

# 集群使用手册

本公司集群配置与北大工作站非常类似，因此可以直接参考此使用手册  
<https://bicmr.pku.edu.cn/~wenzw/pages/index.html>

使用过程中遇到任何问题均可咨询集群运维工程师：@张军广

【请注意：部分集群设计与现行方案略有不同，宣讲会后将很快调整为当前方案】

## 集群组成说明

### 管理节点

按组划分管理节点：

MR组 gpu-61

US组 gpu-62

CT组 gpu-63

GR组 gpu-64

CTA组 gpu-65

登录方式：

1. 在连接VPN情况下，希望登录gpu-XX
  - a. `ssh {your_user_name}@10.2.111.{XX}`
2. 在未连接VPN情况下，希望登录gpu-XX
  - a. 【不推荐使用，随时可能关闭】
  - b. `ssh -p {XX}22 {your_user_name}@122.115.38.132`

说明：

1. 管理节点可以SSH直接登录
2. 各组请使用对应管理节点
3. 管理节点用于提交计算任务、DEBUG
4. 以上节点属于各组分区：
  - a. CT、CTA、MR、US、GR

### 备用公用管理节点兼低性能计算节点

gpu-57 ~ gpu-60

说明：

1. 备用管理节点可以SSH直接登录
2. 各组可公用备用管理节点
3. 备用管理节点可用于DEBUG调试
4. 长时间计算任务请提Slurm Job，否则可联系管理员直接Kill
5. 以上节点属于统一分区：
  - a. backup

## 计算节点

gpu-66 ~ gpu-84 （19台）

说明：

1. 所有计算节点统一在一个share分区中
2. 所有计算节点均为8卡nvidia-3090
3. 所有计算节点均不可SSH直接登录
4. 以上节点属于统一分区：
  - a. gpu

## 存储

### 高性能共享存储

1. /slurm\_data
  - a. 单个用户用量限额1TB
  - b. 存放环境、代码以及训练数据
2. /projects
  - a. 项目公共热数据存储

### 低性能共享存储

1. /data142T/project
  - a. 项目冷数据归档使用
2. /data142T/users
  - a. 个人冷数据归档使用

### 本地存储

除以上两个共享存储外，仅计算节点本身可以访问的存储称为本地存储。

请勿使用本地存储，面临清空风险。之前已使用的本地存储，请尽快迁移到共享存储中。

## 存储权限设计

公司数据权限设计体现在分组上

1. Level0: intern组
  - a. 所有研发实习生
  - b. 所有研发正式员工
2. Level1: algo组:
  - a. 所有研发正式员工
3. Level2: 项目组
  - a. 属于该项目组的正式员工
  - b. 现有: (CT、CTA、DR、MG\_DBT、MG\_X、MRI、SR、US)

创建文件时默认为最高Level的属组，以及750的权限

公用数据原则: 最少可用

情景样例:

1. A组员工参与B组项目，需要使用B组部分数据时:
  - a. 将所需数据拷贝到共享目录，调整属组为该员工最高level的属组或algo组。
    - i. 属组调整命令: chgrp
  - b. 项目结束后删除数据
2. A组实习生需要使用A组数据时:
  - a. 将所需数据拷贝到共享目录，调整属组为intern组

**禁止**直接将数据权限设置为全体可读

## 计算资源限制

任务时长限制

1. salloc、sbatch和srun任务执行最长时间为7天
2. salloc、sbatch和srun任务默认时长3天

个人资源限制

1. 每账号最多同时申请的GPU为16块

排队机制

- 1. 在计算节点已满情况下，新提交的任务会自动排队
- 2. 满足以下条件的任务将优先计算：
  - a. 长时间等待的任务（3天后优先级达到上限）
  - b. 小任务优先（限制时间短的，通过-t指定）
  - c. 公平分配，自身已消耗资源少的会优先得到分配（CPU+GPU）

请在不使用服务器时及时释放自己的任务。  
计算资源使用情况将被记录和审核，严重浪费计算资源的行为会被警告。

集群使用说明

集群状况查询

snode

```
fei.gao@gpu-63:~$ snode
```

NODE	CPU	LOAD	MEM	GPU	JOB [ID USER TIME_LEFT]
gpu-57	0/32	0.38	0/39G	0/4	
gpu-58	0/32	1.11	0/58G	0/2	
gpu-59	2/56	0.75	4/117G	0/8	[22319 nan.an 49692-22:54:00]
gpu-60	24/56	2.21	96/117G	6/8	[23177 tingyang.yang 49708-06:36:13] [23266 tingyang.yang 49710-06:26:18]
gpu-61	0/56	0.17	0/117G	0/7	
gpu-62	0/56	10.71	0/117G	0/7	
gpu-63	0/56	2.51	0/234G	0/8	
gpu-64	28/56	10.31	136/234G	8/8	[21037 zhangkx 49685-22:09:11] [21086 guanglie.jia 49686-03:32:01] [21187 zhangkx 49687-03:32:01]
gpu-65	0/56	1.33	0/234G	0/8	
gpu-66	32/56	21.92	96/234G	8/8	[23234 daiyt 2-16:58:10]
gpu-67	20/56	1.69	16/234G	1/8	[21795 nan.an 49690-05:57:53]
gpu-68	32/56	6.06	144/234G	8/8	[21214 mingyu.yang 49687-02:53:42] [22481 zhangkx 49694-22:39:32] [22551 zhangkx 49695-22:39:32]
gpu-69	0/56	0.22	0/234G	0/8	
gpu-70	16/56	1.26	64/234G	4/8	[23217 zhixing.zhang 3-04:08:18]
gpu-71	0/56	3.64	0/234G	0/8	
gpu-72	0/56	2.25	0/234G	0/8	
gpu-73	33/56	38.13	128/234G	8/8	[23227 zhangjq 49709-07:53:20]
gpu-74	34/56	12.40	132/234G	8/8	[23020 junjie.hou 49705-07:31:16] [23173 tingyang.yang 49708-06:31:51] [23262 zhaoss 13:49:39]
gpu-75	56/56	0.21	128/234G	8/8	[22699 yuhj 6-23:24:59]
gpu-76	22/56	7.63	132/234G	8/8	[22961 haowen.ma 49703-22:00:49] [22967 wangbm 49703-22:57:52]
gpu-77	32/56	27.54	192/234G	8/8	[23259 hao.yu 49710-03:53:36]
gpu-78	54/56	2.60	216/234G	8/8	[22626 yanghao 49696-05:02:24] [23108 shichen 4:48:17]
gpu-79	16/56	1.87	60/234G	0/8	[21039 mingyu.yang 49685-22:10:26] [22905 shuai.qi 49702-22:35:47] [22906 shuai.qi 49702-22:35:47]
gpu-80	52/56	14.30	204/234G	8/8	[21038 mingyu.yang 49685-22:09:56] [23036 yanghao 49706-09:53:48] [23037 yanghao 49706-09:53:48]
gpu-81	14/56	0.25	60/234G	2/8	[21187 sunal 49686-22:28:12] [22697 wangbm 49697-05:46:29] [22842 sunal 49697-05:46:29]
gpu-82	48/56	26.31	192/234G	8/8	[23263 quanlin.wu 2-22:18:23]
gpu-83	8/56	2.11	32/234G	0/8	[22951 zhao.zhang 49703-09:31:33]
gpu-84	36/56	25.66	200/234G	8/8	[21288 nan.an 49687-23:01:42] [22942 na.gao 49703-06:43:11] [23230 na.gao 49703-06:43:11]

计算节点清单

资源使用情况

任务情况

特殊状态说明

NODE LEGEND

- draining/drained (not allocating new jobs)
- # reserved for teaching
- \* down
- \*\* poweroff/not responding

提交、查看与取消计算任务

## **srun**

用于将一个命令行任务提交到计算节点执行。

常用参数：

-h	获得帮助
-w gpu-66	指定计算节点
-p gpu	指定分区
-G 8	指定需要的GPU数量
--cpus-per-gpu=4	指定每个GPU配套的cpu数量
--mem-per-gpu=8G	指定每个GPU配套的内存大小
-N 1	指定需要的节点数量
-t 60	指定最长使用时长（分钟单位）

样例：

```
srun -w gpu-66 -p gpu ls
```

```
srun -w gpu-66 -p gpu -G8 nvidia-smi
```

```
srun -w gpu-66 -p gpu -G8 python my_main.py
```

## **sbatch**

用于将多个命令行任务提交到计算节点执行。

与srun不同之处在于，sbatch提交的是一个shell脚本，可以同时包括多个命令。

样例：

```
1  #!/bin/bash
2
3  # ==== 此段落为sbatch控制参数
4  # ==== #SBATCH之间不能有空格
5  #SBATCH -o joblog/job.%j.out
6  #SBATCH -e joblog/job.%j.out
7  #SBATCH --partition=backup
8  #SBATCH -w gpu-59
9  #SBATCH --gres=gpu:1
10 #SBATCH -t 24:00:00
11 #SBATCH -J SLURMTEST
12
13 # ==== 此后为所需执行的脚本，与一般的shell脚本没有区别
14
15 nvidia-smi
16
17 hostname
18
19 which python
20
21 source /slurm_data/share/software/anaconda3/bin/activate
22 conda activate yzdeploy-3.6
23
24 which python
25
26 python hello_slurm.py
```

## salloc

【注意，集群调整后，gpu分区的所有计算节点都无法ssh登录，哪怕有任务在上面也是一样】

交互式的申请一段时间的资源

在提交salloc任务后，将跳转到一个新的shell中。

在此shell里，可用计算资源将被限制在salloc申请的资源范围内。

此命令会独占一部分计算资源，可以随时提交srun、sbatch任务，不再需要重新排队。

流程如下图所示：

```

(base) ➔ ~ salloc -w gpu-63
salloc: Granted job allocation 23298
salloc: Waiting for resource configuration
salloc: Nodes gpu-63 are ready for job
(base) fei.gao@gpu-63:~$ exit
exit
salloc: Relinquishing job allocation 23298
salloc: Job allocation 23298 has been revoked.
(base) ➔ ~ █

```

常用参数与srun类似，可通过-h详细查看

样例：

```
salloc -w gpu-66 -p gpu -G8
```

请勿使用salloc长时间占用资源而无实际操作，一经发现管理员有权直接终止

## squeue

查看所有已经提交的任务

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
23287	CT-share	Vertebra	na.gao	R	11:13:50	1	gpu-84
22319	CT-share	interact	nan.an	R	17-23:27:48	1	gpu-59
21795	CT-share	interact	nan.an	R	20-16:23:55	1	gpu-67
22942	CT-share	interact	na.gao	R	7-15:38:37	1	gpu-84
21288	CT-share	interact	nan.an	R	22-23:20:06	1	gpu-84
23020	CTA-share	interact	junjie.h	R	5-14:50:32	1	gpu-74
23217	CTA-share	interact	zhixing.	R	1-17:06:14	1	gpu-70
23175	CTA-share	python	tingyang	R	2-15:48:43	1	gpu-74
23174	CTA-share	python	tingyang	R	2-15:49:38	1	gpu-74
23173	CTA-share	python	tingyang	R	2-15:49:57	1	gpu-74
22951	CTA-share	interact	zhao.zha	R	7-12:50:15	1	gpu-83
21086	GR-share	interact	nobody	R	24-18:49:47	1	gpu-64
21821	GR-share	interact	wangzt	R	20-11:42:09	1	gpu-64

JOBID: 任务唯一编号, 当出现未知问题需要寻求管理员帮助时请带上任务编号

PARTITION: 任务所在分区

NAME: 任务名称

USER: 任务提交人

ST: 任务状态, 常见包括

PD PENDINGJob is awaiting resource allocation.

R RUNNINGJob currently has an allocation.

TIME: 任务已经执行的时长

NODES: 任务占用的计算节点数量

NODELIST (REASON) : 任务使用的计算节点名称或出现问题的原因

样例:

```
squeue -u {your_user_name}
```

## scancel

用于取消任务

样例:

```
scancel {your_job_id}
```

## sreport

查看自己最近的资源使用情况

样例:

```
sreport cluster AccountUtilizationByUser --tres="cpu,gres/gpu" start=2021-09-01 where user={your_user_name}
```

## 共享软件包

集群提供共享软件方案, 用于处理计算节点需要不同版本软件的问题。

例如: A同学需要cuda/10.2, B同学需要cuda/11.5

module -h 查看软件包加载帮助

module ava 查看可用的包

module add 添加包



目前集群提供以下共享软件:

anaconda/3、cuda/10.2、cuda/11.1、cuda/11.5、rar、tmux/2.8

如需其他软件, 请联系集群管理员安装

已收集到的软件按照需求:

1. cuda对应版本的cudnn

## 完整使用样例:

1. `module add anaconda/3` # 为自己添加上anaconda
2. `conda init` # 初始化这个conda为默认conda
  - a. `conda init zsh` (不同shell使用不同命令)
3. `conda activate yzdeploy-3.6` # 激活ct部署环境
4. `cd /slurm_data/share/examples` # 切换到样例文件夹
5. `python hello_slurm.py` # 本地测试
  - a. Hello Slurm!
  - b. No GPU
6. `srun -p CT-share -w gpu-59 -G 1 python hello_slurm.py` # srun单次执行测试
  - a. Hello Slurm!
  - b. 1 GPU
7. `sbatch hello_sbatch.sh`
  - a. Submitted batch job {JOBID}
  - b. 去joblog/job.{JOBID}.out 看输出即可
8. `cd /slurm_data/fei.gao/Projects/d2_test/`
9. `sjob`查看任务大致状态, 比如在哪个node上执行
10. 通过joblog/job.JOB\_ID.err 和joblog/job.JOB\_ID.out查看标准错误和标准输出
11. `scontrol show job JOBID`查看任务执行情况
12. `snode`查看资源消耗情况

## 常见问题

slurm 官方FAQ: <https://slurm.schedmd.com/faq.html>

1. 管理节点重启是否会影响现有计算任务?
  - a. `salloc`和`srun`任务会断掉
  - b. `sbatch`任务会继续执行

- c. 因此推荐大家都使用sbatch方式提交计算任务
- 2. sinfo后面的\*是什么意思
  - a. 代表是默认分区
- 3. -n 或 --ntasks 是什么意思
  - a. 代表可以并行起来的任务数量，默认为1。
  - b. 参考: [https://blog.csdn.net/qg\\_38591130/article/details/112769431](https://blog.csdn.net/qg_38591130/article/details/112769431)
- 4. srun后使用ctrl+C停不下来
  - a. 开始另外一个terminal, 使用scancel终止job
- 5. sbatch脚本申请GPU后报错no device found
  - a. 检查sbatch脚本, #SBATCH 中间是没有空格的
- 6. conda install 速度太慢怎么办?
  - a. 可以更换为国内源, 参考:
  - b. <https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/>
  - c. 上面这个镜像站里还有pypi、ubuntu等等各种源的镜像, 可以解决海量类似问题
  - d. 可以自行配置一个代理, 有需要可以联系 @高飞-研发部
- 7. 切换home后, 发现在老节点上登陆tmux, 其home没有切换到共享盘
  - a. tmux kill-server && tmux
  - b. 确定没有重要程序运行后, 将老tmux服务杀死, 然后重启一个新的tmux即可
- 8. 如何debug?
  - a. 请在管理节点进行debug
- 9. 如何在不同计算节点之间同步数据?
  - a. 请将数据放在共享盘
- 10. 如何在提供的公共环境中安装新的包:
  - a. 公共环境无法安装自己的包
  - b. 如果想在公共环境基础上安装自定义的包, 可以先使用conda clone复制一个环境
  - c. 例如: conda create -n yd3.6 --clone yzdeploy3.6
- 11. nvidia-smi速度太慢
  - a. 请管理员执行如下命令: sudo nvidia-persistenced --persistence-mode
- 12. salloc后执行srun报错
  - a. 打开一个新的终端
  - b. 或通过exit退出当前的salloc
- 13. 我的任务没有执行怎么办?
  - a. 通过sjob查看没有执行的原因, 解释可以在下面找到。
  - b. 如果是资源限制, 但检查发现节点资源充足 (snode), 自己也没有占据更多资源, 请联系管理员
  - c. [http://scc.ustc.edu.cn/zlsc/user\\_doc/html/slurm/slurm.html#id9](http://scc.ustc.edu.cn/zlsc/user_doc/html/slurm/slurm.html#id9)
- 14. 在sbatch脚本里, 使用conda activate会报无法activate, 需要conda init的错误:

- a. 使用该环境绝对路径的python 【推荐】
  - b. 在conda activate前主动激活一下 【推荐】
    - i. source /slurm\_data/share/software/anaconda3/bin/activateconda activate yzdeploy-3.6
  - c. 将当前环境切换到所需环境后，再使用sbatch执行 【不推荐，容易出错】
15. ImportError: libGL.so.1: cannot open shared object file: No such file or directory
- a. 找管理员安装: apt install libgl1-mesa-glx
16. ntask和cpus-per-task的区别
- 

**From this question:** if every node has 24 cores, is there any difference between these commands?

```
sbatch --ntasks 24 [...]  
sbatch --ntasks 1 --cpus-per-task 24 [...]
```

**Answer:** (by Matthew Mjelde)

Yes there is a difference between those two submissions. You are correct that usually **ntasks** is for **mpi** and **cpus-per-task** is for **multithreading**, but let's look at your commands:

For your first example, the `sbatch --ntasks 24 [...]` will allocate a job with 24 tasks. These tasks in this case are only 1 CPUs, but may be split across multiple nodes. So you get a total of 24 CPUs across multiple nodes.

For your second example, the `sbatch --ntasks 1 --cpus-per-task 24 [...]` will allocate a job with 1 task and 24 CPUs for that task. Thus you will get a total of 24 CPUs on a single node.

In other words, a task cannot be split across multiple nodes. **Therefore, using `--cpus-per-task` will ensure it gets allocated to the same node, while using `--ntasks` can and may allocate it to multiple nodes.**

a.

---

17. 在一个sbatch执行后，节点变成drained或draining状态：
- a. 使用sinfo -R检查变化原因，通常应该是：Kill task failed
  - b. 参考: <https://www.mail-archive.com/slurm-users@lists.schedmd.com/msg06338.html>
  - c. 可能的原因: Sometimes they are somehow able to generate I/O in a way that slurr thinks the job is finished, but the OS is still catching up on the I/O, and then slurr tries to kill the job... (即产生了一个slurm以为结束，但依然存在的I/O任务，导致slurm

c. kill失败)

d. 推荐在sbatch脚本后添加: sleep 60, 确保任务完全终止。

18. salloc后, ssh登录发现多进程执行效率极低:

a. 利用-c参数增加需要的CPU数量。例如: salloc -p CTA-share -w gpu-76 -c 10

19. 发现log内时间不对

a. 联系管理员调整一下时区设置