IPRC-GPU集群-SLURM简要说明

SLURM 快速上手

在`login`节点上，拷贝以及创建好的 SLURM 模板文件，`cp /tools/template.slurm ~/demo.sh`

It just works 配置

将文件的最后`job step`部分，`EDIT HERE`改成你想进行的操作，例如读取主机名

```
...

#- Job step
sleep 10s
hostname

...
```

执行`sbatch demo.sh`会自动将`demo.sh`发送到默认队列上，等待分配资源执行。此时执行`squeue`可以看到对应的任务状态：

```
$ sbatch demo.sh
Submitted batch job 88

$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                88 nv-gpu        test    xxxxx  R   00:00:01      1 compute-t00
```

样例`demo.sh`脚本的执行结果默认是`ret-$ID.out`与`ret-$ID.err`，所有的标准输入输出流内容都会被放在`sbtach`执行的目录。此时打开`ret-88.out`与`ret-88.err`即可查看运行结果。

`slurm-$ID.out`

交互式环境

当程序运行的时候，例如运行在`compute-t00`节点上，此时用户被允许`ssh`到对应的机器上，注意，SLURM 上线后，用户无法登录没有正在运行自己提交的任务的节点。

即，如果自己在相应节点有一个任务，则用户**可以**通过`ssh`的方式进入该节点。并且只会获取和自己任务相关的那一部分资源。

若超出任务最大时间，用户则会自动登出计算节点，所有未能按时结束的进程会自动终止。

若在目标节点上有两个或以上任务，使用`ssh`登录时会取得最近一次提交的任务所相关的资源。

取消任务

使用 `scancel` 可以取消正在运行的任务

```
$ scancel 88   #任务编号
```

当`scancel` 后 `STATE`长时间处于 `CG` 或 `completing`的时候，意味着程序已经无法响应 `kill -9`信号，此时多半是因为 IO 或者外部 GPU 资源问题卡死，这时候请及时联系管理员重启相应服务器，管理员联系名单见附录。

SLURM 简单介绍

SLURM 是一个开源的 cluster workload manager。

SLURM 有一个中心化的管理者，`slurmctd`，用于监视资源的使用。同时也能有一个备用的 manager 来防止主 manager 失效。

每个计算节点（node）都有一个 `slurmd` daemon，用于等待任务下发，执行任务以及返回结果。同时有一个可选的 `slurmdbd` (slurm database daemon) 用于记录 accounting 信息，`slurmrestd` (slurm REST API daemon) 用于提供 REST api 接口。

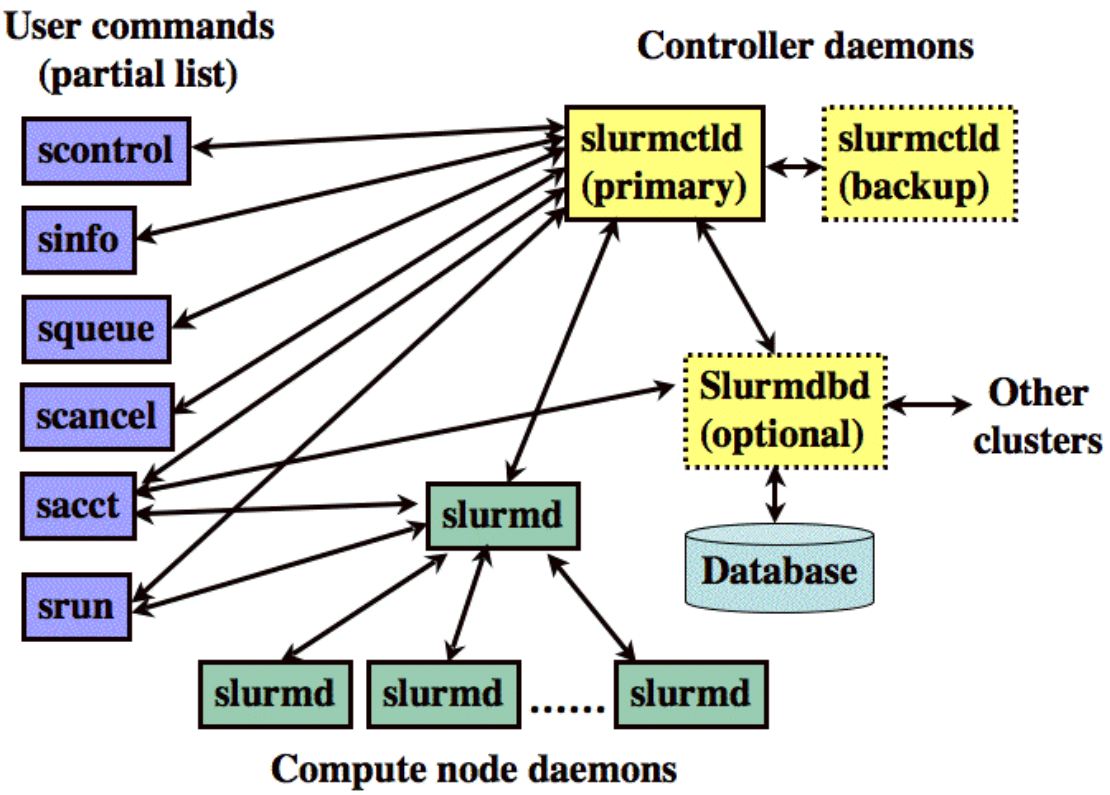
用户使用 `srun` 来初始化 job，`scancel` 来结束排队中或者运行中的 job。`sinfo` 用于汇报系统的状态，`squeue` 用于查询 jobs 排队状态，`sacct` 用于得到正在运行或已经结束的 jobs 和 jobs steps 信息。 `sview` 用于图形化显示系统和 jobs 状态（包含网络拓扑）。

对于管理员还有 `scontrol` 用于监控和修改 cluster 的配置和 status。`sacctmgr` 用于管理数据库信息。

基础概念

SLURM 通过 controller nodes 控制若干个 compute nodes，按照 QOS (Quality of Service) 策略对任务进行分发。

compute nodes (下称 nodes) 被分配到一个或多个 partitions 中，每个 partition 通常用于不同的任务类型，有自己的规则。



分区（partition）

分区是用来划分节点用途的，是一系列节点的集合，分区本身只是一个节点划分策略（可以按照硬件属性 or 特性等来划分），当用户提交任务的时候，如果没有指定分区，将会被提交到有管理员指定的默认分区。

> 同一个节点可以隶属于不同的分区，依靠分区权重来决定分配策略

`sinfo`可以查看系统存在什么队列、节点及其状态。

```
$ sinfo -l
```

当前集群分区列表（以实际squeue看到的为准）

分区/队列	最长时限	适用的QOS
cpu	12天	cpu-debug,cpu-normal,cpu-long
cpu-fat	12天	cpu-debug,cpu-normal,cpu-long
nv-gpu*	12天	gpu-trial, gpu-debug, gpu-short,gpu-normal,gpu-long,gpu-longlong
nv-gpu-hw	12天	gpu-trial, gpu-debug, gpu-short,gpu-normal,gpu-long,gpu-longlong

`nv-gpu`只允许提交 **1, 2, 4** 卡任务，`nv-gpu-hw`只允许提交 **4, 8** 卡任务。

QOS (Quality of Service)

QOS 是用于刻画 job 的属性的，对于不同的 QOS 有不同的优先级。

用户可以自行设定自己的任务依照何种 QOS 来调度，以此来平衡等待时间和资源占用。

使用 `saccmgr` 可以查询系统存在的 QOS 策略：

```
$ sacctmgr show qos
$ sacctmgr show qos format=name,priority,maxtrespu,maxwall,maxjobspu,maxsubmitpu
```

当前集群 QOS 列表（以实际为准）

QOS 名称	优先级	资源限制	最大运行时间	最大作业数	最大提交数
gpu-trial	高	同时不得超过32张GPU	20分钟	3	6
gpu-debug	高	同时不得超过32张GPU	1.5小时	3	6
gpu-short	中	同时不得超过32张GPU	8小时	8	12
gpu-normal	低	同时不得超过32张GPU	30小时	8	12
gpu-long	低	同时不得超过16张GPU	4天	3	4
gpu-longlong	低	同时不得超过8张GPU	10天	1	4

队列

查看队列中的自己的作业信息

```
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
...
```

其中 NODELIST(REASON) 一栏 包含非常有用的信息，在作业未运行时，它会显示未运行的原因；当作业在运行时，它会显示作业是在哪个节点运行的。常见的选项如下：

- `InvalidQOS`：作业QOS无效，建议取消该作业重新采用正确QoS提交
- `PartitionTimeLimit`：用户预估的时间，超过当前分区限制，建议取消该作业更改分区或者更改预估时间提交
- `Priority`：作业所需的队列存在高等级作业或预留
- `Resources`：作业将要等待所需要的资源满足后才运行
- `QOSMaxGRESPerUser`：用户使用的资源超过当前的QOS限制，需等待现有任务结束后执行
- `QOSMaxJobsPerUserLimit`：用户正在执行的任务超过当前的QOS限制，需等待现有任务结束后执行

如果是错误，可以 `scancel [JobID]` 后修改再提交，也可以使用 `scontrol` 修改；
如果只是挂起等待排队，则无需操作。

注意事项

- QOS 和 Partition 同时限定了任务最大运行时间的时候，以最严格的限制为准
- 最长时限是一个强制时限，你的任务在提交时，需要指定一个预估的时间，该时间不可超过提交分区与指定QOS的限制
- `#SBATCH -t` 提交的预估时间，是一个 `wall time`，超时后，系统会自动将任务终止
- **请勿在执行脚本，“.bashrc”，执行脚本调用的其他脚本或者代码的任何位置修改 CUDA_VISIBLE_DEVICES 变量**

Best Pratice 配置

说明

> 【ALERT】SLURM 实际上相当于用 `su $USER` 的形式在执行的 node 上面，切换到 **提交任务** 的用户的账户来执行对应的脚本，因此会读取该用户的 `$HOME/.bashrc` 而非加载空的 `.bashrc`。所以请注意 env 的环境配置会不会产生冲突。

在 cluster + module 挂载的机制下，可以认为每台机子的运行环境都是一样，我们可以通过 module 来加载对应的配置和环境，并利用 cluster 同步来做数据共享。以下是一个完整的 `bash script` 的流程。

- 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。
  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。
  - 在任何非注释及空白之后的“#SBATCH”将不再作为Slurm参数处理。
  - 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。此处已经指定为 ret-%j.out 与 ret-%j.err

代码

```
#!/bin/bash

#- Job parameters

#SBATCH -J test              # The job name
#SBATCH -o ret-%j.out        # Write the standard output to file named 'ret-
<job_number>.out'
#SBATCH -e ret-%j.err        # Write the standard error to file named 'ret-
<job_number>.err'

#- Needed resources

#SBATCH -p nv-gpu,nv-gpu-hw          # Submit to GPU queue
#SBATCH -t 0-12:00:00                # Run for a maximum time of 0 days, 12
hours, 00 mins, 00 secs
#SBATCH --nodes=1                    # Request N nodes
#SBATCH --gres=gpu:4                 # Request M GPU per node
```

```
#SBATCH --gres-flags=enforce-binding # Better CPU-GPU Affinity
#SBATCH --constraint="Volta|RTX8000" # Request GPU Type

###
### The system will alloc 8 cores per gpu by default.
### If you need more or less, use following:
### #SBATCH --cpus-per-task=K        # Request K cores
###

#SBATCH --qos=gpu-normal             # Request QOS Type

#- Operstions
echo "Job start at $(date "+%Y-%m-%d %H:%M:%S")"
echo "Job run at:"
echo "$(hostnamectl)"

#- Load environments
source /tools/module_env.sh
module list                          # list modules loaded by default

##- tools
module load cmake/3.15.7
module load git/2.17.1
module load vim/8.1.2424

##- language
module load python3/3.6.8

##- cuda
module load cuda-cudnn/11.0-8.0.5

##- virtualenv
# source xxxxx/activate

#- Log information

module list                         # list modules loaded by default
echo $(module list)                 # list modules loaded
echo $(which gcc)
echo $(which python)
echo $(which python3)
nvidia-smi --format=csv --query-gpu=name,driver_version,power.limit
echo "Use GPU ${CUDA_VISIBLE_DEVICES}$"
#- Warning! Please not change your CUDA_VISIBLE_DEVICES
#- in `.bashrc`, `env.sh`, or your job script

#- Job step
[EDIT HERE(TODO)]

#- End
echo "Job end at $(date "+%Y-%m-%d %H:%M:%S")"
```

此文件在集群内通过`cp /tools/template.slurm ~/`可以获取。

- 修改`-J [job_name]`，以后可以在队列中快速识别任务
- 修改`-p [partitiion_name]`，选择需要执行任务的分区
- 修改`-t [dd]-[hh]:[mm]:[ss]`，设定最长任务执行时间

- 修改 `--gres=gpu:[m]`，设定每个节点需要使用几块GPU资源
- 修改 `--constraint=[feature]`，设定需要的节点的特征
- 修改 `--qos=[qos_name]`，设定任务的服务优先级
- 修改 `[EDIT HERE]` 部分来设置自己的执行流，上面的各种环境配置如果自己的 `.bashrc` 里面没有配置的话可以保留。

> 适当的输出一些环境信息，可以有助于调试和追踪 bug。

具体的服务器列表和 SLURM 多节点运行等请参照附录。

常见场景

```
# 查看详细队列信息：scontrol show partition
## scontrol show partition 显示全部队列信息
## scontrol show partition PartitionName 或 scontrol show partition=PartitionName 显示队列名为 PartitionName 的队列信息
$ scontrol show partition nv-gpu

# 查看详细节点信息：scontrol show node
## 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NODENAME 的节点信息
$ scontrol show compute-t00

# 提交作业的命令主要有salloc、sbatch与srun，其多数参数、输入输出变量等都是一样的
## sbatch template.slurm 提交任务脚本
$ sbatch template.slurm

# 查看详细作业信息：scontrol show job
## 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
$ scontrol show job 2020

# 查看服务质量(QOS)
## 服务质量(Quality Of Service•QOS)，或者理解为资源限制或者优先级，只有达到QOS要求时作业才能运行，QOS将在以下三个方面影响作业运行：
## • 作业调度优先级
## • 作业抢占
## • 作业限制
## 可以用sacctmgr show|list qos查看
$ sacctmgr show qos

# 更新作业信息：scontrol update SPECIFICATION
## 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，如下面命令将更新作业号为88的作业名为NewJobName：
$ # scontrol update jobid=[jobid] [attributes=value]
$ scontrol update JobId=88 JobName=NewJobName
```

工具

系统中已经添加了更加方便的 slurm 查询工具

```
$ module load slurm-tools
$ module help slurm-tools
$ slurm-gpu-info
$ slurm-gpu-queue
$ slurm-gpu-allocated
```

注意事项

- 变更的时候仍然不能超过系统规定的上限。变更成功后，作业的优先级可能需要重新 来计算。
- 在GPU计算节点，可以执行slurm-gpu-allocated查看自己分配的GPU资源编号
- 当任务已经开始运行时，一般不可以再变更申请资源，分区等参数。特别地，如果发现 自己低估了任务运行时间，**用户不能使用 scontrol 命令延长任务最大时间**。但是可以根据需求减少任务的最大时间。

附录：当前集群的其他信息

服务器列表

具体启用的节点以sinfo查看到为准

域名	GPU	CPU+内存	筛选特征
gpu-a[00-07]	2*A100_40G	24C48T_2.8G+128G	Rome,Ampere,A100,HCGR
gpu-a[08-13]	8*A100_40G	128C128T_2.2G+1TB	IB,Rome,Ampere,A100,HCGR
gpu-t[00-01]	4*RTX8000_48G	36C72T_2.6G+512G	IB,CLSP,Turing,RTX8000,HCGR
gpu-t[04-11]	8*RTX8000_48G	36C72T_2.6G+512G	IB,CLSP,Turing,RTX8000
gpu-t[12-13]	4*T4_16G	40C80T_2.1G+192G	IB,CLSP,Turing,T4,HCGR
gpu-v[00-07]	4*V100S_32G	36C72T_2.6G+512G	IB,CLSP,Volta,V100S,HCGR
gpu-v[10-19]	8*V100_32G	32C64T_2.6G+512G	IB,SSP,Volta,V100
gpu-v20	8*V100_32G	40C80T_2.2G+512G	IB,BEP,Volta,V100,NVLink

注释：

缩写	全称	解释
HCGR	High CPU GPU Ratio	平均每颗CPU可以拥有16线程及以上
IB	InfiniBand	InfiniBand链接
BEP	Broadwell-EP	英特尔 Xeon E5 V4
SSP	Skylake-SP	英特尔 Xeon 可拓展 Gen1
CLSP	CascadeLake-SP	英特尔 Xeon 可拓展 Gen2，支持 Intel® DL Boost

目前**HCGR**机器，最大支持4张GPU，预计2021年3月上线128核8GPU的服务器

提交任务时的其他注意事项

- 一般情况下，请勿自己指定GPU任务的CPU资源，系统会默认每张GPU配置8或16个CPU资源
- nv-gpu 队列只允许提交 1,2,4卡任务，nv-gpu-hw队列只允许提交4,8卡任务
- CPU资源是独占的，所以如果提交需求CPU资源较多的GPU任务，请加上**HCGR**的限制。例如提交了64核4GPU的任务后，未加上**HCGR**限制，如果被分配到64线程8卡节点，即使有4张GPU未被使用，但是由于剩余CPU资源不足，仍然不会向该计算节点分配其他任务，进而造成资源浪费
- 如果用V100或者V100S都可以，可以将限制改为**Volta**

GPU 的兼容性

虽然 GPU 通过 ptx 可以达到一定程度的前后向兼容，但是为了保证程序的高效，推荐使用如下配置

GPU架构	**计算能力分级**	**推荐的最低CUDA**	**推荐最低的cuDNN**
GV100核心（V100）	sm_70	>=9.0	>=7.0
TU102核心（RTX8000），TU104核心（T4）	sm_75	>=10.0	>=7.3
GA100核心（A100）	sm_80	>=11.0	>=8.0
GA102核心（RTX3090，A40）	sm_86	>=11.1	>=8.0.4

建议使用最新的CUDA和深度学习框架，以确保程序可以在任意节点执行，且得到较高的性能。

附录：SLURM 常用命令

sbatch 命令： https://slurm.schedmd.com/sbatch.html

squeue 命令：https://slurm.schedmd.com/squeue.html

scancel 命令：https://slurm.schedmd.com/scancel.html

sinfo 命令：https://slurm.schedmd.com/sinfo.html

scontrol 命令：https://slurm.schedmd.com/scontrol.html

> 不常用

salloc 命令：https://slurm.schedmd.com/salloc.html

sattach 命令：https://slurm.schedmd.com/sattach.html