

# 1. 创建环境

- 根据 mmclassification 的环境要求，需要用 anaconda、cuda、gcc 等基础环境模块。在 N30 分区可以使用 module avail 命令可以使用模块信息。例如：

```
1 $ module avail
2 -----
3 dot          module-git  module-info modules      null          use.own
4
5 -----
6 alphafold/2.0.1                                fftw/3.3.9-mpi-float
   lapack/3.10.1                                openmpi/4.1.1-ucx1.9
7 alphafold/2.1.1                                gcc/11.2
   libevent/2.1.12
   openmpi/4.1.1-ucx1.9-cuda11.2
8 alphafold/parafold-2.1.1                       gcc/5.4
   localcolabfold/1.4                           p7zip/16.02
9 anaconda/2020.11                             gcc/6.3
   namd2/2.14-verbs-linux-x86_64-gcc-smp-CUDA   p7zip/21.02
10 anaconda/2021.05                             gcc/7.3
   nccl/2.11.4-1_cuda11.1                      plumed/2.7.2
11 anaconda/2022.10                             gcc/8.3
   nccl/2.11.4-1_cuda11.2                      pmix/3.2.2
12 arias/1.36.0                                 gcc/9.3
   nccl/2.11.4-1_cuda11.4
   pnetcdf/1.12.2/openmpi-gcc9.3
13 blas/3.10.0                                 go/1.18.2
   netcdf-c/4.8.1/openmpi-gcc9.3
   qe/6.8-nvhpc21.9-openmpi4.0.5-cuda11.4-ucx1.9
14 cmake/3.22.0                                gromacs/2021.2-nompi
   nvhpc/21.5                                  rar/611
15 cp2k/9.1-openmpi-414-gcc-8.3-cuda-11.3       gromacs/2021.2-parallel
   nvhpc/21.9                                  singularity/2.6.0
16 cuda/11.1                                    gromacs/2021.2-plumed-nompi
   nvhpc/nvhpc-byo-compiler/21.5              singularity/3.10.0
17 cuda/11.2                                    gromacs/2021.5_dev_fep_mpi
   nvhpc/nvhpc-byo-compiler/21.9              singularity/3.9.9
18 cuda/11.3                                    gromacs/2022-nompi
   nvhpc/nvhpc-nompi/21.5                     SPONGE/1.2.6
```

19	cuda/11.4	nvhpc/nvhpc-nompi/21.9	hdf5/1.12.1	tensorboard/2.3.0
20	cuda/11.6	nvhpc-byo-compiler/21.5	hwloc/2.1.0	ucx/1.8
21	cuda/11.7	nvhpc-byo-compiler/21.9	intel/parallelstudio/2017.1.5	ucx/1.9
22	cudnn/8.2.1_cuda11.x	nvhpc-nompi/21.5	intel/parallelstudio/2019.3.0	ucx/1.9_cuda11.2
23	cudnn/8.2.4_cuda11.4	nvhpc-nompi/21.9	intel/parallelstudio/2021.1.1	ucx/1.9_cuda11.4
24	cudnn/8.5.0_cuda11.x	oneFlow/0.8.0	jupyter/lab	zlib/1.2.11
25	dos2unix/6.0.3	openmpi/4.0.5_nvhpc21.9_ucx1.9_cuda11.4	jupyter/notebook	
26	fftw/3.3.9	openmpi/4.1.1	lammps/27May2021	

- 1. 加载 anaconda , 创建一个 python 3.8 的环境。

```

1 # 加载 anaconda/2021.05
2 module load anaconda/2021.05
3
4 # 创建 python=3.8 的环境
5 conda create --name openmmlab_mmclassification python=3.8
6
7 # 激活环境
8 source activate openmmlab_mmclassification

```

- 2. 安装 torch, torch 参考[官网](#)需求。注意在 RTX3090 的GPU上, cuda 版本需要  $\geq 11.1$  。如下安装的 torch 是 1.10.0+cu111 。使用 pip 安装的torch 不包括 cuda, 所以需要使用 module 加载 cuda/11.1 模块。

```

1 # 加载 cuda/11.1
2 module load cuda/11.1
3
4 # 安装 torch
5 pip install torch==1.10.0+cu111 torchvision==0.11.0+cu111 torchaudio==0.10.0 -f
https://download.pytorch.org/whl/torch_stable.html

```

- 3. 安装 mmdcv-full 模块, mmdcv-full 模块安装时候需要注意 torch 和 cuda 版本。参考[这里](#)。

```

1 pip install mmdcv-full==1.7.0 -f
https://download.openmmlab.com/mmdcv/dist/cu111/torch1.10/index.html

```

- 4. 安装 openmmlab/miclassification 模块，建议通过下载编译的方式进行安装；安装该模块需要 gcc  $\geq$  5，使用 module 加载一个 gcc，例如 module load gcc/7.3。

```
1 # 加载 gcc/7.3 模块
2 module load gcc/7.3
3
4 # git 下载 mmclassification 代码
5 git clone https://github.com/open-mmlab/miclassification.git
6
7 # 编译安装
8 cd mmclassification
9 pip install -e .
```

- 4. 总结环境信息，可以使用 module list 查看当前环境中加载的依赖模块，如下：

```
1 $ module list
2 Currently Loaded Modulefiles:
3 1) anaconda/2021.05 2) cuda/11.1 3) gcc/7.3
```

- 5. 准备 shell 脚本，将环境信息预先保存在脚本中。

```
1 #!/bin/bash
2 # 加载模块
3 module load anaconda/2021.05
4 module load cuda/11.1
5 module load gcc/7.3
6
7 # 激活环境
8 source activate openmmlab_miclassification
```

## 2. 数据集

- flower 数据集包含 5 种类别的花卉图像:雏菊 daisy 588张，蒲公英 dandelion 556张，玫瑰 rose 583张，向日葵 sunflower 536张，郁金香 tulip 585张。
- 数据集下载链接:
  - 国际网:[https://www.dropbox.com/s/snom6v4zfkY0flx/flower\\_dataset.zip?dl=0](https://www.dropbox.com/s/snom6v4zfkY0flx/flower_dataset.zip?dl=0)
  - 国内网:[https://pan.baidu.com/s/1RJmAoxCD\\_aNPYTRX6w97xQ](https://pan.baidu.com/s/1RJmAoxCD_aNPYTRX6w97xQ) 提取码: 9x5u

## 2.1 划分数据集

- 将数据集按照 8:2 的比例划分成训练和验证子数据集，并将数据集整理成 ImageNet 的格式
- 将训练子集和验证子集放到 train 和 val 文件夹下。

```
1 flower_dataset
2 |--- classes.txt
3 |--- train.txt
4 |--- val.txt
5 |   |--- train
6 |   |   |--- daisy
7 |   |   |   |--- NAME1.jpg
8 |   |   |   |--- NAME2.jpg
9 |   |   |   |--- ...
10 |   |   |--- dandelion
11 |   |   |   |--- NAME1.jpg
12 |   |   |   |--- NAME2.jpg
13 |   |   |   |--- ...
14 |   |   |--- rose
15 |   |   |   |--- NAME1.jpg
16 |   |   |   |--- NAME2.jpg
17 |   |   |   |--- ...
18 |   |   |--- sunflower
19 |   |   |   |--- NAME1.jpg
20 |   |   |   |--- NAME2.jpg
21 |   |   |   |--- ...
22 |   |   |--- tulip
23 |   |   |   |--- NAME1.jpg
24 |   |   |   |--- NAME2.jpg
25 |   |   |   |--- ...
26 |   |--- val
27 |   |   |--- daisy
28 |   |   |   |--- NAME1.jpg
29 |   |   |   |--- NAME2.jpg
30 |   |   |   |--- ...
31 |   |   |--- dandelion
32 |   |   |   |--- NAME1.jpg
33 |   |   |   |--- NAME2.jpg
34 |   |   |   |--- ...
```

```

35 |      |      |--- rose
36 |      |      |      |--- NAME1.jpg
37 |      |      |      |--- NAME2.jpg
38 |      |      |      |--- ...
39 |      |      |--- sunflower
40 |      |      |      |--- NAME1.jpg
41 |      |      |      |--- NAME2.jpg
42 |      |      |      |--- ...
43 |      |      |--- tulip
44 |      |      |      |--- NAME1.jpg
45 |      |      |      |--- NAME2.jpg
46 |      |      |      |--- ...

```

- 创建并编辑标注文件将所有类别的名称写到 `classes.txt` 中，每行代表一个类别。

```

1 tulip
2 dandelion
3 daisy
4 sunflower
5 rose

```

- 生成训练(可选)和验证子集标注列表 `train.txt` 和 `val.txt`，每行应包含一个文件名和其对应的标签。如下，可将处理好的数据集迁移到 `mmclassification/data` 文件夹下。

```

1 ...
2 daisy/NAME**.jpg 0
3 daisy/NAME**.jpg 0
4 ...
5 dandelion/NAME**.jpg 1
6 dandelion/NAME**.jpg 1
7 ...
8 rose/NAME**.jpg 2
9 rose/NAME**.jpg 2
10 ...
11 sunflower/NAME**.jpg 3
12 sunflower/NAME**.jpg 3
13 ...
14 tulip/NAME**.jpg 4
15 tulip/NAME**.jpg 4

```

- 数据集划分代码 split\_data.py 如下，执行：

```
1 python split_data.py [源数据集路径] [目标数据集路径]
```

```
1 import os
2 import sys
3 import shutil
4 import numpy as np
5
6
7 def load_data(data_path):
8     count = 0
9     data = {}
10    for dir_name in os.listdir(data_path):
11        dir_path = os.path.join(data_path, dir_name)
12        if not os.path.isdir(dir_path):
13            continue
14
15        data[dir_name] = []
16        for file_name in os.listdir(dir_path):
17            file_path = os.path.join(dir_path, file_name)
18            if not os.path.isfile(file_path):
19                continue
20            data[dir_name].append(file_path)
21
22        count += len(data[dir_name])
23        print("{} : {}".format(dir_name, len(data[dir_name])))
24
25    print("total of image : {}".format(count))
26    return data
27
28
29 def copy_dataset(src_img_list, data_index, target_path):
30     target_img_list = []
31     for index in data_index:
32         src_img = src_img_list[index]
```

```

33     img_name = os.path.split(src_img)[-1]
34
35     shutil.copy(src_img, target_path)
36     target_img_list.append(os.path.join(target_path, img_name))
37     return target_img_list
38
39
40 def write_file(data, file_name):
41     if isinstance(data, dict):
42         write_data = []
43         for lab, img_list in data.items():
44             for img in img_list:
45                 write_data.append("{} {}".format(img, lab))
46     else:
47         write_data = data
48
49     with open(file_name, "w") as f:
50         for line in write_data:
51             f.write(line + "\n")
52
53     print("{} write over!".format(file_name))
54
55
56 def split_data(src_data_path, target_data_path, train_rate=0.8):
57     src_data_dict = load_data(src_data_path)
58
59     classes = []
60     train_dataset, val_dataset = {}, {}
61     train_count, val_count = 0, 0
62     for i, (cls_name, img_list) in enumerate(src_data_dict.items()):
63         img_data_size = len(img_list)
64         random_index = np.random.choice(img_data_size, img_data_size,
65 replace=False)
66
67         train_data_size = int(img_data_size * train_rate)
68         train_data_index = random_index[:train_data_size]
69         val_data_index = random_index[train_data_size:]
70
71         train_data_path = os.path.join(target_data_path, "train", cls_name)
72         val_data_path = os.path.join(target_data_path, "val", cls_name)

```

```

72         os.makedirs(train_data_path, exist_ok=True)
73         os.makedirs(val_data_path, exist_ok=True)
74
75         classes.append(cls_name)
76         train_dataset[i] = copy_dataset(img_list, train_data_index,
train_data_path)
77         val_dataset[i] = copy_dataset(img_list, val_data_index, val_data_path)
78
79         print("target {} train:{}, val:{}".format(cls_name,
len(train_dataset[i]), len(val_dataset[i])))
80         train_count += len(train_dataset[i])
81         val_count += len(val_dataset[i])
82
83         print("train size:{}, val size:{}, total:{}".format(train_count, val_count,
train_count + val_count))
84
85         write_file(classes, os.path.join(target_data_path, "classes.txt"))
86         write_file(train_dataset, os.path.join(target_data_path, "train.txt"))
87         write_file(val_dataset, os.path.join(target_data_path, "val.txt"))
88
89
90 def main():
91     src_data_path = sys.argv[1]
92     target_data_path = sys.argv[2]
93     split_data(src_data_path, target_data_path, train_rate=0.8)
94
95
96 if __name__ == '__main__':
97     main()

```

### 3. MMCIs 配置文件

- 构建配置文件可以使用继承机制，从 `configs/__base__` 中继承 ImageNet 预训练的任何模型，ImageNet 的数据集配置，学习率策略等。

#### 3.1 模型配置文件



- 可以使用任何模型，这里以 `resnet` 为例进行介绍。
- 首先在 `/configs/resnet` 下创建 `resnet18_b16_flower.py` 文件。
- 为了适配数据集 `flower` 这个 5 分类数据集，需要修改配置文件中模型对应的 `head` 和 `num_classes`。预训练模型的权重，除了最后一层线性层外，其他的部分会复用。

```
1 _base_ = ['../_base_/resnet18.py']
2 model = dict(
3     head=dict(
4         num_classes=5,
5         topk = (1, )
6     ))
```

## 3.2 数据配置

- 同样在 `resnet18_b16_flower.py` 文件中，继承 `ImageNet` 的数据配置，然后根据 `flower` 数据集进行修改。

```
1 _base_ = ['../_base_/models/resnet18.py', '../_base_/datasets/imagenet_bs32.py']
2
3 data = dict(
4     # 根据实验环境调整每个 batch_size 和 workers 数量
5     samples_per_gpu = 32,
6     workers_per_gpu=2,
7     # 指定训练集路径
8     train = dict(
9         data_prefix = 'data/flower_dataset/train',
10        ann_file = 'data/flower_dataset/train.txt',
11        classes = 'data/flower_dataset/classes.txt'
12    ),
13    # 指定验证集路径
14    val = dict(
15        data_prefix = 'data/flower_dataset/val',
16        ann_file = 'data/flower_dataset/val.txt',
17        classes = 'data/flower_dataset/classes.txt'
18    ),
19 )
20
21 # 定义评估方法
22 evaluation = dict(metric_options={'topk': (1, )})
```

### 3.3 学习率

- 模型微调的策略与从头开始训练的策略差别很大。微调一版会要求更小的学习率和更少的训练周期。依旧是在 `resnet18_b16_flower.py` 文件中进行修改。

```
1 # 优化器
2 optimizer = dict(type='SGD', lr=0.001, momentum=0.9, weight_decay=0.0001)
3 optimizer_config = dict(grad_clip=None)
4 # 学习率策略
5 lr_config = dict(
6     policy='step',
7     step=[1])
8 runner = dict(type='EpochBasedRunner', max_epochs=2)
```

### 3.4 加载预训练模型

- 从 `mmcls` 文档找到对应匹配的模型权重参数。并将改权重参数文件下载下来，放到 `checkpoints` 文件夹中。

```
1 mkdir checkpoints
2 wget
  https://download.openmmlab.com/mmcclassification/v0/resnet/resnet18_batch256_imagenet_20200708-34ab8f90.pth -P checkpoints
```

- 然后在 `resnet18_b16_flower.py` 文件中将预训练模型的访问路径加入。

```
1 load_from =
  '${YOUPATH}/mmclassification/checkpoints/resnet18_batch256_imagenet_20200708-34ab8f90.pth'
```

### 3.5 微调

- 使用 `tools/train.py` 进行模型微调

```
1 python tools/train.py ${CONFIG_FILE} [optional arguments]
```

- 指定训练过程中相关文件的保存位置，可以增加一个参数 `--work_dir ${YOUR_WORK_DIR}`.

```
1 python tools/train.py \
2   configs/resnet/resnet18_b16_flower.py \
```

## 3.6 完整示例

- 如下内容可命名为 resnet18\_bs32\_flower.py，在 mmclassification/configs 下创建 resnet18 目录，将该文件放到里面。

```

1  _base_ = ['./_base_/models/resnet18.py', './_base_/datasets/imagenet_bs32.py',
2          './_base_/default_runtime.py']
3
4  model = dict(
5      head=dict(
6          num_classes=5,
7          topk = (1,))
8  )
9  data = dict(
10     samples_per_gpu = 32,
11     workers_per_gpu = 2,
12     train = dict(
13         data_prefix = 'data/flower/train',
14         ann_file = 'data/flower/train.txt',
15         classes = 'data/flower/classes.txt'
16     ),
17     val = dict(
18         data_prefix = 'data/flower/val',
19         ann_file = 'data/flower/val.txt',
20         classes = 'data/flower/classes.txt'
21     )
22 )
23
24 optimizer = dict(type='SGD', lr=0.001, momentum=0.9, weight_decay=0.0001)
25 optimizer_config = dict(grad_clip=None)
26
27 lr_config = dict(
28     policy='step',
29     step=[1])
30
31 runner = dict(type='EpochBasedRunner', max_epochs=100)

```

```
32
33 # 预训练模型
34 load_from =
    '/HOME/shenpg/run/openmmlab/miclassification/checkpoints/resnet18_batch256_image
    net_20200708-34ab8f90.pth'
```

## 4. 提交计算

### 4.1 单卡计算

- 在环境、数据集、MMCls 配置文件准备完成之后就可以提交计算。在 N30 提交计算可以通过作业脚本的方式，操作步骤如下：
  - 1. 新建一个作业脚本 `run.sh`，脚本的解释器可以是 `/bin/sh`、`/bin/bash`、`/bin/csh` 脚本内容如下：

```
1 #!/bin/bash
2 # 加载模块
3 module load anaconda/2021.05
4 module load cuda/11.1
5 module load gcc/7.3
6
7 # 激活环境
8 source activate openmmlab_miclassification
9
10 # 刷新日志缓存
11 export PYTHONUNBUFFERED=1
12
13 # 训练模型
14 python tools/train.py \
15     configs/resnet18/resnet18_b32_flower.py \
16     --work-dir work/resnet18_b32_flower
```

- 2. 使用 `sbatch` 命令提交作业脚本。例如：

```
1 sbatch --gpus=1 run.sh
```

- `--gpus` 可以指定申请 GPU 的卡数，在 N30 分区可以申请的 GPU 卡数范围为 1~8，默认每卡配置 6 核CPU、60GB 内存。
- 执行 `sbatch --gpus=1 run.sh` 命令之后可申请到 1 GPU、6 核 CPU、60GB 内存。
- 提交成功之后会输出作业信息 “Submitted batch job 279689” 其中 279685 为作业ID，可以通过作业ID查看日志信息。

- 3. 使用 `squeue` 或 `parajobs` 查看提交的作业。

- 第一列 JOBID 是作业号，作业号是唯一的。
- 第二列 PARTITION 是作业运行的队列名。
- 第三列 NAME 是作业名称
- 第四列 USER 是超算账号。
- 第五列 ST 是作业状态。R（RUNNING）表示正常运行，PD（PENDING）表示在排队，CG（COMPLETING）表示正在退出，S 是管理员暂时挂起，CD（COMPLETED）已完成，F（FAILED）作业已失败
- 第六列 TIME 是作业运行时间。
- 第七列 NODES 是作业运行的节点数量
- 第八列 Nodelist（Reason）对于运行作业（R状态）显示作业使用的节点列表；如果是排队作业，显示排队原因。

```
1 $ parajobs
2 JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
3 279689      gpu    run.sh    shenpg  R      0:05      1 g0004
4 279689 作业GPU利用率为:
5 g0004: key_load_public: invalid format
6 g0004: index, utilization.gpu [%], utilization.memory [%], memory.total [MiB],
  memory.free [MiB], memory.used [MiB]
7 g0004: 0, 0 %, 0 %, 24576 MiB, 24268 MiB, 0 MiB
```

```
1 $ squeue
2 JOBID PARTITION    NAME     USER ST       TIME  NODES NODELIST(REASON)
3 279689      gpu    run.sh   shenpg R         0:07      1 g0004
```

- 4. 查看作业输出日志。默认标准输出和标准出错都定向到一个 `slurm-%j.log` (“%j”为作业ID) 文件中，当作业状态是 R 的时候，可在当前提交的路径下看到。可以通过 `tail` 等命令查看日志输出。例如：

```
1 $ tail -f slurm-279652.out  
2 2023-01-29 10:15:10,906 - mmcls - INFO - Saving checkpoint at 96 epochs  
3 [>>>>>>>>>>>>>>>>>>>>>>>>>] 572/572, 662.1 task/s, elapsed: 1s, ETA:  
0s2023-01-29 10:15:11,897 - mmcls - INFO - Epoch(val) [96][18] accuracy_top-  
1: 95.8042, accuracy_top-5: 100.0000  
4 2023-01-29 10:15:17,285 - mmcls - INFO - Saving checkpoint at 97 epochs  
5 [>>>>>>>>>>>>>>>>>>>>>>>>>] 572/572, 669.4 task/s, elapsed: 1s, ETA:  
0s2023-01-29 10:15:18,263 - mmcls - INFO - Epoch(val) [97][18] accuracy_top-  
1: 95.6294, accuracy_top-5: 100.0000  
6 2023-01-29 10:15:23,649 - mmcls - INFO - Saving checkpoint at 98 epochs
```

## 4.2 单节点多卡计算

- mmclassification 支持多节点、多卡训练，多节点多卡训练借助 torch.distributed.launch 实现。作业脚本 run.sh 参考如下：
  - --nproc\_per\_node 每个节点进程数量，通常以 GPU 卡数为准。
  - --launcher 是 mmclassification 中 tools/train.py 提供的参数，支持 pytorch、slurm、mpi，这里使用 pytorch。

```
1 #!/bin/bash
2 module load anaconda/2021.05
3 module load cuda/11.1
4 module load gcc/7.3
5
6 source activate openmmlab_mmclassification
7 export PYTHONUNBUFFERED=1
8
9 # 使用2块GPU
10 N_GPUS=2
11 python -m torch.distributed.launch \
12     --nproc_per_node=${N_GPUS} \
13     tools/train.py configs/resnet18/resnet18_b32_flower.py --work-dir
work/resnet18_b32_flower --launcher pytorch
```

- 提交脚本，当脚本中指定需要 2 块GPU计算时候，提交作业时需要指定 --gpus=2，执行 sbatch --gpus=2 run.sh 将申请到 2 块GPU、12核CPU、120GB内存。

```
1 sbatch --gpus=2 run.sh
```

## 4.3 多节点计算

- 如何获取每个节点的 host/IP？
  - 在提交的作业脚本中可以通过执行 scontrol show hostnames 命令来获取申请到的节点主机名。同时通过 for 循环的方式将每个节点的主机名保存到 host 变量中。如下：

```
1 for i in `scontrol show hostnames`
2 do
3     let k=k+1
```

```
4 host[$k]=$i
5 echo ${host[$k]}
6 done
```

- 多节点计算上需要考虑节点之间的通信，如何启用 InfiniBand 高速通信网络？

- 在 MMClassification 中提供的 nccl 的通信模式，在 N30 分区可以配合 **InfiniBand** 高速网络来提升技术效率。启用 **InfiniBand** 通信只需要在脚本中加入如下配置：

```
1 export NCCL_DEBUG=INFO
2 export NCCL_IB_DISABLE=0
3 export NCCL_IB_HCA=mlx5_bond_0
4 export NCCL_SOCKET_IFNAME=bond0
5 export NCCL_IB_GID_INDEX=3
```

- 如下申请 2 个计算节点参考示例、每个节点 2 块GPU、12 核CPU、120GB 内存。多节点运行时需要指定 `--nnodes`、`--node_rank`、`--nproc_per_node`、`--master_addr`、`--master_port`。

- `--nnodes`：节点的数量
- `--node_rank`：每个节点的 rank，通常主节点是 0，后面的每个节点依次是 1、2、3、4...
- `--nproc_per_node`：根据GPU数量而定，如果是 2 块GPU，则为 2。
- `--master_addr`：主节点的IP地址，也可以是 host。
- `--master_port`：主节点的通信端口。

```
1 #!/bin/bash
2 module load anaconda/2021.05
3 module load cuda/11.1
4 module load gcc/7.3
5
6 source activate opennmlab_mmclassification
7 export PYTHONUNBUFFERED=1
8
9 # 设置启用 InfiniBand
10 export NCCL_DEBUG=INFO
11 export NCCL_IB_DISABLE=0
12 export NCCL_IB_HCA=mlx5_bond_0
13 export NCCL_SOCKET_IFNAME=bond0
14 export NCCL_IB_GID_INDEX=3
15
16 # 获取每个节点(机器)的主机名
17 for i in `scontrol show hostnames`
```

```

18 do
19     let k=k+1
20     host[$k]=$i
21     echo ${host[$k]}
22 done
23
24 # 设置GPU卡数、节点数、主机通信端口
25 N_GPUS=2
26 N_NODES=2
27 MASTER_PORT=29501
28
29 # 第一个节点上运行
30 python -m torch.distributed.launch \
31     --nnodes=${N_NODES} \
32     --node_rank=0 \
33     --nproc_per_node=${N_GPUS} \
34     --master_addr="${host[1]}" \
35     --master_port=${MASTER_PORT} \
36     tools/train.py configs/resnet18/resnet18_b32_flower.py --work-dir
work/resnet18_b32_flower --launcher pytorch >> slurm_rank0_${SLURM_JOB_ID}.log
2>&1 &
37
38 # 第二节点上运行
39 srun -N 1 --gres=gpu:${N_GPUS} -w ${host[2]} \
40     python -m torch.distributed.launch \
41     --nnodes=${N_NODES} \
42     --node_rank=1 \
43     --nproc_per_node=${N_GPUS} \
44     --master_addr="${host[1]}" \
45     --master_port=${MASTER_PORT} \
46     tools/train.py configs/resnet18/resnet18_b32_flower.py --work-dir
work/resnet18_b32_flower --launcher pytorch >> slurm_rank1_${SLURM_JOB_ID}.log
2>&1 &
47 wait

```

- 提交计算，提交多节点作业需要用的几个作业参数：

-N	申请的节点（机器）数量。例如 -N 2 表示申请两台机器
--gres	每个节点申请的GPU卡数，需要与 --gpus 做出区分。例如 --gres=gpu:2 表示每个节点申请 2 块GPU



申请开通多节点计算权限后，添加此参数即可提交。例如：`--qos=gpugpu`

- ```
1 sbatch -N 2 --gres=gpu:2 --qos=gpugpu run.sh
```

- ```
1 #!/bin/bash
2 #SBATCH -N 2
3 #SBATCH --gres=gpu:4
4 #SBATCH --qos=gpu
```

- 正常情况下作业计算完成 或者出现异常报错，会自动退出。如果需要手动取消作业，可以执行 `scancel [作业ID]`。例如：

- 1. 在提交作业之后，使用 `parajobs` 或 `squeue` 查看作业的第八列 `NODELIST (REASON)` 对于运行作业（R状态）显示作业使用的节点列表。如下作业使用的节点是 `g0004` 。

```
1 $ squeue
2 JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
3 279689      gpu  run.sh  shenpg R         0:07      1 q0004
```

- ```
1 $ ssh g0004
2 $ nvidia-smi
3 Sun Jan 29 15:37:09 2023
4 +-----+-----+-----+-----+-----+-----+
```

```

5 | NVIDIA-SMI 515.65.01      Driver Version: 515.65.01      CUDA Version: 11.7      |
6 | -----+-----+-----+-----+-----+-----+-----+-----+
7 | GPU      Name              Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
8 | Fan    Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
9 |                                |                  |              MIG M. |
10 |=====+-----+-----+=====+-----+-----+=====+
11 |    0  NVIDIA GeForce ...   On      | 00000000:01:00.0 Off |                     N/A |
12 | 55%    59C    P2   216W / 350W | 2899MiB / 24576MiB |      53%      Default |
13 |                                |                  |              N/A |
14 |-----+-----+-----+-----+-----+-----+-----+
15 |
16 |-----+-----+-----+-----+-----+-----+-----+
17 | Processes:   |
18 | GPU      GI      CI           PID    Type    Process name                      GPU Memory |
19 |          ID      ID                                   |              Usage |
20 |=====+-----+-----+=====+-----+-----+=====+
21 |    0      N/A    N/A         6013      C      python                          2897MiB |
22 |-----+-----+-----+-----+-----+-----+-----+

```