Pytorch翻车记录：单卡改多卡踩坑记！

CV开发者都爱看的 极市平台 2022-12-20 22:00:22 发表于广东 手机阅读

作者丨哟林小平

来源丨夕小瑶的卖萌屋

编辑丨极市平台

极市导读

本文记录了作者尝试单卡改多卡加速的过程中出现的bug记录：一是继承DistributedSampler的漏洞百出，二是master进程无法正常结束，作者详细的阐述了出错的细节以及给出了修改的方法。>>加入极市CV技术交流群，走在计算机视觉的最前沿

先说明一下背景，目前正在魔改以下这篇论文的代码：

https://github.com/QipengGuo/GraphWriter-DGLgithub.com

由于每次完成实验需要5个小时（baseline），自己的模型需要更久（2倍），非常不利于调参和发现问题，所以开始尝试使用多卡加速。

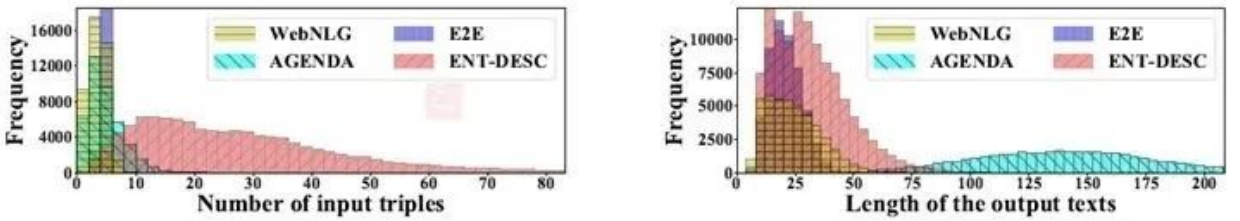
torch.nn.DataParallel ==> 简称 DP

torch.nn.parallel.DistributedDataParallel ==> 简称DDP

一开始采用dp试图加速，结果因为dgl的实现（每个batch的点都会打包进一个batch，从而不可分割），而torch.nn.DataParallel的实现是把一个batch切分成更小，再加上他的加速性能也不如ddp，所以我开始尝试魔改成ddp。

另外，作者在实现Sampler的时候是继承了torch.utils.data.Sampler这个类的，目的在于a

genda数据集的文本长度严重不均衡，如下：



(a) Number of the input triples in each dataset. (b) Length of the output texts in each dataset.

Figure 2: Dataset comparison among WebNLG, AGENDA, E2E and our ENT-DESC.

为了让模型更快train完，把长度相近的文本打包成一个batch（温馨提醒，torchtext也有相关的类 bucketiterator[1]，大概形式如下：

```
class BucketSampler(torch.utils.data.Sampler):
    def __init__(self, data_source, batch_size=32):
        self.data_source = data_source
        self.batch_size = batch_size

    def __iter__(self):
        idxs, lens, batch, middle_batch_size, long_batch_size = basesampler
        for idx in idxs:
            batch.append(idx)
            mlen = max([0]+[lens[x] for x in batch])
            #if (mlen<100 and len(batch) == 32) or (mlen>100 and mlen<220
            if (mlen<100 and len(batch) == self.batch_size) or (mlen>100 a
                yield batch
                batch = []
        if len(batch) > 0:
            yield batch

    def __len__(self):
        return (len(self.data_source)+self.batch_size-1)//self.batch_size
```

这是背景。

写bug第一步：继承DistributedSampler的漏洞百出

我一开始理想当然的把作者的sampler源码crtl-cv下来，唯独只改动了这里：

```
class DDPBaseBucketSampler(torch.utils.data.distributed.DistributedSampler):
```

随后就发现了几个问题：

- dataloader不会发包；

- dataloader给每个进程发的是完整的数据，按武德来说，应该是1/n的数据，n为你设置的gpu数量；

然后我就开始看起了源码[2]，很快啊：

```
def __iter__(self) -> Iterator[T_co]:
        if self.shuffle:
            # deterministically shuffle based on epoch and seed
            g = torch.Generator()
            g.manual_seed(self.seed + self.epoch)
            indices = torch.randperm(len(self.dataset), generator=g).tolist
        else:
            indices = list(range(len(self.dataset)))  # type: ignore

        if not self.drop_last:
            # add extra samples to make it evenly divisible
            padding_size = self.total_size - len(indices)
            if padding_size <= len(indices):
                indices += indices[:padding_size]
            else:
                indices += (indices * math.ceil(padding_size / len(indices
        else:
            # remove tail of data to make it evenly divisible.
            indices = indices[:self.total_size]
        assert len(indices) == self.total_size

        # subsample
        indices = indices[self.rank:self.total_size:self.num_replicas] # 这
        assert len(indices) == self.num_samples

        return iter(indices)
```

这里最关键的问题是是什么呢？首先在torch.utils.data.distributed.DistributedSampler里面，数据集的变量叫self.dataset而不是data_source；其次和torch.utils.data.Sampler要求你_重写__*iter*__函数不同：

```
def __iter__(self) -> Iterator[T_co]:
        raise NotImplementedError
```

DistributedSampler这个父类里有部分实现，如果你没有考虑到这部分，就自然会出现每个进程拿到的数据都是all的情况。

于是我重写了我的DDPBaseBucketSampler类：

```
def basesampler(lens, indices, batch_size):
    # the magic number comes from the author's code
```

```
        t1 = []
        t2 = []
        t3 = []
        for i, l in enumerate(lens):
            if (l<100):
                t1.append(indices[i])
            elif (l>100 and l<220):
                t2.append(indices[i])
            else:
                t3.append(indices[i])
        datas = [t1,t2,t3]
        random.shuffle(datas)
        idxs = sum(datas, [])
        batch = []

        #为了保证不爆卡，我们给不同长度的数据上保护锁
        middle_batch_size = min(int(batch_size * 0.75) , 32)
        long_batch_size = min(int(batch_size * 0.5) , 24)

        return idxs, batch, middle_batch_size, long_batch_size

    class DDPBaseBucketSampler(torch.utils.data.distributed.DistributedSampler):
        '''
        这里要注意和单GPU的sampler类同步
        '''
        def __init__(self, dataset, num_replicas, rank, shuffle=True, batch_siz
            super(DDPBaseBucketSampler, self).__init__(dataset, num_replicas, r
            self.batch_size = batch_size

        def __iter__(self):
            # deterministically shuffle based on epoch
            g = torch.Generator()
            g.manual_seed(self.epoch)
            #print('here is pytorch code and you can delete it in the /home/lz
            if self.shuffle:
                indices = torch.randperm(len(self.dataset), generator=g).tolist
            else:
                indices = list(range(len(self.dataset)))
            # add extra samples to make it evenly divisible
            indices += indices[:(self.total_size - len(indices))]
            assert len(indices) == self.total_size

            indices = indices[self.rank:self.total_size:self.num_replicas]
            assert len(indices) == self.num_samples

            # 然后我也要拿到每个数据的长度 (每个rank不同)
            lens = torch.Tensor([len(x) for x in self.dataset])

            idxs, batch, middle_batch_size, long_batch_size = basesampler(lens[

            for idx in idxs:
                batch.append(idx)
                mlen = max([0]+[lens[x] for x in batch])
                #if (mlen<100 and len(batch) == 32) or (mlen>100 and mlen<220
```

```
            if (mlen<100 and len(batch) == self.batch_size) or (mlen>100 a
                yield batch
                batch = []
        # print('应该出现2次如果是2个进程的话')
        if len(batch) > 0:
            yield batch

    def __len__(self):
        return (len(self.dataset)+self.batch_size-1)//self.batch_size
```

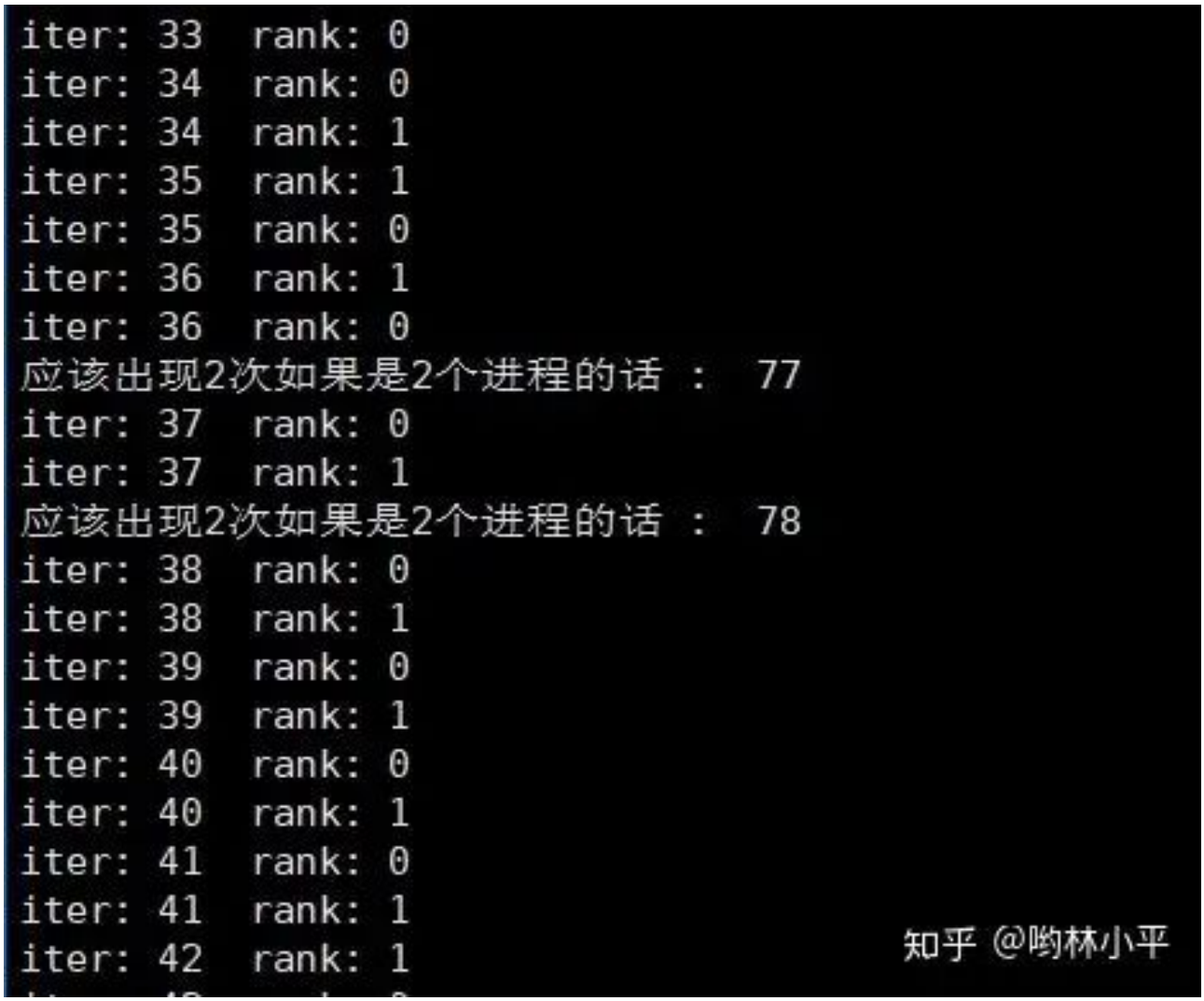
后面每个进程终于可以跑属于自己的数据了（1/n，n=进程数量=GPU数量，单机）

紧接着问题又来了，我发现训练过程正常结束后，主进程无法退出mp.spawn()函数。

写bug第二步，master进程无法正常结束

number workers ddp pytorch下无法正常结束。具体表现为，mp.spawn传递的函数参数可以顺利运行完，但是master进程一直占着卡，不退出。一开始我怀疑是sampler函数的分发batch的机制导致的，什么意思呢？就是由于每个进程拿到的数据不一样，各自进程执行sampler类的时候，由于我规定了长度接近的文本打包在一起，所以可能master进程有一百个iter，slave只有80个，然后我马上试了一下，很快啊：

▲DDPBucketSampler(torch.utils.data.distributed.DistributedSampler)类迭代函数__iter__



▲都能够正常打印，证明__iter__函数没有问题

发现只有细微的差别，并且，程序最后都越过了这些print，应该不会是batch数量不一致导致的问题。（顺便指的一提的是，sampler在很早的时候就把batch打包好了）

加了摧毁进程，也于事无补

```
if args.is_ddp:
      dist.destroy_process_group()
      print('rank destroy_process_group: ' , rank)
```

然后只能点击强制退出

```
File "train.py", line 322, in <module>
    main(args.gpu, args)
  File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/multiprocessin
    while not spawn_context.join():
  File "/home/lzk/anaconda3/lib/python3.7/site-packages/torch/multiprocessin
    timeout=timeout,
  File "/home/lzk/anaconda3/lib/python3.7/multiprocessing/connection.py", li
    ready = selector.select(timeout)
```

```
    File "/home/lzk/anaconda3/lib/python3.7/selectors.py", line 415, in select
      fd_event_list = self._selector.poll(timeout)
  TypeError: keyboard_interrupt_handler() takes 1 positional argument but 2 we
  ^CError in atexit._run_exitfuncs:
  Traceback (most recent call last):
    File "/home/lzk/anaconda3/lib/python3.7/multiprocessing/popen_fork.py", li
      pid, sts = os.waitpid(self.pid, flag)
  TypeError: keyboard_interrupt_handler() takes 1 positional argument but 2 we
```

代码参考：基于Python初探Linux下的僵尸进程和孤儿进程(三)[3]、 Multiprocessing in python blocked[4]

很显然是pytorch master进程产生死锁了，变成了僵尸进程。

再探究，发现当我把dataloader的number workers设为0的时候，程序可以正常结束。经过我的注释大法后我发现，哪怕我把for _i , batch in enumerate(dataloader)内的代码全部注释改为pass，程序还是会出现master无法正常结束的情况。所以问题锁定在dataloader身上。参考：nero：PyTorch DataLoader初探[5]

另外一种想法是，**mp.spawn**出现了问题。使用此方式启动的进程，只会执行和 target 参数或者 run() 方法相关的代码。Windows 平台只能使用此方法，事实上该平台默认使用的也是该启动方式。相比其他两种方式，此方式启动进程的效率最低。参考：Python设置进程启动的3种方式[6]

现在试一下，绕开mp.spawn函数，用shell脚本实现ddp，能不能不报错：

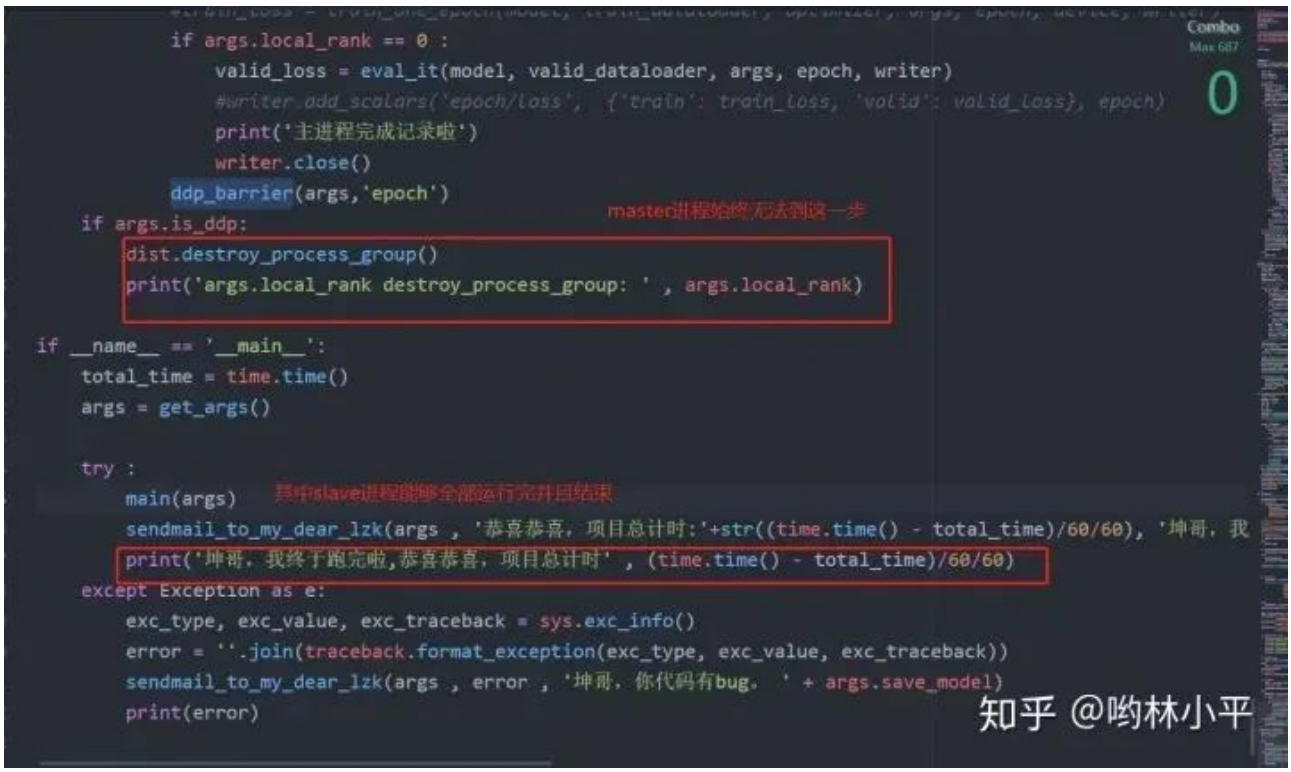
```
python -m torch.distributed.launch --nproc_per_node=2 --nnodes=1 --node_rank
```

参数解释：

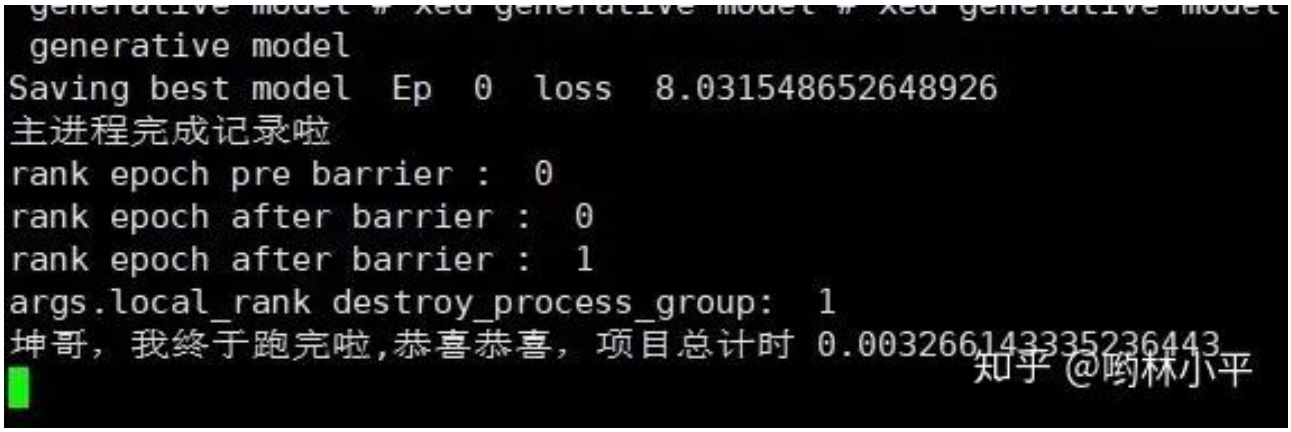
- nnodes：因为是单机多卡，所以设为1，显然node_rank 只能是0了
- local_rank:进程在运行的时候，会利用args插入local_rank这个参数标识进程序号

一番改动后，发现问题有所好转，最直观的感受是速度快了非常多！！现在我没有父进程的问题了，但还是在运行完所有的程序后，无法正常结束：

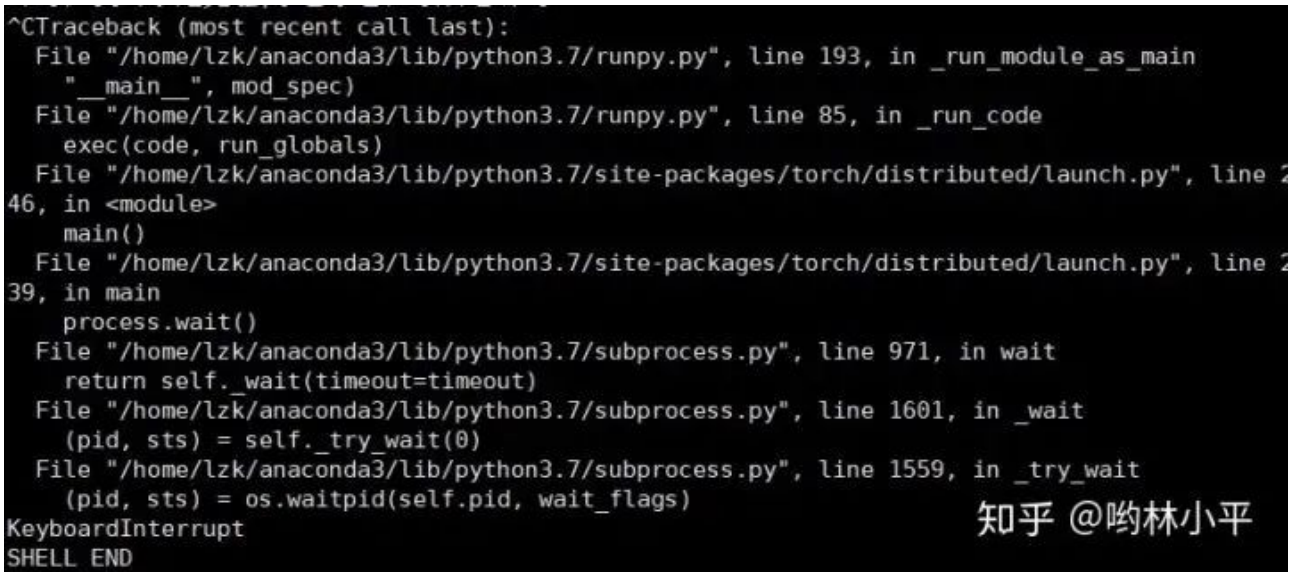
此时我的代码运行到：



上面的代码是main函数，2个进程（master，salve）都可以越过barrier，其中slave顺利结束，但是master却迟迟不见踪影：



这个时候ctrl+c终止，发现：



顺着报错路径去torch/distributed/launch.py, line 239找代码：

```
def main():
    args = parse_args()

    # world size in terms of number of processes
    dist_world_size = args.nproc_per_node * args.nnodes

    # set PyTorch distributed related environmental variables
    current_env = os.environ.copy()
    current_env["MASTER_ADDR"] = args.master_addr
    current_env["MASTER_PORT"] = str(args.master_port)
    current_env["WORLD_SIZE"] = str(dist_world_size)

    processes = []

    if 'OMP_NUM_THREADS' not in os.environ and args.nproc_per_node > 1:
        current_env["OMP_NUM_THREADS"] = str(1)
        print("*****************************************\n"
              "Setting OMP_NUM_THREADS environment variable for each proce
              "to be {} in default, to avoid your system being overloaded,
              "please further tune the variable for optimal performance in
              "your application as needed. \n"
              "*****************************************".format(current_er

    for local_rank in range(0, args.nproc_per_node):
        # each process's rank
        dist_rank = args.nproc_per_node * args.node_rank + local_rank
        current_env["RANK"] = str(dist_rank)
        current_env["LOCAL_RANK"] = str(local_rank)

        # spawn the processes
        if args.use_env:
            cmd = [sys.executable, "-u",
                   args.training_script] + args.training_script_args
        else:
            cmd = [sys.executable,
```

```
                    "-u",
                    args.training_script,
                    "--local_rank={}".format(local_rank)] + args.training_s

        process = subprocess.Popen(cmd, env=current_env)
        processes.append(process)

    for process in processes:
        process.wait() # 等待运行结束
        if process.returncode != 0:
            raise subprocess.CalledProcessError(returncode=process.returnco
                                                cmd=cmd)
```

可恶，master和dataloader到底有什么关系哇。。

这个问题终于在昨天（2020/12/22）被解决了，说来也好笑，左手是graphwriter的ddp实现，无法正常退出，右手是minst的ddp最小例程，可以正常退出，于是我开始了删减大法。替换了数据集，model，然后让dataloader空转，都没有发现问题，最后一步步逼近，知道我把自己的代码这一行注释掉以后，终于可以正常结束了：

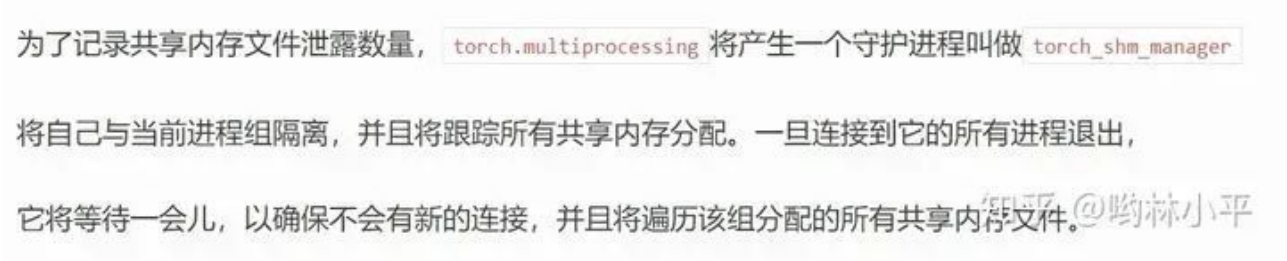
```
def main(args):
    ################################################################
    print('local_rank : ' , args.local_rank )
    if args.is_ddp:
        dist.init_process_group(
            backend='nccl',
                   init_method='env://',
            world_size=args.world_size,
            rank=args.local_rank
        )
    ################################################################
    # torch.multiprocessing.set_sharing_strategy('file_system')  万恶之源

    os.environ["CUDA_VISIBLE_DEVICES"] = os.environ["CUDA_VISIBLE_DEVICES"].
    args.device = torch.device(0)
    ...
```

为什么我当时会加上这句话呢？因为当时在调试number worker的时候（当时年轻，以为越大越好，所以设置成了number workers = cpu.count()），发现系统报错，说超出了打开文件的最大数量限制。在torch.multiprocessing的设定里，共享策略（参考pytorch中文文档[7]）默认是File descriptor，此策略将使用文件描述符作为共享内存句柄。当存储被移动到共享内存中，一个由 `shm_open` 获得的文件描述符被缓存。当时，文档还提到：

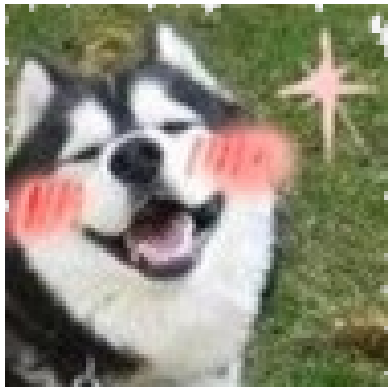
> 如果你的系统对打开的文件描述符数量有限制，并且无法提高，你应该使用 `file_system` 策略。

所以我换成了torch.multiprocessing.set_sharing_strategy('file_system')，但是却忽略文档里的共享内存泄露警告。显然，或许这不是严重的问题，文档里提到：



也有可能我所说的master进程就是这个torch_shm_manager，因为destory进程组始终无法结束0号进程：

这个BUG结束了，真开心，期待下一个BUG快快到来。



技术干货：数据可视化必须注意的30个小技巧总结 | 如何高效实现矩阵乘？万文长字带你从CUDA初学者的角度入门

实操教程：Nvidia Jetson TX2使用TensorRT部署yolov5s模型 | 基于YOLOV5的数据集标注＆训练，Windows/Linux/Jetson Nano多平台部署全流程

点击阅读原文进入CV社区

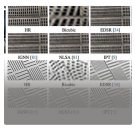
获取更多技术干货

阅读原文

喜欢此内容的人还喜欢

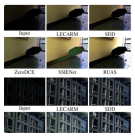
ICCV 2023 | 南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



ICCV23 | 将隐式神经表征用于低光增强，北大张健团队提出NeRCo

极市平台



YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台