STAT243 Lecture 6.4 Introduction to Dask

1 基本概念

- **Dask 概览**
  - Dask 是 Python 中用于并行计算的框架，与 R 的 **future** 包功能类似，可在单台或多台机器上实现任务并行化
  - 其重要特性在于支持 **分布式数据集 (distributed datasets)**：
  - 数据可被拆分为多个块 (chunks/shards)，并行执行计算

> **Logic**
>
> 分布式数据将在 Unit 7 中进一步介绍

- **其他并行框架**
  除 Dask 外，Python 中还有许多并行化工具，例如
  - `ipyparallel`
  - `ray`
  - `multiprocessing`
  - `pp`

---

2 核心思想 (Key Idea)

Dask 与 R 的 **future** 包、Python 的 **ray** 包共享一个核心理念：
将“**并行化的定义**”与“**并行化的执行方式与资源**” **解耦 (abstraction)**

目标包括：

- **分离 what 与 how/where**
  - 在代码中定义要并行化的部分 (what)
  - 而具体如何与在哪里执行 (how/where) 由系统配置决定
- **可移植性**
  - 同一段代码可在不同计算资源上运行，无需修改代码本身

> **Logic**
>
> 代码运行所依赖的计算资源称为 **后端 (backend)**

---

3 并行后端 (Parallel Backends)

通过 **scheduler (调度器)** 控制并行化的执行方式，例如是否跨机器运行、每台机器使用的核心数等

示例：使用多个 Python 进程并行化计算

Python

```python
import dask
dask.config.set(scheduler='processes', num_workers=4)
```

3.1 调度器类型对比表

类型	描述	支持多节点 multi-node	是否复制对象
synchronous	串行执行 (非并行)	否	否
threads	当前 Python 会话内的多线程	否	否
processes	多个后台 Python 进程	否	是
distributed	跨多节点的 Python 会话	是	是

3.2 关于 GIL 与线程调度

- Python 有 **全局解释器锁 (GIL)**，阻止纯 Python 代码的并行执行
- 因此 `threads` 调度器只适用于 **底层由 C/C++/Cython 实现** 的运算，如
  - `numpy` 数组计算
  - `pandas` DataFrame 操作
- 纯 Python 循环或逻辑不会真正并行化

3.3 使用 distributed 调度器的优势

- 可在单机使用 (如笔记本电脑)
- 相较于 `multiprocessing` 的优点包括：
  - 提供 **可视化监控仪表盘 (diagnostic dashboard)**
  - 更高效的内存复制与对象管理
- 进行 **并行 map 操作** 时必须使用 distributed 调度器

---

4 变量与 worker 的访问 (Variables and Workers)

4.1 隐式使用 delayed 装饰器

Dask 会自动识别并复制代码中所需的 **包与全局变量** 到 worker 进程中，无需手动导入或传递

示例：

Python

```python
import dask
dask.config.set(scheduler='processes', num_workers=4, chunksize=1)

import numpy as np
n = 10

@dask.delayed
def myfun(idx):
    return np.random.normal(size=n)

tasks = []
p = 8
for i in range(p):
    tasks.append(myfun(i))  # 延迟创建任务 (lazy task)

results = dask.compute(tasks)  # 并行执行所有任务
```

说明：

- `@dask.delayed` 装饰器将函数转为惰性 (lazy) 任务
- 这些任务会被添加到 Dask 的调度图中 (**使用了 Dynamic Scheduling**)，由调度器自动分配到不同进程执行
- 变量 `n` 和包 `numpy` 自动可用，无需显式传入

4.2 显式使用 delayed 装饰器

无需事先为函数添加 `@delayed` 装饰器，也可以直接调用：

Python

```
tasks.append(dask.delayed(myfun)(i))
```

这种写法的好处是 `myfun()` 可以根据具体场景决定要不要惰性执行：

- 若无需使用 Dask，只调用 `myfun(i)` 即可正常运行
- 若使用 Dask，可通过 `dask.delayed()` 将函数变为可并行的惰性任务

> ⚠ **Remark: Exporting Variables**
>
> - 在其他语言或并行框架中，通常需**显式地将对象复制到 worker 节点**，这一过程称为 **变量导出 (exporting)**
> - 而 Dask 会自动完成此步骤，使用户无需关心底层数据传输与依赖管理