

ipyparallel+MPI分布式并行计算

我坚信现在科学和技术的边界越来越模糊。将来的学生和现在的研究者们，应该具有更多交叉学科的知识基础...

需要学习的东西很多，但回报也将是巨大的...当然，真正的乐趣来自于你将从书中学到的知识应用到自己的工作实际中...

-----Gang Chen

认识python娘快两年了❤️



Ipython控制器和引擎

为了使用Ipython进行并行计算，必须启动一个控制器对象以及多个引擎对象。控制器和引擎可以运行在同一台机器上，也可以运行在不同机器上，这实际上代表可以通过Ipython完成复杂的异构计算。正常情况下直接在命令行中键入jupyter notebook实际上只是启动了一个进程，没有办法跟踪子进程做并行。ipyparallel库提供了对控制器和引擎进行控制的接口，在开出了多个引擎后，即可以对每个引擎创建相应的Client对象，通过不同的Client对象来调用后台不同的Ipython引擎，以进行并行计算。

控制器和引擎之间的逻辑关系大致是这样的：多个引擎负责实际的计算任务，这些引擎可能运行在同一台机器上，也可能运行在多台不同系统的机器上，但是要保证他们都能在网络中连接上控制器。控制器复杂控制调度各个引擎，给他们分发计算任务，并最终收集结果。

在ipyparallel中，有两种方式启动控制器和引擎：

1. 直接使用ipcluster命令：

首先安装好ipyparallel库，然后在命令行中，键入

```
ipcluster start -n 8
```

这里代表启动8个Ipython引擎，默认启动一个控制器，得到的输出是这样的：

```
(1ab) C:\Users\shen>ipcluster start -n 8
2021-05-24 19:06:29.014 [IPClusterStart] Removing pid file: C:\Users\shen\.ipython\profile_default\pid\ipcluster.pid
2021-05-24 19:06:29.015 [IPClusterStart] Starting ipcluster with [daemon=False]
2021-05-24 19:06:29.017 [IPClusterStart] Creating pid file: C:\Users\shen\.ipython\profile_default\pid\ipcluster.pid
2021-05-24 19:06:29.018 [IPClusterStart] Starting Controller with LocalControllerLauncher
2021-05-24 19:06:30.038 [IPClusterStart] Starting 8 Engines with LocalEngineSetLauncher
```

此时在jupyter中，可以发现已经成功启动了8个引擎，此时控制器默认就启动在本机上：

```
from ipyparallel import Client

rc = Client()

rc.ids

[0, 1, 2, 3, 4, 5, 6, 7]
```

注意这里每个引擎分配一个核，如果指定启动的引擎数多于机器的核数会报错。

2. 使用ipcontroller和ipengine手动指定控制器和引擎

当使用多台机器进行计算时，需要告诉控制器去监听来自外部机器接口的连接。这可以通过在命令行中指定ip参数来确定，若指定控制器监听来自所有接口的连接，则--ip='*'; 同时指定location参数为host0的ip，来使得启动引擎的机器能够与启动ipcontroller的host0机器连接，这里我的mac的ip地址为xxx.xx.xx.xxx，于是：

```
ipcontroller --ip='*' --location=xxx.xx.xx.xxx
```

此时会在用户目录下`.ipython/profile_default/security`中生成`ipcontroller-client.json`和`ipcontroller-engine`的配置文件。重新运行`ipcontroller`时，这两个文件会重新生成。其中`--n=8`代表启动8个引擎。假如现在在`host0`机器上启动了`ipcontroller`，想要在`host1-hostn`上分别启动引擎，并构建引擎和控制器之间的连接、需要以下两个步骤：

1. 把`host0`机器上生成的`ipycontroller-engine.json`文件拷贝到`host1-hostn`机器中的`.ipython/profile_default/security`文件夹中。
2. 在`host1-hostn`上启动引擎，在命令行里运行`ipengine`即可，但是由于一次只能生成一个引擎，可以把他们写进一个`shell`脚本里，批量生成多个引擎：

```
for engine in {1..8}
do
ipengine &
done
```

运行这个`shell`脚本，即可生成8个引擎，此时在`host0`机器上，可以检测到了：



```
from ipyparallel import Client
```



```
c = Client()
```



```
len(c)
```

8

当然也可以在mac上或者其他的服务器上继续再开一些引擎，方法都是一样的。

在Ipyparallel中使用MPI

使用MPI程序必须满足的两点：

1. 需要调用MPI的进程必须用mpirun或者支持MPI的批处理系统比如PBS启动
2. 一旦进程启动了，他必须要调用MPI_Init方法做初始化

所以像上面一小节中，虽然启动了多个引擎，但是并没有按照满足MPI需求的方式启动。ipyparallel库提供了很方便的接口：

```
ipcluster start -n 8 --engines=MPIEngineSetLauncher
```

通过指定engines=MPIEngineSetLauncher，ipcluster将首先启动一个控制器，然后使用mpirun启动一组引擎。中断ipcluster时，控制器和引擎也会自动被清理。

当引擎运行在不同机器上时，可以用mpirun手动用mpi启动ipengine：

```
mpiexec -n 8 ipengine --mpi
```

其中`--mpi`参数可以让引擎的rank和MPI的rank一致。这是最基本的使用，当然也可以自定义配置文件，传递给`--profile`参数，实现的东西都是类似的。当然，每次都指定ipcontroller的ip地址，以及拷贝ipycontroller-engine.json太麻烦了，在linux或者mac里可以指定alias把这些过程全部自动化（windows下面应该有对应的东西）：

```
# useful variables
export ip=$(ifconfig -a|grep inet|grep -v 127.0.0.1|\
grep -v inet6|awk '{print $2}'|tr -d "addr:")

# alias
alias ipcontroller='ipcontroller --ip='*' --location=${ip}'
alias ipengines="scp ~/.ipython/profile_default/security/ipcontroller-
engine.json \
your_host@xxx.xxx.xx.xxx:C:/Users/your_host/.ipython/profile_default/sec
urity && \
ssh your_host@xxx.xxx.xx.xxx \
'cd your_work_dir && \
mpiexec -n 8 ipengine --mpi'"
```

在mac里，可以在环境变量中添加当前的ip地址，然后通过alias指定在执行ipengines命令时先把生成的ipycontroller-engine.json直接用scp传递到远程主机上，然后在远程主机的工作目录下启动引擎。在windows上可以安装OpenSSH 服务器来使得可以像用ssh连接linux服务器一样连接windows主机，这里的your_host就是windows主机上用户名，后面是ip地址，your_work_dir是启动引擎的工作目录。工作目录中需要包含要运行的文件。为了使得在controller端修改文件后win上也能同步修改，比较方便的处理办法是把工作目录设置成坚果云的同步文件夹，这样就实现了文件共享。

运行ipcontroller和ipengines：


```
tmux                                                                    11:11
2021-05-26 10:27:08.056 [IPEngineApp] Registering with controller at tcp://183.173.90.34:49370
2021-05-26 10:27:08.056 [IPEngineApp] Registering with controller at tcp://183.173.90.34:49370
2021-05-26 10:27:08.059 [IPEngineApp] Registering with controller at tcp://183.173.90.34:49370
2021-05-26 10:27:08.063 [IPEngineApp] Registering with controller at tcp://183.173.90.34:49370
2021-05-26 10:27:08.074 [IPEngineApp] Registering with controller at tcp://183.173.90.34:49370
2021-05-26 10:27:08.203 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.211 [IPEngineApp] Completed registration with id 0
2021-05-26 10:27:08.217 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.223 [IPEngineApp] Completed registration with id 1
2021-05-26 10:27:08.224 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.230 [IPEngineApp] Completed registration with id 2
2021-05-26 10:27:08.233 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.235 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.239 [IPEngineApp] Completed registration with id 4
2021-05-26 10:27:08.239 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.240 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.242 [IPEngineApp] Completed registration with id 3
2021-05-26 10:27:08.242 [IPEngineApp] Starting to monitor the heartbeat signal from the hub every 3010 ms.
2021-05-26 10:27:08.245 [IPEngineApp] Completed registration with id 5
2021-05-26 10:27:08.246 [IPEngineApp] Completed registration with id 6
2021-05-26 10:27:08.248 [IPEngineApp] Completed registration with id 7
2021-05-26 10:29:36.247 [IPControllerApp] client::client b'4e70c24f-5fbde649cbcc60c2dff58cb' requested 'registration_request'
2021-05-26 10:29:36.249 [IPControllerApp] client::client b'b382c8e1-f27d902e953219978b00d43a' requested 'registration_request'
2021-05-26 10:29:36.250 [IPControllerApp] client::client b'9770df31-839895525c5c8f6a206e9e20' requested 'registration_request'
2021-05-26 10:29:36.252 [IPControllerApp] client::client b'53858efb-755fc5b3cf5190c80a76cfac' requested 'registration_request'
2021-05-26 10:29:36.253 [IPControllerApp] client::client b'bcd6f47a-9bd130cd595c67a0dbd2c414' requested 'registration_request'
2021-05-26 10:29:39.343 [IPControllerApp] registration::finished registering engine 7:bcd6f47a-9bd130cd595c67a0dbd2c414
2021-05-26 10:29:39.345 [IPControllerApp] engine::Engine Connected: 7
2021-05-26 10:29:39.346 [IPControllerApp] registration::finished registering engine 2:0d11124a-72266539d37f48cb47a7ccf2
2021-05-26 10:29:39.346 [IPControllerApp] engine::Engine Connected: 2
2021-05-26 10:29:39.348 [IPControllerApp] registration::finished registering engine 4:b382c8e1-f27d902e953219978b00d43a
2021-05-26 10:29:39.349 [IPControllerApp] engine::Engine Connected: 4
2021-05-26 10:29:39.349 [IPControllerApp] registration::finished registering engine 5:9770df31-839895525c5c8f6a206e9e20
2021-05-26 10:29:39.350 [IPControllerApp] engine::Engine Connected: 5
2021-05-26 10:29:39.351 [IPControllerApp] registration::finished registering engine 0:607692e3-48a6c460bbe413a9d68482b
2021-05-26 10:29:39.352 [IPControllerApp] engine::Engine Connected: 0
2021-05-26 10:29:39.352 [IPControllerApp] registration::finished registering engine 6:53858efb-755fc5b3cf5190c80a76cfac
2021-05-26 10:29:39.353 [IPControllerApp] engine::Engine Connected: 6
2021-05-26 10:29:39.353 [IPControllerApp] registration::finished registering engine 1:b5a5df71-8e21fbde1a8700584b02e10e
2021-05-26 10:29:39.353 [IPControllerApp] engine::Engine Connected: 1
2021-05-26 10:29:42.342 [IPControllerApp] registration::finished registering engine 3:4e70c24f-5fbde649cbcc60c2dff58cb
2021-05-26 10:29:42.343 [IPControllerApp] engine::Engine Connected: 3
[base] 0:ssh*                                                                    "chitongdeMacBook-Pro." 10:29 26- 5-21
```

成功连接。上面是在tmux里一个会话分隔出的两个窗格，tmux可以解绑会话与窗口，关掉后仍能在后台继续保持运行。想要关闭控制器和引擎时，运行tmux kill-session -t xxx即可。

numba+ipyparallel+mpi4py

python由于GIL的存在，本身无法实现多线程的并行计算，写多线程的话可能需要cython或者pybind等写一些C或者C++的程序。目前我自己觉得纯python比较方便实现相对较高的计算效率的方式就是结合numba和mpi4py，串行部分用numba完成，在原始程序外包装一个run.py的mpi4py运行接口，开出多个进程做并行化，以一个简单的向量求和为例，定义一个sum.py文件：

```
from mpi4py import MPI
import numpy as np
import numba as nb

@nb.jit(nopython=True)
def nb_sum(x):
    sum = 0
    for i in x:
        sum += i
    return sum

def psum(x):
    comm = MPI.COMM_WORLD
```

```

rank = comm.Get_rank()
size = comm.Get_size()
if rank==0:
    data = np.array_split(x,size)
else:
    data = None

data = comm.scatter(data,root=0)
local_sum = nb_sum(data)
all_sum = comm.reduce(local_sum,root=0,op=MPI.SUM)

if rank==0:
    return all_sum
else:
    return rank

```

其中局部求和函数用numba做优化，psum作mpi调度，在ipython中，运行如下命令：

```

import numpy as np
import ipyparallel as ipp

c = ipp.Client()
view = c[:]
view.activate()
view.run('sum.py')

view['x'] = np.arange(0,80,dtype='float')

%px sum = psum(x)

view['sum']

```

其中用%px魔法命令运行函数调用，用view.activate指定%px代表的是view，view先运行sum.py，让所有引擎都运行sum.py，都具有函数信息，用view['x']给所有的进程赋值x，最终输出为：

```
[3160.0, 1, 2, 3, 4, 5, 6, 7]
```


