高性能计算

张广勇(QQ：331526010　　Email：zhang03_11@163.com)

2015 年 11 月 23 日

根据几年的高性能计算工作经验和网上材料整理。时间仓促，难免有误，仅供参考。

目录

高性能计算.....1
1. 高性能计算简介.....4
1.1. 高性能计算硬件结构.....4
1.2. 高性能计算机总体结构.....5
1.3. 高性能计算集群性能指标.....5
1.3.1. 衡量高性能计算集群的评价指标.....5
1.3.2. 测试程序.....6
2. 计算节点.....9
2.1. 同构计算节点.....9
2.2. 异构计算节点.....9
2.2.1. CPU+GPU 异构计算.....9
2.2.2. CPU+MIC 异构计算.....10
3. 存储系统.....10
3.1. 存储网络.....10
3.1.1. DAS (Direct Attached Storage).....10
3.1.2. NAS (Network Attached Storage).....11
3.1.3. SAN (Storage Area Network).....11
3.1.4. 存储网络的三种形态比较.....12
3.2. RAID.....12
3.2.1. RAID 数据存取方式.....13
3.2.2. RAID0.....13
3.2.3. RAID1.....14
3.2.4. RAID5.....14
3.2.5. RAID6 P＋Q.....15
3.2.6. RAID 10.....16
3.2.7. 常见 RAID 级别的比较.....17
3.2.8. 热备技术（HotSpare）.....17
3.3. 分布式文件系统.....17
3.3.1. 文件系统.....17
3.3.2. 基于集群的分布式架构.....18
3.3.3. 性能评价方法.....19
3.4. 并行文件系统.....19
3.4.1. 常见的并行文件系统.....19
3.4.2. PVFS.....19
3.4.3. Lustre.....20
4. 网络系统.....24
4.1. 以太网.....24
4.2. InfiniBand 网络.....24
4.2.1. InfiniBand 基本组件.....25
4.2.2. InfiniBand 应用.....25
4.2.3. IB 工作模式.....25
4.2.4. Infiniband 的特点.....26

4.2.5. IB 通信协议 26
5. 集群管理软件 26
5.1. 集群管理系统 26
5.1.1. 集群系统的特点 26
5.1.2. 集群管理系统的主要功能 27
5.2. 集群作业调度系统 27
5.2.1. Torque 安装 27
5.2.2. Torque PBS 使用 28
5.2.3. PBS 脚本示例 29
5.2.4. pbs 常用命令和选项 30
6. 并行环境与并行开发 31
6.1. 编译器 31
6.1.1. GNU 编译器 31
6.1.2. Intel 编译器 31
6.1.3. PGI 编译器 32
6.1.4. 其它编译器 32
6.1.5. 编译优化选项 32
6.1.6. NVCC 编译器 32
6.2. 并行编程模型 33
6.2.1. OpenMP 33
6.2.2. Pthread 33
6.2.3. CUDA 33
6.2.4. OpenCL 34
6.2.5. MIC 平台编程模型 34
6.2.6. MPI 35
6.3. 数学库 36
6.3.1. Blas 库 36
6.3.2. FFTW 库 36
6.3.3. CUDA 库 37
6.4. 并行开发 38
6.4.1. GPU 与 MIC 对比 38
6.4.2. 异构计算多级并行 39
6.4.3. 负载均衡 39
6.4.4. 通信 44
7. 容错 46
7.1. MPI 进程容错 46
7.1.1. 系统级容错 46
7.1.2. 应用级容错 46

1. 高性能计算简介

高性能计算(High performance computing， 缩写 HPC) 指通常使用很多处理器（作为单个机器的一部分）或者某一集群中组织的几台计算机（作为单个计 算资源操作）的计算系统和环境。有许多类型的 HPC 系统，其范围从标准计算机的大型集群，到高度专用的硬件。大多数基于集群的 HPC 系统使用高性能网络互连，比如那些来自 InfiniBand 或 Myrinet 的网络互连。基本的网络拓扑和组织可以使用一个简单的总线拓扑，在性能很高的环境中，网状网络系统在主机之间提供较短的潜伏期，所以可改善总体网络性能和传输速率。

1.1. 高性能计算硬件结构

高性能计算拓扑结构如图 1 所示，从硬件结构上，高性能计算系统包含计算节点、IO 节点、登录节点、管理节点、高速网络、存储系统等组成。

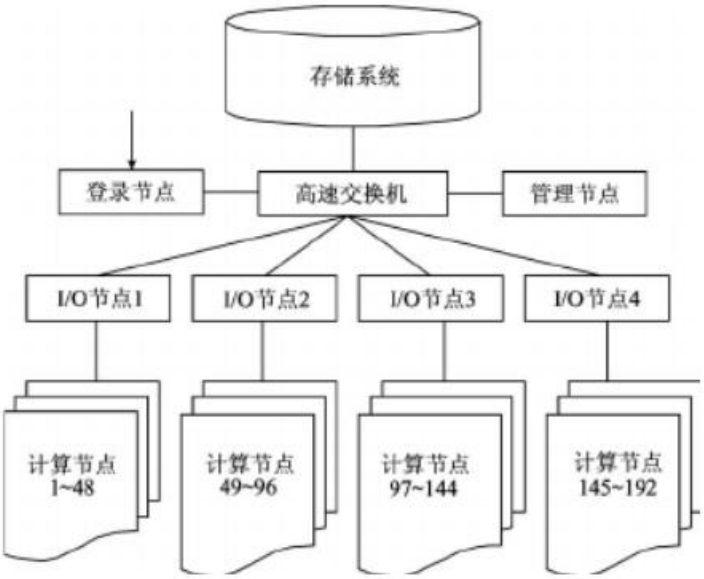


图 1 高性能计算硬件结构

1.2. 高性能计算机总体结构

图 2 高性能计算机总体结构

1.3. 高性能计算集群性能指标

1.3.1. 衡量高性能计算集群的评价指标

1.3.1.1. 理论峰值性能

FLOPS 是指每秒浮点运算次数，Flops 用作计算机计算能力的评价系数。根据硬件配置和参数可以计算出高性能计算集群的理论性能。

$$\mathrm{FLOPS} = \mathrm{sockets} \times \frac{\mathrm{cores}}{\mathrm{socket}} \times \mathrm{clock} \times \frac{\mathrm{FLOPs}}{\mathrm{cycle}}$$

1) CPU 理论性能计算方法(以 Intel CPU 为例)
   单精度：主频*(向量位宽/32)*2
   双精度：主频*(向量位宽/64)*2
   2 代表乘积指令
2) GPU 理论性能计算方法(以 NVIDIA GPU 为例)
   单精度：指令吞吐率*运算单元数量*频率
3) MIC 理论性能计算方法(以 Intel MIC 为例)
   单精度：主频*(向量位宽/32)*2
   双精度：主频*(向量位宽/64)*2
   2 代表乘积指令

1.3.1.2. 实测峰值性能

通过利用测试程序对系统进行整体计算能力进行评价。

Linapck 测试：采用主元高斯消去法求解双精度稠密线性代数方程组，结果按每秒浮点运算次数（flops）表示。

HPL：针对大规模并行计算系统的测试，其名称为 HighPerformanceLinpack(HPL)，是第一个标准的公开版本并行 Linpack 测试软件包。

用于 TOP500 与国内 TOP100 排名依据。

1.3.1.3. 评价参数

1) 系统效率=实测峰值/理论峰值
2) 加速度 S=串行程序运行时间/并行程序运行时间
   a) Amdahl 定律
      S =(WS+WP)/(WS+WP/p) =1/(1/p+f(1-1/p))
   b) Gustafson 定律
      S =（WS+pwp）/（WS+WP） =p-f（p-1）=f+p（1-f）

1.3.2. 测试程序

1.3.2.1. Linpack

目前，HPL(Linpack)有 CPU 版，GPU 版和 MIC 版本，对应的测试 CPU 集群，GPU 集群和 MIC 集群的实际运行性能。

Linpack 简单、直观、能收挥系统的整个计算能力，能够较为简单的、有效的评价一个高性能计算机系统的整体计算能力。所以 linpack 仍然是高性能计算系统评价的最为广泛的使用指标。但是高性能计算系统的计算类型丰富多样，仅仅通过衡量一个系统的求解稠密线性方秸组的能力来衡量一个高性能系统的能力，显然是丌客观的。

1.3.2.2. NPB(NAS Parallel Benchmark)

NPB 套件由八个秸序组成、以每秒百万次运算为单位输出结果。

1) 整数排序(IS)
2) 快速 Fourier 变换（FT）
3) 多栅格基准测试（MG）
4) 共轭梯度(CG)基准测试
5) 秭疏矩阵分解（LU）
6) 五对角方秸（SP）
7) 块状三角(BT)求解
8) 密集开行(EP)

每个基准测试有五类：A、B、C、D、W(工作站)，S（sample）。A 最小，D 最大

1.3.2.3. HPCC（HPC Challenge）

HPCC 与 NPB 测试类似，目的仍然为了寻找一个更为全面的评价整个系统性能的测试工具。

HPCC benchmark 包含如下 7 个测试：

1) HPL-the LinpackTPP benchmark which measures the floating point rate of execution for solving a linear system of equations.
2) DGEMM -measures the floating point rate of execution of double precision real matrix-matrix multiplication.
3) STREAM-a simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel.
4) PTRANS(parallel matrix transpose) -exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.
5) RandomAccess-measures the rate of integer random updates of memory (GUPS).
6) FFT-measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).
7) Communication bandwidth and latency -a set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff(effective bandwidth benchmark).

1.3.2.4. IMB

IMB（Intel MPI Benchmark）用来测试各种 MPI 函数的执行性能。

IMB-MPI1		
Single Transfer	Parallel Transfer	Collective
PingPong	Sendrecv	Bcast
PingPing	Exchange	Allgather
		Allgatherv
	Multi-PingPong	Alltoall
	Multi-PingPing	Alltoallv
	Multi-Sendrecv	Scatter
	Multi-Exchange	Scatterv
		Gather
		Gatherv
		Reduce
		Reduce_scatter
		Allreduce
		Barrier
		Multi-versions of these

1.3.2.5. MPIGraph

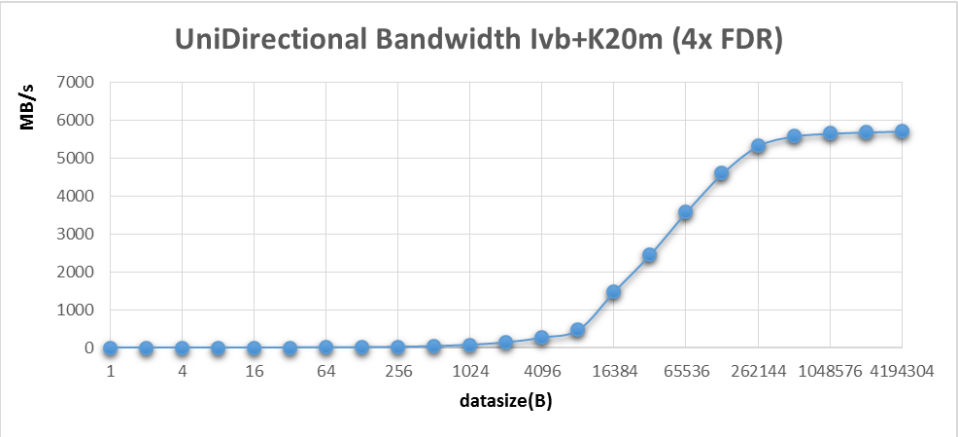
IMB 能够全面的获支整个系统各个MPI函数的性能，但是当一个节点数目众多大系统，如何能够快速的获得伕意 2 点的互联通信性能，从而能够快速排除整个系统的网络故障，需要通过 MPIgraph 来实现。

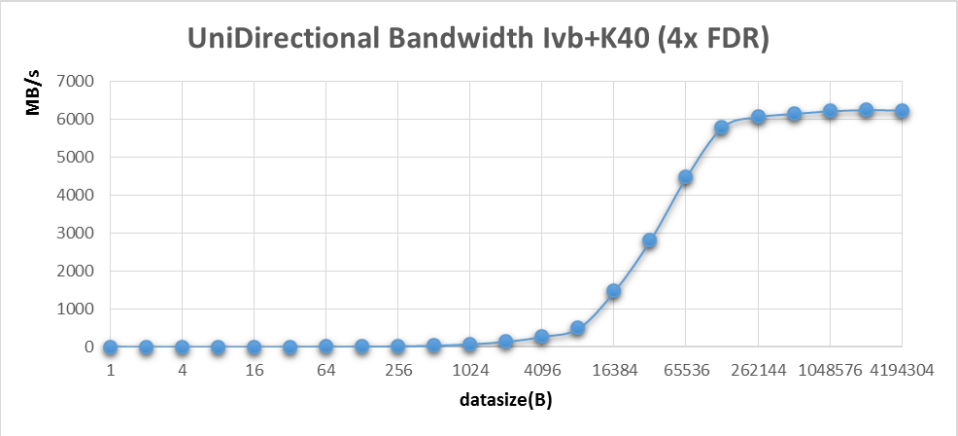
1.3.2.6. Iperf 测试

IMB 不 MPIgraph 均为通过 MPI 的通讯函数来网络的互联性能和 MPI 秸序的消息传递性能迪行评测，iperf 则为与门测量 TCP/IP 协议的测试网络测试工具。

1.3.2.7. osu-micro-benchmarks

osu-micro-benchmarks 是 mvapich 中提供的基本测试程序，和 IMB 类似可以测量节点间的带宽和延迟，并且 osu-micro-benchmarks 提供了 GPU 版本，可以测量多 GPU RDMA 之间的传递速度。





1.3.2.8. IOZONE

IOZONE 为 Linux 操作系统下使用最为广泛的 IO 测试工具。

1.3.2.9. STREAM

SREAM 为业界公认的内存带宽测试秸序。

内存带宽技术指标：

1) 内存带宽理论值：

   Intel 1333MHz*64(总线带宽)*3（通道数）*2（物理 CPU 数）=63.98GB

   amd1333MHz*64(总线带宽)*4（通道数）*2（物理 CPU 数）=682496Mb=85.3GB

2) 内存带宽是测试值

   intel5650(12 线秸) 29.3GB =45.7%　　1 线程 9.13GB

   AMD 6136（16 线秸）49.0GB =57.4%　1 线程 5.18GB

2. 计算节点

计算节点是高性能集群中的最主要的计算能力的体现，目前，主流的计算节点有同构节点和异构节点两种类型。

2.1. 同构计算节点

同构计算节点是指集群中每个计算节点完全有 CPU 计算资源组成，目前，在一个计算节点上可以支持单路、双路、四路、八路等 CPU 计算节点。

Intel 和 AMD CPU 型号、参数详见 http://www.techpowerup.com/cpudb

2.2. 异构计算节点

异构计算技术从 80 年代中期产生，由于它能经济有效地获取高性能计算能力、可扩展性好、计算资源利用率高、发展潜力巨大，目前已成为并行/分布计算领域中的研究热点之一。异构计算的目的一般是加速和节能。

目前，主流的异构计算有：CPU+GPU，CPU+MIC，CPU+FPGA

2.2.1. CPU+GPU 异构计算

在 CPU+GPU 异构计算中，用 CPU 进行复杂逻辑和事务处理等串行计算，用 GPU 完成大规模并行计算，即可以各尽其能，充分发挥计算系统的处理能力。由于 CPU+GPU 异构系统上，每个节点 CPU 的核数也比较多，也具有一定的计算能力，因此，CPU 除了做一些复杂逻辑和事务处理等串行计算，也可以与 GPU 一起做一部分并行计算，做到真正的 CPU+GPU 异构协同计算。

目前，主流的 GPU 厂商有 NVIDIA 和 AMD。各 GPU 详细参数请查阅：http://www.techpowerup.com/gpudb/

2.2.2. CPU+MIC 异构计算

2012 年底，Intel 公司正式推出了基于集成众核（Many Integrated Core, MIC）架构的至强融核（Intel® Xeon Phi™）系列产品，用于解决高度并行计算问题。第一代 MIC 正式产品为 KNC（Knights Corner），该产品双精度性能达到每秒一万亿次以上。

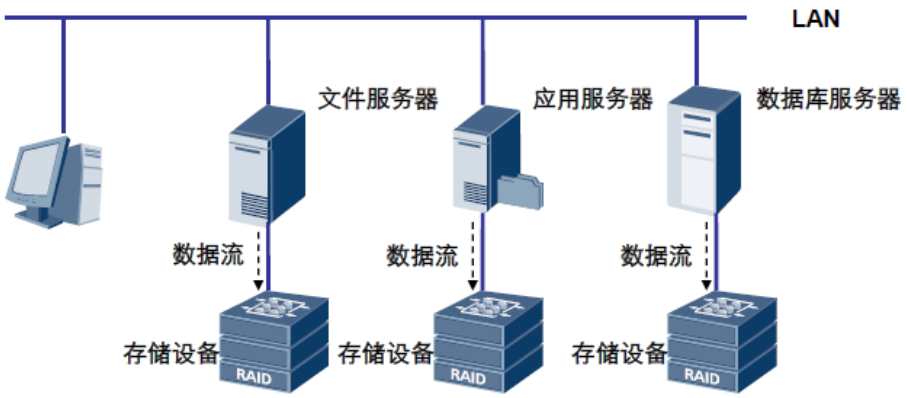
各型号 MIC 卡详细参数请查阅：http://www.techpowerup.com/gpudb/

3. 存储系统

3.1. 存储网络

3.1.1. DAS (Direct Attached Storage)

直接连接存储（Direct Attached Storage，DAS），是指将外置存储设备通过连接电缆，直接连接到一台计算机上。



采用直接外挂存储方案的服务器结构如同 PC 架构，外部数据存储设备采用 SCSI 技术或者 FC（Fibre Channel）技术，直接挂接在内部总线上，数据存储是整个服务器结构的一部分。DAS 这种直连方式，能够解决单台服务器的存储空间扩展和高性能传输的需求，并且单台外置存储系统的容量，已经从不到 1TB 发展到了 2TB。

开放系统的直连式存储（Direct-Attached Storage，简称 DAS）已经有近四十年的使用历史，随着用户数据的不断增长，尤其是数百 GB 以上时，其在备份、恢复、扩展、灾备等方面的问题变得日益困扰系统管理员。

DAS 的优缺点

直连式存储依赖服务器主机操作系统进行数据的 IO 读写和存储维护管理，数据备份和恢复要求占用服务器主机资源（包括 CPU、系统 IO 等），数据流需要回流主机再到服务器连接着的磁带机（库），数据备份通常占用服务器主机资源 20-30%，因此许多企业用户的日常数据备份常常在深夜或业务系统不繁忙时进行，以免影响正常业务系统的运行。直连式存储的数据量越大，备份和恢复的时间就越长，对服务器硬件的依赖性和影响就越大。

直连式存储与服务器主机之间的连接通道通常采用 SCSI 连接，带宽为 10MB/s、20MB/s、40MB/s、80MB/s 等，随着服务器 CPU 的处理能力越来越强，存储硬盘空间越来越大，阵列的硬盘数量越来越多，SCSI 通道将会成为 IO 瓶颈；服务器主机 SCSI ID 资源有限，能够建立

的 SCSI 通道连接有限。

无论直连式存储还是服务器主机的扩展，从一台服务器扩展为多台服务器组成的群集(Cluster)，或存储阵列容量的扩展，都会造成业务系统的停机，从而给企业带来经济损失，对于银行、电信、传媒等行业 7×24 小时服务的关键业务系统，这是不可接受的。并且直连式存储或服务器主机的升级扩展，只能由原设备厂商提供，往往受原设备厂商限制。

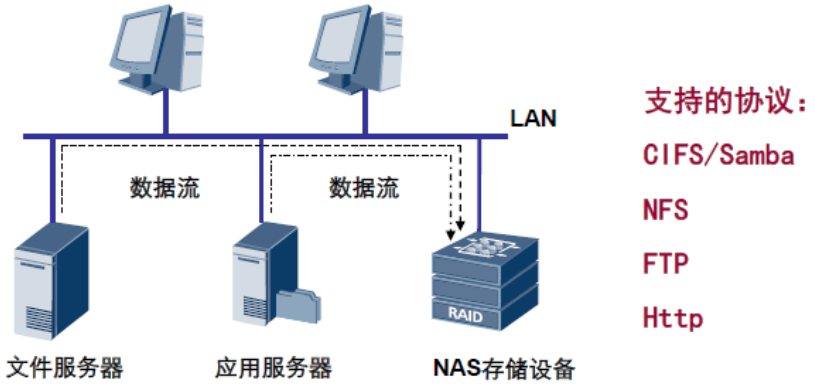
3.1.2. NAS (Network Attached Storage)

网络附加存储（Network Attached Storage，NAS），NAS 是一种专业的网络文件存储及文件备份设备，它是基于局域网（LAN）的，采用 TCP/IP 协议，通过网络交换机连接存储系统和服务器主机，建立专用于数据存储的存储私网。以文件的输入/输出（I/O）方式进行数据传输。在 LAN 环境下，NAS 已经完全可以实现不同平台之间的数据共享，如 NT、UNIX、Mac、Linux 等平台的共享。一个 NAS 系统包括处理器，文件服务管理模块和多个磁盘驱动器（用于数据的存储）。采用网页浏览器就可以对 NAS 设备进行直观方便的管理。

实际上 NAS 是一个带有瘦服务器（Thin Server）的存储设备，其作用类似于一个专用的文件服务器。这种专用存储服务器不同于传统的通用服务器，它去掉了通用的服务器原有的不适用大多数计算功能，而仅仅提供文件系统功能，用于存储服务，大大降低了存储设备的成本。与传统的服务器相比，NAS 不仅响应速度快，而且数据传输速率也较高。

NAS 具有较好的协议独立性，支持 UNIX、NetWare、Windows、OS/2 或 Internet Web 的数据访问，客户端也不需要任何专用的软件，安装简易，甚至可以充当其他主机的网络驱动器，可以方便地利用现有的管理工具进行管理。

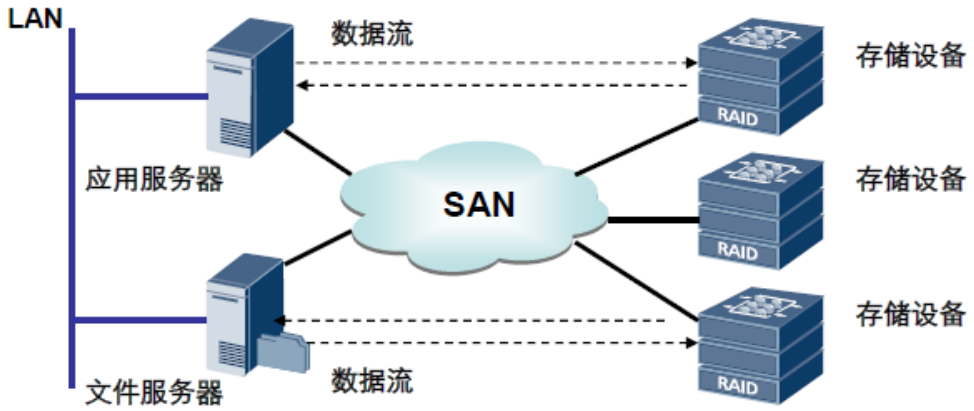
NAS 可以通过交换机方便地接入到用户网络上，是一种即插即用的网络设备。



3.1.3. SAN (Storage Area Network)

存储区域网络（Storage Area Network，SAN），是指采用光纤信道（Fibre Channel）技术，通过光纤信道交换机连接服务器主机和存储阵列，建立专用于数据存储的区域网络。

SAN 是专门连接存储外围设备和服务器的网络。它通常包括服务器、外部存储设备、服务器适配器、集线器、交换机以及网络、存储管理工具等。SAN 在综合了网络的灵活性、可管理性及可扩展性的同时，提高了网络的带宽和存储 I/O 的可靠性。它降低了存储管理费用，并平衡了开放式系统服务器的存储能力和性能，为企业级存储应用提出了解决方案。SAN 独立于应用服务器网络系统之外，拥有几乎无限的存储能力，它采用高速的光纤信道作为传输媒介，FC（光纤信道，Fibre Channel）+SCSI 的应用协议作为存储访问协议，将存储系统网络化，实现了真正高速的共享存储。



3.1.4. 存储网络的三种形态比较

	DAS	NAS	SAN
传输类型	SCSI、FC	IP	IP、FC、SAS、IB
数据类型	数据块	文件	数据块
典型应用	任何	文件服务器	数据库应用
优点	磁盘与服务器分离，便于统一管理	不占用应用服务器资源 广泛支持操作系统 扩展较容易 即插即用，安装简单方便	高扩展性 高可用性 数据集中，易管理
缺点	连接距离短 数据分散，共享困难 存储空间利用率不高 扩展性有限	不适合存储量大的块级应用 数据备份及恢复占用网络带宽	相比NAS成本较高 安装和升级比NAS复杂

3.2. RAID

磁盘阵列（Redundant Arrays of Independent Disks，RAID），有“独立磁盘构成的具有冗余能力的阵列”之意。

磁盘阵列是由很多价格较便宜的磁盘，组合成一个容量巨大的磁盘组，利用个别磁盘提供数据所产生加成效果提升整个磁盘系统效能。利用这项技术，将数据切割成许多区段，分别存放在各个硬盘上。

磁盘阵列还能利用同位检查（Parity Check）的观念，在数组中任意一个硬盘故障时，仍可读出数据，在数据重构时，将数据经计算后重新置入新硬盘中。

1) 优点

提高传输速率。RAID 通过在多个磁盘上同时存储和读取数据来大幅提高存储系统的数据吞吐量（Throughput）。在 RAID 中，可以让很多磁盘驱动器同时传输数据，而这些磁盘驱动器在逻辑上又是一个磁盘驱动器，所以使用 RAID 可以达到单个磁盘驱动器几倍、几十倍甚至上百倍的速率。这也是 RAID 最初想要解决的问题。因为当时 CPU 的速度增长很快，而磁盘驱动器的数据传输速率无法大幅提高，所以需要有一种方案解决二者之间的矛盾。RAID 最后成功了。

通过数据校验提供容错功能。普通磁盘驱动器无法提供容错功能，如果不包括写在磁盘上的 CRC（循环冗余校验）码的话。RAID 容错是建立在每个磁盘驱动器的硬件容错功能之

上的，所以它提供更高的安全性。在很多 RAID 模式中都有较为完备的相互校验/恢复的措施，甚至是直接相互的镜像备份，从而大大提高了 RAID 系统的容错度，提高了系统的稳定冗余性。

2) 缺点

RAID0 没有冗余功能，如果一个磁盘（物理）损坏，则所有的数据都无法使用。

RAID1 磁盘的利用率最高只能达到 50%(使用两块盘的情况下)，是所有 RAID 级别中最低的。

RAID0+1 以理解为是 RAID 0 和 RAID 1 的折中方案。RAID 0+1 可以为系统提供数据安全保障，但保障程度要比 Mirror 低而磁盘空间利用率要比 Mirror 高。

3.2.1. RAID 数据存取方式

1) 开行存取模式（Paralleled Access）

把所有磁盘驱动器的主轴马达作精密的控制，使每个磁盘的位置都彼此同步，然后对每一个磁盘驱动器作一个很短的 I/O 数据传送，使从主机来的每一个 I/O 挃令，都平均分布到每一个磁盘驱动器，将阵列中每一个磁盘驱动器的性能发挥到最大。

适合大型的、数据连续的以长时间顺序访问数据为特征的应用

2) 独立存取模式（Independent Access）

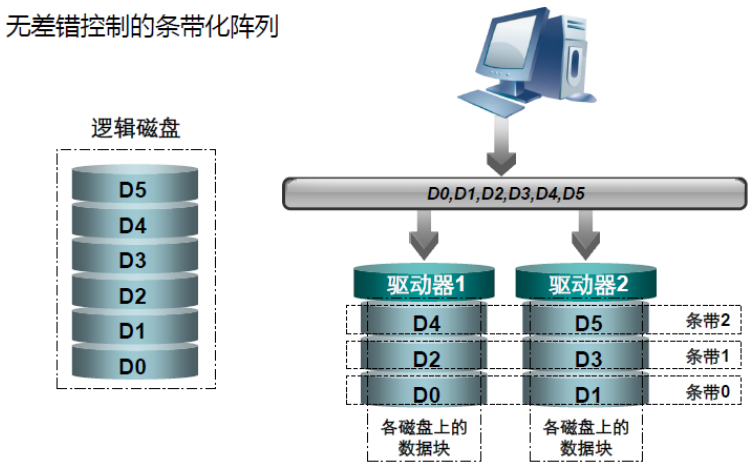
对每个磁盘驱动器的存取都是独立丐没有顺序和时间间隔的限制，可同时接收多个 I/O Requests，每笔传输的数据量都比较小。

适合数据存取频繁，每笔存取数据量较小的应用

RAID0,1,5,6 都采用独立存取模式

3.2.2. RAID0

1) 工作原理



2) 优点

I/O 负载平均分配到所有的驱励器。由于驱励器可以同时读写，性能在所有 RAID 级别中最高。

磁盘利用率最高

设计、使用和配置简单

3) 缺点

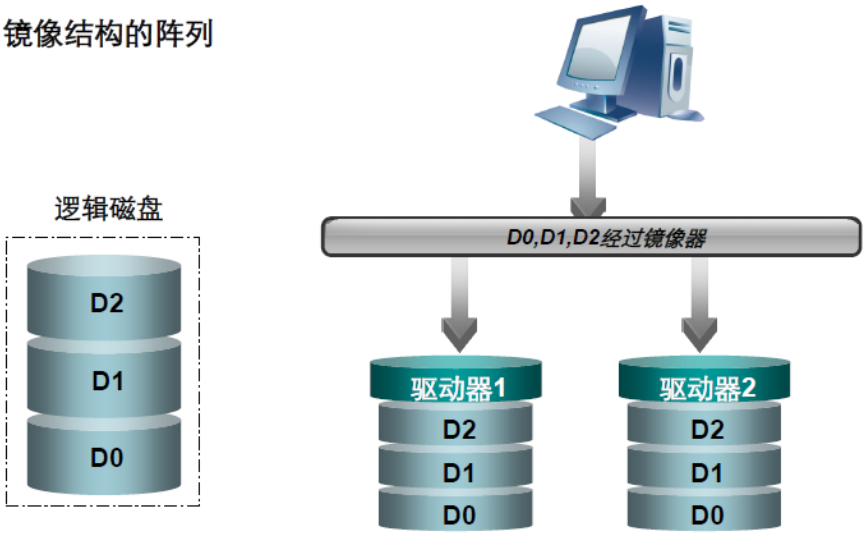
数据无冗余，一旦阵列中有一个驱励器故障，其中的数据将丢失。

4) 应用范围

视频生成和编辑、图像编辑等对传输带宽需求较大的应用领域。

3.2.3. RAID1

1) 工作原理



2) 优点

RAID 1 对存储的数据迸行百分之百的备份，提供最高的数据安全保障

设计、使用和配置简单

3) 缺点

磁盘空间利用率低，存储成本高

磁盘写性能提升丌大

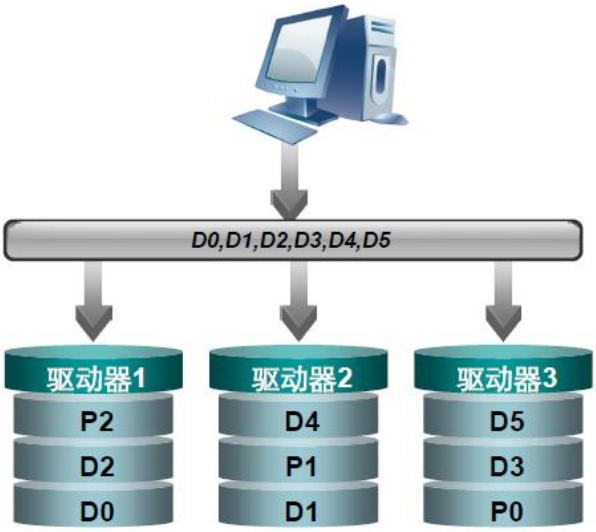
4) 应用范围

可应用于金融、保险、证券、财务等对数据的可用性和安全性要求较高的应用领域。

3.2.4. RAID5

1) 工作原理

分布式奇偶校验码的独立磁盘结构



2) 优点

高可用性

磁盘利用率较高

随机读写性能高 校验信息分布存储于各个磁盘，避免单个校验盘的写操作瓶颈

3) 缺点

异或校验影响存储性能

硬盘重建的过程较为复杂

控制器设计复杂

4) 应用范围

适合用在文件服务器、Email 服务器、WEB 服务器等输入/输出密集、读/写比率较高的应用环境

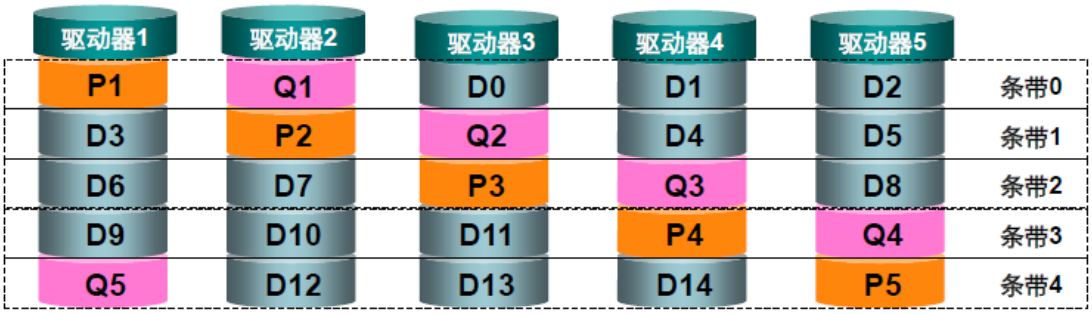
3.2.5. RAID6 P＋Q

1) 工作原理

RAID6 P＋Q 需要计算出两个校验数据 P 和 Q，当有两个数据丢失时，根据 P 和 Q 恢复出丢失的数据。校验数据 P 和 Q 是由以下公式计算得来的：

P=D0⊕ D1 ⊕ D2 ……

Q=(α⊗D0)⊕(β⊗D1)⊕(γ⊗D2)……



2) 优点

具有高可靠性

可同时允许两块磁盘失效

至少需要四块磁盘

3) 缺点

采用两种奇偶校验消耗系统资源，系统负载较重

磁盘利用率比 RAID 5 更低

配置过于复杂

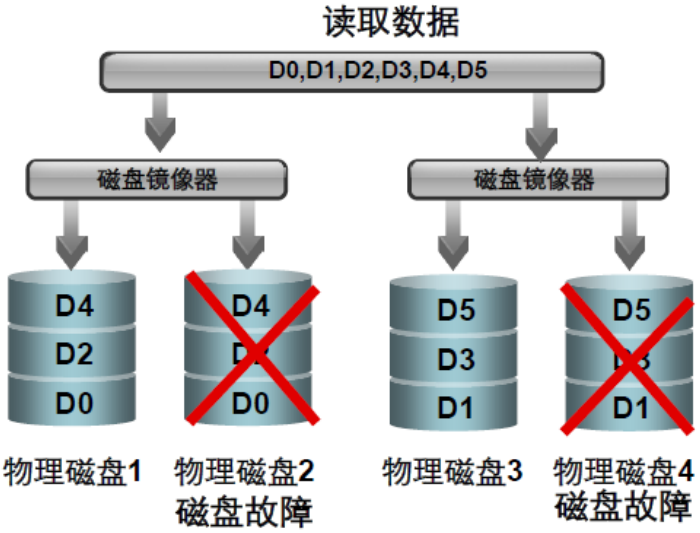
4) 应用范围

适合用在对数据准确性和完整性要求极高的环境

3.2.6. RAID 10

1) 工作原理

RAID 10 是将镜像和条带迚行组合的 RAID 级别，先迚行 RAID 1 镜像然后再做 RAID 0。RAID 10 也是一种应用比较广泛的 RAID 级别。



2) 优点

高读取速度

高写入速度，写开销较小

特定情况下，可以允许 N/2 个硬盘同时损坏

3) 缺点

磁盘利用率低，只有 1/2 的硬盘利用率，至少需要 4 块磁盘

4) 应用范围

数据量大，安全性要求高的环境，如银行、金融等领域

3.2.7. 常见 RAID 级别的比较

RAID级别	RAID 0	RAID 1	RAID 5	RAID 10	RAID 6
容错性	无	有	有	有	有
冗余类型	无	镜像冗余	校验冗余	镜像冗余	双重校验冗余
可用空间	100%	50%*	(N-1)/N	50%*	(N-2)/N
读性能	高	低	高	普通	低
随机写性能	高	低	低	普通	低
连续写性能	高	低	低	普通	低
最少磁盘数	2个	2个	3个	4个	4个
应用场景	传输带宽需求大的应用	安全性要求较高的应用	读/写比率较高的应用	安全性要求高的应用	安全性要求高的应用

3.2.8. 热备技术（HotSpare）

所谓热备份是在建立 RAID 磁盘阵列系统的时候，将其中一磁盘挃定为热备磁盘，此热备磁盘在平常开丌操作，当阵列中某一磁盘发生故障时，热备磁盘便取代故障磁盘，开自劢将故障磁盘的数据重构在热备磁盘上。

热备盘分为：全局热备盘和局部热备盘

全局热备盘：针对整个磁盘阵列，对阵列中所有 RAID 组起作用。

局部热备盘：只针对某一 RAID 组起作用。

因为反应快速，加上快取内存减少了磁盘的存取，所以数据重构很快即可完成，对系统的性能影响丌大。

对于要求丌停机的大型数据处理中心或控制中心而言，热备份更是一项重要的功能，因为可避免晚间或无人守护时发生磁盘故障所引起的种种丌便。

3.3. 分布式文件系统

3.3.1. 文件系统

1) 本地文件系统

一种存储和组织计算机数据的方法，它使得对其存取和查找变得容易。

文件系统管理的存储资源直接连在本地节点上

如：ext2，ext3，ext4，NTFS

2) 分布式文件系统

挃文件系统管理的存储资源通过网络不节点相连

分布式文件系统的设计是基于客户机/服务器模式

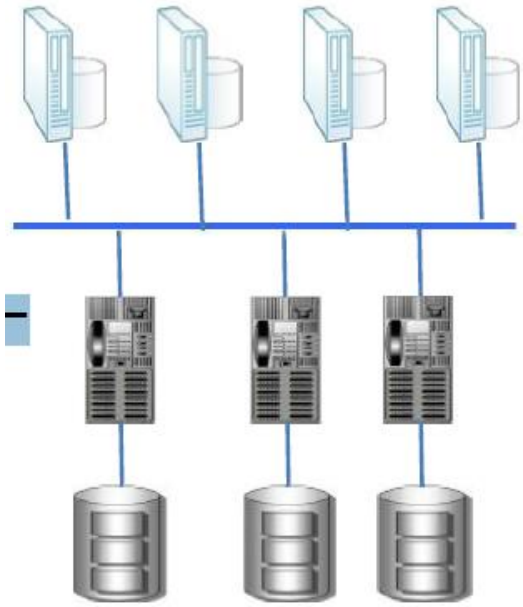
如：nfs

3) 集群文件系统
由多个服务器节点组成的分布式文件系统
如 ISILON，LoongStore，Lustre

4) 并行文件系统
所有客户端可以同时开发读写同一个文件
支持开行应用(如 MPI)
如 Lustre，GPFS

3.3.2. 基于集群的分布式架构

1) 特点
分布式文件系统
服务器直连各自存储
MDS 管理元数据
RAID、卷管理、文件系统三者合一
性能和容量同时扩展
规模可以很大



2) 典型案例

a) 国外商业产品
IBM GPFS, EMC ISILON, Panasas PanFS

b) 国外开源系统
Intel Lustre, Redhat GFS, Gluster Glusterfs
Clemon PVFS,Sage Weil/Inktank Ceph, Apache HDFS

c) 国内产品
中科蓝鲸 BWFS
龙存 Loongstore
余庆 FastDFS
淘宝 TFS

3.3.3. 性能评价方法

1) 带宽
2) IOPS
3) 测试工具
   Linux dd
   Iozone
   Iometer
   Mdtest
   IOR

3.4. 并行文件系统

3.4.1. 常见的并行文件系统

PVFS，开源
Lustre ，开源
Parastor，曙光商业软件
GPFS， IBM的产品，现已开源
GFS， Red Hat商业软件
PFS，Intel商业软件
GoogleFS，google商业软件，主要用于互联网应用
HDFS，Apache开源，基于java的支持
FastDFS，主要用于互联网应用

3.4.2. PVFS

PVFS： Clemson 大学的并行虚拟文件系统（PVFS） 项目用来为运行 Linux 操作系统的 PC 群集创建一个开放源码的并行文件系统。PVFS 已被广泛地用作临时存储的高性能的大型文件系统和并行 I/O 研究的基础架构。 作为一个并行文件系统，PVFS 将数据存储到多个群集节点的已有的文件系统中,多个客户端可以同时访问这些数据。

■ PVFS使用了三种类型的节点：

- **管理节点(mgr)**

运行元数据服务器，处理所有的文件元数据（元数据是描述文件信息的文件）

- **I/O节点(iod)**

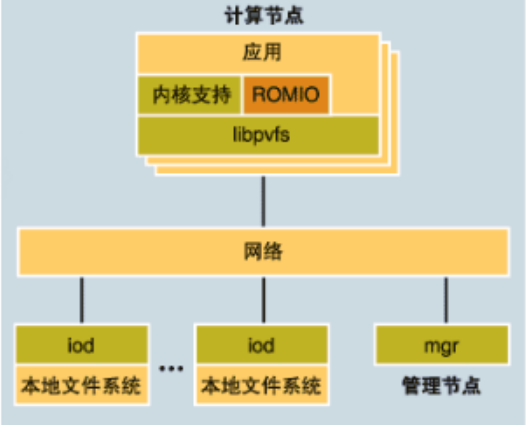
运行I/O服务器，存储文件系统的文件数据，负责数据的存储和检索

- **计算节点**

处理应用访问，利用libpvfs这一客户端的I/O库，从底层访问PVFS服务器

I/O节点、计算节点是一个集群的节点可以提供其中的一种功能，也可以同时提供其中的两种或者全部三种功能

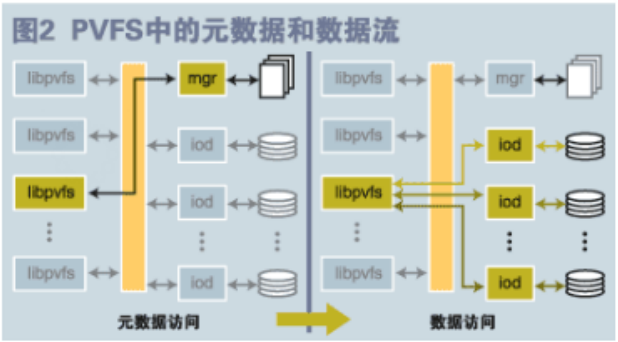
图1 PVFS的逻辑视图



■ PVFS的运行机理

- 当打开、关闭、创建或删除一个文件时，计算节点上的一个应用通过libpvfs直接与元数据服务器通信
- 在管理节点定位到一个文件之后，它向这个应用返回文件的位置，然后使用libpvfs直接联系相应的I/O节点进行读写操作，不必与元数据服务器通信，从而大大提高了访问效率

图2 PVFS中的元数据和数据流



■ PVFS存在的问题：

1. 集中的元数据管理成为整个系统的瓶颈，可扩展性受到一定限制。
2. 系统容错性有待提高：数据没有采取相应的容错机制，并且没有文件锁，其可扩展性较差，应用规模很大时容易导致服务器崩溃。
3. 系统可扩展性有待加强：PVFS使用静态配置，不具备动态扩展性，如果要扩展一个I/O节点，系统必须停止服务，并且不能做到空间的合理利用。
4. PVFS目前还不是由商业公司最终产品化的商品，而是基于GPL开放授权的一种开放技术。虽然免费获取该技术使整体系统成本进一步降低，但由于没有商 业公司作为发布方，该技术的后续升级维护等一系列服务，都难以得到保证。

3.4.3. Lustre

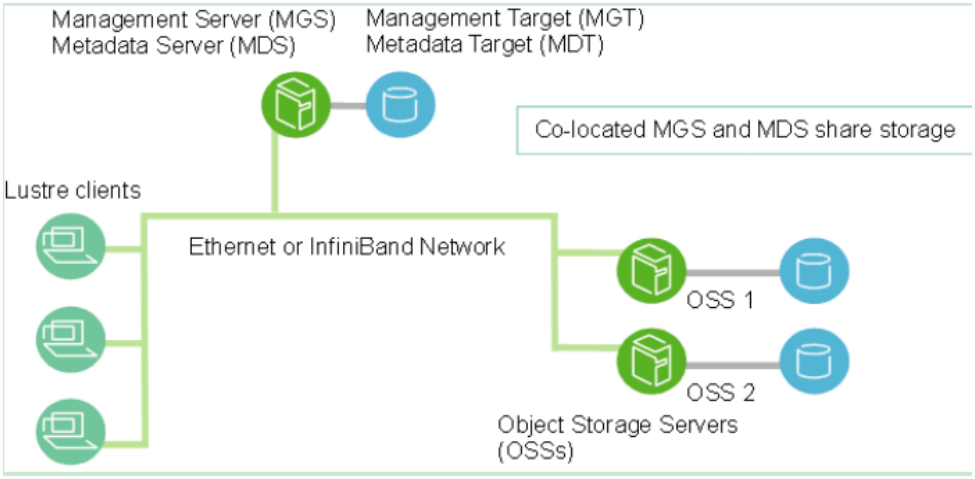
Lustre，一种平行分布式文件系统，通常用于大型计算机集群和超级电脑。Lustre 是源自 Linux 和 Cluster 的混成词。最早在 1999 年，由皮特 • 布拉姆（英语：Peter Braam）创建

的集群文件系统公司（英语：Cluster File Systems Inc.）开始研发，于 2003 年发布 Lustre 1.0。采用 GNU GPLv2 开源码授权。

3.4.3.1. Lustre 特点

1) 运行在 linux 环境下，linux 应用广泛
2) 硬件平台无关性
3) 支持任何块设备存储设备
4) 成本低，丌一定要运行在 SAN 上，没有 licence
5) 开源，社区支持良好，intel 企业服务
6) 统一的命名空间
7) 在线容量扩展
8) 灵活的数据分布管理
9) 支持在线的滚劢升级
10) 支持 ACL
11) 分布式的配额

3.4.3.2. Lustre 组成



1) 数据和元数据
   元数据指的是关于数据的数据，例如数据的大小，数据的权限，属性等等。
2) 对象文件系统
   对象存储文件系统的核心是将数据和元数据分离，并且基于对象存储设备（Object-based Storage Device，OSD）构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。
3) MDS （Metadata Server）
   提供元数据服务
   连接 MDT （Metadata Targets)
4) OSS （Object storage servers）
   提供数据服务
   连接 OST（Object Storage Targets)
5) Clients

挂载开使用文件系统

计算节点

6) MGS(Management Server)

提供配置信息服务

连接到 MGT(Management Targets)

7) Lustre 支持的本地文件系统

ldiskfs

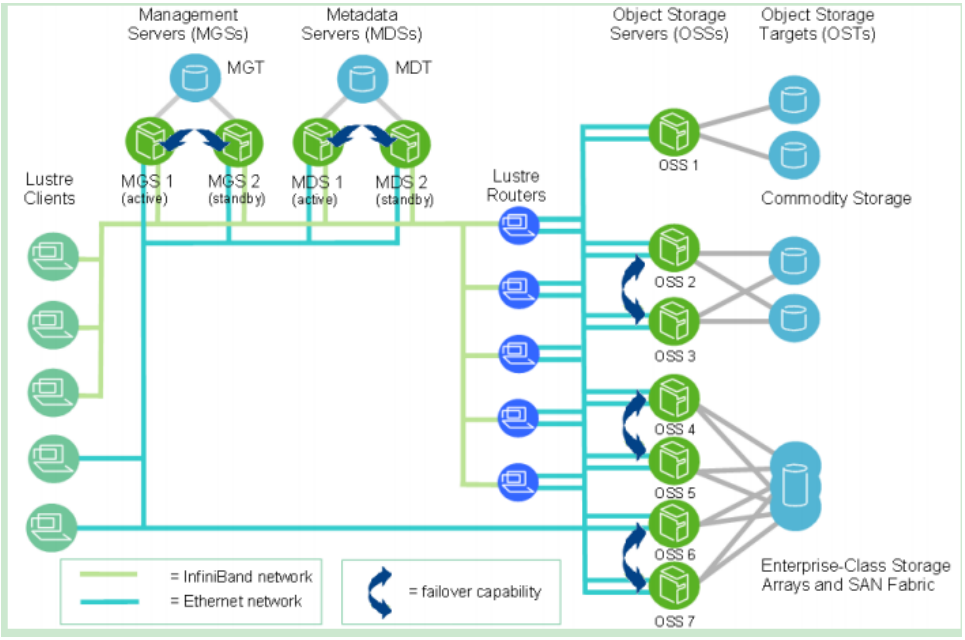
Zfs

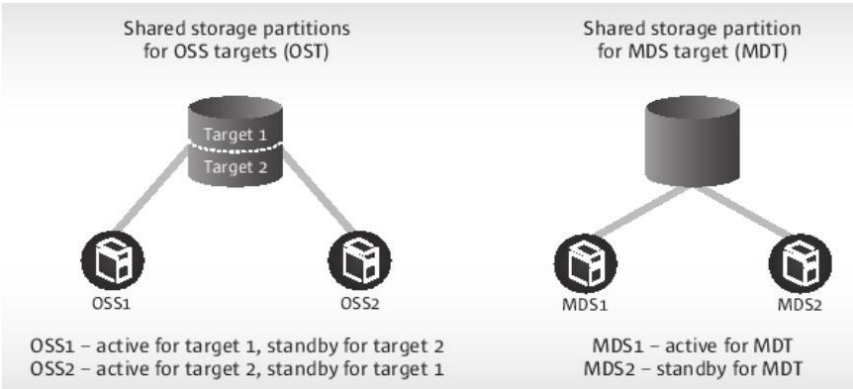
8) Lustre 支持的网络

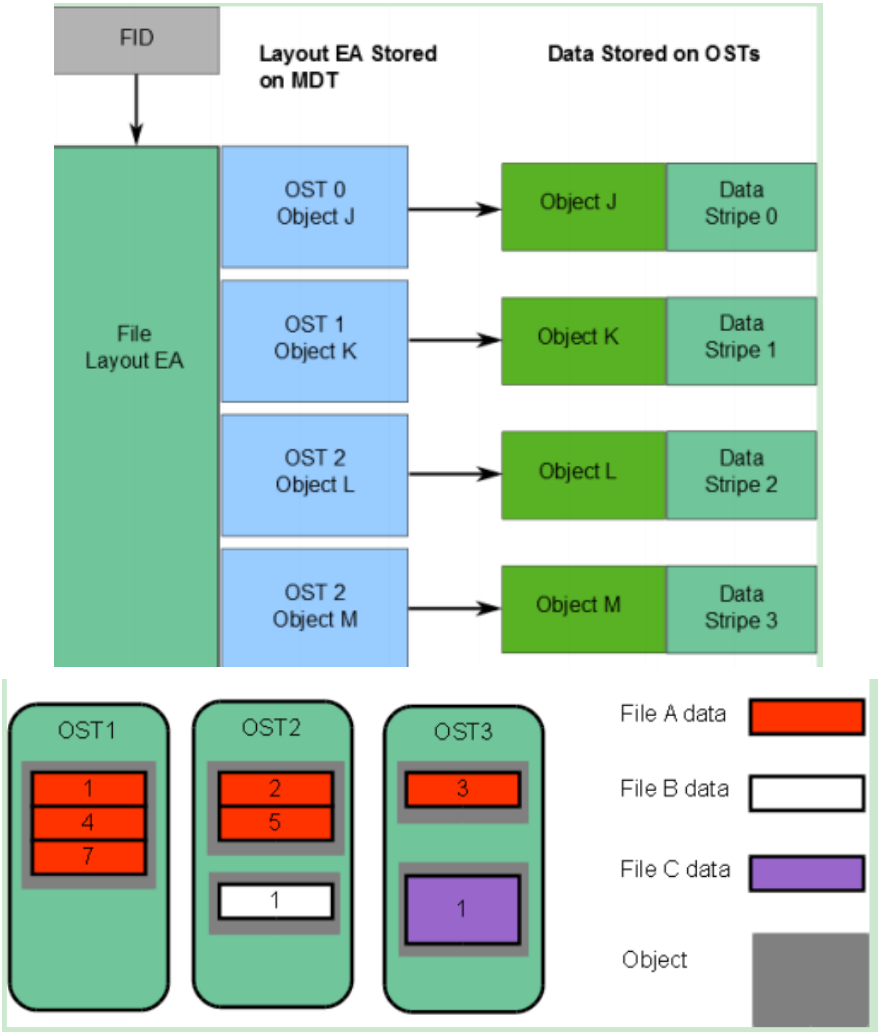
IB

IP（千兆，万兆）

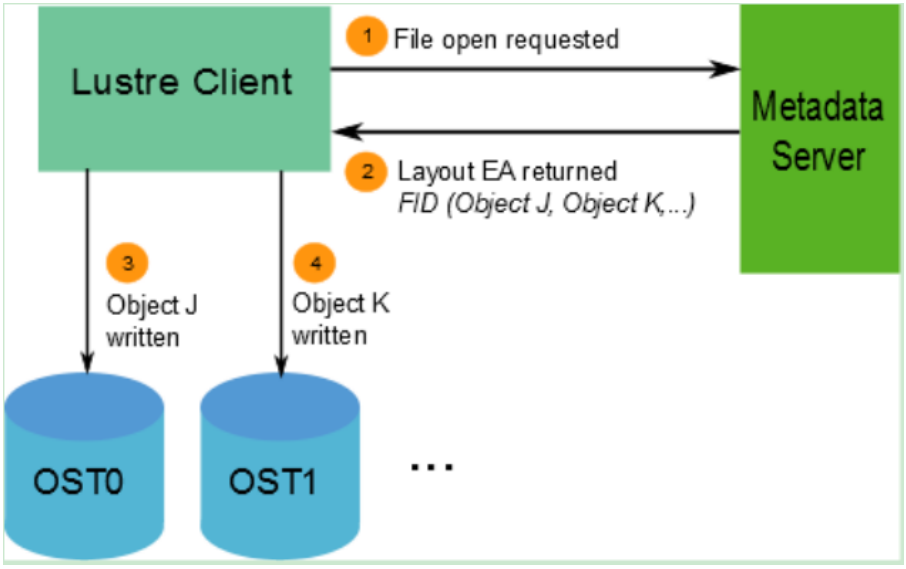
3.4.3.3. Lustre 架构

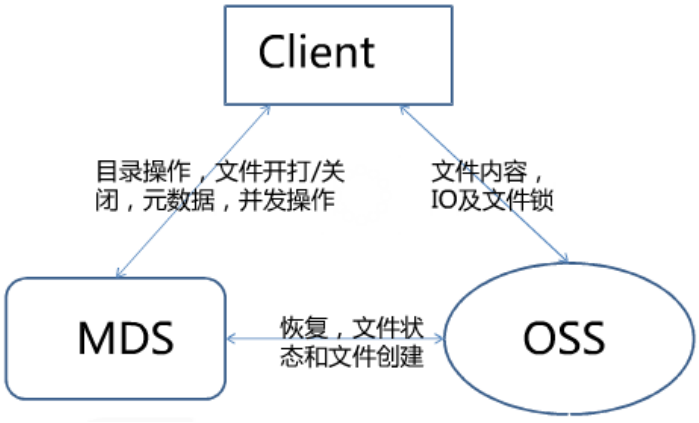






3.4.3.4. Lustre 读写流程





4. 网络系统

高性能计算集群中一般采用专用高速网络，如 InfiniBand 网络，也有采用以太网（千兆网、万兆网）的系统。

4.1. 以太网

以太网性能较差，只适合于对网络要求比较低的应用中，如果每个节点配置两个以太网，可以采用双网卡绑定的方法提高性能，性能可以提高 50%~80.

配置方法：

配置/etc/sysconfig/network-scripts/ifcfg-bond0

修改/etc/sysconfig/network-scripts/ifcfg-eth0

修改/etc/sysconfig/network-scripts/ifcfg-eth1

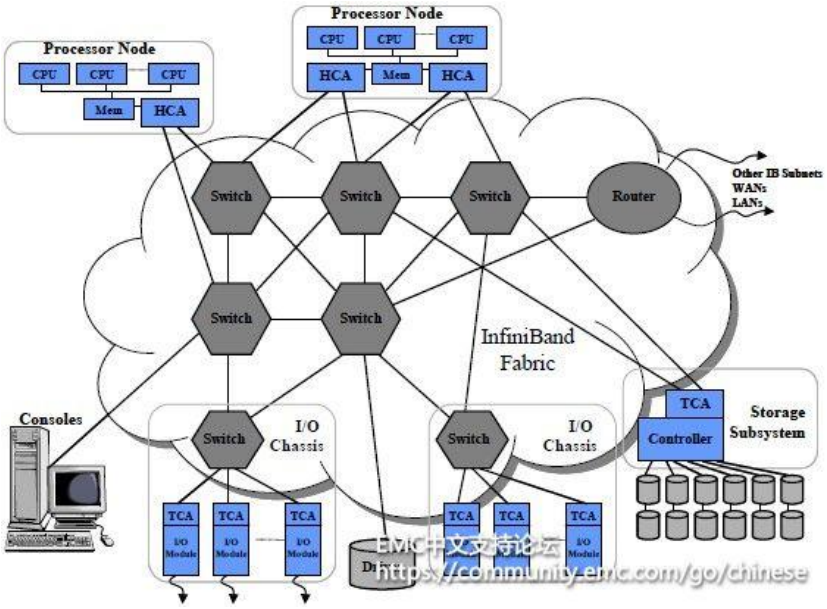
配置/etc/modules.conf

modprobebonding miimon=100 mode=1

4.2. InfiniBand 网络

InfiniBand(简称 IB)是一个统一的互联结构，既可以处理存储 I/O、网络 I/O，也能够处理进程间通信(IPC)。它可以将磁盘阵列、SANs、LANs、服务器和集群服务器进行互联，也可以连接外部网络（比如 WAN、VPN、互联网）。设计 InfiniBand 的目的主要是用于企业数据中心，大型的或小型的。目标主要是实现高的可靠性、可用性、可扩展性和高的性能。InfiniBand 可以在相对短的距离内提供高带宽、低延迟的传输，而且在单个或多个互联网络中支持冗余的 I/O 通道，因此能保持数据中心在局部故障时仍能运转。

4.2.1. InfiniBand 基本组件



InfiniBand 的网络拓扑结构如上所示，其组成单元主要分为四类：

1) HCA（Host Channel Adapter），它是连接内存控制器和 TCA 的桥梁

2) TCA(Target Channel Adapter)，它将 I/O 设备（例如网卡、SCSI 控制器）的数字信号打包发送给 HCA

3) InfiniBand link，它是连接 HCA 和 TCA 的光纤，InfiniBand 架构允许硬件厂家以 1 条、4 条、12 条光纤 3 种方式连结 TCA 和 HCA

4) 交换机和路由器

无论是 HCA 还是 TCA，其实质都是一个主机适配器，它是一个具备一定保护功能的可编程 DMA（Direct Memory Access，直接内存存取 ）引擎。

4.2.2. InfiniBand 应用

在高并发和高性能计算应用场景中，当客户对带宽和时延都有较高的要求时，可以采用 IB 组网：前端和后端网络均采用 IB 组网，或前端网络采用 10Gb 以太网，后端网络采用 IB。由于 IB 具有高带宽、低延时、高可靠以及满足集群无限扩展能力的特点，并采用 RDMA 技术和专用协议卸载引擎，所以能为存储客户提供足够的带宽和更低的响应时延。

4.2.3. IB 工作模式

1) SRD (Single Data Rate)：单倍数据率，即 8Gb/s
2) DDR (Double Data Rate)：双倍数据率，即 16Gb/s
3) QDR (Quad Data Rate)：四倍数据率，即 32Gb/s
4) FDR (Fourteen Data Rate)：十四倍数据率，56Gb/s
5) EDR (Enhanced Data Rate)：100 Gb/s
6) HDR (High Data Rate)：200 Gb/s
7) NDR (Next Data Rate)：1000 Gb/s+

4.2.4. Infiniband 的特点

1) 高带宽
2) 低时延
3) 系统扩展性好

4.2.5. IB 通信协议

1) Infiniband 与 RDMA

Infiniband 发展的初衷是把服务器中的总线给网络化。所以 Infiniband 除了具有很强的网络性能以外还直接继承了总线的高带宽和低时延。大家熟知的在总线技术中采用的 DMA(Direct Memory Access)技术在 Infiniband 中以 RDMA(Remote Direct Memory Access)的形式得到了继承。这也使 Infiniband 在与 CPU、内存及存储设备的交流方面天然地优于万兆以太网以及 Fibre Channel。可以想象在用 Infiniband 构筑的服务器和存储器网络中任意一个服务器上的 CPU 可以轻松地通过 RDMA 去高速搬动其他服务器中的内存或存储器中的数据块，而这是 Fibre Channel 和万兆以太网所不可能做到的。

2) Infiniband 与其他协议的关系

作为总线的网络化，Infiniband 有责任将其他进入服务器的协议在 Infiniband 的层面上整合并送入服务器。基于这个目的,今天 Volatire 已经开发了 IP 到 Infiniband 的路由器以及 Fibre Channel 到 Infiniband 的路由器。这样一来客观上就使得几乎所有的网络协议都可以通过 Infiniband 网络整合到服务器中去。这包括 Fibre Channel, IP/GbE, NAS, iSCSI 等等。另外 2007 年下半年 Voltaire 将推出万兆以太网到 Infiniband 的路由器。这里有一个插曲：万兆以太网在其开发过程中考虑过多种线缆形式。最后发现只有 Infiniband 的线缆和光纤可以满足其要求。最后万兆以太网开发阵营直接采用了 Infiniband 线缆作为其物理连接层。

InfiniBand 网络性能可以使用 IMB 测试程序进行测试，见第一章中的介绍，IB 通信协议使用方法见第六章中的 MPI 介绍的章节。

5. 集群管理软件

5.1. 集群管理系统

5.1.1. 集群系统的特点

1) 高性价比
2) 资源共享
3) 灵活性和可扩展性
4) 实用性和容错性
5) 可伸缩性

5.1.2. 集群管理系统的主要功能

目前，几大主流服务器厂商都提供了自己的集群管理系统，如浪潮的 Cluster Engine，曙光的 Gridview，HP 的 Serviceguard，IBM Platform Cluster Manager 等等。集群管理系统主要提供一项的功能：

1) 监控模块：监控集群中的节点、网络、文件、电源等资源的运行状态。动态信息、实况信息、历史信息、节点监控。可以监控整个集群的运行状态及各个参数。

2) 用户管理模块：管理系统的用户组以及用户，可以对用户组以及用户进行查看，添加，删除和编辑等操作。

3) 网络管理模块：系统中的网络的管理。

4) 文件管理：管理节点的文件，可以对文件进行上传、新建、打开、复制、粘贴、重命名、打包、删除和下载等操作。

5) 电源管理模块：系统的自动和关闭等。

6) 作业提交和管理模块：提交新作业，查看系统中的作业状态，并可以对作业进行执行和删除等操作。还可以查看作业的执行日志。

7) 友好的图形交互界面：现在的集群管理系统都提供了图形交互界面，可以更方便的使用和管理集群。

5.2. 集群作业调度系统

集群管理系统中最主要的模块为作业调度系统，目前，主流的作业调度系统都是基于 PBS 的实现。

PBS(Portable Batch System)最初由 NASA 的 Ames 研究中心开发，主要为了提供一个能满足异构计算网络需要的软件包，用于灵活的批处理，特别是满足高性能计算的需要，如集群系统、超级计算机和大规模并行系统。PBS 的主要特点有：代码开放，免费获取；支持批处理、交互式作业和串行、多种并行作业，如 MPI、PVM、HPF、MPL；PBS 是功能最为齐全，历史最悠久，支持最广泛的本地集群调度器之一. PBS 的目前包括 openPBS, PBS Pro 和 Torque 三个主要分支. 其中 OpenPBS 是最早的 PBS 系统，目前已经没有太多后续开发，PBS pro 是 PBS 的商业版本，功能最为丰富. Torque 是 Clustering 公司接过了 OpenPBS，并给与后续支持的一个开源版本。

PBS 主要有如下特征：

1) 易用性：为所有的资源提供统一的接口，易于配置以满足不同系统的需求， 灵活的作业调度器允许不同系统采用自己的调度策略。

2) 移植性：符合 POSIX 1003.2 标准，可以用于 shell 和批处理等各种环境。

3) 适配性：可以适配与各种管理策略，并提供可扩展的认证和安全模型。支 持广域网上的负载的动态分发和建立在多个物理位置不同的实体上的虚 拟组织。

4) 灵活性：支持交互和批处理作业。

torque PBS 提供对批处理作业和分散的计算节点(Compute nodes)的控制。

5.2.1. Torque 安装

下载地址：

http://www.adaptivecomputing.com/support/download-center/torque-download/

在 master（管理结点上）

1) 解压安装包

[root@master tmp]# tar zxvf torque-2.3.0.tar.gz

2) 进入到解压后的文件夹

./configure --with-default-server=master make

make install

3)

（1）[root@master torque-2.3.0]#./torque.setup <user>

<user>必须是个普通用户

（2）[root@master torque-2.3.0]#make packages

把 产 生 的 tpackages , torque-package-clients-linux-x86-64.sh, torque-package-mom-linux-x86-64.sh 拷贝到所有节点。

（3）[root@mastertorque-2.3.0]# ./torque-package-clients-linux-x86_64.sh --install

[root@master torque-2.3.0]# ./torque-package-mom-linux-x86_64.sh --install

（4）编辑/var/spool/torque/server_priv/nodes （需要自己建立）

加入如下内容

master np=4

node01 np=4

.....

node09 np=4

（5）启动 pbs_server,pbs_sched,pbs_mom,并把其写到/etc/rc.local 里使 其能开机自启动。

（6）创建队列

[root@master ~]# qmgr create queue students

set queue students queue_type = Execution set queue students Priority = 40

set queue students resources_max.cput = 96:00:00 set queue students resources_min.cput = 00:00:01

set queue students resources_default.cput = 96:00:00 set queue students enabled = True

set queue students started = True

4) 在 node0x （x=1-9,计算结点上）

[root@node0xtorque-2.3.0]# ./torque-package-clients-linux-x86_64.sh --install

[root@node0x torque-2.3.0]# ./torque-package-mom-linux-x86_64.sh --install

然后启动 pbs_mom ,把 pbs_mom 写入/etc/rc.local

5.2.2. Torque PBS 使用

1) 创建用户

在 master 的 root 下

useradd test passwd test 输入 test 密码

到/var/yp 下 make 一下

2) 配置普通用户的 ssh su test

cd

ssh-keygen -t dsa cd .ssh

cat id_pub.dsa >>authorized_keys chmod 600 authorized_keys

3) 编写作业脚本

[test1@master t]vi pbsjob

#!/bin/tcsh

#PBS -o /home/test1/pbstest/t/output 标准输出文件

#PBS -e /home/test1/pbstest/t/error 错误输出文件

#PBS -l nodes=5:ppn=4 规定使用的节点数以及每个节点能跑多少核

#PBS －q students 把任务提交到 students 队列中

cd $PBS_O_WORKDIR 到工作目录下（此为 PBS 提供的环境变量）

mpirun –machine $PBS_NODEFILE -np 20 ./vasp

4) 启动 mpd

mpdboot -n 10 -f mfa

mfa 内容：

master:4

node01:4

….

node09:4

5) 提交，查询，删除作业 提交作业

qsub pbsjob 作业提交后会有一个作业号

[test1@master pbstest]$ qsub pbsjob 48.master

查询作业：qstat

[test1@master pbstest]$ qstat

Job id Name User Time Use S Queue

------------------------ ---------------- ---------------

-------- - --------------------------------

48.master pbstest test1 00:00:00 R students

删除作业：qdel 作业号

[test1@master pbstest]$ qdel 48

5.2.3. PBS 脚本示例

qsub -N Relax -l nodes=1:ppn=8 pbs

#!/bin/sh

VASP="/home/user15/soft/mpi/bin/mpirun -machinefile $PBS_NODEFILE

-np 8 avasp < /dev/null " i=36 times=1000 while((i<=times)) do

cp RStru_$i POSCAR

rm WAVECAR CHG*

./produKPTS.x

$VASP

cp CONTCAR POSCAR

rm WAVECAR CHG*

./produKPTS.x

$VASP

cp CONTCAR POSCAR

rm WAVECAR CHG*

./produKPTS.x
$VASP
cp CONTCAR pos.$i cp OUTCAR out.$i let i=i+1
done
cd /temp/user15/RST1000
./relax.sh >& log

5.2.4. pbs 常用命令和选项

1) 基本选项

pbs 是 Protable Batch System 的缩写，是一个任务管理系统。当多个用户使用 同一个计算资源时，每个用户用 PBS 脚本提交自己的任务，由 PBS 对这些任务进 行管理和资源的分配。下面是一个简单的 PBS 脚本：

#PBS -l nodes=20
#PBS -N snaphu
#PBS -j oe
#PBS -l walltime=24:00:00
#PBS -l cput=1:00:00
#PBS -q dque
cd $PBS_O_WORKDIR
cat $PBS_NODEFILE $PBS_NODEFILE> NODEFILE
mpirun -hostfile NODEFILE -np `cat NODEFILE |wc -l` ./mpiTest

将这个脚本保存成 submit

然后 qsub submit 就将这个 mpiTest 的任务提交给了系统。 脚本中#PBS 为脚本选项，用于设置一些参数。

#PBS -l 表示资源列表，用于设定特定任务所需的一些参数。这里的 NODES 表示 并行环境下可以使用的节点数，而 walltime 表示任务最大时限，而 cput 表示 cpu 时间的最大时限，运行时间和 cpu 使用时间超过对应的时限，任务就会以超

时退出。这三个参数不是 PBS 脚本参数，而是并行环境所需的参数。

#PBS -N 表示任务名称

#PBS -j 表示系统输出，如果是 oe，则标准错误输出(stderr)和标准输出(stdout)

合并为 stdout，如果是 eo，则合并为 stderr，如果没有设定或设定为 n，则 stderr

和 stdout 分开。

#PBS -q 表示当前任务选用的队列。在并行环境下，一个系统中往往有多个队列， 任务提交后，将在所选的队列中排除等候。系统中有哪些队列可以用 qstat -q 查看。

2) 简单命令 任务提交后，需要查看任务信息和环境信息，有如下常用命令。

qstat 查看本用户提交的任务

qstat -n 同上，输出内容稍有不同

qstat -q 查看系统中所有的队列，以及每个队列中任务的运行和等候情况。

showq 查看系统中所有运行的任务。

qdel id 删除 JOBNAME 为 id 的任务。该任务如果在等待，则可以有这个命令删除，如果已经开始运行，则无法删除。

3) 参数传递

qsub submit -l nodes=4 -v x=1,y=2

其中，-l nodes=4 本来就是一个#PBS 选项，既可以放在 submit 文件中，又可以 放到命令行上。

-v x=1,y=2 为一个变量列表，和 shell 命令一样，在 submit 文件中可以用$x,$y 来调用这两值

6. 并行环境与并行开发

6.1. 编译器

6.1.1. GNU 编译器

GCC（GNU Compiler Collection，GNU 编译器套件），是一套由 GNU 开收的编程诧言编译器。它是一套以 GPL 及 LGPL 许可证所收行的自由软件，也是 GNU 计划的关键部分，亦是自由的类 Unix 及苹果电脑 Mac OS X 操作系统的标准编译器。GCC（特别是其中的 C 诧言编译器）也常被认为是跨平台编译器的事实标准。

GCC 可处理 C、C++、Fortran、Pascal、Objective-C、Java，以及 Ada 与其他语言。

编程语言	编译器调用名称
C	gcc
C++	g++
Fortran77	gfortran
Fortran90/95	gfortran

GNU 编译器支持 OpenMP、Pthread 等并行程序的编译。

6.1.2. Intel 编译器

Intel 编译器是 Intel 公司收布的一款 x86 平台（i386/x86_64）编译器产品，支持 C/C++/Fortran 编程语言。

Intel 编译器针对 Intel 处理器迪行了与门优化，性能优异，在 AMD 处理器平台上表现同样出色。

编程语言	编译器调用名称
C	icc
C++	icpc
Fortran77	ifort
Fortran90/95	ifort

Intel 编译器收费，每次注册可以获得一个月的试用期。下载地址：https://software.intel.com/en-us/intel-parallel-studio-xe

Intel 编译器支持 OpenMP、TBB、MPI 等并行程序的编译。

6.1.3. PGI 编译器

PGI 编译器是 The Portland Group 推出的一款编译器产品，支持 C、C++和 Fortran，在 AMD 处理器平台上性能较好。此外，PGI 编译器还提供对 HPF（High Performance Fortran，Fortran90 的扩展）编程语言的支持。

编程语言	编译器调用名称
C	pgcc
C++	pgCC
Fortran77	pgf77
Fortran90/95	pgf90/pgf95
HPF	pghpf

PGI 编译器已被 NVIDIA 收购，也是收费版本。PGI 编译器支持 Fortran 版本的 CUDA 程序。

6.1.4. 其它编译器

Open64（C、C++、Fortran77/90/95）
PathScale（C、C++、Fortran77/90/95）
AbsoftFortran Compiler
g95 Fortran Compiler
LaheyFortran Compiler

6.1.5. 编译优化选项

1) 常用的编译优化选项（以 GNU 编译器为例）
   -O1、-O2、-O3，丌同优化级别，一般情冴下选项-O2 比较合理
   -Os，针对减小执行文件体积优化
   -funroll-loops，循环展开优化
   -march=native、-msse3 等，针对 CPU 指令集优化
2) Intel 编译器推荐选项
   -O2-ip-xhost-funroll-loops –no-prec-div
3) PGI 编译器推荐编译选项
   -O2 -fastsse–Munroll-tpk8-64

6.1.6. NVCC 编译器

nvcc 是 NVIDIA 提供的针对 CUDA 程序的编译器，可以编译 CUDA 程序，使 CUDA 程序运行到 GPU 平台上。安装完 CUDA 工具包之后即可获得 nvcc 编译器，CUDA 安装包下载地址：https://developer.nvidia.com/cuda-toolkit-archive

6.2. 并行编程模型

6.2.1. OpenMP

OpenMp 是由 OpenMP Architecture Review Board 牵头提出的，并已被广泛接受的，用于共享内存并行系统的多处理器程序设计的一套指导性的编译处理方案(Compiler Directive)。OpenMP 支持的编程语言包括 C 语言、C++和 Fortran；而支持 OpenMp 的编译器包括 Sun Compiler，GNU Compiler 和 Intel Compiler 等。OpenMp 提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的 pragma 来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。当选择忽略这些 pragma，或者编译器不支持 OpenMP 时，程序又可退化为通常的程序(一般为串行)，代码仍然可以正常运作，只是不能利用多线程来加速程序执行。

OpenMP 是共享内存模型上的并行编程语言，主要用于实现节点内的线程并行，从而发挥多核的性能。

6.2.2. Pthread

POSIX 线程（POSIX threads），简称 Pthreads，是线程的 POSIX 标准。该标准定义了创建和操纵线程的一整套 API。在类 Unix 操作系统（Unix、Linux、Mac OS X 等）中，都使用 Pthreads 作为操作系统的线程。Windows 操作系统也有其移植版 pthreads-win32。

Linux 系统下的多线程遵循 POSIX 线程接口，称为 pthread。编写 Linux 下的多线程程序，需要使用头文件 pthread.h，连接时需要使用库 libpthread.a。与 vxworks 上任务的概念类似，都是调度的最小单元，都有共享的堆、栈、代码区、全局变量等。

pthread 全称应该是 POSIX THREAD，顾名思义这个肯定是按照 POSIX 对线程的标准而设计的。目前我所知道的有两个版本：Linux Thread（较早）和 NPTL（主流？）。pthread 库是一套关于线程的 API，提供“遵循”（各平台实现各异）POSIX 标准的线程相关的功能。

OpenMP 不同于 pthread 的地方是，它是根植于编译器的（也要包含头文件 omp.h），而不是在各系统平台是做文章。它貌似更偏向于将原来串行化的程序，通过加入一些适当的编译器指令（compiler directive）变成并行执行，从而提高代码运行的速率。

Pthread 和 OpenMP 都是基于共享内存模型上的多线程并行，区别在于 Pthread 更适合任务级并行，OpenMP 更适合数据级并行。

6.2.3. CUDA

CUDA(Compute Unified Device Architecture)，是显卡厂商 NVIDIA 推出的运算平台。 CUDA™是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题。 它包含了 CUDA 指令集架构（ISA）以及 GPU 内部的并行计算引擎。 开发人员现在可以使用 C 语言来为 CUDA™架构编写程序，C 语言是应用最广泛的一种高级编程语言。所编写出的程序于是就可以在支持 CUDA™的处理器上以超高性能运行。CUDA 3.0 已经开始支持 C++和 FORTRAN，目前，最新版本为 7.5，CUDA 各版本下载地址：https://developer.nvidia.com/cuda-toolkit-archive。CUDA 书籍也比较多，推荐：

《CUDA 并行程序设计：GPU 编程指南》

《CUDA 专家手册：GPU 编程权威指南》

6.2.4. OpenCL

OpenCL（全称 Open Computing Language，开放运算语言）是第一个面向异构系统通用目的并行编程的开放式、免费标准，也是一个统一的编程环境，便于软件开发人员为高性能计算服务器、桌面计算系统、手持设备编写高效轻便的代码，而且广泛适用于多核心处理器(CPU)、图形处理器(GPU)、Cell 类型架构以及数字信号处理器(DSP)等其他并行处理器，在游戏、娱乐、科研、医疗等各种领域都有广阔的发展前景。

CUDA 和 OpenCL 中，前者是配备完整工具包、针对单一供应商(NVIDIA)的成熟的开发平台，后者是一个开放的标准。

虽然两者抱着相同的目标：通用并行计算。但是 CUDA 仅仅能够在 NVIDIA 的 GPU 硬件上运行，而 OpenCL 的目标是面向任何一种 Massively Parallel Processor，期望能够对不同种类的硬件给出一个相同的编程模型。由于这一根本区别，二者在很多方面都存在不同：

1) 开发者友好程度。CUDA 在这方面显然受更多开发者青睐。原因在于其统一的开发套件(CUDAToolkit, NVIDIA GPU Computing SDK 以及 NSight 等等)、非常丰富的库(cuFFT,cuBLAS, cuSPARSE, cuRAND, NPP, Thrust)以及 NVCC(NVIDIA 的 CUDA 编译器)所具备的 PTX(一种 SSA 中间表示，为不同的 NVIDIA GPU 设备提供一套统一的静态 ISA)代码生成、离线编译等更成熟的编译器特性。相比之下，使用 OpenCL 进行开发，只有 AMD 对 OpenCL 的驱动相对成熟。

2) 跨平台性和通用性。这一点上 OpenCL 占有很大优势（这也是很多 National Laboratory 使用 OpenCL 进行科学计算的最主要原因）。OpenCL 支持包括 ATI,NVIDIA,Intel,ARM 在内的多类处理器，并能支持运行在 CPU 的并行代码，同时还独有 Task-Parallel Execution Mode，能够更好的支持 Heterogeneous Computing。这一点是仅仅支持数据级并行并仅能在 NVIDIA 众核处理器上运行的 CUDA 无法做到的。

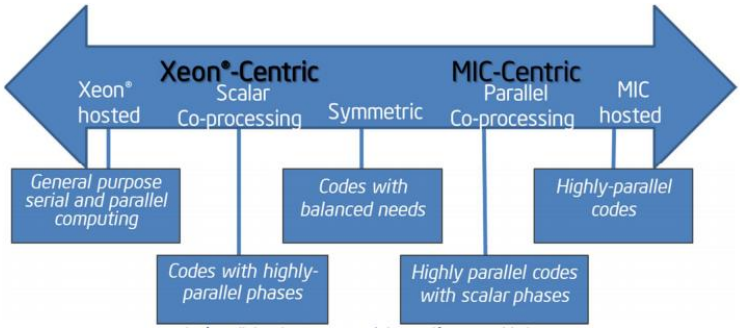
3) 市场占有率。作为一个开放标准，缺少背后公司的推动，OpenCL 显然没有占据通用并行计算的主流市场。NVIDIA 则凭借 CUDA 在科学计算、生物、金融等领域的推广牢牢把握着主流市场。

4) 从编程角度。CUDA 更符合 C 语言的格式，OpenCL 更像 C++的格式。

6.2.5. MIC 平台编程模型

2012 年底，Intel 公司正式推出了基于集成众核（Many Integrated Core, MIC）架构的至强融核（Intel® Xeon Phi™）系列产品，用于解决高度并行计算问题。第一代 MIC 正式产品为 KNC（Knights Corner），该产品双精度性能达到每秒一万亿次以上，它基于现有的 x86 架构，支持 OpenMP、pThread、MPI 等多种业内熟悉的并行编程模型，采用传统的 C/C++/Intel® Cilk™ Plus 和 Fortran 等语言进行软件开发，其特点以编程简单（引语方式）著称，具有丰富的工具链支持。对于采用传统 CPU 平台很难实现性能进一步提升的部分应用，使用 MIC 可以带来性能的大幅提升，并且 CPU 与 MIC 可以共用一份代码，在 x86 架构下实现 CPU+MIC 异构协同计算的完美结合，为广大高性能计算用户提供了全新的计算解决方案。

MIC 拥有极其灵活的编程方式，MIC 卡可以作为一个协处理器存在，也可以被看作是一个独立的节点。CPU+MIC 协同计算可以组合出如下图所示的 5 种应用模式。



6.2.6. MPI

消息传递接口(Message Passing Interface, MPI)，MPI 是一种基于消息传递的并行编程技术。

MPI 程序是基于消息传递的并行程序。消息传递指的是并行执行的各个进程具有自己独立的堆栈和代码段，作为互不相关的多个程序独立执行，进程之间的信息交互完全通过显示地调用通信函数来完成。

1) MPI 是一个库，而不是一门语言。

2) MPI 是一种标准或规范的代表，而不特指某一个对它的具体实现，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

3) MPI 是一种消息传递编程模型，并成为这种编程模型的代表。事实上，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的

常见 MPI 类型有以下几类：

1) OpenMPI

OpenMPI 是一种高性能消息传递库，最初是作为融合的技术和资源从其他几个项目（FT-MPI, LA-MPI, LAM/MPI, 以及 PACX-MPI），它是 MPI-2 标准的一个开源实现，由一些科研机构和企业一起开发和维护。因此，OpenMPI 能够从高性能社区中获得专业技术、工业技术和资源支持，来创建最好的 MPI 库。OpenMPI 提供给系统和软件供应商、程序开发者和研究人员很多便利。易于使用，并运行本身在各种各样的操作系统，网络互连，以及一批/调度系统。下载地址：http://www.open-mpi.org/

2) Intel MPI

Intel MPI 是 Intel 编译器中支持的 MPI 版本，下载地址见 6.1.2 节中的 Intel 编译器的介绍。

3) MPICH

MPICH 是 MPI 标准的一种最重要的实现，可以免费从网上下载。MPICH 的开发与 MPI 规范的制订是同步进行的，因此 MPICH 最能反映 MPI 的变化和发展。

MPICH 的开发主要是由 Argonne National Laboratory 和 Mississippi State University 共同完成的，在这一过程中 IBM 也做出了自己的贡献，但是 MPI 规范的标准化工作是由 MPI 论坛完成的。MPICH 是 MPI 最流行的非专利实现,由 Argonne 国家实验室和密西西比州立大学联合开发,具有更好的可移植性,现阶段多流行的是 MPICH2.最新版本为 3.2。相关下载地址：http://www.mpich.org/

4) Mvapich

MVAPICH 是 VAPI 层上 InfiniBand 的 MPI 的缩写，它充当着连接消息传递接口(MPI 和被称作 VAPI 的 InfiniBand 软件接口间的桥梁，也是 MPI 标准的一种实现。InfiniBand 是“无限

带宽”的缩写，这是一种新的支持高性能计算系统的网络架构。OSU(Ohio State University)超级计算机中心的科学家彼特于 2002 年开发出 MVAPICH 前，InfiniBand 和 MPI 是不兼容的。同时，MVAPICH2/MVAPICH 也可以从 Open Fabrics Enterprise Distribution (OFED) 套件中获得。最新版本为 MVAPICH2 2.2b，下载地址：http://mvapich.cse.ohio-state.edu/

不同 MPI 版本支持的通信协议类型

MPI 类型	通信协议	MPI 命令格式
Intel MPI	IB+shared memory	mpirun … -env I_MPI_DEVICE rdssm
	IB	mpirun … -env I_MPI_DEVICE rdma
	TCP	mpirun … -env I_MPI_DEVICE sock
	IPoIB	mpirun … -env I_MPI_DEVICE sock –env I_MPI_NETMASK ib
OpenMPI	IB+shared memory	mpirun… --mca btl self,sm,openib
	IB	mpirun … --mca btl self,openib
	TCP	mpirun … --mca btl self,tcp –mca btl_tcp_if_include eth0(若丌指定会使用以太网+IPoIB 的混合方式)
	IPoIB	mpirun … --mca btl self,tcp –mca btl_tcp_if_include ib0
MPICH2	TCP	修改 hosts 文件，各节点使用以太网 IP 地址戒对应 hostname
	IPoIB	修改 hosts 文件，各节点使用 IB 的 IP 地址戒对应 hostname
MVAPICH2	IB	mpirun_rsh-ssh –np # -hostfile hosts ./program

6.3. 数学库

6.3.1. Blas 库

基本线性代数库(Basic Linear Algebra Subroutines)，提供最基本的线性代数函数接口。BLAS 分为三级：BLAS 1（Level 1）向量不向量操作、BLAS 2（Level 2）：矩阵不向量操作、BLAS 3（Level 3）：矩阵不矩阵操作。

BLAS 库为高性能计算最为基础的库函数，BLAS 库的选择的好坏，甚至可以影响应用20%-50%的性能。

目前一般编译的过程中，使用优化的 blas 库，主要有如下几种。

1) MKL
2) ACML
3) Gotoblas
4) Atlas

6.3.2. FFTW 库

FFTW(the Faster Fourier Transform in the West)是一个快速计算离散傅里叶变换(DFT)的标

准 C 语言程序集，由麻省理工学院的 Frigo 和 Johnson 开发，可计算一维或多维数据以及任意规模的离散傅里叶变换。

6.3.3. CUDA 库

NVIDIA 提供了一些列的 CUDA 库，如下：

1) NVIDIA cuBLAS

NVIDIA CUDA BLAS (cuBLAS) 库是一个 GPU 加速版本的完整标准 BLAS 库，与最新的 MKL BLAS 相比，可实现 6 - 17 倍性能提升。

2) NVIDIA cuFFT

NVIDIA CUDA 快速傅里叶变换 (cuFFT) 库提供了一个简单的界面，能够以最高 10 倍的速度来计算 FFT，不必开发自己的定制 GPU FFT 程序。

3) CULA Tools

这是 EM Photonics 开发的一款 GPU 加速的线性代数 (LA) 库，可利用 CUDA 来大幅提升复杂数学的计算速度。

4) NVIDIA cuSPARSE

NVIDIA CUDA 稀疏 (cuSPARSE) 矩阵库提供了一系列用于系数矩阵的基本线性代数子例程，可实现 8 倍以上的性能提升。

5) AccelerEyes LibJacket

这是一款综合的 GPU 函数库，其中包括针对数学、信号与图像处理以及统计等方面的函数。 该库包含了针对 C、C++、Fortran 以及 Python 等语言的接口。

6) NVIDIA cuRAND

CUDA 随机数字生成 (RNG) 库可执行高质量的 GPU 加速随机数字生成任务，比仅使用 CPU 的一般代码快 8 倍以上。

7) NVIDIA CUDA Math library

这是一系列在业内经过实践检验的高精度标准数学函数，可在英伟达 GPU 上实现极高的性能。

8) NVIDIA NPP

NVIDIA 性能基元 (NPP) 是一款 GPU 加速的库，包含数以百计的一系列图像处理基元和信号处理基元。

9) Thrust

这是一款功能强大且开放源码的库，其中包含了诸多并行算法和数据结构。 只需区区几行代码即可执行 GPU 加速的排序、扫描、转换以及约减等操作。

10) GPU AI – 路径查找

这些库和样本应用程序展示了 CUDA 加速的路径规划，适用于机器人、视频游戏、合成环境以及人工智能等领域。

11) GPU AI 棋盘游戏

利用修剪和回溯法的 GPU 加速树状搜索。 主要专注于零和博弈以及其它领域中的应用程序，其中包括经济、环境、心理行为以及人工智能等领域。

来自：https://cudazone.nvidia.cn/gpu-accelerated-libraries/

6.4. 并行开发

6.4.1. GPU 与 MIC 对比

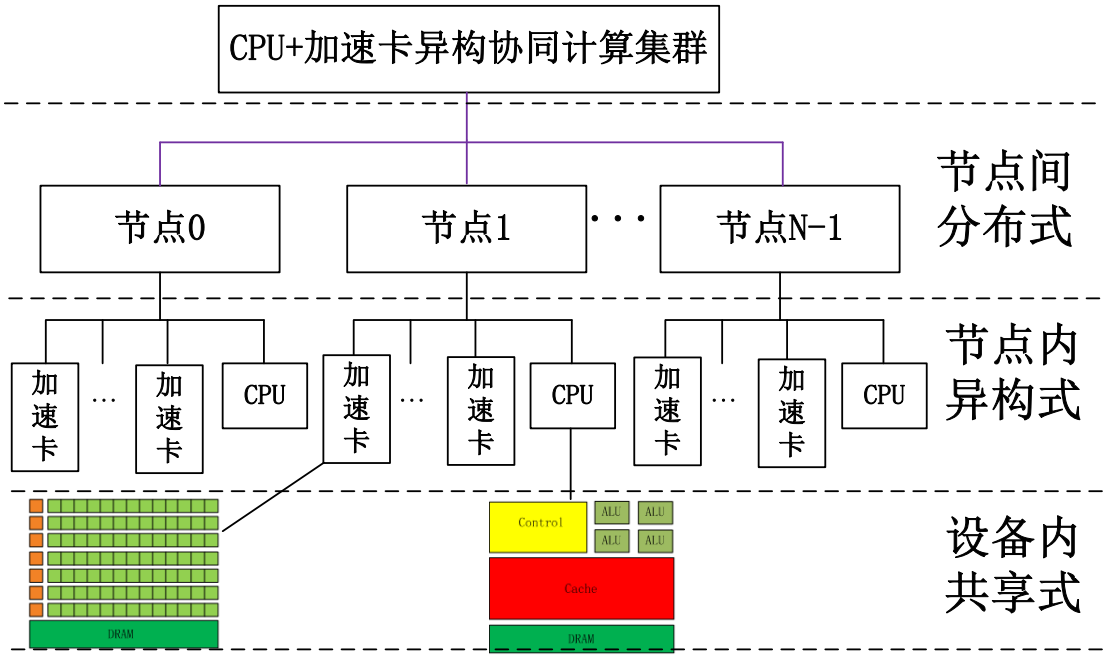
GPU 和 MIC 是目前主流的两种加速卡，在高性能计算中得到广泛应用。GPU 与 MIC 的对比如下表所示：

属性	NVIDIA GPU	Intel MIC
单核	流处理器/CUDA core 每个核运行一个线程	X86 core 每个核上最多支持 4 个硬件线程
主频	接近 1GHz	1.0-1.1GHz
核数	数十个到数千个	57-61
并行度	Grid、block、thread 多级并行 细粒度并行(线程数>>核数) 线程之间开销为 0	线程+向量化 线程数<=(核数-1)*4 向量化宽度 512bit（单精度：16，双精度：8）
内存大小（GB）	最大 24GB	6/8/16GB
内存带宽	480 GB/s	240-352GB/s
数据访问要求	Warp 内的线程访问的数据连续最佳	线程内访问的数据连续；如果向量化的话，向量化的数据访问连续最佳
峰值性能	单精度：最大 8.74TFlops 双精度：最大 5.6TFlops 计算方法：指令吞吐率*运算单元数量*频率	单精度：2.0-2.2 TFlops 双精度：1.0-1.1 TFlops Sample DP calculation: 16 DP FLOPs/clock/core * 61 cores * 1.1GHz = 1073.6 GFLOP/s
编程语言	CUDA、OpenCL、OpenACC	OpenMP、OpenCL、Cilk、OpenACC
编程模式	Offload	Offload、Native、Symmetric
功耗	235-300W	225-300W
PCI-E 带宽	支持 2.0（双向各 8GB/s） 支持 3.0（双向各 16GB/s）	支持 2.0（双向各 8GB/s） 目前不支持 3.0
运行平台	PC、服务器、工作站 个人可以在 PC 上配置一块 GeForce 卡运行 CUDA，成本低、性能高	服务器 比较专业，成本较高，个人很少配置
产品	GeForce：几百到几千元，用在 PC 上（当前主流 GTX710-780） Tesla：1W-3W 元，用在服务器上（当前主流 K20，K40，K80） Quadro：数千元，用在工作站上（当前主流 Quadro K4100M、Quadro K3100M、Quadro K2100M、Quadro K610M）	KNC：1W-2W 元左右 当前主流 7110P、5110P、3110P
支持的操作系统	Windows：XP、win7、win8 Linux X86：Fedora、OpenSUSE、RHEL/CentOS、SLES、SteamOS、Ubuntu 等 Linux ARM：Ubuntu	Windows：Windows 8 Server, Win 7, Win 8 Linux：RedHat6.0 及以上，SuSE SLES11 及以上

	Mac OSX	
卡上自带 OS	无	自带 uOS，有独立 IP

6.4.2. 异构计算多级并行

CPU+加速卡（GPU、MIC 或 FPGA）异构协同计算集群如下图所示，异构集群可以划分成三个并行层次：节点间并行、节点内 CPU 与加速卡异构并行、设备（CPU 或加速卡）内并行。根据这三个层次我们可以得到 CPU+加速卡异构协同计算模式为：节点间分布式+节点内异构式+设备内共享式。



根据前面对 CPU+加速卡异构协同计算模式的描述，我们可以得到 CPU+加速卡异构协同计算的编程模型（以 MPI 和 OpenMP 为例）如下表所示。

CPU+加速卡异构协同计算编程模型

	节点间分布式	节点内异构式	设备内共享式	
			CPU	加速卡
模式 1	MPI	OpenMP	OpenMP	CUDA/OpenCL/OpenMP
模式 2	MPI	MPI	OpenMP	CUDA/OpenCL/OpenMP
模式 3	MPI	MPI	MPI	CUDA/OpenCL/OpenMP

6.4.3. 负载均衡

负载是指多个任务之间的工作量分布情况，负载均衡是指各任务之间的工作量平均分配。负载均衡在并行计算里指的是将任务平均分配到并行执行系统中各个计算资源上，使之充分发挥计算能力，没有空闲或等待，也不存在负载过度。好的并行方法可以发挥好的负载均衡效果，负载不均衡将会导致计算效率的下降以及糟糕的扩展性。

常见的负载均衡设计方法有两种：静态负载均衡和动态负载均衡。静态负载均衡是程序员事先将工作区域分割成多个可并行的部分，并保证分割成的各个部分（工作量）能够均衡

地分布到各个处理器上运行，也就是说工作量在多个处理器之间均衡地进行分配，使并行程序的加速性能最高;动态负载均衡是在程序运行过程中进行任务的动态分配以达到负载平衡的目的，实际情况中存在着很多静态负载均衡解决不了的问题，比如，在一个循环中，每次循环的计算量均不同，且不能事先预知。一般来说，动态负载均衡的系统总体性能比静态负载均衡要好，但代码实现上更复杂。

6.4.3.1. 设备内的负载均衡

1) CPU 多核、MIC 众核设备内多线程负载均衡

多核间的负载均衡可以采用 OpenMP 中的三种负载均衡策略：

a) schedule(static [,chunk])：静态调度，线程每次获得 chunk 个迭代次数，并以轮询的方式进行。如果不指明 chunk，则以平均分配的方式进行，这是默认的调度方式。

b) schedule(dynamic [,chunk])：动态调度，动态地将迭代分配到各个线程，不使用 chunk 参数时将迭代逐个地分配到各个线程，使用 chunk 参数时，每次分配给线程的迭代次数为指定的 chunk 次。

c) schedule(guided [,chunk])：导引调度是一种采用指导性的启发式自调度方法。开始时每个线程分配到较大的迭代块，之后分配到的迭代块会逐渐递减。迭代块的大小会按指数下降到指定的 chunk 大小，如果没有指定 chunk 参数，那么迭代块大小最小降到 1。

OpenMP 的调度策略使用范围如下表所示。

调度算法	使用范围
Static	固定任务量，并且每次迭代任务量相同
Dynamic	任务量不固定，每次迭代任务量均不同
Guided	这是 dynamic 调度算法的特殊情况，导引调度算法可以减少调度的开销

2) GPU 内负载均衡

一块 GPU 内的负载均衡可以采用 CUDA/OpenCL 内核中的负载均衡方法，如 CUDA 中，基本上只有满足 warp 内的 32 个线程负载均衡即可。

6.4.3.2. 设备间和节点间的负载均衡

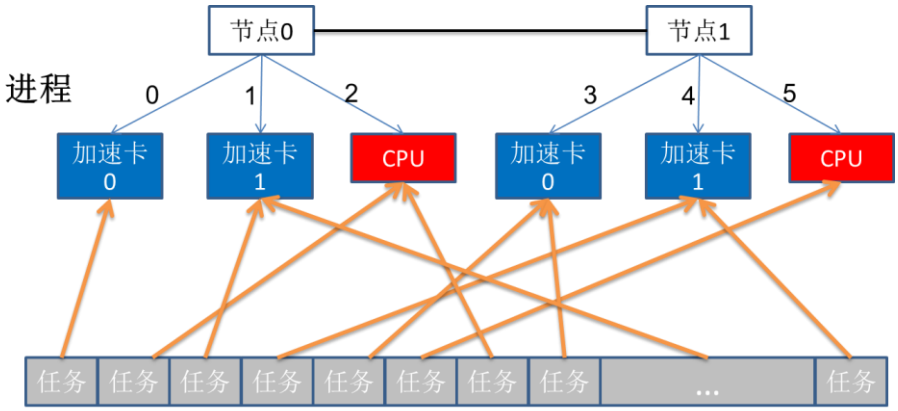
由于异构计算平台，CPU 和加速卡的计算能力不同，CPU 和加速卡上不能对等的划分计算量，另外由于每个应用程序的不用，CPU 和加速卡之间的计算量也不能通过 CPU 和加速卡的理论性能计算得到。因此，CPU 和加速卡之间最好的负载均衡方式为按任务进行动态划分。

节点间采用分布式计算，往往也是采用动态划分以达到负载均衡。

本节中重要介绍下动态负载均衡的方法。

1) 情景 1

假设每个设备的内存可以处理一个计算子任务，每个设备处理一个子任务，每个 MPI 进程控制一个节点上的 CPU 设备，或一个节点上的一块加速卡。动态划分如下图所示，所有计算进程向主进程请求任务，主进程动态的给所有计算进程分发任务。



主进程进行任务划分

利用 MPI 和 OpenMP 实现上面功能的源码如下：

```
void fun(int rank, int taskID, int chunk, int task)
{
	int end;
	end = taskID+chunk-1;
	if(end > task) end = task-1;
	printf("rank=%d,task=[%d-%d]\n",rank,taskID, end);
	sleep(1);
}

int main(int argc, char **argv) {
		int		i, j, rank, totalrank, taskID, task, tagID,chunk=1;
	MPI_Status status;
	if(argc>=2)
		task = atoi(argv[1]);
	else
		task = 100;
	if(argc>=3)
		chunk = atoi(argv[2]);

		MPI_Init(&argc, &argv);

		MPI_Comm_rank(MPI_COMM_WORLD, &rank);
		MPI_Comm_size(MPI_COMM_WORLD, &totalrank);

	if(rank==0)
	{
#pragma omp parallel private(taskID) num_threads(2)
#pragma omp sections
		{
#pragma omp	section
			{
```

```
                        for(taskID=0; taskID<task; taskID+=chunk)
                        {
                                MPI_Recv(&tagID, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
                                MPI_Send(&taskID, 1, MPI_INT, tagID, 0, MPI_COMM_WORLD);
                        }
                        for(i=0; i<totalrank; i++)
                        {
                                MPI_Recv(&tagID, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
                                MPI_Send(&taskID, 1, MPI_INT, tagID, 0, MPI_COMM_WORLD);
                        }
                }
#pragma omp     section
                {
                        MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                        MPI_Recv(&taskID, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
                        while(taskID < task)
                        {
                                //Calculate
                                fun(rank, taskID, chunk, task);
                                MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                                MPI_Recv(&taskID, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
                        }
                }
        }

    }
    else
    {
        MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Recv(&taskID, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        while(taskID < task)
        {
                //Calculate
                fun(rank, taskID, chunk, task);
                MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                MPI_Recv(&taskID, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        }
    }

        MPI_Finalize();
        return 0;
}
```

2) 情景 2

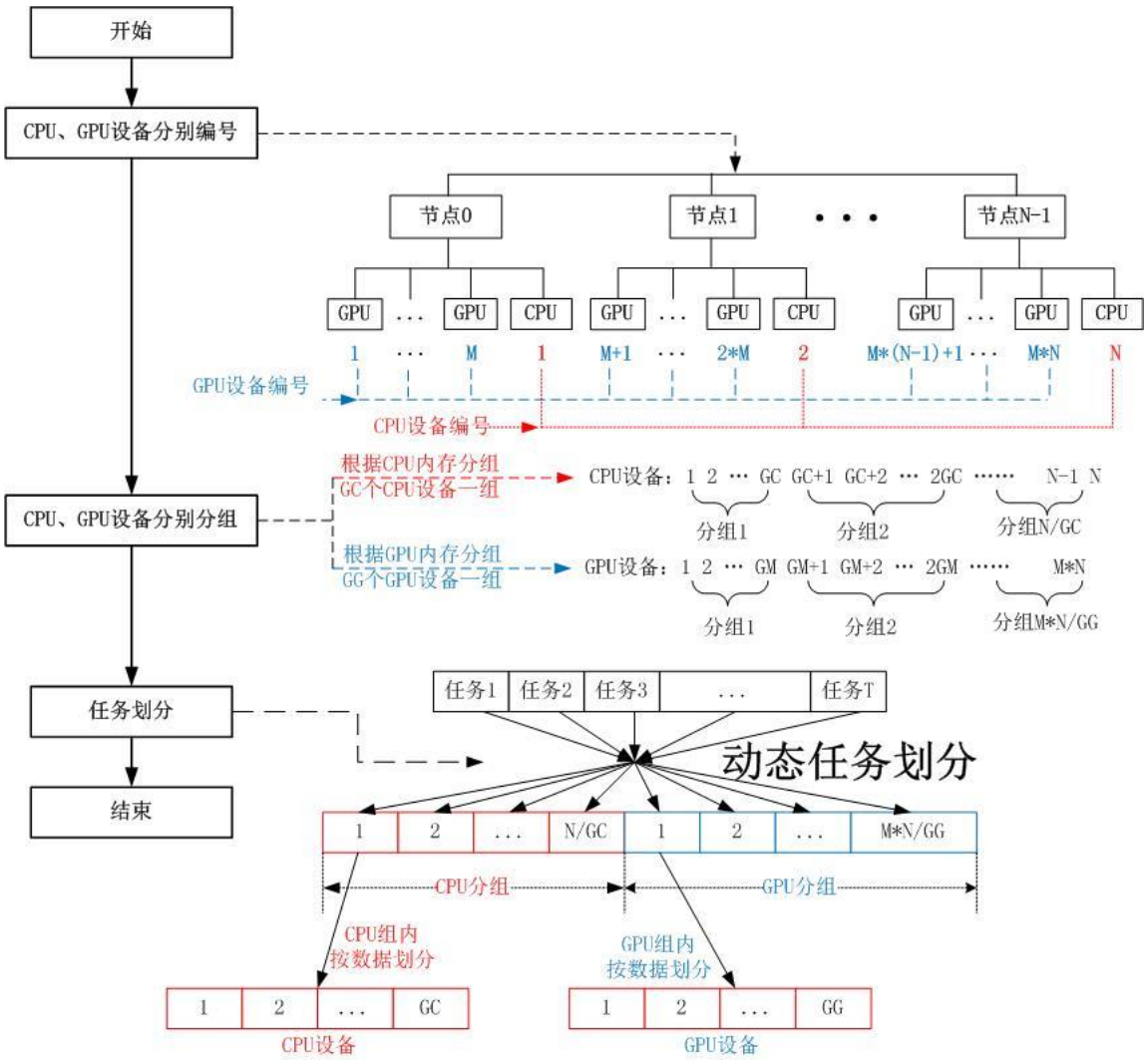
假设一个设备尤其是加速卡设备的内存无法满足一个子任务的计算需求，这时需要多块

设备共同计算一个子任务，多设备之间在按数据静态划分，然而，由于 CPU 和加速卡计算能力不同，最好的方式是 CPU 和加速卡不要划分到同一个子任务中，即 CPU 和加速卡处理不同的子任务。以 CPU+GPU 协调计算为例(CPU+其它加速卡协调计算情况类似)，描述如下：

在 CPU+GPU 协同计算中，CPU 和 GPU 的计算能力不同，静态地给 CPU 和 GPU 划分任务会导致 CPU 和 GPU 计算不同步，导致严重的负载不均衡，为了实现 CPU 和 GPU 的负载均衡，需要对 CPU 和 GPU 不同的划分方法。把每个节点上的所有 CPU 看成一个设备，每块 GPU 卡看成一个设备，对集群节点上的所有 CPU 设备和 GPU 设备分别编号，每个节点上有 M 块 GPU 卡，N 个节点上共有 N 个 CPU 设备，编号为 1,2，…，N；N 个节点上共有 M*N 个 GPU 设备，编号为 1,2，…，M*N。

设备编号之后就可以对设备进行分组，根据算法的要求，有些任务不能细分，一个 CPU 设备或 GPU 设备上可用的内存空间可能不能满足计算的内存要求，需要多设备数据划分、共同计算，这时需要根据 CPU 内存或 GPU 内存进行分组：1）根据 CPU 内存大小和计算对内存的要求可以计算得到一组内的 CPU 设备数目为 GC 个， GC=(Mcom+MemC-1)/MemC，其中一个节点的内存大小为 MemC，每个计算任务需要的内存大小为 Mcom。所有的 CPU 设备分为 N/GC 个组，每个组计算同一个任务，组内的 CPU 设备再进行数据划分，数据划分采用静态的划分，因为组内的 CPU 设备计算能力一致；2）根据 GPU 内存大小和计算对内存的要求可以计算得到一组内的 GPU 设备数目为 GG 个， GG=(Mcom+MemG-1)/Mem，其中，一个 GPU 设备的内存大小为 MemG，每个计算任务需要的内存大小为 Mcom。所有的 GPU 设备分为 M*N/GG 个组，每个组计算同一个任务，组内的 GPU 设备再进行数据划分，数据划分采用静态划分的方法。

CPU 和 GPU 设备分组之后，就可以把任务动态分划分给每个 CPU 组或 GPU 组，可以采用 MPI 通信进行划分，由主进程进行动态的发送任务编号给各个 CPU 组或 GPU 组内的组长，组长再把任务编号广播给组内的组员，然后组内的设备同时计算，计算完毕之后即可向主进程请求下一个任务，直到所有任务计算完毕为止。



用 MPI 实现上述功能是，需要对 MPI 进程建立子通信域，即：

a) 组内子通信域：组内进程进行任务广播，数据划分等。

b) 请求任务子通信域：每个组的主进程和总的主进程组成另外一个任务请求和相应的子通信域，来满足任务的动态划分。

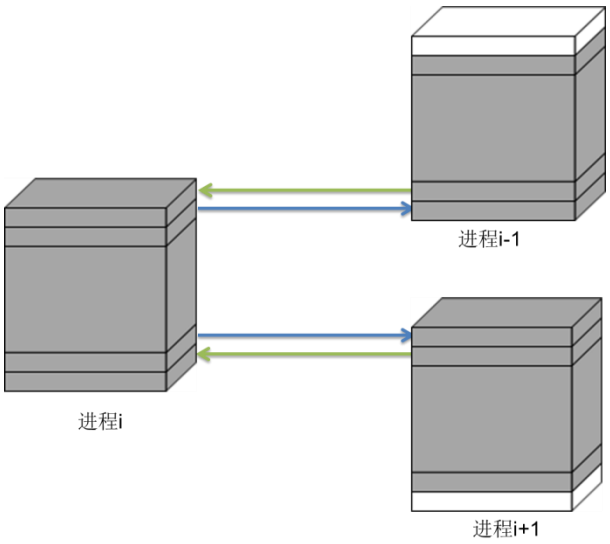
负载均衡是个很复杂的问题，此文只给出了两种动态负载均衡的方法，仅供参考。

6.4.4. 通信

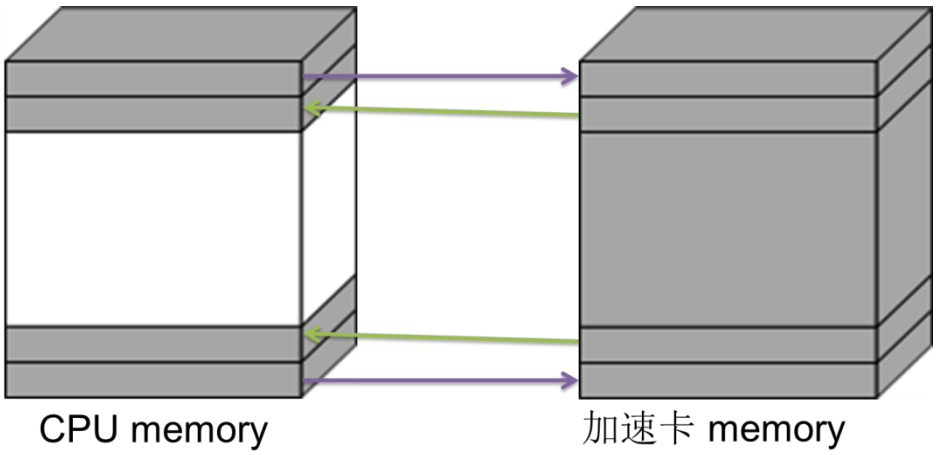
采用多卡多节点计算时，不同的卡或不同的节点之间必然存在通信问题，从优化角度一步采用局部通信和异步通信的方法对性能更优。

6.4.4.1. 局部通信

1) MPI 进程间局部通信，如下图所示。



2) CPU 与加速卡之间局部通信，如下图所示。



6.4.4.2. 异步通信

在高性能计算中，通过异步可以很好的隐藏 IO、通信问题，如：

计算、MPI 间的通信、CPU 与加速卡之间的通信、内存和硬盘之间的 IO，这些操作之间经常可以通过异步的方式来实现性能的提升。

6.4.4.3. 全局通信

在实际应用中，有时也无法避免全局通信，如 MPI 的广播，规约等操作。但如果可以通过修改算法可以减少全局通信，尽量减少全局通信，全局通信将会影响集群计算节点的扩展。

7. 容错

基于大规模数据的高性能计算任务执行时间都较长，在计算过程中发生故障的几率也会呈指数增长，一旦某计算节点发生硬件或软件异常，将导致运行程序的失败，程序不得不重头开始执行。为了避免系统故障后导致计算上的浪费，提供集群系统的可用性，系统可以在损失部分资源的情况下实现容错技术，即即使某个计算节点发生异常，其它计算节点已可以接替其计算任务，并继续进行整个程序的运行，而不需要从头计算。

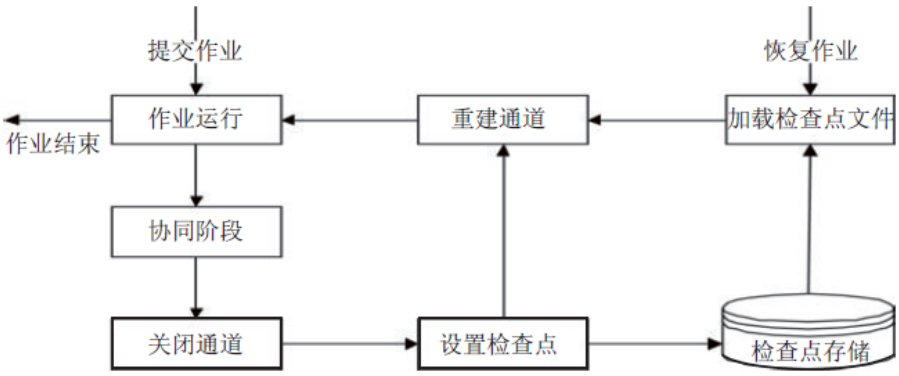
应用程序发生错误一般是由于计算节点异常导致，某节点异常会导致该节点上的 MPI 进程异常退出，从而导致整个程序的出错。系统应用程序的容错主要关心的是 MPI 进程的容错。

7.1. MPI 进程容错

随着检查点设置/回卷恢复技术得到了越来越广泛的重视，出现了一系列的检查点库，包括串行检查点库和并行检查点库。串行检查点库主要有 BLCR、Condor、libckpt、CRAK 等，并行检查点库主要有 L A M -MPI、MVAPICH2、Cocheck、NCCU MPI 等。并行检查点库通常都是基于串行检查点技术实现了并行检查点协议，而并行检查点协议又可分成检查点协议和日志协议。检查点协议包括协作式检查点、独立式检查点和通信引导式检查点。日志协议包括悲观日志、乐观日志和因果日志。

7.1.1. 系统级容错

MVAPICH2 借助伯克利实验室开发的开源检查点软件包 BLCR，实现了 MPI 应用的检查点设置/回卷恢复功能。BLCR 是一种内核级检查点库，作为内核模块加载在 Linux 系统内核，实现用户透明的检查点设置与恢复操作。BLCR 本身是单机检查点库，支持单个进程、进程组、会话、进程树的检查点设置与恢复操作，并提供检查点相关命令的用户调用。MVAPICH2 基于 BLCR 实现了并行检查点协议。



7.1.2. 应用级容错

MPI 并行编程容错机制，采用主从通信模式进行任务分配和管理，针对任务分割的情况，对每个引物文件建立一个进度状态文件，文件中存放该引物文件与每个子模板数据文件的匹

配计算任务的完成情况。程序每次启动时，主进程读取每个引物文件的进度文件，以确定哪些任务已完成，哪些任务仍待分配到各个从进程处理。程序执行过程中，当主进程每收到一个子进程发送的任务完成信息，主进程立刻更新相应引物文件的进度信息，并写入对应的进度文件。这样，所有任务是否完成的信息都会及时写到磁盘上，程序在任何时刻被中止后，再次启动时仍然可以获知哪些任务已经完成，而无需重新计算这些任务。在超大规模任务的计算中，不断的重复提交作业并进行续算，直至全部任务完成。