
北京航空航天大学校级计算平台项目 用户手册

（正式版 V2.2）

2022-8-29

目录

1. 系统简介	3
1.1. 硬件配置	3
1.2. 软件配置	3
1.3. 系统架构	4
2. 快速入门	5
2.1. 系统计费	5
2.1.1. 计费标准	5
2.1.2. 计费策略	6
2.1.3. 缴费查询	6
2.2. 平台规则设置	6
2.2.1. 登录安全设置	6
2.2.2. 登录节点限制	7
2.2.3. 存储配额限制	7
2.3. 客户端连接	7
2.4. 准备程序和文件	8
2.5. 指定运行环境	9
2.6. 编译程序	10
2.7. 编写作业提交脚本	10
2.8. 提交作业	12
2.9. 使用图形交互	12
2.9.1. Linux	12
2.9.2. MacOS	13
2.9.3. Windows	13
3. 配置运行环境	15
3.1. 查看 module	15

3.2.	卸载 module	16
3.3.	在作业调度系统中使用 module	16
3.4.	定制默认载入的 module	17
3.5.	module 与 anaconda	17
4.	作业管理	19
4.1.	查看系统状态	20
4.2.	分区介绍	21
4.3.	查看作业状态	21
4.4.	交互式作业提交	22
4.5.	批处理作业提交	23
4.5.1.	串行作业	24
4.5.2.	并行作业	25
4.5.3.	GPU 作业	26
4.6.	分配式作业提交	26
4.7.	结束作业	27
4.8.	追踪作业	28
4.9.	更新作业	30
4.10.	使用作业数组	31
5.	应用使用示例	33
5.1.	Ansys	33
5.2.	Fluent	34
5.3.	CFX	35
5.4.	VASP	36
5.5.	Lammps	37
6.	技术支持	37

1. 系统简介

1.1. 硬件配置

本系统配置了 3 个登录节点，328 个 CPU 计算节点，14 个 GPU 计算节点，一套 1.8P 共享存储。所有节点通过 100Gb/s EDR Infiniband 互联组成计算和存储网络。系统详细配置如下：

- 1) 登录节点：共 3 个登录节点。
- 2) GPU 计算节点：
 - 10 个 NVIDIA V100 GPU 计算节点，每个节点配置 2 颗 Intel Golden 6240 系列处理器，共 36 个物理核，384GB 内存，8 个 NVIDIA V100 GPU 卡。
 - 2 个 NVIDIA A100 GPU 计算节点，每个节点配置 2 颗 Intel Golden 6240R 系列处理器，共 48 个物理核，384GB 内存，8 个 NVIDIA A100 GPU PCIe 卡。
 - 2 个 NVIDIA A100 GPU 计算节点，每个节点配置 2 颗 AMD EPYC 7742 系列处理器，共 128 个物理核，384GB 内存，8 个 NVIDIA A100 GPU SXM4 卡。
- 3) CPU 计算节点：
 - 260 个 CPU 计算节点，每个节点配置 2 颗 Intel Golden 6240 系列处理器，共 36 物理核，384GB 内存（根据 IB 网络配置，单个作业最多使用 160 个计算节点）。
 - 68 个 CPU 计算节点，每个节点配置 2 颗 Intel Golden 6240R 系列处理器，共 48 个物理核，384GB 内存。
- 4) 并行存储系统：配置一套 DDN 并行存储系统，共配置 1.8PB 存储容量。
- 5) 管理网络：配置一套千兆管理网；
- 6) 带外管理网络：配置一套千兆带外管理网；
- 7) 计算网络：配置一套 100Gb/s 高速 Infiniband 网。

1.2. 软件配置

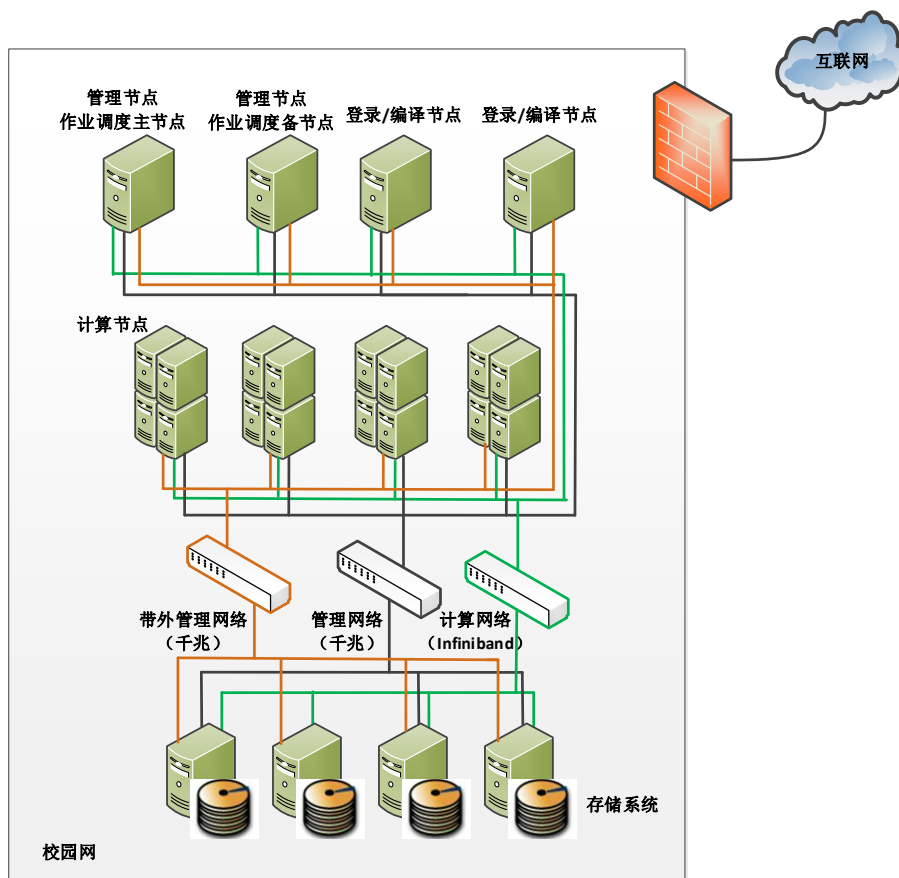
系统软件配置环境如下：

- 1) 操作系统：采用 CentOS 7.6 版本操作系统；

- 2) 作业调度软件：采用 Slurm 19.04 作业调度系统；
- 3) 并行文件系统：采用 DDN 并行文件系统；
- 4) 编译环境：GNU、Intel 编译环境、CUDA；
- 5) 并行环境：MPICH、MVAPICH、OPENMPI、Intel MPI；
- 6) 应用软件：Gromacs、Lammps、R、VASP（需要授权）、OpenFoam、Ansys（试用）、Fluent（试用）、CFX（试用）、Matlab、Moose 等；
- 7) 其他工具软件：Anaconda、fftw、CAFFE、HDF5、QT、PyTorch。

1.3. 系统架构

系统硬件架构图如下：



2. 快速入门

北京航空航天大学 HPC 平台于 2021 年 5 月 6 日全面实施收费运行策略，所有计算作业需要在账户具有足够计算机时的前提下才能提交运行，暂不支持欠费计算。计算机时可以通过缴费购买与奖励申请两种方式获得。为了保持用户的使用习惯，申请账号的基本流程与试运行阶段保持一致，指导老师通过 OA 办公系统申请。

申请高性能计算平台账号：

<http://nic.buaa.edu.cn/content.jsp?urltype=news.NewsContentUrl&wbtreeid=1025&wbnewsid=3486>

缴纳高性能计算平台使用费：

<http://nic.buaa.edu.cn/content.jsp?urltype=news.NewsContentUrl&wbtreeid=1025&wbnewsid=3487>

申请高性能计算平台奖励：

<http://nic.buaa.edu.cn/content.jsp?urltype=news.NewsContentUrl&wbtreeid=1025&wbnewsid=3488>

2.1. 系统计费

2.1.1. 计费标准

根据学校相关管理办法，北航 HPC 平台的收费标准如下：

1) 计算资源收费标准

序号	费用名称	单位	单价(元)	QOS	分区名	最大资源限制	最大运行时长	服务质量
1	CPU 计算费	核/时	0.05	cpu-low	cpu-low	无	7 天	低
2		核/时	0.07	cpu-normal	cpu-normal	无	7 天	中
3		核/时	0.1	cpu-high	cpu-high	无	7 天	高
4		核/时	0.05	cpu-quota	cpu-quota	无	7 天	学科科研优先
5		核/时	0.05	无	cpu68	无	7 天	无
6	GPU	卡/时	2.50	gpu-low	gpu-low	2 卡、8 核	7 天	低

7	计费	卡/时	3.75	gpu-normal	gpu-normal	4 卡、16 核	7 天	中
8		卡/时	5.00	gpu-high	gpu-high	8 卡、36 核	7 天	高
9		卡/时	2.50	gpu-quota	gpu-quota	8 卡、36 核	7 天	学科科研优先
10		卡/时	2.50	无	gpu1	无	7 天	需申请使用
11		卡/时	2.50	无	gpu2	无	7 天	需申请使用

2.1.2. 计费策略

- 1) 计费的最小单元是 0.01 小时；
- 2) QOS 的优先级越高，单位时间的价格越高；
- 3) 机时计算方法如下：
 - CPU 分区机时：CPU 核数*作业运行时间
 - GPU 分区机时：GPU 卡数*作业运行时间
- 4) 收费机时和奖励机时不支持退费。
- 5) 系统每天零时对账户进行一次检测，对于欠费账户，其管理的所有用户将不能再提交作业，已经提交和运行的作业不受影响。

2.1.3. 缴费查询

- 1) 缴费查询地址：<http://10.212.66.3>
- 2) 邀请码说明：邀请码由指导老师通过计费查询系统生成。

2.2. 平台规则设置

根据平台前期运营情况，从以下几方面设定规则进行统一管理。

2.2.1. 登录安全设置

10 次密码错误登录，那么登录时所使用 IP 将被自动封锁，需要联系中心技术人员对 IP 进行解锁。

2.2.2. 登录节点限制

登录节点不能运行负载大的任务，设置定时任务，对负载大的任务进行定时清理。基本规则包括：进程 CPU 使用率 $\geq 100\%$ 且运行时间超过 15 分钟、进程内存使用大小超过 10GB；

2.2.3. 存储配额限制

本系统用户默认的存储配额为 2TB，存储空间超出 2TB 以后读写文件将提示错误。

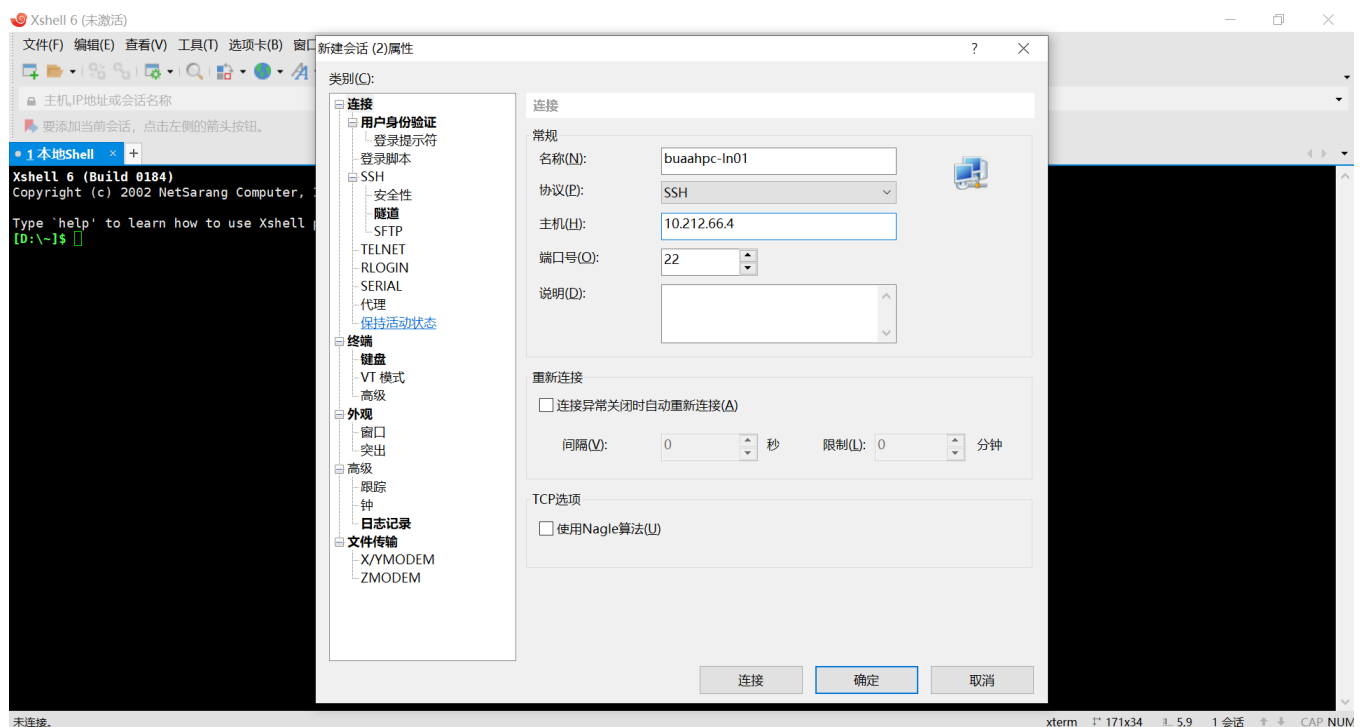
2.3. 客户端连接

建议采用 XShell 或 Putty 等软件连接。

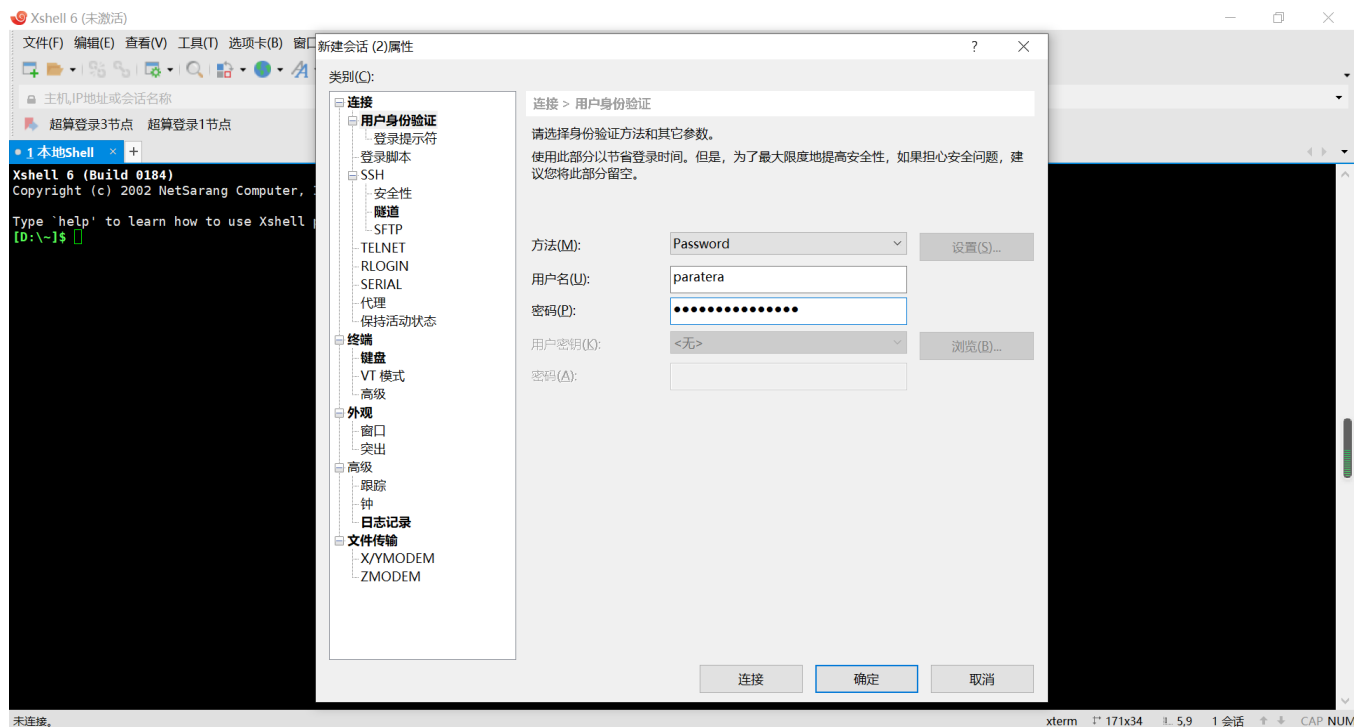
登录节点的 IP 地址为：10.212.66.4/10.212.66.5/10.212.70.128。

下面为 XShell 登录基本操作：

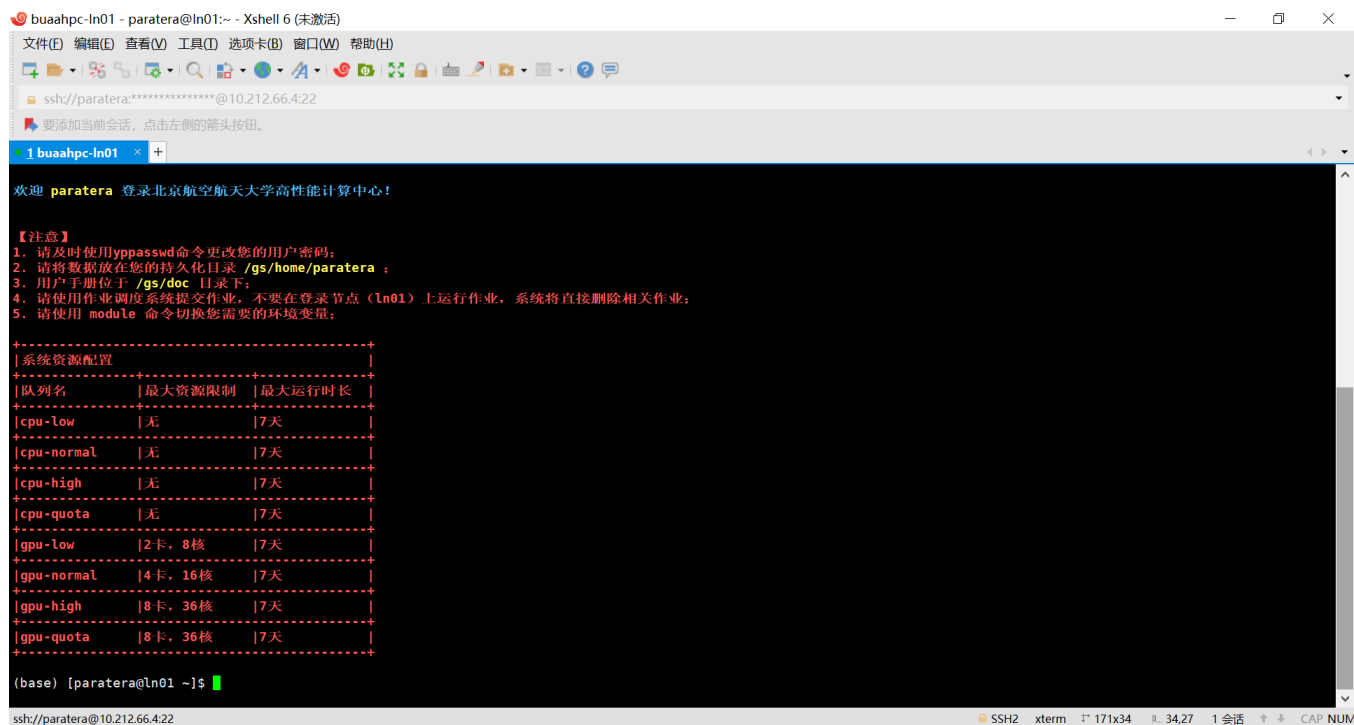
1) 打开 XShell 软件，新增连接，填入主机 IP 地址和端口号，选择 SSH 协议：



2) 输入用户名和密码：



3) 点击确定，登录系统：

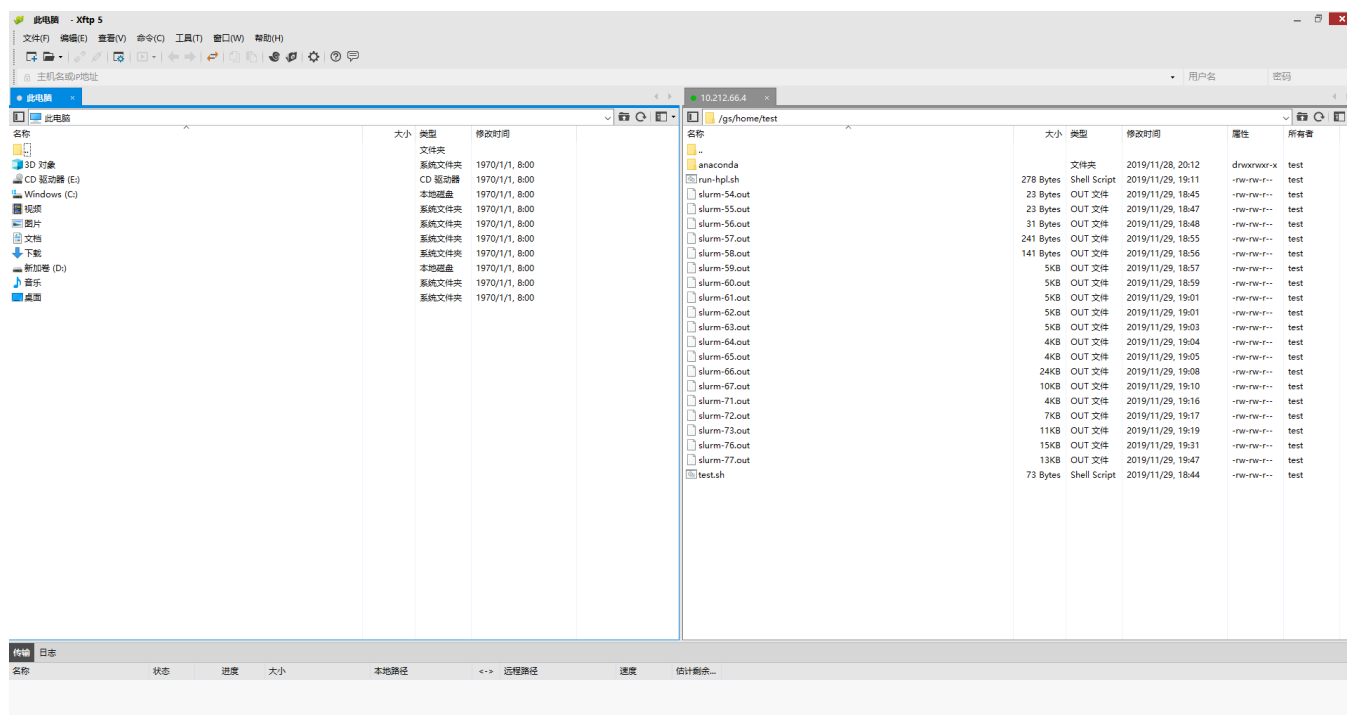


2.4. 准备程序和文件

在登录成功后，就可以准备测试程序和文件了，您可以直接在服务器上编辑，也可以通过 Xftp

将程序和文件上传到服务器上。

系统为每个用户配置了持久化保存的目录，保存在`/gs/home`下，您需要长久保存的文件可以放置在这个目录中，这个目录是共享目录，所有节点都挂载。



2.5. 指定运行环境

程序源码准备完毕后，您需要为程序搭建其所需的运行环境。例如准备相应的库以及编译成可执行程序等。不建议将本地编译好的程序直接上传，因为服务器的环境可能与本地不相同，这样容易导致程序运行出错。

如果程序依赖于系统的环境变量和库，则需要通过 `Environment Module` 进行模块的载入，详细可参考文档第 4 节“配置运行环境”。

这里，我们假设您准备的程序是一个简单的 C 程序，不依赖于更复杂的库，程序如下：

```
[test@ln01 ~]$ vim helloworld.c

#include <stdio.h>

int main(void)

{
```

```
printf( "Hello,World!\n" );

return 0;
}
```

2.6. 编译程序

```
[test@ln01 ~]$ gcc -o helloworld helloworld.c
[test@ln01 ~]$ ls
anaconda helloworld helloworld.c run-hpl.sh test.sh
```

2.7. 编写作业提交脚本

完成上述操作后，您就可以通过作业调度系统提交作业。

本平台配置了 **Slurm** 作业调度系统，您可以参考文档“作业管理”了解作业调度系统的详细说明，本节只给出一个最基本的例子。

使用 **Slurm** 时需要编辑已给作业调度提交脚本，请参考如下示例：

```
#!/bin/bash

#SBATCH -J test           # 作业名是 test
#SBATCH -p cpu-low       # 提交到 cpu-low 分区
#SBATCH -N 1             # 使用一个节点
#SBATCH -n 1             # 使用一个进程（cpu 核）
#SBATCH -t 5:00          # 任务最大运行时间是 5 分钟
#SBATCH -o test.out       # 将屏幕的输出结果保存到当前文件夹的 test.out
#SBATCH -e test.err       # 将屏幕的输出结果保存到当前文件夹的 test.err

./helloworld             # 执行我的 ./helloworld 程序
Sleep 200
```

以上的脚本的第一行指定了这个脚本的解释器为 **bash**。每次编写脚本都必须写上这一行。之后有 **#**开头的若干行表示 **SLURM** 作业的设置区域，它告诉工作站运行任务的详细设定，它被提

交到 `cpu-low` 分区当中，申请 1 个节点的 1 个 核心，限制任务最大运行时间是五分钟，将标准输出放在 `test.out` 中，标准错误输出放在 `test.err` 中。 它的主体内容就是在当前目录执行编译好的程序 `helloworld`。

关于分区的类型，可以先使用 `sinfo` 查看可用资源情况（服务队列）。

```
[test@ln01 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
cpu-low    up      infinite   260    idle compute-[001-260]
cpu-low    up      infinite   260    idle compute-[001-260]
cpu-high   up      infinite   260    idle compute-[001-260]
cpu-quota  up      infinite   260    idle compute-[001-260]
cpu68      up      infinite    68    idle compute-[261-328]
gpu-low    up      infinite   10     idle  gpu-[01-10]
gpu-normal up      infinite   10     idle  gpu-[01-10]
gpu-high   up      infinite   10     idle  gpu-[01-10]
gpu-quota  up      infinite   10     idle  gpu-[01-10]
gpu1       up      infinite    2     idle  gpu-[11-12]
gpu2*      up      infinite    2     idle  gpu-[13-14]
```

其中 `PARTITION` 表示分区，`NODES` 表示结点数，`NODELIST` 为结点列表，`STATE` 表示结点运行状态。其中，`idle` 表示结点处于空闲状态，`allocated` 表示结点中的资源已经完全分配无法接受新的作业。

本系统分为 4 个 `cpu` 队列（包含 260 个计算节点），4 个 `gpu` 队列（包含 10 个 GPU 计算节点），其中*-low 为优先级最低的作业队列，计算费用最便宜；其次是*-normal，优先级比*-low 高，计算费用同样略高于*-low；然后是*-high，优先级和计算费用比前两个队列高；最后是*-quota，该队列为学科科研优先队列，具有最高优先级，只有经相关部门认定后购买的机时才能享受该队列。另外还包含 `cpu68`（包含 68 个计算节点）、`gpu1`（包含 2 个 GPU 计算节点）、`gpu2`（包含 2 个 GPU 计算节点）三个队列。其中 `gpu1`、`gpu2` 作为私有分区，需要向信息化中心的老师进行申请使用。当分区状态为空闲（`idle`）或混合（`mix`）时，作业可以提交。

2.8. 提交作业

准备好 SLURM 脚本之后，可以使用 `sbatch` 提交作业：

```
[test@ln01 ~]$ sbatch run.slurm
Submitted batch job 85
[test@ln01 ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
85	cpu-low	test	test	R	0:13	1	compute-001

在上面的输出中，`sbatch` 返回的信息是 “Submitted batch job 85”，这表示我的作业已经成功提交，作业号是 85。而 `squeue` 显示我提交的作业的具体信息，在这里看到我的作业被放到 `cpu-low` 分区上运行，占用 1 个节点 `compute-001`，状态(ST)是运行状态(R)，并且已经运行了 13 秒钟。

2.9. 使用图形交互

通常情况下，在集群中我们都在命令行中操作。但是有时我们需要打开某些软件的图形界面，此时我们需要借助 X11 转发。

注意：在网络不好的时候请谨慎使用此功能，此时经过 X11 转发的界面操作起来有明显卡顿。尽量在校内有线网或信号较好的无线网的环境下使用。

2.9.1. Linux

在 Linux 环境中启用 X11 转发非常简单，只需要在登录时加入 `-X` 参数。

```
test@lcaol: $ ssh -X user@server_ip
```

此后登录集群可以直接输入命令开启图形界面的程序。例如：

```
(base) [test@ln01 ~]$ module load matlab/R2019b
(base) [test@ln01 ~]$ matlab
MATLAB is selecting SOFTWARE OPENGGL rendering.
```

2.9.2. MacOS

使用 MacOS 系统时需要安装第三方支持 X11 的软件 XQuartz。

安装完毕后，在登录集群时加入 `-X` 参数。

```
test@test: $ ssh -X user@server_ip
```

此后登录集群可以直接输入命令开启图形界面的程序。例如：

```
(base) [test@ln01 ~]$ module load matlab/R2019b
```

```
(base) [test@ln01 ~]$ matlab
```

```
MATLAB is selecting SOFTWARE OPENGL rendering.
```

2.9.3. Windows

推荐使用 MobaXterm 客户端进行连接。

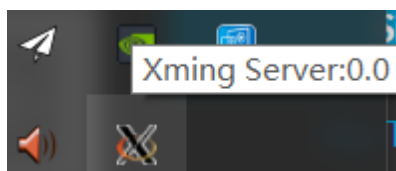
在默认情况下 MobaXterm 会自动打开 X11 的转发，因此只需正常登录然后直接输入打开软件的命令即可。（使用 Xshell 系列的用户需要额外下载 Xmanager）

使用 PuTTY 配合 Xming

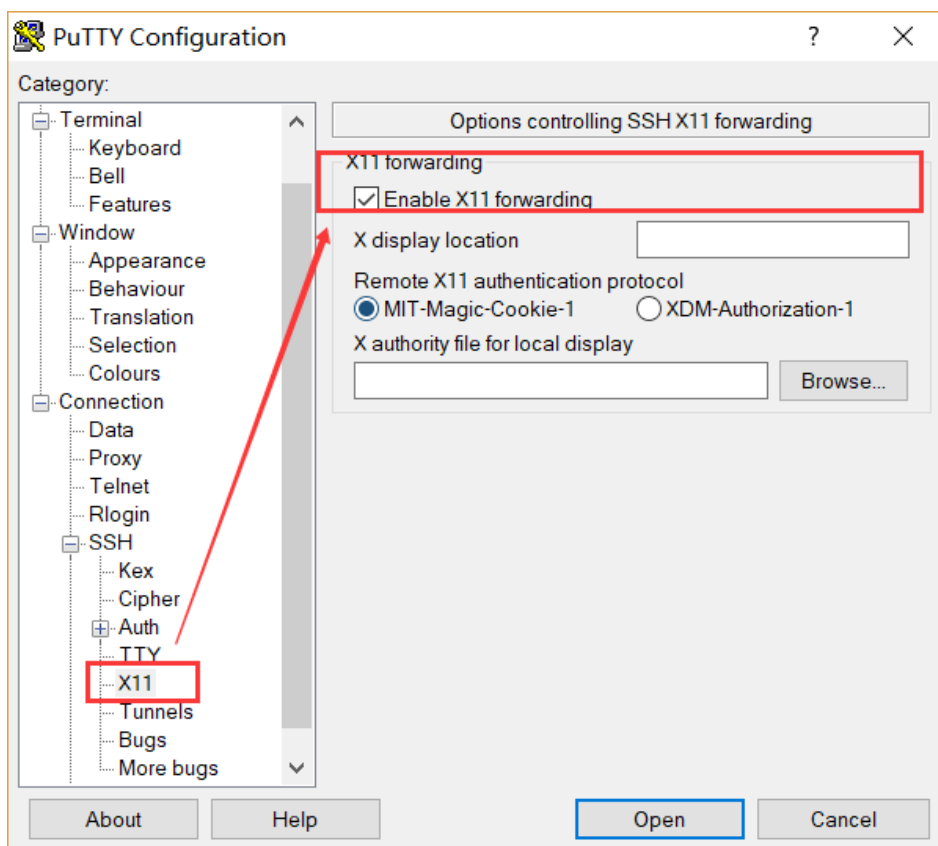
PuTTY 用户可配合 Xming 软件完成 X11 的转发。

首先下载 Xming。

安装完成后双击 Xming.exe 打开 Xming 的 Xserver，此时会在系统任务栏的 右下角看到 Xming 正在运行。



然后打开 PuTTY 客户端，选择服务器的相应 Session 配置，点击 Load 载入这个配置。在左边面板选择 SSH -> X11，在右边勾选 Enable X11 Forwarding。最后回到 Session 界面点击保存即可。



3. 配置运行环境

为了满足同学们计算任务的需求，服务器中安装了各种版本的软件。大家可在同一软件的不同版本之间切换，也可以在同一功能的不同软件之间切换，以此来选择最合适的编程环境和运行环境。使用系统命令 `module` 可以快速地达到这一效果。

3.1. 查看 module

在你使用某个软件之前，您必须通过 `module` 命令来配置此软件的运行环境，否则将无法使用此软件或者使用一个错误的版本。

例如，在没有载入任何 `module` 的情况下，输入 `python` 来尝试打开 `python`

```
[test@ln01 ~]$ python
Python 2.7.5 (default, Oct 30 2018, 23:45:53)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注意到此时 `python` 命令代表系统内置的 `python`，版本是 2.7.5。这是一个比较旧的版本，里面少了很多功能，和最新的许多包都无法兼容。

此时我们用 `module` 查看当前可用的模块，可以看到系统安装了 `anaconda3`：

```
(base) [paratera@ln02 ~]$ module avail

----- /opt/ohpc/pub/modulefiles -----

EasyBuild/3.9.2      cmake/3.14.3      gromacs/2019.4      matlab/R2019b
OpenFOAM/OpenFOAM-6  cuda/9.0          hwloc/2.0.3         petsc/3.11.4
anaconda2/2019.10    cuda/10.1 (D)     intel/18.0.3.222    pmix/2.2.2
anaconda3/2019.10    gnu/5.3.0         intel/19.0.5.281 (D) prun/1.3
ansys/v192          gnu7/7.3.0        jdk/1.8.0_201       valgrind/3.15.0
autotools           gnu8/8.3.0        lammps/20191030
```

使用 `module load anaconda3` 命令载入较新的 `anaconda3`


```
[test@ln01 ~]$ module load anaconda3

[test@ln01 ~]$ python

Python 3.7.4 (default, Aug 13 2019, 20:35:49)

[GCC 7.3.0] :: Anaconda, Inc. on linux

Type "help", "copyright", "credits" or "license" for more information.

>>
```

3.2. 卸载 module

卸载 module 负责对加载的 module 进行卸载:

```
[test@ln01 ~]$ module list

Currently Loaded Modules:

  1) anaconda3/2019.10

[test@ln01 ~]$ module unload anaconda3

[test@ln01 ~]$ module list

No modules loaded
```

一个模块删除之后, 您将无法直接使用与之相关的命令。对应软件的执行目录无法被直接访问, 或者是还原成系统默认的版本。

有的时候, 模块之间会有所冲突, 您无法在同一时间同时加载两个模块。例如同一个软件的不同版本, 或者是接口相同的不同软件。当你载入其中一个后, 再载入另一个就会出错。

3.3. 在作业调度系统中使用 module

module 命令仅作用在当前节点上, 如果使用 SLURM 脚本提交任务, 那么实际运行任务的节点和当前节点是不同的, 因此方便的做法是将 module 命令一并写在 SLURM 脚本中, 例如:

```
#!/bin/bash

#SBATCH -J test

#SBATCH -p cpu-low

#SBATCH -t 5:00
```

```
module load anaconda/3

# do your computation jobs...
```

3.4. 定制默认载入的 module

当您登录系统时，系统会默认载入一些 module。如果这些 module 不是你需要的，或者需要载入更多的 module，那么请直接在个人 HOME 目录下的.bashrc 文件的最后添加“module load 需要加载的模块”。

3.5. module 与 anaconda

随着大数据、机器学习技术的发展，基于 Python 技术的应用场景越来越宽广，传统高性能计算环境也需要使用这些技术。Anaconda 指的是一个开源的 Python 发行版本，其包含了 conda、Python 等 180 多个科学包及其依赖项。conda 是一个开源的包、环境管理器，可以用于在同一个机器上安装不同版本的软件包及其依赖，并能够在不同的环境之间切换。

虽然系统中存在 Anaconda3，但推荐用户在自己的家目录下使用自己的 Anaconda3 来自定义环境，这样用户可以在自己的环境下安装需要的库，这些不会对其他人的环境产生影响。该过程只需要在登陆节点完成，无需申请 GPU 卡。

下面的操作为具体的配置 conda 环境的方法（黑色加粗代表输入的命令，其他是系统输出内容）。

首先下载相关环境：

下载地址：

https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/Anaconda3-2019.10-Linux-x86_64.sh 然后
上传到 HPC 平台

增加执行权限：

chmod +x Anaconda3-2019.10-Linux-x86_64.sh

安装（按照提示操作即可，输出略）：

bash Anaconda3-2019.10-Linux-x86_64.sh

注意默认路径：/gs/home/test/anaconda3

加载环境变量:

```
source ~/.bashrc
```

检查是否安装成功:

```
which python
```

```
~/anaconda3/bin/python
```

在使用 Anaconda 时, 可能需要管理不同版本的软件包, 可以建立虚拟环境管理, 具体如下:

创建一个名字为 `testenv` 的 Conda 环境:

```
conda create -n testenv python=3.7
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done
## Package Plan ##
  environment location: /gs/home/test/anaconda3/envs/testenv
  added / updated specs:
    - python=3.7
The following NEW packages will be INSTALLED:
(安装库的名字省略)
Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
# To activate this environment, use
#   $ conda activate testenv
# To deactivate an active environment, use
#   $ conda deactivate
```

激活 `testenv` 的 conda 环境, 星号为当前使用的环境:

```
conda activate testenv
```

```
conda env list
```

```
# conda environments:
#
testenv          */gs/home/test/anaconda3/envs/testenv
base             /gs/home/test/anaconda3
```

列出 `testenv` 环境已安装的包列表:

```
conda list
```

退出 `testenv` 的 conda 环境:

```
conda deactivate
```

安装 `scipy` 包的例子:

```
conda activate testenv
```

```
pip install scipy
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
```

```
Collecting scipy
```

```
  Downloading
```

```
https://pypi.tuna.tsinghua.edu.cn/packages/13/2e/4843308bb8147e7f86e99d1b233c64eb4ffa70423dfe21458822cb39cb55/scipy-1.7.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (38.2 MB)
```

```
|████████████████████████████████████████████████████████████████████████████████| 38.2 MB 438 kB/s
```

```
Collecting numpy<1.23.0,>=1.16.5
```

```
  Using cached
```

```
https://pypi.tuna.tsinghua.edu.cn/packages/5b/0d/de55834c5ea0dd287cb1cb156c8bc120af2863c36e4d49b4dc28f174e278/numpy-1.21.4-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.7 MB)
```

```
Installing collected packages: numpy, scipy
```

```
Successfully installed numpy-1.21.4 scipy-1.7.2
```

```
(testenv) [test@ln01 ~]$ python -c "import scipy; print(scipy.__version__)"
1.7.2
```

删除 testenv 的 conda 环境

```
(base) [test@ln01]$ conda env remove -n testenv
```

```
Remove all packages in environment /gs/home/test/anaconda3/envs/testenv:
```

```
(base) [test@ln01]$ conda env list
```

```
# conda environments:
```

```
#
```

```
base * /gs/home/test/anaconda3
```

4. 作业管理

下面对作业管理进行介绍，作业管理相关的操作主要包括：查看作业调度系统状态、查看作业状态、交互式作业提交、批处理作业提交、分布式作业提交、结束作业、追踪作业、更新作业、使用作业数组以及分区详解。有关作业管理的相关内容，如果需要更详细的信息，请参考 Slurm

官方网站: <https://slurm.schedmd.com/>。

4.1. 查看系统状态

使用 `sinfo` 可以查看系统的状态:

```
[test@ln01 ~]$ sinfo
```

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
cpu-normal    up   infinite    260   idle compute-[001-260]
cpu-low       up   infinite    260   idle compute-[001-260]
cpu-high      up   infinite    260   idle compute-[001-260]
cpu-quota     up   infinite    260   idle compute-[001-260]
gpu-low       up   infinite    10    idle  gpu-[01-10]
gpu-normal    up   infinite    10    idle  gpu-[01-10]
gpu-high      up   infinite    10    idle  gpu-[01-10]
gpu-quota     up   infinite    10    idle  gpu-[01-10]
cpu68         up   infinite     68   idle compute-[261-328]
gpu1          up   infinite     2    idle gpu-[11-12]
gpu2*         up   infinite     2    idle gpu-[13-14]
```

`sinfo` 输出的关键词说明

关键词	含义
PARTITION	用户可用的计算分区
AVAIL	可用 : up 表示可用; down 表示不可用
TIMELIMIT	该分区的作业最大运行时长限制
NODES	结点数量
STAT	<p>down: 不可用状态</p> <p>idle: 空闲状态, 可以接收新作业</p> <p>alloc: 已被分配状态</p> <p>mix: 混合状态, 节点在运行作业, 但有空闲 CPU 核, 可接受新作业</p> <p>drain: 排空状态, 表示该类结点不再分配到其他作业</p>

NODELIST	结点名字列表
----------	--------

本系统由以下分区组成：

4 个 cpu 分区：包含 260 个 CPU 为 6240 的 CPU 计算节点；

4 个 gpu 分区：包含 10 个 v100 GPU 的 GPU 计算节点；

cpu68 分区：包含 68 个 CPU 为 6240R 的 CPU 计算节点；

gpu1 分区：包含 2 台 8 卡 A100 PCIe GPU，CPU 为 Intel Golden 6240R 的 GPU 计算节点；

gpu2 分区：包含 2 台 8 卡 A100 SXM4 GPU，CPU 为 AMD EPYC 7742 的 GPU 计算节点。

4.2. 分区介绍

不同的节点的特性和硬件属性不同，设置分区可以帮助用户更好确定节点的特点，进而选择最适合自己的节点进行运算。此外，如果集群中部分机器是私有的，那么设置分区可以使得只有部分用户能在这个分区提交作业。总的来说，分区(Partition)可看做一系列节点的集合。

目前我们有两类可以提交作业的分区：

gpu 分区：包含 10 个 GPU 计算节点；

cpu 分区：包含 260 个 CPU 计算节点；

其中*-low 为优先级最低的作业队列，计算费用最便宜；其次是*-normal，优先级比*-low 高，计算费用同样略高于*-low；然后是*-high，优先级和计算费用比前两个队列高；最后是*-quota，该队列为学科科研优先队列，具有最高优先级，只有经相关部门认定后购买的机时才能享受该队列。

cpu68 分区：包含 68 个 CPU 计算节点；

gpu1 和 gpu2 分区：分别包含 2 个 GPU 计算节点。这两个分区作为私有分区，需要向信息化办公室高性能运算中心的老师进行申请使用。

4.3. 查看作业状态

查看作业状态的命令为 squeue：

```
[test@ln01 ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
86	cpu-low	test	test	R	0:10	1	compute-001

squeue 输出的关键词说明

关键词	含义
JOBID	任务 ID
PARTITION	用户作业正在使用的计算分区
NAME	任务名称
USER	用户名
ST	作业状态： R: 运行中，running PD: 排队，pending TD: 超时，timeout S: 挂起中，suspended CG: 作业正在退出状态 CD: 成功结束，completed
NODES	结点个数
NODELIST	结点名字列表
REASON	排队状态： PartitionTimeLimit: 超出分区作业时间限制 QOSMaxGRESPerUser: 超出用户最大 GPU 申请数限制 QOSMAXcpuperuserlimit: 超出单用户最大可使用 CPU 核数限制 Nodes required for job are DOWN: 需要等待节点分配空余资源

4.4. 交互式作业提交

交互式作业是指在 Shell 窗口中执行 `srun` 命令，主要的命令格式如下：

```
srun [options] command
```

`sun` 的常用选项包括：

➤ `-n, --ntasks=number`

指定要运行的任务数。请求为 `number` 个任务分配资源，默认为每个任务一个处理器核。

➤ **-c, --cpus-per-task=ncpus**

告知资源管理系统控制进程，作业步的每个任务需要 **ncpus** 个处理器核。若未指定此选项，则控制进程默认为每个任务分配一个处理器核。

➤ **-N, --nodes=minnodes[-maxnodes]**

请求为作业至少分配 **minnodes** 个结点。调度器可能觉得在多于 **minnodes** 个结点上运行作业。可以通过 **maxnodes** 限制最多分配的结点数目（例如 “-N 2-4” 或 “--nodes=2-4”）。最少和最多结点数目可以相同以指定特定的结点数目（例如，“-N 2” 或 “--nodes=2-2” 将请求两个且仅两个结点）。分区的结点数目限制将覆盖作业的请求。如果作业的结点限制超出了分区中配置的结点数目，作业将被拒绝。如果没有指定 **-N**，缺省行为是分配足够多的结点以满足 **-n** 和 **-c** 参数的需求。在允许的限制范围内以及不延迟作业开始运行的前提下，作业将被分配尽可能多的结点。

➤ **-p, --partitions=partition name**

在指定分区中分配资源。如未指定，则由控制进程在系统默认分区中分配资源。

➤ **-w, --nodelist=node name**

请求指定的结点名字列表。作业分配资源中将至少包含这些结点。列表可以用逗号分隔的结点名或结点范围（如 **cn[1-5,7,...]**）指定，或者用文件名指定。如果参数中包含 “/” 字符，则会被当作文件名。如果指定了最大节点数，如：**-N 1-2**，但是文件中有两个多余的节点，则请求列表中只会使用前两个节点。

➤ **-x, --exclude=node name**

不要将指定的节点分配作业。如果包含 “/” 字符，参数被当作文件名。**Srun** 将把作业请求提交到控制进程，然后在远程节点上启动所有进程。如果资源请求不能被立即满足，**srun** 将阻塞等待，直到资源可以运行作业。如果指定了 **-immediate** 选项，则 **srun** 将在资源不是立即可用时终止作业。

➤ **-h, --help**

如果需要使用 **srun** 的更多选项，可以通过 **-h** 或 **--help** 选项查看。

4.5. 批处理作业提交

批处理作业是指用户编写作业脚本，指定资源需求约束，提交后台执行作业。提交批处理作

业的命令是 **sbatch**，用户提交命令即返回命令行窗口，但此时作业进入调度状态，在资源满足要求时，分配完计算节点之后，系统将在所分配的第一个计算节点（不是登录节点）上执行用户的作业脚本。

批处理作业脚本为一个文本文件，脚本第一行以“**#!**”字符开头，并制定脚本文件的解释程序，如 **sh**，**bash**。由于计算节点为精简环境，只提供 **sh** 和 **bash** 支持。

使用 **Slurm** 时需要编辑已给作业调度提交脚本，请参考如下示例：

4.5.1. 串行作业

```
#!/bin/bash

#SBATCH -J test           # 作业名是 test
#SBATCH -p cpu-low        # 提交到 cpu-low 分区
#SBATCH -N 1              # 使用一个节点
#SBATCH -n 1              # 使用一个进程（cpu 核）
#SBATCH -t 5:00           # 任务最大运行时间是 5 分钟
#SBATCH -o test.out       # 将屏幕的输出结果保存到当前文件夹的 test.out
#SBATCH -e test.err       # 将屏幕的输出结果保存到当前文件夹的 test.err

./helloworld              # 执行我的 ./helloworld 程序

Sleep 200
```

以上的脚本的第一行指定了这个脚本的解释器为 **bash**。每次编写脚本都必须写上这一行。之后有 **#** 开头的若干行表示 **SLURM** 作业的设置区域，它告诉工作站运行任务的详细设定：它被提交到 **cpu-low** 分区当中，申请 1 个节点的 1 个 核心，限制任务最大运行时间是五分钟，将标准输出放在 **test.out** 中，标准错误输出放在 **test.err** 中。它的主体内容就是在当前目录执行编译好的程序 **helloworld**。

当一次性提交的作业数量比较多时，可以在同一个作业中生成批量任务，如下例子所示：

```
#!/bin/bash

#SBATCH -J test           # 作业名是 test
#SBATCH -p cpu-low        # 提交到 cpu-low 分区
#SBATCH -N 1              # 使用一个节点
```

```
#SBATCH -n 1                # 使用一个进程（cpu 核）
#SBATCH -t 5:00             # 任务最大运行时间是 5 分钟
#SBATCH -o test.out         # 将屏幕的输出结果保存到当前文件夹的 test.out
#SBATCH -e test.err         # 将屏幕的输出结果保存到当前文件夹的 test.err

srun --exclusive ./helloworld &
srun --exclusive ./helloworld &
srun --exclusive ./helloworld &
srun --exclusive ./helloworld &

wait
```

上面示例的这个作业中一次性生成了 4 个子任务，该作业将在这 4 个子任务全部结束之后才正常结束。这里的`--exclusive`是为了防止子任务同时运行发生资源抢占。

4.5.2. 并行作业

```
#!/bin/bash

#SBATCH -J vasp              # 作业名是 vasp
#SBATCH -p cpu-low          # 提交到 cpu-low 分区
#SBATCH -N 2                # 使用 2 个节点
#SBATCH -n 16               # 使用 16 个进程（cpu 核）
#SBATCH --ntasks-per-node=8 # 每个节点启动 8 个进程
#SBATCH -t 5:00             # 任务最大运行时间是 5 分钟
#SBATCH -o vasp.out         # 将屏幕的输出结果保存到当前文件夹的 vasp.out
#SBATCH -e vasp.err         # 将屏幕的输出结果保存到当前文件夹的 vasp.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}
NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

module load intel/18.0.3.222
```

```
mpirun -genv I_MPI_FABRICS shm:dapl -np ${NP} -f ./machinefile.${SLURM_JOB_ID}
/gs/software/vasp/vasp.5.4.4/bin/vasp_std # 执行我的 vasp 程序
```

4.5.3. GPU 作业

```
#!/bin/bash

#SBATCH -J gputest # 作业名是 gputest
#SBATCH -p gpu-normal # 提交到 gpu-normal 分区
#SBATCH -N 1 # 使用 1 个节点
#SBATCH -n 1 # 使用 1 个进程 (cpu 核)
#SBATCH --gres=gpu:2 # 使用两个 GPU 卡
#SBATCH -t 5:00 # 任务最大运行时间是 5 分钟
#SBATCH -o gputest.out # 将屏幕的输出结果保存到当前文件夹的 gputest.out
#SBATCH -e gputest.err # 将屏幕的输出结果保存到当前文件夹的 gputest.err

conda activate testenv
python MNIST.py # 执行 GPU 程序
```

4.6. 分配式作业提交

`salloc` 命令为分配式作业提交命令，命令结束后将释放分配的资源，基本语法为：

```
salloc [options] command
```

如果后面的 `command` 为空，则执行 `slurm` 的配置 `slurm.conf` 中通过 `SallocDefaultCommand` 设定的命令。如果 `SallocDefaultCommand` 没有设置，则将执行用户默认的 `shell`。

`salloc` 的主要选项如下：

➤ `--core-per-socket=cores`

分配的节点需要至少每颗 CPU 核

➤ `-I, --immediate=seconds`

在 `seconds` 秒内资源未满足的情况下立即退出。

➤ `-J, --job-name=job name`

设定作业名

➤ **-N, --nodes=minnodes[-maxnodes]**

请求为作业至少分配 **minnodes** 个结点。调度器可能觉得在多于 **minnodes** 个结点上运行作业。可以通过 **maxnodes** 限制最多分配的结点数目（例如 “-N 2-4” 或 “--nodes=2-4”）。最少和最多结点数目可以相同以指定特定的结点数目（例如，“-N 2” 或 “--nodes=2-2” 将请求两个且仅两个结点）。分区的结点数目限制将覆盖作业的请求。如果作业的结点限制超出了分区中配置的结点数目，作业将被拒绝。如果没有指定 **-N**，缺省行为是分配足够多的结点以满足 **-n** 和 **-c** 参数的需求。在允许的限制范围内以及不延迟作业开始运行的前提下，作业将被分配尽可能多的结点。

➤ **-n, --ntasks=number**

指定要运行的任务数。请求为 **number** 个任务分配资源，默认为每个任务一个处理器核。

➤ **-p, --partitions=partition name**

在指定分区中分配资源。如未指定，则由控制进程在系统默认分区中分配资源。

➤ **-w, --odelist=node name**

请求指定的结点名字列表。作业分配资源中将至少包含这些结点。列表可以用逗号分隔的结点名或结点范围（如 **cn[1-5,7,...]**）指定，或者用文件名指定。如果参数中包含 “/” 字符，则会被当作文件名。如果指定了最大节点数，如：**-N 1-2**，但是文件中有两个多余的节点，则请求列表中只会使用前两个节点。

➤ **-x, --exclude=node name**

不要将指定的节点分配作业。如果包含 “/” 字符，参数被当作文件名。**salloc** 将把作业请求提交到控制进程，然后在远程节点上启动所有进程。如果资源请求不能被立即满足，**salloc** 将阻塞等待，直到资源可以运行作业。如果指定了 **-immediate** 选项，则 **srun** 将在资源不是立即可用时终止作业。

➤ **-h, --help**

➤ 如果需要使用 **srun** 的更多选项，可以通过 **-h** 或 **-help** 选项查看。

4.7. 结束作业

使用 **scancel** 命令可以取消自己的作业，命令格式如下：

```
scancel jobId
```

Jobid 可以通过 `squeue` 获得。对于排队的作业，取消作业将简单地把作业标记为 CANCELED 状态而结束作业。对于运行中或挂起的作业，取消作业将终止作业的所有作业步，包括批处理作业脚本，将作业标记为 CANCELED 状态，并回收分配给作业的节点。批处理作业取消后马上终止，交互作业的进程会感知到任务的退出而终止，分配式作业的进程不会自动退出，除非作业执行用户命令因作业或任务的结束而终止。

4.8. 追踪作业

SLURM 提供了丰富的追踪任务的命令，例如 `scontrol`，`sacct` 等。这些命令有助于查看正在运行或已完成的任务状态。当用户认为任务异常时，可使用这些工具来追踪任务的信息。

对于正在运行或排队的任务，可以使用以下命令：

```
scontrol show job jobId
```

其中 JOBID 是正在运行的作业 ID，如果忘记 ID 可以使用 `squeue -u USERNAME` 来查看目前处于运行中的作业。

```
[test@ln01 ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
87	cpu-low	test	test	R	0:12	1	compute-001

```
[test@ln01 ~]$ scontrol show job 87
```

```
JobId=87 JobName=test
```

```
UserId=test(1193) GroupId=test(1193) MCS_label=N/A
```

```
Priority=4294901723 Nice=0 Account=hpc QOS=cpu-low
```

```
JobState=RUNNING Reason=None Dependency=(null)
```

```
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
```

```
RunTime=00:00:19 TimeLimit=00:05:00 TimeMin=N/A
```

```
SubmitTime=2019-12-01T19:37:09 EligibleTime=2019-12-01T19:37:09
```

```
AccrueTime=2019-12-01T19:37:09
```

```
StartTime=2019-12-01T19:37:10 EndTime=2019-12-01T19:42:10 Deadline=N/A
```

```
SuspendTime=None SecsPreSuspend=0 LastSchedEval=2019-12-01T19:37:10
```

```

Partition=cpu-low AllocNode:Sid=ln01:27360
ReqNodeList=(null) ExcNodeList=(null)
NodeList=compute-001
BatchHost=compute-001
NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
TRES=cpu=1,node=1,billing=1
Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/gs/home/test/run.slurm
WorkDir=/gs/home/test
StdErr=/gs/home/test/test.out
StdIn=/dev/null
StdOut=/gs/home/test/test.out
Power=

```

作业完成后，需要用 `sacct` 命令来查看历史作业。默认情况下，用户仅能查看属于自己的历史作业。直接使用 `sacct` 命令会输出从当天 00:00:00 起到现在的全部作业。

```
[test@ln01 ~]$ sacct
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
81	hotname	cpu-low	hpc	1	FAILED	2:0
82	hostname	cpu-normal	hpc	1	COMPLETED	0:0
83	bash	gpu-normal	hpc	1	COMPLETED	0:0
84	test	cpu-normal	hpc	1	COMPLETED	0:0
84.batch	batch		hpc	1	COMPLETED	0:0
85	test	cpu-normal	hpc	1	COMPLETED	0:0
85.batch	batch		hpc	1	COMPLETED	0:0

86	test	cpu-normal	hpc	1	COMPLETED	0:0
86.batch	batch		hpc	1	COMPLETED	0:0
87	test	cpu-low	hpc	1	COMPLETED	0:0
87.batch	batch		hpc	1	COMPLETED	0:0

使用“`sacct -S MMDD`”命令可以输出从 MM 月 DD 日起的所有历史作业，默认情况会输出作业 ID，作业名，分区，账户，分配的 CPU，任务结束状态，返回码。当然我们还可以使用 `--format` 参数来指定到底要输出那些指标。

```
[test@ln01 ~]$ sacct --format=jobid, user, alloccpu, allocgres, state%15,
```

exitcode

JobID	User	AllocCPUS	AllocGRES	State	ExitCode
81	test	1		FAILED	2:0
82	test	1		COMPLETED	0:0
83	test	1	gpu:2	COMPLETED	0:0
84	test	1		COMPLETED	0:0
84.batch		1		COMPLETED	0:0
85	test	1		COMPLETED	0:0
85.batch		1		COMPLETED	0:0
86	test	1		COMPLETED	0:0
86.batch		1		COMPLETED	0:0
87	test	1		COMPLETED	0:0
87.batch		1		COMPLETED	0:0

4.9. 更新作业

有时我们很早就提交了任务，但是在任务开始前却发现作业的属性写错了（例如提交错了分区，忘记申请 GPU 个数），取消了重新排队似乎很不划算。如果作业恰好没在运行 我们是可以 通过 `scontrol` 命令来修改作业的属性。

使用以下命令可以修改 JOBID 任务的部分属性：

```
scontrol update jobid=JOBID ...
```

4.10. 使用作业数组

很多时候我们需要运行一组作业，这些作业所需的资源和内容非常相似，只是一些参数不相同。这个时候借助 Job Array 就可以很方便地批量提交这些作业。Job Array 中的每一个作业在调度时视为独立的作业，仍然受到队列以及服务器的资源限制。

在 SLURM 脚本中使用 `#SBATCH -a range` 即可指定 Job Array 的数字范围，其中的 `range` 参数需要是连续或不连续的整数。下面几种指定方式都是合法的。

- `#SBATCH -a 0-9` 作业编号范围是 0 到 9，均含边界；
- `#SBATCH -a 0,2,4` 作业编号是 0,2,4；
- `#SBATCH -a 0-5:2` 同上，作业编号为 0,2,4；
- `#SBATCH -a 0-9,20,40` 混合指定也是可以的。

在脚本运行中，SLURM 使用环境变量来表示作业数组，具体为

- `SLURM_ARRAY_JOB_ID` 作业数组中第一个作业的 ID；
- `SLURM_ARRAY_TASK_ID` 该作业在数组中的索引；
- `SLURM_ARRAY_TASK_COUNT` 作业数组中作业总数；
- `SLURM_ARRAY_TASK_MAX` 作业数组中最后一个作业的索引；
- `SLURM_ARRAY_TASK_MIN` 作业数组中第一个作业的索引。

可用以上变量来区分不同组内的任务，以便于处理不同的输入参数。

对于每个数组内的作业，它的默认输出文件的命名方式为 `slurm-JOBID_TASKID.out`。

下面是一个很小的 SLURM 脚本例子，它使用 Job Array 来返回一些预设数组中的不同元素。在实际应用中，这些不同的字符串或许就是程序所需输入的文件名。当然您也可以使用一个脚本来包装你的程序，然后在这个脚本中获取这个环境变量。

```
#!/bin/bash
```



```

#SBATCH -J jobarray          # 作业名是 jobarray
#SBATCH -p cpu-low           # 提交到 cpu-low 分区
#SBATCH -N 1                  # 使用一个节点
#SBATCH --cpus-per-task=1     # 每个进程占用一个 cpu 核心
#SBATCH -t 5:00               # 任务最大运行时间是 5 分钟
#SBATCH -o test.out           # 将屏幕的输出结果保存到当前文件夹的 test.out
#SBATCH -a 0-2                # 设置作业数组包含 3 步

input=(foo bar baz)

echo "This is job #${SLURM_ARRAY_JOB_ID}, with parameter
${input[${SLURM_ARRAY_TASK_ID}]}

echo "There are ${SLURM_ARRAY_TASK_COUNT} task(s) in the array."
echo " Max index is ${SLURM_ARRAY_TASK_MAX}"
echo " Min index is ${SLURM_ARRAY_TASK_MIN}"

sleep 5

```

提交上面的作业，使用 `squeue` 查看作业运行情况：

```

[test@ln01 ~]$ sbatch jobarray.slurm

Submitted batch job 94

[test@ln01 ~]$ squeue

```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
94_0	cpu-low	jobarray	test	R	0:11	1	compute-001
94_1	cpu-low	jobarray	test	R	0:11	1	compute-001
94_2	cpu-low	jobarray	test	R	0:11	1	compute-001

取消作业数组的部分或全部任务可以使用 `scancel` 命令。

- `scancel 94_[1-2]`: 取消 94 号作业数组 TASK_ID 为 1 和 2 的作业。
- `scancel 94_0 94_2`: 取消 94 号作业数组 TASK_ID 为 0 和 2 的作业。
- `scancel 94`: 取消 94 号作业数组的全部作业。

5. 应用使用示例

5.1. Ansys

在/gs/script/ansys 下有 Ansys 的提交脚本和测试算例：

```
#!/bin/bash

#SBATCH -J ansys          # 作业名是 ansys
#SBATCH -p cpu-low        # 提交到 cpu-low 分区
#SBATCH -N 2              # 使用 2 个节点
#SBATCH -n 16             # 使用 16 个 CPU 核
#SBATCH --ntasks-per-node 8 # 每个节点使用 8 个 CPU 核
#SBATCH -t 5:00           # 任务最大运行时间是 5 分钟
#SBATCH -o ansys.out       # 将屏幕的输出结果保存到当前文件夹的 ansys.out
#SBATCH -e ansys.err       # 将屏幕的错误输出结果保存到当前文件夹的 ansys.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}

NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

mech_host=""
for host in `sort -u machinefile.${SLURM_JOB_ID}`;do
    n=$(grep -c $host machinefile.${SLURM_JOB_ID})
    mech_host=$(echo "$host:$n:$mech_host")
done
mech_host=$(echo $mech_host|sed "s/:$/ /")

module load ansys/v192
```

```
ansys192 -b -dis -mpi ibmmpi -machines $mech_host -i vm240.txt -o vm240.out
```

5.2. Fluent

在/gs/script/fluent 下有 fluent 的提交脚本和测试算例：

```
#!/bin/bash

#SBATCH -J fluent           # 作业名是 fluent
#SBATCH -p cpu-low         # 提交到 cpu-low 分区
#SBATCH -N 2               # 使用 2 个节点
#SBATCH -n 16              # 使用 16 个 CPU 核
#SBATCH --ntasks-per-node 8 # 每个节点使用 8 个 CPU 核
#SBATCH -t 5:00            # 任务最大运行时间是 5 分钟
#SBATCH -o fluent.out      # 将屏幕的输出结果保存到当前文件夹的 fluent.out
#SBATCH -e fluent.err      # 将屏幕的错误输出结果保存到当前文件夹的 fluent.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}

NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

mech_host=""
for host in `sort -u machinefile.${SLURM_JOB_ID}`;do
    n=$(grep -c $host machinefile.${SLURM_JOB_ID})
    mech_host=$(echo "$host:$n,$mech_host")
done
mech_host=$(echo $mech_host|sed "s/,,$//")
echo $mech_host

module load ansys/v192
module load intel/18.0.3.222
```

```
fluent 2d -g -ssh -pib -mpi=intel -t$NP -cnf="$mech_host" -i inputfile
```

使用 fluent 模块，3d 模式，非图形界面启动，ssh 通讯协议，pib 网络，采用 intel mpi，调用 N 核，N 核分布的主机，输入文件

5.3. CFX

在 /gs/script/cfx 下有 cfx 的提交脚本和测试算例：

```
#!/bin/bash

#SBATCH -J cfx          # 作业名是 cfx
#SBATCH -p cpu-low      # 提交到 cpu-low 分区
#SBATCH -N 2            # 使用 2 个节点
#SBATCH -n 16           # 使用 16 个 CPU 核
#SBATCH --ntasks-per-node 8 # 每个节点使用 8 个 CPU 核
#SBATCH -t 5:00         # 任务最大运行时间是 5 分钟
#SBATCH -o cfx.out       # 将屏幕的输出结果保存到当前文件夹的 cfx.out
#SBATCH -e cfx.err       # 将屏幕的错误输出结果保存到当前文件夹的 cfx.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}

NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

mech_host=""
for host in `sort -u machinefile.${SLURM_JOB_ID}`;do
    n=$(grep -c $host machinefile.${SLURM_JOB_ID})
    mech_host=$(echo "$host*$n,$mech_host")
done
mech_host=$(echo $mech_host|sed "s/,,$//")
echo $mech_host
```

```
module load ansys/v192

cfx5solve -def perf_AirliftReactor.def -start-method "IBM MPI Distributed
Parallel" -par-dist "$mech_host"
```

5.4. VASP

在/gs/script/vasp 下有 vasp 的提交脚本和测试算例：

```
#!/bin/bash

#SBATCH -J vasp          # 作业名是 vasp
#SBATCH -p cpu-low       # 提交到 cpu-low 分区
#SBATCH -N 2             # 使用 2 个节点
#SBATCH -n 16            # 使用 16 个 CPU 核
#SBATCH --ntasks-per-node 8 # 每个节点使用 8 个 CPU 核
#SBATCH -t 5:00          # 任务最大运行时间是 5 分钟
#SBATCH -o vasp.out       # 将屏幕的输出结果保存到当前文件夹的 vasp.out
#SBATCH -o vasp.err       # 将屏幕的输出结果保存到当前文件夹的 vasp.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}

NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

module load intel/18.0.3.222

mpirun -genv I_MPI_FABRICS shm:dapl -np ${NP} -f ./machinefile.${SLURM_JOB_ID}
/gs/software/vasp/vasp.5.4.4/bin/vasp_std          # 执行我的 vasp 程序
```

5.5. Lammps

在/gs/script/lammps 下有 lammps 的提交脚本:

```
#!/bin/bash

#SBATCH -J lammps          # 作业名是 lammps
#SBATCH -p cpu-low         # 提交到 cpu-low 分区
#SBATCH -N 2               # 使用 2 个节点
#SBATCH -n 16              # 使用 16 个 CPU 核
#SBATCH --ntasks-per-node 8 # 每个节点使用 8 个 CPU 核
#SBATCH -t 5:00            # 任务最大运行时间是 5 分钟
#SBATCH -o lammps.out       # 将屏幕的输出结果保存到当前文件夹的 lammps.out
#SBATCH -e lammps.err       # 将屏幕的输出结果保存到当前文件夹的 lammps.err

srun hostname | sort > machinefile.${SLURM_JOB_ID}

NP=`cat machinefile.${SLURM_JOB_ID} | wc -l`

module load lammps/20191030

mpirun -genv I_MPI_FABRICS shm:dapl -np ${NP} -f ./machinefile.${SLURM_JOB_ID}
lmp_mpi
```

6. 技术支持

由于用户手册篇幅有限, 只列出了用户使用系统的基本方法以及常见问题和解决方法, 很难面面俱到, 还请您能够谅解。如果您在系统使用过程中遇到任何问题, 都可以及时与中心技术人员取得联系。中心技术人员会在收到用户问题反馈后的 24 小时工作时间内给予回复。

所有问题均可以在用户交流群中直接提出, 亦可:

1、系统使用、作业运行相关问题联系方式:

Email: hpc@buaa.edu.cn

2、申请账户、机时奖励相关问题联系方式:

Email: hpc@buaa.edu.cn

电话: 82316494

3、平台资费缴纳相关问题联系方式:

电话: 82317005

重点提示: 为了能尽快使您的问题得到定位解决, 请您在通过邮件或电话联系中心技术人员时提供以下基本信息:

系统用户名、作业 jobid、错误现象

包括: 系统用户名, 出错作业号 (jobid), 出错作业日志路径, 作业提交命令, 错误现象描述等信息。

在此, 特别感谢您对我们工作的信任与支持, 同时也祝您在北航校级高算平台使用过程中工作愉快! 希望我们能够共同携手推动我校并行计算技术的发展, 合理使用并行计算资源在各个科研及学科领域不断创新突破!