Asyncio & Asynchronous Programming in Python

David Saccon
Oct 16, '19

About me

- 18+ years in networking, cloud infrastructure and Telecom
- Software engineering, Solutions architect and PLM at Redback Networks and Ericsson
- Currently co-founder and software engineer at ATG Trading
  - Market data, trading tools and analytics for digital currencies
- 🎾 ⛷️ 🛩️

Agenda

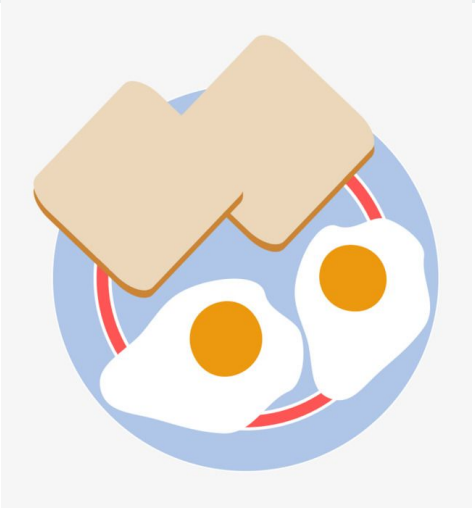
- Definition of key terms: concurrency vs parallelism, sync vs async
- Types of applications that can benefit from these approaches
- The different Python tools that are available
- Dive into some code examples with Python's asyncio library

Open Question

- How can I speed up my software program?

- Some questions:
  - See if it is made up of smaller sub ‘tasks’
  - If so, do these tasks need to run at the same time, or not?
  - How can I organize or schedule these tasks in a smart way?

Taking a step back

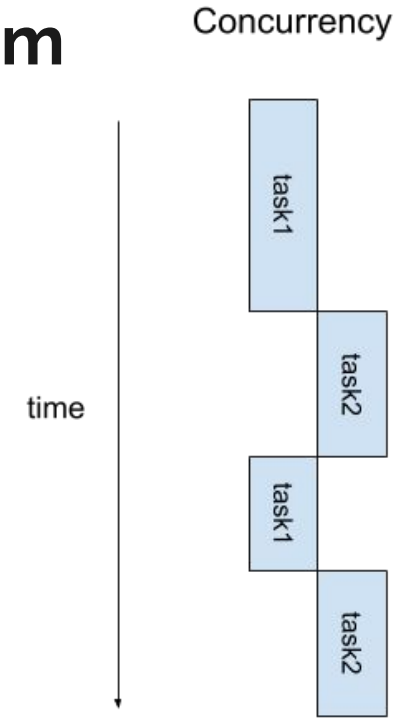


Different approaches to preparing breakfast

1. Cook eggs til finished. Then cook toast. Serve
2. Start both. Keep an eye on things. Serve when both are finished
3. Get your 2 kids to cook, with your spouse coordinating them. Breakfast in bed
4. Hire 2 caterers, one for the eggs and one for the toast. Get them to deliver. Serve

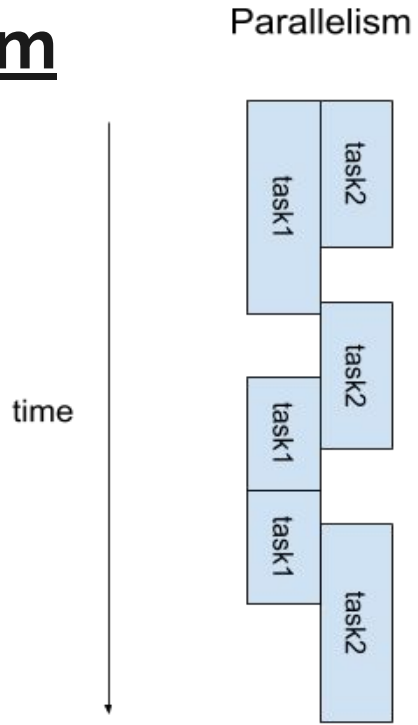
Definition 1: Concurrency vs Parallelism

- **Concurrency**: tasks run in overlapping time periods
  - But not at the same time
- Requires making decisions about when to switch tasks
  - I.e. multi-tasking on a single CPU core



Definition 1: Concurrency vs Parallelism

- **Parallelism**: Tasks literally run at the same time
- Splits work between multiple physical resources
    - I.e. run multiple tasks across CPU cores



Why does this matter?

Programs can be characterized as:

- **IO bound**: most time is spent waiting on I/O
  - I.e. a web server, big data processing/streaming
- **CPU bound**: most time is spent waiting on CPU
  - E.g. math/scientific computations, image processing, ML algos

Definition 2: Asynchronous vs Synchronous

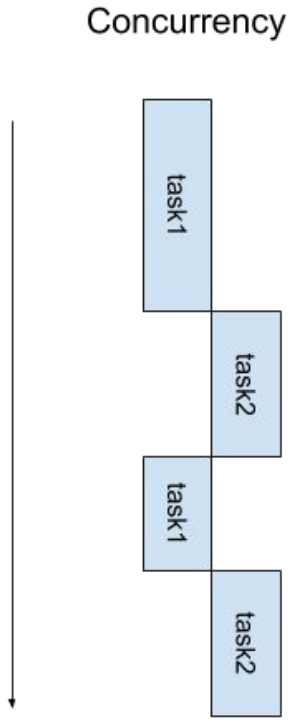
- Synchronous
  - Must complete before proceeding
  - Overall duration longer

- Asynchronous
  - No need to wait before proceeding
  - Overall duration shorter

Definition 3: Cooperative vs Pre-emptive multitasking

Different types of asynchronous concurrency

- Pre-emptive multitasking:
  - Scheduler (e.g. OS) *interrupts* tasks
- Cooperative Multitasking:
  - Tasks *yield* control to scheduler (e.g. event loop) when blocked



Putting it all together

Type	Concurrency/ Parallelism	Sync/Async	Best for..
Single-task	Neither	Sync	-
Pre-emptive multi-tasking	Concurrency	Async	IO-bound
Cooperative multi-tasking	Concurrency	Async	IO-bound
Multiprocessing	Parallelism	Either	CPU-bound

The Python landscape

- Synchronous
  - Regular functions, loops, etc
- Asynchronous (multi-threaded, pre-emptive multi)
  - *threading* library. Run multiple tasks across threads
- Asynchronous (single-threaded, cooperative multi)
  - *asyncio* library. Run multiple tasks on a single thread
  - ...
- Parallel
  - *multiprocessing* library. Run multiple tasks across CPU cores

Asyncio

- Standard library added in Python 3.4, single-thread async concurrency
- Has similarities to concurrency implementations in other languages (i.e. JS)
- Speeds up concurrent programs. Can be faster than Python threading
    - I.e. CPython GIL limitations

The Python landscape - *other stuff*

- *twisted*: Python2+ library for event-driven networking
- *gevent*: Python2+ coroutine-based async library, based on greenlet
- *tornado*: legacy async web server/framework
- *sanic*: Python3 asyncio-based web server/framework
- *curio*: Python3 lower-level alternative to asyncio from David Beasley
- *uvloop*: faster drop-in replacement for asyncio event loop

Asyncio - in a nutshell

- Event loop
  - Scheduler for tasks (aka ‘coroutines’)
  - Like a `while True` loop that monitors and switches between coroutines
- Coroutines
  - Where tasks live. Give control back to EL when they reach a blocking state (e.g. when waiting on network I/O)
- Futures, Tasks...
  - Good to know, but lets not worry about this for now..

Basic Syntax

```
# Python 3.7+

async def coro(delay):
    await asyncio.sleep(delay)
    print('Task done, delay =', delay)

asyncio.run(coro(3))
```

```
# Python 3.4+

async def coro(delay):
    await asyncio.sleep(delay)
    print('Task done, delay =', delay)

loop = asyncio.get_event_loop()
loop.run_until_complete(coro(3))
loop.close()
```

Syntax - Scheduling Tasks

```python
# Python 3.7+

async def coro(delay):
    await asyncio.sleep(delay)
    print(f'Task done, delay = {delay}')

async def coro_main(delay):
    task_1 = asyncio.create_task(coro(delay))
    task_2 = asyncio.create_task(coro(2*delay))
    await task_1
    await task_2

asyncio.run(coro_main(3))
```

```python
# Python 3.4+

async def coro(delay):
    await asyncio.sleep(delay)
    print('Task done, delay =', delay)

async def coro_main(delay):
    task_1 = asyncio.ensure_future(coro(delay))
    task_2 = asyncio.ensure_future(coro(2*delay))
    await task_1
    await task_2

loop = asyncio.get_event_loop()
loop.run_until_complete(coro_main(3))
loop.close()
```

Enough slides already

Let’s get to the code…

Completed demos:

- https://pastebin.com/D4KH13ce
- https://pastebin.com/VCg0HSh5
- https://pastebin.com/406frhxh

Thank you :)