

第 8 章:

Dask (2015)

由 Anaconda 支持的 Dask 是一个 Python 编写的、支持在大于内存 (Larger-than-memory)的数据集上提供多核和分布式并行计算的开源框架。

Dask 不仅提供了模拟 NumPy、Lists 和 Pandas 的高层模块 —— Array、Bag 和 DataFrame，也提供了并行执行任务图的动态任务调度程序以替代复杂情况下直接使用线程库 (Threading) 或多进程库 (Multi-processing) 库或其他任务调度系统 (如 IPython parallel) 方案。

相比 Spark, Dask 是一个更轻盈的分布式计算框架——也体现了 DSM (Distributed Shared Memory)的思想，能在分布式集群中进行分布式并行计算，也可以在单机(多核心)中进行伪分布式并行计算。

Dask 是一个十分灵活的用于 Python 的并行计算库，是一个值得关注的框架——在科学计算和数据科学领域，表现都很不错！

- 作者整理自网络

在大数据时代，Hadoop 和 Spark 为数据科学家提供了强有力的管理大数据并完成计算的技术，但是，由于它们都保留了基于 Key-Value 进行数据转换以完成计算的方式，使得它们不能满足依赖迭代方式的科学计算/HPC 领域。不过，Spark 的成功却也表明了 DSM (Distributed Shared Memory)/PGAS (Partitioned Global Address Space) 的可行性，也启发后来者尝试提供新的框架以满足科学计算领域/HPC 的需求。其中，Dask 便是目前表现不错的框架^{[139][140][141][142][143]}。

8.1 Dask框架的基本信息

Dask 是发布于 2015 年的一个用于并行计算的开源库，所以与 Spark 相比它比较新。这个框架原本是 Continuum Analytics (即现在的 Anaconda 公司)开发的，这个公司是很多 Python 开源包的创造者，包括 Anaconda Python 发行版ⁱ。

Dask 最初的目的只是将 NumPy 并行化，这样它就可以利用具有多个 CPU 和核的工作站计算机。

ⁱ Dask 是 Anaconda 的产品，背后的主要贡献者是 Matthew Rocklin。最近 Matthew Rocklin 加入了 Nvidia，开始做 Dask.CuDF，开发基于 GPU 的 Dask，结合 Rapids CuDF (基于 GPU 的 Pandas)。最新的博客里面，显示他准备建立一个 Dask 的公司，推进 Python 的分布式数据平台。

与 Spark 不同，在 Dask 开发中采用的最初设计原则之一是“什么都不发明”。这一决定背后的想法是，与 Dask 一起使用 Python 进行数据分析应该让开发人员感到熟悉，并且知识升级时间应该很短。根据其创建者的说法，Dask 的设计原则经过了多年的发展，现在它正在被开发成一个用于并行计算的通用库。

Dask 和核心是弥补 Python 在数据科学中的不足，主要是性能上。Python 单机的能力不能够支持数据科学中大数据集的快速计算。Dask 提供了基础的数据结构，底层是分布式计算架构。数据结构包括：Array、Dataframe。Array 兼容 Numpy 的 Ndarray，Dataframe 兼容 Pandas 的 Dataframe。“兼容”是个相对的说法，毕竟 Pandas 和 Numpy 发展多年，本身也在发展，接口非常多。

[140]

Dask 是一个开源库，旨在为现有 Python 堆栈提供并行性ⁱ。Dask 与 Python 库(如 NumPy 数组、Pandas DataFrame 和 Scikit-Learn)集成，无需学习新的库或语言，即可跨多个核心、处理器和计算机实现并行执行。

类似 Nvidia CUDA 为编程人员提供了虚拟的无穷的计算单元，Dask 将集群系统中分散在不同节点中的内存整合起来，为用户提供一个统一的虚拟的无穷内存来执行 Dask 程序——这也就是 DSM 的思想(参看 2.5.2 中的描述)。

用于并行列表、数组和 DataFrame 的 API 集合，可原生扩展 Numpy、NumPy、Pandas 和 Scikit-Learn，以在大于内存环境或分布式环境中运行。Dask 集合是底层库的并行集合(例如，Dask 数组由 Numpy 数组组成)并运行在任务调度程序之上。

8.2 Dask编程

Dask 继承了 Spark 类似的数据处理的能力，并且跳出了 Key-Value 计算模式的限制，使得它既适用于大数据处理，也能够高效地完成依赖迭代计算的科学计算/HPC。

Dask 是一种易于安装、快速配置的方法，可以加速 Python 中的数据分析，无需开发者升级其硬件基础设施或切换到其他编程语言。启动 Dask 作业所使用的语法与其他 Python 操作相同，因此几乎不需要重新写代码。Windows 10 中 Dask 的安装请参看 A.9 中的叙述。

ⁱ <https://www.dask.org/>

ⁱ <https://github.com/dask/dask>

Dask (2015)

8.2.1 Dask 架构和编程模式

Dask 由两部分组成 (图 8-1):

- 一个任务调度程序(Scheduler/Master): 用于构建任务图形, 协调、调度和监控针对跨 CPU 核心和计算机的交互式工作负载优化的任务。
- 多个可共享内存的计算节点集合(Workers): 借助 Dask 的运行库, Workers 的内存被统一管理了起来, 为用户程序的运行提供了一个虚拟的巨大的内存空间。

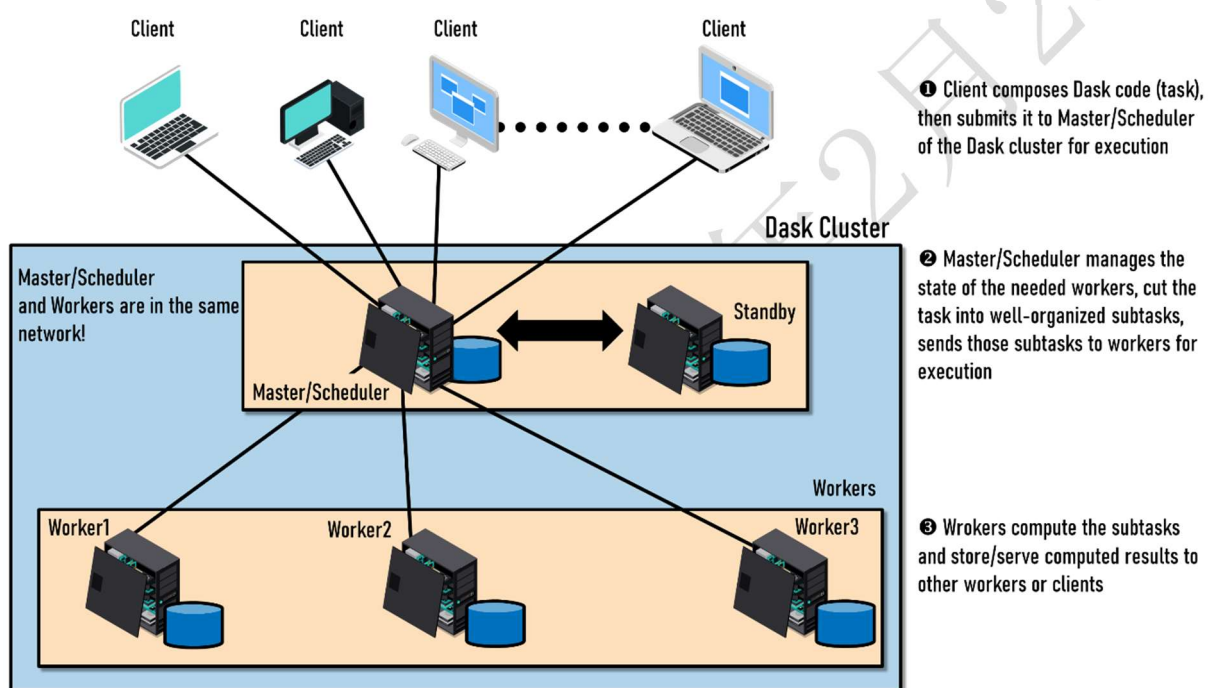


图 8-1 Dask 架构示意图

类似 Spark, 在分布式环境下的 Dask 程序, 与单独开发一个文件的 Dask 程序并无不同: Dask 框架为用户屏蔽了分布式环境下 Dask 程序运行的细节。图 8-2 展示了基于 Dask 编程的模式, 其中(a)表示 Dask 支持的数据类型(参看 8.1), 将不同来源的数据保存在分布式共享内存中用于后续(b)中定义的数据处理操作;(b)为用户编程时定义的基于图描述的任务处理流程——参看后面的例子;(c) 为 Dask 可以使用的计算资源的种类, 由不同的调度器(Scheduler)来对应。

Dask 支持多种调度器, 从单线程、多线程、多进程到本地分布式和集群分布式, 各种调度器在不

同情况下有不同的作用。主要分为两类：

1. 单机任务调度：单机任务调度在本地进程或线程池上提供基本功能。该调度是默认提供的。尽管它只能在单台机器上使用并且无法弹性伸缩，但使用起来简单且门槛很低。

2. 分布式任务调度：分布式任务调度功能更复杂，有更好的性能，但设置起来也需要更多的精力。它可以通过集群在本地运行或分布式执行。图 8-2 (c) 标识出了几种常见分布式系统调度器的 Dask 接口，如图 3-20 中展示过的 YARN、LSF、SLURM、Kubernetes (Kube/K8s)，以及面向云平台的 FargateCluster。

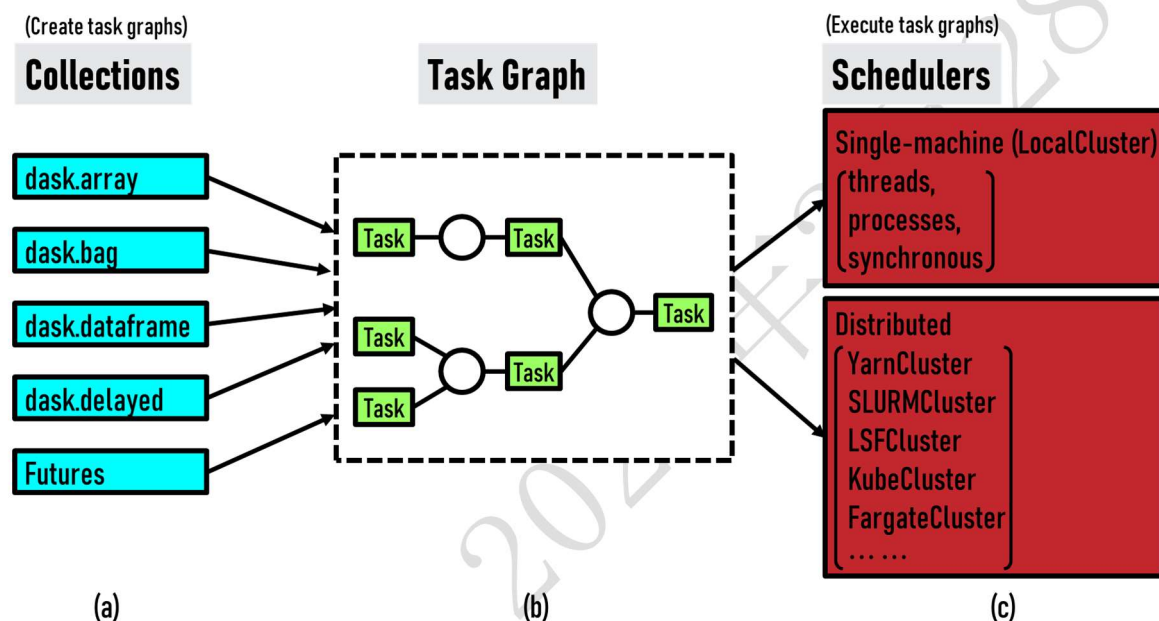


图 8-2 基于 Dask 的编程模式示意图

而基于 Dask 编程时，Dask 程序的结构则遵循(c)→(a)→(b)的顺序，即先选定适当的调度器(本地的或分布式的)，然后导入所需要处理的数据(使用 dask.array, dask.bag, dask.dataframe 等)，之后借助 Dask 的 API 描述要对数据进行处理的任务流程即可。

需要说明的是，Dask 提供的描述任务流程的 API，也类似 Spark 的 API——具有懒操作(Lazy Operator，即 Spark 中的 Transformation 操作)和执行操作(Action Operator)。

对 Dask 编程的讲解，请参看下面的例子。

需要说明的是，本章中的 Dask 代码的例子，主要是展示单机上的 Dask 编程。分布式环境下的 Dask 编程，请参看 D.4.3 中的 Dask 示例。

Dask (2015)

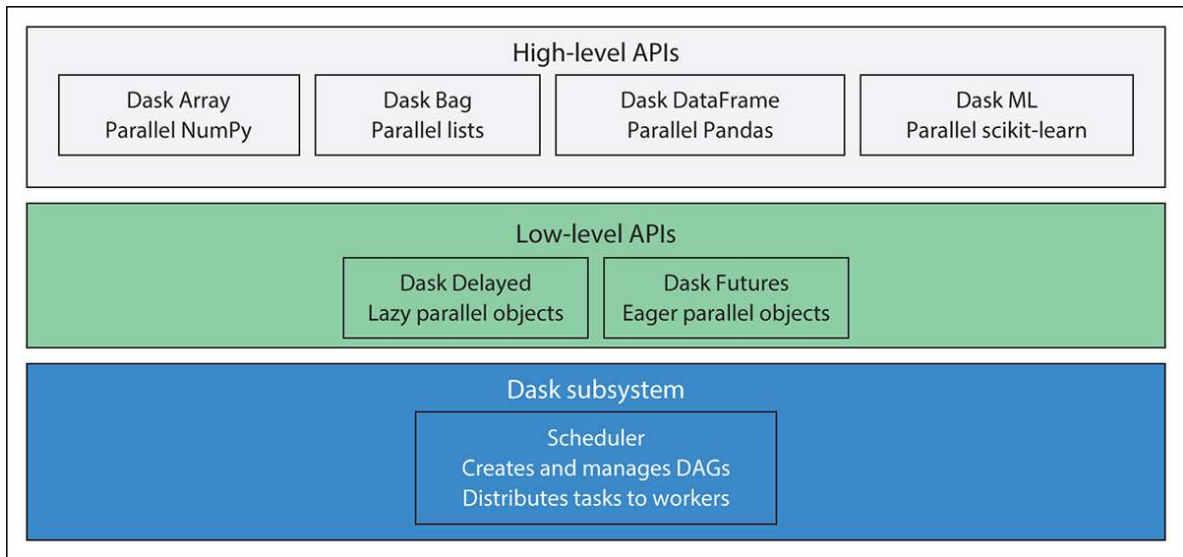


图 8-3 The components and layers than make up Dask

Dask

8.2.2 基本使用样例 –

8.2.2.1 装饰符 – `@dask.delayed`

D:\myCodes\HPCbook\08Other-Dask,Ray\10dask\daskTest2.py

代码 8-1 daskTest2.py

```
1. import time
2. import random
3. import dask
4. import sys
5.
6. @dask.delayed
7. def calcpfit(a, b):
8.     # time.sleep(random.random())
9.     return a + b
10.
11. @dask.delayed
12. def calcloss(a, b):
13.     # time.sleep(random.random())
14.     return a - b
```

```

15.
16. @dask.delayed
17. def calctotal(a, b):
18.     # time.sleep(random.random())
19.     return a + b
20.
21. if __name__ == "__main__":
22.     profit = calcprofit(10, 22)
23.     loss = calcloss(18, 3)
24.     total = calctotal(profit, loss)
25.
26.     print(total)
27.     total.visualize()
28.     print(total.compute())
29.     # print(dask.compute(total))

```

Task Graph

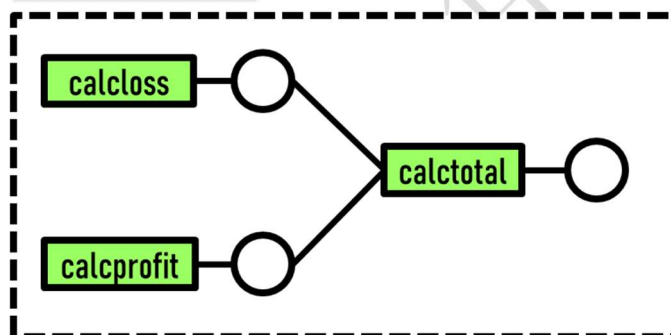


图 8-4 代码 8-1 代码定义的任务图

8.2.2.2 函数

D:\myCodes\HPCbook\08Other-Dask,Ray\10dask\daskTest2-delayed.py

代码 8-2 daskTest2-delayed.py

```

1. import time
2. import random
3. import sys
4. import dask
5. from dask import delayed

```

Dask (2015)

```
6.
7. def calcprofit(a, b):
8.     # time.sleep(random.random())
9.     return a + b
10.
11. def calcloss(a, b):
12.     # time.sleep(random.random())
13.     return a - b
14.
15. def calctotal(a, b):
16.     # time.sleep(random.random())
17.     return a + b
18.
19. if __name__ == "__main__":
20.     profit = delayed(calcprofit)(10, 22)
21.     loss = delayed(calcloss)(18, 3)
22.     total = delayed(calctotal)(profit, loss)
23.
24.     print(total)
25.     print(total.compute())
26.     # print(dask.compute(total))
```

比较代码 8-2 和代码 8-1, 可以清楚地看出二者的异同。

8.2.2.3 单机本地 Client 和 LocalCluster 的使用

代码 8-3 daskTest2-Client.py

```
1. import time
2. import random
3. import sys
4. import dask
5. from dask.distributed import Client
6.
7. @dask.delayed
8. def calcprofit(a, b):
```

```

9.     # time.sleep(random.random())
10.    return a + b
11.
12. @dask.delayed
13. def calcloss(a, b):
14.     # time.sleep(random.random())
15.     return a - b
16.
17. @dask.delayed
18. def calctotal(a, b):
19.     # time.sleep(random.random())
20.     return a + b
21.
22. if __name__ == "__main__":
23.     client = Client()
24.     # cluster.scale(3)
25.     profit = client.submit(calcprofit, 10, 22)
26.     loss = client.submit(calcloss, 18, 3)
27.     total = client.submit(calctotal, profit, loss)
28.     res = client.gather(total)
29.     print(res)
30.     print(dask.compute(res)[0])
31.     client.close()

```

代码 8-4 daskTest2-LocalClient.py

```

1. import time
2. import random
3. import sys
4. import dask
5. from dask.distributed import Client, LocalCluster
6.
7. @dask.delayed
8. def calcprofit(a, b):
9.     # time.sleep(random.random())
10.    return a + b
11.
12. @dask.delayed
13. def calcloss(a, b):
14.     # time.sleep(random.random())
15.     return a - b
16.
17. @dask.delayed

```


Dask (2015)

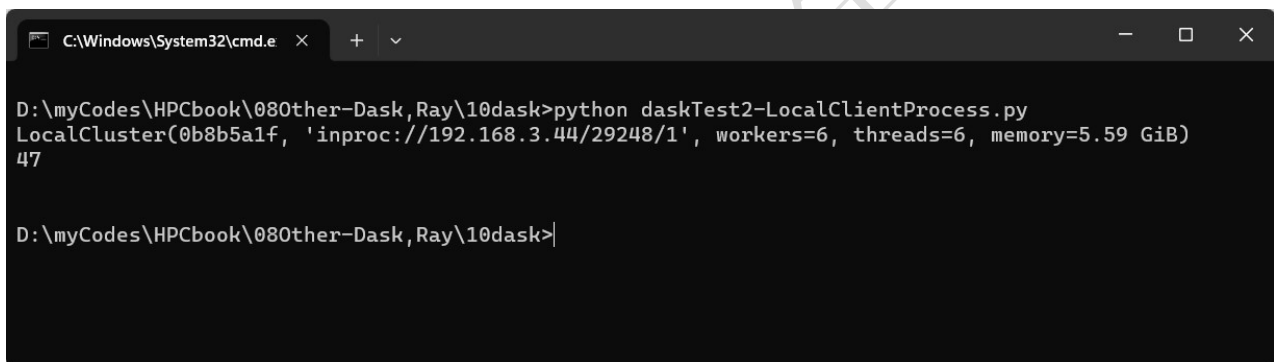
```
18. def calctotal(a, b):
19.     # time.sleep(random.random())
20.     return a + b
21.
22. if __name__ == "__main__":
23.     cluster = LocalCluster()
24.     client = Client(cluster)
25.     # cluster.scale(3)
26.     profit = client.submit(calcpfit, 10, 22)
27.     loss = client.submit(calcloss, 18, 3)
28.     total = client.submit(calctotal, profit, loss)
29.     res = client.gather(total)
30.     print(dask.compute(res)[0])
31.     print()
32.     client.close()
```

8.2.2.4 单机本地 LocalCluster 更详细的设置

代码 8-5 daskTest2-LocalClient.py

```
1. import time
2. import random
3. import sys
4. import dask
5. from dask.distributed import Client, LocalCluster
6.
7. @dask.delayed
8. def calcpfit(a, b):
9.     # time.sleep(random.random())
10.    return a + b
11.
12. @dask.delayed
13. def calcloss(a, b):
14.    # time.sleep(random.random())
15.    return a - b
16.
17. @dask.delayed
18. def calctotal(a, b):
19.    # time.sleep(random.random())
20.    return a + b
```

```
21.
22. if __name__ == "__main__":
23.     # cluster = LocalCluster(n_workers = 6)
24.     # cluster = LocalCluster(processes=True, n_workers=6,
25.     memory_limit='1GB', threads_per_worker=1)
26.     cluster = LocalCluster(processes=False, n_workers=6,
27.     memory_limit='1GB', threads_per_worker=1)
28.     client = Client(cluster)
29.     # cluster.scale(3)
30.     print(client.cluster)
31.     profit = client.submit(calcpfit,10, 22)
32.     loss = client.submit(calcloss,18, 3)
33.     total = client.submit(calctotal, profit, loss)
34.     res = client.gather(total)
35.     print(dask.compute(res)[0])
36.     print()
37.     client.close()
```



```
C:\Windows\System32\cmd.e x + v
D:\myCodes\HPCbook\080ther-Dask,Ray\10dask>python daskTest2-LocalClientProcess.py
LocalCluster(0b8b5a1f, 'inproc://192.168.3.44/29248/1', workers=6, threads=6, memory=5.59 GiB)
47
D:\myCodes\HPCbook\080ther-Dask,Ray\10dask>
```

图 8-5

代码行 25 表示选用线程 (即 “processes=False” 的作用)作为任务的计算节点, 一共使用 6 个计算节点(即 “n_workers=6” 的作用), 此时共计有 6 个线程 (即 “threads_per_worker=1” 和 “n_workers=6” 的共同作用)。

8.2.3 Dask 自己的 Array 的使用

Dask (2015)

Dask.array 是 Dask 提供的类似于 Numpy 的数组数据结构，它允许用户在大规模数据集上执行类似 Numpy 的操作。Dask.array 将数组拆分成多个小块，并使用延迟计算的方式来执行操作，从而实现并行计算。这使得 Dask.array 能够处理大型数据，同时充分利用计算资源。

```
import dask.array as da
x = da.random.random((100000, 100000), chunks=(1000, 1000))
# x = da.random.random((100000, 100000), chunks=(400, 400))

y = x + x.T
z = y[:, :2, 5000:].mean(axis=1)
# z.visualize()
print(z.compute())
```

D:\myCodes\HPCbook\08Other-Dask,Ray\10dask\daskTest4.py

8.2.4 Pi 的计算

8.3 单机版计算热传导的Dask 代码示意(1)

8.3.1 完整的代码

代码 8-6 heatEqu2DDaskImgsT.py

```
1. from __future__ import print_function
2. import numpy as np
3. import time
4.
5. import dask
6.
7. import matplotlib
```

```

8. matplotlib.use('Agg')
9. import matplotlib.pyplot as plt
10. plt.figure(dpi=300)
11. # Set the colormap
12. # plt.rcParams['image.cmap'] = 'BrBG'
13. plt.rcParams['image.cmap'] = 'jet' #you can try: colourMap =
    plt.cm.coolwarm
14. # Set colour interpolation and colour map
15. colorinterpolation = 100
16. colourMap = plt.cm.jet #you can try: colourMap = plt.cm.coolwarm
17.
18. # Set Dimension
19. lenX = lenY = 400 #we set it rectangular
20.
21. # Set meshgrid
22. X, Y = np.meshgrid(np.arange(0, lenX), np.arange(0, lenY))
23.
24. # Basic parameters
25. a = 0.1 # Diffusion constant
26. timesteps = 10000 # Number of time-steps to evolve system
27. image_interval = 1000 # Write frequency for png files
28.
29. # Grid spacings
30. dx = 0.01
31. dy = 0.01
32. dx2 = dx ** 2
33. dy2 = dy ** 2
34.
35. # For stability, this is the largest interval possible
36. # for the size of the time-step:
37. dt = dx2 * dy2 / (2 * a * (dx2 + dy2))
38.
39. # Boundary condition
40. Ttop = 100
41. Tbottom = 0
42. Tleft = 0
43. Tright = 30
44. # Initial guess of interior grid
45. Tguess = 0
46.
47. def init_fields():
48.     # Set array size and set the interior value with Tguess
49.     field = np.empty((lenX, lenY))
50.     field.fill(Tguess)
51.
52.     # Set Boundary condition

```

Dask (2015)

```

53.     field[(lenY-1):, :] = Ttop
54.     field[:, 1] = Tbottom
55.     field[:, (lenX-1):] = Tright
56.     field[:, :1] = Tleft
57.
58.     print("size is ",field.size)
59.     print(field,"\n")
60.
61.     return field
62.
63. @dask.delayed
64. def evolve(u, a, dt, dx2, dy2):
65.     u[1:-1, 1:-1] = u[1:-1, 1:-1] + a * dt * (
66.         (u[2:, 1:-1] - 2 * u[1:-1, 1:-1] +
67.          u[:-2, 1:-1]) / dx2 +
68.         (u[1:-1, 2:] - 2 * u[1:-1, 1:-1] +
69.          u[1:-1, :-2]) / dy2)
70.
71. def write_field(field, step):
72.     # plt.gca().clear()
73.     # plt.cla()
74.     plt.clf()
75.
76.     # Configure the contour
77.     plt.title("Contour of Temperature")
78.     plt.contourf(X, Y, field, colorinterpolation, cmap=colourMap)
79.     # Set Colorbar
80.     plt.colorbar()
81.     plt.axis('on')
82.     plt.savefig('heat_DaskBook_{0:03d}.png'.format(step))
83.
84. def main():
85.     field = init_fields()
86.     write_field(field, 0)
87.
88.     t0 = time.time()
89.     for m in range(1, timesteps + 1):
90.         dask.compute(evolve(field, a, dt, dx2, dy2))
91.         if m % image_interval == 0:
92.             write_field(field, m)
93.     t1 = time.time()
94.     print("Running time: {0}".format(t1 - t0))
95.
96. if __name__ == '__main__':

```

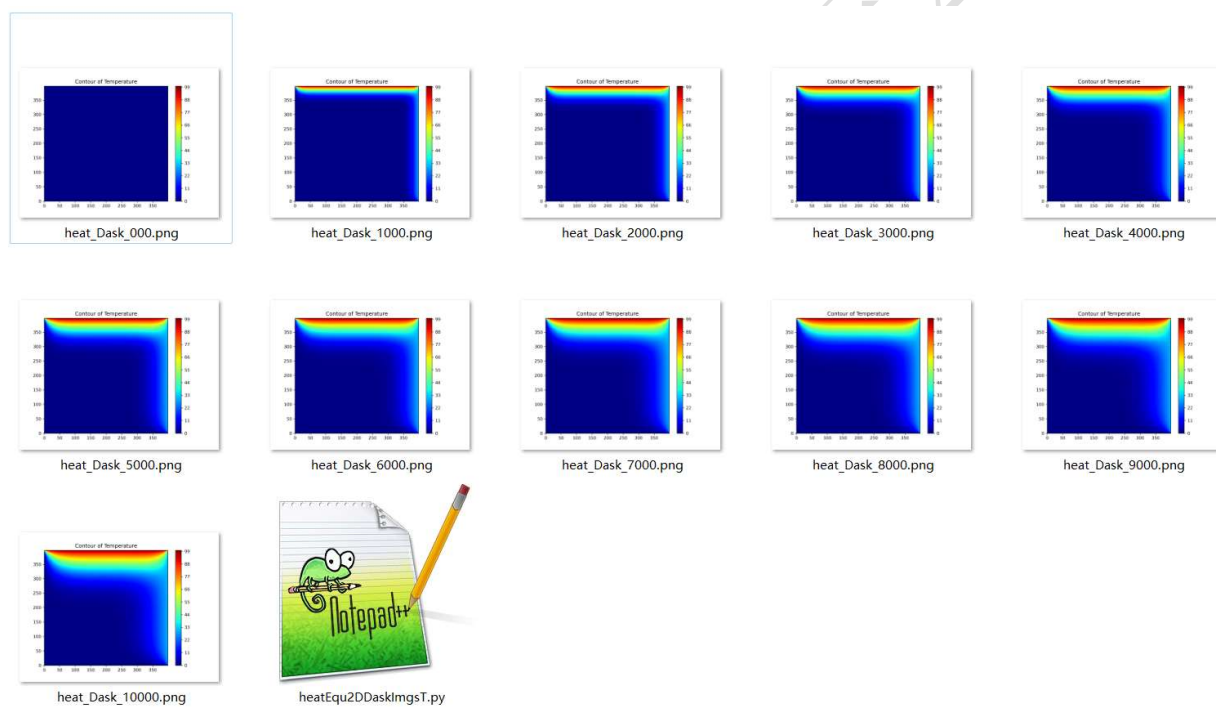
```
97.     main()
```

```
C:\Windows\System32\cmd.e  X  +  v

D:\myCodes\HPCbook\080ther-Dask,Ray>python heatEqu2DDaskImgsT.py
size is 160000
[[ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 ...
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0. 100. 100. ... 100. 100. 30.]]

Running time: 68.84738731384277

D:\myCodes\HPCbook\080ther-Dask,Ray>
```



8.3.2 代码讲解

8.4 单机版计算热传导的Dask 代码示意(2) – Cluster

所有大型的 Dask 集合变量(例如 Dask Array, Dask DataFrame 和 Dask Bag)以及细粒度的 API(例如 Delay 和 Future)都会生成任务图, 其中图中的每个节点都是常规的 Python 函数, 而节点之间的边缘是常规的 Python 对象, 由一个任务创建为输出, 并在另一任务中用作输入。

在 Dask 生成这些任务图之后, 它需要在并行硬件上执行它们。这就是任务调度。Dask 存在不同的任务调度, 每个调度程序将使用一个任务图并计算得到相同的结果, 但是它们的性能差别很大。

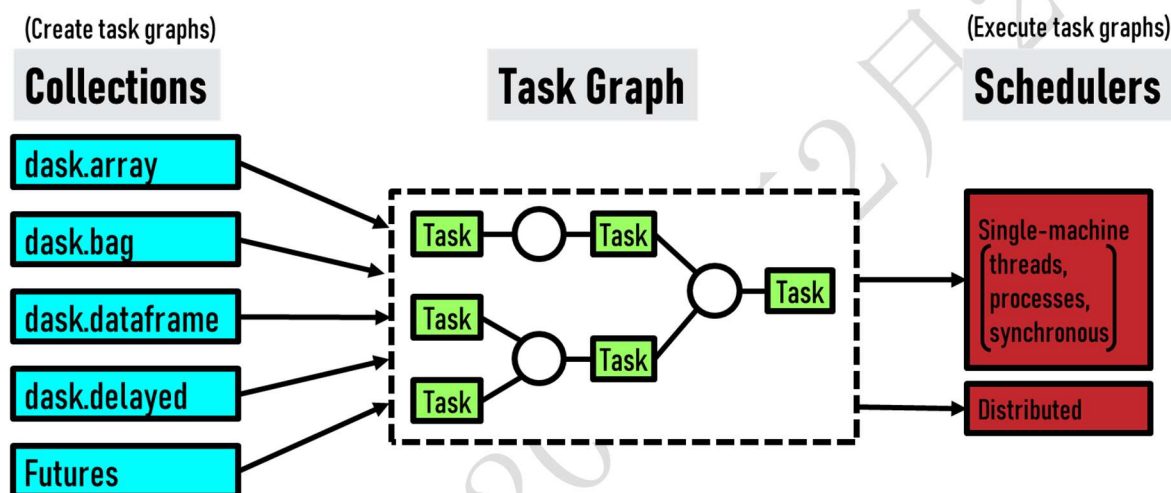


图 8-6 <https://docs.dask.org/en/stable/>

Dask 有两类任务调度:

1. 单机任务调度: 单机任务调度在本地进程或线程池上提供基本功能。该调度是默认提供的。尽管它只能在单台机器上使用并且无法弹性伸缩, 但使用起来简单且门槛很低。

2. 分布式任务调度: 分布式任务调度功能更复杂, 有更好的性能, 但设置起来也需要更多的精力。它可以通过集群在本地运行或分布式执行。

对于不同的计算任务来说, 使用不同的任务调度有更好的性能。

代码 8-7 heatEqu2DDaskLocalClientImgs.py

```
1. from __future__ import print_function
2. import numpy as np
```

```

3. import timeit
4.
5. import dask
6. dask.config.set({'logging.distributed': 'error'}) # 设定执行过程中显示信息的
   等级，用于屏蔽不必要的警告信息
7. from dask.distributed import Client, LocalCluster
8.
9. # cluster = LocalCluster(n_workers=4,
10.                          # threads_per_worker=1,
11.                          # memory_limit='16GB')
12. # client = Client(cluster)
13. import dask.array as dArray
14.
15. import matplotlib
16.
17. matplotlib.use('Agg')
18. import matplotlib.pyplot as plt
19. plt.figure(dpi=300)
20. # Set the colormap
21. # plt.rcParams['image.cmap'] = 'BrBG'
22. plt.rcParams['image.cmap'] = 'jet' #you can try: colourMap =
   plt.cm.coolwarm
23.
24. # Set colour interpolation and colour map
25. colorinterpolation = 100
26. colourMap = plt.cm.jet #you can try: colourMap = plt.cm.coolwarm
27.
28. # Basic parameters
29. a = 0.1 # Diffusion constant
30. timesteps = 10000 # Number of time-steps to evolve system
31. image_interval = 1000 # Write frequency for png files
32.
33. # Grid spacings
34. dx = 0.01
35. dy = 0.01
36. dx2 = dx ** 2
37. dy2 = dy ** 2
38.
39. # For stability, this is the largest interval possible
40. # for the size of the time-step:
41. dt = dx2 * dy2 / (2 * a * (dx2 + dy2))
42.
43. # Boundary condition
44. Ttop = 100
45. Tbottom = 0
46. Tleft = 0

```


Dask (2015)

```

47. Tright = 30
48.
49. # Initial guess of interior grid
50. Tguess = 0
51.
52. # Set Dimension
53. lenX = lenY = 400 #we set it rectangular
54. # Set meshgrid
55. X, Y = np.meshgrid(np.arange(0, lenX), np.arange(0, lenY))
56.
57. def init_fields():
58.     # Set array size and set the interior value with Tguess
59.     field = np.empty((lenX, lenY))
60.     field.fill(Tguess)
61.
62.     # Set Boundary condition
63.     field[(lenY-1):, :] = Ttop
64.     field[:, 1] = Tbottom
65.     field[:, (lenX-1):] = Tright
66.     field[:, :1] = Tleft
67.     return field
68.
69. def write_field(field, step):
70.     # plt.gca().clear()
71.     # plt.cla()
72.     plt.clf()
73.
74.     # Configure the contour
75.     plt.title("Contour of Temperature")
76.     plt.contourf(X, Y, field, colorinterpolation, cmap=colourMap)
77.     # Set Colorbar
78.     plt.colorbar()
79.     plt.axis('on')
80.     plt.savefig('heat_DaskLocalClientBook_{0:03d}.png'.format(step))
81.
82. @dask.delayed
83. def evolve(u, a, dt, dx2, dy2):
84.     u[1:-1, 1:-1] = u[1:-1, 1:-1] + a * dt * (
85.         (u[2:, 1:-1] - 2 * u[1:-1, 1:-1] +
86.          u[:-2, 1:-1]) / dx2 +
87.         (u[1:-1, 2:] - 2 * u[1:-1, 1:-1] +
88.          u[1:-1, :-2]) / dy2)
89.     return u
90.

```

```
91. def main():
92.     cluster = LocalCluster()
93.     client = cluster.get_client()
94.
95.     field = init_fields()
96.     write_field(field, 0)
97.
98.     print("size is ", field.size)
99.     print(field, "\n")
100.
101.     # Iteration (We assume that the iteration is convergence in maxIter
    = 500)
102.     print("Please wait for a moment")
103.
104.     starting_time = timeit.default_timer()
105.     future = []
106.     for iteration in range(0, timesteps+1):
107.         future = client.compute(evolve(field, a, dt, dx2, dy2))
108.         field = client.gather(future)
109.         if iteration % image_interval == 0:
110.             write_field(field, iteration)
111.
112.         print("Iteration finished. {} Seconds for Time
    difference:".format(timeit.default_timer() - starting_time))
113.
114.     cluster.close()
115.     client.close()
116.
117. if __name__ == '__main__':
118.     main()
```

Dask (2015)

```

C:\Windows\System32\cmd.e x + v
D:\myCodes\HPCbook\080ther-Dask,Ray>python heatEqu2DDaskLocalClientImgs.py
size is 160000
[[ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 ...
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0.  0.  0. ...  0.  0. 30.]
 [ 0. 100. 100. ... 100. 100. 30.]]

Please wait for a moment
C:\Users\lbkon\AppData\Local\Programs\Python\Python38\lib\site-packages\distributed\worker.py:2975: UserWarning: Large object
of size 1.22 MiB detected in task graph:
(array([[ 0.,  0.,  0., ...,  0.,  0., 30.], ... 0.0001, 0.0001])
Consider scattering large objects ahead of time
with client.scatter to reduce scheduler burden and
keep data on workers

    future = client.submit(func, big_data)    # bad

    big_future = client.scatter(big_data)    # good
    future = client.submit(func, big_future) # good
warnings.warn(
Iteration finished. 297.6761769 Seconds for Time difference:
D:\myCodes\HPCbook\080ther-Dask,Ray>

```

8.5 单机版计算热传导的Dask 代码示意(3)–大数据

测试下 Dask 单机可以支持多大的数据的计算

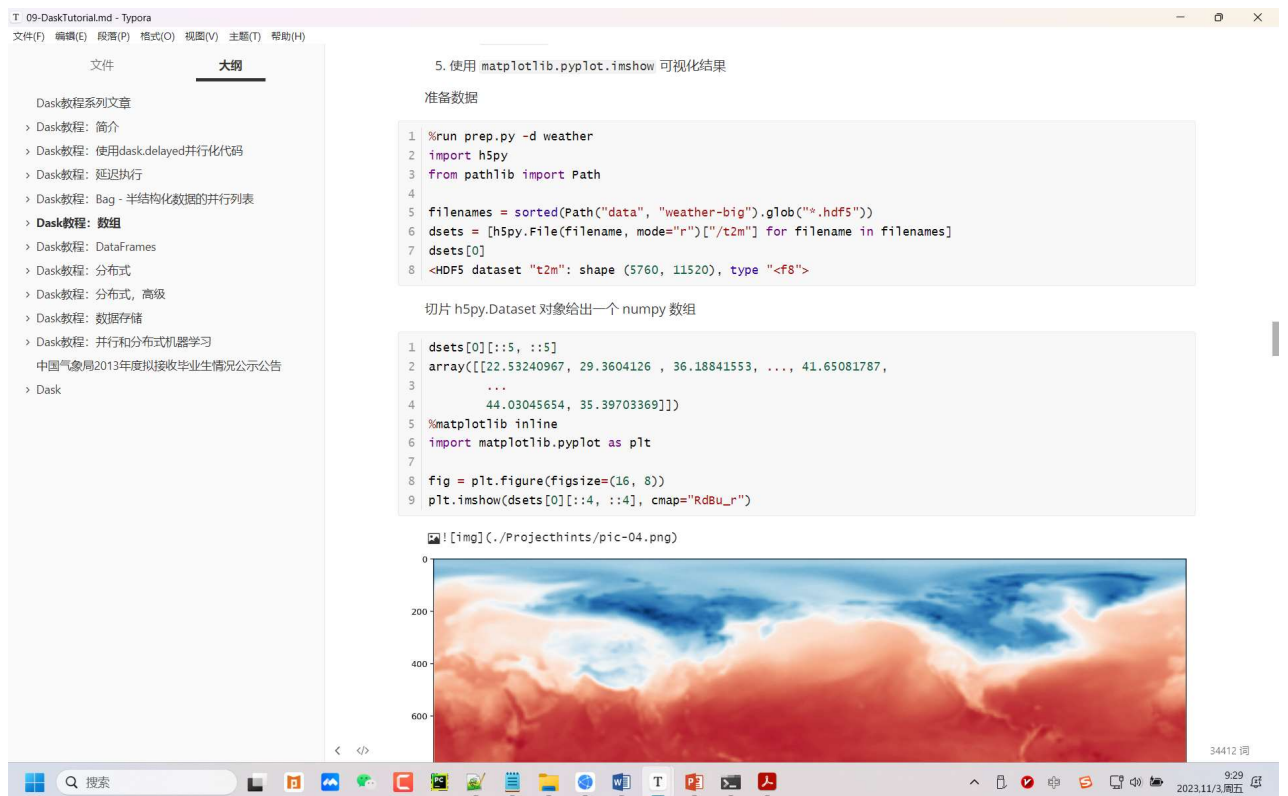
```

dsets[0][::5, ::5]
array([[22.53240967, 29.3604126 , 36.18841553, ..., 41.65081787,
        ...,
        44.03045654, 35.39703369]])

%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(16, 8))
plt.imshow(dsets[0][::4, ::4], cmap="RdBu_r")

```



这几天测试一下 2023 年 11 月 3 日 09:30:22

- 直接使用 Dask.array (而非前面一直沿用的 Numpy.array)
- 测试下单机上大数据的 Dask 计算和图像显示

8.6 本章小结

8.7 补充

Dask, Ray 和 Modin

Ray, and Dask have different scheduling strategies. All jobs for a cluster are managed by a central

Dask (2015)

scheduler used by Dask. Since Ray is decentralised, each computer has its own scheduler, allowing problems with scheduled tasks to be resolved at the level of the specific machine rather than the entire cluster. Ray lacks the rich high-level collections APIs that Dask offers (such as dataframes, distributed arrays, etc.).

Modin, on the contrary hand, rides over Dask or Ray. With the simple addition of a single line of code, `import modin.pandas as pd`, we can quickly expand our Pandas process with Modin. Although Modin makes an effort to parallelize a large portion of the Pandas API as feasible, Dask DataFrame sometimes doesn't scale the full Pandas API.

8.7.1 PGAS (Partitioned Global Address Space Languages: 分区全局地址空间语言)

之所以需要了解 PGAS，是因为

8.7.2 Dask 编程概览

8.7.2.1 Dask.array 的使用

8.7.2.2 Dask 的数据处理

```
import dask.array as da

x = da.random.random((100000, 100000), chunks=(1000, 1000))

y = x + x.T

z = y[:, :2, 5000:].mean(axis=1)

# z.visualize()

print(z.compute())
```

```
D:\myCodes\HPCbook\080ther-Dask,Ray\10dask>python daskTest4.py
[0.99849011 0.99967519 1.00012835 ... 0.99960796 1.00081094 0.99894543]

D:\myCodes\HPCbook\080ther-Dask,Ray\10dask>python daskTest4.py
|
```

在计算机内存较小的情况下，比如使用个人电脑等，读者可能希望在比内存更大的数据集上进行训练。Dask-ML 包实现了可以在大于计算机内存的 Dask 数组或数据帧上进行机器学习训练。

```
python -m pip install dask-ml
```

[143]

8.7.3 其它类似框架

<https://blog.csdn.net/kittyzc/article/details/105731219>

8.7.3.1 Celery (2009)

A task queue based on distributed message passing.

First commit in 2009 as "crunchy" (adj. 硬脆的，松脆的；<美，非正式> 政治上自由主义且注重环保的)

8.7.3.2 Dispy (2012)

足够简单，支持异构，真心好用ⁱ。

pip install dispy 即可安装完成。

这里用一台电脑做演示。

在服务端起节点服务：

```
python anaconda3/bin/dispynode.py --clean
```

ⁱ <http://dispy.org/>

Dask (2015)

在客户端运行下面的脚本

```
def compute(n):
    import time, socket
    time.sleep(n)
    host = socket.gethostname()
    return (host, n)

if __name__ == '__main__':
    # executed on client only; variables created below, including modules imported,
    # are not available in job computations
    import dispy, random
    # distribute 'compute' to nodes; 'compute' does not have any dependencies
    (needed from client)
    cluster = dispy.JobCluster(compute, ip_addr='客户端 ip 地址')
    # run 'compute' with 20 random numbers on available CPUs
    jobs = []
    for i in range(20):
        job = cluster.submit(random.randint(5,20))
        job.id = i # associate an ID to identify jobs (if needed later)
        jobs.append(job)
    # cluster.wait() # waits until all jobs finish
    for job in jobs:
        host, n = job() # waits for job to finish and returns results
        print('%s executed job %s at %s with %s' % (host, job.id, job.start_time,
n))

        # other fields of 'job' that may be useful:
        # job.stdout, job.stderr, job.exception, job.ip_addr, job.end_time
    cluster.print_status() # shows which nodes executed how many jobs etc.
```

结果如下:

```

ok-Pro.local executed job 0 at 1587791693.5012422 with 11
ok-Pro.local executed job 1 at 1587791693.501492 with 13
ok-Pro.local executed job 2 at 1587791693.501193 with 11
ok-Pro.local executed job 3 at 1587791693.501162 with 14
ok-Pro.local executed job 4 at 1587791693.501413 with 17
ok-Pro.local executed job 5 at 1587791693.501395 with 5
ok-Pro.local executed job 6 at 1587791693.501366 with 5
ok-Pro.local executed job 7 at 1587791693.50128 with 19
ok-Pro.local executed job 8 at 1587791698.5221078 with 10
ok-Pro.local executed job 9 at 1587791698.525928 with 10
ok-Pro.local executed job 10 at 1587791704.5162508 with 10
ok-Pro.local executed job 11 at 1587791704.520787 with 5
ok-Pro.local executed job 12 at 1587791706.528142 with 10
ok-Pro.local executed job 13 at 1587791707.512098 with 13
ok-Pro.local executed job 14 at 1587791708.533039 with 11
ok-Pro.local executed job 15 at 1587791708.536777 with 13
ok-Pro.local executed job 16 at 1587791709.532337 with 15
ok-Pro.local executed job 17 at 1587791710.5235431 with 11
ok-Pro.local executed job 18 at 1587791712.519975 with 8
ok-Pro.local executed job 19 at 1587791714.529252 with 7

```

Node	CPUs	Jobs	Sec/Job	Node Time Sec	Sent	Rcvd
ook-P	8	20	10.9	218.2	3.5 K	5.1 K

<https://sourceforge.net/projects/dispy/files/>

	Open Source Software	Business Software	Resources
dispy3-3.7.tar.gz	2013-07-16	88.6 kB	0 ⓘ
dispy-3.6.tar.gz	2012-10-24	86.2 kB	0 ⓘ
dispy-3.6.zip	2012-10-24	96.9 kB	0 ⓘ
dispy3-3.6.zip	2012-10-24	97.2 kB	0 ⓘ
dispy3-3.6.tar.gz	2012-10-24	86.6 kB	0 ⓘ
readme	2012-10-21	373 Bytes	0 ⓘ
Totals: 174 Items		42.5 MB	5

dispy is a Python framework for parallel execution of computations by distributing them across multiple processors on a single machine (SMP), among many machines in a cluster or large clusters of nodes. The computations can be Python functions or standalone programs.

Files dispy-* are for use with Python versions 2.7+ and dispy3-* are for use with Python versions 3.1+.

Source: readme, updated 2012-10-21

8.7.3.3 SCOOP (2012)

Scalable Concurrent Operations in Python (SCOOP) ^{i[144]} is a Python module to distribute concurrent

ⁱ [https://en.wikipedia.org/wiki/Python_SCOOP_\(software\)](https://en.wikipedia.org/wiki/Python_SCOOP_(software))

SCOOP was initiated by Yannick Hold and Marc Parizeau at the Computer Vision and Systems Laboratory of [Université Laval](#). It is an iterative step over the now deprecated DTM module of the DEAP framework for developing [evolutionary algorithm](#). While DTM used [MPI](#) for its communications, SCOOP uses instead ØMQ.

Dask (2015)

tasks (called Futures) on heterogeneous computational nodes. Its architecture is based on the ØMQ package, which provides a way to manage Futures between the distributed systems. The main application of SCOOP resides in scientific computing that requires the execution of many distributed tasks using all the computational resources available[103].

8.7.3.4 Ipyparallel (2015)

大多数时候要用 python 做数据处理的时候仅仅是因为单机资源不够了, 想把其它机器连接起来一起使用处理或是计算, 或者说是要经常在不同机器上运行计算程序处理一部分需求之后手动合并。

ipyparallel 是一个 python 包ⁱ, 直接安装 `pip install ipyparallel` 就可以了, 当然要在你需要连接成集群的机器上都安装上这个工具。

无论是 Mac 还是 Linux, 配置文件都在 `~/.ipython/profile_default/security/` 下面。在 master 节点上启动服务

```
ipcontroller --ip=""
```

首先设置所有机器免密登录 master 节点, 以及 master 节点免密登录所有机器。可以参考这篇:
https://blog.csdn.net/snail_bing/article/details/81772982

然后用脚本将生成的 `ipcontroller-engine.json` 文件复制到所有的计算节点上, 注意将第一列的 ssh 填写为 master 的 ip 地址

然后用脚本在每台机器上执行计算引擎, master 节点也是可以启动 engines:

```
ipcluster engines --n=8
```

然后启动 jupyter notebook 即可:

```
from ipyparallel import Client
c = Client()
v = c[:]
print len(v)
```

`v.map_sync` 是一个同步的并行计算函数, 它会把计算函数发送到不同的计算引擎上去计算, 同时负责把结果返回整理出来, 它的第二个参数是一个由计算函数所需要的参数组成的列表, 如果参数还

ⁱ <https://ipyparallel.readthedocs.io/en/latest/>

ⁱ <https://github.com/ipython/ipyparallel>

有，可以继续在后面增加列表，`map_sync` 会把参数一组一组分别从每个列表相应位置取出来传递给计算函数，之后把计算结果收集成一个列表。我们的计算引擎的个数是 64 个，我们发的计算函数实际上由参数展开是 800 个，我们不需要去担心怎么分配给每个计算引擎多少任务，怎么收集计算好的结果，直接等最终结果就好了:)。

https://blog.csdn.net/weixin_30556161/article/details/99597798

```
from numpy.random import rand
def sample(n):
    return (rand(n)**2+rand(n)**2<=1).sum()
n = 1000000
pi = 4*sample(n)*1.0/n
print(pi)
```

程序改写成并行版本:

```
from ipyparallel import *
rc = Client()
dview = c[:]

with dview.sync_imports():
    from numpy.random import rand

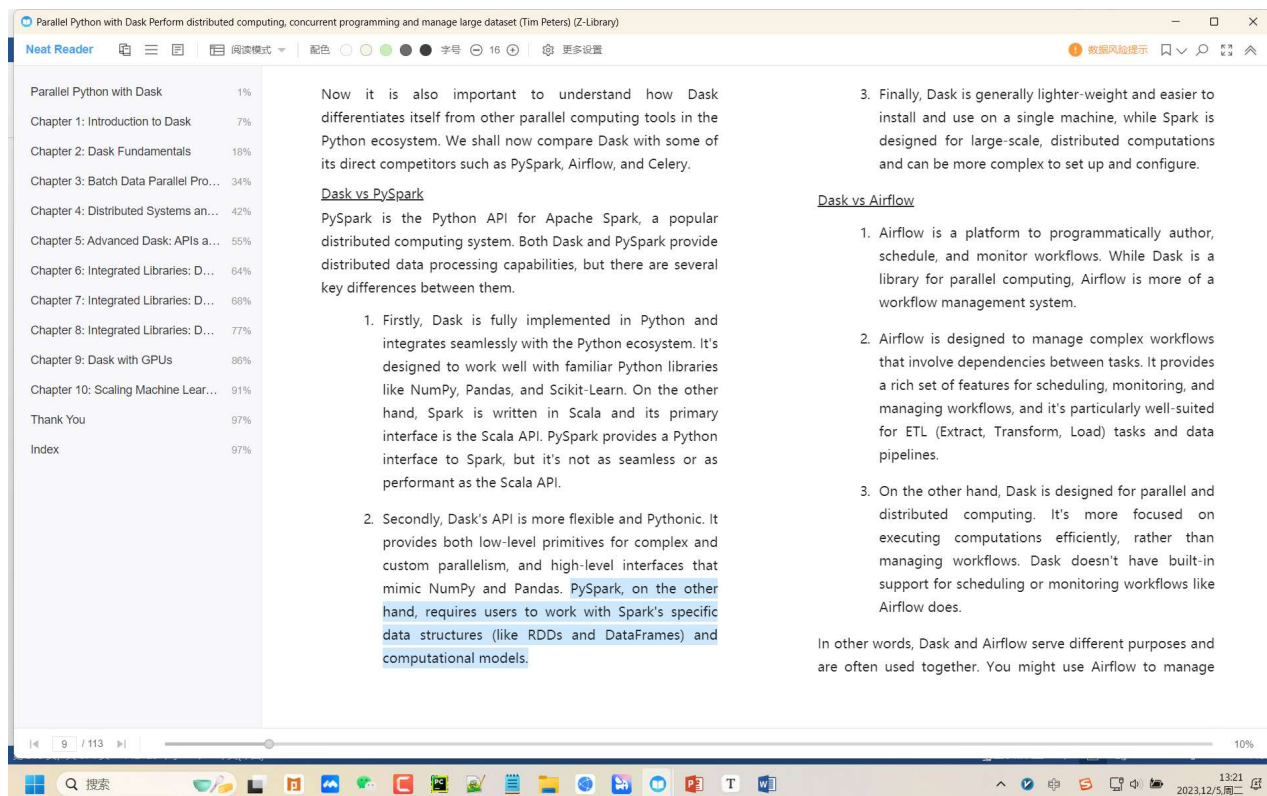
def sample(n):
    return (rand(n)**2+rand(n)**2<=1).sum()

n = 1000000
list=[n]*len(dview)
#list 由 len(dview) 个 n 构成, 正好每一个 engine 一个
pi = 4*sum(dview.map_sync(sample,list))*1.0/(n*len(dview))
print(pi)
```

`Ipyparallel` 是另一个专注于多核处理和任务分发的系统，专门用于跨集群并行地执行 Jupyter notebook 代码。已经在 Jupyter 工作的项目和团队可以立即开始使用 `Ipyparallel`。

`Ipyparallel` 支持许多并行代码的方法。在简单的一端上是 `map`，它将任何函数应用于序列，并在可用节点之间均匀地分割工作。对于更复杂的工作，可以将特定函数装饰为始终远程运行或并行运行。

Dask (2015)



and schedule your workflows, and use Dask to perform the actual computations within those workflows.

Dask vs Celery

Celery is an asynchronous task queue/job queue based on distributed message passing. It's focused on real-time operation, but supports scheduling as well. Celery is a great tool for background tasks, especially for web applications where you need to perform tasks outside the usual request-response cycle. It's also good for simple distributed computing tasks, where you need to distribute tasks across multiple workers.

Dask, on the other hand, is more suited for complex, large-scale computations. It provides more advanced features for parallel computing, like task dependencies and lazy evaluation. Dask also integrates well with the Python scientific stack, which makes it a good choice for data-intensive computations.

Dask, PySpark, Airflow, and Celery, they all provide capabilities for distributed computing, they each have their own strengths and are suited to different types of tasks. Dask's strength lies in its seamless integration with the Python ecosystem, its flexible and Pythonic API, and its capabilities for large-scale, data-intensive computations.

8.7.3.5 Ray (2017)

2023 年 7 月 19 日 01:11:04

觉得先不写 Ray 吧！

Ray 是加州大学伯克利分校的另一个项目，其使命是“简化分布式计算”^[145]。Ray 由两个主要组件组成——Ray Core(一个分布式计算框架)和 Ray Ecosystem(广义上说，Ray Ecosystem 是与 Ray 打包的一系列特定于任务的库)(例如，Ray Tune——一个超参数优化框架，用于分布式深度学习的 RaySGD，用于强化学习的 RayRLib 等等)。

Ray 类似于 Dask，因为它允许用户在台多机器上以并行的方式运行 Python 代码。然而，与 Dask 不同的是，Ray 并没有试图模拟 NumPy 和 Pandas 接口——它的主要设计目标不是临时替代 Data Science 的工作负载，而是提供一个通用的底层框架来并行化 Python 代码。这使得它更像一个通用的集群和并行框架，可以用来构建和运行任何类型的分布式应用程序。由于 Ray Core 的架构方式，它经常被认为是一个用于构建框架的框架。还有越来越多的项目与 Ray 集成，以利用加速 GPU 和并行计算。

Ray 分布式计算框架介绍

<https://zhuanlan.zhihu.com/p/111340572>

Dask 的目标是为了弥补 Python 在数据科学上的不足。而 Ray 的出发点是为了加速机器学习的调优和训练的速度。Ray 除了基础的计算平台，还包括 Tune(超参数调节)和 RLlib(增强学习)

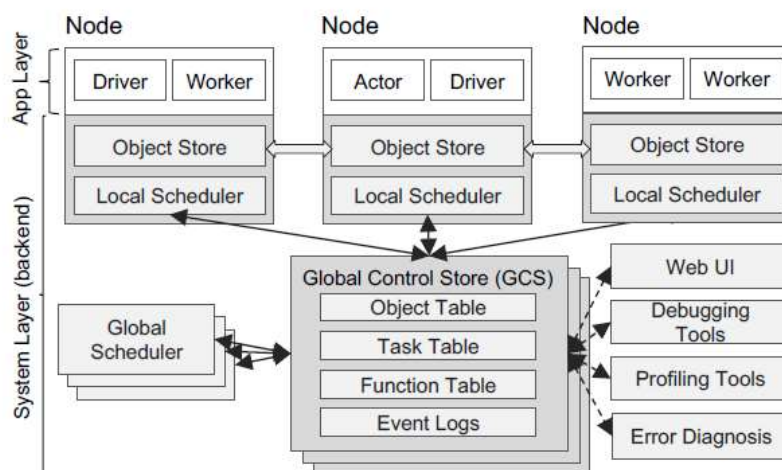


图 8-7 Ray 的运行架构示意图

让事情更复杂的是，还有一个 Dask-on-Ray 项目，它允许你运行 Dask 工作流而不使用[Dask 分布

Dask (2015)

式调度器](<https://distributed.dask.org/en/latest/>)。为了更好地理解 Dask-on-Ray 试图填补的空白, 我们需要看看 Dask 框架的核心组件。其中包含集合抽象(DataFrames、数组等)、任务图(一个 DAG, 它代表了一组类似于 Apache Spark DAG 的操作)和调度器(负责执行 Dask 图)。分布式调度器是 Dask 中可用的调度器之一, 它负责协调分布在多台机器上的多个工作进程的操作。这个调度器非常棒, 因为它易于设置、保持最小的延迟、允许点对点数据共享, 并支持比简单的 map-reduce 链复杂得多的工作流。另一方面, 分布式调度器也不是没有缺陷。它的一些缺点包括:

- 它是一个单点 - 分布式调度程序没有高可用性机制, 因此, 如果失败, 则需要重置整个群集, 并且所有过程中的任务都会丢失。

- 它是用 Python 编写的, 这使得它易于安装和调试, 但它也引入了通常与 Python 一起使用的标准性能考虑因素。

- Client API 在设计时考虑到了数据科学家, 并没有针对来自高可用性生产基础设施的调用进行定制(例如, 它假设客户端是长期存在的, 可能通过 Jupyter 会话在集群上操作)

- 它对有状态执行提供了最低限度的支持, 因此很难实现容错管道。

- 它可能会成为瓶颈, 而且无法自行扩展

相比之下, 容错和性能是深深嵌入在 Ray 调度器设计中的原则。它是完全去中心化的(没有瓶颈)、提供更快数据共享(通过 Apache Plasma)、单个调度器是无状态的(容错)、支持有状态的参与者等等。这使得在 Ray 集群上运行 Dask 任务具有很大吸引力是非常可以理解的, 这也是搞 [Dask-on-Ray](#) 调度器的原因。对 Dask-on-Ray 项目的深入研究超出了这篇博文的范围, 但是如果你有兴趣更深入地比较两者的性能, 请随意查看 Anyscale 所做的[内存管理和性能基准测试](#)。

混合框架的重要性已经通过集成库的出现得到了明显体现, 这些集成库使框架间的通信更加精简流线化。比如, [Spark on Ray](#) 正是这样做的——它“结合你的 Spark 和 Ray 集群, 使用 PySpark API 轻松地进行大规模数据处理, 并使用 TensorFlow 和 PyTorch 无缝地使用这些数据来训练你的模型”。还有 [Ray on Spark](#) 项目它使我们能够在 Apache Hadoop/Yarn 上运行 Ray 程序。这种方法已经在实际的生产工作负载中成功地测试过了。例如, Uber 的机器学习平台 [Michelangelo](#) 定义了 Ray Extiator API, 该 API 为终端用户抽象了在 Spark 和 Ray 之间移动的过程。Uber Engineering 最近发布的一篇文章详细介绍了这一点, 其中介绍了一种涉及 Spark 和 XGBoost on Ray 的[分布式培训架构](#)。

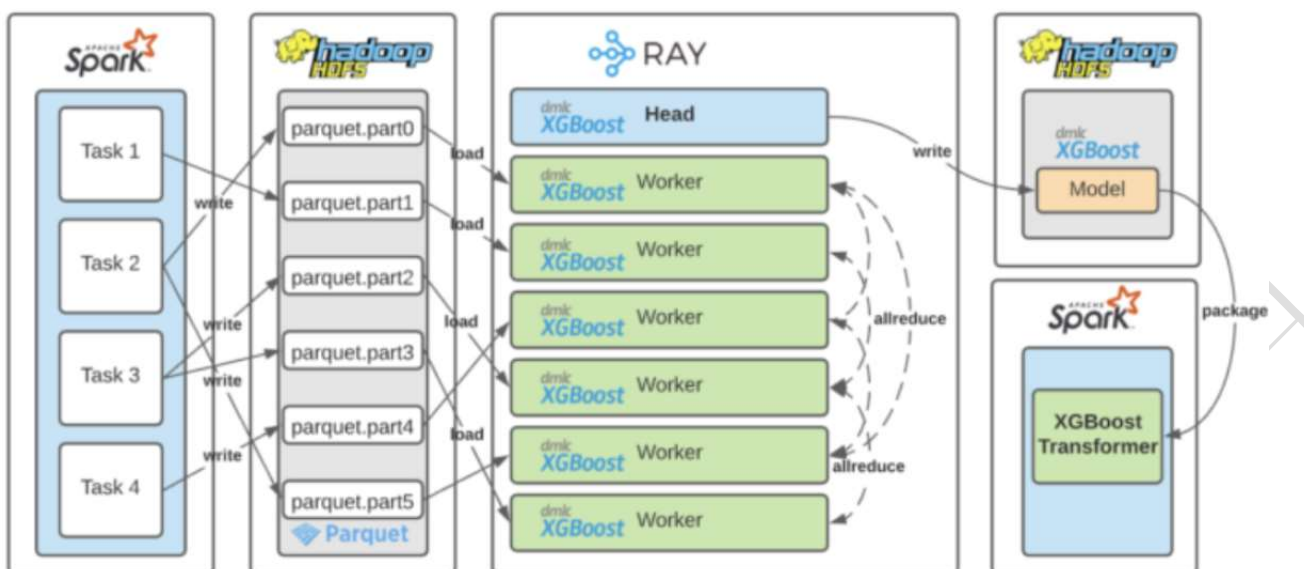


图 8-8 从 Spark (DataFrames) 到 Ray (分布式训练) 再到 Spark (转换) 的高层概述。Ray 评估器将这种复杂性封装在 Spark 评估器接口中。图片来源: <https://eng.uber.com/elastic-xgboost-ray/>

代码 8-8 heatEqu2DRayImgsT.py

```

1. from __future__ import print_function
2. import numpy as np
3. import time
4.
5. import ray
6. ray.init()
7.
8. import matplotlib
9. matplotlib.use('Agg')
10. import matplotlib.pyplot as plt
11. # Set the colormap
12. plt.rcParams['image.cmap'] = 'jet' # you can try: colourMap =
    plt.cm.coolwarm
13. # Set colour interpolation and colour map
14. colorinterpolation = 100
15. colourMap = plt.cm.jet # you can try: colourMap = plt.cm.coolwarm
16.
17. # Basic parameters
18. a = 0.1 # Diffusion constant
19. timesteps = 10000 # Number of time-steps to evolve system
20. image_interval = 1000 # Write frequency for png files
21.
22. # Grid spacings
23. dx = 0.01

```

Dask (2015)

```
24.dy = 0.01
25.dx2 = dx ** 2
26.dy2 = dy ** 2
27.
28.# For stability, this is the largest interval possible
29.# for the size of the time-step:
30.dt = dx2 * dy2 / (2 * a * (dx2 + dy2))
31.
32.# Boundary condition
33.Ttop = 100
34.Tbottom = 0
35.Tleft = 0
36.Tright = 30
37.# Initial guess of interior grid
38.Tguess = 0.25
39.
40.# Set Dimension
41.lenX = lenY = 400 # we set it rectangular
42.# Set meshgrid
43.X, Y = np.meshgrid(np.arange(0, lenX), np.arange(0, lenY))
44.
45.def init_fields():
46.    # Set array size and set the interior value with Tguess
47.    field = np.empty((lenX, lenY))
48.    field.fill(Tguess)
49.
50.    # Set Boundary condition
51.    field[(lenY - 1):, :] = Ttop
52.    field[:, 0] = Tbottom
53.    field[:, (lenX - 1):] = Tright
54.    field[:, 1] = Tleft
55.
56.    print("size is ", field.size)
57.    print(field, "\n")
58.
59.    return field
60.
61.def write_field(field, step):
62.    # plt.gca().clear()
63.    plt.cla()
64.    plt.clf()
65.    plt.figure(dpi=300)
66.
67.    # Configure the contour
```



```

68. plt.title("Contour of Temperature")
69. plt.contourf(X, Y, field, colorinterpolation, cmap=colourMap)
70. # Set Colorbar
71. plt.colorbar()
72. plt.axis('on')
73. plt.savefig('heatRay_{0:03d}.png'.format(step))
74.
75. # Turn evolve into a Ray task.
76. @ray.remote
77. def evolve(u_in, a, dt, dx2, dy2):
78.     u = np.copy(u_in)
79.     u[1:-1, 1:-1] = u[1:-1, 1:-1] + a * dt * (
80.         (u[2:, 1:-1] - 2 * u[1:-1, 1:-1] +
81.          u[:-2, 1:-1]) / dx2 +
82.         (u[1:-1, 2:] - 2 * u[1:-1, 1:-1] +
83.          u[1:-1, :-2]) / dy2)
84.     # print(u)
85.     return u
86.
87. def main():
88.     field = init_fields()
89.     write_field(field, 0)
90.
91.     t0 = time.time()
92.     for m in range(1, timesteps + 1):
93.         # for m in range(1, 3 + 1):
94.             future = []
95.             # Start evolving the field in parallel.
96.             future = evolve.remote(field, a, dt, dx2, dy2)
97.             # print("future - ", future)
98.             field = ray.get(future)
99.
100.            # Get the results as they become available.
101.            if m % image_interval == 0:
102.                # Block until the results are ready.
103.                # field = ray.get(field)
104.                write_field(field, m)
105.                future = []
106.            t1 = time.time()
107.            print("Running time: {0}".format(t1 - t0))
108.
109.
110. if __name__ == '__main__':
111.     main()

```


Dask (2015)

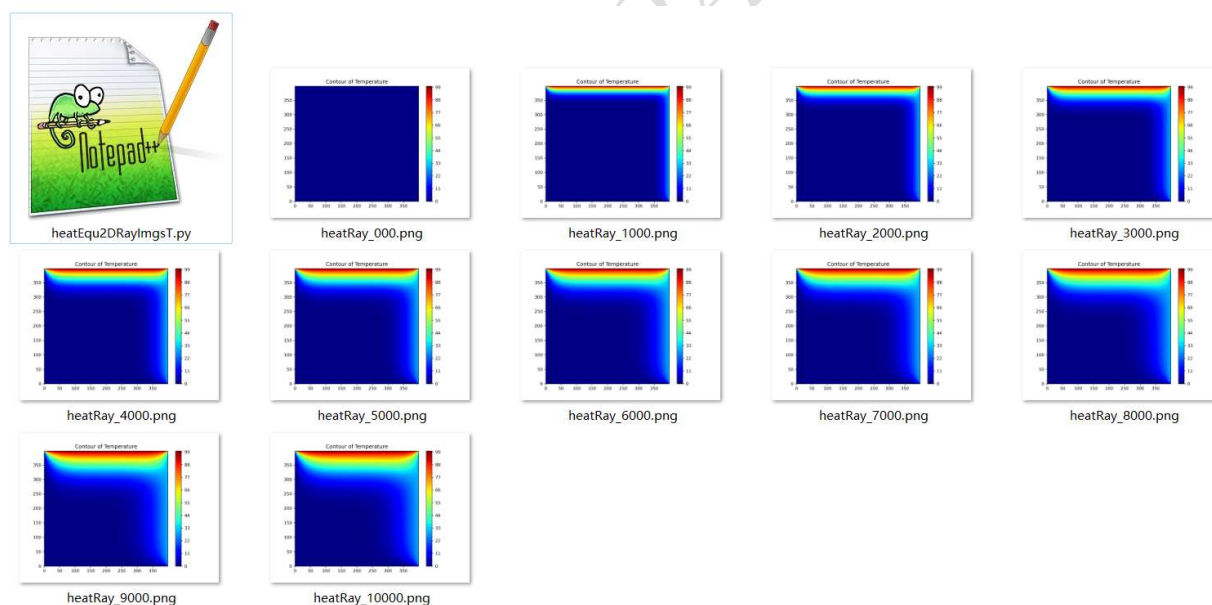
```
C:\Windows\System32\cmd.e  X  +  v

Microsoft Windows [版本 10.0.22621.1992]
(c) Microsoft Corporation。保留所有权利。

D:\myCodes\HPCbook\080ther-Dask,Ray>python heatEqu2DRayImgsT.py
2023-07-17 14:41:26,408 INFO worker.py:1625 -- Started a local Ray instance.
size is 160000
[[ 0.    0.    0.    ...  0.    0.    30. ]
 [ 0.    0.25  0.25  ...  0.25  0.25  30. ]
 [ 0.    0.25  0.25  ...  0.25  0.25  30. ]
 ...
 [ 0.    0.25  0.25  ...  0.25  0.25  30. ]
 [ 0.    0.25  0.25  ...  0.25  0.25  30. ]
 [ 0.   100.  100.   ... 100.  100.  30. ]]

Running time: 108.91146492958069

D:\myCodes\HPCbook\080ther-Dask,Ray>
```



8.7.3.6 阿里巴巴的 Mars (2018)

在 2018 年 9 月的杭州云栖大会上, 阿里巴巴就公布了一个基于张量的统一分布式计算框架——Mars——的开源计划。Mars 突破了现有大数据计算引擎的关系代数为主的计算模型, 将分布式技术

引入科学计算/数值计算领域，极大地扩展了科学计算的计算规模和效率。使用 Mars 进行科学计算，不仅使得完成大规模科学计算任务从 MapReduce 实现上千行代码降低到 Mars 数行代码，更在性能上有大幅提升。

感兴趣的读者，可以移步 Mars 网站去学习和了解ⁱ。

8.7.3.7 Charm4py (2018)

<https://charm4py.readthedocs.io/en/latest/>

Charm4py is a distributed computing and parallel programming framework for Python, for the productive development of fast, parallel and scalable applications. It is built on top of [Charm++](#), an adaptive runtime system that has seen extensive use in the scientific and high-performance computing (HPC) communities across many disciplines, and has been used to develop applications like [NAMD](#) that run on a wide range of devices: from small multi-core devices up to the largest supercomputers.

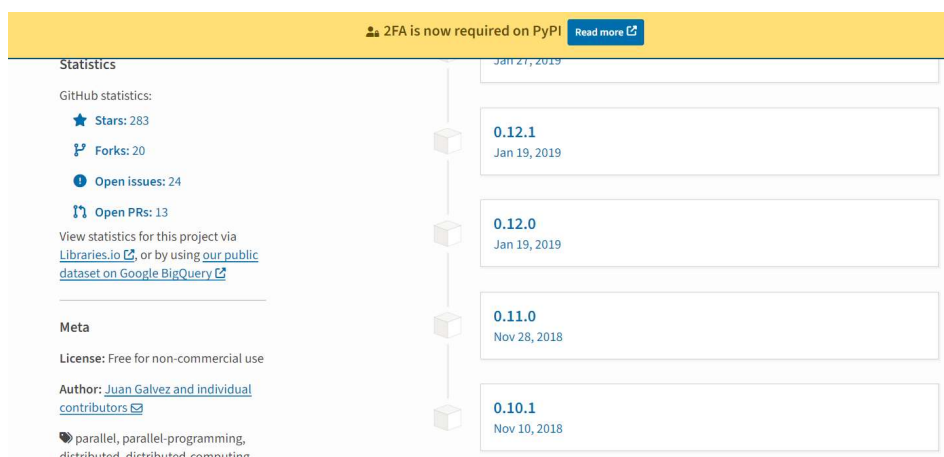
<https://github.com/UIUC-PPL/charm4py>

The screenshot shows the GitHub repository for `UIUC-PPL/charm4py`. The repository is public and has 281 stars and 20 forks. The main branch is `main`. The repository structure includes folders like `.github/workflows`, `charm4py`, `charmrun`, `docs`, `examples`, and `tests`. The commit history shows a recent commit by `matthiasdiener` and `evan-charmworks` titled "Limit Cython version, fix handling of `args.slice.value` in `'is_tag_c...`" 4 months ago. The right sidebar shows repository statistics and links to the README, license, and activity.

<https://pypi.org/project/charm4py/#history>

ⁱ <https://mars-project.readthedocs.io/en/latest/index.html>

Dask (2015)



8.7.3.8 Pygion (2019)

论文^[146]。

新加坡 (D:) > Local > ++ 写书 > 18 高性能计算 > HPCBook-MD > materials > 10-Other Frameworks >

名称	修改日期	类型	大小
并行计算的编程模型	2023,11/14,周二 11:29	文件夹	
A Newcomer In The PGAS World - UP...	2023,11/14,周二 13:17	Adobe Acrobat ...	501 KB
Advanced MPI Capabilities-2014.pdf	2023,11/14,周二 12:27	Adobe Acrobat ...	2,882 KB
bao_pgas_upc.pdf	2023,11/14,周二 13:20	Adobe Acrobat ...	842 KB
FSHMEM-2022.pdf	2023,11/14,周二 9:43	Adobe Acrobat ...	1,355 KB
gasnet-openfabrics-2016.pdf	2023,11/14,周二 13:11	Adobe Acrobat ...	3,477 KB
Partitioned Global Address Space Pr...	2023,11/14,周二 10:39	Adobe Acrobat ...	16,505 KB
PGAS programming models - my 20-...	2023,11/14,周二 12:44	Adobe Acrobat ...	2,908 KB
pgas-tutorial.pdf	2023,11/14,周二 13:24	Adobe Acrobat ...	1,804 KB
Programming Models for Parallel Co...	2023,7/26,周三 14:48	Adobe Acrobat ...	2,102 KB
Pygion=Flexible, Scalable Task-Based...	2023,11/15,周三 11:01	Adobe Acrobat ...	295 KB
charm4py-main.zip	2023,11/15,周三 10:54	WinRAR ZIP arch...	452 KB
CharmPy=A Python Parallel Program...	2023,11/15,周三 10:53	Adobe Acrobat ...	303 KB

Dynamic languages provide the flexibility needed to implement expressive support for task-based parallel programming constructs. We present Pygion, a Python interface for the Legion task-based programming system, and show that it can provide features comparable to Regent, a statically typed programming language with dedicated support for the Legion programming model. Furthermore, we show that the dynamic nature of Python permits the implementation of several key optimizations (index launches, futures, mapping) currently implemented in the Regent compiler. Together these features enable Pygion code that is comparable in expressiveness but more flexible than Regent, and substantially more concise, less error prone, and easier to use than C++ Legion code. Pygion is designed to interoperate with Regent and can use Regent to generate high- performance CPU and GPU kernel implementations. We show that, in combination with high-performance kernels written in Regent, Pygion is able to achieve efficient, scalable execution on up to 512 nodes of the heterogeneous supercomputer Piz Daint.

8.7.3.9 更多的类似框架，请参看 [wiki.python.org](https://wiki.python.org/moin/ParallelProcessing) 网站ⁱ

-
- [1] Matthew Rocklin. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. Proceedings of the 14th Python in Science Conference (SCIPY). 2015. 126-132.
 - [2] Jesse C. Daniel. Data Science with Python and Dask. Manning Publications. 2019.
 - [3] Holden Karau, Mika Kimmins. Scaling Python with Dask: From Data Science to Machine Learning. O'Reilly Media. 2023.
 - [4] Tim Peters. Parallel Python with Dask: Perform distributed computing, concurrent programming and manage large dataset. GitforGits (India). 2023.
 - [5] 白琰冰. 数据科学并行计算. 中国人民大学出版社. 2021.
 - [6] Yannick Hold-Geoffroy, Olivier Gagnon, Marc Parizeau. Once you SCOOP, no need to fork. Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE). 2014. 1--8.
 - [7] Max Pumperla, Edward Oakes, Richard Liaw. Learning Ray Flexible Distributed Python for Machine Learning. O'Reilly Media, Inc. 2022.
 - [8] Elliott Slaughter, Alex Aiken. Pygion: Flexible, Scalable Task-Based Parallelism with Python. IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM). 2019. 58-72.

ⁱ <https://wiki.python.org/moin/ParallelProcessing>

课程讲义 2024年2月28日