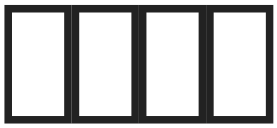

使用手册

使用前请仔细阅读本手册

远程登录

登录方式

远程控制 -登录方式

参考上述远程控制 -登录方式链接内容，并合并覆盖以下内容：

账号密码

1. 账号：集群用户名
2. 密码：集群密码

Web

5. Web支持图形界面 xfwd、命令行界面 ssh，文件传输 sftp

SSH

2. Username: 输入集群用户名 /10.1.0.101/self（如用户名是 yaoge，则输入 yaoge/10.1.0.101/self ）

SFTP

2. Username: 输入集群用户名 /10.1.0.101/self（如用户名是 yaoge，则输入 yaoge/10.1.0.101/self ）

注意事项

4. 登录后用 passwd 命令更改密码，密码至少 8 个字符，包含小写字母、大写字母、数字和特殊字符中至少三种，不能是常见密码

5. [illegible]

VPN

[illegible] VPN

1. [illegible] [illegible] VPN
2. [illegible] VPN[illegible] [illegible]

eScience[illegible] VPN

1. [illegible] VPN Client
2. [illegible] njucm.yaoge123.com
3. [illegible] 4433
4. [illegible]
5. [illegible] VPN[illegible] eScience[illegible]

[illegible]

1. SFTP[illegible] SFTP[illegible]
2. ZMODEM[illegible] rz[illegible] sz[illegible]
3. [illegible]
4. [illegible]

[illegible]

[illegible] ext4/XFS/exFAT [illegible]

- [illegible] ext4/XFS [illegible] Linux[illegible] Windows[illegible] WSL(Windows Subsystem for Linux)[illegible]
- exFAT [illegible] Windows [illegible] Linux [illegible]

集成云盘

独立账号

高性能计算集群用户可以使用集群账号直接登录云盘，用户名为 `集群账号@hpc.nju.edu.cn`，密码即集群账号密码；如集群账号为 `yaoge`，用户名填写 `yaoge@hpc.nju.edu.cn`。首次使用需要登录网页端激活账号，但集群账号不能使用云盘的上下传外链功能。

云盘上的集群账号与南京大学统一身份认证账号相互独立，如集群账号 `yaoge@hpc.nju.edu.cn` 和南京大学统一身份认证账号 `0102003` 是两个完全独立的账号，需要分别激活才能使用。

文件同步

通过云盘的多平台多终端同步功能，可将集群中的目录和本地计算机的目录同步，对本地目录的操作几乎立刻反应在集群的目录中，不再需要通过 SFTP 上下传文件。

- 计算结果输出到集群同步目录中，本地计算机会自动下载，可在本地直接查看和编辑输出文件；
- 计算输入文件放到本地同步目录中，集群会自动下载，直接引用输入文件即可提交作业。

客户端

登录节点已安装云盘同步客户端和挂载盘客户端

- 同步客户端命令行界面: seaf-cli（Linux命令行同步客户端手册）
- 同步客户端图形化界面: seafile-applet
- 挂载盘客户端命令行界面: seadrive（Linux命令行挂载盘手册）
- 挂载盘客户端图形化界面: seadrive-gui

除官方客户端外，也支持 `rclone` 便捷操作。

挂载盘命令行客户端

1. 获取 Token（`<username>` 替换为集群账号，`<password>` 替换为密码）

```
curl -d 'username=<username>@hpc.nju.edu.cn' -d 'password=<password>'
https://box.nju.edu.cn/api2/auth-token/
```

2. 创建配置文件 `~/seadrive.conf`，`<username>` 替换为集群账号，`<token>` 替换为上一步的返回值

```
[account]
server = https://box.nju.edu.cn
username = <username>@hpc.nju.edu.cn
token = <token>
is_pro = true
[general]
client_name = hpc-login
[cache]
size_limit = 10GB
clean_cache_interval = 10
```

3. 启动挂载盘，命令在前台运行，新开一个窗口进行其它操作

```
seadrive -c ~/seadrive.conf -f -d ~/.seadrive/data -l ~/.seadrive/data/logs/seadrive.log ~/SeaDrive
```

4. 如果报错 `Transport endpoint is not connected`，执行: `fusermount -u ~/SeaDrive`
5. `~/SeaDrive` 目录就是云盘挂载在本地的目录，可以直接读写访问

同步命令行客户端

1. 创建客户端本地数据目录（目录名可更改）

```
mkdir ~/Seafile
```

2. 初始化客户端并指定本地数据目录（上一步创建的目录）

```
seaf-cli init -d ~/Seafile
```

3. 启动客户端

```
seaf-cli start
```

4. 列出云盘资料库 ID（`<username>` 替换为集群账号）

```
seaf-cli list-remote -s https://box.nju.edu.cn -u <username>@hpc.nju.edu.cn
```

5. 将本地目录与云盘资料库同步（`<library-id>` 替换为上一步列出的某个资料库 ID，`<folder>` 替换为本地目录）

```
seaf-cli sync -s https://box.nju.edu.cn -u <username>@hpc.nju.edu.cn -l <library-id> -d <folder>
```

将云盘资料库下载到本地目录

```
seaf-cli download -s https://box.nju.edu.cn -u <username>@hpc.nju.edu.cn -l <library-id> -d
<folder>
```

6. 查看当前同步状态

```
seaf-cli status
```

7. 解除本地目录同步

```
seaf-cli desync -d <folder>
```

8. 停止客户端

```
seaf-cli stop
```

使用 rclone 进行云端文件操作

rclone 是一个强大的云存储管理工具，云盘也在其支持的存储类型内。使用 `rclone` 可以更加符合命令行用户使用习惯地去存取云盘文件。

1. （登录后第一次需要）启用 `rclone` 工具

```
module load rclone
```

2. （仅需设置一次）初次使用 `rclone` 需要进行设置。在 `~/.config/rclone/` 目录下创建一名为 `rclone.conf` 的文件，内容（ `<username>` 替换为集群账号）如下：

```
;不一定必须叫 hpc_box，你可以任意设置这个标签，但是后文必须相应地使用该标签
[hpc_box]
type = seafile
url = https://box.nju.edu.cn
user = <username>@hpc.nju.edu.cn
;如果开启了两步认证，下条设置为 true
2fa = false
```

3. （仅需设置一次）设置密码。 `<password>` 替换为密码：

```
rclone config update hpc_box pass <password>
```

4. 设置完毕后，之后按照下方使用即可。列出云端目录：

```
rclone lsd hpc_box:
```

5. 创建云端资料库（ `<library>` 替换为资料库名）：

```
rclone mkdir hpc_box:<library> --seafile-create-library
#例如，rclone mkdir hpc_box:test --seafile-create-library 将在云盘创建一个名为 `test` 的资料库。
```

6. 向云端上传文件。`<destination_path>`替换为需要上传的本地路径，`<source_path>`替换为云端路径:

```
rclone copy <destination_path> hpc_box:<source_path>

#例如，假设本地当前目录下有个 myfile.txt 文件:
#rclone copy myfile.txt hpc_box:test 将把本地当前目录的 myfile.txt 文件上传至云端的 test 资料库下。
#再例如，假设本地当前目录下有个 myfolder 文件夹，里面有很多文件:
#rclone copy myfoler hpc_box:test 将把本地当前目录的 myfolder 文件夹内的所有文件夹和文件（不包括 myfolder 自身）上传至云端的 test 资料库下。
```

7. 从云端下载文件:

```
rclone copy hpc_box:<source_path> <destination_path>

#例如，假设云端资料库 `test`下有个 myfile.txt 文件:
#rclone copy hpc_box:test/myfile.txt . 将把云端资料库 `test`下的 myfile.txt 下载至本地当前目录。
#例如，假设云端资料库 `test`下有个 myfolder 文件夹，里面有很多文件:
#rclone copy hpc_box:test/myfolder . 将把云端资料库 `test`下的 myfolder 文件夹内的所有文件夹和文件（不包括 myfolder 自身）下载至本地当前目录。
```

8. 在传输过程中，如果加上`-P`会显示进度条，强烈建议添加；如果上传或下载过程中，文件较多，建议再加上`--no-traverse`避免列出文件导致传输时间过长:

```
rclone copy hpc_box:<source_path> <destination_path> -P --no-traverse

#例如，假设云端资料库 `test`下有个 myfile.txt 文件:
#rclone copy hpc_box:test/myfolder . -P --no-traverse 将把云端资料库 `test`下的 myfolder 文件夹内的所有文件夹和文件（不包括 myfolder 自身）下载至本地当前目录，且显示进度条，不列出所有文件。
```

9. 更多`rclone`的用法见 rclone - commands; 若有加密资料库等其他云盘需求，或查看`rclone`对云盘的支持功能，请参考 rclone - seafile或 rclone - storage systems。

环境变量

集群使用 Environment Modules 以模块的形式对环境变量进行管理。在高性能计算集群系统中，安装有多种软件及其不同版本，它们需要设置不同的环境变量，Environment Module可以将这些环境变量做成模块文件（modulefile）。模块可以被加载(load)、卸载(unload)、切换(switch)等，这些操作会改变相应的环境变量设置，让用户方便的在不同环境之间切换。相比将环境变量设置写入/etc/profile、~/.bashrc或~/.bash_profile，Environment Module操作只影响当前用户的当前登录环境；相比直接source文件，Environment Module的操作可以撤销（卸载）。普通用户还可以自己编写module，具有很好的定制性。用户不仅可以在命令行中Environment Modules，也能在作业提交脚本中使用，对编译和计算环境都能够很好的控制。

常用命令

- 显示module帮助: module help
- 显示所有可用模块: module avail
- 显示已加载模块: module list
- 加载模块: module load MODULEFILE
- 卸载模块: module unload MODULEFILE
- 切换模块: module switch OLD_MODULEFILE NEW_MODULEFILE (等价于: module unload OLD_MODULEFILE; module load NEW_MODULEFILE)
- 卸载所有已加载模块: module purge
- 显示模块说明: module whatis MODULEFILE
- 显示模块内容: module display MODULEFILE
- 增加搜索目录: module use
- 减少搜索目录: module unuse

非 Bash

使用非 Bash 作为 Login Shell 的用户，请在作业提交脚本的首行指明 Shell 或使用 bsub 选项 -L 指定作业运行时 Login Shell 为 Bash

如 Login Shell 为 tcsh，使用下面两种方法中的一种

```
#!/bin/tcsh #首行指明 Shell
#BSUB ...
...
```

```
...
#BSUB -L /bin/bash #指定运行时        Login Shell为 Bash
...
```

加载模块示例

```
$ module avail #查看所有可用模块

---------------------------- /fs00/software/modulefiles ----------------------------
gcc/5.2.0                       impi/5.0.3.048
iccifort/15.0.3                 ips/2011sp1u3
ics/2013                        ips/2015u3
ics/2013sp1u1                   openmpi/1.10.0-gcc-5.2.0
imkl/11.2.3                     openmpi/1.10.0-iccifort-15.0.3

$ module list #显示已加载模块
No Modulefiles Currently Loaded. #没有模块被加载

$ icc --version #故 icc找不到
-bash: icc: command not found

$ module whatis ips/2015u3 #查看模块说明
ips/2015u3          : Intel Parallel Studio XE 2015 Update 3 Cluster Edition

$ module load ips/2015u3 #加载模块
$ icc --version
icc (ICC) 15.0.3 20150407
Copyright (C) 1985-2015 Intel Corporation.  All rights reserved.

$ module list #显示已加载模块
Currently Loaded Modulefiles:
  1) ips/2015u3

$ module unload ips/2015u3 #卸载模块
```

搜索路径示例

```
$ module use
Search path for module files (in search order):
  /fs00/software/modulefiles

$ module avail
--- /fs00/software/modulefiles ---
gcc/12.1.0
......



$ module use /fs00/software/modulefiles/oneapi/2024.0 #□□   MODULEPATH□□□□

$ module use
Search path for module files (in search order):
  /fs00/software/modulefiles/oneapi/2024.0
  /fs00/software/modulefiles

$ module avail
--- /fs00/software/modulefiles/oneapi/2024.0 ---
mkl32/2024.0
compiler/2024.0.2
mkl/2024.0
......

--- /fs00/software/modulefiles ---
gcc/12.1.0
......
```

[illegible]

[illegible] `bsub` [illegible] `bsub` [illegible]

bsub [illegible]

[illegible]

`bsub [options] command [arguments]`

- `[options]` [illegible] bsub [illegible] CPU[illegible]
- `command` [illegible] MPI [illegible] mpirun [illegible]
- `[arguments]` [illegible]

> [illegible] e5v3ib [illegible] 24[illegible] MPI [illegible]
>
> ```
> $ bsub -q e5v3ib -n 24 "module load oneapi/2024.0/mpi && mpirun ./app"
> Job <3206000> is submitted to queue <e5v3ib>
> ```

[illegible]

```
bsub < jobfile
```

jobfile [illegible] shell [illegible]

```
#BSUB [options]
command [arguments]
```

[illegible] `#BSUB` [illegible] `bsub` [illegible]

例：提交一个作业到 e5v3ib 队列，需要 48核，需要大内存节点，作业名为 MgSiO3，标准输出文件为 out，标准错误输出文件为 err， Intel MPI 的并行作业 0

```
$ cat job.lsf
#BSUB -q e5v3ib
#BSUB -n 48
#BSUB -J MgSiO3
#BSUB -o out
#BSUB -e err
module load ips/2018u4
mpirun ./app

$ bsub < job.lsf
Job <3207099> is submitted to queue <e5v3ib>.
```

等价如下命令行方式

```
$ bsub -q e5v3ib -n 48 -J MgSiO3 -o out -e err "module load ips/2018u4;mpirun ./app"
Job <3207099> is submitted to queue <e5v3ib>.
```

bsub 常用选项

- `-J job_name`：作业名称

资源请求

- `-n min_tasks[,max_tasks]`：作业需要 CPU核数；例：需要四核 `-n 4`；需要 4~8核均可 `-n 4,8`
- `-m`：作业运行的节点或节点组，多个节点写在双引号内并用空格分隔，节点组对应的具体节点可用 `bmgroup` 命令查看，此选项很复杂。在 hostname/hostgroup 前后可用这些符号：后加 `!` 指定头结点、后加 `+[num]` 指定节点使用顺序。例：指定在 c04n01 和 c04n02 运行 `-m "c04n01 c04n02"`；指定可在 f01n01~n03，但是最希望在 f01n01、次希望在 f01n02 `-m "f01n01+2 f01n02+1 f01n03"`；
- `-R "res_req"`：资源请求串，此选项非常复杂；例：有的队列某些节点内存较大，需要大内存节点可以指定 -R largemem
- `-R "select[hname!=host_name]"`：排除 host_name节点，如果要排除多个节点中间用 &&连接，如 `-R "select[hname!=x001 && hname!=x002]"` 排除 x001和 x002节点
- `-x`：作业需要独占节点，无论申请多少核，作业均独占所运行的节点
- `-W [hour:]minute`：作业运行最长时间，超过这个时间则被 kill

CPU绑定

- `-R affinity[core:cpubind=core:membind=localprefer:distribute=pack]`：作业调度系统将进行 CPU 亲和性绑定，注意可能会和程序本身（如 MPI）的绑定冲突，使用前请测试!

自动重运行

- `-r`：如果计算节点或系统故障则自动重新运行作业
- `-Q "exit_code [exit_code ...]"`：根据作业退出码自动重新提交作业。使用空格分隔多个退出码，all 指所有退出码，加 `~` 排除一个或多个退出码。

输入输出

- `-I`：交互式作业，可在作业运行期间和程序进行交互（如输入参数等），可在调试期间使用，正常计算请勿使用
- `-K`：等待作业执行完才返回
- `-i input_file`：标准输入文件
- `-o output_file`：标准输出文件
- `-e error_file`：标准错误输出文件
- 以上三个选项的文件名中可以包含 `%J` 用于表示 JOBID。如果没有用 `-o` 或 `-oo` 指定标准输出文件，那么系统会自动设定为 `output_%J`；如不想要输出文件请设置 `-o /dev/null`

更多选项见 官方文档

GPU 作业

提交作业时使用 `-gpu` 选项申请所需的 GPU 资源，计算进程只可见作业调度系统分配的 GPU。CPU 核自动按照申请节点 GPU 的比例分配，如一节点 8个 GPU 和 40个 CPU 核，申请 2个 GPU 则分配 10个 CPU 核。

`-gpu` 的各个选项用 `:`分隔，默认值为 `num=1:mode=shared:mps=no:j_exclusive=yes`，常用选项如下

- `num=number`：每台主机需要 GPU的数量
- `mode=shared | exclusive_process`： GPU运行模式， `shared` 对应 Nvidia/AMD DEFAULT compute mode， `exclusive_process` 对应 Nvidia EXCLUSIVE_PROCESS
- `mps=yes | no`：开启或关闭 Nvidia Multi-Process Service (MPS)。关闭 MPS，多进程通过时间分片的方式共享 GPU; 开启 MPS, 多进程共享一个 CUDA Context 并发执行，增加了 GPU利用率
- `aff=yes | no`：是否强制进行严格的 GPU-CPU 亲和性绑定，还需要配合 `-R affinity[core:cpubind=core:membind=localprefer:distribute=pack]` 才能一同完成 GPU-CPU亲和性绑定

[illegible]

[illegible]
[illegible] bsub [illegible] -w 'dependency_expression'
[illegible]

- `-w 'done(job_ID | "job_name")'` [illegible] job_ID [illegible] job_name [illegible] DONE [illegible] 0
- `-w 'ended(job_ID | "job_name")` '[illegible] job_ID [illegible] job_name [illegible] EXIT [illegible] DONE
- [illegible] && (AND)[illegible] || (OR)[illegible] ! (NOT)
- [illegible] 1 [illegible]

[illegible] [illegible]

MPI/OpenMP [illegible]

OpenMP (Open Multi-Processing) [illegible] MPI (Message Passing Interface) [illegible] OpenMP [illegible] MPI [illegible]

[illegible] OpenMP [illegible] MPI [illegible] NUMA [illegible] OpenMP [illegible] NUMA [illegible] MPI [illegible] MPI [illegible] NUMA [illegible]

mpirun [illegible] LSB_MCPU_HOSTS [illegible] MPI [illegible] CPU [illegible] MPI/OpenMP [illegible] MPI [illegible]

1. `#BSUB -n` [illegible] CPU [illegible]
2. [illegible]

   ```
   source /fs00/software/lsf/misc/ompthreads.sh [N]
   ```

3. [illegible] MPI [illegible] OpenMP [illegible] `OMP_NUM_THREADS` [illegible] CPU [illegible]

[illegible]

作业运行时

- LSB_JOBID: 作业 ID
- LSB_QUEUE: 队列名称
- LSB_JOBNAME: 作业名称
- LSB_DJOB_NUMPROC: 分配的 CPU总核数
- LSB_DJOB_HOSTFILE: 分配的节点列表文件，每行一个
- LSB_HOSTS: 分配的节点列表，每个 CPU核一个节点名的纯节点列表
- LSB_MCPU_HOSTS: 分配的节点和核数列表，每个节点名和 CPU核数的列表

```
LSB_DJOB_NUMPROC=6
LSB_HOSTS="node1 node1 node1 node2 node2 node2"
LSB_MCPU_HOSTS="node1 3 node2 3"
```

```
$ cat $LSB_DJOB_HOSTFILE
node1
node1
node1
node2
node2
node2
```

> LSB_HOSTS 和 LSB_MCPU_HOSTS 以不同的格式包含相同的信息，LSB_MCPU_HOSTS 比 LSB_HOSTS 更短更精简，如果 LSB_HOSTS 超过 4096 字节，则仅有 LSB_MCPU_HOSTS。

作业脚本示例

串行作业

提交一个串行作业到 e52660 队列，命令行方式和脚本方式分别为：

```
$ bsub -q e52660 ./app
Job <3279929> is submitted to queue <e52660>.
$ cat job.lsf
#BSUB -q e52660
```

```
./app
```

```
$ bsub < job.lsf
Job <3279930> is submitted to queue <e52660>.
```

MPI 并行作业

MPI程序需要使用 `mpirun` 启动

提交一个需要 48核的 Intel MPI 并行作业到 e5v3ib, 命令行方式为:

```
$ bsub -q e5v3ib -n 48 "module load ips/2018u4;mpirun ./app"
Job <3280120> is submitted to queue <e5v3ib>.
```

提交一个需要 48核的 Open MPI 并行作业到 e5v3ib, 脚本方式为:

```
$ cat job.lsf
#BSUB -q e5v3ib
#BSUB -n 48
module load iccifort/15.0.3 imkl/11.2.3 openmpi/1.10.0-iccifort-15.0.3
mpirun ./app

$ bsub < job.lsf
Job <3280122> is submitted to queue <e5v3ib>.
```

OpenMP 并行作业

OpenMP 不能跨节点, 因此 `-n` 不能指定超过一个节点的 **CPU**核数

提交一个需要 64核的 OpenMP 并行作业到 e7v4ib, 使用程序参数 -nt 指定线程数量, 命令行方式为:

```
$ bsub -q e7v4ib -n 64 "./app-nt \$LSB_DJOB_NUMPROC"
Job <3348175> is submitted to queue <e7v4ib>.
```

提交一个需要 64核的 OpenMP 并行作业到 e7v4ib, 使用环境变量 `OMP_NUM_THREADS` 指定线程数量, 脚本方式为:

```
$ cat job.lsf
#BSUB -q e7v4ib
```

```
#BSUB -n 64
OMP_NUM_THREADS="$LSB_DJOB_NUMPROC"
./app

$ bsub < job.lsf
Job <3348182> is submitted to queue <e7v4ib>.
```

MPI/OpenMP 混合作业

每个 MPI进程跑 6个 OpenMP线程

通过环境变量 `OMP_NUM_THREADS` 指定

```
#BSUB -q 6140ib
#BSUB -n 72
export OMP_NUM_THREADS=6
source /fs00/software/lsf/misc/ompthreads.sh
module load ips/2018u4
mpirun ./run
```

通过命令行参数指定（有些计算程序需要通过命令行参数指定线程数量）

```
#BSUB -q 6140ib
#BSUB -n 72
source /fs00/software/lsf/misc/ompthreads.sh 6
module load ips/2018u4
mpirun ./openmx -nt 6
```

GPU 作业

提交一个需要 1个 GPU 的作业到 e5v4p100ib 队列

```
bsub -q e5v4p100ib -gpu num=1 ./gpu_app
```

提交一个需要 4个 GPU 的作业到 62v100ib 队列，进行 GPU-CPU 绑定

```
bsub -q 62v100ib -gpu "num=4:aff=yes" ./gpu_app
```

控制作业

常用控制命令

命令	功能
`bjobs`	查看自己未结束的作业
`bjobs -l JOBID`	查看某个未结束作业的详情
`bhist`	查看自己已结束的历史作业
`bhist -l JOBID`	查看某个已结束历史作业的详情
`bpeek JOBID`	查看正在运行某个作业的 stdout/stderr
`bkill JOBID`	终止某个作业
`btop JOBID`	设置作业最先运行
`bbot JOBID`	设置作业最后运行

作业状态

`bjobs` 命令的作业状态可能值包括:

状态	描述
PEND	作业正在等待中。也就是说，作业尚未开始。
PROV	作业已被派发到一个正在唤醒的节能状态主机。在作业可以发送到 sbatchd 之前，它处于 PROV 状态。
PSUSP	作业在等待期间被挂起，可能是作业所有者或 LSF 管理员操作的。
RUN	作业当前正在运行。
USUSP	作业在运行期间被挂起，可能是作业所有者或 LSF 管理员操作的。
SSUSP	作业被 LSF 挂起。
DONE	作业以状态 0 终止。
EXIT	作业以非零状态终止。

状态	描述
UNKWN	一般是两种情况之一，如果作业状态长时间处于 UNKWN 状态，一般来说就是计算节点坏了可以直接杀掉作业。 ① 因为计算节点负载过高，未能及时获取作业状态导致状态未知，这种情况一般只需要等待即可，待负载下降获取状态后就正常了。 ② 因为计算节点出现故障且长时间未恢复，调度系统无法获取作业状态，此时如果登录不到相应的计算节点，可以直接杀掉作业。
WAIT	对于提交到块作业队列的作业，块作业中的成员正在等待运行。
ZOMBI	①当 sbatchd在执行主机上不可达时，非可重新运行的作业被 `bkill` 杀死，并且作业显示为 UNKWN。 ② 运行可重新运行作业的主机不可用，并且 LSF 已将作业重新排队，分配了新的作业 ID，就像提交了新作业一样。 ③在执行主机可用之后， LSF尝试杀死 ZOMBI作业。 ZOMBI作业成功终止后，作业的状态将更改为 EXIT。 使用 MultiCluster 时，当在远程执行群集上运行的作业变为 ZOMBI作业时，执行群集将像本地 ZOMBI 作业一样处理该作业。此外，它还会通知提交群集作业处于 ZOMBI 状态，并且提交群集将重新排队作业。

作业等待

`bwait -w "wait_condition" [-t timeout]`

暂停并等待作业条件满足，不满足一直暂停等待，满足则执行完毕返回。

典型用法：在脚本中不要循环使用 bjobs判断作业状态，而用 bwait 等待作业运行完成，这样更优雅且能显著降低对集群的压力。

- `-w wait_condition`：要满足的等待条件，此表达式与上述 `bsub -w` 选项的格式相同。
- `-t timeout`：等待条件的超时，范围为 1-525600分钟，默认为一年。

海量作业

作业组

为了便于管理海量的作业，可以对作业定义作业组。

作业组的名字是类似于 Linux的路径名的树状结构，如作业组 /test，而 /test/1 和 /test/2 都属于 /test

需要注意的是作业组是全局的，创建者是这个作业组的拥有者，作业组拥有者可以对这个作业组及其子组内的所有作业进行控制（哪怕这个作业是其他人提交的，拥有者也能进行控制）。如果您不想作业被别人控制，请确保提交到的作业组从 /开始拥有者都是自己。

创建作业组

```
bgadd /test□□□□创建作业组
bsub -g /test□□用  bsub提交作业时指定一个不存在的作业组，则作业组会被隐式创建
```

查看作业组

```
bjgroup /test□□后一列就是作业组的拥有者，请注意拥有者也是层次继承的
```

删除作业组

```
bgdel /test□□□□已经配置自动清理空的隐式创建的作业组
```

有了作业组以后就可以对一组作业进行控制了

```
bjobs -g /test□□□□指定作业组的作业
bkill -g /test 0□□止指定作业组的所有作业
```

作业序列

作业序列是一系列作业，这些作业执行相同的操作和资源要求，但是输入输出文件不同。这些作业共享同一个作业 ID，并可以通过索引来区分每个子作业。

bsub 使用 `-J "arrayName[indexList, ...]"`
参数命名并创建一个作业序列，也可以理解为一个作业数组，中括号里面就是数组的下标，可以是一维数组也

可以是多维数组。 indexList = start[-end[:step]] 下标的起始、结束和步长均可指定。

作业调度系统提供了两个运行时变量 %I 和 %J， %I 为子作业的索引值， %J 为作业 ID ，一般用于输入输出文件名中；以及运行时环境变量 LSB_JOBINDEX

```
bsub -J "myArray[1-10]" myJob□□□一个有 10个子作业的作业
bsub -J "myArray[1-10]" -i "input.%I" -o "output.%I" myJob□□□子作业定义不同的输入输出文件
bkill 123[1]□□掉 jobid 是 123的第一个子作业
bkill 123□□□ jobid 是 123的整个作业
```

海量作业

提交大量的作业，将使得作业的管理将变得困难，虽然可以用上述作业组和作业序列进行批量管理，但是海量作业仍然对调度系统会造成很大的压力，特别是分钟级或秒级的短作业在调度时会浪费大量的时间，因此可以根据作业情况将多个作业合并为一个作业提交。对于不同的作业可以用两种方法来进行合并:

多作业串行执行

申请 1个 CPU 核，顺序执行每个串行作业，前一个运行完成后再运行下一个，可以将非常多的串行作业合并成一个。对于每个串行作业运行时间都很短且运行时间不一定相同的适用这种方式提交。

4个串行作业串行执行合并为一个作业，提交到 x5650队列，脚本方式为:

```
$ cat job.lsf
#BSUB -q x5650
./a.out >& 1.out
./a.out >& 2.out
./a.out >& 3.out
./a.out >& 4.out

$ bsub < job.lsf
Job <3366369> is submitted to queue <x5650>.
```

多作业并行执行

申请 N个 CPU核，同时执行 N个串行作业， N不可大于单节点 CPU 核数，每个串行作业运行时间需要相同，最后需要 wait 命令等待所有作业运行完毕返回。对于每个串行作业运行时间较长且运行时间都完全相同的适用这种方式提交

[illegible]

12[illegible] x5650[illegible]

```
$ cat job.lsf
#BSUB -q x5650
#BSUB -n 12
( ./a.out >& 1.out )&
( ./a.out >& 2.out )&
( ./a.out >& 3.out )&
( ./a.out >& 4.out )&
( ./a.out >& 5.out )&
( ./a.out >& 6.out )&
( ./a.out >& 7.out )&
( ./a.out >& 8.out )&
( ./a.out >& 9.out )&
( ./a.out >& 10.out )&
( ./a.out >& 11.out )&
( ./a.out >& 12.out )
wait

$ bsub < job.lsf
Job <3366370> is submitted to queue <x5650>.
```

查看信息

查看队列

查看所有队列: `bqueues`

```
$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
x7542!           50  Open:Active       -    -    -    -    24     0    24     0
e5645!           50  Open:Active       -    -    -    -     0     0     0     0
e52643tgb!       50  Open:Active       -    -    -    -     8     0     8     0
......
6226rib          30  Open:Active       -    -    -    -     0     0     0     0
5218             30  Open:Active       -    -    -    -     0     0     0     0
6230r            30  Open:Active       -    -    -    -    32     0    32     0
```

- `QUEUE_NAME`: 队列名称
- `PRIO`: 队列优先级，越大优先级越高
- `STATUS`: 队列状态。 Open/Closed 表示是否可以提交，即用户是否可以提交作业到该队列; Active/Inact 表示否可以派发，即该队列的作业是否会被分发到计算节点运行。 Open:Active 表示可提交可派发， Open:Inact 表示可提交但是不派发。
- `NJOBS`: 排队、运行和挂起的作业所占总 CPU核数
- `PEND`: 排队中的作业所需总 CPU核数
- `RUN`: 运行中的作业所占总 CPU核数
- `SUSP`: 挂起的作业所占总 CPU核数

查看队列详细信息: `bqueues -l`

```
$ bqueues -l e5v3ib


QUEUE: e5v3ib
  -- CPU: 2*E5-2680v3, RAM: 256GB/128GB, NET: 56Gb FDR InfiniBand


......


SCHEDULING POLICIES:  FAIRSHARE  EXCLUSIVE
```

FAIRSHARE_QUEUES: e5v3ib e5v3ib! e7v4ib x5650ib 6140ib 62v100ib 722080tiib 72rtxib 7702ib ……
DISPATCH_ORDER: QUEUE
USER_SHARES: [root=, 999999] ……

SHARE_INFO_FOR: e5v3ib/
 USER/GROUP   SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME   ADJUST GPU_RUN_TIME
root=   999999 202255.328   0   0   8456.5   1542   0.000   0
……

USERS: all ~test/
HOSTS: f01+10 f02+10 f03+10 f04+10 f05s+10 f05l/
RES_REQ: span[ptile=24]
Maximum slot reservation time: 43200 seconds

查看节点

lshosts□□□□#点配置和资源

lshosts -gpu□#看节点　　GPU配置和拓扑结构

lsload□□□□#点当前负载信息

lsload -gpu□□□#节点　　GPU整体负载

lsload -gpuload□#看节点每个　　GPU负载

bhosts□□□□#有节点状态

自动关机

集群会对动力环境进行监控，遇市电中断或温度过高，将会自动终止所有作业，按照安全顺序进行关机操作。

关机时会在 `/fs00/reports/bjobs/` 目录下会自动保存一份作业列表备查。如文件 /fs00/reports/bjobs/bjobs.20130728070457 表明 2013年 07月 28日 07点 04分 57秒时刻所有作业的状态（ `bjobs -uall -w` 的输出），同时也说明这个时间点开始自动关机。

[illegible]

[illegible]

[illegible]

[illegible]
/fs00/reports/process [illegible]

- [illegible]
- [illegible] CPU[illegible] CPU[illegible]
- [illegible] 2[illegible] CPU[illegible]

容器化

容器技术

容器技术能够对应用及其整个运行时环境（包括全部所需文件）一起进行打包或隔离。从而可以在不同环境（如开发、测试和生产等环境）之间轻松迁移应用，同时还可保留应用的全部功能。有了容器不再需要管理员为你安装任何东西。容器是一个独立的系统，你可以在里面做任何想做的事情。

Apptainer

2021年 11月， Singularity 开源项目加入 Linux 基金会，并更名为 Apptainer。Apptainer 是一套类似于 Docker 的容器解决方案，是用于 HPC容器系统。 Apptainer 兼容 Singularity Image File (SIF)、Singularity 的环境变量、 Singularity 的命令、自动迁移 Singularity 的用户配置。

镜像

一般保存为压缩只读 Singularity Image File (SIF)格式的镜像文件。

公共镜像

在 `/fs00/software/singularity-images/` 已存放了许多常用的 SIF镜像文件，可直接使用。

导入镜像

以从 Docker Hub 等镜像仓库或镜像文件直接创建 SIF镜像文件

- 从 Docker Hub 导入镜像: `apptainer build ubuntu.sif docker://docker.nju.edu.cn/library/ubuntu`
- 从 NVIDIA NGC 导入镜像: `apptainer build ngc_cuda.sif docker://ngc.nju.edu.cn/nvidia/cuda`
- 从 docker save 保存的镜像文件导入镜像: `apptainer build abc.sif docker-archive://abc.tar`

制作镜像

1. 创建沙盒目录: `apptainer build --fix-perms --sandbox build docker://docker.nju.edu.cn/library/ubuntu`
2. 进入沙盒容器: `apptainer shell build/`
3. 在容器中安装和编译软件：如 `apt` `make`
4. 退出容器: `exit`
5. 将沙盒打包成 SIF镜像文件: `apptainer build abc.sif build`

因登录节点无 root权限，可能出现问题，因此建议在自己的系统上制作镜像。

CI创建镜像

用 git.nju.edu.cn 上的 CI/CD 自动调用 kaniko 构建 Docker 镜像，详见 官方文档，然后再导入 `apptainer build ocr.sif docker://reg.nju.edu.cn/yaoge123/ocr`

一个简单的示例见: CI/CD自动化构建 Docker镜像

提交作业

提交容器作业的示例脚本

```
#BSUB -q 62v100ib
#BSUB -gpu num=4


apptainer exec --nv cuda.sif app
```

常用选项

- `--nv`: 提供 NVIDIA GPUs & CUDA支持
- `--bind/-B src[:dest[:opts]]`: 绑定额外的路径

相关网站

- Apptainer
- Apptainer User Guide
- Docker Hub
- NVIDIA NGC

常见问题

网络

登录节点进行网络接入认证

在登录节点登录校园网账号

集群中的登录节点进行网络接入认证（即登入南京大学校园网认证 `p.nju.edu.cn` ）后才可访问互联网，认证后登录节点 所有用户均可访问互联网（相当于整个登录节点共享），请注意网络与账号隐私安全!

命令行登录和登出 `p.nju.edu.cn` 的方法有:

```
curl -X POST https://p.nju.edu.cn/api/portal/v1/login -H "Content-type: application/json" -d '{"username":"<username>","password":"<password>"}'
curl -X POST https://p.nju.edu.cn/api/portal/v1/logout -H "Content-type: application/json" -d '{}'
```

```
curl -s "http://p2.nju.edu.cn/portal_io/login?username=<username>&password=<password>"
curl -s http://p2.nju.edu.cn/portal_io/logout
```

**【强烈推荐】 **如果有一些软件源需求，如 `conda` 、 `pip` 等包管理，或者需要外部代码库、容器，您可以通过中心提供的下列服务直接在集群上使用，无需登录校园网认证:

- `mirror.nju.edu.cn` : `conda` 、 `pip` 等大量常用软件源或代码仓库镜像
- 代码托管_____: 可手动镜像各大在线 git仓库作为中转
- 私服仓库_____: 各软件源、容器缓存

计算节点访问网络

所有计算节点均不能自由无限制的访问外网（含校园网和互联网），如需访问可以单独申请开放。当前已经放开的网络访问

https://table.nju.edu.cn/dtable/view-external-links/custom/hpc-internet/

SSH/SFTP登录时报 no matching host key type found

SSH/SFTP登录时报错 `no matching host key type found. Their offer: ssh-rsa,ssh-dss`，命令行添加选项 `-o HostKeyAlgorithms=+ssh-rsa` 或在 `~/.ssh/config` 中增加 `HostKeyAlgorithms +ssh-rsa`

安装

pip安装包到自己的目录下

Python的大多数包不需要 root权限也能安装，只需在 pip install后加 -t指定安装目录即可，如:

```
cd scikit-opt-master
pip install -t $HOME .
```

这样就装到自己的家目录下。在 ~/.bashrc里或者作业脚本中加上环境变量

```
export PYTHONPATH=$HOME:$PYTHONPATH
```

作业

作业运行时实际占用 CPU核数过多

1. 在作业中限定使用的 CPU核数与申请核数相同，否则超过申请核数使用资源的作业会被杀掉。

特别是 Python，很多 Python包会自动多核并行计算，需要使用环境变量（如 `OMP_NUM_THREADS`、`NUMEXPR_NUM_THREADS`、`OPENBLAS_NUM_THREADS`、`MKL_NUM_THREADS`）等方式设定线程数，与进程数一起匹配申请的核数。

```
export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
export NUMEXPR_NUM_THREADS=$LSB_DJOB_NUMPROC
export OPENBLAS_NUM_THREADS=$LSB_DJOB_NUMPROC
export MKL_NUM_THREADS=$LSB_DJOB_NUMPROC
```

2. 可以尝试在提交作业时添加如下参数，做 CPU亲和绑定

```
#BSUB -R affinity[core:cpubind=core:membind=localprefer:distribute=pack]
```

3. 如果实在不行可以 `#BSUB -x` 独占节点运行作业

排队作业数上限

动态限制用户排队作业数不能超过 MAX(1000,MIN((30000-总作业数)/10,(20000-总排队作业数)/6))

即用户排队作业数，不超过三万减去总作业数的十分之一，也不超过二万减去总排队作业数的六分之一，但保底一千个。

作业状态 UNKWN

一般是两种情况之一，如果作业状态长时间处于 UNKWN状态，一般来说就是计算节点坏了可以直接杀掉作业。

1. 因为计算节点负载过高，未能及时获取作业状态导致状态未知，这种情况一般只需要等待即可，待负载下降获取状态后就正常了。
2. 因为计算节点出现故障且长时间未恢复，调度系统无法获取作业状态，此时如果登录不到相应的计算节点，可以直接杀掉作业。