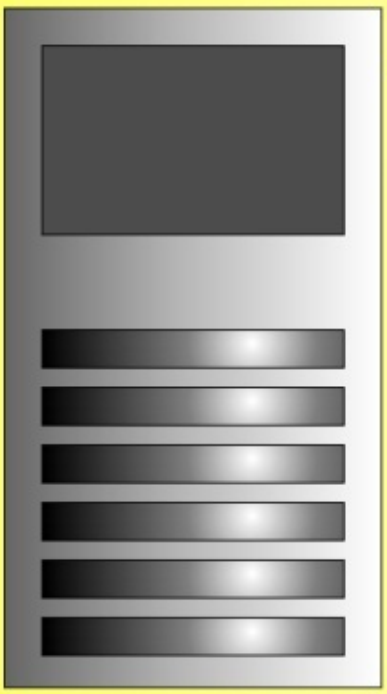
$$FC=SC\varepsilon$$

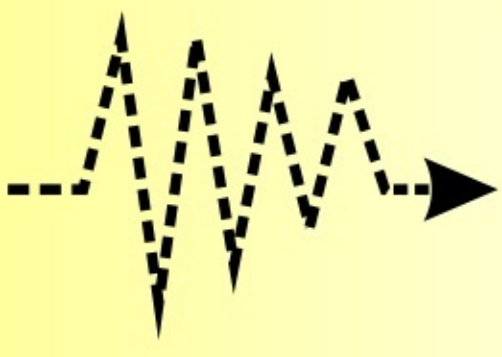
$$\mathbf{R}(t+\delta t)=2\mathbf{r}(t)+\mathbf{r}(t-\delta t)+\delta t^{2}\mathbf{a}(t)$$

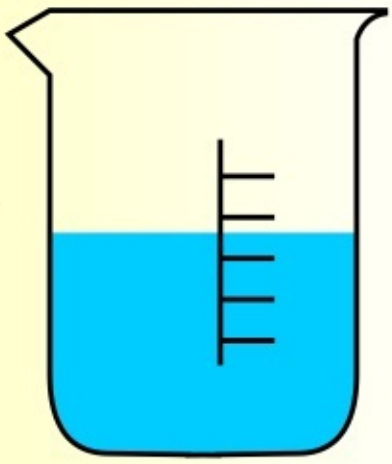
Scientist@compuchem:~$ genuine < H2O > water

Scientist@compuchem:~$ genuine < H2O > water

```
do 99,J=I+1,N
   vx=vxi - rx(J)
   vy=vyi - ry(J)
   ...
If (error .lt. thresfold)
   call EDIIS (...
else
```







计算化学

集群构建教程

张鋆 编

前　言

本教程的目的是为读者提供计算机集群组建及其在计算化学中的应用的入门知识。我们将先讲一些基本概念，然后以最快的速度进入实践阶段，使大家能够在最短的时间内迅速建立起一个可以从事高性能计算的集群。

本教程的主要内容是以普通的高端 PC，服务器，千兆网等多数实验室所能负担的起的硬件为基础，介绍如何构建一个具有完整文件共享，并行计算和作业排队与调度功能的高性能计算机集群。这个教程，可以为从来没有组装过集群的读者提供一些入门的知识，从而可以在组内构建自己的集群而免去在别的地方“交钱”“排队”的苦恼。

有人会问：自己组建的集群和付费购买机时的集群会不会性能有很大差异？这个严格来说取决于读者研究的体系。从单机计算的能力来看，高端 PC 已经和最好的单机相差不大，而并行方面，由于专业机构有高水平的优化工程师和性能卓越的网络硬件（当然这一切是以高成本为代价的），普通的研究组一般难以到达它们的水平。但是，对于计算化学科研领域，大多数组需要计算的体系，即使是很大蛋白质的长时间动力学（如 100000 原子的 10ns 动力学这种任务），借助本教程里面的技术，即 MPI 并行+GPU 加速+自行编译，有两台 8 核 CPU 高端 PC，就基本可以在 10 天以内完成。因此，作者认为没有必要再去追逐更昂贵的硬件。毕竟，科研的灵魂是 *idea*，而不是比谁的体系大，谁的速度快。当然，对于特别的领域，如神经系统蛋白质的模拟，需要 512CPU 并行的，本教程里的技术就略显不足了，不过依然可以作为一个入门来介绍。

本教程选取的软件技术如 Debian，Torque，NFS 等都是比较简单而实用的软件，都经过作者测试。这些东西足以满足大多数要求，而且网上的文章也很多，遇到错误便于随时查阅。专业大型集群的技术则更加高级，如我们使用 NFS 来

共享同步文件，而大型集群则会建立专门的“升级服务器”等来实现。这些技术读者如有兴趣可以参考这方面的资料。

在本教程的第三章还介绍了一些软件安装，编译和编程环境的设置的知识。一些计算化学的初学者常常面对软件的编译无从下手（如某人曾经半年也没编译出 SAPT2008 来）。本来作者想将编译过的全部软件介绍一下，但是这样会使本教程变为一个无聊的软件集锦。所以这一章以 Gaussian，NAMD，Autodock 和 MKL 为例，分别介绍了四个类型的计算化学软件的配置技术。通过这四个软件的配置的练习，再面对新的软件，读者就应该有足够的经验来应对了。如果还有困难，可以参考网上的文章，如作者的博客：http://hi.baidu.com/coolrainbow/home。

本书并没有专门介绍 Linux，NIS，NFS 系统或者 Torque，Maui 软件的知识，读者可以参考相应的教材或者网上的资料，最重要的则是软件本身的**文档**，这个非常重要！

这个文档最早写成于 2009 年 5 月，曾在小范围内传播过。后来在 2010 年左右因集群升级，添加了一些新技术。在 2011 年 5 月时再次重新整理并定稿。希望这个教程能够为读者提供一些帮助！如果读者在教程中发现了错误并对作者进行指正，作者将不胜荣幸！

作者：张鋆，南开大学(coolrainbow@yahoo.cn)

2011 年 5 月 29 日

目　录

第一章 绪论.....1
§1.1 计算机集群的基本概念.....1
§1.2 计算机集群的软件结构.....4
第二章 集群的系统配置.....7
§2.1 操作系统的安装.....7
§2.2 网络通信系统的建立.....9
§2.2.1 SSH：主机间无密码的远程登录.....10
§2.2.2 NFS：主机间的文件共享.....11
§2.2.3 NIS：主机间的用户同步.....13
§2.3 编译器的安装.....15
§2.3.1 GNU编译器.....15
§2.3.2 intel编译器.....16
§2.4 并行计算环境的配置.....17
§2.5 作业系统的建立.....19
§2.5.1 Torque的安装.....20
§2.5.2 Maui的安装.....20
§2.5.3 Torque和Maui的配置.....21
§2.5.4 mpiexec的安装.....23
§2.6 第一次运行集群.....23
第三章 集群的计算化学软件配置.....26
§3.1 Gaussian03 安装与配置.....26
§3.1.1 Gaussian03 的安装.....26
§3.1.2 Gaussian03 的单机配置.....27
§3.2 NAMD安装与配置.....28
§3.2.1 组件安装.....29
§3.2.2 charm++安装.....29
§3.2.3 NAMD编译.....31
§3.2.4 NAMD运行.....32
§3.3 Autodock的安装与配置.....33
§3.4 MKL编程环境的构建.....34
第四章 集群的管理技巧.....38
§4.1 用户管理.....38
§4.1.1 用户权限.....38
§4.1.2 高效的管理.....38
§4.2 作业管理.....39
§4.2.1 管理员管理.....40
§4.2.2 普通用户管理.....43
第五章 GPU在集群上的应用.....45
§5.1 GPU/CUDA的安装配置.....45
§5.1.1 驱动程序的安装.....45
§5.1.2 程序开发环境.....45
§5.1.3 CUDA编程测试.....46

§5.2 NAMD CUDA版本的安装 46
继续前进! 49

第一章 绪论

§1.1 计算机集群的基本概念

正如题目所示，本教程的目的是为读者提供计算机集群组建及其在计算化学中的应用的入门知识。可能对大多数读者而言，“集群”什么的是一个全新的概念——没关系，万丈高楼平地起，我们先讲一些基本概念，然后将以最快的速度进入实践阶段，使大家能够在最短的时间内迅速建立起一个可以从事高性能计算的集群，为大家的科研提供帮助。

本教程主要面向从事计算化学的读者（当然也包括其它有计算机集群需要的读者），相信大家知道，计算化学研究中常常需要巨大的计算量，无论对CPU速度、存储容量和总线带宽都要很高的要求。例如，conventional SCF计算中，一个有 1000 个基函数的体系，所需要计算的双电子积分量约为 10^{12}个，按双精度存储需要 1.8TB，对于计算速度和存取速度都是极大的负担；在对神经系统的分子动力学模拟中，钾通道动态工作的时间尺度为毫秒级，而即使利用 500 核的Alpha机器，也只能达到每天 4 纳秒的速度。这样的速度是人们无法接受的。

解决这些计算问题的方法，从理论上应当发展新的更高效的算法，诸如量子化学中的线性标度方法等等；从硬件角度说，需要发展更新的硬件，但由于物理定律的限制，单个计算机的硬件能力已经接近极限，例如 CPU 主频的提高是以发热量的增加为代价的，而现代 CPU 的主频已达到 4.00GHz 左右（实验室产品），更高的主频由于发热量巨大而难以实现。从计算机的角度上来说，可以通过使用超级计算机来解决。但是，这类超级计算机，硬件研发费用昂贵，软件投资更加高居不下，开发出来的软件无法移植，并且学习代价非常高，非一般研究组所能负担的起。

那么对于一般的研究组，如何才能突破计算瓶颈呢？答案就是本教程的核心概念：**计算机集群**（computer cluster）！

相信很多实验室内部都有局域网，即一大堆计算机通过路由器等连成一个整体，大家可以通过路由器上外网，而内网中，大家可以通过网络共享文件，如Windows 下的“共享”。好了，既然局域网可以共享文件，为什么不可以共享**计**

算能力呢？好了，我们学院派的定义一下**计算机集群**：通过一组松散集成的计算机软件和硬件连接起来高度紧密地协作完成计算工作的计算机系统，可以被看作是一台计算机，这种系统就称为计算机集群。也就是说，只要大家有几台计算机（PC 或服务器），一个路由器，几根网线，我们就可以把它们连接起来，通过一些软件，使它们的计算能力得以共享，实现并行计算（parallel computing），不仅可以实现高速的计算，而且可以使计算工作**规范化**，从而大大的提升科研效率！

如图 1-1-1 所示，这是作者所在实验室的组建的一个集群的实景（摄于 2009 年）。这个集群现在已扩充升级。

图 1-1-1 集群系统实景图（摄于 2009 年）

下面我们讲几个计算机集群中常用的术语和概念。这些概念可以为读者组建集群打下坚实的基础。

节点（node）。集群中单独的每一台计算机称为节点。这些计算机，既可以是普通 PC 也可以是服务器。按照现在的发展（2011 年前后），高端 PC 和服务器的计算能力已经相差无几，而 PC 的成本较低，不失为是个很好的选择。服务器的优点在于可靠性高，即平均无故障时间长。比如，一个 500 台计算机构成的集群连续运转一年，若全部为服务器，一个月平均可能只有几台机器出故障，若

全部为 PC，则可能平均每一天都会有机器崩溃。当然，对于普通研究组而言，情况没有那么夸张。读者根据自己的需要决定是选择 PC 还是服务器。

控制节点。集群中需要多台计算机相互协作的完成任务，因此必须有一台计算机作为“核心”领导者，控制其它的计算机，为其它的计算机分配计算任务，调整负载等等。这个节点习惯称为控制节点。一般而言，控制节点可以选一台 CPU、内存一般（512MB 内存足矣！）的机器以节约成本，但是必须有大容量的硬盘来存储用户的文件，即使是很小的组也一般最好是 1TB 以上（貌似更小容量的硬盘已经没有卖的了），根据本组的人员数量和科研任务决定，作者知道某个分子动力学研究组每个月都会有几个 TB 的轨迹文件产生。

计算节点。计算节点就是集群中直接进行计算任务的计算机，这是决定计算机集群性能的最关键因素之一。自然，CPU 主频越高，内存越大，总线带宽越高，对计算就越有利。当然，具体的任务对这些硬件的要求不同。对于分子动力学模拟计算，内存的要求很低，一般 1～4GB 的内存就可以，但是要求 CPU 具有强大的计算能力，最好在 8 核（或双 4 核）、主频 2.5GHz 以上。对于量子化学计算，除了要求有强劲的 CPU 外，还要求有高容量和高读写速度的内存、硬盘，因为量子化学计算常常需要大容量的缓存文件，如果存储容量不够，会对计算速度带来很大的影响。通常，内存最好在 16GB 以上，如果经常计算金属有机化合物，内存 32GB 也不嫌多；硬盘最好在 1TB 以上，特别是需要大量相关方法计算（如 MP2 或 CCSD(T)计算）时，硬盘的读写速度也十分重要。有些计算，诸如分子对接这种计算，对计算机性能要求很低，普通的高端配置即可。

现代计算机有 32 位，64 位等多种架构。64 位架构的机器当然是最好，但是大多数研究组可能无法承担它的购买成本。现在的计算主流是选用虚拟 64 位的机器，如 em64t 和 AMD64 等。本教程就是以虚拟 64 位机器为例介绍的，即所谓 x86_64 或者 amd64 机器。

通信网络。顾名思义，就是集群中连接各个节点的网络。这是决定计算机集群性能的另一个最关键因素。在进行并行计算时，网络速度是整个计算速度的瓶颈。这个网络，当然可以选择使用普通实验室中的局域网连接即“宽带网加路由器”结构。集群中，这个网的带宽最低不能低于百兆，即至少在千兆以上，再低的网在计算时可能会出显各种由于网络延时过长而导致的错误。如果读者能使用

高性能的集群专用网络，如 Myrinet 或者 InfinitBand，那当然更好，这可以大大地提升集群的并行效率！不过，这是要很高的成本的。

维护部件。这是维持集群能够正常工作的部件，包括散热系统如空调和电源、机柜等。因为集群常常要长时间高功耗的工作，如果不能保持适当的温度和稳定的电源，计算机内部出错的概率就会大大的提高，轻则死机导致作业失败，重则烧毁硬件造成损失。特别是热天，集群高速运转时会有很高的温度，一定要做好降温的工作；如果有断电通知，记得事先将总插头拔下，以免恢复供电时瞬时电流的冲击将机器烧毁。

好了，下面我们来讨论一下构建集群所需要的软件基础。

离开之前，我们必须声明，本教程介绍的是面向科学计算用户的“高性能集群（High Performance Cluster，HPC）”，而不是面向企业的“高可用集群（High Availability Cluster，HAC）”。后者常常用于建立企业网站，服务器，大型数据库（如 Oracle 数据库）。这两者，应该来说，完全不同！

§1.2 计算机集群的软件结构

有了硬件后，要真正实现集群，还必须有优秀的软件进行支持。集群的软件结构包括以下几部分：

操作系统。虽然世界 top100 的集群中确实有 Windows 集群，但是我们强烈推荐读者使用 Linux 操作系统。Linux 免费，方便，最重要的是效率高，很多计算化学软件都是专门为 Linux 编写的，而且 Linux 非常适合于程序开发和批量处理，这都是 Windows 难以实现的，即使能，往往也比较臃肿。

Linux 操作系统也有很多不同的分支。比较常见的就是 RedHat 系列，但是本教程不使用它。我们选择 Debian 操作系统，与 RedHat 相比它有很多优点，最大的优点是体积小，安装完系统后只有不到 100MB，并且随时可以扩充新的功能，没有冗余的文件，因此非常适合于科学计算。关于 Debian 可以在网上找到很多信息。本教程不会直接讲解 Linux 或者 Debian 的使用方法，对 Linux 不熟悉的读者可以参考：

王旭。Debian 标准教程。北京：人民邮电出版社，2009

关于哪个版本的 Linux 更适合组装集群是一个老掉牙的问题，和“C 和 C++哪个好”一

样，作者认为这是一个特别主观的问题。有很多人推荐过别的 Linux，如：

OpenSuse。这个系统很好用，但是作者不喜欢，因为它的系统太臃肿。

RedHat。“正统”的 Linux，而且带有 Cluster Tool。但是作者认为那个 Cluster Tool 不适合做 HPC，仅仅适合 HAC。

Rocks。这个系统非常适合做生物计算的集群，但是因为种种原因，作者后来没有使用它，还是因为它太大了！

当然，如果读者的个人偏好和作者不同，也没有关系。大多数操作在各个 Linux 间是兼容的。

网络通信系统。这是集群得以成为“集群”的核心系统。没有这个系统，集群只是几台互不相干的计算机组成的“机群”。这里的网络通信系统，不止是 TCP/IP 协议等网络连接功能，还包括文件共享系统、远程登录系统、信息管理系统等，只有这样才能使集群中的计算机成为一个相互协作的整体。这些功能可以直接使用 Debian 操作系统的组件完成。

并行计算系统。这是实现高性能计算的平台。并行计算系统是实现某种并行计算协议的软件，各种并行计算的程序都要调用并行计算的函数和库才能运行，例如量子化学软件 GAMESS，NWChem 和分子动力学软件 NAMD,Gromacs，Amber 等。它本质上是一种通信协议。本教程将使用 MPICH2 作为并行计算系统。

作业管理系统。这是管理集群的工具。它负责计算作业的提交、管理、控制，集群资源的分配、调度，用户权限的分配、区分等，对于管理、维护特别是使用集群具有重要的意义。本教程使用免费软件 TORQUE 和 Maui 实现。

应用软件。这是集群的实用部分，比如 Gaussian，NAMD 和各种编译器等等。

本教程涉及到的软件，除了 Windows 操作系统和 Gaussian 等以外，所有软件均为免费软件！

在继续之前，有一些话需要交代给读者。Linux 系列的软件和其它免费学术类软件都是自由开发的，因此，它们的不同版本之间可能有某些不兼容情形，一些安装时候发生的状况和本教程所描述的可能有很大的不同。作者曾经帮人装过 NWChem，因为作者本人安装过 NWChem5.2，没有遇到过什么问题。可是别人

安装的是 6.0 版本，在安装的时候碰到了无数编译错误，最后通过作者手动修改了 NWChem6.0 的源码才得以解决（有经验的读者应该意识到这是编译器版本的问题）。读者在继续之前，应该意识到在组建集群的过程中可能会遇到巨大的困难。这时，可以在网上搜索或咨询身边有经验的人。不要灰心，一旦集群组建完成，读者的计算机水平会有质的飞跃！

好了，现在说一下本教程的集群的各种信息：1 个控制节点，8 个计算节点，千兆局域网，安装 Debian6 系统，安装有；Gaussian, Gamess, NWChem, NAMD, Gromacs, Autodock，以及 GNU 和 intel 编译器等等，这个集群可以实现各种计算化学软件的运行，并提供完整的编程开发功能！

说了那么多，读者读完本教程并搭建完成了一个计算机集群后，对于使用集群的人，集群无非是这样一个东西：

“……他装了个集群，在隔壁房间里，好像是好多台计算机，有啥计算节点控制节点的，好像还装得神马 Linux，我也不懂。我就知道，我每天在我的实验室，打开 Win7，用 putty 登陆到集群，输入命令 qstat，我发现有两个 Gaussian 和一个 NAMD 任务在运行。哦，我前天算的那个 NAMD 的任务已经结束了，算了七天可算完了，把输出文件下下来看看。我这有还个 gjf，传上去，好，算上了。现在关了 putty，分析分析 NAMD 的轨迹吧……”

好了，概念性的东西到此结束，下面，我们组建集群的激动人心的过程就开始了！

第二章 集群的系统配置

§2.1 操作系统的安装

到 2011 年 5 月时，Debian 的最新版本为 6.0，可以在 Debian 社区的网站 http://cdimage.debian.org 上获得其 DVD 镜像文件。根据读者的机器架构下载到相应的镜像，由于我们的机器是 amd64 架构，只需要下载第一个：

debian-6.0.1a-amd64-CD-1.iso

当然，完整的镜像包括 50 多个 iso 文件，但出于我们组建集群的目的只需要第一个就可以。如果读者有兴趣可以全部下载下来。

将第一个 iso 刻录为 DVD 光盘，利用光盘进行安装。注意，安装操作系统的过程中，一定要保证外网的连通，因为安装过程需要在外网下载一些组件。安装操作系统的步骤对控制节点和全部计算节点基本一致，叙述如下。

注意：读者在初学时，可以只装两台机器：一台 compuchem，一台 node021，等把全部细节弄懂之后，再根据自己的需要扩充新的节点。事实上，如果读者的**计算节点的硬件完全相同**的话，完全可以在配置好一台计算节点以后，用 Ghost 软件拷贝到其它所有机器上，然后将其 IP（修改/etc/network/interface）和主机名（修改/etc/hostname）修改成正确的即可。当然，逐台计算机修改 IP 和主机名也可以使用 shell 脚本在控制节点上轻松的实现，因此，工作量不是问题。下面提到的“在全部计算节点上……”是为了让读者明确的知道，哪些操作只在控制节点上做，哪些只在计算节点上做，哪些在全部节点上做。

1. 将 Debian 6.0 的光盘插入光驱，在 BIOS 中选择光盘启动，启动计算机。

2. 如启动成功，屏幕出现 Debian 的 logo，选择 text mode 按 Enter 键开始安装。

3. 语言选择 American，后面的选项根据具体情况选择，这里选择默认选项。

4. 进入 Network 设置界面时，选择 Cancle 取消 DHCP，选择手动安装网络，在随后将出现一系列对话框，分别设置 IP 地址、子网掩码、默认网关、DNS 服务器、主机名、域名。其中 IP 地址、主机名设置如表 2-1-1。其它选项根据实验室的实际情况设定。

表 2-1-1 集群的主机名 IP 地址设置

节点	主机名	IP 地址
控制节点	compuchem	192.168.1.100
计算机点	node021 ~ node029	192.168.1.121 ~ 192.168.1.129

当然，IP 和主机名可以根据读者的喜好来定义。

5. 设置完网络后，会进入分区设置。对于计算节点，可以直接使用默认设置，整体划分为一个区，没有关系。对于控制节点，建议至少划分两个区，一个用于挂载根目录“/”，一个用于挂载家目录“/home”，这个目录作为今后集群用户的工作目录，应该划分的很大，如 500GB 以上。

6. 在用户设定的界面，为 root 定制一个密码，密码最好包含有大小写字母、数字、符号等，这样可增加系统的安全性。对于普通用户，要定义一个合法的用户名（以下以 coolrainbow 为例）以及一个安全的密码（以下假设为 123456）。

7. 分区完成后，继续安装，在选择组件的地方，只选择“Base System”，安装最基本的系统。注意，选了这个选项，我们就安装了一个最小的 Debian 操作系统，并且没有图形界面。

8. 系统会请求选择安装的源（source）。所谓源就是 Linux 在安装时候下载更新文件的地址。这个源非常重要，我们要反复用到，建议选择 Taiwan 或者 Japanese，它们的速度比较快。

9. 稍微等待后，安装即可完成，重启计算机。

至此，系统安装完成。这个过程与网速有关，可能需要不到 20 分钟，也可能需要 1 个小时。

现在需要对所有的节点做些准备工作：

1. 添加域名解析地址。打开/etc/hosts 文件，添加如下信息：

```
compuchem    192.168.1.100
node021      192.168.1.121
node022      192.168.1.122
node023      192.168.1.123
node024      192.168.1.124
node025      192.168.1.125
node026      192.168.1.126
node027      192.168.1.127
node028      192.168.1.128
node029      192.168.1.129
```

这样就建立了相应的解析关系。如 ping node023 就相当于执行 ping 192.168.1.123。

2. 将所有计算节点的普通用户及其家目录删除。也就是说，删除掉 node021 ~ node029 节点上的 coolrainbow 用户及其家目录/home/coolrainbow，使用命令：

```
# userdel –r coolrainbow
```

注意不要删除控制节点的 coolrainbow！

好了，操作系统的安装结束，现在开始安装网络通信系统！

§2.2 网络通信系统的建立

至此，所建立的机器还是互不相干的九台计算机“机群”，而不是相互协作的“集群”。现在的任务是将它们连成一体，成为一个有组织的系统。下面把这个问题详细的说明一下。这部分是整个集群技术的**核心**！

这一节可能会很罗嗦，对于 Linux 系统原理很熟悉的读者，可以跳过某些解释的部分，直接阅读具体的配置步骤，相信读者一看就会明白集群的组建思想，也许能设计出比作者的配置更简单更高效的方法：如果这样，作者将非常高兴。如果读者对 Linux 原理不熟（比如，不知道/etc/fstab 是什么东西），可以一边操作，一边搜索其中的知识（如，google 一下 fstab），这样才能进步。

前面说过，控制节点用来操作计算节点。前面说了，compuchem 是控制节点，node021 ~ node029 是计算节点。实际上，对于用户（而不是像读者一样的集群构建者），控制节点起到了一个黑箱的作用。比如读者的实验室有两个人将要使用集群，名叫 coolrainbow 和 bingtear。当用户 coolrainbow 使用控制节点（比如在 Windows7 下远程登录到控制节点）时，只要告诉控制节点：“在 node023 上用 8 个 CPU 拿 Gaussian 算我这个 C60.gjf”，控制节点就会执行这个任务，用户完全不需要知道任何关于这个过程如何实现的细节。事实上，背后的细节是这样的：

coolrainbow 登录到控制节点 compuchem，并发出了上述指令时，控制节点首先检查 node023 是否可用，如果不可用（比如已经运行了别的任务，或者 coolrainbow 没有权限使用 node023，或者 node023 已经关机，或者根本不存在，）则拒绝执行；如果可用，使用 coolrainbow 的用户名登录到 node023，在 node023

上面执行 Gaussian 算 C60.gjf 的任务。当然，C60.gjf 文件在 compuchem 的硬盘上，控制节点必须把 compuchem 上的 C60.gjf 以某种方式让 node023 得到并运行，且监控这个任务。一旦算完，控制节点就会报告任务结束，并使 C60.out 和 C60.chk 出现在 compuchem 的硬盘上。node023 重新进入了可用状态。Done!

好吧，这个过程看着容易，实际做起来似乎并不那么简单。比如，要想 coolrainbow 能登录各个节点，我们必须从 compuchem 到 node029 挨个建立一个叫 coolrainbow 的相同用户，密码还得全部知道；要执行 Gaussian，还得从 node021 ~ node029 全部安装 Gaussian，路经、配置还得完全一样（不然并行计算会出毛病）；还得设计套拷贝文件的系统，让机器自动把 C60.gjf 拷贝到 node023 上面，算完了再把 C60.out 和 C60.chk 拷贝回来……这个过程太可怕了，倘若集群要把 Gaussian 从 03 升级到 09，那就得 9 台全部重新来了。如果读者的组里面很有钱，有 100 台机器让你升级，嘿嘿……

作者的老师曾经说过一句话：**只要有一件事情用计算机办是很繁琐的，那么肯定有一种技术你没有学会**。没错，无数天才的程序员已经开发了很多优秀的程序来完美的解决上述问题，下面作者将逐步给出解决方法。再次强调：这部分是整个集群技术的核心！

§2.2.1 SSH：主机间无密码的远程登录

第一步，要实现 coolrainbow 在各个节点间自由登录。注意，在§2.1 最后，我们已经在全部计算节点中删除了 coolrainbow，因此 coolrainbow 不能登录到计算节点上。没关系，首先，安装 ssh。

ssh 的英文全称是 secure shell。它能够实现计算机间的远程登录，ssh 把所有传输的数据进行加密，能够防止数据泄密、DNS 和 IP 欺骗，所以是“secure”，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。最初 ssh 是由芬兰的一家公司开发的。但是因为受版权和加密算法的限制，现在很多人都转而使用 OpenSSH。OpenSSH 是 ssh 的替代软件，而且是免费的，可以预计将来会有越来越多的人使用它而不是 ssh。

安装 OpenSSH，在控制节点和全部计算节点执行：

```
# aptitude install ssh
```

这里，aptitude 是 Debian 下安装程序的命令，此时 Debian 会自动在它的网上的源中搜索 ssh 并安装，因此要保持外网通畅。

强烈建议读者搜索一下 aptitude 的用法！我们会反复用到。

如果读者用过旧的 Debian，可能更喜欢 apt-get，没有关系，但是千万不要 aptitude 和 apt-get 同时使用，这会使依赖关系变得混乱！

如果读者就是喜欢用 RedHat 的话，上述命令就得改为 yum install 之类。

有了 ssh 后，现在配置**无密码登录**，以后 coolrainbow 用 ssh 登陆节点时就不用输入密码了。

在控制节点，以 coolrainbow 身份（不能是 root！）执行：

```
$ ssh-keygen –t dsa
$ cd /home/coolrainbow/.ssh
$ cat id_dsa.pub >> authorized_keys
```

它的原理：假设要想实现从机器 A 无密码登录到 B，A 和 B 双方必须互相有一个的密钥。第一个命令即是产生密钥，自动生成在家目录的.ssh 文件夹。然后密钥另一方必须也拥有，这个文件名叫 authorized_keys，因此执行最后一行命令，将密钥导入这个文件中。只要 A 机器的 authorized_keys 和 B 的.ssh/中的内容匹配，就可以无密码的登录。

现在 ssh 本机：

```
$ ssh compuchem
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
RSA key fingerprint is e4:05:08:65:e4:b8:a0:88:cd:be:16:c8:9e:4b:96:87.
Are you sure you want to continue connecting (yes/no)?
```

回答 yes，即可。这样本机到本机便可实现无密码 ssh 登录，以后再次登陆时，就不会出现上面的问题了，也不需要密码了。

有人问：自己到自己 ssh 有什么用，跟计算节点有什么关系？没事，马上就有关系了。

§2.2.2 NFS：主机间的文件共享

第二步，配置文件共享。coolrainbow 要想在 node023 上面运行 Gaussian，首先 node023 上面得有 Gaussian，还得有要运行的文件 C60.gjf 等等。保持这些文件的同步是个很困难的事情。解决办法就是事用 NFS 系统。

NFS（Network File System）即**网络文件系统**，是一种在网络上的机器间共享文件的方法，文件就如同位于客户的本地硬盘驱动器上一样。NFS 服务器可以把文件系统导入给其它系统，NFS 客户端可以挂载从其它机器上导出的文件系统。如果上面的描述看不懂，没关系，我们先把它安装上，然后详细解释。

在控制节点上安装 NFS 服务器：

```
# aptitude install nfs-kernel-server
```

现在，打开/etc/exportfs 文件，这个文件专门用来配置要共享的文件系统。现在在此文件里添加如下内容：

```
/home        192.168.1.0/255.255.255.0(rw,sync,no_root_squash,subtree_check)
/opt         192.168.1.0/255.255.255.0(rw,sync,no_root_squash,subtree_check)
/usr/local   192.168.1.0/255.255.255.0(rw,sync,no_root_squash,subtree_check)
```

这个 exportfs 文件，就是我们要共享文件的配置文件，控制节点 compuchem 会按照这个文件里面的内容，给计算节点共享文件。以第一行为例，解释其中的含义：

/home：要导出的目录。/home 是用户的家目录，必须要共享计算节点，这样我们的文件才能被计算节点获取。我们还共享了另外两个目录：/opt 和 /usr/local。前者是为了共享后面将要安装的 intel 编译器；后者，习惯上，Linux 下的应用软件都安装到这一目录（相当于 Windows 下的 C:\Program Files），整个集群的计算化学软件如 Gaussian 等都会装到这个目录下，因此也要共享出去。

192.168.1.0/255.255.255.0：可以挂载的 IP 范围，这个设置用了一个掩码，表示共享给所有 192.168.1.X 的 IP，我们的计算节点的 IP 是 192.168.0.021 ~ 030，所以计算节点可以共享到。

(rw,sync,no_root_squash,subtree_check)：挂载的配置。这里表示可读可写，同步执行，若 NFS 主机使用分享目录的使用者，如果是 root 的话，那么对于这个分享的目录来说，他就具有 root 的权限，强制 NFS 检查父目录的权限。

重启 NFS 服务：

```
# /etc/init.d/nfs-kernel-server restart
```

现在，这些目录就被挂载出去。但是，计算节点还没有“接手”这些目录，现在，让计算节点将这些目录挂载。打开所有计算节点的/etc/fstab 文件，在文件下面添加：

```
compuchem:/home   /home      nfs     rw,defaults     0       0
compuchem:/opt    /opt       nfs     rw,defaults     0       0
```

```
compuchem:/usr/local /usr/local     nfs         rw,defaults      0          0
```

以第一行为例解释一下这个文件的含义：

compuchem:/home /home：表示把 compuchem 主机的/home 目录挂载到本地的/home 目录上。通俗的说，就是当读者在计算节点上访问/home 时，实际上访问的是控制节点上的/home。这和 Windows 下的共享是类似的。后面几个参数照抄就可以。

然后重启网络服务：

```
# /etc/init.d/networking restart
```

这样子节点就把服务器共享的文件挂载了。例如，在 node021 上，执行如下命令就可以看到.

```
# df
Filesystem              1K-blocks       Used    Available    Use% Mounted on
/dev/sda1               151225248     714456 142828916      1%   /
tmpfs                    1037944          0     1037944      0%   /lib/init/rw
udev                       10240         40       10200     1%     /dev
tmpfs                     037944          0     1037944      0%   /dev/shm
compuchem:/home       480719072     452384 455847488      1%   /home
compuchem:/opt        151225248    1533472 142009888      2%   /opt
compuchem:/usr/local 151225248     1533472 142009888      2% /usr/local
```

看到最后三行了吗？我们已经成功的共享了目录！比如，当我们在控制节点上在/usr/local 下安装了一个 Gaussian03C 时，所有的计算节点都可以通过/usr/local 找到完全相同的 Gaussian，因为它们实际上是通过网络访问的 compuchem 上的/usr/local。NFS 的作用，就是把这些操作屏蔽起来，使计算节点在访问网络上的文件时如同访问本地一样。

好了，coolrainbow 的目录已经通过 NFS 共享到计算节点了，但是 coolrainbow 要想登陆计算节点，计算节点上必须有 coolrainbow 才行。好，现在就来这最后一步！

§2.2.3 NIS：主机间的用户同步

第三步，配置用户信息。这里就要用 NIS。NIS（Network Information Service）即网络信息服务，是集中控制几个系统管理数据库的网络用品。NIS 简化了 UNIX 和 LINUX 用户的管理工作，客户端利用它可以使用 NIS 服务器的管理文件，无

需建立他们自己的/etc/passwd，只简单的使用和维护在 NIS 服务器的文件即可。

1. 在所有机器上安装 NIS：

```
# aptitude install nis
```

安装过程中会要求输入 NIS 的域名，这里必须输入控制节点 compuchem。

配置 NIS 端。以下操作在控制节点上：

2. 确认域名。打开/etc/defaultdomain（这是包含 NIS 域名的文件），确认里面有一行：

```
compuchem
```

如果没有则添上；

3. 打开/etc/default/nis，修改 NISSERVER=master，NISCLIENT=true，这样本机便成为 NIS 服务器；

4. 配置 NIS 主机。打开/etc/yp.conf，写入：

```
ypserver compuchem
```

5. 重启 NIS 服务：

```
# /etc/init.d/nis restart
```

6. 编译用户数据库：

```
# cd /var/yp
# make
```

注意：以后无论控制节点对 passwd，group，shadow 文件做了任何修改，都必须进入/var/yp 重新 make，才能保证服务器和客户端的用户信息同步。

以下操作在计算节点上：

6. 确认域名。打开/etc/defaultdomain（这是包含 NIS 域名的文件），确认里面有一行：

```
compuchem
```

如果没有则添上；

7. 打开/etc/default/nis，修改 NISSERVER=false，NISCLIENT=true，这样本机便成为 NIS 客户端；

8. 打开/etc/yp.conf，通过写入 ypserver compuchem 配置 NIS 主机名；

9. 写入“伪用户”信息，即：

打开/etc/passwd，最后一行写入字符串：+::::::（一个“+”六个“:”）；

打开/etc/group，最后一行写入字符串：+:::（一个“+”三个“:”）；

打开/etc/shadow，最后一行写入字符串：+::::::::（一个“+”八个“:”）；

10. 打开/etc/nsswitch 文件，将

```
hosts          files dns
```

改为

```
hosts          files nis dns
```

11. 重启 nis 服务：

```
# /etc/init.d/nis restart
```

这样整个集群的 NIS 服务都打通了。那么 NIS 究竟有什么用呢？我们知道，控制节点 compuchem 已经有了 coolrainbow 的用户信息，而计算节点如 node021 没有。node021 开启了 NIS，那么它就会自动寻找 NIS 服务器即 compuchem，此时 compuchem 所有的用户信息：用户名，密码，组信息等就都会被 node021 所接受，这样在 NIS 的帮助下，coolrainbow 就成为了 node021 的合法用户，就可以登陆了。

到此为止，本节开头所指出的问题彻底的解决了。用户信息、文件系统都已经在整个集群间共享，并且，ssh 可以在整个集群间无密码登陆了——为什么？很简单，当 coolrainbow 通过 ssh 连接 node021 时（命令：ssh node021），由于 NIS，coolrainbow 成为了 node021 的合法用户，所以有资格登陆；登陆后，系统自动搜索家目录，由于 NFS，这个家目录实际就是 compuchem 上的家目录；而在§2.2.1 中，我们已经在./ssh 中生成了 authorized_keys，所以登陆就不需要密码了（当然，第一次登陆时会遇到提问，只要回答一次 yes 以后就不用了，所以现在**最好把所有的计算节点都 ssh 一下**，这样 ssh 无密码的问题就解决了）。

我们发现，通过 ssh，NFS 和 NIS 这三个相互关联的技术，我们彻底的解决了集群通信的问题。下一步，我们就可以开始安装软件了！

§2.3 编译器的安装

编译器是 Linux 下最重要的软件之一，几乎所有的软件都可以在 Linux 下重新编译，从而达到最高的效率。在本教程中，需要两种编译器：GNU 编译器和 intel 编译器。

§2.3.1 GNU 编译器

我们需要两种编译器，Fortran 和 C 的。非常简单：

```
# aptitude install gcc-4.3 g++-4.3 gfortran
```

此时将安装 C，C++和 Fortran95 的编译器。

§2.3.2 intel 编译器

intel 编译器是 intel 公司专门为其处理器量身打造的编译器，对于 intel 处理器具有非常高的性能。本教程使用的机器就是 intel 芯片的，因此将会大量使用 intel 编译器以获取最高性能。如果读者使用 AMD 处理器，便可以略去安装。

这个编译器可以在 intel 主页申请非商业版，利用邮箱注册即可，可以下载到 intel 的 Fortran 和 C 编译器 11.1 版，注意申请 64 位版，因为我们选用 AMD64 机器：

l_cproc_p_11.1.069_intel64.tgz

l_cprof_p_11.1.069_intel64.tgz

intel 编译器是有最新版的，但是可能不稳定，这个 11.1 是经过作者测试的，至少在我的集群上没有问题。

intel 编译器本身安装非常简单，但是它有几个依赖的库比较讨厌：第一，尽管我们确实安装 64 位编译器，但是必须要装全套的 32 位的库才行；第二，现在的 Linux 上的 libstdc++.so 都是 6 版本，但是 intel 编译器需要 5 版本；第三，要装 intel 编译器必须先装 GNU 编译器。

这些问题都很容易解决。GNU 编译器已经装好，首先解决 libstdc++的问题。

1. 在 http://packages.debian.org/lenny/libstdc++5 上下载 amd64 架构的 libstdc++.so.5 的库，这个文件为 libstdc++5_3.3.6-18_amd64.deb。

2. 下载后，在全部节点上执行：

```
# dpkg –I libstdc++5_3.3.6-18_amd64.deb
```

这样 libstdc++.so.5 就有了，现在安装 32 位库，全部节点执行一个命令：

```
# aptitude install ia32-lib
```

好了，现在正式安装 intel 编译器，以 C 编译器为例，在控制节点上进行。

3. 解压，进入文件夹，安装：

```
# tar –xvzf l_cproc_p_11.1.069_intel64.tgz
# cd l_cproc_p_11.1.069_intel64/
# ./install.sh
```

下面的工作就是一路默认安装，注意中间输入 intel 公司的授权号。安装完

成后，在/opt 下会出现 intel 文件夹。用同样的方法安装 Fortran 编译器即可。

此时，intel 的命令还不在命令路径中，我们解决它，在所有节点上打开/etc/profile，将 PATH 改为：

```
if [ "`id -u`" -eq 0 ]; then
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/maui/bin:/usr/local/maui/sbin:/opt/intel/Compiler/11.1/069/bin/intel64:/usr/local/mpich2/bin:/usr/local/mpich2/sbin "
else
PATH="/usr/local/bin:/usr/bin:/bin:/usr/games:/usr/local/maui/bin:/opt/intel/Compiler/11.1/069/bin/intel64:/usr/local/mpich2/bin"
fi
```

并且最后一行添加上 intel 编译器的库路径：

```
export LD_LIBRARY_PATH=/opt/intel/Compiler/11.1/069/lib/intel64:/opt/intel/Compiler/11.1/069/mkl/lib/em64t
```

这样，intel 编译器就可以使用了！如果读者选取了不同的安装路径，注意按照正确的路径修改。注意：上面的 PATH 中还有一些新加的 mpich2 之类，将在下一节用到。

§2.4 并行计算环境的配置

既然要并行计算，就得有并行环境。目前比较流行的并行计算环境被称为 MPI（Message Passing Interface）即消息传递接口，是一种并行计算的标准或规范，迄今为止，所有的并行计算机制造商都提供对 MPI 的支持，可以在网上免费得到 MPI 在不同并行计算机上的实现，一个正确的 MPI 程序可以不加修改地在所有的并行机上运行。标准 MPI 虽然很庞大，但是它的最终目的是服务于进程间通信这一目标。

MPI 的实现有多种，这里选用 MPICH2。它是免费软件，可以从 http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads 上得到。这里要下载源码，得到 mpich2-1.3.1p1.tar.tar。

注意：不要选择 MPICH1，它有很多缺点！

可以开始编译 MPICH2。这个过程只需要在控制节点完成。

1. 解压：

```
# tar –xvf mpich2-1.3.1p1.tar.tar
# cd mpich2-1.3.1p1
```

2. 设置编译选项。我们要把 mpich2 装到/usr/local 下，可以实现全部集群共享，不妨建立一个 mpich2 的文件夹，装 MPICH2 并且配置 configure：

```
# mkdir /usr/local/mpich2
#./configure --prefix=/usr/local/mpich2 --enable-cxx --enable-f90 --enable-threads =multiple
```

这几项的含义：--prefix 设置安装的路径；--enable-cxx 和--enable-f90 使 MPICH2 支持 C，C++和 FORTRAN 语言；--enable-threads=multiple 表示支持线程，这样可以更好的利用共享内存，前提是机器上必须有多个 CPU。

3. 设置编译器。如果使用 GNU 编译器：

```
# export CC=gcc
```

如果使用 intel 编译器：

```
# export CC=icc
```

如果读者的机器是 intel CPU，强烈建议使用 intel 编译器。

4. 编译和安装：执行：

```
# make
# make install
```

完成后，在/usr/local/mpich2/bin 中就会出现很多命令，如 mpicc，mpif90 等等，都是并行的 C、Fortran 编译器。这些命令应该加入 PATH，一般来说，/bin 的命令对普通用户和 root 都需要加入，而/sbin 则最好只加入到 root 的 PATH。这就是上一节中 PATH 加入的 mpich2 之类的含义，当然，如果读者的命令路径不一样，需要转换。

要运行并行计算，还需要进行一些配置，告诉 MPICH2 哪些机器能进行并行计算。对我们来说，当然就是计算节点 node021 ~ node029，配置方法：在控制节点 compuchem 的/root 下，建立一个文件：mpd.hosts，里面写入：

```
node021
node022
node023
node024
node025
node026
```

node027
node028
node029

到此为止，我们的 MPICH2 已经配置完成了。本来我们现在应该测试一下 MPICH2，但是由于我们建立的是集群，我们希望并行的任务更加的有条理，因此，我们暂时先把 MPICH2 放一放，现在的任务是建立集群的作业管理系统。

§2.5 作业系统的建立

计算机集群的任务调度是一项非常重要的工作。要想充分发挥集群的硬件和软件威力，必须将资源合理的分配给所运行的作业，使得整个集群的使用效率达到最大。因此，必须选用一个自动化的作业管理系统来进行作业的调度。

举例来说，如果 coolrainbow 想“在 node023 上用 8 个 CPU 拿 Gaussian 算我这个 C60.gjf”的话，只需要在控制节点上发出这个指令就行了；如果 bingtears 想“用 node021 ~ node025 并行运行 NAMD”的话，也只需要在控制节点上发出这个指令就行了；如果 coolrainbow 已经占用了 node023，那么 bingtears 的任务显然不能运行了，那么作业管理系统就会对任务“排队”，等待 coolrainbow 的 node023 使用完毕（也许是三天以后），自动将 bingtears 的任务提交，运行 NAMD。每个任务运行完后，作业系统会自动报告该任务占用的资源数量和运行时间。这个系统还应具有权限分配功能。比如，某组在你的集群中花钱独占了 node025 的一个月的使用权，那么可以对作业系统进行设置，使没有花钱的用户不能在 node025 上发作业……

作业管理软件有很多，如 OpenPBS 等等。这里，我们选用一套作者认为最稳定的组合：Torque+Maui+mpiexec。这三个都是免费软件。

Torque：最主要的作业分配工具。

Maui：Torque 的辅助工具。前面提到的限制权限的功能，单独用 Torque 无法实现，必须加上这个软件。

Mpiexec：Torque 与 MPICH2 连接的工具。它有什么用呢？如果我们直接使用 Torque，比如，分配 node021 和 node022 给 MPICH2 让它并行运行，运行的结果，可能 MPICH2 根本就不是使用的 node021 和 node022，而是随机使用的节点。mpiexec 的功能，就是使用 Torque 分配的节点来执行 MPICH2 的任务，使得两者之间无缝结合。

注意：MPICH2 的可执行文件中也有一个 mpiexec（位于/usr/local/mpich2/bin 下），这个与我们上面提到的 mpiexec 名称确实完全相同，但是两者完全不是一个东西！千万不要混淆！

下面开始构造这个系统。

§2.5.1 Torque 的安装

到 2011 年 5 月时，Torque 的最新版本为 3.0，作者选用较为稳定的 2.4.6。可以从 http://clusterresources.com/downloads/torque 下载其安装文件 torque-2.4.6.tar.gz。

Torque 的安装十分简单。在控制节点上依次执行下列命令即可：

```
# tar -xvzf torque-2.4.6.tar.gz
# cd torque-2.4.6
# ./configure --with-scp CC=icc
# make
# make install
```

注意：如果不使用 intel 编译器，上面第三行的命令把 CC=icc 去掉。

这就完成了控制节点的安装。此时 compuchem 已经成为了一个 PBS 服务器。现在执行：

```
# make packages
```

这时会生成一些.sh 文件，其中两个重要文件：torque-package-mom- linux-i686.sh 和 torque-package-client-linux-i686.sh 将用来安装计算节点。在计算节点上，执行：

```
# ./torque-package-mom-linux-i686.sh --install
# ./torque-package-client-linux-i686.sh --install
```

至此，Torque 安装结束。

§2.5.2 Maui 的安装

现在安装 Maui。这个软件需要在 http://clusterresources.com 注册才能下载，这里选用稳定的版本 3.3，下载后得到文件 maui-3.3.tar.gz，它的安装也很简单，只要在控制节点执行即可：

```
# tar -xvzf maui-3.3.tar.gz
# cd maui-3.3
```

```
# ./configure CC=icc
# make
# make install
```

注意：如果不使用 intel 编译器，上面第三行的命令把 CC=icc 去掉。

§2.5.3 Torque 和 Maui 的配置

Torque 和 Maui 需要进行许多配置才能正常工作。在控制节点：

1. Torque 必须配置一位普通用户的管理员，这就需要在刚才的安装目录即 torque-2.4.6 中，执行：

```
# ./torque-setup coolrainbow
```

这就选择了 coolrainbow 作为 Torque 的普通管理员。注意：如果不设普通用户的管理员就无法运行程序。

2. 进入/var/spool/torque/server_priv，建立名为 nodes 的文件，写入

```
node021 np=8
node022 np=8
node023 np=8
node024 np=8
node025 np=8
node026 np=8
node027 np=8
node028 np=8
node029 np=8
```

这就是 Torque 计算节点的列表，np 代表 CPU 个数。

3. 重启 pbs 服务。方法：

```
# ps –A | grep pbs          # 得到 pbs 的进程号，如 2798
# kill 2798
# pbs_server
# pbs_sched
```

4. 使 PBS 服务器开机自动启动。进入/etc/init.d，建立一个文件如 pbsserver，写入：

```
/usr/local/sbin/pbs_server
/usr/local/maui/sbin/maui
```

第一行是启动 PBS 服务器；第二行，本来应该是/usr/local/sbin/pbs_sched，这个是 Torque 的任务调度系统，但是我们已经装了 Maui，就不需要它，而是启动更高级的 Maui 任务调度系统。

这个文件建立好后，执行：

```
# chmod 755 pbsserver
# update-rc.d pbsserver start 30 2 3 4 5 . stop 30 1 6 0 .
```

最后的命令调用了 update-rc.d，这样以后重启时，PBS 服务就会自动启动，而不用手动启动。

对 update-rc.d 不熟悉的读者可以搜索下它的用法。

下面进行计算节点的配置，以下操作在计算节点上进行。

5. 进入/var/spool/torque/mom_priv，建立名为 config 的文件，写入：

```
$pbsserver    compuchem
$logevent     255
$usecp *:/home /home
```

6. 设置自动启动。进入/etc/init.d，建立一个文件如 pbsmom，写入：

```
/usr/local/sbin/pbs_mom
```

然后，运行：

```
# chmod 755 pbsmom
# update-rc.d pbsmom start 30 2 3 4 5 . stop 30 1 6 0 .
```

这样以后重启时，PBS 客户端就会自动启动。

此时配置已经基本完成。

激动人心的时刻到了！当所有的 PBS 服务器和客户端都启动好后，可以执行命令 pbsnodes（这个命令显示所有的节点的情况）查看节点的情况；

```
# pbsnodes
…
node025
state = free
np = 8
ntype = cluster
status = opsys=linux,uname=Linux node025 2.6.18-6-686 #1 SMP Sun Feb 10 22:11:31 UTC 2008 i686, sessions=2839, nsessions=1, nusers=1, idletime=197844, totmem=4726576kb, availmem=4685196kb, physmem=2075892kb, ncpus=8, loadave=0.00, netload=1043928615, state=free, jobs=, varattr=, rectime=1238950166
node026
state = free
np = 8
ntype = cluster
status = opsys=linux,uname=Linux node025 2.6.18-6-686 #1 SMP Sun Feb 10 22:11:31 UTC 2008 i686, sessions=2839, nsessions=1, nusers=1, idletime=197844, totmem=4726576kb, availmem=4685196kb, physmem=2075892kb, ncpus=8, loadave=0.00, netload=1043928615, state=free, jobs=, varattr=, rectime=1238950166
```

…

也就是说，作业管理系统已经找到了我们的计算节点了！现在，整个集群的作业管理系统已经接近完成！现在就差 mpiexec 了！

§2.5.4 mpiexec 的安装

好了，现在安装 mpiexec。它的最新版本为 0.84，可在 http://www.osc.edu/~djohnson/mpiexec/index.php 下载到 mpiexec-0.84.gz。在上面安装 Torque 和 Maui 时，没有指定安装路径，对于它，我们有必要指定一个，比如下面的路径：

```
# mkdir /usr/local/mpitorque
```

下面安装。在控制节点上执行：

```
# tar –xvzf mpiexec-0.84.gz
# cd mpiexec-0.84
# ./configuration --prefix=/usr/local/mpitorque --with-pbs=/usr/local --with-default-comm=mpich-p4
```

这一行，--prefix 配置安装路径，--with-pbs 指定 Torque 的链接库、头文件的上级路径，--with-default-comm 指定 MPICH2 的类型。

然后编译安装即可：

```
# make
# make install
```

此时，mpiexec 安装到了/usr/local/mpitorque 中。以后所有的并行任务将由它执行。

OK！集群到此就全部安装完毕！现在可以试运行一个任务了！

§2.6 第一次运行集群

切换为普通用户比如 coolrainbow。当然，我们的集群上还没有没有任何软件，但是有编译器，我们将用编译器编一个并行程序，并通过作业管理系统运行。

比如，在家目录下，用 vi 编辑一个并行的 Fortran 程序，叫 hellocluster.f95，见脚本 2-6-1。

编译这个程序（注意全部工作要以普通用户的身份执行）：

```
$ mpif90 hellocluster.f90 -o hellocluster
```

脚本 2-6-1 并行的 Fortran 程序

```
! hellocluster.f95

program main
Include 'mpif.h'
implicit none
    character*(MPI_MAX_PROCESSOR_NAME) processor_name
    integer id, nprocs, namelen, rc,ierr
    call MPI_INIT( ierr )
    call MPI_COMM_RANK( MPI_COMM_WORLD, id, ierr )
    call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
    call MPI_GET_PROCESSOR_NAME(processor_name, namelen, ierr)
    write(*,1100) id,nprocs,processor_name
    call MPI_FINALIZE(rc)
1100 FORMAT('Hello Cluster! Process ',I2,' of ',I1,' on ', 20A)
end
```

现在得到了可执行文件 hellocluster。我们调用作业管理系统来执行它。

建立一个叫做 submit.pbs 的文件（也可以是你喜欢的任何名字），写入如下内容：

脚本 2-6-2 作业提交脚本

```
#!/bin/sh
#PBS -l nodes=node023:ppn=8+node027:ppn=8
#PBS -q batch
#PBS -j oe
#PBS -N first_task
cd /home/coolrainbow
/usr/local/mpitorque/bin/mpiexec    ./hellocluster > result
```

这个脚本，是提交作业的标准脚本，所有的用户都应该通过 Torque 使用提交作业的脚本执行任务，这是一切集群（从我们这个学术用小集群到上千台计算机的超级集群）执行任务的标准。

当然，商业或更高级的集群是不使用 Torque 这个免费的作业管理系统的，它们往往使用一些商业软件（价格常常相当昂贵），但是都是要通过某些脚本提交作业的。

在脚本 2-6-2 中，以#PBS 开头的都是与 Torque 有关的选项，其他的命令（如 6,7 行）则是要执行的任务。这在第四章会详细介绍 。这里，只介绍几个：第 2 行，我们了**分配** node023 的 8 个 CPU 和 node027 的 8 个 CPU 给我们的任务；第 5 行，是给我们的任务随便起的一个名字；第 7 行，调用 mpiexec 并行执行

hellocluster，并将输出重定向到一个叫 result 的文件里。

mpiexec 会自动根据 Torque 分配的资源并行执行 hellocluser。如果这里使用 MPICH2 自带的那个 mpiexec 的话，不仅步骤繁琐（需要启动 mpd 守护者进程之类），而且不能保证我们并行的任务是在 node023 和 node027 上执行的。

有这个脚本后，提交作业：

```
$ qsub submit.pbs
```

不久，会生成一个叫 first_task.o0 的文件，注意与我们的任务名相同，当然这个文件里面是空的。还生成一个 result 文件，里面应该是我们的结果，打开看看：

```
Hello Cluster! Process 5 of 16 on node023
Hello Cluster! Process 12 of 16 on node023
Hello Cluster! Process 0 of 16 on node027
……
Hello Cluster! Process 1 of 16 on node027
```

没错，两台机器 16 个 CPU 并行执行了 16 个进程（process）！我们的集群成功了！

到此，我们的集群就全部组装完毕了！相信读者可以喘口气了，休息一下，哈哈！

上面的作业提交太简单了，不能满足要求。我们将在第四章讨论更有用的 Torque 的操作。下面，我们先装几个计算化学软件吧。

第三章 集群的计算化学软件配置

计算化学的软件，据说现在已经超过 20 种。这里不可能全部介绍安装的方法。但是考虑到集群的需要，读者如果没有什么 Linux 基础，面对计算化学软件的源码常常只能望而却步，因此这里作者挑选几种以“难编译”出名的软件介绍，相信读者即使没有编译软件的基础，在读完本章后，对大多数计算化学的软件的编译就没有问题了。

这里的软件安装，如无特殊声明，只要在控制节点上操作即可。

§3.1 Gaussian03 安装与配置

§3.1.1 Gaussian03 的安装

Gaussian03 是最著名的计算化学软件，通过商业途径可以购买到这个软件。这里以其光盘镜像 G03.ISO 来安装。同时，Gaussian03 的版权控制非常严格，只允许一定的用户使用，这要求管理员必须在系统中设定一个专门的组来运行 Gaussian03，不妨就叫 gaussian 组：

```
# groupadd gaussian
```

然后将 Gaussian03 的光盘镜像加载：

```
# mount –t iso9660 –o loop /root/G03.ISO /mnt
```

现在开始安装：

1. Gaussian03 必须用 C shell 安装：

```
# csh
```

如果没有 C shell，安装：

```
# aptitude install csh
```

2. 设定环境变量：

```
# setenv mntpnt "/mnt”
# seten g03root “/usr/local/chemsoft”
```

这两个环境变量的含义：mntpnt 表示光盘镜像挂载的路径；g03root 表示 g03 目录安装的路径。注意，这里在/usr/local 里建立了一个 chemsoft 的文件夹，以后所有的计算化学软件都装到 chemsoft 里。

3. 进入安装目录并解压安装文件：

```
# cd $g03root
# cat $mntpnt/tar/*.taz | zcat | tar xvf –
```

4. 现在，将会生成一个 g03 目录，将它的组改为 gaussian：

```
# chgrp –R gaussian g03
# cd g03
```

5. 执行安装脚本：

```
# ./bsd/install
```

6. 将 g03 目录及所有文件的权限改为 750：

```
# chmod –R 750 g03
```

现在只要把需要运行 Gaussian03 的用户加入 gaussian 组，用户就可以运行 gaussian 了。如修改/etc/group：

```
gaussian:x:1001:coolrainbow
```

这样 coolrainbow 就具有了运行 Gaussian 的权限。注意要重新 make 一下 NIS 数据库。

§3.1.2 Gaussian03 的单机配置

每次运行 Gaussian03，还需要设置一些环境变量。这里不采用通常的方法即在用户的.bashrc 中声明这些变量（作者的观点是既然有了集群，就把与具体任务有关的东西全部限制在作业脚本中，做到“**影响极小化**”），而是建立相应的 PBS 脚本，在提交作业前做这些处理。所设计的脚本可见脚本 3-1：

脚本 3-1-1 Gaussian03 的作业提交脚本

```
#!/bin/sh
#PBS -l nodes=node023:ppn=8
#PBS -q parallel
#PBS -j oe
#PBS -N UF4
export g03root="/usr/local/chemsoft"
mkdir /tmp/${PBS_JOBID}
export GAUSS_SCRDIR=/tmp/${PBS_JOBID}
source $g03root/g03/bsd/g03.profile
cd /home/coolrainbow
g03 < UF4.gjf > UF4.out
rm -rf /tmp/${PBS_JOBID}
```

解释下这个脚本含义。以#PBS 开头的行是 Torque 的设置内容，以后介绍；

6 行导出 Gaussian03 所在的路径变量；7 行建立计算所需的缓存文件目录（最好使用本地目录），位于/tmp，其名称为该作业的编号（PBS_JOBID 是 Torque 内部设定的变量，后面会介绍）；8 行导出缓存文件目录；9 行执行 Gaussian03 的一个脚本；10 行进入 gjf 文件所在的目录，根据情况修改；11 行正式运行 Gaussian03，UF4.gjf 就是要运行的文件，UF4.out 是输出文件；12 行在任务结束后删除缓存文件目录。

这样，这个脚本可以给任何属于 gaussian 组的普通用户运行 Gaussian03。只要修改相应的路径和输入输出文件名后，运行（假设脚本名称为 g03_submit.pbs）：

```
$ qsub g03_submit.pbs
```

即可运行。至此单机的 Gaussian03 就配置完毕。

如果安装 Gaussian09，完全可以参照上述步骤运行，Gaussian09 的环境变量照样写入作业提交脚本中（这也看出了不写入.bashrc 的优点，至少防止 GAUSS_SCRDIR 冲突）。

§3.2 NAMD 安装与配置

NAMD 是并行效率极高的分子动力学软件，特别适合超大规模体系的处理，常常用来进行生物大分子研究和大规模的分子动力学模拟。

NAMD 的使用比较简单，直接运行可执行文件即可，但是 NAMD 的安装非常复杂，配置文件繁多，有一点儿错误就无法安装，因此这里将安装过程详细介绍。

需要说明的是：对于不同的软件和硬件，安装的配置是有一定差别的，但是只要掌握了本节中例子，其差别就不言自明。

到 2011 年 5 月 NAMD 的最新版本为 2.8，但这是测试版，不稳定（如已经发现 2.8b1 存在 bug），这里我们会讲解 2.7 的编译方法。可以从 http://www.ks.uiuc.edu 上下载到其源码文件 NAMD_2.7_Source.tar.gz。

在本节中，我们主要讲解 NAMD 多机并行版本的编译方法，相信这个版本掌握后，再编译别的版本（如单机多核版）时基本不会遇到太大困难。另外，最新的 GPU 加速版将在§5.2 节单独介绍。

如果读者掌握了这个软件的编译方法，基本任何 Linux 软件的编译都不在话下。作者正是当初在初学 Linux 摸索 NAMD 的编译时，学到了大量的知识，现在基本可以没有任何困难的编译各种 Linux 软件。

§3.2.1 组件安装

首先要安装一些组件。NAMD 需要两个组件支持：TCL 语言的模块 tcl8.4 和快速 Fourier 变换模块 fftw。要安装这两个组件，只要在所有节点上执行（注意：这些组件会自动安装到/usr 下的某些目录，不在共享的目录下，所以要全部安装）：

```
# aptitude install fftw3 fftw3-dev sfftw-dev sfftw2 tcl8.4 tcl8.4-dev
```

如果读者有自己的高效的 fftw 库，也可以自行编译安装，过程很简单，就不重复了。至于 TCL，使用 aptitude 安装的比较可靠。

§3.2.2 charm++安装

编译 NAMD 只要在控制节点进行即可。首先要编译 charm++。

1. 解压并进入相应目录：

```
# tar –xvzf NAMD_2.7_Source.tar.gz
# cd NAMD_2.7_Source
# tar –xvf charm-6.1.3.tar
# cd charm-6.1.3
```

2. 配置。要安装的 NAMD 必须支持并行，因此要给 charm++指明并行并行库，首先给出 mpi-linux 告诉编译器要编译并行，然后要通过--bashdir 参数给出 MPI 库的位置，因为前面 MPICH 安装到了 usr/local/mpich2 下，MPI 库这位于这个目录下的 lib 中，故给出 MPICH 的安装路径（不需要加/lib！）就可以。同时，给出所用的编译器和优化级别。charm++表示对 charm++只做基本安装。

下面是一个非常需要注意的问题。如果读者使用 GNU 编译器编译了 MPICH2，在 build 后面加上“cc”选项，表示用 GNU C 编译器编译 charm++：

```
# ./build charm++ mpi-linux --basedir=/usr/local/mpich2 cc -O
```

如果读者已经使用 intel 编译器编译了 MPICH2，并且打算使用 intel 编译器编译 charm++的话，**一定不要在命令行中指定任何编译器！**即正确的方法：

```
# ./build charm++ mpi-linux --basedir=/usr/local/mpich2 -O
```

上面的命令会自动用 intel 编译器实现 charm++。

如果读者使用下面的命令:

```
# ./build charm++ mpi-linux --basedir=/usr/local/mpich2 icc -O
```

就会遇到 MPICH2 函数符号不能识别的情形，即下面的错误：

```
`CmiNotifyIdle':
machine.c:(.text+0x202): undefined reference to `MPI_Recv'
machine.c:(.text+0x232): undefined reference to `MPI_Get_count'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiAsyncSendFn_':
machine.c:(.text+0xe82): undefined reference to `MPI_Isend'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiAsyncBroadcastAllFn':
machine.c:(.text+0x11c2): undefined reference to `MPI_Test'
> > ./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiAsyncBroadcastFn':
machine.c:(.text+0x1622): undefined reference to `MPI_Test'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiReleaseSentMessages':
machine.c:(.text+0x2702): undefined reference to `MPI_Test'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function `PumpMsgs':
machine.c:(.text+0x28f2): undefined reference to `MPI_Iprobe'
machine.c:(.text+0x2942): undefined reference to `MPI_Get_count'
machine.c:(.text+0x2992): undefined reference to `MPI_Recv'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiAsyncMsgSent':
machine.c:(.text+0x2e52): undefined reference to `MPI_Test'
./../../bin/../lib/libconv-cplus-y.a(machine.o): In function
`CmiBarrierZero':
machine.c:(.text+0x3182): undefined reference to `MPI_Recv'
machine.c:(.text+0x3232): undefined reference to `MPI_Send'
```

../../../bin/../lib/libconv-cplus-y.a(machine.o): In function

CmiCpuTimer':

machine.c:(.text+0x32d2): undefined reference to `MPI_Wtime'

./../../bin/../lib/libconv-cplus-y.a(machine.o): In function

`CmiWallTimer':

machine.c:(.text+0x3352): undefined reference to `MPI_Wtime'

./../../bin/../lib/libconv-cplus-y.a(machine.o): In function `CmiTimer':

machine.c:(.text+0x33d2): undefined reference to `MPI_Wtime'

./../../bin/../lib/libconv-cplus-y.a(machine.o): In function

因此不能在命令行中指定 icc。这个错误作者曾经遇到过，是 charm++的编写者给出的这个解决方法。

完成后，如果看到如下字样：

---

charm++ built successfully.
Next, try out a sample program like tests/charm++/simplearrayhello

就表明编译成功！

§3.2.3 NAMD 编译

编译 NAMD 修改需要很多参数，因为其默认的参数很多无法符合集群的要求。

1. 首先要保证机器上必须有 C shell；

2. 指向 charm++。修改 NAMD_2.7_Source/Make.charm，将 CHARMBASE 一行改为：

CHARMBASE = .rootdir/charm-6.1.3

这里.rootdir 是一个指向安装路径的符号链接，在随后的安装中会自动生成。

3. 然后进入 NAMD_2.7_Source/arch，由于我们要编译 MPI 版本，故根据机器的硬件架构选择 Linux-<architecture of the machine>-MPI.arch 文件，如本教程中使用 x86_64 架构，故打开 Linux-x86_64-MPI-icc.arch，将 CHARMACH 一行改为:

CHARMACH = mpi-linux-x86_64-smp

如果使用 gcc 编译，则打开 Linux-x86_64-MPI-g++.arch 做相应修改。

4. 给出 tcl 和 fftw 的库的路径。同样根据机器选择适当的文件，如本组机器，打开 NAMD_2.7_Source/arch/Linux-x 86_64.tcl，将前几行改为如下内容：

```
TCLDIR=/usr
TCLINCL=-I$(TCLDIR)/include -I$(HOME)/tcl/include -I/usr/include/tcl8.4/
TCLLIB=-L$(TCLDIR)/lib -L$(HOME)/tcl/lib -ltcl8.4 -ldl
```

第一行给出了 tcl 库所在的目录，第二行最后的-I/usr/include/tcl8.4/同前；第三行则给出了 tcl 库的版本。

打开 NAMD_2.7_Source/arch/Linux-x86_64.fftw，修改第一行以给出 fftw 库所在的位置：

```
FFTWDIR = /usr
```

如果读者的 tcl 和 fftw 在别的位置，或者版本不同（比如读者使用了 tcl8.5 或者使用 intel MKL 的 fftw 库），则需要对上述选项做修改。这个可以由读者自己完成。

5. 编译 NAMD。配置并编译：

```
# ./config tcl fftw Linux-x86_64-MPI-icc
# cd Linux-x86_64-MPI-icc
# make
```

至此，NAMD 终于编译完毕。

7. 移动 NAMD。在/usr/local/chemsoft 中建立 namd2.7/bin 文件夹，将生成的可执行文件 namd2，psfgen，charmrun（位于 Linux-x86_64-MPI-icc 中）等移动到该文件夹里，这样 namd 就可以为整个集群共享，现在就可以实现并行计算了。

§3.2.4 NAMD 运行

通过 PBS 脚本运行 NAMD 是很容易的事。对于并行运行的 NAMD， 可以像脚本 2-6-1 那样运行这个程序。

下面给出一个脚本的范例：

脚本 3-2-1 **NAMD** 的作业提交脚本

```
#!/bin/sh
#PBS -l nodes=node023:ppn=8+node027:ppn=8
#PBS -q batch
#PBS -j oe
#PBS -N namd
cd /home/coolrainbow
/usr/local/mpitorque/bin/mpiexec /usr/local/chemsoft/namd2.7/namd2 ubq.conf > ubq.log
```

这个脚本比较好理解。

§3.3 Autodock 的安装与配置

Autodock 是一个非常著名的分子对接程序。可以从其网站 http://autodock.scripps.edu/上下载到其最新版本 4.2 的压缩包 autodocksuite-4.2-src.tar.tar。

Autodock 的安装十分容易。在主节点上建立文件夹/usr/local/chemsoft/autocock4.2 以便于安装软件的执行文件，然后依次执行：

```
# tar -xvf autodocksuite-4.2-src.tar.tar
# cd autodocksuite-4.2
# cd autocock
# ./configure --prefix=/usr/local/chemsoft/autocock4.2
# make
# make install
# cd ../autogrid
# ./configure --prefix=/usr/local/chemsoft/autocock4.2
# make
# make install
```

安装完毕。运行脚本也非常容易：

脚本 3-3-1 **Autodock** 的作业提交脚本

```
#!/bin/sh
#PBS -l nodes=1:ppn=8
#PBS -q parallel
#PBS -j oe
#PBS -N heme
cd /home/coolrainbown
/usr/local/chemsoft/autodock4.2/bin/autodock4 -p heme.dpf -l heme.dlg
```

这个编译步骤非常典型，大部分 Linux 软件的编译都是

configure——make——make install

这个流程。configure 中的一些选项在不同 Linux 软件中意义都是一致的。比如，“--prefix”都是设定安装路径的，“--help”都是显示帮助信息的，“CC”设定 C 编译器，“CXX”设定 C++编译器，“COPT”设定 C 编译器的选项，等等。建议读者，拿到一个新的 Linux 软件时，首先寻找含有 README，INSTALL，GUIDE 字眼的文件，里面通常就有编译的步骤，然后./configure --help 查看一下需要注意的事项，然后 configure，make，make install。通常，大部分 Linux 软件都可以顺利编译了。

我们关于软件编译的东西就讲到这。有些软件，如 NWCHEM，GAMESS 等，它们的编译相当复杂（特别是生成高效的源码），这里就不重复详细的步骤了。网上有很多这方面的文章，比如，作者的博客（广告 ：P）。

§3.4 MKL 编程环境的构建

前面已经介绍了很多计算化学软件的使用，如果还需要安装其他的软件可以依照前面的经验通过软件文档和自己摸索完成。

Linux操作系统是一个非常适合进行开发工作的系统，当需要自己编程完成一些工作时，可以利用系统自带的make，vi等完成，这些内容可以参考相关的技术文档，本文不介绍。由于涉及到了intel编译器，这里将MKL程序的链接方法介绍以下。

作者曾经在集群上开发过C，Fortran，Python和PHP+MySQL的程序（大多3000行以下），深感结合make，vi，grep，awk，shell脚本等精巧而强大的工具，开发程序原来可以变得那么简单！这些操作在Windows下是无法想象的。

经验表明，第一次做MKL程序编译时，大多数人都会走很多弯路，编译几个小时也不成功。其实这个原因很简单，就是：**懒于阅读技术文档**！事实上，作者当时就是这样，看到3000多页的MKL手册头都大了，于是在编译时就“跟着感觉走”，弄得找不到库或者函数错误。其实，花一个小时读下文档，绝对比自己瞎折腾要强的多，这里把编译MKL的一些经验与大家分享下，作为快速入门。真正深入的话，还是那句：读文档！这里以Lapack库的链接为例。

1. MKL的环境变量

安装好MKL后，需要设置一些环境变量，这样才能找到所需要的库，这可以通过/opt/intel/Compiler/11.1/064/mkl/lib/tools/enviroments/mklvars {your-

architecutre} .{sh|csh}实现。如果需要的话，可以加入到/etc/profile或你的.bashrc中。在程序编译或运行时，如果发生can not find libXXX之类，记得导出相应的LD_LIBRARY_PATH。如：

can not find libmkl_intel_thread: cannot open shared object file...

什么？不知道什么是LD_LIBRARY_PATH？Google it！

如果你的这个库位于/opt/intel/Compiler/11.1/064/mkl/lib/em64t，那么执行命令即可：

$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/Compiler/11.1/064/mkl/lib/em64t

2. 函数选择：Lapack

MKL提供了3000多页的技术文档，对每一个函数进行了详细的介绍。这里仅对Lapack系列做一些介绍。

初学Lapack的读者可能会问：我怎么知道哪个函数是做向量点积的，哪个函数是做奇异值分解的。问的最多的是：有没有什么书可以参考的？答案：至少作者没见过哪个书像讲MFC或PHP那样详细的把Lapack的每一个函数讲解的书。那么唯一的途径就是阅读技术文档。不要头疼，那个文档没有中文版的。好好学英文吧。

Lapack中的函数的名称都是**XYYZZZ**形式（BLAS类似）的，

X：精度。如s表示单精度（对应于Fortran中的real(kink = 4)，C中的float）等。

YY：对应的矩阵类型和存储方式。如：

ge：为一般的矩阵，存储方式full。

所谓full就是说，矩阵A的元素a_{ij}以数组a(i, j)或a[i][j]的形式存储于计算机中。这是最简单的方法，但是对稀疏矩阵或元素有一定相关性的矩阵效率较低

sy：对称矩阵，以full方式存储

sp：对称矩阵，以pack方式存储

pack方式对symmetric matrix采用了一维数组优化存储的方法，可以减小内存的使用。有两种存储方法，在Lapack程序中分别用'U' 'L'表示，对于“U”，简单的说：a_{ij}位于a(i+j*(j-1/2)当i <j时。“V”类似。

还有其他的类型，请参考技术文档。

ZZZ：任务类型。就是你要干什么。如dot表示点积，trd表示三对角化等等

知道了这个命名规则后，找函数就方便多了。

3. 函数参数：Lapack

Lapack中的每一个函数都有很长的参数，常常把初学者吓倒。并不是说对矩阵A，B求乘法，只要简单的输入A，B就可以了。还有一些辅助空间，任务类型之类的，都要输入，如矩阵乘法：

```
call dgemm('n','n',N,M,K,a,x,N,y,K,b,z,N)
```

执行矩阵乘法**z=xy**，但还有一些character*1的'n','n'以及integer的N,K等等，这些含义一定要仔细弄清，不然可能会使任务执行错误！

4. 编译连接

这是最头痛的问题。MKL的连接选项比较长，最好使用make工具，作者常用的一个模板如下：

脚本 3-4-4 MKL 程序编译 Makefile 模板

```
mkllib=/opt/intel/Compiler/11.1/064/mkl/lib/em64t
mklinc=/opt/intel/Compiler/11.1/064/mkl/include
mkllnk= -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lmkl_lapack95_lp64 -liomp5 -lpthread
foo: foo.o
        ifort -o foo foo.o -I$(mklinc) -L$(mkllib) $(mkllnk)
```

在连接MKL程序时，要在每一个“LAYER”中选择一个库，

1. interface layer：提供程序接口，即libmkl_intel{lp64}，其中lp64是对em64t或ia64版本。

2. threading layer：线程接口。一般都是libmkl_intel_thread。注意，有了它后，一定要加上iomp5和pthread库，不然会出现连接错误。

3. computational layer，计算接口，即libmkl_core和你所需要的，如lapack的libmkl_lapack95_lp64或者线性方程组的libmkl_solver_lp64等等。

4. RTL接口，即pthread等等。

这样才可以连接成功。

看一个实例：

现在以一个实例完成。想算一个矩阵乘法，矩阵用双精度，一般存储模式，则选用dge???函数，查文档得知?gemm做矩阵乘法，于是选用dgemm，这个算矩阵乘法的程序代码见脚本3-4-2，编译用的Makefile见脚本3-4-3。

编译完成后，一个叫foo的可执行文件出现了。它将会调用MKL运行矩阵乘

法程序。

脚本 3-4-2 MKL 程序实例

```
! foo.f90
program main
      integer,parameter::N=3000
      integer,parameter::M=2000
      integer,parameter::K=4000
      real(kind=8) a,b
      real(kind=8),pointer::x(:,:)
      real(kind=8),pointer::y(:,:)
      real(kind=8),pointer::z(:,:)

      a=1.0
      b=0.0
      allocate(x(N,K))
      allocate(y(K,M))
      allocate(z(N,M))
      x=10.0
      y=20.0
      z=0.0
      write (*,*) "Performing matrix multiplication by MKL..."
      call dgemm('n','n',N,M,K,a,x,N,y,K,b,z,N)
      write (*,*) "DONE!"
      deallocate(x)
      deallocate(y)
      deallocate(z)
end
```

脚本 3-4-3 脚本 3-4-2 的 Makefile 模板

```
mkllib=/opt/intel/Compiler/11.1/064/mkl/lib/em64t
mklinc=/opt/intel/Compiler/11.1/064/mkl/include
mkllnk= -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lmkl_lapack95_lp64 -liomp5 -lpthread
foo: foo.o
        ifort -o foo foo.o -I$(mklinc) -L$(mkllib) $(mkllnk)
foo.o: foo.f90
        ifort foo.f90 -c -g
```

至此，整个计算化学平台已经搭建完毕，已经可以投入使用。下面将讨论一些与集群相关的管理技巧。

第四章 集群的管理技巧

§4.1 用户管理

§4.1.1 用户权限

读者建立的集群可能要给组里的所有人使用，也许是两三个人，也许是二三十个人。对于这种规格的集群，在建立用户时候，一定要有条理，对每个用户的权限都要把握适当，不然，不仅会使集群混乱，管理不便，而且有可能破坏集群，甚至带来法律问题。

例如，如果读者的组里有 VASP，则每个用户对 VASP 的权限要严格控制。如果用户不再 VASP 的使用者列表上而具有 VASP 的运行权限，则可能给读者的组里惹来麻烦。

下面讨论一下集群用户建立的经验。

1. 用户名的设置一定要有条理。在作者本组的集群中，除了 root 和一个普通管理帐户外，所有使用者一律以“userXX”命名（如 user01，user02 等）。当然，如果读者想使用别的命名原则也可以。

2. 设置好适当的组。有些软件对组的要求是必须的，如 Gaussian，必须要有一个组专门运行 Gaussian。另外有些组的设置与集群的权限有关。比如有的用户只允许发实验性的小作业（如最多连续运行 1 个 CPU 小时），而有的用户允许运行大型作业（如连续几天的大作业），可以分别设置两个组 trail 和 large，并对 Torque 进行设置，这样就可以对不同用户做精确的权限限制。

注意每个用户可以同时加入很多组。

§4.1.2 高效的管理

对于前面提到的 coolrainbow，它的各种通信都已经设计完备，这些东西的实现是通过其家目录下的文件夹.ssh 实现，并且，由于 coolrainbow 已经与所有的节点进行过通信（这只要在各个节点上 ssh 本机即可实现），它的.ssh/knownhost 里已经记载了全部信任主机的主机名，即通信时不会再询问

“Are you sure you want to continue connecting (yes/no)?”这样的问题。而且，由于所有的用户都将使用计算软件，故将 coolrainbow 的 PBS 脚本（如 g03_submit.pbs 等）给所有用户共享也是个不错的选择。因此，如果在建立新用户时能够将通信文件和脚本文件自动加到新用户的家目录下，就可以大大提高工作效率。

这十分容易实现。/etc/skel 文件夹是个系统文件夹，新用户建立时 skel 里的所有文件会自动拷贝到新用户的家目录下。因此，可将/home/coolrainbow 下的.ssh，*.pbs 全部拷贝到 skel 目录下，新用户建立后便可直接实现无密码 SSH 通信并且自动获得大量脚本，省去了许多配置。

当然，添加完新用户后，一定要在/var/yp 下 make 一下，以更新 NIS 数据库。同时还要设置其 group。这一系列工作可以通过一个 shell 脚本即脚本 4-1-1 实现：

脚本 4-1-1 集群添加用户的脚本

```
#!/bin/sh
username = $1
password = $2
useradd -m $username
echo $username:$password | chpasswd
cd /var/yp
make
```

假设这个脚本叫 useradd2cluster，要添加一个名为 user01、密码为 123456 的用户，只要执行：

```
# ./useradd2cluster user01 123456
```

当集群需要批量添加用户时，可以用循环的方式执行上述脚本。

删除用户非常简单，例如要删除 user01 及其家目录下的所有文件：

```
# usedel -r user01
# cd /var/yp
# make
```

§4.2 作业管理

Torque 的作业管理是以“队列”的形式实现的。一个队列是指具有某种权限的用户的集合。例如，可以设置一个 big32 的队列，凡是这个队列的用户可以最多同时用 32 个 CPU；而一个 trail 队列的用户只能用 1 个 CPU 最多算一个小时。

作业的管理分为管理员和普通用户管理两部分。

§4.2.1 管理员管理

这里的管理员即可以是 root，也可以是配置过程中的配置的普通用户管理员。管理员通常不提交作业，只用来管理。

1. 管理队列

集群建立后，用户要想通过 Torque 提交作业，必须以一定的队列的身份提交。Torque 在安装时已经创建了一个默认队列 batch，要创建一个新队列，需要使用 qmgr 命令行提交一系列命令，可以创建一个 shell 脚本来实现这个功能，即脚本 4-2-1。

这个脚本中，“qmgr -c”表示要调用 Torque 的 qmgr 命令进行工作，第四行用表示创建一个名为 queue_name 变量值的队列，剩下的每一行“set queue”分别代表设定队列的某一种属性。以后再添加新队列的时候，只要修改相应的队列的属性值即可。常见属性的含义见表 4-2-1。

2. 管理服务器端

Torque 安装完成后，服务器端的默认配置基本能满足要求。要修改可以采用命令格式为“qmgr -c “set server attribute = values””，重要的内容可以参见表 4-2-2。

3. prologue 和 epilogue 文件

这两个文件夹位于/var/spool/torque/mom_priv 内，它们都是一般的 shell 脚本，只能由管理员修改，它们分别在 PBS 脚本之前和之后执行，在本集群的设计中用来添加一些对作业的提示，这些东西输入到作业的结束文件中如 gaussian.o112 中。

这两个文件在执行时，系统将会自动为其添加一写命令行参数，这些命令行参数的意义可以参见表 4-2-3。

下面给出本集群中使用的 prologue 和 epilogue 文件。它要在所有的客户端的/var/spool/torque/mom_priv 中添加，权限为 755，可以参见脚本 4-2-2 和脚本 4-2-3。

脚本 4-2-1 建立队列的 shell 脚本

```
#!/bin/sh
queue_name="paralle"
# Then you can modify the following items.
qmgr -c "create queue $queue_name "
qmgr -c "set queue $queue_name queue_type = Execution"
qmgr -c "set queue $queue_name Priority = 70"
qmgr -c "set queue $queue_name max_running = 8"
qmgr -c "set queue $queue_name resources_max.cput = 5000:00:00"
qmgr -c "set queue $queue_name resources_min.cput = 00:00:01"
qmgr -c "set queue $queue_name resources_default.cput = 5000:00:00"
qmgr -c "set queue $queue_name resources_default.walltime = 5000:00:00"
qmgr -c "set queue $queue_name max_user_run = 8"
qmgr -c "set queue $queue_name keep_completed = 0"
qmgr -c "set queue $queue_name enabled = True"
qmgr -c "set queue $queue_name started = True"
```

表 4-2-1 队列常用属性及其含义

属性值	含义
queue_type	队列类型，一般为 execution（执行）
Prioty	作业优先级
max_running	做多提交作业数
resources_max.cput	允许最大占用资源
resources_min.cput	最小使用资源
resources_default.cput	默认分配资源
resources_default.walltime	默认作业最长时间
max_user_run	每个用户最多运行的作业数
keep_completed	完成后，查询任务时延迟的时间
enabled	队列是否可用
started	队列是否启动

表 4-2-2 服务器端管理常用属性及其含义

属性值	含义
default_queue	默认的队列
max_running	同时最多运行的作业数
max_user_run	每个用户最多运行的作业数
query_other_jobs	允许用户查询其他用户的作业

表 4-2-3 prologue 和 epilogue 文件的名令行参数含义

参数	含义
$1	任务号
$2	用户名

$3	用户组名
$4	任务名
$5	活动号
$6	请求资源清单
$7	用过的资源清单
$8	队列

脚本 4-2-2 prologue 脚本示例

```
#!/bin/sh
echo "Job ID:              $1"
echo "Job username:    $2"
echo "Job group:         $3"
echo
echo "The job begins at `date`"
exit 0
```

脚本 4-2-3 epilogue 脚本示例

```
#!/bin/sh
echo "The job ends at `date`"
echo -n "Nodes for use are:     "
for i in $(sort /var/spool/torque/aux/$1 | uniq)
do
   echo -n "$i "
done
echo -e "\nJob ID:                $1"
echo "Job username:          $2"
echo "Job group:             $3"
echo "Job name:              $4"
echo "Job session:           $5"
echo "Job queue:             $8"
echo "Request Resource:      $6"
echo "Available Resource:    $7"
exit 0
```

运行这个脚本后，任务的结束报告单上会打印出类似下面的字样：

```
Job ID:              168.compuchem
Job username:    user02
Job group:         user02

The job begins at Tue May 12 14:43:33 EDT 2011
<<<…用户软件的输出（实际为 stdout 的重定位）…>>>
The job ends at Tue May 12 14:43:35 EDT 2011
```

Nodes for use are: node022

Job ID: 168.compuchem
Job username: user02
Job group: user02
Job name: H2
Job session: 2874
Job queue: parallel
Request Resource: cput=5000:00:00, neednodes=1: ppn=8, nodes=1: ppn=8, walltime=5000:00:00
Available Resource: cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:02

这对管理作业非常有帮助。

管理员还可以删除用户的作业，见下节。

§4.2.2 普通用户管理

普通用户的管理任务比较简单，常用的只有三个：

1. 提交作业:qsub

最简单的例子，提交一个 PBS 脚本（普通用户）：

```
$ qsub submit.pbs
```

以脚本 2-6-2 为例，最重要的几个参数如下：

```
#PBS -l nodes=node023:ppn=8+node027:ppn=8
```

这是表明要使用节点的个数及每个节点所要使用的 CPU 数。通过下面的表 4-2-4 说明它的用法。

表 4-2-4 Torque 的资源分配方法举例

nodes=3:ppn=4	Torque将给本作业分配3个节点（由Torque自动选择），每个节点使用4个CPU。3个节点将有Torque自动选择。
nodes=node021:ppn=8+node025:ppn=8	Torque将给本作业分配到节点node021和node025，分别使用8个CPU。

```
#PBS -q batch
```

表示用户要提交的队列，这里是“batch”。

```
#PBS -N first_task
```

表示用户的作业名称，这里是“first_task”。它不能以数字开头。这个名称最好具有一定的意义以便于识别。

其它的选项较少用，就不提了。

2. 查看作业:qstat

这个选项比较丰富。直接查看作业可以：

```
$ qstat
```

则会显示当前正在运行和排队的作业：

```
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
57.compuchem              pc2-l            user03          32:17:33 R parallel
64.compuchem              graphite_hf_opt  user02          7:45:49  R parallel
```

从左到右依次为作业号、作业名、提交作业的用户、作业已消耗的时间、作业状态、作业所属队列。对于状态，R 表示正在运行，Q 表示等待排队中。

加上参数-n 则可以显示作业占用资源的详细情况。

```
$ qstat -n
                                                                  Req'd    Req'd       Elap
Job ID    Username Queue    Jobname    SessID NDS   TSK Memory Time    S Time
--------------------------------------------------------------------------------------------------------------
57.compuchem    user03   parallel pc2-l    15995   1    --   --    5000: R 32:46
   node023/7+node023/6+node023/5+node023/4+node023/3+node023/2+node023/1
   +node023/0
64.compuchem    user02   parallel graphite_hf_opt   6359   1    --   --    5000: R 7:15
   node026/7+node026/6+node026/5+node026/4+node026/3+node026/2+node026/1
   +node026/0
```

3. 删除作业:qdel

例如要删除作业号为 123 的作业，只需：

```
$ qdel 123
```

可以同时删除多个作业。如要删除作业号为 97、98 两个作业，可以：

```
$ qdel 97 98
```

一般用户只能删除自己的作业，管理员则可以删除所有人的作业。

第五章 GPU 在集群上的应用

§5.1 GPU/CUDA 的安装配置

GPU计算是现代计算机发展的一个非常有趣的新方向。它利用GPU中超大规模的计算单元来优化科学领域中的计算任务。现在，这个领域一直在蓬勃发展，蒸蒸日上。很多最新的超级计算机，如天河一号，都在硬件中加入了GPU以提高速度。至2011年5月，大量的分子动力学软件，如Gromacs，NAMD等，和量子化学软件TeraChem都已经提供了GPU计算的功能。因此，本教程在最后有必要提供一些与GPU和CUDA配置有关的东西。

这里，将以GeForce GTX 460为例进行安装。

§5.1.1 驱动程序的安装

读者有了GPU的硬件后，需要将其配置在操作系统中。Linux（这里是Debian）必须事先装好图形界面。

其实也可以不装，但是这样的话实现方法就会变相当麻烦，这里就不讲了。

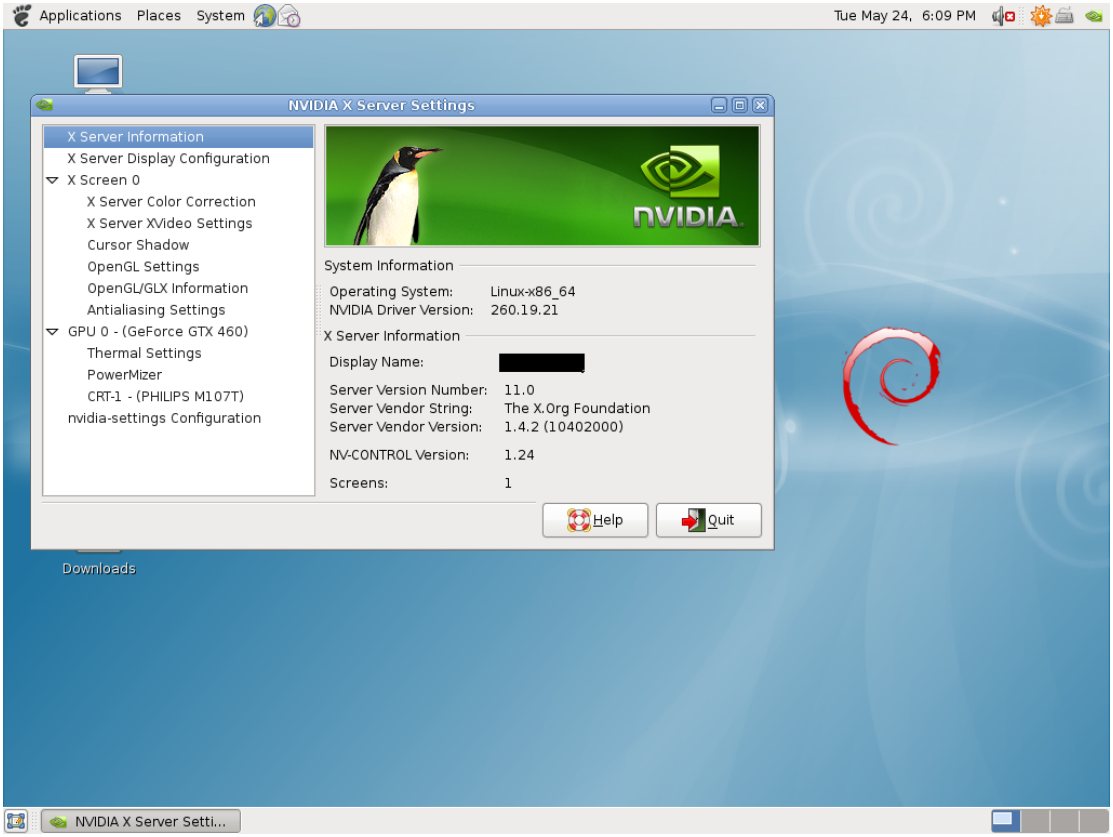
GPU安装非常容易。要安装GPU的驱动，首先停止掉X，然后在nVIDIA的官网下载其驱动devdriver_3.2_linux_64_260.19.21.run，执行即可。不用重启，执行startx就可以看到新的图形界面。可以在Linux的菜单System/Preference/中可以找到nVIDIA的配置菜单，见图5-1-1。

§5.1.2 程序开发环境

要利用GPU进行计算和编程，还需要安装toolkit和SDK，它们的安装非常简单，在nVIDIA的官方网站可以下载到符合系统名称的文件，直接安装即可。

我们的系统是Debian，在nVIDIA下没有专门适应Debian的，我们选择Ubuntu的即可，安装文件为：cudatoolkit_3.2.16_linux_64_ubuntu10.04.run和gpucomputingsdk_3.2.16_linux.run。这两个文件直接执行，即可，一般不会有什么错误。

图5-1-1 GPU驱动程序安装成功效果图



§5.1.3 CUDA 编程测试

CUDA编程所需的编译器命令为nvcc，我们通过一个程序来调试一下，见脚本5-1-1。这个脚本的功能是计算机器中GPU硬件的数量，并列出每一个GPU的名称。这个程序命名为firstcuda.cu，编译：

```
$ nvcc firstcuda.cu -o firstcuda
```

如果编译无误，执行后输出为：

```
There are 1 devices supporting CUDA!
Device 1: "GeForce GTX 460"
```

§5.2 NAMD CUDA 版本的安装

NAMD2.7增加了对CUDA的支持。这里讲一下NAMD CUDA版的编译方法。

其实和§3.2的步骤类似，但是要注意：在CUDA 3.0版本之前，**NAMD CUDA版本不能用intel编译器编译**，因为nVIDIA在Linux下目前只支持gcc，不支持icc。作者的经验表明，如果用icc编译，可以通过，但是运行NAMD后会出现一些莫名奇妙的内存访问错误，如Signal: segmentation violation。而且在编译与CUDA

相关的文件时候，可能会有成员函数不识别的情况。

如果读者拿到这份文档时，nVIDIA和intel公司已经达成了合作，那么，没准CUDA版的NAMD就可以用intel编译器顺利编了。

脚本 5-1-5 CUDA 程序示例

```
// firstcuda.cu
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime_api.h>

int main(int argc, char** argv)
{
    int count;
    int i;
    cudaDeviceProp devprop;

    if(cudaGetDeviceCount(&count) != cudaSuccess)
    {
        printf("Error: CUDA test failed!\n");
        exit(1);
    }
    printf("There are %d devices supporting CUDA!\n", count);
    for(i = 0; i < count; i++)
    {
        if(cudaGetDeviceProperties(&devprop, i) == cudaSuccess)
            printf("Device %d: \"%s\"\n", count, devprop.name)
        else
        {
            printf("Error: Get CUDA properties failed!\n");
            exit(1);
        }
    }
    return 0;
}
```

如果要编译CUDA，只要修改相应的arch/Linux-<your arch>.cuda即可，将CUDADIR改成你安装CUDA的路径。特别注意

LIBCUDARTSO=libcudart.so.2

这一行，如果你安装的是CUDA3，需要把它改成

LIBCUDARTSO=libcudart.so.3

如果是CUDA4，自然要改成LIBCUDARTSO=libcudart.so.4。

然后在文件夹下，选择编译选项：

```
# ./config tcl fftw --with-cuda Linux-x86_64-g++-cuda
```

就可以编译了。

NAMD的CUDA和MPI并行是不矛盾的。执行NAMD-CUDA，需要对namd2后面加上一些参数：+idlepoll以及+devices 0（如果读者有多个GPU）等等。如脚本3-2-1的第7行改为：

```
/usr/local/mpitorque/bin/mpiexec /usr/local/chemsoft/namd2.7/namd2 +idlepoll ubq.conf > ubq.log
```

好了，我们的集群成功的使用了GPU加速！根据作者的实验表明，一个GPU至少相当于12个CPU（比如，12个2.83GHz的Q9550）的速度！

继续前进！

如果读者在教程中还有不清楚的地方，可以参考一下资料。

一些与科学计算集群有关的网站：

大名鼎鼎的China Unix：

http://www.chinaunix.net/

中国集群网，formal的信息：

http://www.cncluster.com.cn/

Debian Cluster，专门介绍以Debian为基础的集群构建：

http://debianclusters.org/index.php/Main_Page

小木虫论坛的计算模拟版，有一些与计算软件配置的信息：

http://emuch.net/bbs

作者博客：

http://hi.baidu.com/coolrainbow/home

以及两个你应该时刻不离手的：

http://www.google.com/

http://www.baidu.com/

一些科学计算集群相关的书籍。

一本非常实用的书籍，讲集群组建技术的：

车静光. 微机集群组建、优化和管理. 北京：机械工业出版社，2004.

MIT出的一本介绍集群的非常全面的书，从硬件，软件配置到MPI编程都有所介绍：

Thomas Sterling. *Beowulf Cluster Computing with Linux*. MIT Press. 2001.

到此，一个功能完整的集群就建立完毕了。

读者也可以歇口气了。我们这个山寨，便宜，精巧的集群已经具备了一个一般集群所有的性质。当然，这里没有介绍集群的更高级技术，比如专业的集群会使用专门的文件系统如Lustre，专业的作业调度软件Slurm，高级的网络如InfiniBand等，某些IP转发、虚拟主机等技术也会应用。本教程只是起到了一个入门的作用。好吧，如果读者有兴趣继续研究集群，那么就来研究这些更高级的技术吧！加油！