中山大学曙光 GPU 集群平台使用手册

中山大学网络与信息技术中心

广东省计算科学科研团队

2013 年 3 月 26 日

目　录

第一部分：中山大学高性能与网格计算平台用户守则

为加强中山大学网络与信息技术中心高性能与网格计算分中心（以下简称高性能与网格计算分中心）对我校高性能计算平台的管理和服务，引导用户合理、科学地使用计算资源，促进高性能计算平台在我校科研和教学工作中发挥更大的作用，特制订如下用户守则。

1. 根据集中建设、资源共享的原则，高性能与网格计算分中心负责我校高性能计算平台的建设运维，并提供相应的科研服务和技术支持。
2. 为促进学校科研创新能力提升，我校高性能计算平台免费向教师、研究人员或研究团队提供服务。用户在使用计算平台之前应先向高性能与网格计算分中心提出申请，获得批准后方能使用。
3. 高性能与网格计算分中心根据用户使用目的、研究内容及其应用情况等信息来分配用户可使用的计算资源数量及优先级。
4. 高性能与网格计算分中心将定期根据用户的研究进度和成果等信息调整用户使用计算资源的优先级。
5. 用户使用高性能计算资源，应严格遵守操作守则，一律通过统一的管理界面或在登录节点提交作业和获得结果，并在计算节点中进行计算。严禁使用非计算节点执行任何计算任务。
6. 用户应对自己的作业和数据的安全负责，定期备份或整理数据，防止发生数据损坏、丢失和泄密的问题。
7. 用户应自觉遵守国家有关保护知识产权的各项法律规定，不在高性能计算平台上擅自复制和使用未经授权的程序和文件，不擅自传播或拷贝他人享有版权的软件。
8. 用户应自觉遵守有关保守国家机密的各项法律规定，不利用高性能计算资源泄露国家机密或从事违法犯罪活动。
9. 用户不得擅自转让、出借帐号，不将口令随意告诉他人；也不借用他人帐户使用计算资源。
10. 用户不得使用软件的或硬件的方法窃取他人口令、非法入侵他人帐户，不阅读他人文件，不窃取他人计算和研究成果或受法律保护的资源。
11. 用户不得利用高性能计算机制造和传播计算机病毒；禁止破坏数据、破坏程序或其他恶作剧行为。
12. 未经许可，严禁使用学校高性能计算平台用于一切商业用途。
13. 用户应积极配合高性能与网格计算分中心进行校内高性能计算知识推广和普及，推动高性能计算的应用，加强与国内外高性能计算领域的联系与沟通，提高我校高性能计算及其它相关学科的水平和在国内外的影响。
14. 用户在使用学校高性能计算平台获取有用数据后，在发表科研论文、技术报告或其他形式的成果时，应以适当的方式表明该成果受到我校高性能计算平台的支持（具体表述方式请参照开户申请表相关信息），以扩大我校高性能计算平台的影响。
15. 增强自我保护意识，及时向高性能与网格计算分中心反映违反用户守则的人和事。
16. 对于用户违反该守则的行为，高性能与网格计算分中心将视其严重程度，给予一定的处理，包括警告、降低优先级、暂停使用、取消用户帐号等，并保留追究相关责任的权利。

第二部分：曙光 GPU 集群平台环境及使用简介

一、 集群系统简介与登陆

本集群为曙光 GPU 集群计算平台，由 GPU 计算节点、存储节点、登录节点、管理节点、千兆以太交换网络、高速通信交换网络（Infiniband）等部分组成。集群的整体理论峰值约为 130 万亿次。

其中 GPU 计算节点为 54 台曙光 W780 服务器，其中 35 台含 4 块 GPU 卡、19 台含 6 块 GPU 卡，每台计算节点均配备 2 颗 6 核 Intel 5650 CPU，72GB 内存；其他存储、登录、管理节点均为曙光 I620r-G 服务器，共 6 台。均配备 2 颗 4 核 Intel 5620 CPU，24GB 内存；整个集群配备存储 47TB；数据网配置 Infiniband 40Gb/s；管理网配置千兆以太网。

本集群基于 Rocks 5.4 构建，各节点操作系统为 CentOS 5.5（与 RedHat Linux 企业版 5.5 相应版本兼容）。

集群目前使用 SSH 登录，登录地址为：csl.sysu.edu.cn。

二、 系统默认运行环境

1、编译器 GCC v4.1.2，支持 gcc、g++、gfortran

2、编译器 Intel v11.1，支持 icc、icpc、ifort

3、并行环境是 Intel 编译器编译的 openmpi v1.4.3

说明：用户登录后，缺省使用系统默认运行环境，需要修改运行环境，可修改~/.bashrc 文件，添加相应环境的 source 调用命令，然后重新登录后 env 查看设置结果。恢复集群默认运行环境，把添加的 source 语句去掉后重新登录即可。

三、 编译系统

1、 默认 GNU 编译器：GCC v4.1.2：gcc、g++、gfortran

2、 可选 GNU 编译器：GCC v4.4.6：gcc、g++、gfortran
安装位置：/public/soft/gcc-4.4.6/
调用命令：source /public/scripts/gcc-4.4.6.sh

3、 Intel 编译器：Intel Compiler 11.1：icc、icpc、ifort
安装位置：/public/soft/intel/Compiler/11.1/073

4、 Open64 高性能编译器：Open64 v4.2.4：opencc、openCC、openf90/openf95
安装位置：/public/soft/open64-424/
调用命令：source /public/scripts/open64-424.sh

四、 并行环境

1、 MPICH 1.2.7 （GCC v4.1.2）
安装位置：/public/soft/mpich1.2.7-gnu/
调用命令：source /public/scripts/mpich1.2.7-gnu.sh
环境检查：mpicc -show

2、 MPICH 1.2.7 （Intel Compiler 11.1）
安装位置：/public/soft/mpich1.2.7-intel11.1/
调用命令：source /public/scripts/mpich1.2.7-intel.sh
环境检查：mpicc -show

3、 MPICH2 1.3.2p1 （Intel Compiler 11.1）
安装位置：/public/soft/mpich2-1.3.2p1-intel11.1/
调用命令：source /public/scripts/mpich2-1.3.2p1-intel11.1.sh
环境检查：mpicc -show

4、 OPENMPI 1.4.0 （GCC v4.1.2）
安装位置：/public/soft/ompi140-gnu/
调用命令：source /public/scripts/openmpi1.4-gnu.sh
环境检查：mpicc -show

5、 OPENMPI 1.4.0 （Intel Compiler 11.1）
安装位置：/public/soft/ompi140-intel11.1/
调用命令：source /public/scripts/openmpi1.4-intel.sh
环境检查：mpicc -show

6、 OPENMPI 1.4.3 （GCC v4.1.2）
安装位置：/public/soft/openmpi143-gnu/
调用命令：source /public/scripts/openmpi143-gnu.sh
环境检查：mpicc -show

7、 OPENMPI 1.4.3 （Intel Compiler 11.1）
安装位置：/public/soft/openmpi143-intel
调用命令：source /public/scripts/openmpi143-intel.sh
环境检查：mpicc -show

8、 MVAPICH 1.2rc1 （Intel Compiler 11.1）
安装位置：/public/soft/mvapich-1.2rc1-intel11.1/
调用命令：source /public/scripts/mvapich1.2rc1-intel.sh
环境检查：mpicc -show

9、 MVAPICH2 1.6 （Intel Compiler 11.1）
安装位置：/public/soft/mvapich2.1.6-intel11.1/
调用命令：source /public/scripts/mvapich2.1.6-intel.sh
环境检查：mpicc -show

10、MVAPICH2 1.6　(GCC v4.1.2)
安装位置：/public/soft/mvapich2.1.6-gnu/
调用命令：source /public/scripts/mvapich2.1.6-gnu.sh
环境检查：mpicc -show

五、　数学库

1、 Intel 编译器自带库：IPP、TBB、MKL、BLAS、LAPACK 和 ScaLPACK
安装位置：/public/soft/intel/Compiler/11.1/073

2、 BLAS、LAPACK 和 SCALPACK：
安装位置：/public/soft/mathlib/lib/

3、 FFTW2 2.1.5（GCC 4.1.2 编译，单精度）：
安装位置：/public/soft/mathlib/fftw215-float

4、 FFTW2 2.1.5（GCC 4.1.2 编译，双精度）：
安装位置：/public/soft/mathlib/fftw215-double

5、 FFTW3 3.2（GCC 4.1.2 编译，单精度）：
安装位置：/public/soft/mathlib/fftw32-float

6、 FFTW3 3.1.2（GCC 4.1.2 编译，双精度）：
安装位置：/public/soft/mathlib/fftw312-double

六、 GPU 编程环境

本 GPU 计算平台的计算节点 GPU 加速卡驱动使用 devdriver_4.0_linux_64_270.40，经过设备供应商曙光公司测试，可兼容 cuda3.2 程序。

1、 CUDA 编程环境：

系统默认环境为 CUDA4.0 编程环境。

如果需要使用 CUDA5.0.35 编程环境，执行：
source /public/scripts/cuda5.0.35.sh

如果需要使用 CUDA3.2 编程环境，执行：
source /public/scripts/cuda3.2.sh

如果返回 CUDA4.0 编程环境，可退出并重新登录集群，或执行：
source /public/scripts/cuda4.0.sh

a. CUDA3.2 编程环境
CUDA Toolkit 3.2.16 安装位置：/public/gpu/cuda3.2.16/cuda
CUDA3.2 头文件路径：/public/gpu/cuda3.2.16/cuda/include/
库路径：/public/gpu/cuda3.2.16/cuda/lib64/
CUDA3.2 编译命令例子：

```
nvcc main.cu -I/public/gpu/cuda3.2.16/cuda/include/   \
    -L/public/gpu/cuda3.2.16/cuda/lib64/ -o main
```

b. CUDA4.0 编程环境
CUDA Toolkit 4.0.17 安装位置：/public/gpu/cuda4.0.17/cuda
CUDA 4.0 头文件路径：/public/gpu/cuda4.0.17/cuda/include/
库路径：/public/gpu/cuda4.0.17/cuda/lib64/
CUDA4.0 编译命令例子：

```
nvcc main.cu -I/public/gpu/cuda4.0.17/cuda/include/   \
    -L/public/gpu/cuda4.0.17/cuda/lib64/ -o main
```

c. CUDA5.0 编程环境
CUDA Toolkit 5.0.35 安装位置：/public/gpu/cuda5.0.35
CUDA 5.0 头文件路径：/public/gpu/cuda5.0.35/include/
库路径：/public/gpu/cuda5.0.35/lib64/
CUDA5.0 编译命令例子：

```
nvcc main.cu -I/public/gpu/cuda5.0.35/include/   \
    -L/public/gpu/cuda5.0.35/lib64/ -o main
```

2、 OpenCL 编程环境：

计算节点 OpenCL 计算版本为 v1.0 规范，驱动库 lib 路径/usr/lib/libOpenCL.so.1.0.0，默认使用 GCC v4.1.2 编译器编译，头文件 include 路径/public/gpu/cuda4.0.17/cuda/include/。编译实例：

```
g++ main.c -I/public/gpu/cuda4.0.17/cuda/include/ -o main -lOpenCL
```

用户可调用 GCCv4.4.6：source /public/scripts/gcc-4.4.6.sh，来编译 OpenCL 程序。

a. OpenCL 3.2 的编程环境：
OpenCL3.2 头文件路径：/public/gpu/cuda3.2.16/cuda/include/CL/
OpenCL3.2 动态库路径：/usr/lib/libOpenCL.so.1.0.0
OpenCL3.2 编译例子：

```
g++ main.c -I/public/gpu/cuda3.2.16/cuda/include/ -o main -lOpenCL
```

b. OpenCL4.0 的编程环境：
OpenCL4.0 头文件路径：/public/gpu/cuda4.0.17/cuda/include/CL/
OpenCL4.0 动态库路径：/usr/lib/libOpenCL.so.1.0.0
OpenCL4.0 编译例子：

```
g++ main.c -I/public/gpu/cuda4.0.17/cuda/include/ -o main –lOpenCL
```

3、 GPU Computing SDK (CUDA 和 OpneCL 实例代码)

安装位置：/public/gpu/NVIDIA_GPU_Computing_SDK/
用户需把 NVIDIA_GPU_Computing_SDK/复制到用户目录下，执行编译：make

4、 Tesla C2050 高性能计算卡参数

a. CUDA 硬件参数

Device Tesla C2050 CUDA 4.0	
Device 0: "Tesla C2050"	
CUDA Driver Version / Runtime Version	4.0 / 4.0
CUDA Capability Major/Minor version number:	2.0
Total amount of global memory:	2687 MBytes (2817982464 bytes)
(14) Multiprocessors x (32) CUDA Cores/MP:	448 CUDA Cores
GPU Clock Speed:	1.15 GHz
Memory Clock rate:	1500.00 Mhz
Memory Bus Width:	384-bit
L2 Cache Size:	786432 bytes
Max Texture Dimension Size (x,y,z)	1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers	1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	49152 bytes
Total number of registers available per block:	32768
Warp size:	32
Maximum number of threads per block:	1024
Maximum sizes of each dimension of a block:	1024 x 1024 x 64
Maximum sizes of each dimension of a grid:	65535 x 65535 x 65535
Maximum memory pitch:	2147483647 bytes
Texture alignment:	512 bytes
Concurrent copy and execution:	Yes with 2 copy engine(s)
Run time limit on kernels:	No
Integrated GPU sharing Host Memory:	No
Support host page-locked memory mapping:	Yes
Concurrent kernel execution:	Yes
Alignment requirement for Surfaces:	Yes
Device has ECC support enabled:	Yes
Device is using TCC driver mode:	No
Device supports Unified Addressing (UVA):	Yes
Device PCI Bus ID / PCI location ID:	136 / 0

b. OpenCL 硬件参数

```
------------------------------------------------------------
  Device Tesla C2050 OpenCL 1.0
------------------------------------------------------------
    CL_DEVICE_NAME:                                 Tesla C2050
    CL_DEVICE_VENDOR:                               NVIDIA Corporation
    CL_DRIVER_VERSION:                              270.40
    CL_DEVICE_VERSION:                              OpenCL 1.0 CUDA
    CL_DEVICE_TYPE:                                 CL_DEVICE_TYPE_GPU
    CL_DEVICE_MAX_COMPUTE_UNITS:                    14
    CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:             3
    CL_DEVICE_MAX_WORK_ITEM_SIZES:                  1024 / 1024 / 64
    CL_DEVICE_MAX_WORK_GROUP_SIZE:                  1024
    CL_DEVICE_MAX_CLOCK_FREQUENCY:                  1147 MHz
    CL_DEVICE_ADDRESS_BITS:                         32
    CL_DEVICE_MAX_MEM_ALLOC_SIZE:                   671 MByte
    CL_DEVICE_GLOBAL_MEM_SIZE:                      2687 MByte
    CL_DEVICE_ERROR_CORRECTION_SUPPORT:             yes
    CL_DEVICE_LOCAL_MEM_TYPE:                       local
    CL_DEVICE_LOCAL_MEM_SIZE:                       48 KByte
    CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:             64 KByte
    CL_DEVICE_QUEUE_PROPERTIES:
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
    CL_DEVICE_QUEUE_PROPERTIES:                     CL_QUEUE_PROFILING_ENABLE
    CL_DEVICE_IMAGE_SUPPORT:                        1
    CL_DEVICE_MAX_READ_IMAGE_ARGS:                  128
    CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                 8
    CL_DEVICE_SINGLE_FP_CONFIG:                     denorms  INF-quietNaNs  round-to-nearest
round-to-zero round-to-inf fma

    CL_DEVICE_IMAGE <dim>                           2D_MAX_WIDTH        4096
                                                    2D_MAX_HEIGHT       32768
                                                    3D_MAX_WIDTH        2048
                                                    3D_MAX_HEIGHT       2048
                                                    3D_MAX_DEPTH        2048

    CL_DEVICE_EXTENSIONS:                           cl_khr_byte_addressable_store
                                                    cl_khr_icd
                                                    cl_khr_gl_sharing
                                                    cl_nv_compiler_options
                                                    cl_nv_device_attribute_query
                                                    cl_nv_pragma_unroll
                                                    cl_khr_global_int32_base_atomics
                                                    cl_khr_global_int32_extended_atomics
                                                    cl_khr_local_int32_base_atomics
```

```
                                cl_khr_local_int32_extended_atomics      cl_khr_fp64

CL_DEVICE_COMPUTE_CAPABILITY_NV:        2.0
NUMBER OF MULTIPROCESSORS:               14
NUMBER OF CUDA CORES:                    448
CL_DEVICE_REGISTERS_PER_BLOCK_NV:       32768
CL_DEVICE_WARP_SIZE_NV:                  32
CL_DEVICE_GPU_OVERLAP_NV:                CL_TRUE
CL_DEVICE_KERNEL_EXEC_TIMEOUT_NV:        CL_FALSE
CL_DEVICE_INTEGRATED_MEMORY_NV:          CL_FALSE
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>    CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 1,
 DOUBLE 1
```

七、 作业调度系统使用简介

使用本集群的所有用户，均应通过 Torque 作业调度系统提交任务，作业提交与管理方式，参照以下介绍的命令。

1、后台提交作业 qsub

语法: qsub [-a date_time] [-A account_string] [-e path] [-h] [-I] [-j join] [-k keep] [-l resource_list] [-m mail_options] [-n Node_allocation_Method [-L v1,[v2,[v3,[v4]]]]] [-M user_list] [-N name] [-o path] [-p priority] [-q pool] [-r y|n] [-u user_list] [-v variable_list] [-V] [script]

参数: script 参数被省略时，该命令可以从标准输入获得脚本文件名。

-a 间。格式为[[[[CC]YY]MM]DD]hhmm[.SS]。CC 表示世纪，YY 表示年（后两位数字），MM 表示月（两位数字），DD 表示天（两位数字），hh 表示小时（两位数字），mm 表示分（两位数字），SS 表示秒（两位数字）。如果 DD 指定的是未来日子，而未指定 MM，则 MM 缺省值为当前月，否则，MM 的缺省值为下个月。如果 hhmm 指定的未来时间，而未指定 DD,则 DD 的缺省值为当天，否则，DD 的缺省值为明天。如果提交作业时使用该选项，当指定时间还没到时，作业状态显示为”W”。

-e 指定错误输出文件名，格式为[hostname:]path_home。Hostname 是返回错误输出文件的主机名，path_home 是错误输出文件的绝对路径，如果指定了相对路径，则相对用户的主目录。不使用该选项时，缺省值是在用户主目录下，以“作业名.e 作业 ID”命名的文件

-o 指定输出文件名，格式为[hostname:]path_home。缺省值是在用户主目录下，以“作业名.e 作业 ID”命名的文件

-h 指定在提交作业时，设置用户级’u’挂起。如果不指定，则设置挂起类型为’n’，即不挂起。

-I 指定作业以交互方式运行。

-j 指定合并错误输出和实际输出。如果指定’oe’，则合并到标准输出文件中；如果指定’eo’，则合并到标准错误输出文件中.

-k 指定执行主机是否保留错误输出和实际输出。如果指定‘o’,则仅保留标准输出；如果指定’e’，则仅保留标准错误输出；如果指定’oe’或‘eo’,则保留标准输出和标准错误输出；如果指定’n’,则不保留任何输出。

-l 指定作业所需要的资源，设定对可消耗资源的限制。如果不设置，则无限制。例如： resource_name[=[value]][,resource_name[=[value]],…]

LINUX 系统可以设置的资源有 cput, file, pcput, pmem, pvmem, vmem, walltime, arch, nodes, ncpus 等；

Cput 指作业的所有进程使用 cpu 最长时间；

File 指作业可以建立单个文件大小的最大限制；

Pcput 指作业的单个进程可以使用 CPU 的最长时间；

vmem 指作业可以使用的物理内存的最大值；

Pmem 指作业的单个进程可以使用的物理内存的最大值；

Pvmem 指作业的单个进程可以使用的虚拟内存的最大值；

walltime 指作业处于运行状态的最长时间；

arch 指定系统管理员所定义的系统结构类型；

host 指定作业运行的主机名；

nodes 指定作业独占使用的结点数目和属性，使用“+”可以连接多种结点的定义。

结点的属性和主机名或数目之间通过“：”分隔。如果不指定结点数，则缺省为 1。结点的属性包括 ppn（每个结点上的进程数,缺省为 1）和系统管理员设置的属性（如 batch、bigmem）

例如：请求 12 个结点，不管其属性

```
-l nodes=12
```

请求 12 个属性为 batch 的结点和 14 个属性为 bigmem 的结点

```
-l nodes=12:batch+14:bigmem
```

请求 4 个结点，每个结点上使用 2 个 CPU

```
-l nodes=4:ppn=2
```

software 指作业要求的软件包

-m 定义何时给用户发送有关作业的邮件。可设定的选项有：

n 不发送邮件

a 当作业被批处理系统中断时，发送邮件

b 当作业开始执行时，发送邮件

e 当作业执行结束时，发送邮件

-M 指定发送有关作业信息的邮件用户列表。格式为 user[@host][,user@[host],…]缺省值为提交作业的用户。

-N 指定作业的名字。缺省值为脚本的名字，如果没有指定脚本，则为 STDIN。

-p 指定作业的优先级，优先级的范围是[-1024, +1023]。缺省值是没有优先级。

-q 指定作业的目的地(结点池)，目的地可有三种格式：

```
pool
@server
pool@server
```

-r y|n 指定作业是否可重新运行。指定‘y’时，作业可以重新运行；指定’n’时，作业不能重新运行。缺省值为’n’。

-v 格式为 variable1,variable2,…或 variable1=value,variable2=value,…这些变量和其值可以传递到作业中。

-V 指定 qsub 命令的所有的环境变量都传递到批处理作业中。

作用：以脚本文件的形式向批处理服务器提交作业。

举例：

运行 MPI 程序的脚本 cpi.pbs 如下：

```
#!/bin/sh
### Job name
#PBS -N test
```

```
### Declare job non-rerunable
#PBS -r n
### Output files
#PBS -e test.err
#PBS -o test.log
### Mail to user
#PBS -m ae
### pool name (small, medium, long, verylong)
#PBS -q dque
### Number of nodes (node property ev67 wanted)
#PBS -l nodes=8:ppn=8
# This job's working directory
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR
echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`
# Define number of processors
NSLOTS=`cat ${PBS_NODEFILE} | wc -l`
echo This job has allocated $NPROCS nodes
# Run the parallel MPI executable "a.out"
mpirun -v -hostfile $PBS_NODEFILE -np ${NSLOTS} cpi >& out.dat
```

1).把脚本文件 cpi.pbs 提交到结点池 dque 中运行。

```
$ qsub cpi.pbs
35.console
$ qstat
Job id Name User TimeUse S Pool
35.console cpi zhangxq 0:00:00 R dque
```

2).把脚本作业 cpi.pbs 提交到结点池 long 中运行，并且当作业开始运行时，给用户发送电子邮件。

```
$qsub -P long -m b cpi.pbs $qstat
Job id Name User TimeUse S Pool
36.console cpi zhangxq 0 R long
```

3).使用 15 个结点运行 cpi.pbs，运行时间不能大于 2 小时，作业占用的内存能大于 15mb

```
$qsub -l nodes=15,walltime=2:00:00,mem=15mb cpi.pbs
```

4).综合考虑 CPU、内存、缓冲区、硬盘等因素的使用情况，设置其权值分别为 5, 4, 3, 2.

```
$qsub -l nodes=5 -n syn -L 5, 4, 3, 2 cpi.ljrs
```

5).使用-I 和 -l 选项在指定的结点上交互式运行作业

```
$ qsub -I -l nodes=c0101
```

运行该命令后,系统将自动切换到 c0101 的虚拟终端,所有的标准输入和标准输出

和标准错误输出均都显示在该终端上. 在系统出现”waiting to jobid to start”时,用户若按^C键,即可中断该交互作业.当该交互作业已经启动后,用户除了交互式运行作业外,还可进行以下操作:

~. 该命令将使 qsub 退出，且终止作业。

~^Z 挂起 qsub,作业依然保持运行状态，无任何输入输出。用户可以在本地 shell 上提交命令。

~^Y 部分挂起 qsub，因为还可以给作业发送输入，而且接受作业的输出的同时，还可以在本地 shell 提交命令。

附录：PBS 作业调度系统中的变量

$PBS_O_WORKDIR：作业所在目录；

$PBS_NODEFILE：系统根据用户请求资源生成的节点列表；

$PBS_JOBID：用户提交作业后系统返回的作业 id 号；

有了以上变量，我们可以定义：

NP =`cat ${PBS_NODEFILE} | wc -l`：计算可用节点数。

2、查看状态 qstat

语法：qstat [-f][-W site_specific] [job_identifier... | destination...]

qstat -Q [-f][-W site_specific] [destination...]

qstat -B [-f] [-W site_specific] [server_name...]

参数：destination 可以为 pool，@server，pool@server

作用：查看作业、结点池和批处理服务器的状态。命令格式一可以输出所指定作业 ID 或者结点池中所有作业的状态，命令格式二可以输出每个结点池的状态信息，命令格式三可以输出服务器的状态。

举例：

1). 显示已经配置的所有结点池状态信息。

```
qstat -q
```

2). 显示已经提交的作业状态信息

```
qstat -a
```

3). 显示指定作业的所有状态信息

```
$ qstat -f 23.console
mtime = Sun Apr 28 19:54:48 2002
Output_Path = console:/home/zhangxq/cpi.o23
Priority = 0
qtime = Sun Apr 28 19:54:48 2002
Rerunable = True
Resource_List.cput = 00:00:59
Resource_List.nodect = 2
Resource_List.nodes = 2:ppn=1
Variable_List = LJRS_O_HOME=/home/zhangxq,LJRS_O_LANG=en_US
……
```

4). 显示服务器的状态

```
$qstat  -B
Server Max Tot Que Run Hld Wat Trn Ext Status
console 0 0 0 0 0 0 0 0 Active
```

3、查询作业 qselect

语法：qselect [-a [op]date_time] [-A account_string] [-h old_list] [-l resource_list] [-N name] [-p [op]priority] [-q destination] [-r y|n] [-s states] [-u user_list]

参数：op 表示某一个作业属性值和选项参数值之间的关系。如

.eq. （等于）

.ne. （不等于）

.ge. （大于或等于）

.gt. （大于）

.le. （小于）

其它参数含义见 qsub 命令。

作用：列出符合选项要求的作业 ID。这些作业来自于单个服务器。如果没有任何选项，该命令则列出该用户被授权的服务器上的所有作业。对那些普通用户来说，该命令只显示该用户所提交的作业。

举例：

1). 查询用户所提交的作业

```
$ qsub -q long cpi.ljrs
28.console
$ qselect
25.console
28.console
```

2). 查询指定结点池中的作业

```
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
29.console cpi zhangxq 0 R long
$ qselect -P dque
25.console
```

3). 查询指定主机上的作业

```
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
29.console cpi zhangxq 0 R long
$ qselect -P @console
25.console
29.console
```

4、挂起作业 qhold

语法：qhold [-h hold_list] job_identifier …

作用：挂起批处理作业。 挂起有三种类型：普通用户级‘u’、管理员级（操作员级）‘o’、系统级‘s’，缺省值为不挂起‘n’。用户只能在用户级别挂起自己提交的作业，操作员可以在用户级和操作员级挂起任何作业，系统管理员可以在任何级别上挂起任何作业。

在执行该命令时，如果作业在运行结点池里排队，那么作业将直接被挂起；如果作业处于运行状态，为了中断作业的执行，必须采取其他办法。如果被挂起作业的主机系统支持一致点检查或者重新启动，则挂起正在运行作业将引发以下操作：首先检查作业的一致性，然后释放该作业所占用的资源，最后该作业位于执行结点池中，处于挂起状态；如果被挂起

作业的主机系统不支持一致点检查或者重新启动，则仅设置指定的挂起作业类型，而实际上并不能挂起，除非调用 qrerun 命令重新运行该作业时，该挂起请求才生效。

举例：

1). 使用普通用户身份挂起作业

```
$ qhold -h u 25.console $ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 H dque
```

2). 以操作员身份挂起作业

```
$ qhold -h o 25
qhold: Unauthorized Request 25.console
$ su root
$ qhold -h o 25
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 H dque
```

5、释放作业 qrls

语法：qrls [-h hold_list] job_identifier ...

作用：释放被挂起的批处理作业。由于作业的挂起有三种类型：USER、OPERATOR 和 SYSTEM。所以，要释放不同类型的作业挂起，用户就必须具有相应的权限。缺省为 USER 级

举例：释放在普通用户和操作员级被挂起的作业。

```
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 H dque
$ qrls -h uo 25
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
```

6、重新运行作业 qrerun

语法：qrerun job_identifier ...

作用：重新运行所指定的作业。

举例：如果作业允许别重新执行（缺省值是可以重新运行）。ROOT 用户运行该命令，可以终止本次运行，把该作业放入其原来所在的结点池中，重新运行。

```
$ qrerun 27
qrerun: Unauthorized Request 27.console
$ su - root
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
27.console cpi zhangxq 0 R dque
$ qrerun 27
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
27.console cpi zhangxq 0 R dque
```

7、更改作业属性 qalter

语法：qalter [-a date_time] [-A account_string] [-e path] [-h hold_list] [-j join] [-k keep] [-l resource_list] [-m mail_options] [-M user_list] [-N name] [-o path] [-p priority] [-r c] [-u user_list] job_identifier...

参数：各参数的含义见 qsub 命令。

作用：更改批处理作业的属性。主要修改所指定作业 ID 的相关属性（选项表中所列出的属性）。

举例：

1). 更改批处理作业的运行时间属性。

```
$ qalter -a 0309251000 23.console
```

2). 更改给用户发送邮件的时间为作业中止和运行结束时

```
$ qalter  - m ae 23.console
```

8、删除作业 qdel

语法：qdel [-W delay|force] job_identifier ...

参数：-W 当指定 delay 时，表示在删除作业前需要等待的时间（秒）
当指定 force 时，强制删除该作业。

作用：删除批处理作业。按照命令行中所指定的作业 ID 的顺序来删除作业。

举例：

```
$ qstat
   Job id Name User TimeUse S Pool
   22.console cpi zhangxq 0 Q long
   $ qdel 22
```

9、给作业发送消息 qmsg

语法：qmsg [-E] [-O] message_string job_identifier ...

参数：-E 将消息串写入错误输出文件
-O 将消息串写入输出文件

作用：给批处理作业发送消息。该命令通过给作业的所有者（批处理服务器）发送消息，从而把消息写入作业的输出文件，也就是说，该命令并不是直接把消息写入作业的输出文件。

举例：给正在运行的作业发送消息，该消息被写入所指定作业的错误输出文件中。

```
$ qstat
Job id Name User TimeUse S Pool
25.console cpi zhangxq 0 Q dque
26.console cpi zhangxq 0 R dque
$ qmsg “The job is running” 26
$ more cpi.e26
The job is running
```

10、在结点池之间移动作业 qmove

语法：qmove destination job_identifier ...

参数：destination 可以为结点池名

作用：把批处理作业移到其他结点池中去运行。即把作业从所在的结点池中删除，并放在其他结点池中。

举例：

```
$ qstat
Job id Name User TimeUse S Pool
22.console cpi zhangxq 0 Q dque
$ qmove long 22
$ qstat
Job id Name User TimeUse S Pool
22.console cpi zhangxq 0 Q long
```

11、给作业发送信号 qsig

语法：qsig [-s signal] job_identifier ...

参数：参数 signal 可以为信号名称，如 SIGKILL,KILL,SIGNULL 或者无符号整数，如 9、0。

作用：给正在运行的批处理作业发送信号。如果不指定-S 选项，则发送“SIGTERM”信号。如果有下列原因之一，如用户无权给作业发送信号、作业未处于运行状态和该信号请求对作业所运行的系统无效，则拒绝执行该命令请求。

12、查看和管理结点 ljrsnodes

语法：ljrsnodes [- {c|o|r}] [nodename …] ljrsnodes - {a|l }]

参数：-a 列出所有结点及其属性，属性包括“state”和“properties”

-c 清除结点列表中的“offline”或“down”状态设置，使结点可以被分配给作业。

-l 以行的方式列出被标记的结点的状态 -o 将指定结点的状态标记为“offline”。这将帮助管理员暂时停止某些结点的服务。

-r 清除指定结点的“offline”状态

作用：该命令可以标记结点的状态为“offline”、“down”或“free”,也可以帮助用户查看结点信息。除了”-a” 和“-l”选项外，使用其他选项需要拥有操作员和管理员权限。

举例：1、查看所有结点信息

```
ljrsnodes -a
```

2、将结点标记为“offline”状态

```
ljrsnodes －o c0108
```

3、清除结点的“offline”状态设置

```
ljrsnodes －c c0108 或 ljrsnodes －r c0108
```

13、查看所有计算结点的状态 shownodes

语法：shownodes [-h]

作用：按照结点池分类显示各结点池中结点的状态，结点状态以不同颜色显示。并在结点名后以数字代表可用 CPU 数和总共 CPU 数。例如 c0101-2/2,表示 c0101 结点可用 2CPU 总共 2CPU，c0102-1/2 表示 C0102 可用 1CPU 总共 2CPU。

举例：$shownodes －h

八．资源分配策略及 PBS 脚本相应参数设置指南

使用本集群的所有用户，均建议通过 Torque 作业调度系统提交任务，作业提交后直接调度到缺省队列运行，缺省队列的资源分配策略见下表，用户申请资源时在各自应用的 pbs 脚本作相应设置，设置的参数值不要超过队列允许的最大值，如果不做相应设置，系统默认使

用缺省值。

参数 / 范围	节点数 (nodes)	CPU 核数 (ppn)	运行时间 （walltime）
最小值	1 个	1 个	00:00:10 (小时:分钟:秒)
缺省值	1 个	12 个	12:00:00
最大值	5 个	60 个	168:00:00

说明：

1、用户在 pbs 脚本中应设定节点数目（nodes）、每节点 cpu 核数（ppn），作业大致的运行时（walltime）间，系统根据以上参数为作业分配资源，确保作业顺利运行。

2、用户如果没有指定作业大致的运行时间，系统将会在到达上述设定的缺省运行时间后强行终止相应作业的运行，所以建议用户能够指定一个合理的运行时间。

3、用户在 pbs 脚本里面设定的相应参数值不在上表范围之内，系统将无法分配资源给用户运行作业，给出提示：qsub: Job rejected by all possible destinations。

附件：PBS 作业模板

```
#!/bin/bash -x
#PBS -N hpcc_test
#PBS -l nodes=4:ppn=8
#PBS -l walltime=07:30:00
#PBS -j oe

#define variables
MPI_HOME=/public/soft/ompi140-gnu/
NSLOTS=`cat ${PBS_NODEFILE} | wc -l`
echo "This jobs is "$PBS_JOBID@$PBS_QUEUE

#running jobs
cd $PBS_O_WORKDIR
time -p ${MPI_HOME}/bin/mpirun -np ${NSLOTS} -hostfile ${PBS_NODEFILE} ./oceanM
ocean_upwelling.in >& out.dat
exit 0
```