BME集群使用手册

1. BME 高性能集群使用规范
    1.1 系统架构
    1.2 使用规范
    1.3 用户资源配置（重要）
2. 账户申请
    2.1 高性能账号申请
3. 账户登录
    3.1 登录节点介绍
    3.2 集群账户登录办法
4. 文件传输
    4.1 文件上传下载工具
    4.2 Windows操作系统实现方式
        4.2.1 Xftp
        4.2.2 **Windows ssh登陆服务器后使用rz和sz命令**
        4.2.3 Filezilla
        4.2.4 winscp
    4.3 **Linux和MacOS操作系统直接使用命令**
        4.3.1 scp
        4.3.2 sftp
        4.3.3 XQuartz
5. 常用Linux命令及操作
    5.1 ls
    5.2 mkdir
    5.3 cd
    5.4 pwd
    5.5 touch
    5.6 vi/vim
    5.7 echo
    5.8 cat
    5.9 tac
    5.10 paste
    5.11 sort
6. 集群硬件架构
    6.1 BME 计算节点配置信息
    6.2 集群要素介绍
    6.3 用户登录拓扑
7. 作业提交及变更
    7.1 job array在Torque作业调度系统的实现
        7.1.1 确定pbs对用户计算任务的适用性
        7.1.2 什么是PBS job array?
        7.1.3 如何在高性能计算系统下实现PBS job array？
        7.1.4 如果监测PBS job array状态？
        7.1.5 如何删除PBS job array?
        7.1.6 如何查看job array的屏幕输出？
    7.2 job array在Slurm作业调度系统的实现
        7.2.1 Slurm任务脚本示例：
        7.2.2 如何提交Slurm任务？
        7.2.3 如何监测Slurm任务状态？
        7.2.4 如何取消Slurm任务
        7.2.5 如何查看Slurm调度系统输出？
        7.2.6 References
    7.3 **PBS调度系统上的python程序Job Array使用指南**
        7.3.1 什么是job array?
        7.3.2 什么时候需要使用job array?
        7.3.3 如何在PBS调度系统上实现Job Array?
        7.3.4 如何监测PBS job array的运行状态？
        7.3.5 如何删除PBS job array?
        7.3.6 如何检查job array的输出
    7.4 **SLURM调度系统上的R程序Job Array使用指南**

7.4.1 什么是job array?
7.4.2 什么时候需要使用Job Array?
7.4.3 如何在Slurm调度系统上实现job array?
7.4.4 如何监测Slurm job array的运行状态？
7.4.5 如何删除Slurm job array?
7.4.6 如何检查job array的输出
8. 集群软件使用及常见问题
8.1 Freesurfer
8.1.1 软件介绍
8.1.2 下载及安装
8.1.3 软件使用
8.1.3.1 转化.mgz文件到nifti文件
8.1.3.2 Recon-all
8.1.3.3 可视化输出
8.1.4 参考资料
8.2 MATLAB
8.2.1 **概述**
8.2.2 MATLAB加载过程
8.2.3 并行计算的使用说明
8.2.4 使用公共节点并行并行计算方法（PBS）
8.2.5 使用BME节点的并行计算方法（SLURM）
8.3 FSL用户使用说明
8.3.1 概述
8.3.2 安装设置
8.3.3 FSL使用
8.3.4 教程
8.3.5 参考
9. 计算场景举例
10. 可视化应用

1. BME 高性能集群使用规范

BME高性能计算集群基于slurm调度系统，在使用本集群前，请仔细阅读集群使用规范以及集群使用手册，如因操作不当产生的违反使用规定的产生的问题，由用户本人承担，本文档长期更新。

1.1 系统架构

BME高性能计算集群由上科大图信中心托管，并与图信中心共享/hpc和/public存储。目前本集群主要包括以下节点(截止2022.11):

- 管理节点 （1个）
  - 用于管理员登录，不对普通用户开放
- 登录节点 （2个）
  - bme_login1，bme_login2
  - 用于用户登录、环境配置和提交计算任务
  - **禁止在登录节点直接运行作业**

IP地址	主机名	型号	配置
10.15.49.6	bme_login1		
10.15.49.7	bme_login2	R620 G30	4214R CPU @ 2.40GHz/16GB*8/480G SSD*2

- VNC节点（1个）
  - bme_gpu02
  - 用于执行需可视化的程序，代码调试
  - 禁止长时间运行计算资源

IP地址	主机名	型号	配置
10.15.49.9	bme_gpu02	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4

- GPU节点（17个）
  - bme_gpu01-17
  - GPU计算节点
  - 请使用作业调度系统使用GPU资源

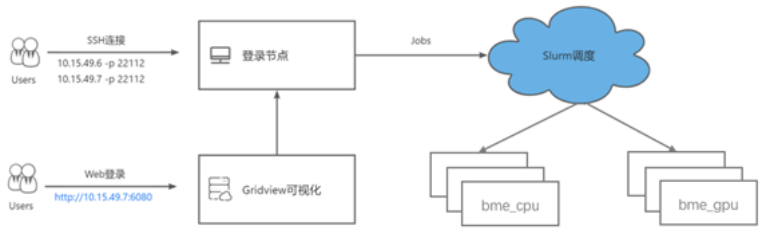
IP地址	主机名	型号	配置
10.15.49.8	bme_gpu01	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4
10.15.49.10	bme_gpu03	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.11	bme_gpu04	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.12	bme_gpu05	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.13	bme_gpu06	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.14	bme_gpu07	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.15	bme_gpu08	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*4
10.15.49.16	bme_gpu09	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*8
10.15.49.17	bme_gpu10	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100S*8
	bme_gpu11-17	NF5468M6	Gold 5318Y 2.10GHz 20C*2/32G*16/960G SSD*2/A100S*8

- CPU节点（19个）
  - bme_comput01-19
  - 用于执行不涉及GPU资源的程序
  - 请通过slurm调度系统使用计算资源

IP地址	主机名	型号	配置
10.15.48.18	bme_comput1	R620 G30	Gold 6248R 3.00GHz 24C*2/32G*16/480G SSD*2
10.15.48.19	bme_comput2	R620 G30	Gold 6248R 3.00GHz 24C*2/32G*16/480G SSD*2

- 存储节点
  - OStor-K30-436(NAS)_suma
  - 存储容量: 1152T
  - 挂载路径: /public_bme

集群拓扑结构示意图：



1.2 使用规范

1. 登录节点用于任务提交，环境配置等；禁止在登录节点运行计算程序。用户因在登录节点运行计算程序导致的后果由个人承担。
2. 集群通过slurm调度系统实现作业管理、任务分配、调度等功能，所有计算任务应由slurm提交至计算节点。
3. 集群ssh只用于监控任务运行情况，禁止ssh至登录节点运行程序。
4. 如使用salloc命令占用节点资源，请确保资源分配后会及时运行作业，避免长时间空卡。长时间占用资源且不使用的任务会被直接取消，并将此行为反馈至学院且及后续处理。

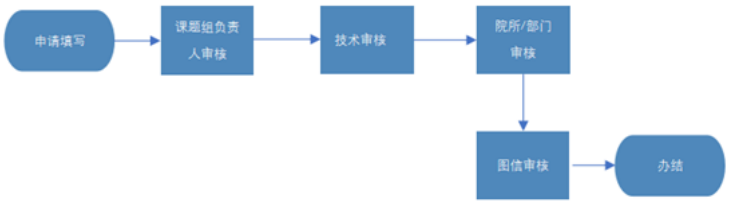
1.3 用户资源配置（重要）

单个用户在gpu队列中最多同时占用两张GPU；任务数量限制为2；单个任务内存限制为128G, CPU数量为8， 时间限制为120h。

2. 账户申请

2.1 高性能账号申请

上海科技大学高性能计算共享服务平台实行申请制，如有科研计算及教学任务需要使用，需由在校教职工发起线上申请，并经所在院所/部处及图信审批通过后交付用户使用。



使用申请链接：高性能服务申请

说明：

1. 资源负责人：

科研用途的账号的负责人为课题组负责人；课程用途的账号负责人为课程负责人。

2. 续期、延期和注销：

1）原则上，平台根据用户毕业/离校/离职手续的办理，同步进行访问账号的关停操作，用户将不能再使用账号访问平台；6个月后清理该用户账户下的所有数据。

2）账号所属用户在其离职/离校前，应做好在计算平台上的数据和作业等情况的梳理、备份和交接。

3）资源负责人应考虑其申请资源下的用户关停等产生的影响，如安装的应用、课题或成果产出的支撑数据等，对需要迁移或备份的数据，应提前做好处理，若因处理过程较长或由其他情况需要延长数据保留时间，应尽早与平台进行沟通。
4）若因课题需要，需要申请访问账号延期的，应提前与资源负责人进行沟通，经资源负责人发起，所在院所同意后，平台予以账号延期处理。
5）对于资源负责人离职或资源“申请使用时间”到期，平台认定资源可用性超期，对于资源可用性超期的资源，原则上，超期1个月后，平台会关停相关资源的队列权限，用户将无法再进行作业投递操作，超过3个月后，将对随资源申请一并开通的账号关停，6个月后进行数据清理。
6）资源负责人（或其授权的学校在职人员）应在资源超期前，尽早与平台沟通，并根据后续课题需要尽早发起（续期）申请或资源变更申请。
7）资源负责人在离职前，应充分考虑课题开展所需资源的延续性等问题，并提前与所在院所沟通，是否需要提请相关资源的延期；对于提出延期申请的资源负责人（或其授权的我校在职人员）应提前与平台联系，就延期资源量、延期资源的校内联络人、可调用延期资源的访问账号、资源可用时间等进行充分沟通，经院所审议出具说明后，根据相关流程办理资源延期手续。

如有疑问，请联系：
IT运维电话：021-20685566
IT运维邮箱：it-support@shanghaitech.edu.cn
孙思思：sunss@shanghaitech.edu.cn

3. 账户登录

3.1 登录节点介绍

现阶段， 可供用户使用的高性能集群有两个：学校统一管理的集群和BME学院的集群。 两个集群总有11个登录节点可供使用，其中BME集群有两个登录节点可供使用，具体情况如下所示：

- BME集群下的登录节点
  - 10.15.19.6
  - 10.15.49.7
- 学校集群下的登录节点（公共节点）
  - 10.15.22.109
  - 10.15.22.110
  - 10.15.22.111
  - 10.15.22.191

3.2 集群账户登录办法

对于不同的操作系统， 访问集群的方式和软件略有不同。例如Windows下， 可以使用MobaXterm软件来进行访问。

关于如何使用MobaXterm软件登录计算节点来访问集群，详见
https://it.shanghaitech.edu.cn/2022/0722/c8851a711935/page.htm，

注意端口号需要改为22112， 而非默认端口号22。
注意: BME节点下的/public_bme对公共节点是不可见的，如果使用公共节点登录，该目录下的文件是不可见的。

4. 文件传输

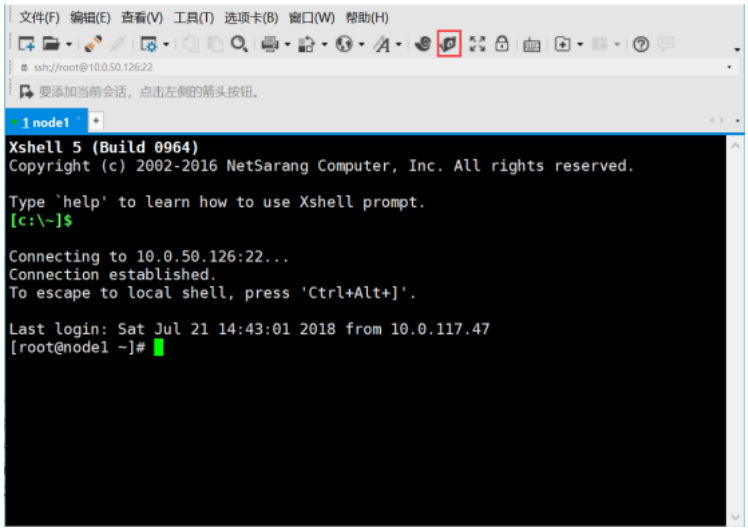
4.1 文件上传下载工具

Windows用户可以用SSH Secure Shell Client、winscp、Xftp等软件实现文件的上传下载。

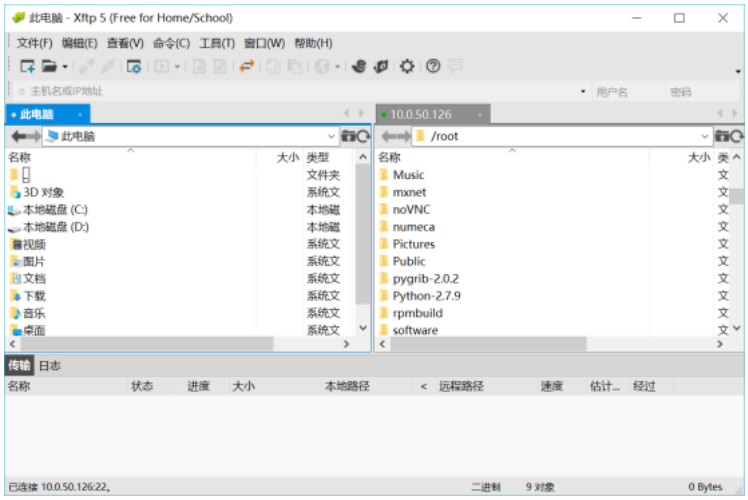
4.2 Windows操作系统实现方式

4.2.1 Xftp

Xftp为xmanager商业组件，Xftp在Xshell界面进行了集成，使用xshell登陆到集群后，点击工具栏上文件夹的图标，可以打开Xftp文件传输的窗口。

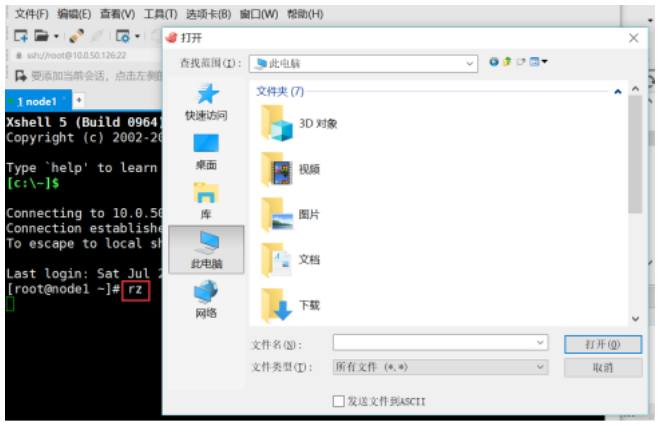


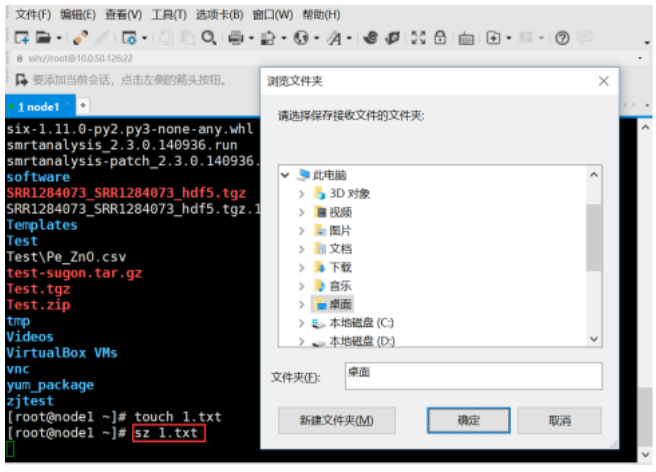
左侧为本机，右侧为高性能集群，可直接拖动进行文件上传和下载。



4.2.2 Windows ssh登陆服务器后使用rz和sz命令

除上述工具外，在命令行中使用rz和sz命令也可以方便的进行小文件的上传和下载（xshell可以，putty不行）。

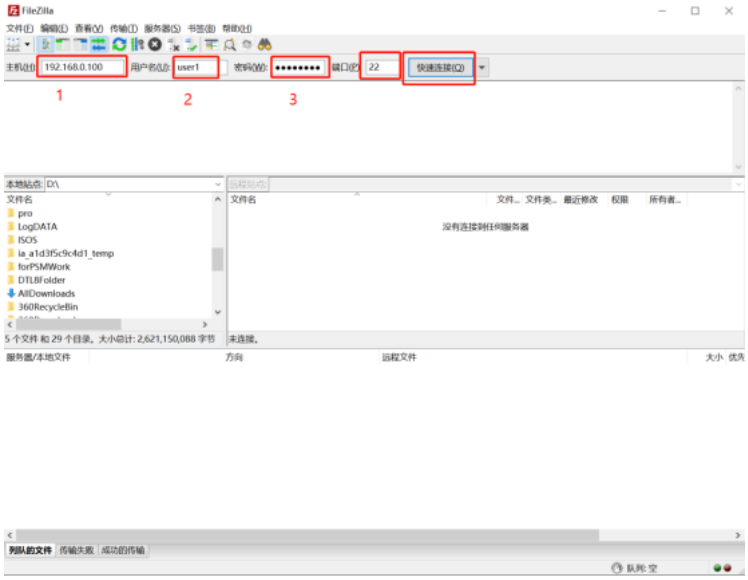




4.2.3 Filezilla

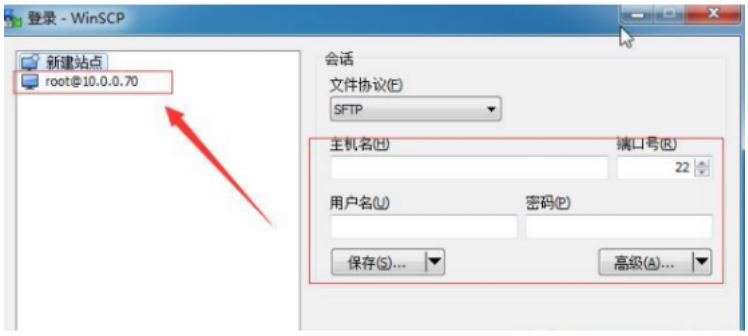
Filezilla不仅有windows版本也有MacOS和Linux版本，推荐官网下载地址为：https://filezilla-project.org/download.php?type=client

依次输入服务器IP地址、用户名、密码、端口号，即可正常连接。连接后左面为本地目录，右侧为服务器目录，两面文件可以相互拖拽即可形成文件上传或下载（filezilla在带宽充足的情况下多任务表现优异，推荐使用）。

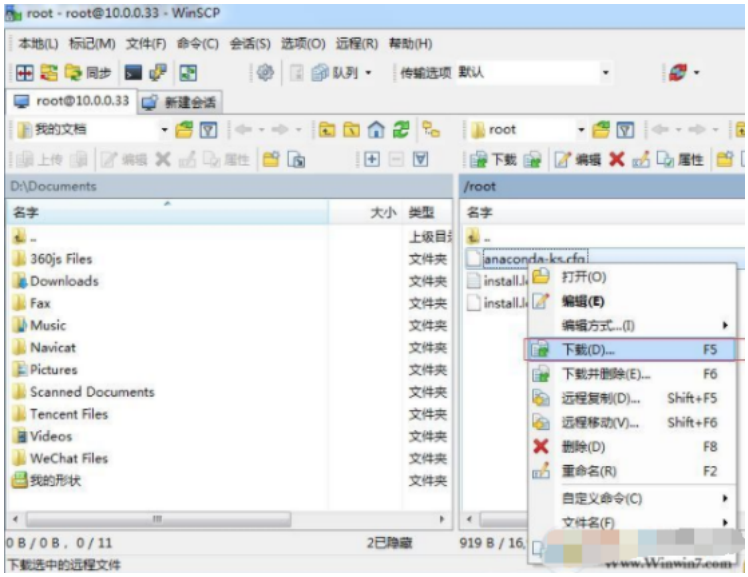


4.2.4 winscp

打开winscp程序以后，输入主机地址、用户名、密码、端口号即可连接服务器。



同样左侧为本地目录，右侧为服务器目录两面文件可以相互拖拽即可形成文件上传或下载。



4.3 Linux和MacOS操作系统直接使用命令

4.3.1 scp

Linux和MacOS主要使用系统命令“scp”，即可实现高效传输。

scp命令有两个可选的选项-p和-r分别是“preserve permission”和“recursive copy”的意思。

举例如下：
1.从本地复制到远程：scp local_file remote_username@remote_ip: remote_folder
2.从远程复制到本地：scp username@remote_ip:remote_file local_file

3.从“当前本地文件”到“远程类unix主机用户名@ip地址:/具体目录”：scp filename test@ip:/home/test

Linux和MacOS主要使用系统命令“scp”，即可实现高效传输。

通过scp命令，scp [-pr]，两个可选的选项-p和-r分别是“preserve permission”和“recursive copy”（对于目录 copy 来说）的意思，更详细的参考 man scp。

从本地复制到远程：scp local_file remote_username@remote_ip: remote_folder

从远程复制到本地: scp username@remote_ip:remote_file local_file

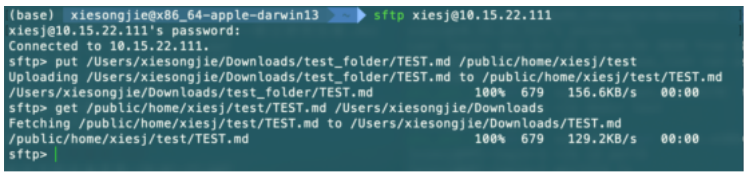
Scp “当前本地文件” “远程类unix主机用户名@ip地址:/具体目录”。

命令参考如下：

scp filename test@ip:/home/test

4.3.2 sftp

在终端中输入：sftp username@remote ip(or remote host name),
例如： sftp xiesj@10.15.22.111
使用sftp的常用命令get 和put 来上传本地文件和下载远程文件



4.3.3 XQuartz

Mac 不再随附 X11，但 XQuartz 项目会提供 X11 服务器和客户端库。
XQuartz 项目提供适用于 MacOS 的 X11 服务器和客户端库，网址是 www.xquartz.org。下载可用的最新版本。下载解压安装成功后，在应用的Utilities中点击运行XQuartz，Dock栏出现如下XQuartz的小图标。
终端中输入：ssh -X username@remote_ip，之后即可启用GUI程序。

5. 常用Linux命令及操作

5.1 ls

ls　#默认列出当前目录下的所有文件。
ls -l(long) #以长格式查看文件。
ls -d(directorys) #查看目录。
ls -F #给不同文件的结尾加标识
ls -p #给目录结尾加斜线
ls -a #显示所有文件，包括隐藏文件，默认.开头的文件都是隐藏不显示的
ls -r #倒排序
ls -t #按修改时间排序，一般rt结合，查看最近被修改的文件。
ls -i /data/ #显示inode，文件索引。
ls –l --time-style=long-iso /data #规范时间，（2016-03-04这种格式）。

5.2 mkdir

创建目录。

mkdir /data #在根目录下创建data目录。
mkdir –p /aa/bb #创建目录bb，如果没有aa目录则自动创建。

5.3 cd

切换目录。

cd /etc #从当前目录切换到/etc路径下。

5.4 pwd

打印工作目录。

echo $PWD #可以看到这个变量的值

5.5 touch

不存在就创建文件，存在则更新文件时间戳信息。

touch /data.txt #直接在/目录下创建data.txt文件。
cd /; touch data.txt #切换到/目录下，创建data.txt文件。

5.6 vi/vim

vi编辑器。

#a或i进入插入状态，点击Esc退出编辑状态进入命令状态。
#命令状态按:wq保存退出。（wq为write quit）
#命令模式下：
dd #直接删除一行
set nu #显示行号
set nonu #不显示行号
G :$]] #光标移动到文件的最后一行
:0 gg [[#光标移动到文件的第一行
0 ^ home #从光标所在位置将光标移动到当前行的开头
$ end #从光标所在位置将光标移动到当前行的结尾
u #取消上一次的动作
/ #向下搜索，继续搜索n，反向搜索N
? #向上搜索，继续搜索n，反向搜索N

5.7 echo

打印

echo ‘I like linux’ #打印后边的字符串。
echo –n “oldboy”;echo “oldboy” #不换行输出
echo -e “oldboy\toldboy” #加特殊符号，比如制表符\t，换行\n等。

5.8 cat

查看文件内容

cat /data.txt #查看data.txt文件中的内容
cat >>/data.txt<<EOF
I like linux
you like linux
EOF
#以上用法结合了cat和>>和<<，可以追加多行内容，内容用EOF包裹，
#EOF可以用任意重复字符替换，只要内容不存在就可以。文件不存在会自动创建文件。
cat -n /data.txt #显示行号
cat test{,1} >/tmp/aaa.txt #将test.txt和test1.txt文件内容合并到aaa.txt里面，行合并，上下合并。
cat -T test.txt #区分tab键和空格，tab键会被^I替代。
cat -E test.txt #会在行尾加$符号，空行也会有。

5.9 tac

和cat相反，倒序读取文件。最后一行先输出，然后倒数第二行…

5.10 paste

粘帖，可以用作合并。

paste test1 test2 #列合并，即左右合并。
paste -d ： test1 test2 #以冒号分隔，列合并。
paste -s test1 test2 #将每个文件列转行之后再按行合并。

5.11 sort

排序命令

sort aa.txt #将内容按ascII码进行排序。
sort -n aa.txt #讲内容按数值排序。
sort -r aa.txt #倒序
sort -u aa.txt #压缩重复行
sort -k2 aa.txt #-k默认以空格分隔，指定第2列进行排序
sort -t “:” -k 2 aa.txt #用冒号分隔后以第2列进行排序。

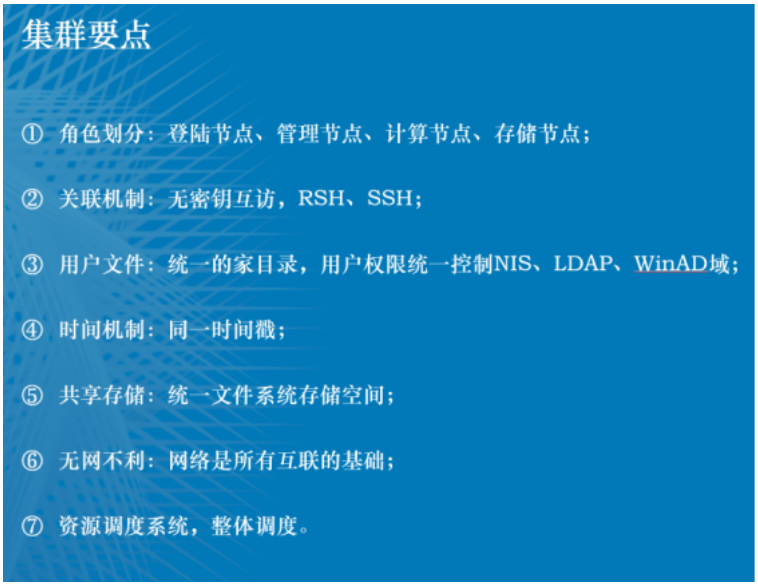
6. 集群硬件架构

6.1 BME 计算节点配置信息

序号	功能	IP地址	主机名	型号	配置	序列号
1	管理节点	10.15.34.162	amdadmin	A320-G30	AMD EPYC 7502 32-Core/16G*16/480G SSD*2	9800135402500172
2	登陆节点1	10.15.49.6	bme_login1			
3	登陆节点2	10.15.49.7	bme_login2	R620 G30	4214R CPU @ 2.40GHz/16GB*8/480G SSD*2	9800140901727026
4	GPU计算节点	10.15.49.8	bme_gpu01	X640 G30	Gold 5218R 2.10GHz	9800143301727025

					20C*2/32G*12/960G SSD*2/V100S*4	
5	VNC节点	10.15.49.9	bme_gpu02	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800143301727024
6	GPU计算节点	10.15.49.10	bme_gpu03	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520743
7	GPU计算节点	10.15.49.11	bme_gpu04	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520739
8	GPU计算节点	10.15.49.12	bme_gpu05	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520742
9	GPU计算节点	10.15.49.13	bme_gpu06	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520741
10	GPU计算节点	10.15.49.14	bme_gpu07	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520740
11	GPU计算节点	10.15.49.15	bme_gpu08	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/V100S*4	9800164102520744
12	GPU计算节点	10.15.49.16	bme_gpu09	X640 G35	Gold 5218R 2.10GHz 20C*2/64G*12/960G SSD*2/V100*8	9800164202535478
13	GPU计算节点	10.15.49.17	bme_gpu10	X640 G30	Gold 5218R 2.10GHz 20C*2/32G*12/960G SSD*2/A100*8	9800163302512950
14	CPU计算节点	10.15.49.18	bme_comput1	R620 G30	Gold 6248R 3.00GHz 24C*2/32G*16/480G SSD*2	9800147903023988
15	CPU计算节点	10.15.49.19	bme_comput2	R620 G30	Gold 6248R 3.00GHz 24C*2/32G*16/480G SSD*2	9800147903023989

6.2 集群要素介绍



6.3 用户登录拓扑

- **登陆节点**

一般整个集群几十到几百甚至几千台服务器，登陆节点是整套集群的入口。登陆节点只是为了让用户通过网络能够连通集群，真正的计算资源是计算节点，所以要通过调度系统把作业任务分发到计算节点上。

- **管理节点**

管理节点一般是作业调度系统的主节点，负责所有计算节点的资源监控、作业调度和监控、硬件监控、有的还是用户账户认证等一系列重要的主服务，如果管理节点异常，计算节点将一盘散沙无法有效作业。

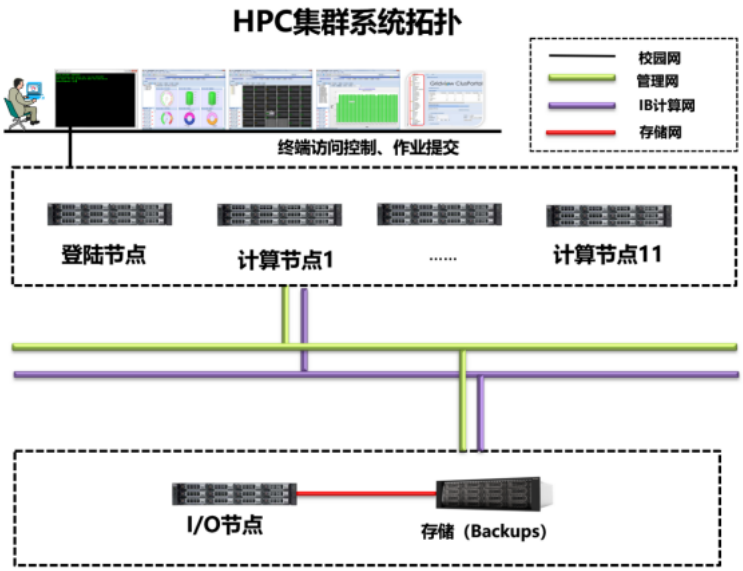
- **计算节点**

计算节点为实际提供计算资源的主机，客户作业通过调度系统分发到计算节点，计算节点利用自身的处理器、内存、GPU等进行数据处理后生成最终计算结果进行保存。

- **存储节点**

存储节点顾名思义是为数据存储而存在的。在高性能集群中存储为所有节点共同使用，对其稳定性要求较高，同时时存储应当有较大的容量和很好的性能。

- **集群常规拓扑结构**



7. 作业提交及变更

7.1 job array在Torque作业调度系统的实现

7.1.1 确定pbs对用户计算任务的适用性

用户有时需要运行多个并行且独立的计算任务。例如，用户可能需要用同一种方式分析数据集中多个主题的数据。另一个例子是，用户实现的算法有独立的迭代/复制机制，或者需要使用多个不同的参数来执行算法。这类计算任务在计算机科学领域通常被称为“尴尬的并行”问题，因为任务显然可以分解为许多相同但独立的子任务。PBS job array是这种类型计算任务的合适选择。

7.1.2 什么是PBS job array?

Job array是由单个PBS job脚本启动的一组PBS job。Job array中每个独立的作业(称为“子作业”)都有一个唯一的标识符，该标识符被分配给Linux环境变量“PBS_ARRAYID”中相应的子作业。用户可以指定PBS_ARRAYID的范围为连续正整数，如1 ~ 5。这些整数的范围不需要从1开始：可以根据用户的选择从任何正整数开始。用户可以在单任务PBS脚本中添加一个PBS指令“#PBS -t”指定这个范围，或者通过qsub命令的“-t”选项提交作业。可以使用[$PBS_ARRAYID]获取环境变量PBS_ARRAYID。Job array的解释性PBS脚本和相应程序将在本文档的下一节给出。

7.1.3 如何在高性能计算系统下实现PBS job array？

以下是PBS job array 脚本示例(文件名: exec_freesurfer_job_array.pbs).

```
#!/bin/bash
#PBS -N fs6   #  指定作业名，默认为脚本名
#PBS -j oe      # 标准输出与标准错误输出合流
#PBS -q bme_pub_cpu # 绑定队列
#PBS -l nodes=1:ppn=8 # 申请1个节点，每个节点8个core
#PBS -l mem=24gb    # 申请内存大小
#PBS -l walltime=72:00:00 # 每个作业运行时能使用的最大wallclock time
#PBS -t 1-5   # PBS_ARRAYID标识符范围

#
export FREESURFER_HOME=/your/freesurfer/home/directory
export SUBJECTS_DIR=/file/save/directory  # 运行完成后文件的存储路径
source $FREESURFER_HOME/SetUpFreeSurfer.sh
cd $PBS_O_WORKDIR # 切换至目标文件目录下
#
files=(*.nii.gz)  # 文件后缀为.nii.gz
recon-all -i ${files[$PBS_ARRAYID]} -subject ${files[$PBS_ARRAYID]//.nii.gz} -all -openmp 8
```

在上面的例子中，这个作业数组由5个子作业组成，通过PBS指令 ”PBS -t 1-5” (用红色表示)实现PBS_ARRAYID从1到5，每个子任务都执行了FreeSurfer的recon-all命令。用户可以使用以下命令提交上述PBS job array脚本：

```
qsub exec_freesurfer_job_array.pbs
```

注意：在这种情况下，qsub命令的“-t”选项是不需要的，因为我们已经在PBS脚本中使用 “#PBS -t”指定了PBS_ARRAYID的范围。关于上述脚本中使用的所有其他PBS指令的定义，请参阅*用户指南*

7.1.4 如果监测PBS job array状态？

用户可以通过qstat命令的“-t”选项查看PBS job array的状态：

```
qstat -tu username
```

```
(base) [mazhw_f@hpc-login x]$ ls
exec_freesurfer_job_array.pbs
(base) [mazhw_f@hpc-login x]$ vim exec_freesurfer_job_array.pbs
(base) [mazhw_f@hpc-login x]$ qsub exec_freesurfer_job_array.pbs
4872764[].node1
(base) [mazhw_f@hpc-login x]$ qstat -tu mazhw_f

node1:
                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
4872764[1].node1        mazhw_f     pub_blad CQ_fs6-1            --      1      8   24gb  72:00:00 Q       --
4872764[2].node1        mazhw_f     pub_blad CQ_fs6-2            --      1      8   24gb  72:00:00 Q       --
4872764[3].node1        mazhw_f     pub_blad CQ_fs6-3            --      1      8   24gb  72:00:00 Q       --
4872764[4].node1        mazhw_f     pub_blad CQ_fs6-4            --      1      8   24gb  72:00:00 Q       --
4872764[5].node1        mazhw_f     pub_blad CQ_fs6-5            --      1      8   24gb  72:00:00 Q       --
4872764[6].node1        mazhw_f     pub_blad CQ_fs6-6            --      1      8   24gb  72:00:00 Q       --
4872764[7].node1        mazhw_f     pub_blad CQ_fs6-7            --      1      8   24gb  72:00:00 Q       --
4872764[8].node1        mazhw_f     pub_blad CQ_fs6-8            --      1      8   24gb  72:00:00 Q       --
4872764[9].node1        mazhw_f     pub_blad CQ_fs6-9            --      1      8   24gb  72:00:00 Q       --
4872764[10].node1       mazhw_f     pub_blad CQ_fs6-10           --      1      8   24gb  72:00:00 Q       --
4872764[11].node1       mazhw_f     pub_blad CQ_fs6-11           --      1      8   24gb  72:00:00 Q       --
4872764[12].node1       mazhw_f     pub_blad CQ_fs6-12           --      1      8   24gb  72:00:00 Q       --
4872764[13].node1       mazhw_f     pub_blad CQ_fs6-13           --      1      8   24gb  72:00:00 Q       --
4872764[14].node1       mazhw_f     pub_blad CQ_fs6-14           --      1      8   24gb  72:00:00 Q       --
4872764[15].node1       mazhw_f     pub_blad CQ_fs6-15           --      1      8   24gb  72:00:00 Q       --
4872764[16].node1       mazhw_f     pub_blad CQ_fs6-16           --      1      8   24gb  72:00:00 Q       --
4872764[17].node1       mazhw_f     pub_blad CQ_fs6-17           --      1      8   24gb  72:00:00 Q       --
4872764[18].node1       mazhw_f     pub_blad CQ_fs6-18           --      1      8   24gb  72:00:00 Q       --
4872764[19].node1       mazhw_f     pub_blad CQ_fs6-19           --      1      8   24gb  72:00:00 Q       --
4872764[20].node1       mazhw_f     pub_blad CQ_fs6-20           --      1      8   24gb  72:00:00 Q       --
4872764[21].node1       mazhw_f     pub_blad CQ_fs6-21           --      1      8   24gb  72:00:00 Q       --
4872764[22].node1       mazhw_f     pub_blad CQ_fs6-22           --      1      8   24gb  72:00:00 Q       --
4872764[23].node1       mazhw_f     pub_blad CQ_fs6-23           --      1      8   24gb  72:00:00 Q       --
4872764[24].node1       mazhw_f     pub_blad CQ_fs6-24           --      1      8   24gb  72:00:00 Q       --
4872764[25].node1       mazhw_f     pub_blad CQ_fs6-25           --      1      8   24gb  72:00:00 Q       --
4872764[26].node1       mazhw_f     pub_blad CQ_fs6-26           --      1      8   24gb  72:00:00 Q       --
4872764[27].node1       mazhw_f     pub_blad CQ_fs6-27           --      1      8   24gb  72:00:00 Q       --
4872764[28].node1       mazhw_f     pub_blad CQ_fs6-28           --      1      8   24gb  72:00:00 Q       --
4872764[29].node1       mazhw_f     pub_blad CQ_fs6-29           --      1      8   24gb  72:00:00 Q       --
```

能够看到这里列了29个子任务，他们由job array ID (方括号前的数字，如上图中4872764) 以及子任务 ID (方括号中的数字，如上图中1-5)组成： *Job_ID*[*i*]。作业状态(“Q”, “R”, “E”, “H”)的详细解释，请参照*用户指南*

7.1.5 如何删除PBS job array?

使用以下命令删除已提交的job array的子任务i：

```
qdel Job_ID[i]
```

使用以下命令删除已提交job array的所有子任务：

```
qdel -p Job_ID\[\]
```

7.1.6 如何查看job array的屏幕输出？

默认情况下，PBS会为job array的每个子任务输出保存在默认文件“*JobName*.o*Job_ID-i*”中，将错误保存在默认文件“*JobName*.e*Job_ID-I*”中。如果在一个PBS job array脚本中写明了 “*#PBS -j oe*” (像例子中一样)，则无论是否报错，输出都会保存在 “*JobName*.o*Job_ID-i*”文件中。

7.2 job array在Slurm作业调度系统的实现

7.2.1 Slurm任务脚本示例：

以下是Slurm job array 脚本示例(文件名：exec_freesurfer.job).

```
#!/bin/bash
#SBATCH -J freesurfer_job_array # 作业名
#SBATCH -o job%j.o    # 标准输出
#SBATCH -e job%j.e    # 标准错误
#SBATCH -n 1      # 指定core数量
#SBATCH -p bme_cpu    # 指定分区
#SBATCH -N 1      # 指定node数量
#SBATCH --mem=24gb    # 指定内存大小
#SBATCH --time=72:00:00   # 每个作业运行时能使用的最大wallclock time
#SBATCH --array=1-5   # SLURM_ARRAY_TASK_ID标识符范围1-i,i可取1到5的任意整数

#
export FREESURFER_HOME=/your/freesurfer/home/directory
export SUBJECTS_DIR=/file/save/directory  # 运行完成后文件的存储路径
source $FREESURFER_HOME/SetUpFreeSurfer.sh
cd $PBS_O_WORKDIR # 切换至目标文件目录下
#

Files=(*.nii.gz)  # 文件后缀为.nii.gz
recon-all -i ${files[$SLURM_ARRAY_TASK_ID]} -subject ${files[$SLURM_ARRAY_TASK_ID]//.nii.gz} -all -openmp 8
# recon-all运算。//.nii.gz意为保存文件时，文件后缀.nii.gz不显示。
# -openmp 8意为8个线程，共享内存并行计算。运用环境变量SLURM_ARRAY_TASK_ID。
```

在上面的例子中，这个作业数组由5个子作业组成，通过SLURM指令 ”SBATCH --array 1-5” (用红色表示)实现SLURM_ARRAY_TASK_ID从1到5，每个子任务都执行了FreeSurfer的recon-all命令。用户可以使用以下命令提交上述脚本：

```
sbatch exec_freesurfer.job
```

7.2.2 如何提交Slurm任务？

用户可以通过salloc命令申请可交互节点，也可以通过sbatch提交作业，用于运行完整的训练任务：

```
sbatch Job_name
```

7.2.3 如何监测Slurm任务状态？

用户可以通过squeue命令的”-u”选项查看指定用户任务的状态：

```
squeue -u username
```

用户可以通过squeue命令的”-j”选项查看指定任务的状态：

```
squeue -j Job_ID
```

命令输出示例如下：

```
(base) [mazhw_f@bme_login1 CQ_exec_sbatch]$ sbatch exec_freesurfer.job
Submitted batch job 17718
(base) [mazhw_f@bme_login1 CQ_exec_sbatch]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          17718_1   bme_cpu freesurf  mazhw_f  R    INVALID      1 bme_comput1
          17718_2   bme_cpu freesurf  mazhw_f  R    INVALID      1 bme_comput1
          17718_3   bme_cpu freesurf  mazhw_f  R    INVALID      1 bme_comput1
          17718_4   bme_cpu freesurf  mazhw_f  R    INVALID      1 bme_comput1
          17718_5   bme_cpu freesurf  mazhw_f  R    INVALID      1 bme_comput1
```

7.2.4 如何取消Slurm任务

取消任务的方法是输入scancel加上任务的ID：

```
scancel Job_ID
```

7.2.5 如何查看Slurm调度系统输出？

默认情况下，SLURM会为每个子任务的标准输出保存在“job*Job_ID*+*i*-1.o”(*Job_ID*为作业号，*i*为子任务号)，标准错误保存在“job*Job_ID*+*i*-1.e”(*Job_ID*为作业号，*i*为子任务号)。

7.2.6 References

1. https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki
2. https://surfer.nmr.mgh.harvard.edu/fswiki/DownloadAndInstall
3. https://surfer.nmr.mgh.harvard.edu/fswiki/Tutorials

7.3 PBS调度系统上的python程序Job Array使用指南

7.3.1 什么是job array?

对于PBS调度系统来说，一个 Job Array 是由一个PBS作业脚本发起的一组PBS作业。Job Array 的每一个单独的作业，称为 "子作业"，都有一个唯一的标识符，这个标识符被分配给Linux环境变量"PBS_ARRAYID"中的相应子作业。用户可以将PBS_ARRAYID的范围指定为连续的正整数，例如，从1到5。这些整数的范围不需要从1开始：它可以根据用户的选择从任何正整数开始。为了指定这个范围，用户可以在之前的单次作业PBS脚本中加入PBS指令 "#PBS -t"，或者通过qsub命令的-t选项提交作业。环境变量PBS_ARRAYID可以在你的python程序中通过函数 "os.getenv "以字符串的形式获得，如果你愿意，还可以通过函数 "int()"将其转换为数字变量。

7.3.2 什么时候需要使用job array?

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。

7.3.3 如何在PBS调度系统上实现Job Array?

下面是一个PBS Job Array的示例脚本。

脚本存放在集群路径：

/hpc/data/home/bme/mazhw/group/temp/cluster/python/PBS/exec_python_job_array.pbs

```
#PBS -N python_job_array
#PBS -j oe
#PBS -l nodes=1:ppn=1
#PBS -l mem=4gb
#PBS -l walltime=01:00:00
#PBS -M qiyy1@shanghaitech.edu.cn
#PBS -q bme_pub_cpu
#PBS -m abe
#PBS -t 1-5

cd /hpc/data/home/bme/mazhw/group/temp/cluster/python/PBS/
module load apps/python/3.7.1
python MyProgram.py ${PBS_ARRAYID}
```

在上面的例子中，这个 Job Array 由5个子工作组成，每个工作的PBS_ARRAYID从1到5。这是通过PBS指令#PBS -t 1-5实现的。用户可以通过以下命令提交上述PBS作业阵列脚本。

```
qsub exec_python_job_array.pbs
```

请注意，在这种情况下不需要qsub命令的"-t "选项，因为我们已经通过在PBS脚本中使用PBS指令 "#PBS -t "来指定PBS_ARRAYID的范围。关于上述脚本中使用的所有其他PBS指令的定义，请参考用户指南

需要着重说明的是#PBS -M qiyy1@shanghaitech.edu.cn这段脚本需要改为自己的邮箱。同时#PBS -q bme_pub_cpu作为BME学院的用户使用PBS调度系统时必不可少。在这个例子中，这个 Job Array 的PBS脚本将执行5个子作业，每个子作业将执行一个名为 "MyProgram.py"的 python程序。下面是利用环境变量PBS_ARRAYID的python程序 "MyProgram.py"的例子。

```
import os

# obtain the SLURM array ID
array_id = os.getenv('PBS_ARRAYID')
array_id_num = int(array_id)

# your program
result = array_id_num * array_id_num
print(result)
```

在这个小例子中，python程序将获得环境变量PBS_ARRAYID，并将其平方后作为结果输出，你可以用你自己的 python程序来替换这个python脚本的第二部分（名为 "your program"），以执行你的特定计算任务。

7.3.4 如何监测PBS job array的运行状态？

用户可以使用qstat命令的"-t "选项来检查PBS Job Array的状态: qstat -tu username

下图是该命令的示例输出的截图

```
(base) [qiyy1@hpc-login PBS]$ qstat -tu qiyy1

node1:
                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
4872837[1].node1        qiyy1       pub_blad python_job_array    --      1      1    4gb  01:00:00 Q       --
4872837[2].node1        qiyy1       pub_blad python_job_array    --      1      1    4gb  01:00:00 Q       --
4872837[3].node1        qiyy1       pub_blad python_job_array    --      1      1    4gb  01:00:00 Q       --
4872837[4].node1        qiyy1       pub_blad python_job_array    --      1      1    4gb  01:00:00 Q       --
4872837[5].node1        qiyy1       pub_blad python_job_array    --      1      1    4gb  01:00:00 Q       --
```

你可以看到有5个子工作，它们由 Job ID（方括号前的数字，即本例中的4872837）和子工作ID（方括号之间的数字，即本例中的1-5）组合命名。 Job_ID[i]。关于工作状态（即倒数第二列的"Q"、"R"、"E"、"H"等）的详细解释，请参考用户指南。

7.3.5 如何删除PBS job array?

用户可以通过qdel Job_ID[i]删除在 Job Array 中的某一个子工作i（由Job_ID标识）。用户也可以通过qdel -p Job_ID[\]删除 Job Array 中的所有子工作（由Job_ID标识）。

7.3.6 如何检查job array的输出

默认情况下，PBS将把 Job Array 中每个子作业的输出写到以下文件"JobName.oJob_ID-i"。PBS将把每个子工作的错误输出写到以下文件中"JobName.eJob_ID-i"。如果在PBS脚本中使用了PBS指令 "#PBS -j oe"（如上面的例子），非错误输出和错误输出将同时写入文件 "JobName.oJob_ID-i"。

7.4 SLURM调度系统上的R程序Job Array使用指南

7.4.1 什么是job array?

对于SLURM调度系统来说，一个 Job Array 是由一个SLURM作业脚本发起的一组SLURM作业。Job Array 的每一个单独的作业，称为 "子作业"，都有一个唯一的标识符，这个标识符被分配给Linux环境变量"SLURM_ARRAY_TASK_ID"中的相应子作业。用户可以将SLURM_ARRAY_TASK_ID的范围指定为连续的正整数，例如，从1到5。这些整数的范围不需要从1开始：它可以根据用户的选择从任何正整数开始。为了指定这个范围，用户可以在之前的单次作业SLURM脚本中加入SBATCH指令"#SBATCH --array"。环境变量SLURM_ARRAY_TASK_ID可以在你的R程序中通过函数 "Sys.getenv "以字符串的形式获得，如果你愿意，还可以通过函数 "as.numeric "将其转换为数字变量。

7.4.2 什么时候需要使用Job Array?

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。

7.4.3 如何在Slurm调度系统上实现job array?

下面是一个SLURM Job Array的示例脚本。

脚本存放在集群路径为

/hpc/data/home/bme/mazhw/group/temp/cluster/R/SLURM/exec_R_job_array.job

```
#!/bin/bash
#SBATCH -J R_job_array
#SBATCH -o job%j.o
#SBATCH -e job%j.e
#SBATCH -n 1
#SBATCH -N 1
#SBATCH --mem=4G
#SBATCH --time=01:00:00
#SBATCH --array=1-5

cd /hpc/data/home/bme/mazhw/group/temp/cluster/R/SLURM/
module load apps/R/4.1.0
R CMD BATCH MyProgram.R MyProgram.${SLURM_ARRAY_TASK_ID}.log
```

在上面的例子中，这个 Job Array 由5个子工作组成，每个工作的SLURM_ARRAY_TASK_ID从1到5。这是通过SLURM指令 #SBATCH --array=1-5实现的。用户可以通过以下命令提交上述SLURM作业阵列脚本。

```
sbatch exec_R_job_array.job
```

在这个例子中，这个 Job Array 的SLURM脚本将执行5个子作业，每个子作业将执行一个名为

"MyProgram.R"的R程序。下面是利用环境变量SLURM_ARRAY_TASK_ID的R程序 "MyProgram.R"的例子。

```
# obtain the PBS array ID
ARRAYID<-Sys.getenv("SLURM_ARRAY_TASK_ID")
ARRAYID_num<-as.numeric(ARRAYID)

# your program
mat<-replicate(10,ARRAYID_num)
filename<-paste('R results_',ARRAYID,'.txt',sep='')
write.table(mat,file=filename,col.names=F,row.names=F)

# exit R
q(save = "no")
```

在这个小例子中，R程序将获得环境变量SLURM_ARRAY_TASK_ID，生成一个10X1的向量，每个元素都等于数组的ID，并将这个向量保存到一个文本文件。如果你操作完全正确，那么在这个 Job Array 完成后，你会得到5个文本文件 "R_results_1.txt"、"R_results_2.txt"、"R_results_3.txt"、"R_results_4.txt "和"R_results_5.txt"，而每个文本文件都存储了相应的10X1向量，其每个元素相当于那个特定的array ID。你可以用你自己的R程序来替换这个R脚本的第二部分（名为 "your program"），以执行你的特定计算任务。

7.4.4 如何监测Slurm job array的运行状态？

用户可以使用squeue命令的"-t "选项来检查SLURM Job Array的状态: squeue --array下图是该命令的示例输出的截图

```
(base) [qiyy1@bme_login2 SLURM]$ squeue --array
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              5321   bme_gpu  firefox    qiyy1 CG       0:00      1
          17736_1   bme_gpu R_job_ar    qiyy1 PD       0:00      1 (Priority)
          17736_2   bme_gpu R_job_ar    qiyy1 PD       0:00      1 (Priority)
          17736_3   bme_gpu R_job_ar    qiyy1 PD       0:00      1 (Priority)
          17736_4   bme_gpu R_job_ar    qiyy1 PD       0:00      1 (Priority)
          17736_5   bme_gpu R_job_ar    qiyy1 PD       0:00      1 (Priority)
```

除去第一个ID为5321正在进行的其他任务，你可以看到有5个子工作，它们由 Job ID（下划线前的数字，即本例中的17736）和子工作ID（下划线后数字，即本例中的1-5）组合命名为Job_ID_i。

7.4.5 如何删除Slurm job array?

用户可以通过scancel Job_ID_i删除在 Job Array 中的某一个子工作i（由Job_ID标识）。用户也可以通过scancel Job_ID删除 Job Array 中的所有子工作（由Job_ID标识）。

7.4.6 如何检查job array的输出

默认情况下，SLURM将把 Job Array 中每个子作业的输出写到以下文件"Job_ID.o"。SLURM将把每个子工作的错误输出写到以下文件中"Job_ID.e"。

8. 集群软件使用及常见问题

8.1 Freesurfer

8.1.1 软件介绍

FreeSurfer是一个软件包，用于分析和可视化结构和功能神经影像数据。FreeSurfer提供结构MRI数据的完整处理流程，包括：1) 颅骨剥离，B1偏压场校正，灰白质分割；2) 皮层表面模型的重建；3) 标记皮层表面的区域，以及皮层下的大脑结构；4) 立体定位图谱个体皮质表面的非线性配准；5) 组间形态计量学差异的统计学分析。

8.1.2 下载及安装

FreeSurfer 6 下载链接

Windows: https://surfer.nmr.mgh.harvard.edu/fswiki/rel6downloads

Linux: https://surfer.nmr.mgh.harvard.edu/pub/dist/freesurfer/6.0.0/freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0.tar.gz

本文档以FreeSurfer 6的Linux版本安装为例。

Step1: 从官网下载好相应的FreeSurfer 6 的工具包，上传到用户自己的集群空间。（https://surfer.nmr.mgh.harvard.edu/fswiki/rel6downloads）

Step2: 解压工具包

```
# 解压工具包
tar -C /usr/local -xzvf freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0.tar.gz
# 注意 /usr/local 可以自定义为FreeSurfer的安装目录
```

Step3: 配置环境变量

```
# 注意/usr/local为用户自定义的FreeSurfer安装目录
vim ~/.bashrc
# 在打开的.bashrc中添加下面的两行命令
export FREESURFER_HOME=/usr/local/freesurfer
source $FREESURFER_HOME/SetUpFreeSurfer.sh
```

Step4: 更新环境变量

```
source ~/.bashrc
```

出现以下内容即正确配置环境变量

```
FREESURFER_HOME /usr/local/freesurfer
FSFAST_HOME     /usr/local/freesurfer/fsfast
FSF_OUTPUT_FORMAT nii
SUBJECTS_DIR    /usr/local/freesurfer/subjects
MNI_DIR         /usr/local/freesurfer/mni
```

Step5: 获取Licences

必须获得许可证密钥Licences才能使 FreeSurfer 工具运行。用户需要在https://surfer.nmr.mgh.harvard.edu/registration.html 填写一些信息，最后获得一个Licence.txt文件，最后将该文件放到FREESURFER_HOME路径下即可。

8.1.3 软件使用

FreeSurfer要求一个名为**SUBJECTS_DIR**的环境变量，这个目录是用户存储的被处理数据的目录。使用以下命令暂时导入**SUBJECTS_DIR**的环境变量。

```
export SUBJECTS_DIR=<path to subject data>
```

8.1.3.1 转化.mgz文件到nifti文件

```
cp $FREESURFER_HOME/subjects/sample-001.mgz .
mri_convert sample-001.mgz sample-001.nii.gz
# 以下为结果显示
...
reading from sample-001.mgz...
TR=7.25, TE=3.22, TI=600.00, flip angle=7.00
i_ras = (-0, -1, -0)
j_ras = (-0, 0, -1)
k_ras = (-1, 0, 0)
writing to sample-001.nii.gz...
```

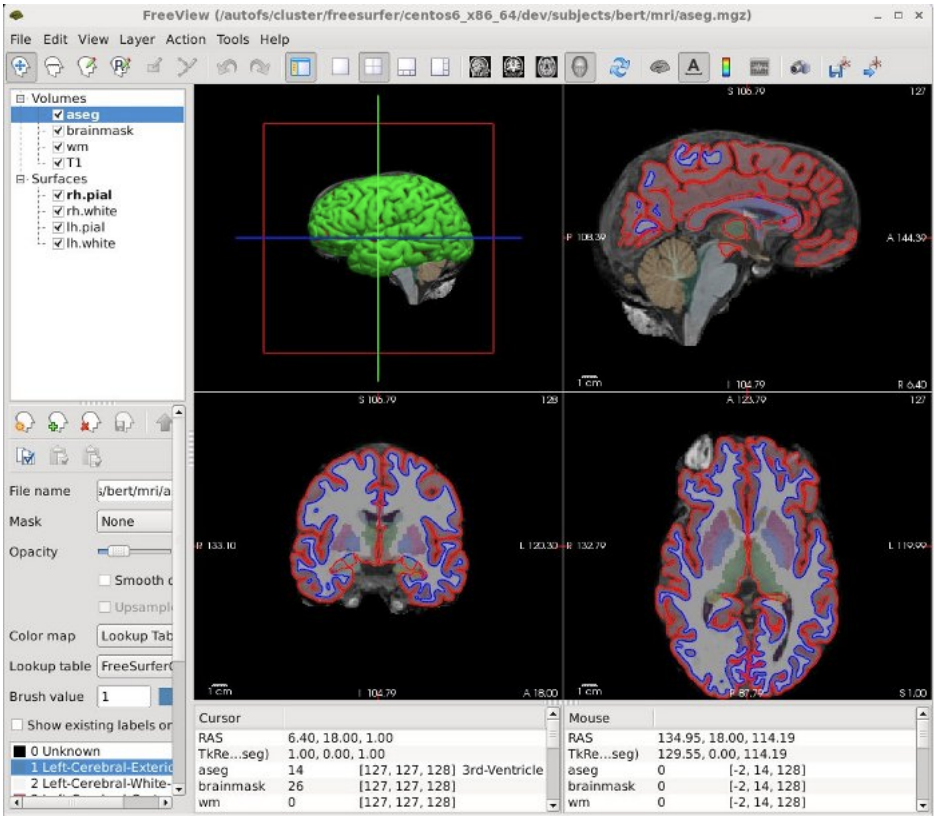
8.1.3.2 Recon-all

```
export SUBJECTS_DIR=<path to subject directory>
recon-all -i sample-001.nii.gz -s bert -all (creates a folder called bert in SUBJECTS_DIR)
```

8.1.3.3 可视化输出

```
cd $SUBJECTS_DIR
freeview -v \
    bert/mri/T1.mgz \
    bert/mri/wm.mgz \
    bert/mri/brainmask.mgz \
    bert/mri/aseg.mgz:colormap=lut:opacity=0.2 \
    -f \
    bert/surf/lh.white:edgecolor=blue \
    bert/surf/lh.pial:edgecolor=red \
    bert/surf/rh.white:edgecolor=blue \
    bert/surf/rh.pial:edgecolor=red
```

可视化输出的结果为



8.1.4 参考资料

https://surfer.nmr.mgh.harvard.edu/fswiki/rel6downloads

8.2 MATLAB

8.2.1 概述

MATLAB是矩阵实验室（Matrix Laboratory）之意。除具备卓越的数值计算能力外，它还提供了专业水平的符号计算，文字处理，可视化建模仿真和实时控制等功能。

MATLAB的基本数据单位是矩阵，它的指令表达式与数学,工程中常用的形式十分相似,故用MATLAB来解算问题要比用C,FORTRAN等语言完相同的事情简捷得多.在新的版本中也加入了对C,FORTRAN,c++ ,JAVA的支持.可以直接调用,用户也可以将自己编写的实用程序导入到MATLAB函数库中方便自己以后调用，此外许多的MATLAB爱好者都编写了一些经典的程序，用户可以直接进行下载就可以用，非常的方便。

MATLAB的基础是矩阵计算,但是由于他的开放性,并且mathwork也吸收了像maple等软件的优点,使MATLAB成为一个强大的数学软件。当前流行的MATLAB 6.5/7.0包括拥有数百个内部函数的主包和三十几种工具包(Toolbox).工具包又可以分为功能性工具包和学科工具包.功能工具包用来扩充MATLAB的符号计算,可视化建模仿真,文字处理及实时控制等功能.学科工具包是专业性比较强的工具包,控制工具包,信号处理工具包,通信工具包等都属于此类。

开放性使MATLAB广受用户欢迎.除内部函数外,所有MATLAB主包文件和各种工具包都是可读可修改的文件,用户通过对源程序的修改或加入自己编写程序构造新的专用工具包.

8.2.2 MATLAB加载过程

在集群上使用MATLAB时，需要先加载相应的模块（module）信息。

```
module load apps/matlab/2021b
```

注意上面的命令加载的2021b这个版本的MATLAB，如果想加载其他版本的MATLAB可以使用下面的命令来查询其他可以使用的MATLAB版本（如果还有其他版本的MATLAB的情况下）：

```
module avail matlab
```

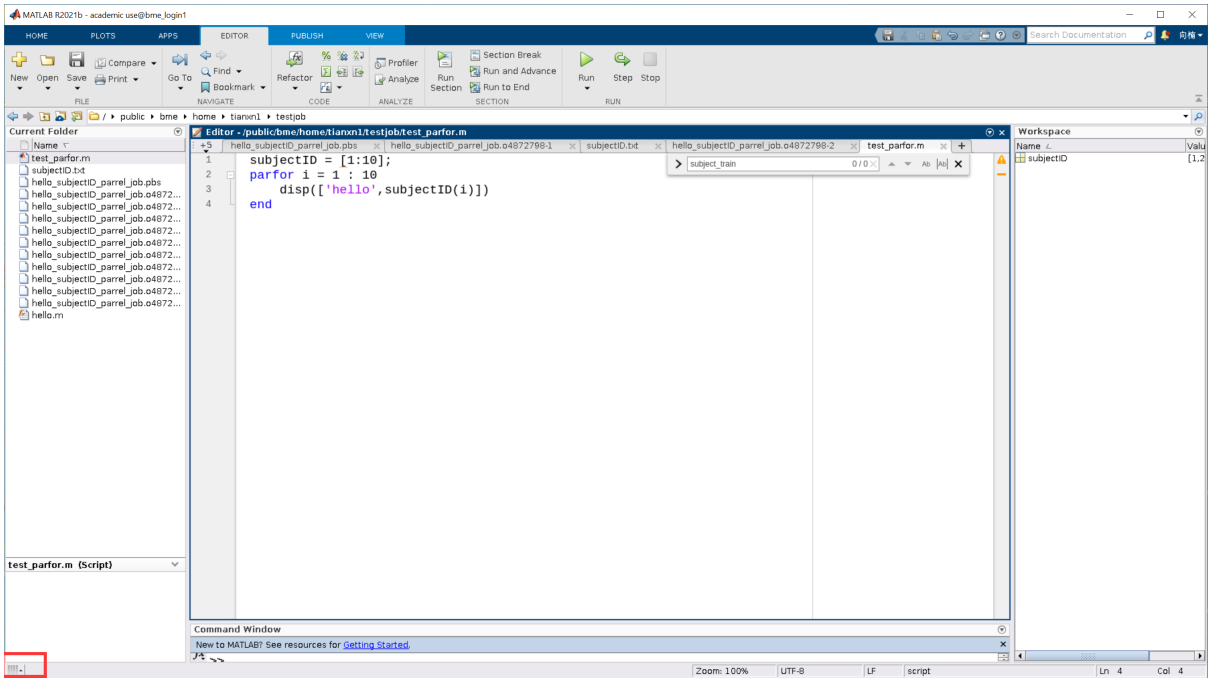
关于**module**这个命令更多的细节可以在user guide() 中查到.

8.2.3 并行计算的使用说明

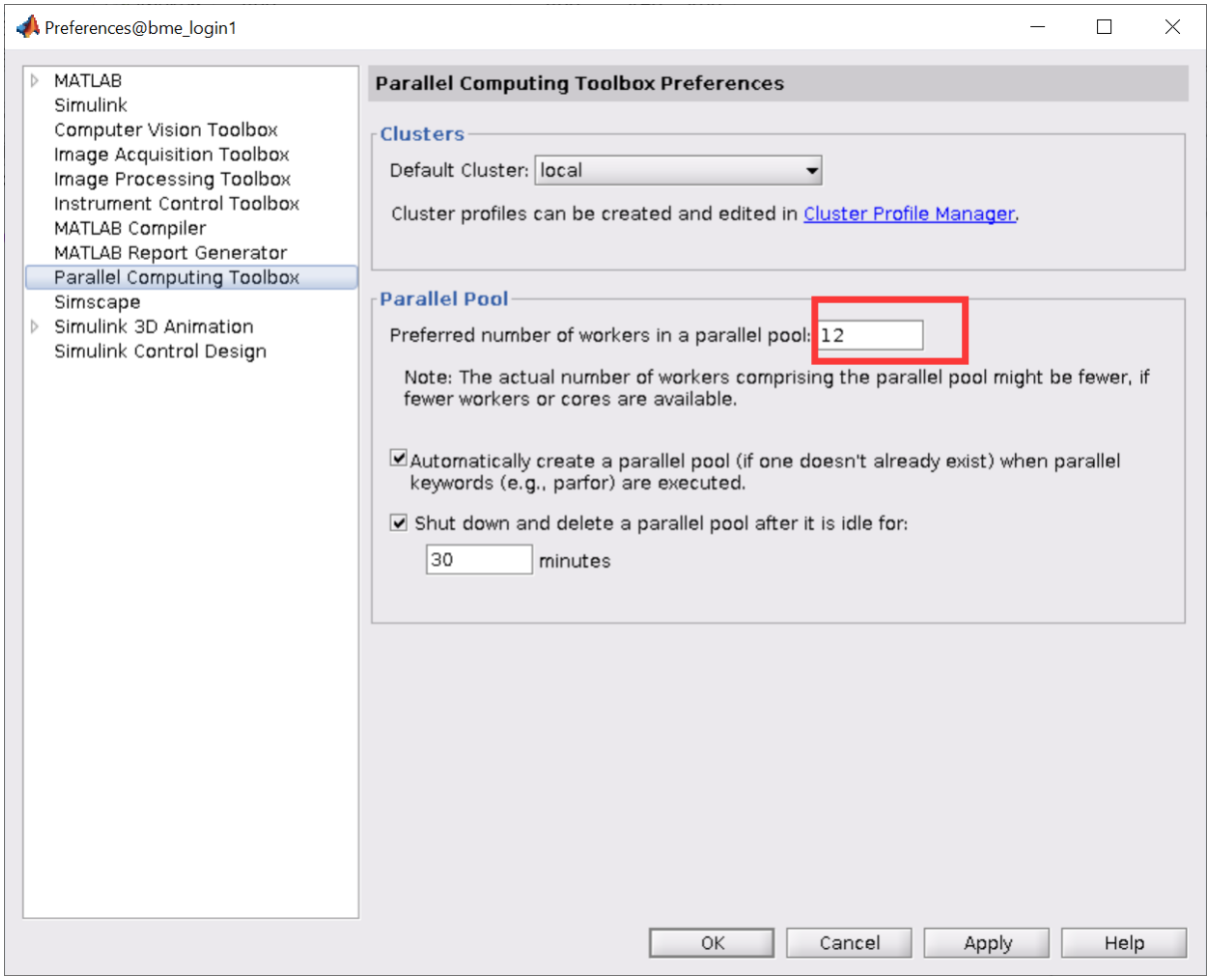
使用MATLAB自带的parfor方法

```
module load apps/matlab/2021b
```

首先，通过点击MATLAB左下角的矩形阵列按钮来设置并行计算所需的资源



然后设置图片中选中的地方，这个框内的数值即为能够同时并行的执行的任务数



然后在该需要进行并行计算的地方使用parfor即可，这里以test_parfor.m为例:

```
parfor i = 1 : 10
  disp(['hello',num2str(i)])
end
```

结果是：

```
>> test_parfor
hello1
hello2
hello3
hello4
hello5
hello6
hello7
hello8
hello9
hello10
```

8.2.4 使用公共节点并行并行计算方法（PBS）

首先使用MobaXterm远程连接到任意一个公共结点上，这里以运行一个打印Hello World的程序为例，首先用户需要创建一个MATLAB的函数文件hello.m,这个函数的主要功能是打印Hello World 和 一个受试者的ID

```
function hello(subjectID)
printf([‘Hello’,num2str(subjectID)])
```

为了能够使用jobID去对应不同的subjectID，此时需要先创建一个subjectID.txt.然后使用下面的PBS脚本进行调用：

```
#PBS -N hello_subjectID_parrel_job
#PBS -j oe
#PBS -l nodes=1:ppn=1
#PBS -l mem=5gb
#PBS -l walltime=05:00:00
#PBS -q bme_pub_cpu
#PBS -t 1-10
cd ~/testjob/
module load apps/matlab/2021b
array=($(cat ./subjectID.txt))
subjectID=${array[${PBS_ARRAYID} - 1]}
matlab -nodisplay -nodesktop -nosplash -r “subj=$subjectID;hello(subj);exit;”
```

关于以上脚本，接下来做一个详细的介绍，首先是脚本中的第一行**#PBS -N hello_world_parrel_job**，这一句中的hello_world_parrel_job代表了用户的所提交的任务名，用户可以根据自己所提交任务的实际意义来对其进行命名。第二行**#PBS -j oe**代表的是输出提交的作业所产生的输出和错误（o:输出 e:错误）接下来的三到五行代表的是用户所申请的系统资源，**#PBS -l nodes=1:ppn=1 #PBS -l mem=5gb #PBS -l walltime=05:00:00，nodes**代表结点的数量，**ppn**代表核心对的数量，**mem**
代表内存的大小，**walltime**代表所申请资源的时间，第六行**#PBS -q bme_pub_cpu** 代表指定任务提交的队列名称，目前公共结点上只申请到这一个队列，因此都这样写就可。**#PBS -t 1-10**代表的是所提交jobid的范围，这里是1-10，具体的范围可以根据用户的实际情况进行修改；**cd ~/testjob/**，代表切换到根目录的**testjob**文件夹下，**module load apps/matlab/2021b**，加载MATLAB，**array=($(cat ./subjectID.txt))** 读取**subjectID.txt**文本中的内容并将其赋值给array.
subjectID=${array[${PBS_ARRAYID} - 1]},获取数组array中的具体某个元素的值，PBS_ARRAYID是一个常量表示的是这个任务是第几个任务，即其取值为**-t** 后面所跟的范围。**matlab -nodisplay -nodesktop -nosplash -r “subj=$subjectID;hello(subj);exit;”**最后一行代表的是，在不使用GUI（图形化界面）的情况下调用MATLAB，**-nodisplay -nodesktop -nosplash** 这三个选项代表不显示GUI **r “subj=$subjectID;hello(subj);exit;”** ，**-r** 后““里的内容即为所要执行的内容。

在写完**hello_subjectID_parrel_job.pbs**这个文件之后，在公共结点上可以使用**qsub hello_subjectID_parrel_job.pbs** 这个命令去提交该脚本。提交之后会得到一个jobID（如下图）：

```
(base) [tianxn1@hpc-login-sbp testjob]$ qsub hello_subjectID_parrel_job.pbs
4872798[].node1
```

之后可以通过qstat -a -subjectid 查询这个被提交作业的状态，如下图：

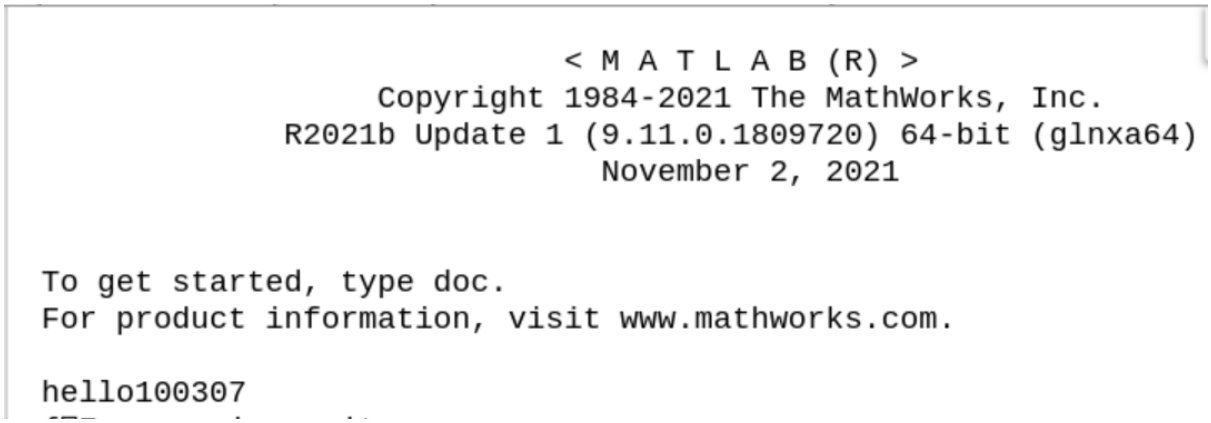
```
(base) [tianxn1@hpc-login-sbp testjob]$ qstat -a 4872798[].node1

node1:
                                                                  Req'd    Req'd       El
ap
Job ID                  Username    Queue           Jobname         SessID  NDS   TSK   Memory   Time    S   Ti
me
----------------------- ----------- --------------- --------------- ------ ----- ------ ------ --------- - ----
-----
4872798[].node1         tianxn1     pub_blad        hello_subjectID_    --      1      1    5gb  05:00:00 R  05:
00:00
```

注：subjectID.txt中的内容为

```
100206
100307
100408
100610
101006
101107
101300
101410
101915
102008
```

最后执行之后的结果为：



8.2.5 使用BME节点的并行计算方法（SLURM）

这里同样需要使用到hello.m这个文件，具体的文件内容见上部分，这里不再重复

slurm系统所使用到的脚本**hello_subjectID_parrel_job.job**内容如下：

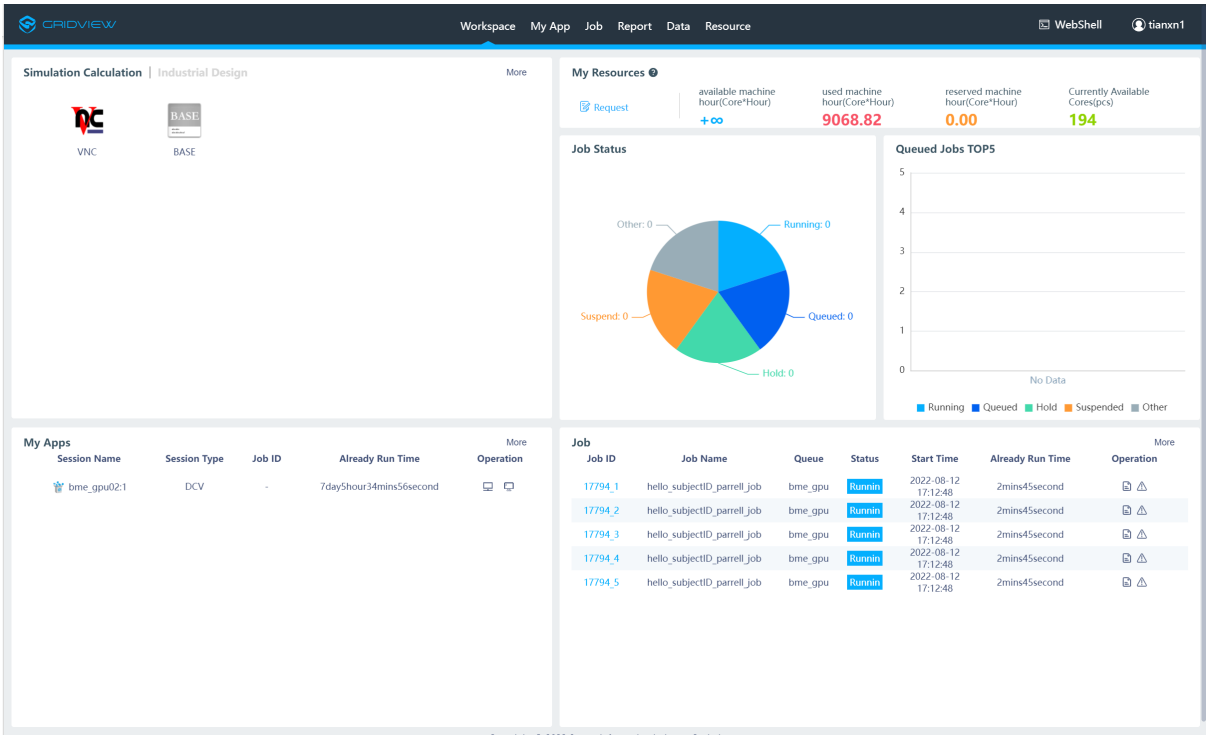
```
#!/bin/bash
#SBATCH -J hello_subjectID_parrell_job
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -o job%j.out
#SBATCH -e job%j.err
#SBATCH --time=00:10:00
#SBATCH --array=1-5

module load apps/matlab/2021b
cd ~/testjob/

array=($(cat subjectID.txt))
subject_ID=${array[${SLURM_ARRAY_TASK_ID}-1]}
matlab -nodisplay -nodesktop -nosplash -r "subj=$subject_ID;hello(subj);exit;"
```

关于以上脚本接下来做一个详细的解释，第一行**!/bin/bash**代表调用bash解释器来运行以下的命令，第二行代表的是作业的名字，第三行代表申请的结点数量，第四行代表的是申请核心的数量，第五行代表打印所有输出到一个指定文件，第六行代表的是打印输出所有的错误到一个指定的文件，第七行代表申请资源的时间，第八行，代表申请并行的数量（注：目前BME的·结点甚至的最大并行数为5）,脚本剩余部分与上一个脚本大体相同，这里就不再做赘述。其中**SLURM_ARRAY_TASK_ID**的作用与**PBS_ARRAYID**作用相同

程序的运行情况可以登录10.15.49.7:6080查看



其中一个并行脚本运行的结果为：

```
< M A T L A B (R) >
                  Copyright 1984-2021 The MathWorks, Inc.
             R2021b Update 1 (9.11.0.1809720) 64-bit (glnxa64)
                               November 2, 2021


To get started, type doc.
For product information, visit www.mathworks.com.

hello100206
```

8.3 FSL用户使用说明

8.3.1 概述

FSL 是一个综合性的分析工具库，用于处理人脑的结构MRI、功能 MRI (fMRI) 和扩散成像 (DTI) 数据。FSL 库中的一些计算量大的程序/脚本，例如 FMRIB 的Diffusion Toolbox(FDT) 中的 bedpostx，或Tract-Based Spatial Statistics(TBSS) 中的 tbss_2_reg，都内置了 SGE 集群的自提交机制。对于其他没有自提交机制的 FSL 程序/脚本，也可以使用 FSL 脚本“fsl_sub”将它们提交到计算机集群。但是，此脚本也是为 SGE 集群设计的。 由于我们的系统使用了 PBS 调度器，这些具有自提交机制的 FSL 程序/脚本以及 FSL 脚本“fsl_sub”需要进行修改，以便使用我们系统的并行计算能力。我们为用户提供修改后的 FSL 程序：bedpostx（新文件名：launch_bedpostx）、tbss_2_reg（新文件名：launch_tbss_2_reg）和 fsl_sub（新文件名：fsl_sub_PBS）。
注：如果用户仅计划以串行方式使用 FSL 程序/脚本，则无需修改。

8.3.2 安装设置

在开始使用 FSL 之前加载模块。

```
module load apps/fsl/6.0
```

注：此命令将加载FSL 6.0版本。关于module命令的更多细节可以在用户指南（）中找到。

8.3.3 FSL使用

1. 交互式运行： 我们的系统可以通过 MobaXterm 远程访问。 有关 MobaXterm 的更多信息，请参见以下网页 ()。 登录后，用户可以通过在终端中键入以下内容来启动 FSL GUI：

```
fsl &
```

2. 批量运行（串行）: 实现批处理运行 FSL，用户需要知道他/她打算使用什么 FSL 程序/脚本，并提供该 FSL 程序/脚本的参数。 例如，如果用户计划执行以下 FSL 命令：

```
fslmaths inputVolume -add inputVolume2 output_volume
```

那么用户需要将该命令添加到以下 PBS 脚本中：

```
#PBS -N fslmaths
#PBS -j oe
#PBS -l nodes=1:ppn=1
#PBS -l mem=4gb
#PBS -l walltime=01:00:00
#PBS -M abc123@shanghaitech.edu.cn
#PBS -m abe

# $PBS_O_WORKDIR为pbs系统为该进程分配的存储地址，一般为/tmp
cd $PBS_O_WORKDIR
module load apps/fsl/6.0
fslmaths inputVolume -add inputVolume2 output_volume
```

3. 批量运行（并行）: 如本文档概述部分所述，一些 FSL 程序/脚本在SEG集群中具有内置的自提交机制能够实现并行计算。由于我们的系统使用 PBS 调度程序，因此需要修改这些 FSL 程序/脚本和 FSL 脚本并行运算需要的“fsl_sub”，以便在我们的系统中以并行方式执行批处理。这里我们为用户提供了两个修改后的FSL程序，分别是bedpostx（修改后的程序文件名：*launch_bedpostx*）和tbss_2_reg（修改后程序的文件名：*launch_tbss_2_reg*），以及fsl_sub修改后的脚本（新文件名：*fsl_sub_PBS*）， 以满足使用 bedpost 和 tbss_2_reg 时并行计算的需要。 以下是如何使用这些修改后的程序的说明。
4. 用户可以先下载这些修改后的程序，然后将fsl_sub_PBS、launch_bedpostx和launch_tbss_2_reg这些文件的路径添加到用户根目录下的.bashrc文件中。 例如，如果您将这三个文件放在~/work目录下，那么您可以将“export PATH=$PATH:~/work”添加到您的 .bashrc 文件中。

a) bedpostx

为了尝试脚本 launch_bedpostx，用户需要从 FSL 网站下载示例数据集。 用户可以将此数据集下载到临时文件夹中，然后使用以下命令将其解压缩：

```
mkdir scratch #创建一个临时文件夹
cd scratch
wget -c http://fsl.fmrib.ox.ac.uk/fslcourse/downloads/fdt.tar.gz
tar -zxf fdt.tar.gz
```

然后用户需要将brain mask文件（nodif_brain_mask.nii.gz）复制到 subj1 目录，然后删除现有的输出文件夹“subj1.bedpostX”，因为用户将通过尝试此处的示例重新生成输出：

```
cd fsl_course_data/fdt2/
cp ./subj1.bedpostX/nodif_brain_mask.nii.gz ./subj1
rm -rf subj1.bedpostX
rm -rf subj1_2fibres.bedpostX
launch_bedpostx ./subj1 --nf=2 --fudge=1  --bi=1000
```

这个 launch_bedpostx 脚本将向系统提交三个 PBS 作业，名称分别是：bpx_preproc、bedpostx、bpx_postproc。 用户可以使用 qstat 检查这些作业的状态：

```
qstat
```

“bpx_preproc”作业是bedpostx之前的预处理，它将文件复制到指定路径，并且几乎立即完成。 “bedpostx”作业是一个 PBS 作业数组，其每个子作业为每个切片排队。 用户可以使用 qstat 的“-t”选项来检查这些子作业的状态。 在“bedpostx”的所有子作业成功

完成后，“bpx_postproc”作业将启动。

b) TBSS

为了尝试脚本 launch_tbss_2_reg，用户需要从 FSL 网站下载示例数据集。 用户可以将此数据集下载到临时文件夹中，然后使用以下命令将其解压缩：

```
cd scratch
wget http://fsl.fmrib.ox.ac.uk/fslcourse/tbss.tar.gz
tar zxf tbss.tar.gz
```

用户应删除现有的输出文件夹“precomputed_registration”，因为用户将通过尝试此处的示例重新生成输出:

```
cd tbss
rm -rf precomputed_registration
fsl_sub_PBS tbss_1_preproc *.nii.gz
```

上面的命令会向我们的系统提交一个 TBSS 预处理的 PBS 作业。 完成此作业后，用户可以尝试脚本 launch_tbss_2_reg，如下所示：

```
launch_tbss_2_reg -n
```

这将向系统提交一个名为“tbss_2_reg”的 PBS 作业数组。 用户可以使用带有“-t”选项的 qstat 来检查此作业数组的各个子作业的状态。

8.3.4 教程

一些关于FSL教程的讲座PPT可以通过 FSL 课程网站找到：http://fsl.fmrib.ox.ac.uk/fslcourse/

如果您需要有关此主题的进一步技术帮助，请联系（）.

8.3.5 参考

1. http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/
2. https://www.mail-archive.com/hcp-users@humanconnectome.org/msg01287.html
3. https://www.jiscmail.ac.uk/cgi-bin/webadmin?A2=fsl;bd7f223b.1308

9. 计算场景举例

10. 可视化应用