**HPC常用命令**

华北电力大学高性能计算平台HPC部分共38台高性能CPU计算节点，每个节点有64核，CPU资源池共有2432核，采用Slurm任务调度技术进行管理。

**Slurm简介：**Slurm（http://slurm.schedmd.com）是一个开源，容错，高度可扩展的集群管理和作业调度系统，适用于大型和小型Linux集群，已被全世界的国家超级计算中心广泛采用。



**Slurm 基本命令：**

|  |  |
| --- | --- |
| **Slurm命令** | **功能** |
| sinfo | 查看集群资源使用状态 |
| squeue | 查看提交作业的排队情况 |
| scontrol | 查看和修改作业参数 |
| sacct | 查询已经结束的作业信息 |
| sbatch | 批处理作业提交 |
| scancel | 删除作业 |

**sinfo：查看集群资源使用状态**

|  |  |
| --- | --- |
| **Slurm命令** | **功能** |
| sinfo | 显示集群全部节点信息 |
| sinfo -N | 按每个节点一行的格式显示节点信息 |
| sinfo -n<nodes\_name>注：<nodes\_name>可选cpu01~cpu36,cpu-mem01~cpu\_mem02 | 显示指定节点信息 |
| sinfo -N --states=<node\_states>注：<node\_states>可选idel，mix，allocidel为空闲，mix为节点部分核心可以使用，alloc为已被占用 | 显示可用节点状态信息 |
| sinfo -help | 使用帮助信息 |

例如：

$sinfo

分区 可用 时限 节点 状态节点列表

PARTITION AVAIL TIMELIMIT NODES STATE NODELIST

buffer\* up infinite 0 n/a

rtg-1698907986158-cbe228ff8a18 up infinite 0 n/a

rtg-1700703404077-dd9c4d248c1d up infinite 0 n/a

rtg-1700703611526-86f7d669b230 up infinite 3 mix cpu[07,09,29]

rtg-1700703611526-86f7d669b230 up infinite 35 alloc cpu[01-06,08,10-28,30-36],cpu-mem[01-02]

注：用户可忽略前三行数据。idel为空闲，mix为节点部分核心可以使用，alloc为已被占用。

cpu-mem[01-02]为大内存计算节点，单节点配置为64核，2TB内存；cpu[01-36]为普通CPU计算节点，单节点配置为64核，512GB内存。

**squeue：查看提交作业的排队情况**

|  |  |
| --- | --- |
| **Slurm 命令** | **功能** |
| squeue  | 查询队列排队情况 |
| squeue -j<jobids> | 查看指定作业号的作业信息 |
| squeue -n<job\_names> | 查看指定作业名的节点信息 |
| squeue -u<usernames> | 查看指定用户的作业信息 |
| squeue --states=<job\_states> | 查看指定状态的作业信息 |
| squeue -w<nodelist> | 查看指定节点上运行的作业信息 |
| squeue --help | 使用帮助信息 |

例如：

$ squeue

作业号 分区 作业名 用户名 作业状态 运行时间 使用节点数 使用的节点名称（未运行原因）

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

4313 rtg-17007 lammps.s s1202321 PD 0:00 1 (Priority)

4314 rtg-17007 lammps.s s1202321 PD 0:00 1 (Priority)

4279 rtg-17007 Ce-O5 s1202222 R 19:06:09 1 cpu28

4278 rtg-17007 Ce-O4 s1202222 R 23:08:20 1 cpu01

4272 rtg-17007 phonon-2 t5320264 R 1-17:04:08 1 cpu26

4273 rtg-17007 phonon-3 t5320264 R 1-17:04:08 1 cpu27

4251 rtg-17007 CoL\_XaOC s1202221 R 1-18:14:06 2 cpu[05-06]

4271 rtg-17007 phonon-1 t5320264 R 1-18:52:30 1 cpu30

注：作业状态ST：PD为等待，R为运行中

**scontrol: 查看和修改作业参数**

|  |  |
| --- | --- |
| **Slurm命令** | **功能** |
| scontrol show job JOB\_ID | 查看作业信息 |
| scontrol show node=<nodelist> | 查看指定节点上的信息 |
| scontrol hold JOB\_ID | 暂停作业 |
| scontrol release JOB\_ID | 恢复暂停的作业 |
| scontrol update<specification> | 修改作业信息 |
| scontrol update dependency=JOB\_ID | 添加作业依赖 |
| scontrol --help | 使用帮助信息 |

例1：查看作业ID为4325的作业信息

scontrol show job 4325

例2：查看cpu07节点上的信息

scontrol show node=cpu07

例3：暂停作业ID为4324的作业

scontrol hold job 4324

例4：恢复运行作业ID为4322的作业

scontrol release job 4322

例5：修改已提交的作业ID为4322的作业参数，运行时间修改为1天零12小时

scontrol update JobId=74000 TimeLimit=1-12:00:00

例6：修改作业ID为4324的作业，在作业ID为4323完成后才开始运行

scontrol update JobId =4324 dependency=after:4323

**sacct：查询已经结束的作业信息**

例如:

$sacct -j 3321

JobID JobName Partition Account AllocCPUS State ExitCode

------------ ---------- ---------- ---------- ---------- ---------- --------

3321 Si2ZrFe-R+ rtg-17007+ root 64 COMPLETED 0:0

3321.batch batch root 64 COMPLETED 0:0

**sbatch：批处理作业提交**

|  |  |
| --- | --- |
| **Slurm命令** | **功能** |
| sbatch -n<ntasks>,sbatch --ntasks=< ntasks > | 申请的总任务数，默认一个任务一个核心 |
| sbatch --cpus-per-task=<ncups> | 每个任务所需要的核心数，默认为1 |
| sbatch --ntasks-per-node=<n> | 每个节点上运行的任务数 |
| sbatch -J<jobname>,sbatch --job-name=<jobname> | 指定作业名 |
| sbatch -o<out>,sbatch --output=<stdout> | 输出文件，作业脚本中的输出将会输出到该文件 |
| sbatch --error=<stderr> | 错误文件, 作业脚本中的错误将会输出到该文件 |
| sbatch -t<time>,sbatch --time=<time > | 作业最大运行时间 |
| sbatch --exclusive | 独占节点 |
| sbatch -n<nodes\_number>,sbatch --ntasks=<nodes number> | 申请运行的节点数 |
| sbatch -w<nodes>,sbatch –nodelist=<nodes> | 指定申请节点 |
| sbatch -x<nodes>,sbatch --exclude=<nodes> | 排除指定节点 |
| sbatch --get-user-env | 加载当前的环境变量 |
| sbatch -D=<directory>,sbatch --chdir=<directory> | 指定用户工作目录 |
| sbatch --array=<indexes> | 作业数组 |
| sbatch --help | 使用帮助信息 |

**例1：串行作业**

串行程序只使用一个CPU核。下面是一个串行作业的脚本

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name= serial-job

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=1

#SBATCH --mem-per-cpu=4G

#SBATCH --time=00:01:00

#SBATCH --output=%j.out

#SBATCH -- error=%j.err

module purge

module load CentOS/7.9/Anaconda3/2023.03.01

conda activate <your conda env>

python your\_script.py

假设上面作业脚本的文件名为job.sh，通过以下命令提交：

sbatch job.sh

#SBATCH --job-name= serial-job 指定作业名为serial-job。

#SBATCH --nodes=1 申请节点为1，如果作业不能跨节点（MPI）运行，申请节点数应不超过1。

#SBATCH --ntasks=1 指定了所有节点的总任务数为1。

#SBATCH --cpus-per-task=1 指定了每个任务的 cpu 核数，如果大于1，表示多线程任务。

#SBATCH --time=00:01:00 指定了作业的最大运行时间。用HH:MM:SS表示。

#SBATCH --output=%j.out 作业脚本中的输出将会输出到%j.out文件下，%j表示作业号。

#SBATCH -- error=%j.err 作业脚本中的错误将会输出到%j.err文件下，%j表示作业号。

**例2：多线程作业**

一些程序能够通过使用OpenMP、pthreads或Intel Threading Building Blocks等共享内存并行编程模型编写的库来使用多个CPU核。例如，OpenMP程序可以在一个节点上运行多个 "线程 "，每个线程使用一个CPU核。下面是一个多线程作业的脚本。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=multithread

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=4

#SBATCH --mem-per-cpu=4G

#SBATCH --time=00:15:00

#SBATCH --output=%j.out

#SBATCH --error=%j.err

export OMP\_NUM\_THREADS=$SLURM\_CPUS\_PER\_TASK

module purge

module load CentOS/7.9/Anaconda3/2023.03.01

python your\_script.py

对于一个单节点的多线程作业，要确保ntasks和cpus-per-task的乘积等于或小于该节点 上的CPU-cores数量。

**例3：多节点作业、并行MPI作业**

许多科学计算程序使用基于分布式内存的MPI并行库编写，这些代码能够同时使用多个节 点上的多个CPU核。下面是在两个节点上各使用64个CPU核的脚本。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=multinode

#SBATCH --nodes=2

#SBATCH --ntasks-per-node=64

#SBATCH --cpus-per-task=1

#SBATCH --mem-per-cpu=2G

#SBATCH --time=00:05:00

#SBATCH --output=%j.out

#SBATCH --error=%j.err

module purge

module load CentOS/7.9/intel\_compiler/2022u2

module load CentOS/7.9/openmpi/4.0.4

srun your\_prog

在大多数情况下，使用多个节点计算，建议使用每个节点上的所有CPU核。

**例4：多节点、多线程作业**

许多程序通过使用混合OpenMP/MPI方法实现多节点多线程并行。下面是一个多节点、多线 程作业的脚本。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=hybrid-job

#SBATCH --nodes=2

#SBATCH --ntasks-per-node=8

#SBATCH --cpus-per-task=4

#SBATCH --mem-per-cpu=4G

#SBATCH --time=00:01:00

#SBATCH --output=%j.out

#SBATCH --error=%j.err

export OMP\_NUM\_THREADS=$SLURM\_CPUS\_PER\_TASK

module purge

module load CentOS/7.9/intel\_compiler/2022u2

module load CentOS/7.9/openmpi/4.0.4

srun your\_prog

**例5：作业数组**

主要用于大量运行同一作业，而作业之间只有一些细微的差别。下面是一个Job Arrays作 业的脚本。脚本中的三个名为job\_arrays\_0.py、job\_arrays\_1.py、job\_arrays\_2.py 文件是被 用来输入的三个独立的python作业。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=array-job

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=1

#SBATCH --mem-per-cpu=4G

#SBATCH --time=00:01:00

#SBATCH --array=0-2

#SBATCH --output=slurm-%A.%a.out

#SBATCH --error=slurm-%A.%a.err

echo "My SLURM\_ARRAY\_JOB\_ID is $SLURM\_ARRAY\_JOB\_ID."

echo "My SLURM\_ARRAY\_TASK\_ID is $SLURM\_ARRAY\_TASK\_ID"

echo "Executing on the machine:" $(hostname)

module purge

module load CentOS/7.9/Anaconda3/2023.03.01

conda activate env\_name\_xxx

python job\_arrays\_${SLURM\_ARRAY\_TASK\_ID}.py

#SBATCH --array=0-2 这一行指定了数组的大小（3）和数组的索引（0，1，和2） SLURM\_ARRAY\_TASK\_ID环境变量引用--array=0-2 产生的数组索引值，并把该值作为唯 一的输出文件名分配给数组中的每个作业。

**例6：将多个任务作为单个作业并行处理**

假设我们有三个程序，我们想把它们作为一个单一的Slurm作业并行处理，我们可以考虑 使用下面的脚本。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=your\_job

#SBATCH --nodes=1

#SBATCH --ntasks=3

#SBATCH --cpus-per-task=1

#SBATCH --mem-per-cpu=4G

#SBATCH --time=00:01:00

#SBATCH --output=%j.out

#SBATCH --error=%j.err

module purge

module load CentOS/7.9/Anaconda3/2023.03.01

srun -N 1 -n 1 --exclusive python python1.py 0 &

srun -N 1 -n 1 --exclusive python python2.py 1 &

srun -N 1 -n 1 --exclusive python python3.py 2 &

wait

**例7：按顺序运行作业（作业依赖关系）**

下面是如何使用Slurm作业依赖性来自动运行顺序作业的脚本。

#!/bin/bash

#SBATCH --partition=rtg-1700703611526-86f7d669b230

#SBATCH --job-name=long\_job

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=1

#SBATCH --mem=4G

#SBATCH --time=00:01:00

#SBATCH --dependency=singleton

#SBATCH --output=%j.out

#SBATCH --error=%j.err

module purge

module load CentOS/7.9/Anaconda3/2023.03.01

conda activate env\_name\_xxx

python your\_script.py

连续顺序运行5次上面的脚本程序：

$ sbatch sequence\_job.slurm # step 1

$ sbatch sequence\_job.slurm # step 2

$ sbatch sequence\_job.slurm # step 3

$ sbatch sequence\_job.slurm # step 4

$ sbatch sequence\_job.slurm # step 5

#SBATCH --dependency=singleton 这一行指定了作业依赖关系，singleton 参数工作方式是 将具有相同作业名称的后续作业排队，直到具有该名称的前一个作业完成。更多的依赖关 系参数请参考：https://slurm.schedmd.com/sbatch.html

上述作业第一步立即执行，在第1步完成之前，第2步不能开始运行，以此类推。也就是 说，第一个步骤之后的每一步都要等待前面的步骤。

**Scancel： 删除作业**

|  |  |
| --- | --- |
| **Slurm命令** | **功能** |
| scancel -n<job\_name> | 取消指定作业名的作业 |
| scancel – states=<job\_states> | 取消指定状态的作业 |
| scancel -u <user\_name> | 取消指定用户的作业 |
| scancel --help | 使用帮助信息 |

例1：取消作业ID为3688的作业

scancel 3688

例2：取消t5320264用户账户下的所有作业

scancel -u t5320264

**Slurm 环境变量**

|  |  |
| --- | --- |
| **Slurm 命令** | **功能** |
| SLURM\_ JOBID | 作业ID号 |
| SLURM\_NTASKS | 总进程数 |
| SLURM\_JOB\_PARTITION | 分区的名称 |
| SLURM\_SUBMIT\_DIR | 工作目录 |
| SLURM\_SUBMIT\_HOST | 提交作业的主机名 |
| SLURM\_JOB\_NODELIST | 作业分配的节点列表 |
| SLURM\_ARRAY\_TASK\_ID | 作业序列中的作业ID |
| SLURM\_JOB\_NTASKS\_PER\_NODE | 每节点请求的任务数 |
| SLURM\_JOB\_NUM\_NODES | 作业分配的节点数 |

**Modules**

Modules 是在一个在高性能计算平台环境中广泛使用的软件环境变量管理工具。它可以实现高性能计算环境中软件和库之间复杂依赖关系的智能化管理。平台预安装了一些常用软件，用户可以直接使用module命令加载使用。

|  |  |
| --- | --- |
| **Module命令** | **功能** |
| module avail/ml av | 列出集群可用的模块 |
| module load<module\_name> | 加载模块 |
| module list | 列出所有已加载的模块 |
| module unload<module\_name> | 卸载模块 |
| module purge | 卸载所有已加载的模块 |
| module help | 显示帮助信息 |

**module avail/ml av:查看集群可用模块**



**module load \*：加载模块**

例如：module load CentOS/7.9/Anaconda3/2023.03.01：加载Anaconda3模块

$ module load CentOS/7.9/Anaconda3/2023.03.01

$ conda activate <your conda env>

$ python --version

Python 3.10.9

**module list：查询所有已加载的模块情况**



**module unload \*：卸载模块**

例如：module unload CentOS/7.9/Anaconda3/2023.03.01：卸载Anaconda3模块

$ module unload CentOS/7.9/Anaconda3/2023.03.01

$ python --version

Python 2.7.5

**module purge:卸载所有已加载的模块**

$ module purge

$ module list

No Modulefiles Currently Loaded.