

基于 Ray 分布式框架搭建 Windows 集群环境

刘 铭

用于解决 DETRIA 程序分布式计算



核科学与技术学院
哈尔滨工程大学

2021 年 4 月 18 日

前言

分布式系统是位于联网计算机上的组件仅通过传递消息来通信和协调其动作的系统。此定义导致分布式系统具有以下特别重要的特征：组件并发、缺少全局时钟以及组件独立故障 [1]。群集由并行执行实际处理的多台“工作”计算机组成。这些工作程序节点通常对直接用户访问是隐藏的。用户登录到提供系统管理的“头节点”计算机，在头节点上提交并行处理任务，然后该并行处理任务将并行任务分配给某些（或所有）工作程序节点。用户提供诸如需要多少工作节点之类的规范。为了允许多个用户以有组织的方式使用集群，可以使用开源集群管理软件 [2]。Ray 实现了一个统一的界面，该界面可以表示任务并行和基于参与者的计算，并由单个动态执行引擎支持。为了满足性能要求，Ray 使用了分布式调度程序和分布式且容错的存储来管理系统的控制状态。Ray 将任务并行和参与者编程模型统一在一个单一的动态任务图中，并采用了一个可伸缩的架构，该架构由全局控制存储和自底向上的分布式调度程序支持。这种体系结构同时实现的编程灵活性、高吞吐量和低延迟对于新兴的人工智能工作负载特别重要，这些工作负载产生的任务在资源需求、持续时间和功能上各不相同 [3]。

动态事件树风险分析软件（Dynamic Event Tree Risk-Informed Analysis, DETRIA）是一个为核电厂确定论与概率论耦合安全分析的软件。该软件在运行时需调用 RELAP5 为其计算（RELAP5 仅允许在 Windows 系统运行），再而为完成核电站安全分析时需要大量抽样，因此单台计算机运行该软件显现出运行时间久、效率低等缺点。为减少运行时间，提高该软件的效率，需要搭建 Windows 系统分布式集群环境，来弥补这些缺点。

目录

1	介绍	1
1.1	分布式集群框架	1
1.1.1	ParallelPython	1
1.1.2	Dask	2
1.1.3	mpi4py	3
1.1.4	Ray	4
2	准备工作	5
2.1	硬件准备	5
2.2	安装操作系统	5
2.3	记录节点 IP 地址	5
3	搭建 Ray 分布式集群	5
3.1	安装 AnaConda	5
3.2	安装 Visual C++	6
3.3	安装 Ray	7
4	启动 Ray 搭建集群环境	7
4.1	启动 Ray head 节点	7
4.2	向集群添加子节点	8
4.3	在 Ray 集群上运行 Ray 程序	9
4.3.1	初始化 Ray	9
4.3.2	在代码中调用 ray 的 API	10
5	运行分布式 DETRIA	10
6	总结	10

1 介绍

在 Python 的wiki 列表中, 存在许多与 Python 相关的库, 可用于在对称多处理 (SMP) 或共享内存环境中使用多个 CPU 或多核 CPU 以及在群集或网格环境中使用大量计算机。

与 SMP 体系结构不同, 尤其是与基于线程的并发相反, 群集 (和网格) 体系结构由于相对缺乏共享资源而提供了高可伸缩性, 尽管这会使编程范式对于未开始的开发人员而言似乎有些陌生。在此域中, 可能会发现与其他分布式计算技术有一些重叠。

1.1 分布式集群框架

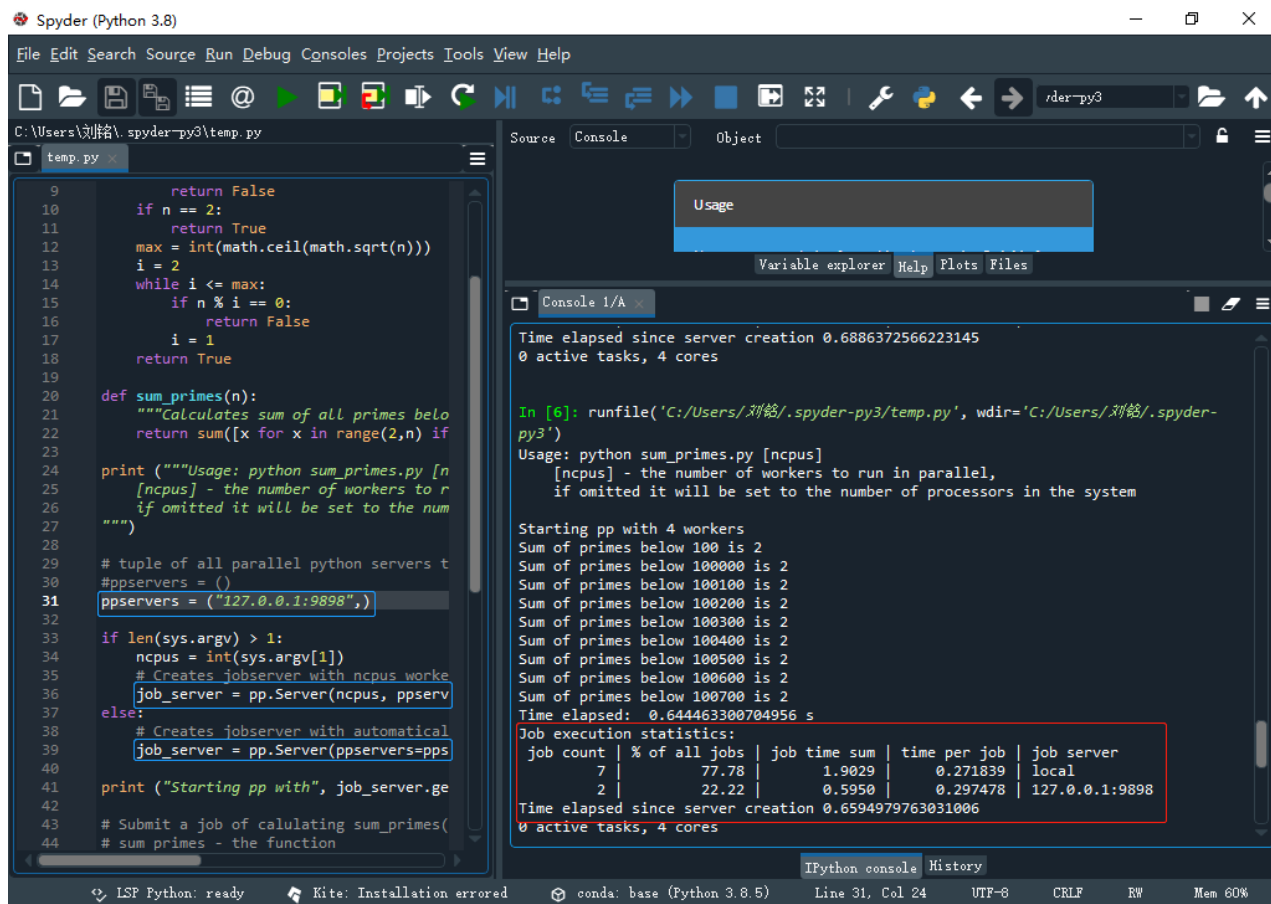
下面介绍几个常用的分布式集群框架。

1.1.1 ParallelPython

ParallelPython是一个是用纯 Python 编写的开源和跨平台模块, 提供了在 SMP (具有多个处理器或核心的系统) 和群集 (通过网络连接的计算机) 上并行执行 python 代码的机制。它比较轻巧, 易于安装并与其他 python 软件集成。具有以下一些特征:

- 可在 SMP 和集群上并行执行 python 代码
- 易于理解和实现基于作业的并行化技术 (易于并行转换串行应用程序)
- 可自动检测最佳配置 (默认情况下, 工作进程数设置为有效处理器数)
- 可动态处理器分配 (工作进程数可以在运行时更改)
- 具有相同功能的后续作业的开销很低 (实现透明缓存以减少开销)
- 可动态负载平衡 (作业在运行时在处理器之间分配)
- 容错 (如果其中一个节点失败, 则将任务重新安排到其他节点上)
- 可自动发现计算资源
- 可动态分配计算资源 (自动发现和容错的后果)
- 基于 SHA 的网络连接身份验证
- 跨平台的可移植性和互操作性 (Windows, Linux, Unix, Mac OS X)
- 跨体系结构的可移植性和互操作性 (x86, x86-64 等)
- 开源

该开源框架存在的不足之处在于开发版本较少，且大部分版本基于 python2. 开发，而 DETRIA 程序的代码是为 python3.。因此，在调用该模块需更改大量代码并且可能遇到因 python3. 不向下兼容而导致无法调用的情况。图1为 ParallelPython 实现简单分布式程序的结果。



The screenshot shows the Spyder Python IDE interface. The left pane displays a Python script named `temp.py` which implements a parallel prime number summation using ParallelPython. The script defines a `sum_primes(n)` function and a `ppservers` tuple. The right pane shows the console output, which includes the usage instructions, the start of the ParallelPython server with 4 workers, a list of prime sums for various ranges, and a table of job execution statistics.

```
9         return False
10    if n == 2:
11        return True
12    max = int(math.ceil(math.sqrt(n)))
13    i = 2
14    while i <= max:
15        if n % i == 0:
16            return False
17        i = i + 1
18    return True
19
20 def sum_primes(n):
21     """Calculates sum of all primes below n"""
22     return sum([x for x in range(2,n) if
23
24 print ("""Usage: python sum_primes.py [n
25 [ncpus] - the number of workers to r
26 if omitted it will be set to the num
27 """)
28
29 # tuple of all parallel python servers t
30 #ppservers = ()
31 ppservers = ("127.0.0.1:9898",)
32
33 if len(sys.argv) > 1:
34     ncpus = int(sys.argv[1])
35     # Creates jobserver with ncpus worke
36     job_server = pp.Server(ncpus, ppserv
37 else:
38     # Creates jobserver with automatical
39     job_server = pp.Server(ppservers=pps
40
41 print ("Starting pp with", job_server.ge
42
43 # Submit a job of calculating sum_primes(
44 # sum primes - the function
```

```
Usage
Variable explorer Help Plots Files

Console 1/A
Time elapsed since server creation 0.6886372566223145
0 active tasks, 4 cores

In [6]: runfile('C:/Users/刘皓/.spyder-py3/temp.py', wdir='C:/Users/刘皓/.spyder-py3')
Usage: python sum_primes.py [ncpus]
[ncpus] - the number of workers to run in parallel,
if omitted it will be set to the number of processors in the system

Starting pp with 4 workers
Sum of primes below 100 is 2
Sum of primes below 100000 is 2
Sum of primes below 100100 is 2
Sum of primes below 100200 is 2
Sum of primes below 100300 is 2
Sum of primes below 100400 is 2
Sum of primes below 100500 is 2
Sum of primes below 100600 is 2
Sum of primes below 100700 is 2
Time elapsed: 0.644463300704956 s
Job execution statistics:
job count | % of all jobs | job time sum | time per job | job server
7 | 77.78 | 1.9029 | 0.271839 | local
2 | 22.22 | 0.5950 | 0.297478 | 127.0.0.1:9898
Time elapsed since server creation 0.6594979763031006
0 active tasks, 4 cores
```

图 1: ParallelPython 分布式计算

1.1.2 Dask

Dask是用于 Python 中并行计算的灵活库，该库由两部分组成：

1. 动态任务调度针对计算进行了优化。这类似于 Airflow, Luigi, Celery 或 Make，但针对交互式计算工作负载进行了优化。
2. “大数据”集合（例如并行数组，数据框和列表）将诸如 NumPy, Pandas 或 Python 迭代器之类的通用接口扩展到了大于内存或分布式的环境。这些并行集合在动态任务计划程序之上运行。

Dask 有以下特点：

- Familiar: 提供并行的 NumPy 数组和 Pandas DataFrame 对象
- Flexible: 提供任务计划界面，用于更多自定义工作负载并与其他项目集成
- Native: 在纯 Python 中启用分布式计算，并可以访问 PyData 堆栈
- Fast: 以低开销，低延迟和快速数值算法所需的最少序列化操作
- Scales up: 在具有 1000 个核心的集群上弹性运行
- Scales down: 在单个过程中轻松设置并在笔记本电脑上运行
- Responsive: 设计时考虑了交互式计算，它提供了快速的反馈和诊断功能，以帮助用户

Dask 分布式框架提供了丰富的集群配置方法：1) 命令行设置；2) SSH 配置；3) 使用 MPI 等工具在传统 HPC 环境中配置；4) 通过流行的 Kubernetes 资源管理器部署；5) 在 YARN 群集上部署；6) 自定义 Python 的 API；7) Docker 容器；8) 在常见云服务器上部署。图2是通过命令行方式配置的 Dask 集群。

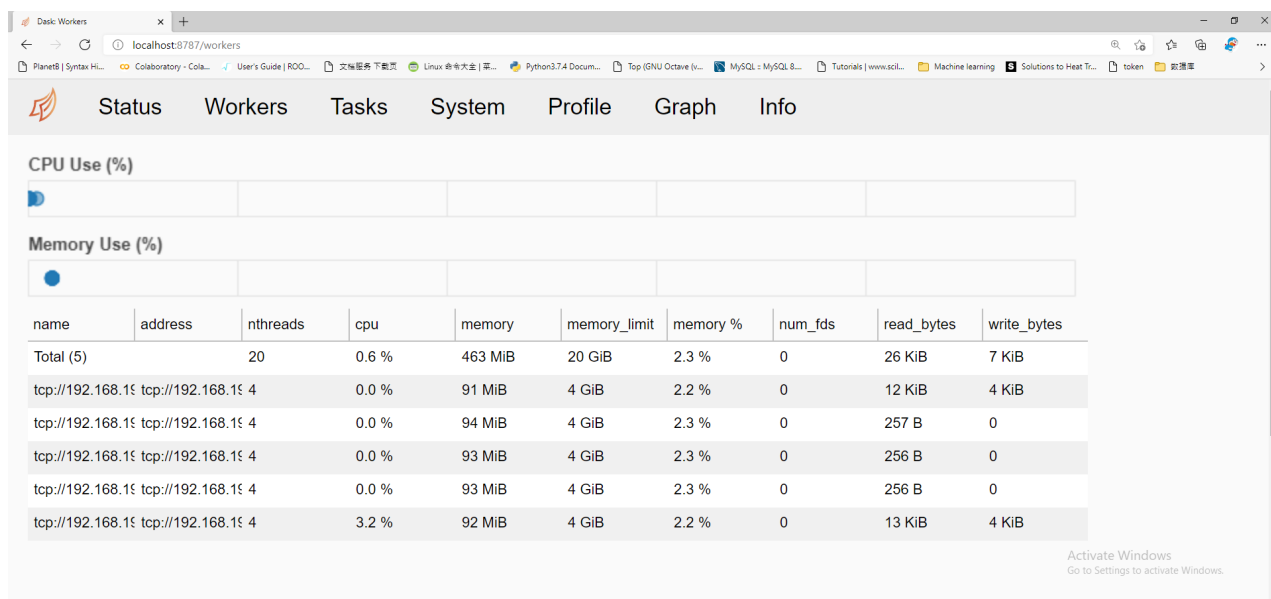


图 2: Dask 集群 Dashboard

1.1.3 mpi4py

mpi4py(MPI for python) 是一个构建在 MPI 之上的 Python 库，它使得 Python 的数据结构可以方便的在多进程中传递。它实现了很多 MPI 标准中的接口，包括点对点通信、集合通信、阻塞 / 非阻塞通信、组间通信等，基本上能用到的 MPI 接口都有相应的实现。而MPI是一种标准化的和便携式的消息传递系统，其设计功能上各种各样的并行计算机。虽

然 mpi4py 是一个框架库，但是其本质仅仅是为了用户在编写 python 程序时更加方便的调用 MPI 接口而包装的一个框架。因此，在使用 mpi4py 时，需要用户在计算机中安装 MPICH 或 Open MPI。

所以，尽管 mpi4py 在理论上可以实现分布式集群环境的搭建，但是相比于现有其他强大的 python 库，mpi4py 需要用户自己编写更加底层的代码，再者 DETRIA 程序的整体已经编写完成，若要使用 mpi4py 搭建分布式集群计算，需要从程序开发框架上重新编写，工作量较大。综上，mpi4py 不考虑作为 DETRIA 的集群计算环境搭建框架。

1.1.4 Ray

Ray提供了用于构建分布式应用程序的简单通用 API。Ray 的工作是确保应用程序能够以分布式方式运行，并具有分布式计算所需的所有节点内通信、数据传输和抗故障能力。Ray 具有以下特征：

1. 提供用于构建和运行分布式应用程序的简单 API
2. 使用户能够并行化仅在单台机器编写代码，几乎不用更改代码
3. 在核心 Ray 之上包括一个大型的应用程序、库和工具生态系统，以支持复杂的应用程序

但是，Ray 当前支持 MacOS 和 Linux。现在 Windows 稳定版暂时未推出，但是有 Windows 试验版并且正在开发中。图3为 Ray 在 Linux 系统上搭建的集群环境。

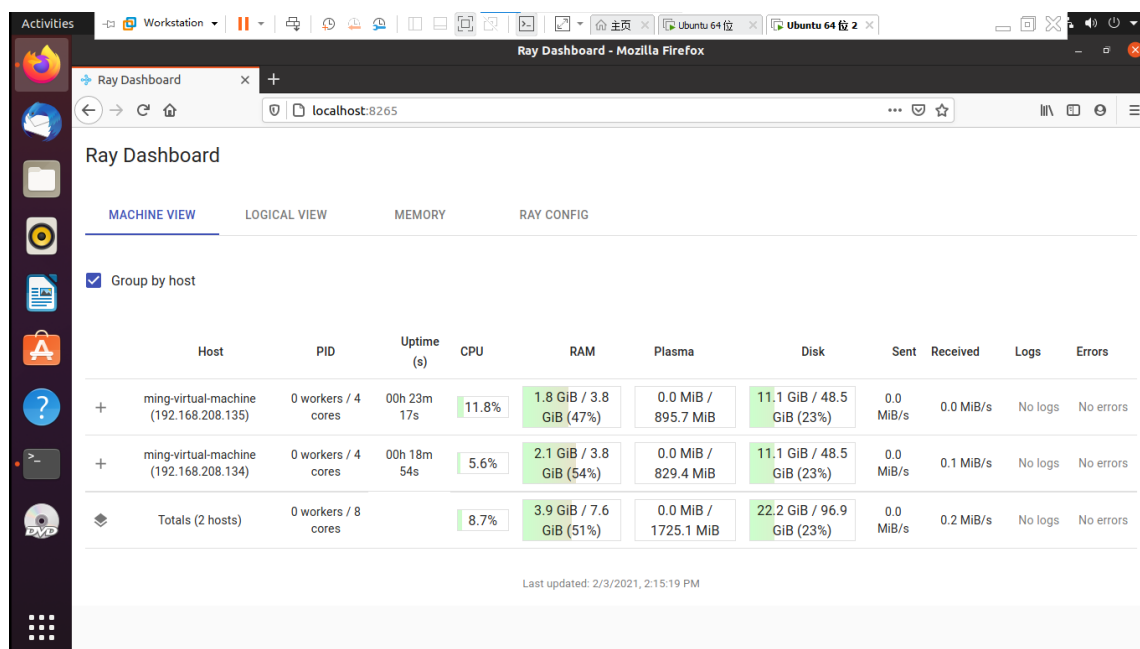


图 3: Ray 在 Ubuntu 系统上搭建集群环境

除了上述介绍的几种分布式框架外，Python 还有非常多分布式集群库，例如pathos、celery、yami4等等。

2 准备工作

搭建分布式集群环境需要准备几台互联的计算机，并且在每台计算机上都安装好分布式环境以及 DETRIA 软件运行需要的环境（RELAP5 等）。

2.1 硬件准备

1. 若干台计算机节点
2. 连接工具：交换机、网线、路由器

2.2 安装操作系统

Windows Server 2019 是由微软（Microsoft）官方推出的最新版服务器版操作系统，该系统基于 Win Server 2016 开发而来，后者是微软迄今为止普及速度最快的服务器系统。WinServer 2019 与 Win10 同宗同源，提供了 GUI 界面，包含了大量服务器相关新特性，也是微软提供长达十年技术支持（简称 LTSC）的新一代产品。Windows Server 2019 主要用于 VPS 或服务器上，可用于架设网站或者提供各类网络服务。

安装 Windows Server 2019 的主要步骤：

1. 在Microsoft 官网下载 ISO 镜像文件
2. 使用安装介质（USB 闪存驱动器、DVD 等）安装系统
3. 设置密码（每台计算机密码尽量设置相同），激活系统

2.3 记录节点 IP 地址

安装好操作系统后，打开命令提示符，输入 *ifconfig* 命令查看计算机 ip，查看计算机 ip 地址，如4所示。

3 搭建 Ray 分布式集群

3.1 安装 AnaConda

Conda 是一个开源跨平台语言无关的包管理与环境管理系统。由“连续统分析”基于 BSD 许可证发布。Conda 允许用户方便地安装不同版本的二进制软件包与该计算平台需要


```
C:\Windows\system32\cmd.exe
C:\Users\刘铭>ipconfig

Windows IP 配置

以太网适配器 以太网:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 1:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 2:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

以太网适配器 VMware Network Adapter VMnet1:

    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::4038:f766:ea3:295b%6
    IPv4 地址 . . . . . : 192.168.128.1
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . :

以太网适配器 VMware Network Adapter VMnet8:

    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::ad84:713:3571:7f9a%2
    IPv4 地址 . . . . . : 192.168.195.1
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . :

无线局域网适配器 WLAN:

    连接特定的 DNS 后缀 . . . . . :
    IPv6 地址 . . . . . : 2409:8a50:3421:dac0:f5a6:26cc:38d:751f
    临时 IPv6 地址. . . . . : 2409:8a50:3421:dac0:38f3:71e2:5729:5ef
    本地链接 IPv6 地址. . . . . : fe80::f5a6:26cc:38d:751f%10
    IPv4 地址 . . . . . : 192.168.1.118
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : fe80::1%10
    192.168.1.1
```

图 4: ipconfig 查看 IP 地址

的所有库。还允许用户在不同版本的包之间切换、从一个软件仓库下载包并安装。Conda 是用 Python 语言开发，但能管理其他编程语言的项目，包括多语言项目。Anaconda 是一个免费开源的 Python 和 R 语言的发行版本，用于计算科学（数据科学、机器学习、大数据处理和预测分析），Anaconda 致力于简化软件包管理系统和部署。Anaconda 的包使用软件包管理系统 Conda 进行管理。

安装 anaconda 的步骤：

1. 下载Anaconda 安装包
2. 使用SHA-256验证数据完整性，防止下载到被篡改的安装包或下载过程中文件损坏
3. 双击安装程序以启动
4. 根据安装提示进行安装，也可查看官方安装步骤
5. 安装完成后，在命令提示符中输入 `conda list` 查看安装是否成功

3.2 安装 Visual C++

对于 Windows，在使用 Ray 之前，必须安装最新的 Visual C++。否则，当 Ray 无法找到运行时库文件（例如 VCRUNTIME140_1.dll）时，可能会收到类似于以下错误：

FileNotFoundError: Could not find module '`__raylet.pyd`' (or one of its dependencies).

安装 Visual C++ 的步骤:

1. 下载VisualStudio 引导程序安装包
2. 下载完成后通过命令行输入以下命令 (也可以双击安装包打开图形化界面进行安装):

```
1 vs_community.exe --layout c:\vslayout --add Microsoft.VisualStudio.Workload.  
NativeDesktop --includeRecommended --lang en-US  
2 c:\vslayout\vs_community.exe --noweb --add Microsoft.VisualStudio.Workload.  
NativeDesktop --includeRecommended
```
3. 安装最新的Visual C++

3.3 安装 Ray

安装 Ray 的步骤:

1. 使用 Conda 创建 Ray 环境

```
conda create -n ray python=3.8
```
2. 开启 Ray 环境

```
conda activate ray
```
3. 通过 pip 安装 Ray

```
pip install -U ray
```

4 启动 Ray 搭建集群环境

4.1 启动 Ray head 节点

通过命令行运行以下代码启动 Ray 的 head 节点:

```
ray start --head
```

屏幕会正常输出以下内容:

```
(base) C:\Users\刘铭\Downloads>ray start --head
```

```
Local node IP: 192.168.1.118
```

```
2021-02-23 16:55:53,531 INFO services.py:1171 -- View the Ray dashboard at http://  
localhost:8265
```

```
-----  
Ray runtime started.  
-----
```

Next steps

To connect to this Ray runtime from another node, run

```
ray start --address='192.168.1.118:6379' --redis-password='5241590000000000'
```

Alternatively, use the following Python code:

```
import ray  
ray.init(address='auto', _redis_password='5241590000000000')
```

If connection fails, check your firewall settings and network configuration.

To terminate the Ray runtime, run

```
ray stop
```

4.2 向集群添加子节点

根据 head 节点启动后的提示, 在另外几台计算机中, 输入以下命令:

```
ray start --address='192.168.1.118:6379' --redis-password='5241590000000000'
```

如果该节点 ray 正常安装, 并且该节点能够与 head 节点正常通信, 屏幕会输出以下内容:

```
Local node IP: 192.168.1.134
```

```
-----  
Ray runtime started.  
-----
```

To terminate the Ray runtime, run

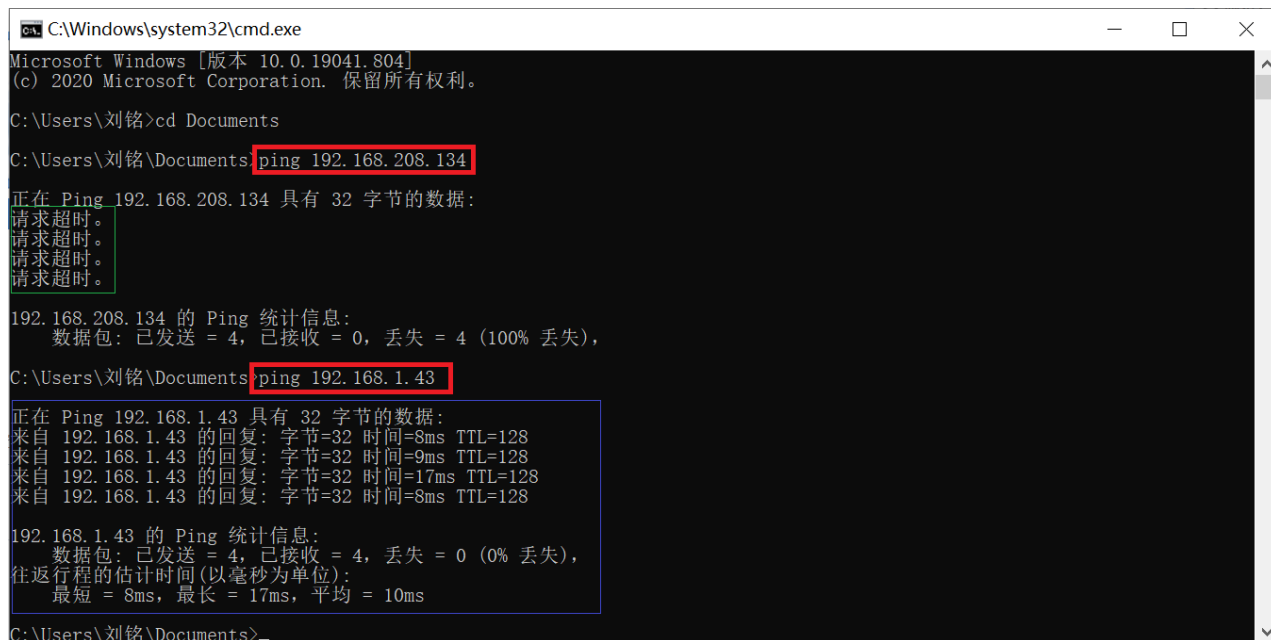
```
ray stop
```

在添加子节点时, 可指定一台计算机有几个 CPU 和 GPU, 用命令 `--num-cpus = 10 --num-gpus = 1` 表示, 更多 Ray 配置信息可查看官方页面。

若出现以下错误信息, 则是因为网络 IP 无法访问:

Unable to connect to Redis. If the Redis instance is on a different machine, check that your firewall is configured properly.--port

可通过 *ping* 工具进行检测，测试数据包能否透过 IP 协议到达 head 节点主机，如图5所示。



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19041.804]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\刘铭>cd Documents
C:\Users\刘铭\Documents>ping 192.168.208.134

正在 Ping 192.168.208.134 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

192.168.208.134 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 0, 丢失 = 4 (100% 丢失),

C:\Users\刘铭\Documents>ping 192.168.1.43

正在 Ping 192.168.1.43 具有 32 字节的数据:
来自 192.168.1.43 的回复: 字节=32 时间=8ms TTL=128
来自 192.168.1.43 的回复: 字节=32 时间=9ms TTL=128
来自 192.168.1.43 的回复: 字节=32 时间=17ms TTL=128
来自 192.168.1.43 的回复: 字节=32 时间=8ms TTL=128

192.168.1.43 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 8ms, 最长 = 17ms, 平均 = 10ms

C:\Users\刘铭\Documents>
```

图 5: Ping 工具检测网络连接

4.3 在 Ray 集群上运行 Ray 程序

要在分布式 Ray 集群环境下运行程序，需要在与节点之一的机器上执行程序，并且在其他每台计算机中安装好相同的运行环境。

4.3.1 初始化 Ray

在程序/代码中，您必须调用 `ray.init` 并将 `address` 参数添加到 `ray.init`，使得 Ray 连接到现有群集。例如：

```
ray.init(address="auto")
```

若要验证已成功加入集群的节点数量，可以运行以下代码：

```
1 import ray
2 import time
3
4 @ray.remote
```

```
5 def f():
6     time.sleep(0.01)
7     return ray.services.get_node_ip_address()
8
9 set(ray.get([f.remote() for _ in range(1000)]))
```

4.3.2 在代码中调用 ray 的 API

Ray 为方便用户更好的使用,提供了大量 API 供用户使用,如 *ray.init*、*ray.is_initialized*、*ray.remote*、*ray.get*、*ray.method* 等等, 具体使用方式见 Ray 官方 API 文档。

5 运行分布式 DETRIA

6 总结

单台计算机在执行 DETRIA 存在计算时间长、效率低等缺点。本文以 1 个 head 节点和 3 个 node 节点为例,通过搭建基于 Ray 分布式框架的 Windows 集群环境,减少了 DETRIA 的计算时间 ** %, 提高运行效率 ** %。

参考文献

- [1] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.
- [2] A. M. Pfalzgraf and J. A. Driscoll. A low-cost computer cluster for high-performance computing education. In *IEEE International Conference on Electro/Information Technology*, pages 362–366, 2014.
- [3] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.