



地球与空间科学系

DEPARTMENT OF EARTH AND SPACE SCIENCES

地空系并行集群使用说明

TCHE LIU* GONGHENG ZHANG[†]

2019 年 4 月 22 日

Version: 0.2

*tcheliu@mail.ustc.edu.cn

[†]11749276@mail.sustc.edu.cn

目录

1 注意事项	3
2 LSF 作业调度系统	3
2.1 集群队列设置	3
2.2 查询系统运行情况	3
2.3 LSF 作业提交	4
2.3.1 作业提交脚本的书写	4
2.3.2 依赖作业提交	5
2.4 LSF 作业管理	5
2.4.1 LSF 作业状态	5
2.4.2 LSF 作业调度	6
2.5 参考阅读	7
3 Module Environment 的使用	7
3.1 系统软件环境	7
3.1.1 系统软件安装目录	7
3.1.2 编译器及解释器	7
3.1.3 运行库	8
3.1.4 工具	9
3.2 module 的使用	9
3.2.1 搜索路径 MODULEPATH	10
3.2.2 自定义 modulefile 文件	10
3.2.3 module 模块命令	11
3.2.4 参考阅读	11
4 常见问题解决	11
4.1 作业运行问题	11

一、注意事项

- (1) 严禁在任何节点直接运行并行程序，所有多核任务必需经由作业系统提交运行；
- (2) 提交作业时务必注意节点的内存限制，切勿提交超负荷任务；
- (3) 提交作业时烦请注意队列的提交限制，以防提交无效作业；
- (4) 用户 HOME 目录所在磁盘 share 为全节点共享，该磁盘所有用户的用户配额为 1TB，当用户的总数据量超过该限制时，该用户将无法创建新文件；
- (5) 请注意及时备份集群上的重要数据，本集群不保证用户数据的安全性；
- (6) 为规范使用，仅允许用户在其作业运行期间，ssh 登录到相应的计算节点查看作业运行情况，其他时间已设置为禁止登录；
- (7) 本集群虽安装有 matlab 等绘图软件，但不推荐在集群上直接成图，建议将数据拷贝到本地机器上，再利用本地机器作图。

二、LSF 作业调度系统

2.1 集群队列设置

表 1: 集群队列设置

节点	队列	核数	内存	作业最长运行时间	优先级	交互式提交
mn01	dataq	20	62.1G	-	33	禁止
c001-c028	short	28	125G	12:00	40	允许
c001-c028	long	28	125G	无限制	30	允许
c029-c042	large	24	188G	48:00	30	禁止
s001-s002	smp	72	508G	24:00	20	禁止

注意，mn01 为主节点，禁止在此节点运行作业。

在设置程序使用每节点核数时，尽量设置为节点总核数的整除数。

2.2 查询系统运行情况

通过命令 `lnodes`，可查看系统各节点连接情况。

通过命令 `lshosts`，可查看系统各节点硬件配置情况。

通过命令 `lsload`，可查看系统各节点当前负载。

通过命令 `lsmon`，可动态查看系统各节点当前负载。

通过命令 `bhosts`，可查看系统各节点作业运行情况。

通过命令 `bqueues`，可查看系统中各队列作业运行情况。

通过命令 `busers all`，可按用户查看所有用户的作业运行整体情况。

通过命令 `bjobs -u all`，可按作业查看所有用户的作业运行详细情况。

通过命令 `bhist -t -T <起始时刻, 终止时刻>`，可按时间顺序查看某一时间段内作业系统的调度历史。

2.3 LSF 作业提交

2.3.1 作业提交脚本的书写

```
#!/bin/bash
#BSUB -J MPIJob          ### set the job name
#BSUB -q short           ### specify queue
#BSUB -n 40              ### ask for number of cores (default:1)
5 #BSUB -R "span[ptile=20]"  ### ask for 20 cores per node
#BSUB -W 10:00           ### set walltime limit: hh:mm
#BSUB -o std_%J.out       ### specify the output and error file. %J is the job-id
#BSUB -e std_%J.err       ### -o and -e mean append, -oo and -eo mean overwrite

10 # here follow the commands you want to execute

# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
15 module load mpi/intel/2018.4

### This uses the LSB_DJOB_NUMPROC to assign all the cores reserved
### This is a very basic syntax.
mpirun -np $LSB_DJOB_NUMPROC ./Your_MPI_Program
```

Listing 1: MPI 并行作业提交脚本示例

```
#!/bin/bash
#BSUB -J OpenMPJob       ### set the job name
#BSUB -q short           ### specify queue
#BSUB -n 20              ### ask for number of cores (default: 1)
5 #BSUB -R "span[hosts=1]"  ### specify that the cores MUST BE on a single host!
#BSUB -m "c001"          ### specify the host that the job will run on
#BSUB -W 10:00           ### set walltime limit: hh:mm
#BSUB -o std_%J.out       ### specify the output and error file. %J is the job-id
#BSUB -e std_%J.err       ### -o and -e mean append, -oo and -eo mean overwrite

10 # set OMP_NUM_THREADS and export!
export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC

# run your program
15 ./Your_OpenMP_Program [options]
```

Listing 2: OpenMP 并行作业提交脚本示例

在书写作业提交脚本时，LSF 作业系统主要的解析制导指令有：

- (1) `#BSUB -J <作业名>`：指定作业名；
- (2) `#BSUB -q <队列名>`：指定排队队列，此项不指定则调用默认队列；
- (3) `#BSUB -n <申请总核数>`：指定总核数，此项不指定则总核数默认为 1；
- (4) `#BSUB -R "<描述信息>"`：指定资源限制，这里的描述信息可以为：
`span[ptile=<每个节点的核数>]` 指定每节点使用核数、
`span[hosts=1]` 指定使用一个节点、
`rusage[mem=<每个节点的内存限制>]` 指定每核可用内存最低限制、
等内容；
- (5) `#BSUB -m "<节点列表>"`：指定作业运行节点，多个节点以空格分隔；
- (6) `#BSUB -W <挂钟时间>`：指定作业最长运行时间，时间格式为 hh:mm；
- (7) `#BSUB -o <标准输出文件名>`：指定标准输出文件；
- (8) `#BSUB -e <错误输出文件名>`：指定标准错误输出文件；

另外注意，在编译程序过程中若使用了特别版本的编译器或运行库，需在作业提交脚本中通过命令 `module load <模块名>` 加载这些编译器或运行库环境模块。

2.3.2 依赖作业提交

通过在作业提交脚本头部中加入 `#BSUB -w "<复合作业状态>(<作业号>)"` 的内容，可以实现相互依赖的多个作业任务的提交。当指定作业达到指定状态时，当前作业才会被派发。这里的复合作业状态可以设置为：

- (1) `done`: 基本作业状态 DONE；
- (2) `ended`: 基本作业状态 EXIT 或 DONE；
- (3) `exit`: 基本作业状态 EXIT，可通过 `(<作业号>,[操作符]<退出码>)` 指定特殊的退出码，这里的操作符可以设为 `>`、`>=`、`<`、`<=`、`==` 或 `!=`；
- (4) `started`: 基本作业状态 DONE、EXIT、USUSP、SSUSP 或 RUN。

基本作业状态参见下文内容。

通过命令 `bjdepinfo <作业号>`，可查看已提交作业的依赖关系。

2.4 LSF 作业管理

2.4.1 LSF 作业状态

LSF 调度系统的基本作业状态有：

- (1) PEND: 在系统中排队，等待派发；
- (2) RUN: 已经被派发，在运行中；
- (3) DONE: 正常运行完毕，程序退出码为 0；
- (4) PSUSP: 被用户自己或管理员在派发前挂起；
- (5) USUSP: 被用户自己或管理员在派发后挂起；
- (6) SSUSP: 被 LSF 系统在派发后挂起；
- (7) EXIT: 被异常终止，或程序退出码非 0。

被用户自己或管理员挂起的作业不会被系统自动派发；被系统挂起的作业待满足运行条件后系统会自动派发。

作业被系统挂起的原因有：该作业依赖于其他待完成作业、该作业在队列中的优先级较低、没有足够的计算资源等。

2.4.2 LSF 作业调度

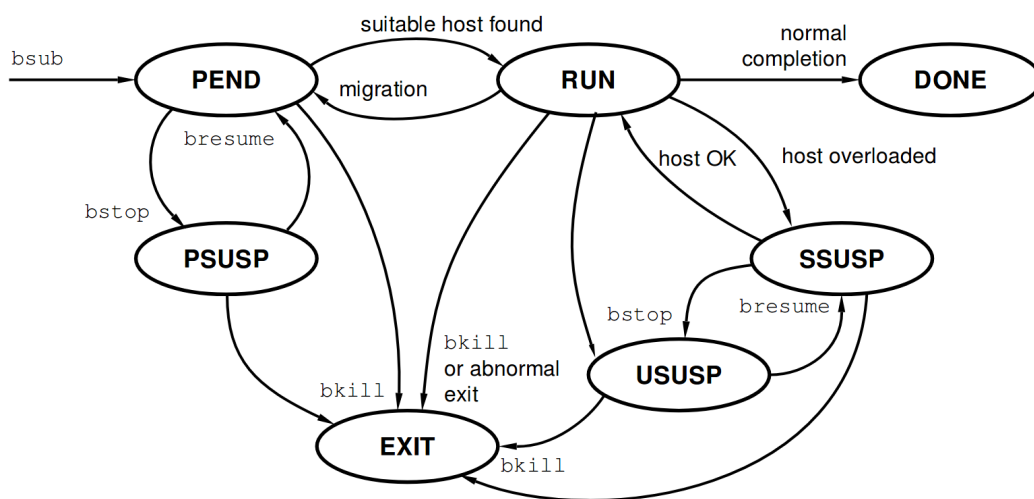


图 1: LSF 基本作业状态

参照前述示例正确书写作业提交脚本 `job.lsf` 后，即可通过命令 `bsub <job.lsf>` 提交作业。正确提交作业后，该命令会返回系统分配的作业号。区别于 PBS 作业系统，注意这里的“<”是必需的。

作业提交后，通过命令 `bjobs -l <作业号>`，可以查看作业状态。

作业提交后，通过命令 `bstop <作业号>`，用户可以将自己的正在排队（**PEND** 状态）的作业挂起（**PSUSP** 状态）；此时，可以通过命令 `bresume <作业号>`，将挂起（**PSUSP** 状态）的作业重新参与系统排队（**PEND** 状态），等待系统派发。

作业开始运行后，同样可以通过 `bstop` 命令，用户可以将自己的正在运行（RUN 状态）的作业挂起（USUSP 状态）；同样地，通过 `bresume` 命令，再次将挂起（USUSP 状态）的作业重新参与系统排队（SSUSP 状态），等待系统派发。

通过命令 `bkill <作业号>`，用户可以将自己待运行或正在运行的作业取消（EXIT 状态）。

通过命令 `bkill -s <系统消息代码> <作业号>`，用户可以向自己的正在运行的作业发送系统消息。

通过命令 `bhist -l <作业号>`，用户可以查看指定作业的系统调度历史。

通过命令 `bpeek <作业号>`，用户可以查看自己的未完成作业的标准输出和错误输出。

在作业完成后，LSF 系统会保留作业记录 5 分钟，此时，还可以通过作业号查看作业的运行记录。

2.5 参考阅读

LSF 用户手册: http://hpc.sustech.edu.cn/ref/lsf_users_guide_v10.1.pdf

太乙用户手册: http://hpc.sustech.edu.cn/ref/TaiYi_User_Manual_v0.1.pdf

IBM Spectrum LSF V10.1 documentation: https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_welcome.html

LSF Batch User's Guide: <https://ls11-www.cs.tu-dortmund.de/people/hermes/manuals/LSF/users.pdf>

三、Module Environment 的使用

3.1 系统软件环境

3.1.1 系统软件安装目录

系统软件的安装目录主要有：

/share/apps: 安装了 python 和 java 的解释器；

/share/base: 安装了大部分的运行库和一部分工具；

/share/intel: 安装了 intel 编译器及附带工具；

/share/tools: 安装了一些工具。

3.1.2 编译器及解释器

系统安装了多个版本的 C 和 Fortran 语言编译器：

系统默认的 gcc/4.8.5 （系统标准安装目录）、

gcc/8.2.0 （/share/base/gcc/8.2.0 目录）、

intel/2017.8 （/share/intel/2017u8 目录）、

intel/2018.3 (/share/intel/2018u3 目录)、
intel/2018.4 (/share/intel/2018u4 目录)。

系统安装了多个版本的 Python 语言解释器：

python/2.7.15 (/share/apps/python/2.7.15 目录)、
python/3.7.0 (/share/apps/python/3.7.0 目录)、
python/anaconda2/5.2.0 (/share/apps/anaconda2/5.2.0 目录)、
python/anaconda3/5.2.0 (/share/apps/anaconda3/5.2.0 目录)、
python/intelpython2/2018.3.039 (/share/apps/ipython/2018.3.039/intelpython2 目录)、
python/intelpython2/2019.9.047 (/share/apps/ipython/2019.9.047/intelpython2 目录)、
python/intelpython3/2018.3.039 (/share/apps/ipython/2018.3.039/intelpython3 目录)、
python/intelpython3/2019.9.047 (/share/apps/ipython/2019.9.047/intelpython3 目录)。

系统安装了一个版本的 matlab 语言解释器：

matlab/R2018b (/share/apps/matlab/Matlab 目录)。

系统安装了一个版本的 go 语言编译器：

go/1.11.1 (/share/base/go/1.11.1 目录)。

系统安装了两个版本的 java 语言解释器：

java/1.8.0_181 (/share/apps/java/1.8.0_181 目录)、
java/10.0.2 (/share/apps/java/10.0.2 目录)。

3.1.3 运行库

系统安装的运行库大部分都同时有 gcc/4.8.5 和 intel/2018.4 两个编译器的版本。

系统安装的 MPI 并行库有：

mpi/intel (在各自对应版本的 intel 编译器安装目录下)、
mpi/mpich (/share/base/mpich 目录)、
mpi/openmpi (系统标准安装目录或 /share/base/openmpi 目录)。

系统安装的其他第三方运行时函数库主要有：

zlib (/share/base/zlib 目录)、
fftw (/share/base/fftw 目录)、
hdf5 (/share/base/hdf5 目录)、
netcdf-c (/share/base/netcdf-c 目录)、
netcdf-fortran (/share/base/netcdf-fortran 目录)、
netcdf-cxx (/share/base/netcdf-cxx 目录)、
BLAS (/share/base/netlib/BLAS 目录)、
LAPACK (/share/base/netlib/LAPACK 目录)、
petsc (/share/base/petsc 目录)。

3.1.4 工具

系统安装的专业工具有：

SAC/101.6a (/share/tools/SAC/101.6a 目录)。

系统安装的绘图工具有：

GMT/4.5.18 (/share/tools/GMT/4.5.18 目录)、

GMT/5.4.5 (/share/tools/GMT/5.4.5 目录)。

系统安装的软件编译安装工具有：

系统默认的 make/3.82 (系统标准安装目录)、

make/4.2 (/share/tools/make/4.2 目录)、

系统默认的 cmake/2.8.12.2 (系统标准安装目录)、

cmake/3.12.2 (/share/base/cmake/3.12.2 目录)、

meson/0.50.0 (/share/tools/meson/0.50.0 目录)、

ninja/1.9.0 (/share/tools/ninja/1.9.0 目录)。

系统安装的其他辅助工具有：

系统默认的 git/1.8.3.1 (系统标准安装目录)、

git/2.18.0 (/share/base/git/2.18 目录)、

htop/2.2.0 (/share/base/htop/2.2.0 目录)、

parallel/20180922 (即 GNU Parallel, /share/base/parallel 目录)、

ghostscript/9.26 (/share/tools/ghostscript/9.26 目录)、

feh/3.1.3 (/share/tools/feh/3.1.3 目录)、

valgrind/3.14.0 (/share/tools/valgrind/3.14.0 目录)、

ImageMagick/7.0.8-39 (/share/tools/ImageMagick/7.0.8-39 目录)、

autojump/22.5.3 (/share/tools/autojump/22.5.4 目录, 在加载此模块后, 还需再执行命令 `source $AUTOJUMPROOT/.autojump/etc/profile.d/autojump.sh`, 方可调用 `j` 命令快速跳转目录)。

3.2 module 的使用

当在 Linux 下存在多个版本的同一个编译器或运行库时, 如果每次编译都写上绝对路径就很麻烦, 使用 `module-environment` (以下简称 `module`) 来管理环境变量则相比方便很多。

例如, 在我们服务器中安装了多个版本的 intel 编译器, 包括: intel/2017.8、intel/2018.3 和 intel/2018.4。在调用 intel/2017.8 时, 一般先要修改 `PATH` 和 `LD_LIBRARY_PATH` 等系统环境变量。此时若要改为调用 intel/2018.4, 又要再次修改这些环境变量, 容易造成混乱。使用 `module` 则可以通过方便地 `modulefile` 进行系统环境变量的配置。

3.2.1 搜索路径 MODULEPATH

module 的搜索路径由系统环境变量 MODULEPATH 定义，在 MODULEPATH 包含的路径下的 modulefile 文件，可以自动被 module 识别。非 root 用户可以通过修改 MODULEPATH 加入自己的 modulefile 目录，从而实现通过 module 来管理自己在 home 目录下单独安装的软件。

3.2.2 自定义 modulefile 文件

在自定义 modulefile 文件时需注意，文件必需以 `##Module` 开头，这样才会被 module 识别为 modulefile 文件。

常用的 modulefile 命令有：

- (1) `module-whatis` "模块描述"：添加对该 module 的描述信息；
- (2) `conflict` < 模块名 >：如果这里指定的模块已被加载，当前定义的模块将不会被加载；
- (3) `set` < 变量名 > < 值 >：设置变量，这里定义的变量仅在该 modulefile 文件中生效；
- (4) `setenv` < 系统环境变量名 > < 值 >：设置系统环境变量，将系统环境变量重置。这里定义的变量将在加载该模块后对整个系统生效；
- (5) `prepend-path` < 系统环境变量名 > < 新添值 >：设置系统环境变量，在原变量值前添加新的内容。

```
##Module

conflict mpi

5 set version 3.1.1rc1

set OPENMPIROOT "/usr/mpi/gcc/openmpi-${version}"

module-whatis "Enable usage for openmpi version ${version}"

10 setenv MPI_ROOT "${OPENMPIROOT}"

#use 'normal' compact core binding as a default
setenv OMPI_MCA_rmaps_base_mapping_policy core
15 setenv OMPI_MCA_hwloc_base_binding_policy core

prepend-path PATH "${OPENMPIROOT}/bin"
prepend-path INCLUDE "${OPENMPIROOT}/include"
prepend-path LD_LIBRARY_PATH "${OPENMPIROOT}/lib"
20 prepend-path MANPATH "${OPENMPIROOT}/share/man"
```

Listing 3: modulefile 文件书写示例

更多示例参见服务器上 `/share/base/modulefiles/` 目录。

3.2.3 module 模块命令

module 模块命令有：

- (1) `module avail [模块名]`：查看系统可用的模块；
- (2) `module list`：查看已经加载的模块；
- (3) `module whatis [模块名]`：查看模块描述信息；
- (4) `module load <模块名>`：加载指定模块；
- (5) `module unload <模块名>`：卸载指定模块；
- (6) `module purge`：卸载所有已加载的模块。

例如，我们要调用 intel/2017.8 来编译程序，可先由命令 `module load intel/2017.8` 加载该软件模块环境后，即可直接通过命令 `icc/ifort` 调用该模块的编译器来编译程序。

此时，若想换用 intel/2018.4 模块，可先由命令 `module unload intel/2017.8` 卸载 intel/2017.8 模块后，再由命令 `module load intel/2018.4` 加载 intel/2018.4 模块后即可。

另外，在调用 mpi/intel 模块时请注意，mpicc/mpif90 实际最终调用的还是 GNU 编译器；若想调用 intel 编译器，请使用 *mpiicc/mpiiifort* 命令，此时还需额外加载相应的 intel 编译器模块。

3.2.4 参考阅读

[module 命令手册](https://modules.readthedocs.io/en/stable/module.html)：https://modules.readthedocs.io/en/stable/module.html

[modulefile 配置手册](https://modules.readthedocs.io/en/stable/modulefile.html)：https://modules.readthedocs.io/en/stable/modulefile.html

[module 简单使用](https://blog.csdn.net/jslove1997/article/details/80338370)：https://blog.csdn.net/jslove1997/article/details/80338370

四、常见问题解决

4.1 作业运行问题

若使用 mpi/intel/* 模块编译运行并行程序，程序异常退出并报错 “unexpected DAPL connection event 0x4008” 时，可尝试使用其他版本的 MPI 库来编译运行。