Slurm作业管理系统使用

李会民
hmli@ustc.edu.cn

中国科学技术大学 超级计算中心

2023-05-30

过来人提示

- 学会基本Linux用法，别只会用鼠标键盘，学会命令脚本后会发现更加省事：
  - 降低重复性处理
  - 自动监控提交作业，抢占资源
- 学会问问题：提问的智慧：https://lug.ustc.edu.cn/wiki/doc/smart-questions/
- 别只会问问题，要勇于自己解决：看手册、搜索、尝试
- 学会看手册
- 尽信书则不如无书，有疑问，建议看官方原始资料，即使原始资料也有可能有问题
- 培训，主要是有个概念，枯燥乏味，不试试不用很快就忘了

查看命令手册：man

man 可以格式化并显示某命令的联机帮助手册，输出的手册页主要包括以下几个部分：

- NAME：命令的名称和用法
- SYNOPSIS：显示命令的语法格式，列出其所有可用的选用的选项及参数
  - *[ ]*内的可选，即可以没有
  - 不在*[ ]*内的为必选
  - |：或，选项互斥
  - 如：*who [OPTION] ... [FILE | ARG1 ARG2]*
- DESCRIPTION：描述命令的详细用法及每个选项的功能
- OPTION：对命令的每一个选项进行详细的说明

man主要选项

- [section]：显示某[section]内的该命令帮助，[section]是man的分类，一般为：
  - 1：命令
  - 1P：Posix命令
  - 2：系统调用
  - 3：库函数
  - 3P：Posix程序
  - 4：特殊文件
  - 5：文件格式与约定
  - 6：游戏
  - 7：约定与杂项
  - 8：系统管理命令
  - 9：内核进程
- -a：显示所有[section]内的帮助
- -L：以某种语言显示帮助，如：*man -L zh_CN.UTF-8 man*
- 环境变量MANPATH设置*man*的搜索路径，如：*man printf*和*man 3 printf*

阅读info帮助：info/pinfo

info 和 *pinfo* 用于阅读info文档，此种文档相比 *man* 文档更友好，*pinfo* 比 *info* 操作更方便，支持颜色加亮等，一般格式为：

- *info [OPTION]... [MENU-ITEM...]*
- *pinfo [options] [infopage]*

将光标移到链接处（*pinfo*支持彩色链接，*info*看不出来），按回车进入子章节，按小键盘的左右箭头可以来回跳转

显示命令简要说明：–help | -h等

许多命令支持运行时添加如下选项查看简要帮助：

- -h
- -help（注意：这里是一个-，显示为-）
- –help（注意：这里是两-，显示为–）
- -H
- -?

提问的智慧

- 提问前请先看《提问的智慧》：https://lug.ustc.edu.cn/wiki/doc/smart-questions
- 请先查看超算中心主页http://scc.ustc.edu.cn：用户申请、系统介绍、运行监控、资料手册、常见问题、VPN等常用信息
- 请注意系统登录后的提示
- 提问时务必准确，提供相应信息，如账号、系统、作业号、作业目录、具体现象等，要有礼貌
  - 错误：为啥我的作业运行不了？
  - 正确：您好，我在X系统上的Y作业号的作业，目录是XXX，运行时报XXX错误，我自己尝试解决不了，请问能指点下么？

Slurm作业管理系统简介

- Slurm(Simple Linux Utility for Resource Management)是开源的、具有容错性和高度可扩展大型和小型Linux集群资源管理和作业调度系统
- 超算系统可利用Slurm进行资源和作业管理，可设定作业间相互依赖关系，并可避免相互干扰，提高运行效率
  - 所有需运行的作业无论是用于程序调试还是业务计算均必须通过交互式并行*srun*、批处理式*sbatch*或分配式*salloc*等命令提交
  - 提交后可以利用相关命令查询作业状态等
  - 请不要在登录节点直接运行作业（编译除外），以免影响其余用户的正常使用
- 主页：http://slurm.schedmd.com/

三种模式区别

- 批处理作业（采用*sbatch*命令提交）：
  使用*sbatch*命令提交作业脚本，作业被调度运行后，在所分配的首个节点上执行作业脚本，在作业脚本中也可使用*srun*命令加载作业任务
- 交互式作业提交（采用*srun*命令提交）：
  资源分配与任务加载两步均通过*srun*命令进行：当在登录shell中执行*srun*命令时，*srun*首先向系统提交作业请求并等待资源分配，然后在所分配的节点上加载作业任务
- 分配模式作业（采用*salloc*命令提交）：
  类似于交互式作业模式（不释放当前shell）和批处理作业模式的融合。用户需指定资源分配的需求条件，向资源管理器提出作业的资源分配请求。作业排队，当用户请求资源被满足时，将在用户提交作业的节点上执行用户所指定的命令，指定的命令执行结束后，用户申请的资源被释放，才释放。
  *salloc*后面如果没有跟定相应的脚本或可执行文件，则默认选择*/bin/sh*，用户获得了一个合适环境变量的shell环境
  *salloc*和*sbatch*最主要的区别：
  - *salloc*：当资源请求被满足时，直接在提交作业的节点执行相应任务，适合需要指定运行节点和其它资源限制，并有特定命令的作业
  - *sbatch*：当资源请求被满足时，在分配的第一个节点上执行相应任务

基本用户命令

- *sacct*：显示激活的或已完成作业或作业步的记账信息
- *salloc*：为需实时处理的作业分配资源，典型场景为分配资源并启动一个shell，然后用此shell执行*srun*命令去执行并行任务
- *sattach*：吸附到运行中的作业步的标准输入、输出及出错，通过吸附，使得有能力监控运行中的作业步的IO等
- *sbatch*：提交作业脚本使其运行。此脚本一般会含有一个或多个*srun*命令启动并行任务
- *sbcast*：将本地存储中的文件传递分配给作业的节点上
- *scancel*：取消排队或运行中的作业或作业步，还可用于发送任意信号到运行中的作业或作业步中的所有进程
- *scontrol*：显示或设定Slurm作业、队列、节点等状态
- *sinfo*：显示队列或节点状态，具有非常多过滤、排序和格式化等选项
- *squeue*：显示队列中的作业及作业步状态，含非常多过滤、排序和格式化等选项
- *srun*：运行并行作业。具有非常多过滤、排序和格式化等选项，包含最小和最大节点数、处理器数、是否指定使用或排除的节点、节点特征（内存、磁盘空间、特定需求特征等）。一个作业可含有多个作业步，可以是串行的、独立或共享资源并行的

基本术语

- socket：CPU插槽，可以简单理解为CPU
- core：CPU核，单颗CPU可以具有多颗CPU核
- job：作业
- job step：作业步，单个作业（job）可以有多个作业步
- tasks：任务数，单个作业或作业步可有多个任务，一般一个任务需一个CPU核，可理解为所需的CPU核数
- rank：秩，如MPI进程号
- partition：队列、分区。作业需在特定队列中运行，一般不同队列允许的资源不一样，比如单作业核数等
- stdin：标准输入文件，一般指可以通过屏幕输入或采用<文件名方式传递给程序的文件，对应C程序中的文件描述符0
- stdout：标准输出文件，程序运行正常时输出信息到的文件，一般指输出到屏幕的，并可采用>文件名定向到特定文件，对应C程序中的文件描述符1
- stderr：标准出错文件，程序运行出错时输出信息到的文件，一般指也输出到屏幕，并可采用2>文件名定向到特定文件（注意这里的2），对应C程序中的文件描述符2

显示队列、节点信息：sinfo

*sinfo*可以查看系统存在什么队列、节点及其状态。如*sinfo -l*：

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE	ROOT	OVERSUBS	GROUPS	NODES	STATE	NODELIST
CPU-Large*	up	infinite	1-infinite	no	NO	all	720	idle	cnode[001-720]
GPU-V100	up	infinite	1-infinite	no	NO	all	10	idle	gnode[01-10]
2TB-AEP-Mem	up	infinite	1-infinite	no	NO	all	8	mixed	anode[01-08]
ARM-CPU	up	infinite	1-infinite	no	NO	all	2	down*	rnode[01,09]
ARM-CPU	up	infinite	1-infinite	no	NO	all	2	allocated	rnode[02-03]
ARM-CPU	up	infinite	1-infinite	no	NO	all	5	idle	rnode[04-08]

注：系统队列根据需要会调整，请根据上述命令确定可用队列。

sinfo主要输出项 I

- AVAIL：up表示可用，down表示不可用
- CPUS：各节点上的CPU数
- S:C:T：各节点上的CPU插口sockets(S)数（CPU颗数，一颗CPU含有多颗CPU核，以下类似）、CPU核cores(C)数和线程threads(T)数
- SOCKETS：各节点CPU插口数，CPU颗数
- CORES：各节点CPU核数
- THREADS：各节点线程数
- GROUPS：可使用的用户组，all表示所有组都可以用
- JOB_SIZE：可供用户作业使用的最小和最大节点数，如果只有1个值，则表示最大和最小一样，infinite表示无限制
- TIMELIMIT：作业运行墙上时间（walltime，指的是用计时器，如手表或挂钟，度量的实际时间）限制，infinite表示没限制，如有限制，格式为 “days-hours:minutes:seconds”
- MEMORY：实际内存大小，单位为MB

sinfo主要输出项 II

- NODELIST：节点名列表
- NODES：节点数
- NODES(A/I)：节点数，状态格式为“available/idle”
- NODES(A/I/O/T)：节点数，状态格式为“available/idle/other/total”
- PARTITION：队列名，后面带有*的，表示此队列为默认队列
- ROOT：是否限制资源只能分配给root账户
- OVERSUBSCRIBE：是否允许作业分配的资源超过计算资源（如CPU数）：
  - no：不允许超额
  - exclusive：排他的，只能给这些作业用（等价于*srun --exclusive*）
  - force：资源总被超额
  - yes：资源可以被超额
- TMP_DISK：/tmp所在分区空间大小，单位为MB
- STATE：节点状态，可能的状态包括：

sinfo主要输出项 III

- allocated、alloc：已分配，作业结束前不能分配给其他作业
- completing、comp：作业完成中，即将释放资源
- down：宕机，不可用
- drained、drain：已失去活力
- draining、drng：失去活力中
- fail：失效
- failing、failg：失效中
- future、futr：将来可用
- idle：空闲，可以接收新作业
- maint：保持，处于预留状态，具有标识符“maintenance”
- mixed：混合，节点在运行作业，但有些空闲CPU核，可接受新作业
- perfctrs、npc：因网络性能计数器使用中导致无法使用
- power_down、pow_dn：已关机
- power_up、pow_up：正在开机中
- reserved、resv：预留
- unknown、unk：未知原因

sinfo主要输出项 IV

注意，以上状态该可以带有后缀:

- *：节点没响应，将不接受新作业。如仍旧处于没响应，将会变为DOWN状态（COMPLETING、DRAINED、DRAINING、FAIL、FAILING节点除外）
- ：节点目前关机
- #：节点目前将开机或被配置
- !：节点将关机
- %：节点目前正在关机
- $：节点处于预留状态，具有标识符“maintenance”
- @：节点重启中
- ^：节点重启指令已发出
- -：节点计划用于更高级别的作业

sinfo主要参数 I

- -a、–all：显示全部队列信息，如显示隐藏队列或本组没有使用权的队列
- -d、–dead：仅显示无响应或已宕机节点
- -e、–exact：精确而不是分组显示显示各节点
- –help：显示帮助
- -i <seconds>、–iterate=<seconds>：以<seconds>秒间隔持续自动更新显示信息
- -l、–long：显示详细信息
- -n <nodes>、–nodes=<nodes>：显示<nodes>节点信息
- -N, –Node：以每行一个节点方式显示信息，即显示各节点信息
- -p <partition>、–partition=<partition>：显示<partition>队列信息
- -r、–responding：仅显示响应的节点信息
- -R、–list-reasons：显示不响应（down、drained、fail或failing状态）节点的原因
- -s：显示摘要信息

sinfo主要参数 II

- -S <sort_list>、–sort=<sort_list>：设定显示信息的排序方式。排序字段参见后面输出格式部分，多个排序字段采用,分隔，字段前面的+和-分表表示升序（默认）或降序。队列字段P前面如有#，表示以Slurm配置文件slurm.conf中的顺序显示。例如：*sinfo -S +P,-m*表示以队列名升序及内存大小降序排序
- -t <states>、–states=<states>：仅显示<states>状态的信息。<states>状态可以为（不区分大小写）：ALLOC、ALLOCATED、COMP、COMPLETING、DOWN、DRAIN、DRAINED、DRAINING、ERR、ERROR、FAIL、FUTURE、FUTR、IDLE、MAINT、MIX、MIXED、NO_RESPOND、NPC、PERFCTRS、POWER_DOWN、POWER_UP、RESV、RESERVED、UNK和UNKNOWN
- -T, –reservation：仅显示预留资源信息
- –usage：显示用法
- -v、–verbose：显示冗余信息，即详细信息
- -V：显示版本信息

sinfo主要参数 III

- -o <output_format>、–format=<output_format>：按照<output_format>格式输出信息，默认为“%#P %.5a %.10l %.6D %.6t %N”：
  - %all：所有字段信息
  - %a：队列的状态及是否可用
  - %A：以“allocated/idle”格式显示状态对应的节点数
  - %b：激活的特性，参见%f
  - %B：队列中每个节点可分配给作业的CPU数
  - %c：各节点CPU数
  - %C：以“allocated/idle/other/total”格式状态显示CPU数
  - %d：各节点临时磁盘空间大小，单位为MB
  - %D：节点数
  - %e：节点空闲内存，单位为MB
  - %E：节点无效的原因（down、draine或ddraining状态）
  - %f：节点可用特性，参见%b
  - %F：以“allocated/idle/other/total”格式状态的节点数

sinfo主要参数 IV

- %g：可以使用此节点的用户组
- %G：与节点关联的通用资源（gres）
- %h：作业是否能超用计算资源（如CPUs），显示结果可以为yes、no、exclusive或force
- %H：节点不可用信息的时间戳
- %I：队列作业权重因子
- %l：以“days-hours:minutes:seconds”格式显示作业可最长运行时间
- %L：以“days-hours:minutes:seconds”格式显示作业默认时间
- %m：节点内存，单位MB
- %M：抢占模式，可以为no或yes
- %n：节点主机名
- %N：节点名
- %o：节点IP地址
- %O：节点负载
- %p：队列调度优先级
- %P：队列名，带有*为默认队列，参见%R
- %R：队列名，不在默认队列后附加*，参见%P

sinfo主要参数 V

- %s：节点最大作业大小
- %S：允许分配的节点数
- %t：以紧凑格式显示节点状态
- %T：以扩展格式显示节点状态
- %v：slurmd守护进程版本
- %w：节点调度权重
- %X：单节点socket数
- %Y：单节点CPU核数
- %Z：单核进程数
- %z：扩展方式显示单节点处理器信息：sockets、cores、threads（S:C:T）数

- -O <output_format>, –Format=<output_format>：按照<output_format>格式输出信息，类似-o <output_format>、–format=<output_format>
  每个字段的格式为“type[:[.]size]”：
  - size：最小字段大小，如没指明，则最大为20个字符
  - .：指明为右对齐，默认为左对齐

sinfo主要参数 VI

- 可用type：
  - all：所有字段信息
  - allocmem：节点上分配的内存总数，单位MB
  - allocnodes：允许分配的节点
  - available：队列的State/availability状态
  - cpus：各节点CPU数
  - cpusload：节点负载
  - freemem：节点可用内存，单位MB
  - cpusstate：以“allocated/idle/other/total”格式状态的CPU数
  - cores：单CPU颗CPU核数
  - disk：各节点临时磁盘空间大小，单位为MB
  - features：节点可用特性，参见features_act
  - features_act：激活的特性，参见features
  - groups：可以使用此节点的用户组
  - gres：与节点关联的通用资源（gres）
  - maxcpuspernode：队列中各节点最大可用CPU数
  - memory：节点内存，单位MB

sinfo主要参数 VII

- nodeai：以“allocated/idle”格式显示状态对应的节点数
- nodes：节点数
- nodeaiot：以“allocated/idle/other/total”格式状态的节点数
- nodehost：节点主机名
- nodelist：节点名
- oversubscribe：作业是否能超用计算资源（如CPUs），显示结果可以为yes、no、exclusive或force
- partition：队列名，带有*为默认队列，参见%R
- partitionname：队列名，默认队列不附加*，参见%P
- preemptmode：抢占模式，可以为no或yes
- priorityjobfactor：队列作业权重因子
- prioritytier或priority：队列调度优先级
- reason：节点无效的原因（down、draine或ddraining状态）
- size：节点最大作业数
- statecompact：紧凑格式节点状态
- statelong：扩展格式节点状态
- sockets：各节点CPU颗数
- socketcorethread：扩展方式显示单节点处理器信息：sockets、cores、threads（S:C:T）数

sinfo主要参数 VIII

- time：以“days-hours:minutes:seconds”格式显示作业可最长运行时间
- timestamp：节点不可用信息的时间戳
- threads：CPU核线程数
- weight：节点调度权重
- version：slurmd守护进程版本

查看节点负载：sload

监测节点负载命令*sload*（类似LSF的lsload），基本用法*sload [nodelist]*。
注：该*sload*命令是本人写的，不是slurm官方命令，在其它系统上不一定有，可以自己添加

```
#!/bin/sh
if␣[␣$#␣-lt␣1␣];␣then
⟶N=''
else
⟶N="-n $1"
fi
HEADER="HOSTNAMES CPUS CPU_LOAD FREE_MEM(GB)\tSTATE\t\tREASON\tTIMESTAMP"
echo␣-e␣$HEADER
sinfo␣-h␣-o␣"%n %c %O %e %T %E %H"␣$N␣|␣sort␣|␣awk␣'{printf("%8s %4i %7.1f %9.1f %12s %15s %15s\n",$1,$2,$3,$4/1000.0,$5,$6,$7)}'
echo␣-e␣$HEADER
if␣[␣$#␣-lt␣1␣];␣then
⟶echo␣"Usage: sload [nodelist]"
fi
```

注：

- →：表示tab缩进
- ␣：表示空格
- 下载：http://hmli.ustc.edu.cn/doc/training/slurm/sload

查看队列中的作业信息：squeue

显示队列中的作业信息。如*squeue*显示：

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
75	ARM-CPU	arm_job	hmli	R	2:27	2	rnode[02-03]
76	GPU-V100	gpu.slurm	hmli	PD	0:00	5	(Resources)

squeue主要输出项 I

- JOBID：作业号
- PARTITION：队列名（分区名）
- NAME：作业名
- USER：用户名
- ST：状态
  - PD：排队中，PENDING
  - R：运行中，RUNNING
  - CA：已取消，CANCELLED
  - CF：配置中，CONFIGURING
  - CG：完成中，COMPLETING
  - CD：已完成，COMPLETED
  - F：已失败，FAILED
  - TO：超时，TIMEOUT
  - NF：节点失效，NODE FAILURE
  - SE：特殊退出状态，SPECIAL EXIT STATE

squeue主要输出项 II

- TIME：已运行时间
- NODELIST(REASON)：分配给的节点名列表（原因）：
  - AssociationJobLimit：作业达到其最大允许的作业数限制
  - AssociationResourceLimit：作业达到其最大允许的资源限制
  - AssociationTimeLimit：作业达到时间限制
  - BadConstraints：作业含有无法满足的约束
  - BeginTime：作业最早开始时间尚未达到
  - Cleaning：作业被重新排入队列，并且仍旧在执行之前运行的清理工作
  - Dependency：作业等待一个依赖的作业结束
  - FrontEndDown：没有前端节点可用于执行此作业
  - InactiveLimit：作业达到系统非激活限制
  - InvalidAccount：作业用户无效
  - InvalidQOS：作业QOS无效
  - JobHeldAdmin：作业被系统管理员挂起
  - JobHeldUser：作业被用户自己挂起

squeue主要输出项 III

- JobLaunchFailure：作业无法被启动，有可能因为文件系统故障、无效程序名等
- Licenses：作业等待相应的授权
- NodeDown：作业所需的节点宕机
- NonZeroExitCode：作业停止时退出代码非零
- PartitionDown：作业所需的队列出于DOWN状态
- PartitionInactive：作业所需的队列处于Inactive状态
- PartitionNodeLimit：作业所需的节点超过所用队列当前限制
- PartitionTimeLimit：作业所需的队列达到时间限制
- Priority：作业所需的队列存在高等级作业或预留
- Prolog：作业的PrologSlurmctld前处理程序仍旧在运行
- QOSJobLimit：作业的QOS达到其最大作业数限制
- QOSResourceLimit：作业的QOS达到其最大资源限制
- QOSTimeLimit：作业的QOS达到其时间限制
- ReqNodeNotAvail：作业所需的节点无效，如节点宕机
- Reservation：作业等待其预留的资源可用
- Resources：作业等待其所需的资源可用

squeue主要输出项 IV

- SystemFailure：Slurm系统失效，如文件系统、网络失效等
- TimeLimit：作业超过去时间限制
- QOSUsageThreshold：所需的QOS阈值被违反
- WaitingForScheduling：等待被调度中

squeue主要参数 I

- -A <account_list>, –account=<account_list>：显示用户<account_list>的作业信息，用户以,分隔
- -a, –all：显示所有队列中的作业及作业步信息（含被配置为对用户组隐藏队列的信息）
- -r, –array：以每行一个作业元素方式显示）
- -h, –noheader：不显示头信息，即不显示第一行“PARTITION AVAIL TIMELIMIT NODES STATE NODELIST”
- –help：显示帮助信息
- –hide：不显示隐藏队列中的作业和作业步信息。此为默认行为，不显示配置为对用户组隐藏队列的信息
- -i <seconds>, –iterate=<seconds>：以间隔<seconds>秒方式循环显示信息
- -j <job_id_list>, –jobs=<job_id_list>：显示作业号<job_id_list>的作业，作业号以,分隔。–jobs=<job_id_list>可与–steps选项结合显示特定作业的步信息。作业号格式为“job_id[_array_id]”，默认64字节，可用环境变量SLURM_BITSTR_LEN设定更大字段

squeue主要参数 II

- -l, –long：显示更多的作业信息
- -L, –licenses=<license_list>：指定使用授权文件<license_list>，以,分隔
- -n, –name=<name_list>：显示具有特定<name_list>名字的作业，以,分隔
- –noconvert：不对原始单位做转换，如2048M不转换为2G
- -p <part_list>, –partition=<part_list>：显示特定队列<part_list>信息，<part_list>以,分隔
- -P, –priority：对于提交到多个队列的作业，按照各队列显示其信息。如果作业要按照优先级排序时，需考虑队列和作业优先级
- -q <qos_list>, –qos=<qos_list>：显示特定qos的作业和作业步，<qos_list>以,分隔
- -R, –reservation=reservation_name：显示特定预留信息作业
- -s, –steps：显示特定作业步。作业步格式为“job_id[_array_id].step_id”
- -S <sort_list>, –sort=<sort_list>：按照显示特定字段排序显示，<sort_list>以,分隔。如-S P,U

squeue主要参数 III

- –start：显示排队中的作业的预期执行时间
- -t <state_list>, –states=<state_list>：显示特定状态<state_list>的作业信息。<state_list>以,分隔，可为：PENDING(PD)、RUNNING(R)、SUSPENDED(S)、STOPPED(ST)、COMPLETING(CG)、COMPLETED(CD)、CONFIGURING(CF)、CANCELLED(CA)、FAILED(F)、TIMEOUT(TO)、PREEMPTED(PR)、BOOT_FAIL(BF)、NODE_FAIL(NF)和SPECIAL_EXIT(SE)，注意是不区分大小写的，如“pd”和“PD”是等效的
- -u <user_list>, –user=<user_list>：显示特定用户<user_list>的作业信息，<user_list>以,分隔
- –usage：显示帮助信息
- -v, –verbose：显示squeue命令详细动作信息
- -V, –version：显示版本信息
- -w <hostlist>, –nodelist=<hostlist>：显示特定节点<hostlist>信息，<hostlist>以,分隔

squeue主要参数 IV

- -o <output_format>, –format=<output_format>：以特定格式<output_format>显示信息。参见 -O <output_format>, –Format=<output_format>，采用不同参数的默认格式为：
  - default：“%.18i %.9P %.8j %.8u %.2t %.10M %.6D %R”
  - -l, –long：“%.18i %.9P %.8j %.8u %.8T %.10M %.9l %.6D %R”
  - -s, –steps：“%.15i %.8j %.9P %.8u %.9M %N”

  每个字段的格式为“%[[.]size]type”：
  - size：字段最小尺寸，如果没有指定size，则按照所需长度显示
  - .：右对齐显示，默认为左对齐
  - type：类型，一些类型仅对作业有效，而有些仅对作业步有效，可为：
    - %all：显示所有字段
    - %a：显示记帐信息（仅对作业有效）
    - %A：作业步生成的任务数（仅适用于作业步）
    - %A：作业号（仅适用于作业）
    - %b：作业或作业步所需的通用资源（gres）
    - %B：执行作业的节点
    - %c：作业每个节点所需的最小CPU数（仅适用于作业）

squeue主要参数 V

- %C：如果作业还在运行，显示作业所需的CPU数；如果作业正在完成，显示当前分配给此作业的CPU数（仅适用于作业）
- %d：作业所需的最小临时磁盘空间，单位MB（仅适用于作业）
- %D：作业所需的节点（仅适用于作业）
- %e：作业结束或预期结束时间（基于其时间限制）（仅适用于作业）
- %E：作业依赖剩余情况。作业只有依赖的作业完成才运行，如显示NULL，则无依赖（仅适用于作业）
- %f：作业所需的特性（仅适用于作业）
- %F：作业组作业号（仅适用于作业）
- %g：作业用户组（仅适用于作业）
- %G：作业用户组ID（仅适用于作业）
- %h：分配给此作业的计算资源能否被其它作业预约（仅适用于作业）。可被预约的资源包含节点、CPU颗、CPU核或超线程。值可以为：
- %H：作业所需的单节点CPU数，显示srun –sockets-per-node提交选项，如–sockets-per-node未设定，则显示*（仅适用于作业）
- %i：作业或作业步号，在作业组中，作业号格式为“<base_job_id>_<index>”，默认作业组索引字段限制到64字节，可以用环境变量*SLURM_BITSTR_LEN*设定为更大的字段大小

squeue主要参数 VI

- %I：作业所需的每颗CPU的CPU核数，显示的是srun –cores-per-socket设定的值，如–cores-per-socket未设定，则显示*（仅适用于作业）
- %j：作业或作业步名
- %J：作业所需的每个CPU核的线程数，显示的是srun –threads-per-core设定的值，如–threads-per-core未被设置则显示*（仅适用于作业）
- %k：作业说明（仅适用于作业）
- %K：作业组索引默认作业组索引字段限制到64字节，可以用环境变量*SLURM_BITSTR_LEN*设定为更大的字段大小（仅适用于作业）
- %l：作业或作业步时间限制，格式为“days-hours:minutes:seconds”：NOT_SET表示没有建立；UNLIMITED表示没有限制
- %L：作业剩余时间，格式为“days-hours:minutes:seconds”，此值由作业的时间限制减去已用时间得到：NOT_SET表示没有建立；UNLIMITED表示没有限制（仅适用于作业）
- %m：作业所需的最小内存，单位为MB（仅适用于作业）
- %M：作业或作业步已经使用的时间，格式为“days-hours:minutes:seconds”
- %n：作业所需的节点名（仅适用于作业）
- %N：作业或作业步分配的节点名，对于正在完成的作业，仅显示尚未释放资源回归服务的节点

squeue主要参数 VII

- %o：执行的命令
- %O：作业是否需连续节点（仅适用于作业）
- %p：作业的优先级（0.0到1.0之间），参见%Q（仅适用于作业）
- %P：作业或作业步的队列
- %q：作业关联服务的品质（仅适用于作业）
- %Q：作业优先级（通常为非常大的一个无符号整数），参见%p（仅适用于作业）
- %r：作业在当前状态的原因，参见JOB REASON CODES（仅适用于作业）
- %R：参见JOB REASON CODES（仅适用于作业）：
- %S：作业或作业步实际或预期的开始时间
- %t：作业状态，以紧凑格式显示：PD（排队pending）、R（运行running）、CA（取消cancelled）、CF(配置中configuring）、CG（完成中completing）、CD（已完成completed）、F（失败failed）、TO（超时timeout）、NF（节点失效node failure)和SE（特殊退出状态special exit state），参见JOB STATE CODES（仅适用于作业）
- %T：作业状态，以扩展格式显示：PENDING、RUNNING、SUSPENDED、CANCELLED、COMPLETING、COMPLETED、CONFIGURING、FAILED、TIMEOUT、PREEMPTED、NODE_FAIL和SPECIAL_EXIT，参见JOB STATE CODES（仅适用于作业）
- %u：作业或作业步的用户名

squeue主要参数 VIII

- %U：作业或作业步的用户ID
- %v：作业的预留资源（仅适用于作业）
- %V：作业的提交时间
- %w：负载特征关键Workload Characterization Key（wckey）（仅适用于作业）
- %W：作业预留的授权（仅适用于作业）
- %x：作业排他性节点名（仅适用于作业）
- %X：系统使用需每个节点预留的CPU核数（仅适用于作业）
- %y：Nice值（调整作业调动优先级）（仅适用于作业）
- %Y：对于排队中作业，显示其开始运行时期望的节点名
- %z：作业所需的每个节点的CPU颗数、CPU核数和线程数（S:C:T），如（S:C:T）未设置，则显示*（仅适用于作业）
- %Z：作业的工作目录

- -O <output_format>, –Format=<output_format>：以特定格式<output_format>显示信息，参见-o <output_format>, –format=<output_format> 每个字段的格式为“%[[.]size]type”：
  - size：字段最小尺寸，如果没有指定size，则最长显示20个字符
  - .：右对齐显示，默认为左对齐

squeue主要参数 IX

- type：类型，一些类型仅对作业有效，而有些仅对作业步有效，可为：
  - account：作业记账信息（仅适用于作业）
  - allocnodes：作业分配的节点（仅适用于作业）
  - allocsid：用于提交作业的会话ID（仅适用于作业）
  - arrayjobid：作业组中的作业ID
  - arraytaskid：作业组中的任务ID
  - associd：作业关联ID（仅适用于作业）
  - batchflag：是否批处理设定了标记（仅适用于作业）
  - batchhost：执行节点（仅适用于作业）；
  - chptdir：作业checkpoint的写目录（仅适用于作业步）
  - chptinter：作业checkpoint时间间隔（仅适用于作业步）
  - command：作业执行的命令（仅适用于作业）
  - comment：作业关联的说明（仅适用于作业）
  - contiguous：作业是否要求连续节点（仅适用于作业）
  - cores：作业所需的每颗CPU的CPU核数，显示的是srun –cores-per-socket设定的值，如–cores-per-socket未设定，则显示*（仅适用于作业）
  - corespec：为了系统使用所预留的CPU核数（仅适用于作业）

squeue主要参数 X

- cpufreq：分配的CPU主频（仅适用于作业步）
- cpuspertask：作业分配的每个任务的CPU颗数（仅适用于作业）
- deadline：作业的截止时间（仅适用于作业）
- dependency：作业依赖剩余。作业只有依赖的作业完成才运行，如显示NULL，则无依赖（仅适用于作业）
- derivedec：作业的起源退出码，对任意作业步是最高退出码（仅适用于作业）
- eligibletime：预计作业开始运行时间（仅适用于作业）
- endtime：作业实际或预期的终止时间（仅适用于作业）
- exit_code：作业退出码（仅适用于作业）
- feature：作业所需的特性（仅适用于作业）
- gres：作业或作业步需的通用资源
- groupid：作业用户组ID（仅适用于作业）
- groupname：作业用户组名（仅适用于作业）
- jobarrayid：作业组作业ID（仅适用于作业）
- jobid：作业号（仅适用于作业）
- licenses：作业预留的授权（仅适用于作业）
- maxcpus：分配给作业的最大CPU颗数（仅适用于作业）

squeue主要参数 XI

- maxnodes：分配给作业的最大节点数（仅适用于作业）
- mcslabel：作业的MCS_label（仅适用于作业）
- minmemory：作业所需的最小内存大小，单位MB（仅适用于作业）
- mintime：作业的最小时间限制（仅适用于作业）
- mintmpdisk：作业所需的临时磁盘空间，单位MB（仅适用于作业）
- mincpus：作业所需的各节点最小CPU颗数，显示的是srun –mincpus设定的值（仅适用于作业）
- name：作业或作业步名
- network：作业运行的网络
- nice Nice值(调整作业调度优先值)（仅适用于作业）
- nodes：作业或作业步分配的节点名，对于正在完成的作业，仅显示尚未释放资源回归服务的节点
- nodelist：作业或作业步分配的节点，对于正在完成的作业，仅显示尚未释放资源回归服务的节点
- ntpercore：作业每个CPU核分配的任务数（仅适用于作业）
- ntpernode：作业每个节点分配的任务数（仅适用于作业）
- ntpersocket：作业每颗CPU分配的任务数（仅适用于作业）
- numcpus：作业所需的或分配的CPU颗数

squeue主要参数 XII

- numnodes：作业所需的或分配的最小节点数（仅适用于作业）
- numtask：作业或作业号需的任务数，显示的–ntasks设定的
- oversubscribe：分配给此作业的计算资源能否被其它作业预约（仅适用于作业）。可被预约的资源包含节点、CPU颗、CPU核或超线程。值可以为：
- partition：作业或作业步的队列
- priority：作业的优先级（0.0到1.0之间），参见%Q（仅适用于作业）
- prioritylong：作业优先级（通常为非常大的一个无符号整数），参见%p（仅适用于作业）
- profile：作业特征（仅适用于作业）
- preemptime：作业抢占时间（仅适用于作业）
- qos：作业的服务质量（仅适用于作业）
- reason：作业在当前的原因，参见JOB REASON CODES（仅适用于作业）
- reasonlist：参见JOB REASON CODES（仅适用于作业）
- reqnodes：作业所需的节点名（仅适用于作业）
- requeue：作业失败时是否需重新排队运行（仅适用于作业）
- reservation：预留资源（仅适用于作业）
- resizetime：运行作业的变化时间总和（仅适用于作业）
- restartcnt：作业的重启checkpoint数（仅适用于作业）

squeue主要参数 XIII

- resvport：作业的预留端口（仅适用于作业步）
- schednodes：排队中的作业开始运行时预期将被用的节点列表（仅适用于作业）
- sct：各节点作业所需的CPU数、CPU核数和线程数（S:C:T），如（S:C:T）未设置，则显示*（仅适用于作业）
- selectjobinfo：节点选择插件针对作业指定的数据，可能的数据包含：资源分配的几何维度（X、Y、Z维度）、连接类型（TORUS、MESH或NAV == torus else mesh），是否允许几何旋转（yes或no），节点使用（VIRTUAL或COPROCESSOR）等（仅适用于作业）
- sockets：作业每个节点需的CPU数，显示srun时的–sockets-per-node选项，如–sockets-per-node未设置，则显示*（仅适用于作业）
- sperboard：每个主板分配给作业的CPU数（仅适用于作业）
- starttime：作业或作业布实际或预期开始时间
- state：扩展格式作业状态：排队中PENDING、运行中RUNNING、已停止STOPPED、被挂起SUSPENDED、被取消CANCELLED、完成中COMPLETING、已完成COMPLETED、配置中CONFIGURING、已失败FAILED、超时TIMEOUT、预取PREEMPTED、节点失效NODE_FAIL、特定退出SPECIAL_EXIT，参见JOB STATE CODES部分（仅适用于作业）

squeue主要参数 XIV

- statecompact：紧凑格式作业状态：PD（排队中pending）、R（运行中running）、CA（已取消cancelled）、CF(配置中configuring）、CG（完成中completing）、CD（已完成completed）、F（已失败failed）、TO（超时timeout）、NF（节点失效node failure）和SE（特定退出状态special exit state），参见JOB STATE CODES部分（仅适用于作业）
- stderr：标准出错输出目录（仅适用于作业）
- stdin：标准输入目录（仅适用于作业）
- stdout：标准输出目录（仅适用于作业）
- stepid：作业或作业步号。在作业组中，作业号格式为“<base_job_id>_<index>”（仅适用于作业步）
- stepname：作业步名（仅适用于作业步）
- stepstate：作业步状态（仅适用于作业步）
- submittime：作业提交时间（仅适用于作业）
- threads：作业所需的每颗CPU核的线程数，显示srun的–threads-per-core参数，如–threads-per-core未设置，则显示*（仅适用于作业）
- timeleft：作业剩余时间，格式为“days-hours:minutes:seconds”，此值是通过其时间限制减去已运行时间得出的：如未建立则显示“NOT_SET”；如无限制则显示“UNLIMITED”（仅适用于作业）
- timelimit：作业或作业步的时间限制

squeue主要参数 XV

- timeused：作业或作业步以使用时间，格式为“days-hours:minutes:seconds”，days和hours只有需要时才显示。对于作业步，显示从执行开始经过的时间，因此对于被曾挂起的作业并不准确。节点间的时间差也会导致时间不准确。如时间不对（如，负值），将显示“INVALID”
- tres：显示分配给作业的可被追踪的资源
- userid：作业或作业步的用户ID
- username：作业或作业步的用户名
- wait4switch：需满足转轨器数目的总等待时间（仅适用于作业）
- wckey：工作负荷特征关键（wckey）（仅适用于作业）
- workdir：作业工作目录（仅适用于作业）

查看详细队列信息：scontrol show partition I

- *scontrol show partition*显示全部队列信息（在不引起歧义的情况下，可以不必输全，如可以只输入*scontrol show pa*，其他类似）
- *scontrol show partition PartitionName*或*scontrol show partition=PartitionName* 显示队列名为PartitionName的队列信息
- 输出类似：

```
PartitionName=CPU-Large
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=YES ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=cnode[001-720]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=28800 TotalNodes=720 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=GPU-V100
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
```

查看详细队列信息：scontrol show partition II

```
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=NONE DisableRootJobs=YES ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=gnode[01-10]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=400 TotalNodes=10 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=2TB-AEP-Mem
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=NONE DisableRootJobs=YES ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=anode[01-08]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=320 TotalNodes=8 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

scontrol show partition主要输出项 I

- PartitionName：队列名
- AllowGroups：允许的用户组
- AllowAccounts：允许的用户
- AllowQos：允许的QoS
- AllocNodes：允许的节点
- Default：是否为默认队列
- QoS：服务质量
- DefaultTime：默认时间
- DisableRootJobs：是否禁止root用户提交作业
- ExclusiveUser：排除的用户
- GraceTime：抢占的款显时间，单位秒
- Hidden：是否为隐藏队列

scontrol show partition主要输出项 II

- MaxNodes：最大节点数
- MaxTime：最大运行时间
- MinNodes：最小节点数
- LLN：是否按照最小负载节点调度
- MaxCPUsPerNode：每个节点的最大CPU颗数
- Nodes：节点名
- PriorityJobFactor：作业因子优先级
- PriorityTier：调度优先级
- RootOnly：是否只允许Root
- ReqResv：要求预留的资源
- OverSubscribe：是否允许超用
- PreemptMode：是否为抢占模式

scontrol show partition主要输出项 III

- State：状态：
  - UP：可用，作业可以提交到此队列，并将运行
  - DOWN：作业可以提交到此队列，但作业也许不会获得分配开始运行。已运行的作业还将继续运行
  - DRAIN：不接受新作业，已接受的作业可以被运行
  - INACTIVE：不接受新作业，已接受的作业未开始运行的也不运行
- TotalCPUs：总CPU核数
- TotalNodes：总节点数
- SelectTypeParameters：资源选择类型参数
- DefMemPerNode：每个节点默认分配的内存大小，单位MB
- MaxMemPerNode：每个节点最大内存大小，单位MB

查看详细节点信息：scontrol show node I

- *scontrol show node*显示全部节点信息
- *scontrol show node NODENAME*或*scontrol show node=NODENAME* 显示节点名为NODENAME的节点信息
- 输出类似:

```
NodeName=anode01 Arch=x86_64 CoresPerSocket=20
   CPUAlloc=0 CPUTot=40 CPULoad=0.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=anode01 NodeHostName=anode01 Version=19.05.4
   OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
   RealMemory=2031623 AllocMem=0 FreeMem=1989520 Sockets=2 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=2TB-AEP-Mem
   BootTime=2019-11-09T15:47:56 SlurmdStartTime=2019-12-01T19:01:59
   CfgTRES=cpu=40,mem=2031623M,billing=40
   AllocTRES=
   CapWatts=n/a
```

查看详细节点信息：scontrol show node II

```
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

NodeName=gnode01 Arch=x86_64 CoresPerSocket=20
   CPUAlloc=0 CPUTot=40 CPULoad=0.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=gpu:v100:2
   NodeAddr=gnode01 NodeHostName=gnode01 Version=19.05.4
   OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
   RealMemory=385560 AllocMem=0 FreeMem=368966 Sockets=2 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=GPU-V100
   BootTime=2019-11-13T16:51:31 SlurmdStartTime=2019-12-01T19:54:55
   CfgTRES=cpu=40,mem=385560M,billing=40,gres/gpu=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

scontrol show node主要输出项 I

- NodeName：节点名
- Arch：系统架构
- CoresPerSocket：12
- CPUAlloc：分配给的CPU核数
- CPUErr：出错的CPU核数
- CPUTot：总CPU核数
- CPULoad：CPU负载
- AvailableFeatures：可用特性
- ActiveFeatures：激活的特性
- Gres：通用资源。如上面Gres=gpu:v100:2指明了有两块V100 GPU
- NodeAddr：节点IP地址
- NodeHostName：节点名

scontrol show node主要输出项 II

- Version：Slurm版本
- OS：操作系统
- RealMemory：实际物理内存，单位GB
- AllocMem：已分配内存，单位GB
- FreeMem：可用内存，单位GB
- Sockets：CPU颗数
- Boards：主板数
- State：状态
- ThreadsPerCore：每颗CPU核线程数
- TmpDisk：临时存盘硬盘大小
- Weight：权重
- BootTime：开机时间
- SlurmdStartTime：Slurmd守护进程启动时间

查看详细作业信息：scontrol show job I

- *scontrol show job*显示全部作业信息
- *scontrol show job JOBID*或*scontrol show job=JOBID* 显示作业号为JOBID的作业信息
- 输出类似：

```
JobId=77 JobName=gres_test.bash
   UserId=hmli(10001) GroupId=nic(10001) MCS_label=N/A
   Priority=4294901755 Nice=0 Account=(null) QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:11 TimeLimit=UNLIMITED TimeMin=N/A
   SubmitTime=2019-12-01T20:10:15 EligibleTime=2019-12-01T20:10:15
   AccrueTime=2019-12-01T20:10:15
   StartTime=2019-12-01T20:10:16 EndTime=Unknown Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2019-12-01T20:10:16
   Partition=GPU-V100 AllocNode:Sid=login01:1016
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=gnode01
   BatchHost=gnode01
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,node=1,billing=1
```

查看详细作业信息：scontrol show job II

```
Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/nic/hmli/gres_test.bash
WorkDir=/home/nic/hmli
StdErr=/home/nic/hmli/job-77.err
StdIn=/dev/null
StdOut=/home/nic/hmli/job-77.log
Power=
```

scontrol show job主要输出项 I

- JobId：作业号
- JobName：作业名
- UserId：用户名（用户ID）
- GroupId：用户组（组ID）
- MCS_label：
- Priority：优先级，越大越优先，如果为0则表示被管理员挂起，不允许运行
- Nice：Nice值，越小越优先，-20到19
- Account：记账用户名
- QOS：作业的服务质量
- JobState：作业状态
  - PENDING：排队中
  - RUNNING：运行中

scontrol show job主要输出项 II

- CANCELLED：已取消
- CONFIGURING：配置中
- COMPLETING：完成中
- COMPLETED：已完成
- FAILED：已失败
- TIMEOUT：超时
- NODE FAILURE：节点失效
- SPECIAL EXIT STATE：特殊退出状态

- Reason：原因
- Dependency：依赖关系
- Requeue：节点失效时，是否重排队，0为否，1为是
- Restarts：失败时，是否重运行，0为否，1为是
- BatchFlag：是否为批处理作业，0为否，1为是
- Reboot：节点空闲时是否重启节点，0为否，1为是

scontrol show job主要输出项 III

- ExitCode：作业退出代码
- RunTime：已运行时间
- TimeLimit：作业允许的剩余运行时间
- TimeMin：最小时间
- SubmitTime：提交时间
- EligibleTime：获得认可时间
- StartTime：开始运行时间
- EndTime：预计结束时间
- Deadline：截止时间
- PreemptTime：先占时间
- SuspendTime：挂起时间
- SecsPreSuspend：0

scontrol show job主要输出项 IV

- Partition：对列名
- AllocNode:Sid：分配的节点:系统ID号
- ReqNodeList：去要的节点列表
- ExcNodeList：排除的节点列表
- NodeList：实际运行节点列表
- BatchHost：批处理节点名
- NumNodes：节点数
- NumCPUs：CPU核数
- NumTasks：任务数
- CPUs/Task：CPU核数/任务数
- ReqB:S:C:T：所需的主板数:每主板CPU颗数:每颗CPU核数:每颗CPU核的线程数，:<socket_per_baseboard_count>:<core_per_socket_count>:

scontrol show job主要输出项 V

- TRES：显示分配给作业的可被追踪的资源
- Socks/Node：每节点CPU颗数
- NtasksPerN:B:S:C：每主板数:每主板CPU颗数:每颗CPU的核数:每颗CPU核的线程数启动的作业数，
  <tasks_per_node>:<tasks_per_baseboard>:<tasks_per_socket>:<tasks_per_core>
- CoreSpec：各节点系统预留的CPU核数，如未包含，则显示*
- MinCPUsNode：每节点最小CPU核数
- MinMemoryNode：每节点最小内存大小，0表示未限制
- MinTmpDiskNode：每节点最小临时存盘硬盘大小，0表示未限制
- Features：特性
- Gres：通用资源
- Reservation：预留资源

scontrol show job主要输出项 VI

- OverSubscribe：是否允许与其它作业共享资源，OK允许，NO不允许
- Contiguous：是否要求分配连续节点，OK是，NO否
- Licenses：软件授权
- Network：网络
- Command：作业命令
- WorkDir：工作目录
- StdErr：标准出错输出文件
- StdIn：标准输入文件
- StdOut：标准输出文件

查看作业屏幕输出：speek I

- 监测作业屏幕输出的命令*speek*（类似LSF的bpeek），基本用法*speek [-e] [-f] 作业号*
- 默认显示正常屏幕输出
  - 如添加-f参数，则连续监测输出
  - 如添加-e参数，则监测错误日志
- 注：*speek*命令是本人写的，非slurm官方命令，在其它系统上不一定有，可自己添加

```
#!/bin/bash
#Author: HM Li <hmli@ustc.edu.cn>
if [ $# -lt 1 ]; then
    echo "Usage: speek [-e] [-f] jobid"
    echo "-e: show error log."
    echo "-f: output appended data as the file grows."
fi
NO=1
STD=StdOut
while getopts 'ef' OPT; do
    case $OPT in
        e)
            STD=StdErr
```

查看作业屏幕输出：speek II

```
⟶⟶⟶;;
⟶⟶f)
⟶⟶⟶T='-f'
⟶⟶⟶;;
⟶esac
done
JID=${!#}
F=`scontrol show job $JID 2>/dev/null | awk -v STD=$STD -F='{if($1~"STD") print $2}'`
if [ -f "$F" ]; then
⟶tail $T $F
else
⟶echo "Job $JID has no $STD file."
fi
```

注意:

- `：键盘左上角反引号`，尾巴朝上，在Shell中一对`内的命令的输出将作为值传递给=前的变量，如：*A=`date`*
- '：单引号'，尾巴朝下
- "：双引号"，双尾巴朝下
- 下载：http://hmli.ustc.edu.cn/doc/training/slurm/speek

查看作业负载：jload

- 监测作业负载命令*jload*，基本用法*jload [jobid]*
- 注：该*jload*命令是本人写的，不是slurm官方命令，在其它系统上不一定有，可以自己添加

```
#!/bin/sh
if␣[␣$#␣-lt␣1␣];␣then
⟶echo␣"Usage: jload [jobid]"
⟶echo␣"Current jobs: "
⟶squeue␣-u␣$USER
⟶exit
fi
jobid=$1
HEADER="HOSTNAMES CPUS CPU_LOAD FREE_MEM(GB)\tSTATE\t\tREASON\tTIMESTAMP"
echo␣-e␣$HEADER
sinfo␣-h␣-o␣"%n %c %O %e %T %E %H"␣-n␣'squeue␣-h␣-j␣$jobid␣-o␣"%N"'␣|␣sort␣|␣\
⟶awk␣'{printf("%8s %4i %7.1f %9.1f %12s %15s %15s\n",$1,$2,$3,$4/1000.0,$5,$6,$7)}'
echo␣-e␣$HEADER
```

- 注意：以上 sinfo行末尾有个\，表示下一行是续行
- 下载：http://hmli.ustc.edu.cn/doc/training/slurm/jload

交互式提交并行作业：srun

*srun*可以交互式提交运行并行作业，提交后，作业等待运行，等运行完毕后，才返回终端。
语法为：*srun [OPTIONS...] executable [args...]*

srun主要参数 I

- –begin=<time>：设定作业开始时间，时间到了如果资源满足，则运行，否则继续等待。常用格式如下：
  - –begin=16:00：16:00开始
  - –begin=now+1hour：1小时后开始
  - –begin=now+60：60分钟后开始
  - –begin=2017-02-20T12:34:00：2017-02-20T12:34:00开始
- –comment=<string>：作业说明
- –contiguous：需分配到连续节点，一般来说连续节点之间网络会快一点，如在同一个IB交换机内，但有可能导致开始运行时间推迟（需等待足够多的连续节点）
- –cores-per-socket=<cores>：每颗CPU的核数
- -c, –cpus-per-task=<ncpus>：每个进程需<ncpus>颗CPU核，一般运行OpenMP等多线程程序时需，普通MPI程序不需

srun主要参数 II

- –deadline=<OPT>：截止时间，如果作业到了截至时间而未完成，那么也将结束。时间格式为：
  - HH:MM[:SS] [AM|PM]
  - MMDD[YY] 或 MM/DD[/YY] 或 MM.DD[.YY]
  - MM/DD[/YY]-HH:MM[:SS]
  - YYYY-MM-DD[THH:MM[:SS]]]
- -d, –dependency=<dependency_list>：依赖于<dependency_list>满足后才运行，如：
  - after:job_id[:jobid...]：作业号为job_id[:jobid...]的所有作业结束后
  - afterany:job_id[:jobid...]：作业号为job_id[:jobid...]的任意之一作业结束
  - aftercorr:job_id[:jobid...]：作业号为job_id[:jobid...]的所有作业成功结束（退出码为0）后
  - afternotok:job_id[:jobid...]：作业号为job_id[:jobid...]的所有作业失败结束（如退出状态为：非零退出码、节点失效、超时等）后
  - afterok:job_id[:jobid...]：作业号为job_id[:jobid...]的所有作业成功执行（退出码为0）后
  - expand:job_id：分配给此作业的资源将被扩展到job_id作业。扩充到的作业不许共享相同的QOS和队列。此队列中的连接资源调度不被支持

srun主要参数 III

- singleton：任意先前共享相同作业名和用户的作业结束后
- -D, –chdir=<path>：改变工作目录到<path>执行命令
- -e, –error=<mode>：设定标准错误如何重定向。非交互模式下，默认srun重定向标准错误到与标准输出同样的文件（如果指定了）。此参数可以指定重定向到不同文件。如果指定的文件已经存在，那么将被覆盖。参见IO重定向
- -E, –preserve-env：将环境变量*SLURM_NNODES*和*SLURM_NTASKS*传递给可执行文件，而无需通过计算命令行参数
- –epilog=<executable>：作业结束后执行<executable>程序做相应处理
- –exclusive[=user]：排他性运行，独占性运行，此节点不允许其他用户或不允许user用户共享运行作业
- –export=<environment variables | NONE>：将环境变量传递给应用程序，如“–export=NONE,PATH=/bin,SHELL=/bin/bash”
- -h, –help：显示帮助信息

srun主要参数 IV

- -i, –input=<mode>：指定标准输入如何重定向。默认，srun对所有任务重定向标准输入为从终端。参见IO重定向
- -J, –job-name=<jobname>：赋予作业的作业名为<jobname>
- -l, –label：在标注正常输出或标准错误输出的行前面添加作业号
- –mpi=<mpi_type>：指定使用的MPI环境，<mpi_type>可以主要为：
  - list：列出可用的MPI以便选择
  - pmi2：启用PMI2支持
  - pmix：启用PMIx（https://pmix.github.io）支持
  - none：默认选项，针对其它MPI实现，如Intel MPI
- –multi-prog：让不同任务运行不同的程序及参数，需指定一个配置文件，参见MULTIPLE PROGRAM CONFIGURATION
- -N, –nodes=<minnodes[-maxnodes]>：采用特定节点数运行作业，如没指定maxnodes则需特定节点数，注意，这里是节点数，不是CPU核数，实际分配的是节点数×每节点CPU核数

srun主要参数 V

- -n, –ntasks=<number>：运行<number>个任务，默认一个节点一个作业，注意是节点，不是CPU核。仅对作业起作用，不对作业步起作用
- –ntasks-per-core=<ntasks>：每颗CPU核运行<ntasks>个任务，需与-n, –ntasks=<number>配合，并自动绑定<ntasks>个任务到每个CPU核。仅对作业起作用，不对作业步起作用
- –ntasks-per-node=<ntasks>：每个节点运行<ntasks>个任务，需与-n, –ntasks=<number>配合。仅对作业起作用，不对作业步起作用
- –ntasks-per-socket=<ntasks>：每颗CPU运行<ntasks>个任务，需与-n, –ntasks=<number>配合，并自动绑定<ntasks>个任务到每颗CPU。仅对作业起作用，不对作业步起作用
- -o, –output=<mode>：指定标准输出重定向。在非交互模式中，默认srun收集各任务的标准输出，并发送到吸附的终端上。采用–output可以将其重定向到同一个文件、每个任务一个文件或/dev/null等。参见IO重定向

srun主要参数 VI

- –open-mode=<append|truncate>：标准输出和标准错误输出打开文件的方式：
  - append：追加
  - truncate：截断覆盖
- -p, –partition=<partition_names>：使用<partition_names>队列
- –prolog=<executable>：作业开始运行前执行<executable>程序，做相应处理
- -t, –time=<time>：作业最大运行总时间<time>，到时间后将被终止掉。时间<time>的格式可以为：分钟、分钟:秒、小时:分钟:秒、天-小时、天-小时:分钟、天-小时:分钟:秒
- –task-epilog=<executable>：任务终止后立即执行<executable>，对应于作业步分配
- –task-prolog=<executable>：任务开始前立即执行<executable>，对应于作业步分配
- –threads-per-core=<threads>：每颗CPU核运行<threads>个线程
- –usage：显示简略帮助信息
- -v, –verbose：显示详细信息，多个v会显示更详细的详细

srun主要参数 VII

- -W, –wait=<seconds>：设定在第一个任务结束后多久结束全部任务
- -w, –nodelist=<host1,host2,... or filename>：在特定<host1,host2>节点或filename文件中指定的节点上运行
- -x, –exclude=<host1,host2,... or filename>：在特定<host1,host2>节点或filename文件中指定的节点之外的节点上运行

srun IO重定向 I

- 默认标准输出文件和标准出错文件将从所有任务中被重定向到srun的标准输出文件和标准出错文件
- 标准输入文件从srun的标准输入文件重定向到所有任务
- 如果标准输入仅仅是几个任务需要，建议采用读文件方式而不是重定向方式，以免输入错误数据
- 以上行为可以通过 –output、–error和–input（-o、-e、-i）等选项改变，有效的格式为：
  - all：标准输出和标准出错从所有任务定向到srun，标准输入文件从srun的标准输输入文件重定向到所有任务（默认）
  - none：标准输出和标准出错不从任何任务定向到srun，标准输入文件不从srun定向到任何任务
  - taskid：标准输出和/或标准出错仅从任务号为taskid的任务定向到srun，标准输入文件仅从srun定向到任务号为taskid任务
  - filename: srun将所有任务的标准输出和标准出错重定向到filename文件，标准输入文件将从filename文件重定向到全部任务

srun IO重定向 II

- 格式化字符：srun允许生成采用格式化字符命名的上述IO文件，如可以结合作业号、作业步、节点或任务等
  - \\：不处理任何代替符
  - %%：字符“%”
  - %A：作业组的主作业分配号
  - %a：作业组ID号
  - %J：运行作业的作业号.步号（如128.0）
  - %j：运行作业的作业号
  - %s：运行作业的作业步号
  - %N：短格式节点名，每个节点将生成的不同的IO文件
  - %n：当前作业相关的节点标记（如“0”是运行作业的第一个节点），每个节点将生成的不同的IO文件
  - %t：与当前作业相关的任务标记（rank），每个rank将生成一个不同的IO文件
  - %u：用户名
- 在%与格式化标记符之间的数字可以用于生成前导零，如：
  - job%J.out：job128.0.out
  - job%4j-%2t.out：job0128-00.out、job0128-01.out、...

srun主要输入环境变量 I

一些srun选项可通过环境变量来设置，命令行的选项优先级高于设置的环境变量，将覆盖掉环境变量的设置。环境变量与对应的参数如下：

- *SLURM_ACCOUNT*：类似 -A, –account
- *SLURM_ACCTG_FREQ*：类似 –acctg-freq
- *SLURM_BCAST*：类似 –bcast
- *SLURM_CHECKPOINT*：类似 –checkpoint
- *SLURM_CHECKPOINT_DIR*：类似 –checkpoint-dir
- *SLURM_CNLOAD_IMAGE*：类似 –cnload-image
- *SLURM_COMPRESS*：类似 –compress
- *SLURM_CONN_TYPE*：类似 –conn-type
- *SLURM_CORE_SPEC*：类似 –core-spec
- *SLURM_CPU_BIND*：类似 –cpu-bind

srun主要输入环境变量 II

- *SLURM_CPU_FREQ_REQ*：类似 –cpu-freq.
- *SLURM_CPUS_PER_TASK*：类似 -c, –cpus-per-task
- *SLURM_DEBUG*：类似 -v, –verbose
- *SLURM_DEPENDENCY*：类似 -P, –dependency=<jobid>
- *SLURM_DISABLE_STATUS*：类似 -X, –disable-status
- *SLURM_DIST_PLANESIZE*：类似 -m plane
- *SLURM_DISTRIBUTION*：类似 -m, –distribution
- *SLURM_EPILOG*：类似 –epilog
- *SLURM_EXCLUSIVE*：类似 –exclusive
- *SLURM_EXIT_ERROR*：Slurm出错时的退出码
- *SLURM_EXIT_IMMEDIATE*：当–immediate使用时且资源当前无效时的Slurm退出码
- *SLURM_GEOMETRY*：类似 -g, –geometry

srun主要输入环境变量 III

- *SLURM_GRES_FLAGS*：类似 –gres-flags
- *SLURM_HINT*：类似 –hint
- *SLURM_GRES*：类似 –gres，参见 *SLURM_STEP_GRES*
- *SLURM_IMMEDIATE*：类似 -I, –immediate
- *SLURM_IOLOAD_IMAGE*：类似 –ioload-image
- *SLURM_JOB_ID*：类似 –jobid
- *SLURM_JOB_NAME*：类似 -J, –job-name
- *SLURM_JOB_NUM_NODES*：分配的总节点数
- *SLURM_KILL_BAD_EXIT*：类似 -K, –kill-on-bad-exit
- *SLURM_LABELIO*：类似 -l, –label
- *SLURM_LINUX_IMAGE*：类似 –linux-image
- *SLURM_MEM_BIND*：类似 –mem-bind

srun主要输入环境变量 IV

- *SLURM_MEM_PER_CPU*：类似 –mem-per-cpu
- *SLURM_MEM_PER_NODE*：类似 –mem
- *SLURM_MLOADER_IMAGE*：类似 –mloader-image
- *SLURM_MPI_TYPE*：类似 –mpi
- *SLURM_NETWORK*：类似 –network
- *SLURM_NNODES*：类似 -N, –nodes
- *SLURM_NO_ROTATE*：类似 -R, –no-rotate
- *SLURM_NTASKS*：类似 -n, –ntasks
- *SLURM_NTASKS_PER_CORE*：类似 –ntasks-per-core
- *SLURM_NTASKS_PER_NODE*：类似 –ntasks-per-node
- *SLURM_NTASKS_PER_SOCKET*：类似 –ntasks-per-socket
- *SLURM_OPEN_MODE*：类似 –open-mode

srun主要输入环境变量 V

- *SLURM_OVERCOMMIT*：类似 -O, –overcommit
- *SLURM_PARTITION*：类似 -p, –partition
- *SLURM_POWER*：类似 –power
- *SLURM_PROFILE*：类似 –profile
- *SLURM_PROLOG*：类似 –prolog
- *SLURM_QOS*：类似 –qos
- *SLURM_RAMDISK_IMAGE*：类似 –ramdisk-image
- *SLURM_REMOTE_CWD*：类似 -D, –chdir=
- *SLURM_RESERVATION*：类似 –reservation
- *SLURM_RESTART_DIR*：类似 –restart-dir
- *SLURM_RESV_PORTS*：类似 –resv-ports
- *SLURM_SIGNAL*：类似 –signal

srun主要输入环境变量 VI

- *SLURM_STDERRMODE*：类似 -e, –error
- *SLURM_STDINMODE*：类似 -i, –input
- *SLURM_SRUN_REDUCE_TASK_EXIT_MSG*：如被设置，并且非0,那么具有相同退出码的连续的任务退出消息只显示一次
- *SLURM_STEP_GRES*：类似 –gres（仅对作业步有效，不影响作业分配），参见*SLURM_GRES*
- *SLURM_STEP_KILLED_MSG_NODE_ID=ID*：如被设置，当作业或作业步被信号终止时只特定ID的节点下显示信息
- *SLURM_STDOUTMODE*：类似 -o, –output
- *SLURM_TASK_EPILOG*：类似 –task-epilog
- *SLURM_TASK_PROLOG*：类似 –task-prolog
- *SLURM_TEST_EXEC*：如被定义，在计算节点执行之前先在本地节点上测试可执行程序

srun主要输入环境变量 VII

- *SLURM_THREAD_SPEC*：类似 –thread-spec
- *SLURM_THREADS*：类似 -T, –threads
- *SLURM_TIMELIMIT*：类似 -t, –time
- *SLURM_UNBUFFEREDIO*：类似 -u, –unbuffered
- *SLURM_USE_MIN_NODES*：类似 –use-min-nodes
- *SLURM_WAIT*：类似 -W, –wait
- *SLURM_WCKEY*：类似 -W, –wckey
- *SLURM_WORKING_DIR*：类似 -D, –chdir

srun主要输出环境变量 I

*srun*会在执行的节点上设置如下环境变量：

- *SLURM_CHECKPOINT_IMAGE_DIR*：Checkpoint镜像的存储目录
- *SLURM_CLUSTER_NAME*：集群名
- *SLURM_CPU_BIND_VERBOSE*：–cpu-bind 详细情况(quiet、verbose)
- *SLURM_CPU_BIND_TYPE*：–cpu-bind 类型(none、rank、map-cpu:、mask-cpu:)
- *SLURM_CPU_BIND_LIST*：–cpu-bind 映射或掩码列表
- *SLURM_CPU_FREQ_REQ*：需要的CPU频率资源，参见–cpu-freq和输入环境变量*SLURM_CPU_FREQ_REQ*
- *SLURM_CPUS_ON_NODE*：节点上的CPU颗数
- *SLURM_CPUS_PER_TASK*：每作业的CPU颗数，参见–cpus-per-task选项指定
- *SLURM_DISTRIBUTION*：分配的作业的分布类型，参见-m, –distribution
- *SLURM_GTIDS*：此节点上分布的全局任务号，从0开始，以,分隔

srun主要输出环境变量 II

- *SLURM_JOB_ACCOUNT*：作业的记账名
- *SLURM_JOB_CPUS_PER_NODE*：每个节点的CPU颗数
- *SLURM_JOB_DEPENDENCY*：依赖关系，参见–dependency选项
- *SLURM_JOB_ID*：作业号
- *SLURM_JOB_NAME*：作业名，参见–job-name选项或srun启动的命令名
- *SLURM_JOB_PARTITION*：作业使用的队列名
- *SLURM_JOB_QOS*：作业的服务质量QOS
- *SLURM_JOB_RESERVATION*：作业的高级资源预留
- *SLURM_LAUNCH_NODE_IPADDR*：任务初始启动节点的IP地址
- *SLURM_LOCALID*：节点本地任务号
- *SLURM_MEM_BIND_VERBOSE*：内存绑定详细情况，参见–mem-bind verbosity（quiet、verbose）

srun主要输出环境变量 III

- *SLURM_MEM_BIND_TYPE*：–mem-bind类型（none、rank、map-mem:、mask-mem:）
- *SLURM_MEM_BIND_LIST*：–mem-bind映射或掩码列表（<list of IDs or masks for this node>）
- *SLURM_NNODES*：分配的节点总数
- *SLURM_NODE_ALIASES*：分配的节点名、通信IP地址和节点名，每组内采用:分隔，组间通过,分隔，如：*SLURM_NODE_ALIASES*=0:1.2.3.4:foo,ec1:1.2.3.5:bar
- *SLURM_NODEID*：当前节点的相对节点号
- *SLURM_NODELIST*：分配的节点列表
- *SLURM_NTASKS*：任务总数
- *SLURM_PRIO_PROCESS*：作业提交时的调度优先级值（nice值）
- *SLURM_PROCID*：当前MPI秩号
- *SLURM_SRUN_COMM_HOST*：节点的通信IP

srun主要输出环境变量 IV

- *SLURM_SRUN_COMM_PORT*：srun的通信端口
- *SLURM_STEP_LAUNCHER_PORT*：作业步启动端口
- *SLURM_STEP_NODELIST*：作业步节点列表
- *SLURM_STEP_NUM_NODES*：作业步的节点总数
- *SLURM_STEP_NUM_TASKS*：作业步的任务总数
- *SLURM_STEP_TASKS_PER_NODE*：作业步在每个节点上的任务总数
- *SLURM_STEP_ID*：当前作业的作业步号
- *SLURM_SUBMIT_DIR*：提交作业的目录
- *SLURM_SUBMIT_HOST*：提交作业的节点名
- *SLURM_TASK_PID*：任务启动的进程号

srun主要输出环境变量 V

- *SLURM_TASKS_PER_NODE*：每个节点上启动的任务数，以*SLURM_NODELIST*中的节点顺序显示，以,分隔。如果两个或多个连续节点上的任务数相同，数后跟着(x#)，其中#是对应的节点数，如*SLURM_TASKS_PER_NODE*=2(x3),1”表示，前三个节点上的作业数为3，第四个节点上的任务数为1
- *SLURM_UMASK*：作业提交时的umask掩码
- *SLURMD_NODENAME*：任务运行的节点名
- *SRUN_DEBUG*：srun命令的调试详细信息级别，默认为3（info级）

srun多程序运行配置 I

Slurm支持一次申请多个节点，在不同节点上同时启动执行不同任务。为实现次功能，需要生成一个配置文件，在配置文件中做相应设置
配置文件中的注释必需第一列为#，配置文件包含以空格分隔的以下域（字段）：

- 任务范围(Task rank)：一个或多个任务秩
  - 多个值的话可以用逗号,分隔
  - 范围可以用两个用-分隔的整数表示，小数在前，大数在后
  - 如果最后一行为*，则表示全部其余未在前面声明的秩
  - 如没有指明可执行程序，则会显示错误信息：“No executable program specified for this task”
- 需要执行的可执行程序(Executable)：也许需要绝对路径指明
- 可执行程序的参数(Arguments)：“%t”将被替换为任务号；“%o”将被替换为任务号偏移（如配置的秩为“1-5”，则偏移值为“0-4”）。单引号可以防止内部的字符被解释。此域为可选项，任何在命令行中需要添加的程序参数都将加在配置文件中的此部分

srun多程序运行配置 II

例如，配置文件silly.conf内容为：

```
###################################################################
#␣srun␣multiple␣program␣configuration␣file
#
#␣srun␣-n8␣-l␣--multi-prog␣silly.conf
###################################################################
#4、5、6任务秩执行
4-6␣␣␣␣␣␣␣hostname
#1,7任务秩
1,7␣␣␣␣␣␣␣echo␣␣task:%t
#0、2、3任务秩执行
0,2-3␣␣␣␣␣echo␣␣offset:%o
```

srun多程序运行配置 III

运行：*srun -n8 -l --multi-prog silly.conf*
输出结果：

```
0: offset:0
1: task:1
2: offset:1
3: offset:2
4: node1
5: node2
6: node4
7: task:7
```

srun常见例子：使用8个CPU核(-n8)运行作业

使用8个CPU核(-n8)运行作业，并在标准输出上显示任务号(-l)：

srun -n8 -l hostname

输出结果:

```
0: node0
1: node0
2: node1
3: node1
4: node2
5: node2
6: node3
7: node3
```

srun常见例子：在脚本中使用-r2参数使其在第2号（分配的节点号从0开始）开始的两个节点上运行，并采用实时分配模式而不是批处理模式运行

脚本*test.sh*内容：

```
#!/bin/sh
echo SLURM_NODELIST
srun -lN2 -r2 hostname
srun -lN2 hostname
```

运行：*salloc -N4 test.sh*
输出结果：

```
dev[7-10]
0: node9
1: node10
0: node7
1: node8
```

srun常见例子：在分配的节点上并行运行两个作业步

脚本*test.sh*内容：

```
#!/bin/bash
srun -lN2 -n4 -r 2 sleep 60 &
srun -lN2 -r 0 sleep 60 &
sleep 1
squeue
squeue -s
wait
```

注意：& 表示前面命令后台运行，直接执行后面命令

运行：*salloc -N4 test.sh*

输出结果：

```
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST
  65641     batch  test.sh   grondo  R       0:01      4 dev[7-10]

STEPID     PARTITION     USER      TIME NODELIST
65641.0        batch   grondo      0:01 dev[7-8]
65641.1        batch   grondo      0:01 dev[9-10]
```

srun常见例子：运行MPICH作业

脚本*test.sh*内容：

```
#!/bin/sh
MACHINEFILE="nodes.$SLURM_JOB_ID"

# 生成MPICH所需的包含节点名的machinfile文件
srun -l /bin/hostname | sort -n | awk '{print $2}' > $MACHINEFILE

# 运行MPICH作业
mpirun -np $SLURM_NTASKS -machinefile $MACHINEFILE mpi-app

rm $MACHINEFILE
```

采用2个节点（-N2）共4个CPU核（-n4）运行：*salloc -N2 -n4 test.sh*

srun常见例子：利用不同节点号（*SLURM_NODEID*）运行不同作业

节点号从0开始

脚本*test.sh*内容：

```
case␣$SLURM_NODEID␣in
⟶0)␣echo␣"I am running on "
⟶⟶hostname
⟶⟶;;
⟶1)␣hostname
⟶⟶echo␣"is where I am running"
⟶⟶;;
esac
```

运行：*srun -N2 test.sh*

输出：

```
dev0
is where I am running
I am running on
ev1
```

srun常见例子：利用多核选项控制任务执行

采用2个节点（-N2），每节点4颗CPU每颗CPU 2颗CPU核（-B 4-4:2-2），运行作业：

srun -N2 -B 4-4:2-2 a.out

srun常见例子：运行GPU作业

脚本*gpu.script*内容：

```
#!/bin/bash
srun -n1 -p GPU-V100 --gres=gpu:v100:2 prog1 &
wait
```

- -p GPU-V100 指定采用GPU队列GPU-V100
- –gres=gpu:v100:2指明每个节点使用2块NVIDIA V100 GPU卡

srun常见例子：排它性独占运行作业 –exclusive

脚本*my.script*内容：

```
#!/bin/bash
srun␣--exclusive␣-n4␣prog1␣&
srun␣--exclusive␣-n3␣prog2␣&
srun␣--exclusive␣-n1␣prog3␣&
srun␣--exclusive␣-n1␣prog4␣&
wait
```

批处理方式提交作业：sbatch

Slurm支持利用*sbatch*命令采用批处理方式运行作业

- *sbatch*命令在脚本正确传递给作业调度系统后立即退出，同时获取到一个作业号
- 作业等所需资源满足后开始运行
- *sbatch*提交一个批处理作业脚本到Slurm
  - 批处理脚本名可以在命令行上通过传递给*sbatch*
  - 如没有指定文件名，则*sbatch*从标准输入中获取脚本内容

sbatch脚本格式

脚本文件基本格式：

- 第一行以#!/bin/sh等指定该脚本的解释程序，/bin/sh可以变为/bin/bash、/bin/csh等
- 在可执行命令之前的每行“#SBATCH”前缀后跟的参数作为作业调度系统参数

默认，标准输出和标准出错都定向到同一个文件*slurm-%j.out*，“%j”将被作业号代替

脚本*myscript*内容：

```
#!/bin/sh
#SBATCH --time=1
#SBATCH -p serial
srun hostname | sort
```

sbatch脚本运行

采用4个节点（-N4）运行：*sbatch -p batch -N4 myscript*

- 虽然脚本中“#SBATCH -p serial”指定用serial队列，但命令行中的*-p batch*优先级更高
- 提交成功后有类似输出：

```
salloc: Granted job allocation 65537
```

其中65537为分配的作业号

- 程序结束后的作业日志文件*slurm-65537.out*显示：

```
node1
node2
node3
node4
```

从标准输入获取脚本内容

可采用以下两种方式之一：

1. 运行*sbatch -N4*，显示等待后输入：

```
#!/bin/sh
srun␣hostname␣|␣sort
```

输入以上内容后按CTRL+D终止输入

2. 运行*sbatch -N4 <<EOF*回车

```
>␣#!/bin/sh
>␣srun␣hostname␣|␣sort
>␣EOF
```

- 第一个EOF表示输入内容的开始标识符
- 最后的EOF表示输入内容的终止标识符，在两个EOF之间的内容为实际执行的内容
- >实际上是每行输入回车后自动在下一行出现的提示符

以上两种方式输入结束后将显示：

```
sbatch: Submitted batch job 65541
```

sbatch选项

常见主要选项与*srun*类似，下面主要介绍下特有的选项：

- –export-file=<filename | fd>：通过文件filename设定环境变量。文件中的环境变量格式为NAME=value，变量之间通过空格分隔
- –no-requeue：任何情况下都不重新运行
- -F, –nodefile=<node file>：类似–nodelist，只不过是通过文件设定节点名，节点名可以在几行内，重复的节点名将被忽略，节点名顺序不重要，将被自动排序
- –ignore-pbs：忽略脚本文件中的全部“#SBATCH”选项
- –parsable：仅输出作业号和集群名（如果有的话），采用;分隔。错误仍旧显示
- –test-only：作业不真正执行，仅仅测试脚本是否可以执行，并给出预计开始执行时间

sbatch主要输入环境变量 I

一些选项可通过环境变量来设置，*命令行的选项优先级高于设置的环境变量*，将覆盖掉环境变量的设置。
环境变量与对应的参数如下：

- *SBATCH_ACCOUNT*：类似-A、–account
- *SBATCH_ACCTG_FREQ*：类似–acctg-freq
- *SBATCH_ARRAY_INX*：类似-a、–array
- *SBATCH_BLRTS_IMAGE*：类似–blrts-image
- *SBATCH_BURST_BUFFER*：类似–bb
- *SBATCH_CHECKPOINT*：类似–checkpoint
- *SBATCH_CHECKPOINT_DIR*：类似–checkpoint-dir
- *SBATCH_CLUSTERS* 或 *SLURM_CLUSTERS*：类似–clusters
- *SBATCH_CNLOAD_IMAGE*：类似–cnload-image

sbatch主要输入环境变量 II

- *SBATCH_CONN_TYPE*：类似–conn-type
- *SBATCH_CORE_SPEC*：类似–core-spec
- *SBATCH_DEBUG*：类似-v、–verbose
- *SBATCH_DISTRIBUTION*：类似-m、–distribution
- *SBATCH_EXCLUSIVE*：类似–exclusive
- *SBATCH_EXPORT*：类似–export
- *SBATCH_GEOMETRY*：类似-g、–geometry
- *SBATCH_GET_USER_ENV*：类似–get-user-env
- *SBATCH_GRES_FLAGS*：类似–gres-flags
- *SBATCH_HINT* 或 *SLURM_HINT*：类似–hint
- *SBATCH_IGNORE_PBS*：类似–ignore-pbs
- *SBATCH_IMMEDIATE*：类似-I、–immediate

sbatch主要输入环境变量 III

- *SBATCH_IOLOAD_IMAGE*：类似–ioload-image
- *SBATCH_JOBID*：类似–jobid
- *SBATCH_JOB_NAME*：类似-J、–job-name
- *SBATCH_LINUX_IMAGE*：类似–linux-image
- *SBATCH_MEM_BIND*：类似–mem-bind
- *SBATCH_MLOADER_IMAGE*：类似–mloader-image
- *SBATCH_NETWORK*：类似–network
- *SBATCH_NO_REQUEUE*：类似–no-requeue
- *SBATCH_NO_ROTATE*：类似-R、–no-rotate
- *SBATCH_OPEN_MODE*：类似–open-mode
- *SBATCH_OVERCOMMIT*：类似-O、–overcommit
- *SBATCH_PARTITION*：类似-p、–partition

sbatch主要输入环境变量 IV

- *SBATCH_POWER*：类似–power
- *SBATCH_PROFILE*：类似–profile
- *SBATCH_QOS*：类似–qos
- *SBATCH_RAMDISK_IMAGE*：类似–ramdisk-image
- *SBATCH_RESERVATION*：类似–reservation
- *SBATCH_REQUEUE*：类似–requeue
- *SBATCH_SIGNAL*：类似–signal
- *SBATCH_THREAD_SPEC*：类似–thread-spec
- *SBATCH_TIMELIMIT*：类似-t、–time
- *SBATCH_USE_MIN_NODES*：类似–use-min-nodes
- *SBATCH_WAIT*：类似-W、–wait
- *SBATCH_WAIT_ALL_NODES*：类似–wait-all-nodes

sbatch主要输入环境变量 V

- *SBATCH_WAIT4SWITCH*：需要交换的最大时间，参见See–switches
- *SBATCH_WCKEY*：类似–wckey
- *SLURM_CONF*：Slurm配置文件路径
- *SLURM_EXIT_ERROR*：设定Slurm出错时的退出码
- *SLURM_STEP_KILLED_MSG_NODE_ID=ID*：如果设置，当作业或作业步被信号终止时，只有指定ID的节点记录

sbatch主要输出环境变量 I

Slurm将在作业脚本中输出以下变量，作业脚本可以使用这些变量：

- *SBATCH_CPU_BIND*：由–cpu-bind选项设定
- *SBATCH_CPU_BIND_VERBOSE*：如–cpu-bind选项包含verbose选项时，则由其设定
- *SBATCH_CPU_BIND_TYPE*：由–cpu-bind选项设定
- *SBATCH_CPU_BIND_LIST*：CPU绑定时设定的bit掩码
- *SBATCH_MEM_BIND*：–mem-bind选项设定
- *SBATCH_MEM_BIND_VERBOSE*：如–mem-bind选项包含verbose选项时，则由其设定
- *SBATCH_MEM_BIND_TYPE*：由–mem-bind选项设定
- *SBATCH_MEM_BIND_LIST*：内存绑定时设定的bit掩码
- *SLURM_ARRAY_TASK_ID*：作业组ID（索引）号
- *SLURM_ARRAY_TASK_MAX*：作业组最大ID号
- *SLURM_ARRAY_TASK_MIN*：作业组最小ID号

sbatch主要输出环境变量 II

- *SLURM_ARRAY_TASK_STEP*：作业组索引步进间隔
- *SLURM_ARRAY_JOB_ID*：作业组主作业号
- *SLURM_CLUSTER_NAME*：集群名
- *SLURM_CPUS_ON_NODE*：分配的节点上的CPU颗数
- *SLURM_CPUS_PER_TASK*：每个任务的CPU颗数，只有–cpus-per-task选项设定时才有
- *SLURM_DISTRIBUTION*：类似 -m, –distribution
- *SLURM_GTIDS*：在此节点上运行的全局任务号。以0开始，逗号,分隔
- *SLURM_JOB_ID*：作业号
- *SLURM_JOB_CPUS_PER_NODE*：每个节点上的CPU颗数
- *SLURM_JOB_DEPENDENCY*：作业依赖信息，由–dependency选项设置
- *SLURM_JOB_NAME*：作业名
- *SLURM_JOB_NODELIST*：分配的节点名列表

sbatch主要输出环境变量 III

- *SLURM_JOB_NUM_NODES*：分配的节点总数
- *SLURM_JOB_PARTITION*：使用的队列名
- *SLURM_JOB_RESERVATION*：作业预留
- *SLURM_LOCALID*：节点本地任务号
- *SLURM_MEM_PER_CPU*：类似–mem-per-cpu，每颗CPU的内存
- *SLURM_MEM_PER_NODE*：类似–mem，每个节点的内存
- *SLURM_NODE_ALIASES*：分配的节点名、通信IP地址和主机名组合，类似 *SLURM_NODE_ALIASES*=ec0:1.2.3.4:foo,ec1:1.2.3.5:bar
- *SLURM_NODEID*：分配的节点号
- *SLURM_NTASKS*：类似-n, –ntasks，总任务数，CPU核数
- *SLURM_NTASKS_PER_CORE*：每个CPU核分配的任务数
- *SLURM_NTASKS_PER_NODE*：每个节点上的任务数

sbatch主要输出环境变量 IV

- *SLURM_NTASKS_PER_SOCKET*：每颗CPU上的任务数，仅–ntasks-per-socket选项设定时设定
- *SLURM_PRIO_PROCESS*：进程的调度优先级（nice值）
- *SLURM_PROCID*：当前进程的MPI秩
- *SLURM_PROFILE*：类似–profile
- *SLURM_RESTART_COUNT*：因为系统失效等导致的重启次数
- *SLURM_SUBMIT_DIR*：sbatch启动目录，即提交作业时目录
- *SLURM_SUBMIT_HOST*：sbatch启动的节点名，即提交作业时节点
- *SLURM_TASKS_PER_NODE*：每节点上的任务数，以*SLURM_NODELIST*中的节点顺序显示，以,分隔。如果两个或多个连续节点上的任务数相同，数后跟着(x#)，其中#是对应的节点数，如“*SLURM_TASKS_PER_NODE*=2(x3),1”表示前三个节点上的作业数为3，第四个节点上的任务数为1
- *SLURM_TASK_PID*：任务的进程号PID
- *SLURMD_NODENAME*：执行作业脚本的节点名

串行作业提交

对于串行程序，可类似下面两者之一：

1. 直接采用*sbatch -n1 -o job-%j.log -e job-%.err yourprog*方式运行
2. 编写名为*serial_job.sh*（此脚本名可以按照用户喜好命名）的串行作业脚本，内容如下：

```
#!/bin/sh
#An example for serial job.
#SBATCH -J job_name
#SBATCH -o job-%j.log -e job-%j.err
echo Running on hosts
echo Time is `date`
echo Directory is $PWD
echo This job runs on the following nodes:
echo $SLURM_JOB_NODELIST
module load intel/2016.3.210
echo This job has allocated 1 cpu core.
./yourprog
```

必要时需在脚本中利用*module*命令设置所需的环境，如*module load intel/2016.3.210*

作业脚本编写完成后，可以按照下面命令提交作业：

sbatch -n1 -p serail serial_job.sh

单节点OpenMP共享内存并行作业提交

对于单节点OpenMP共享内存并行程序，可编写名为*omp_job.sh*的作业脚本，其内容如下：

```
#!/bin/sh
#An example for serial job.
#SBATCH -J job_name
#SBATCH -o job-%j.log -e job-%j.err
#SBATCH -N 1 -n 8
echo Running on hosts
echo Time is `date`
echo Directory is $PWD
echo This job runs on the following nodes:
echo $SLURM_JOB_NODELIST
module load intel/2016.3.210
echo This job has allocated 1 cpu core.
export OMP_NUM_THREADS=8
./yourprog
```

- 相对于串行作业脚本，主要增加export OMP_NUM_THREADS=8和#SBATCH -N 1 -n 8：表示使用一个节点内部的8个核(-n 8)，从而保证是在同一个节点内(-N 1)，需几个核就设置-n为几，-n后跟的不能超过单节点内CPU核数
- 作业脚本编写完成后，可以按照下面命令提交作业：

  sbatch omp_job.sh

MPI并行作业提交 I

与串行作业类似，对于MPI并行作业，则需编写类似下面脚本*mpi_job.sh*：

```
#!/bin/sh
#An example for MPI job.
#SBATCH -J job_name
#SBATCH -o job-%j.log -e job-%j.err
#SBATCH -N 1 -n 8
echo Time is `date`
echo Directory is $PWD
echo This job runs on the following nodes:
echo $SLURM_JOB_NODELIST
echo This job has allocated $SLURM_JOB_CPUS_PER_NODE cpu cores.
module load intelmpi/5.1.3.210
#module load mpich/3.2/intel/2016.3.210
#module load openmpi/2.0.0/intel/2016.3.210
MPIRUN=mpirun #Intel mpi and Open MPI
#MPIRUN=mpiexec #MPICH
MPIOPT="-env I_MPI_FABRICS shm:ofa" #Intel MPI 2018
#MPIOPT="-env I_MPI_FABRICS shm:ofi" #Intel MPI 2019
#MPIOPT="--mca plm_rsh_agent ssh --mca btl self,openib,sm" #Open MPI
#MPIOPT="-iface ib0" #MPICH3
#目前新版的MPI可以自己找寻高速网络配置，可以设置MPIOPT为空，如去掉下行的#
#MPIOPT=
$MPIRUN $MPIOPT ./yourprog
```

MPI并行作业提交 II

- 与串行程序的脚本相比，主要不同之处在于需采用mpirun或mpiexec的命令格式提交并行可执行程序
- 采用不同MPI提交时，需要打开上述对应的选项
- 与串行作业类似，可使用下面方式提交：
  sbatch mpi_job.sh

GPU作业提交

运行GPU作业，需提交时利用–gres=gpu等指明需要的GPU资源并用-p指明采用等GPU队列
脚本*gpu_job.sh*内容：

```
#!/bin/sh
#An example for gpu job.
#SBATCH -J job_name
#SBATCH -o job-%j.log
#SBATCH -e job-%j.err
#SBATCH -N 1 -n 8 -p GPU-V100 --gres=gpu:v100:2
./your-gpu-prog
wait
```

- -p GPU-V100 指定采用GPU队列GPU-V100
- –gres=gpu:v100:2指明每个节点使用2块NVIDIA V100 GPU卡

作业获取的节点名及对应CPU核数解析

- 作业调度系统主要负责分配节点及该节点分配的CPU核数等，在Slurm作业脚本中利用环境变量可以获取分配到的节点名(SLURM_JOB_NODELIST及对应核数(SLURM_JOB_CPUS_PER_NODE)或对应的任务数(SLURM_TASKS_PER_NODE)，然后根据自己程序原始的命令在Slurm脚本中进行修改就成
- SLURM_JOB_NODELIST及SLURM_TASKS_PER_NODE有一定的格式，可以在自己的Slurm脚本中获取自己的节点等信息并解析成自己所需的

作业获取的节点名及对应CPU核数解析脚本 I

- 要加在自己的Slurm脚本中，而不是先提交下面脚本获取节点名后放到Slurm脚本中，其原因在于除非提交时指定节点名，否则每次提交后获取的节点名等是有可能变的
- 比如针对star-cmm+软件，原来的方式是：

STATCMM+PATH/bin/starccm+ -rsh ssh -batch -power -on cnode400:40,cnode432:40 FireAndSmokeResampled.sim >residual.log

- 则修改为以下脚本：

```
#!/bin/bash
#Auther␣HM_Li<hmli@ustc.edu.cn>
#SLURM_JOB_NODELIST=cnode[001−003,005,440−441]
BASENAME=${SLURM_JOB_NODELIST%[*}
LIST=${SLURM_JOB_NODELIST#*[}
LIST=${LIST%]}
IDLIST=''
for␣i␣in␣‘echo␣$LIST␣|␣tr␣',’␣’ ’‘
do
−−→if␣[[␣$i␣=~␣'-'␣]];␣then
−−→−−→IDLIST=$IDLIST’ ’‘seq␣-w␣‘echo␣$i␣|␣tr␣'-'␣’ ’‘‘
−−→else
−−→−−→IDLIST=$IDLIST’ ’$i
```

作业获取的节点名及对应CPU核数解析脚本 II

```
    fi
done
NODELIST=''
for i in $IDLIST
do
    NODELIST=$NODELIST" $BASENAME"$i
done
echo -e "Node list: \n\t"$NODELIST

#SLURM_TASKS_PER_NODE='40(x3),23,20(x2)'
#SLURM_JOB_CPUS_PER_NODE='40(x3),23,20(x2)'
CORELIST=''
for i in `echo $SLURM_JOB_CUPS_PER_NODE| tr ',' ' '`
do
    if [[ $i =~ 'x' ]]; then
        read CORES NODES <<< `echo $i| tr '(x)' ' '`
        for n in `seq 1 $NODES`
        do
            CORELIST=$CORELIST' '$CORES
        done
    else
        CORELIST=$CORELIST' '$i
```

作业获取的节点名及对应CPU核数解析脚本 III

```
    fi
done
echo -e "\nCPU Core list: \n\t$CORELIST"

echo -e "\nNode list with corresponding CPU Cores: "
CARR=(${CORELIST})
i=0
for n in $NODELIST
do
    echo -e "\t"$n: ${CARR[$i]}
    i=$((i+1))
done
```

下载：http://hmli.ustc.edu.cn/doc/training/slurm/node-parse.sh

分配式提交作业：salloc

- *salloc*将获取作业的分配后执行命令，当命令结束后释放分配的资源。其基本语法为：
  - *salloc [options] [<command> [command args]]*
  - command可以是任何是用户想要用的程序，典型的为xterm或包含srun命令的shell。如果后面没有跟命令，那么将执行Slurm系统slurm.conf配置文件中通过SallocDefaultCommand设定的命令。如果SallocDefaultCommand没有设定，那么将执行用户的默认shell
- 注意：*salloc*逻辑上包括支持保存和存储终端行设置，并且设计为采用前台方式执行。如果需要后台执行*salloc*，可以设定标准输入为某个文件，如：
  salloc -n16 a.out </dev/null &

salloc主要选项 I

- -A, –account=<account>：指定此作业的责任资源为账户<account>
- –acctg-freq：指定作业记账和剖面信息采样间隔。支持的格式为–acctg-freq=<datatype>=<interval>，其中<datatype>=<interval>指定了任务抽样间隔或剖面抽样间隔。多个<datatype>=<interval>可以采用,分隔（默认为30秒）：
  - task=<interval>：以秒为单位的任务抽样（需要jobacct_gather插件启用）和任务剖面（需要acct_gather_profile插件启用）间隔
  - energy=<interval>：以秒为单位的能源剖面抽样间隔，需要acct_gather_energy插件启用
  - network=<interval>：以秒为单位的InfiniBand网络剖面抽样间隔，需要acct_gather_infiniband 插件启用
  - filesystem=<interval>：以秒为单位的文件系统剖面抽样间隔，需要acct_gather_filesystem插件启用

salloc主要选项 II

- -B –extra-node-info=<sockets[:cores[:threads]]>：选择满足<sockets[:cores[:threads]]>的节点，*可以表示对应选项不做限制。对应限制可以采用下面对应选项：
  - –sockets-per-node=<sockets>
  - –cores-per-socket=<cores>
  - –threads-per-core=<threads>
- –begin=<time>：设定开始分配资源运行的时间。时间格式可以为HH:MM:SS，或者添加AM、PM等，也可以采用MMDDYY、MM/DD/YY或YYYY-MM-DD格式指定日期，含有日期及时间的格式为：YYYY-MM-DD[THH:MM[:SS]]，也可以采用类似now+时间单位的方式，时间单位可以为seconds（默认）、minutes、hours、days和weeks、today、tomorrow等，例如：
  - –begin=16:00
  - –begin=now+1hour
  - –begin=now+60
  - –begin=2010-01-20T12:34:00
- –bell：分配资源是终端响铃，参见–no-bell

salloc主要选项 III

- –comment=<string>：添加注释
- –contiguous：设定分配的节点必须是连续的
- –cores-per-socket=<cores>：分配的节点需要至少每颗CPU <cores>CPU核
- -c, –cpus-per-task=<ncpus>：设定每个任务的CPU核数
- –deadline=<OPT>：如果在此deadline（start > (deadline - time[-min]）之前没有结束，那么移除此作业。默认没有deadline，时间格式可为：
  - HH:MM[:SS] [AM|PM]
  - MMDD[YY]或MM/DD[/YY]或MM.DD[.YY]
  - MM/DD[/YY]-HH:MM[:SS]
  - YYYY-MM-DD[THH:MM[:SS]]]
- -d, –dependency=<dependency_list>：满足依赖条件<dependency_list>后开始分配。<dependency_list>可以为<type:job_id[:job_id][,type:job_id[:job_id]]>或<type:job_id[:job_id][?type:job_id[:job_id]]>。依赖条件如果用,分隔，则各依赖条件都需要满足；如果采用?分隔，那么只要任意条件满足即可。可以为：

salloc主要选项 IV

- after:job_id[:jobid...]：当指定作业号的作业结束后开始运行
- afterany:job_id[:jobid...]：当指定作业号的任意作业结束后开始运行
- aftercorr:job_id[:jobid...]：当相应的任务号任务结束后，此作业组中的任务运行
- afternotok:job_id[:jobid...]：当指定作业号的作业结束时具有异常状态（非零退出码、节点失效、超时等）时
- afterok:job_id[:jobid...]：当指定的作业正常结束（退出码为0）时开始运行
- expand:job_id：分配给此作业的资源将扩展给指定作业
- singleton：等任意通账户的相同作业名的前置作业结束时

- -D, –chdir=<path>：在执行前切换到<path>目录
- –exclusive[=user|mcs]：不与user用户或mcs选项的作业共享节点
- -F, –nodefile=<node file>：类似–nodelist指定需要运行的节点，但此列表包含一个含有节点名的文件
- -H, –hold：设定作业将被提交为挂起状态。挂起的作业可以利用scontrol release <job_id>使其排队运行

salloc主要选项 V

- -h, –help：显示帮助信息
- –hint=<type>：绑定任务到应用暗示：
  - compute_bound：选择设定计算边界应用：采用每个socket的所有CPU核，每颗CPU核一个进程
  - memory_bound：选择设定内存边界应用：仅采用每个socket的1颗CPU核，每颗CPU核一个进程
  - [no]multithread：在in-core multi-threading是否采用额外的线程，对通信密集型应用有益。仅当task/affinity插件启用时
  - help：显示帮助信息
- -I, –immediate[=<seconds>]：在<seconds>秒内资源未满足的话立即退出。格式可以为“-I60”，但不能之间有空格是“-I 60”
- -J, –job-name=<jobname>：设定作业名，默认为命令名

salloc主要选项 VI

- -K, –kill-command[=signal]：设定需要终止时的signal，默认，如没指定，则对于交互式作业为SIGHUP，对于非交互式作业为SIGTERM。格式类似可以为“-K1”，但不能包含空格为“-K 1”
- -k, –no-kill：如果分配的节点失效，那么不会自动终止
- -L, –licenses=<license>：设定使用的<license>
- –mem=<MB>：设定每个节点的内存大小，后缀可以为[K|M|G|T]
- –mem-per-cpu=<MB>：设定分配的每颗CPU对应最小内存，后缀可以为[K|M|G|T]
- –mincpus=<n>：设定每个节点最小的逻辑CPU核/处理器
- -N, –nodes=<minnodes[-maxnodes]>：设定需要的最小和最大数
- -n, –ntasks=<number>：设定所需要的任务总数
- –nice[=adjustment]：设定NICE调整值。负值提高优先级，正值降低优先级。调整范围为：+/- 2147483645

salloc主要选项 VII

- –ntasks-per-core=<ntasks>：设定每颗CPU核运行的任务数
- –ntasks-per-node=<ntasks>：设定每个节点运行的任务数
- –ntasks-per-socket=<ntasks>：设定每颗CPU运行的任务数
- –no-bell：资源分配时不终端响铃。参见–bell
- –no-shell：分配资源后立即退出，而不运行命令。但Slurm作业仍旧被生成，在其激活期间，且保留这些激活的资源。用户会获得一个没有附带进程和任务的作业号，用户可以采用提交srun命令到这些资源
- -O, –overcommit：采用此选项可以使得每颗CPU运行不止一个任务
- -p, –partition=<partition_names>：设定使用的队列
- -Q, –quiet：采用安静模式运行，一般信息将不显示，但错误信息仍将被显示
- -S, –core-spec=<num>：指定预留的不被作业使用的各节点CPU核数

salloc主要选项 VIII

- –signal=<sig_num>[@<sig_time>]：设定到其终止时间前信号时间<sig_time>秒时的信号。由于Slurm事件处理的时间精度，信号有可能比设定时间早60秒。信号可以为10 或USER1，信号时间sig_time必须在0到65535之间，如没指定，则默认为60秒
- –sockets-per-node=<sockets>：设定每个节点的CPU颗数
- -t, –time=<time>：设定作业总运行时间，当设定的时间到时，作业将被杀掉。时间格式为：minutes、minutes:seconds、hours:minutes:seconds、days-hours、days-hours:minutes和 days-hours:minutes:seconds
- –thread-spec=<num>：设定指定预留的不被作业使用的各节点线程数
- –threads-per-core=<threads>：设定需要的各CPU核线程数
- –time-min=<time>：设定作业分配的最小时间。时间格式为：minutes、minutes:seconds、hours:minutes:seconds、days-hours、days-hours:minutes和 days-hours:minutes:seconds
- –tmp=<MB>：设定/tmp目录最小磁盘空间
- -u, –usage：显示简要帮助信息

salloc主要选项 IX

- –use-min-nodes：设定如果给了一个节点数范围，选择较小的数
- -V, –version：显示版本信息
- -v, –verbose：显示冗余信息
- -w, –nodelist=<node name list>：设定运行的节点列表
- –wait-all-nodes=<value>：控制当节点准备好时何时运行命令。默认，当分配的资源准备好后*salloc*命令立即返回。<value>可以为：
  - 0：当分配的资源可以分配时立即执行，比如有节点以重启好
  - 1：只有当分配的所有节点都准备好时才执行
- -x, –exclude=<node name list>：设定排除运行此作业的节点列表

salloc主要输入环境变量 I

- *SALLOC_ACCOUNT*：类似-A, –account
- *SALLOC_ACCTG_FREQ*：类似–acctg-freq
- *SALLOC_BELL*：类似–bell
- *SALLOC_BURST_BUFFER*：类似–bb
- *SALLOC_CONN_TYPE*：类似–conn-type
- *SALLOC_CORE_SPEC*：类似–core-spec
- *SALLOC_DEBUG*：类似-v, –verbose
- *SALLOC_EXCLUSIVE*：类似–exclusive
- *SALLOC_GEOMETRY*：类似-g, –geometry
- *SALLOC_GRES_FLAGS*：类似–gres-flags
- *SALLOC_HINT or SLURM_HINT*：类似–hint
- *SALLOC_IMMEDIATE*：类似-I, –immediate

salloc主要输入环境变量 II

- *SALLOC_JOBID*：类似–jobid
- *SALLOC_KILL_CMD*：类似-K, –kill-command
- *SALLOC_MEM_BIND*：类似–mem-bind
- *SALLOC_NETWORK*：类似–network
- *SALLOC_NO_BELL*：类似–no-bell
- *SALLOC_NO_ROTATE*：类似-R, –no-rotate
- *SALLOC_OVERCOMMIT*：类似-O, –overcommit
- *SALLOC_PARTITION*：类似-p, –partition
- *SALLOC_POWER*：类似–power
- *SALLOC_PROFILE*：类似–profile
- *SALLOC_QOS*：类似–qos
- *SALLOC_RESERVATION*：类似–reservation

salloc主要输入环境变量 III

- *SALLOC_SIGNAL*：类似–signal
- *SALLOC_THREAD_SPEC*：类似–thread-spec
- *SALLOC_TIMELIMIT*：类似-t, –time
- *SALLOC_USE_MIN_NODES*：类似–use-min-nodes
- *SALLOC_WAIT_ALL_NODES*：类似–wait-all-nodes
- *SALLOC_WCKEY*：类似–wckey
- *SLURM_CONF*：Slurm配置文件路径
- *SLURM_EXIT_ERROR*：错误退出代码
- *SLURM_EXIT_IMMEDIATE*：当–immediate选项时指定的立即退出代码

salloc主要输出环境变量 I

- *SLURM_CLUSTER_NAME*：集群名
- *SLURM_CPUS_PER_TASK*：每个任务分配的CPU数
- *SLURM_DISTRIBUTION*：类似-m, –distribution
- *SLURM_JOB_ACCOUNT*：账户名
- *SLURM_JOB_ID*（*SLURM_JOBID*为向后兼容）：作业号
- *SLURM_JOB_CPUS_PER_NODE*：分配的每个节点CPU数
- *SLURM_JOB_NODELIST*（*SLURM_NODELIST*为向后兼容）：分配的节点名列表
- *SLURM_JOB_NUM_NODES*（*SLURM_NNODES*为向后兼容）：作业分配的节点数
- *SLURM_JOB_PARTITION*：作业使用的队列名
- *SLURM_JOB_QOS*：作业的QOS
- *SLURM_JOB_RESERVATION*：预留的作业资源
- *SLURM_MEM_BIND*：–mem-bind选项指定的值

salloc主要输出环境变量 II

- *SLURM_MEM_PER_CPU*：类似–mem-per-cpu
- *SLURM_MEM_PER_NODE*：类似–mem
- *SLURM_SUBMIT_DIR*：运行salloc时的目录
- *SLURM_SUBMIT_HOST*：运行salloc时的节点名
- *SLURM_NODE_ALIASES*：分配的节点名、通信地址和主机名，格式类似 *SLURM_NODE_ALIASES*=ec0:1.2.3.4:foo,ec1:1.2.3.5:bar
- *SLURM_NTASKS*：类似-n, –ntasks
- *SLURM_NTASKS_PER_NODE*：–ntasks-per-node选项设定的值
- *SLURM_PROFILE*：类似–profile
- *SLURM_TASKS_PER_NODE*：每个节点的任务数。值以,分隔，并与 *SLURM_NODELIST* 顺序一致。如果连续的节点有相同的任务数，那么数后面跟有“(x#)”，其中“#”是重复次数。如：“*SLURM_TASKS_PER_NODE*=2(x3),1

salloc例子

- 获取分配，并打开xterm，以便srun可以交互式输入：*salloc -N16 xterm* 将输出：

```
salloc: Granted job allocation 65537
(at this point the xterm appears, and salloc waits for xterm to exit)
salloc: Relinquishing job allocation 65537
```

- 获取分配并并行运行应用：*salloc -N5 srun -n10 myprogram*

将文件同步到各节点：sbcast

*sbcast*命令可以将文件同步到各计算节点对应目录

- 一般用户主目录是共享的，一般不需要此命令
- 如果用户需要将某些文件传递到分配给作业的各节点*/tmp*等非共享目录，那么可以考虑此命令
- *sbcast*命令的基本语法为：*sbcast [-CfFjpstvV] SOURCE DEST*
- 此命令仅对批处理作业或在Slurm资源分配后生成的shell中起作用。SOURCE是当前节点上文件名，DEST为分配给此作业的对应节点将要复制到文件全路径

sbcast主要参数

- -C [library], –compress[=library]：设定采用压缩传递，及其使用的压缩库，[library]可以为lz4（默认）、zlib
- -f, –force：强制模式，如果目标文件存在，那么将直接覆盖
- -F number, –fanout=number：设定用于文件传递时的消息扇出，当前最大值为8
- -j jobID[.stepID], –jobid=jobID[.stepID]：指定使用的作业号
- -p, –preserve：保留源文件的修改时间、访问时间和模式等
- -s size, –size=size：设定广播时使用的块大小。size可以具有k或m后缀，默认单位为比特(Byte)。默认大小为文件大小或8MB
- -t seconds, fB–timeout=seconds：设定消息的超时时间
- -v, –verbose：显示冗余信息
- -V, –version：显示版本信息

sbcast主要环境变量

- SBCAST_COMPRESS：类似-C, –compress
- SBCAST_FANOUT：类似-F number, fB–fanout=number
- SBCAST_FORCE：类似-f, –force
- SBCAST_PRESERVE：类似-p, –preserve
- SBCAST_SIZE：类似-s size, –size=size
- SBCAST_TIMEOUT：类似-t seconds, fB–timeout=seconds
- SLURM_CONF：Slurm配置文件

sbcast例子

将*my.prog*传到*/tmp/my.proc*，且执行：

- 生成脚本*my.job*：

```
#!/bin/bash
sbcast␣my.prog␣/tmp/my.prog
srun␣/tmp/my.prog
```

- 提交：
  sbatch --nodes=8 my.job

吸附到作业步：sattach

*sattach*可以吸附到一个运行中的Slurm作业步

- 通过吸附，可以获取所有任务的IO流等，有时也可以用于并行调试器
- 基本语法：*sattach [options] <jobid.stepid>*

sattach主要参数

- -h, –help：显示帮助信息
- –input-filter[=]<task number>、–output-filter[=]<task number>、–error-filter[=]<task number>：仅传递标准输入到一个单独任务或打印一个单个任务中的标准输出或标准错误输出
- -l, –label：在每行前显示其对应的任务号
- –layout：联系slurmctld获得任务层信息，打印层信息后退出吸附作业步
- –pty：在伪终端上执行0号任务。与–input-filter、–output-filter或–error-filter不兼容
- -Q, –quiet：安静模式。不显示一般的sattach信息，但错误信息仍旧显示
- -u, –usage：显示简要帮助信息
- -V, –version：显示版本信息
- -v, –verbose：显示冗余信息

sattach主要输入环境变量

- SLURM_CONF：Slurm配置文件
- SLURM_EXIT_ERROR：Slurm退出错误代码

sattach例子

- *sattach 15.0*
- *sattach --output-filter 5 65386.15*

终止作业：scancel job_id

- 如想终止一个作业，可利用*scancel job_id*来取消
- job_list可以为以,分隔的作业ID
- 如：*scancel 7*

挂起排队中尚未运行的作业：scontrol hold job_list

- *scontrol hold job_list*（job_list可以为以,分隔的作业ID或jobname=作业名）命令可使得排队中尚未运行的作业（设置优先级为0）暂停被分配运行，被挂起的作业将不被执行，这样可以让其余作业优先得到资源运行
- 被挂起的作业在用*squeue*命令查询显示的时NODELIST(REASON)状态标志为JobHeldUser（被用户自己挂起）或JobHeldAdmin（被系统管理员挂起）
- 下面命令将挂起作业号为7的作业：
  scontrol hold 7

继续排队被挂起的尚未运行作业：scontrol release job_list

- 被挂起的作业可以利用*scontrol release job_list*来取消挂起，重新进入等待运行状态
- job_list可以为以,分隔的作业ID或jobname=作业名
  scontrol release 7

挂起已在运行的作业：scontrol suspend job_list

- *scontrol suspend job_list*命令可使得已运行的作业暂停运行，这样可以让其余作业优先得到资源运行
- 被挂起的作业在用qstat命令查询时显示的状态标志为H
- 下面命令将挂起作业号为7.admin0的作业：
  scontrol suspend 7

继续运行被暂停运行的作业：scontrol resume job_list

被暂停运行的作业可以利用*scontrol resume job_list*继续运行：

scontrol resume job_list 7

挂起、继续运行作业

作业状态	挂起	继续
尚未开始运行	*scontrol hold job_list*	*scontrol release job_list*
已开始运行	*scontrol suspend job_list*	*scontrol resume job_list*

重新运行作业：scontrol requeue job_list

- 利用*scontrol requeue job_list*重新使得运行中的、挂起的或停止的作业重新进入排队等待运行
- job_list可以为以,分隔的作业ID
  scontrol requeue 7

重新挂起作业：scontrol requeuehold job_list

- 利用*scontrol requeuehold job_list*重新使得运行中的、挂起的或停止的作业重新进入排队，并被挂起等待运行
- job_list可以为以,分隔的作业ID
- 之后可利用*scontrol release job_list*使其运行

scontrol requeuehold 7

最优先等待运行作业：scontrol top job_id

- 利用*scontrol top job_list*可以使得尚未开始运行的job_list作业排到用户自己排队作业的最前面，最优先运行
- job_list可以为以,分隔的作业ID
  scontrol top 7

等待某个作业运行完：scontrol wait_job job_id

利用*scontrol wait_job job_id*可以等待某个job_id结束后开始运行，一般用于脚本中

scontrol wait_job 7

更新作业信息：scontrol update SPECIFICATION

- 利用*scontrol update SPECIFICATION*可以更新作业、作业步等信息
- SPECIFICATION格式为*scontaol show job*显示出的
- 下面命令将更新作业号为7的作业信息:
  scontrol update JobId=7 JobName=NewJobName Partition=NewPartition NumTasks=10

显示激活的或已完成作业的记账（与机时费对应）信息：sacct I

主要参数：

- -b, –brief：显示简要信息，主要包含：作业号jobid、状态status和退出码exitcode
- -c, –completion：显示作业完成信息而非记账信息
- -e, –helpformat：显示当采用 –format指定格式化输出的可用格式
- -E end_time, –endtime=end_time：显示在end_time时间之前（不限作业状态）的作业。时间格式：
  - HH:MM[:SS] [AM|PM]
  - MMDD[YY] or MM/DD[/YY] or MM.DD[.YY]
  - MM/DD[/YY]-HH:MM[:SS]
  - YYYY-MM-DD[THH:MM[:SS]]
- -i, –nnodes=N：显示在特定节点数上运行的作业(N = min[-max)]
- -j job(.step), –jobs=job(.step)：限制特定作业号（步）的信息，作业号（步）可以以,分隔
- -l, –long：显示详细信息

显示激活的或已完成作业的记账（与机时费对应）信息：sacct II

- -n, –noheader：不显示信息头（显示出的信息的第一行，表示个列含义）
- -N node_list, –nodelist=node_list：显示运行在特定节点的作业记账信息
- –name=jobname_list：显示特定作业名的作业记账信息
- -o, –format：以特定格式显示作业记账信息，格式间采用,分隔，利用-e, –helpformat可以查看可用的格式。各项格式中
- -r, –partition：显示特定队列的作业记账信息。
- -R reason_list, –reason=reason_list：显示由于reason_list（以,分隔）原因没有被调度的作业记账信息。
- -s state_list, –state=state_list：显示state_list（以,分隔）状态的作业记账信息。
- -S, –starttime：显示特定时间之后开始运行的作业记账信息，时间格式参见前面-E参数。

服务质量QOS

对资源进行限制，比如限制用户的单个作业CPU核数、运行中作业总CPU核数、作业时长等，除了利用队列分区等外，还可以利用服务质量(Quality of Service, QOS)。利用Slurm提交作业时，可以为每个作业赋予一个QOS，与作业相关的QOS有三种途径：

- 作业调度优先级：Job Scheduling Priority
- 作业抢占：Job Preemption
- 作业限制：Job Limits

作业提交时，需要通过给*sbatch*、*salloc*和*srun*命令添加*--qos=*选项来设定需要的QOS

作业限制

- 每个QOS被赋予OverPartQOS一系列的限制，这些限制将应用于作业
- 这些限制反映了在Slurm数据库中定义的并在下面资源限制部分中描述的用户/帐户/集群/队列(user/account/cluster/partition)关联所施加的限制
- 当定义了QOS的限制时，它们将优先于关联的限制
- Slurm的层级限制按照下述顺序强制执行，作业QOS和队列QOS顺序如采用了QOS标记OverPartQOS参数则可逆
  1. 队列QOS限制
  2. 作业QOS限制
  3. 用户关联
  4. 帐户关联，升序排列
  5. Root/Cluster关联
  6. 队列限制
  7. 无

QOS优先级举例

若在层级中定义了多个限制，则在这个列表中的最先定义的限制有效。如下面例子：

- MaxJobs=20，且MaxSubmitJobs未在队列QOS中被定义
- 在作业QOS中没有任何限制
- MaxJobs=4和MaxSubmitJobs=50在用户关联中定义

上面限制实际生效的是MaxJobs=20和MaxSubmitJobs=50

QOS限制选项 I

- GrpTRESMins=：能从关联及其子项或QOS运行的过去、现在和未来作业可能使用的TRES分钟总数。当该限制到达时，所有依赖于该限制的作业将被杀掉，且没有新的作业被允许运行。该用法被衰减（以PriorityDecayHalfLife设定的衰减速率）。该选项可以被重置（取决于 PriorityUsageResetPeriod 选项）以便允许作业重新基于这些关联树或QOS运行。QOS，如没有NoDecay被设置，那么GrpTRESMins将不被衰减。这仅在启用优先级多因子插件时生效。
- GrpTRESRunMins=：用于限制所有同一个关联和其子项或QOS组合的总TRES分钟数。考虑到运行作业的时间限制并使用它，如果达到限制，在其它作业完成之前不会启动新作业，以允许时间释放作业。
- GrpTRES=：在给定的任意时间，能从关联及其子项或QOS运行作业的TRES总数。如达到该限制，新作业将处于排队状态，仅当资源被从该组释放时才能开始运行。
- GrpJobs=：在给定的任意时间，能从关联及其子项或QOS运行作业总数。如达到该限制，新作业将处于排队状态，仅当之前作业结束时才能开始运行。

QOS限制选项 II

- GrpJobsAccrue=：在给定的任意时间，能从关联及其子项QOS中获得年龄优先级的排队中作业的总数。如达到此限制，新作业将排队，但在此组中等待的先前作业删除之前不会累积年龄优先级。此限制并不决定作业是否可以运行，它只限制了优先级的年龄因素。当在QOS上设置时，此限制仅适用于作业QOS，而不适用于队列QOS。
- GrpSubmitJobs=：在给定的任意时间，能从关联及其子项QOS中提交到系统中的作业总数。如达到该限制，提交新作业将被拒绝，仅当之前作业结束时才能提交。
- GrpWall：能从关联及其子项或QOS运行作业的墙上时钟最大值。当该限制到达时，在该QOS或关联的未来作业将处于排队状态，直到期能在该限制内运行。该使用量会被衰减（以PriorityDecayHalfLife选项规定的比率）。它也可以被重置（取决于PriorityUsageResetPeriod选项），以便允许作业基于该关联数或QOS运行。具有NoDecay值的QOS不会衰减GrpWall 。
- MaxTRESMinsPerJob=：作业占用TRES分钟的限制。如达到该限制，该作业如没有运行在Safe模式下将被杀掉，否则该作业将排队直到被给予足够的时间以完成该作业。

QOS限制选项 III

- MaxTRESPerJob=：每个作业能从关联/QOST获取的TRES最大数。
- MaxTRESPerNode=：每个作业能从关联/QOST获取的每个节点的TRES最大数。
- MaxWallDurationPerJob=：每个作业能从关联/QOST获取的最大墙上时钟时长。如达到该限制，作业在提交时将被拒绝。
- MinPrioThreshold=：从关联/QOST获取的用于预留资源的最小优先级。可用于覆盖bf_min_prio_reserve选项设置。

支持的关联指定调度策略

- Fairshare=：用于决定公平共享优先级的整数。本质上，这是对上述系统针对该关联和其子项的请求总数。也可以使用字符串” parent”，当被用户使用时，意味着针对公平共享采用其父关联。如在该账户设置Fairshare=parent，该账户的子成员将被有效地利用它们的第一个不是Fairshare=parent的父母重新支付公平共享计算。限制保持不变，仅影响其公平共享值。
- MaxJobs=：在给定的任意时间，针对该关联能同时运行的作业总数。如该达到限制，新作业将处于排队状态，只有当该关联的作业有结束时才能运行。
- MaxJobsAccrue=：在给定的任意时间，能从关联允许的排队中作业累计年龄优先级的最大数。当该达到限制，新作业将处于排队中状态，但不累计年龄优先级，直到有关联的作业从排队状态有退出排队。该限制不决定作业是否能运行，它只限制优先级的年龄因子。
- MaxSubmitJobs=：在给定的任意时间，能从该关联提交到系统中的作业最大数。如达到该限制，新提交申请将被拒绝，直到存在该关联的作业退出。
- QOS=：以逗号（,）分隔的能运行的QOS列表。

支持的QOS特定限制

- MaxJobsAccruePerAccount=：在任意时间，一个账户（或子账户）能从关联允许的排队中作业累计年龄优先级的最大数。该限制不决定作业是否能运行，它只限制优先级的年龄因子。
- MaxJobsAccruePerUser=：在任意时间，一个用户能从关联允许的排队中作业累计年龄优先级的最大数。该限制不决定作业是否能运行，它只限制优先级的年龄因子。
- MaxJobsPerAccount=：一个账户（或子账户）能允许的最大同时运行作业数。
- MaxJobsPerUser=每个用户能同时运行的最大作业数。
- MaxSubmitJobsPerAccount=每个账户（或子账户）能同时运行和排队等待运行的最大作业数。
- MaxSubmitJobsPerUser=每个用户能同时运行和排队等待运行的最大作业数。
- MaxTRESPerAccount=每个账户能同时分的配最大TRES数。
- MaxTRESPerUser=每个用户能同时分配的最大TRES数。
- MinTRESPerJob=每个作业能申请的最小TRES尺寸。

查看自己可用QOS

- 以默认格式显示自己可用QOS：*sacctmgr show qos* 或 *sacctmgr list qos*
- 以特定格式显示自己可用QOS：*sacctmgr show qos format=name,priority* 或 *sacctmgr list qos format=name,priority*
- 查看队列对应的QOS：*scontrol show partition 队列名* 查看对应QOS参数
- 查看自己关联的账户：
  *sacctmgr show assoc*或*sacctmgr show assoc format=cluster,user,qos*等

提交作业时添加QOS参数

如队列等有QOS限制，则提交作业时需要利用*--qos=QOS*名等添加对应的参数
如：*scontrol show parti GPU-8A100*显示：

```
PartitionName=GPU-8A100
   AllowGroups=ALL AllowAccounts=ALL AllowQos=gpu_8a100
   AllocNodes=ALL Default=NO QoS=gpu_8a100
   DefaultTime=NONE DisableRootJobs=YES ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=8 MaxTime=5-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=gnode[12,14,17-19,21]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=384 TotalNodes=6 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
   TRES=cpu=384,mem=6000000M,node=6,billing=384,gres/gpu=48
```

则提交作业时需要添加*--qos=gpu_8a100*
更多QOS说明参见：
http://hmli.ustc.edu.cn/doc/linux/slurm-install/slurm-install.html#qos

联系信息

- 中国科大超级计算中心：
  - 电话：0551-63602248、63601541
  - 邮箱：sccadmin@ustc.edu.cn
  - 主页：http://scc.ustc.edu.cn 大量的文档
  - 办公室：中国科大东区新图书馆一楼东侧126、124室
- 李会民：
  - 电话：0551-63600316
  - 邮箱：hmli@ustc.edu.cn
  - 主页：http://hmli.ustc.edu.cn
  - 办公室：中国科大东区新科研楼A座三楼301室