Asyncio in production

Hrafn Eiriksson
Async/await in Python 3.5 and why it is awesome

Python 3's Killer Feature: asyncio

Jun 21, 2017 | By Michael Flaxman, Principal Engineer

REQUESTS → Register Callback → INTENSIVE OPERATION
Why you no asyncio?!
Why you no asyncio?!

Asynchronous programming is different
Why you no asyncio?! 

asyncio is relatively new
Why you no asyncio?!

Converting existing Python apps to use asyncio is not simple
Why you no asyncio?! 

The community has built multiple concurrency libraries
Asynchronous programming is not always what you want
My goal today

1. Discuss why I went all in on asyncio (and try to convince you to do the same)
2. Migrating to asyncio and the inevitable issues you run into
3. Asyncio in production: A before/after comparison

Not my goal today

4. An introduction into asyncio
Part 1: Why bother with asyncio?

My software already works!
A bit of background...
A bit of background...

```
krummi@krummi:$ node

> [] + []

> [] + {}
'[object Object]'

> {} + []
0

> {} + {}
'[object Object][object Object]'  
```

Credit: https://www.destroyallsoftware.com/talks/wat
A bit of background...
How we (typically) scale our services

Clients → Load balancer → Instance 1 → Gunicorn master → Gunicorn worker 1

Clients → Load balancer → Instance 2 → Gunicorn master → Gunicorn worker 2

Clients → Load balancer → Instance 3 → Gunicorn master → Gunicorn worker 3

Clients → Load balancer → Instance 4 → Gunicorn master → Gunicorn worker y

...
import eventlet

eventlet.monkey_patch()
import eventlet

eventlet.monkey_patch()
What makes asyncio so attractive is that it’s:

- Explicit
- Part of the language
Part 2: Migrating to asyncio
The asyncio ecosystem

Previously people relied on monkey patching

Now it seems to be becoming quite mature:

- Dozens of web frameworks (aiohttp, Sanic, Quart)
- Loads of database drivers (asyncpg, aiomysql, aioredis, etc)
- ...and way more [1]

A microservice migration
Based on a true story
Asyncio web frameworks

- **aiohttp**
  - Mature
  - Flask compatible
  - HTTP 2.0
  - WebSockets support

- **Sanic**
  - Flask-like
  - Fast
  - HTTP 2.0

- **Quart**

- **Vibora**
An example: Quart

Flask

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Quart [2]

```python
from quart import Quart
app = Quart(__name__)

@app.route('/')
async def hello_world():
    return 'Hello, World!'
```

[Migrating from Flask](#)

It should be possible to migrate to Quart from Flask by a find and replace of `flask` to `quart` and then adding `async` and `await` keywords. See the [docs](https://gitlab.com/pgjones/quart) for full details.

from aiohttp import web

async def handle(request):
    name = request.match_info.get('name', 'Anonymous')
    text = "Hello, " + name
    return web.Response(text=text)

app = web.Application()
app.add_routes([web.get('/', handle),
                 web.get('/{name}', handle)])

web.run_app(app)
from raven import Client as SentryClient
from aiohttp import web
from raven_aiohttp import AioHttpTransport

@web.middleware
async def sentry.middleware(request, handler):
    try:
        return await handler(request)
    except Exception:
        request.app.sentry_client.captureException()
    raise

app = web.Application(middlewares=[sentry.middleware])
app.sentry_client = SentryClient(  
    sentry_dsn,
    transport=AioHttpTransport,
)
Then problems hit...

Springfox petstore API

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Created by springfox
Apache License Version 2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/api/pet</td>
<td>Add a new pet to the store</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/pet</td>
<td>Update an existing pet</td>
</tr>
<tr>
<td>GET</td>
<td>/api/pet/findByStatus</td>
<td>Finds Pets by status</td>
</tr>
<tr>
<td>GET</td>
<td>/api/pet/findByTags</td>
<td>Finds Pets by tags</td>
</tr>
</tbody>
</table>
Issues with asyncio

- A lot of new concepts to wrap your head around
- `async/await` everywhere
- Debugging asyncio code can be problematic
- Be wary of running synchronous code in async functions
TLDR

1. Map out your dependencies to see if asyncio-compatible versions exist
2. Experiment with asyncio versions of your dependencies
3. Watch out for asyncio gotchas
4. Profit!
Part 3: asyncio in production
Before/after comparison

Before: Flask + psycopg2 + eventlet

VS

After: aiohttp + asyncpg + uvloop
Methodology

- Use wrk (https://github.com/wg/wrk) to do HTTP benchmarking.
- Ran each configuration of the benchmark:
  - For 30 seconds
  - 10 times
  - Using a variable number of open HTTP connections
  - Noted the median and the 25%/75% latency for each run
  - 10 seconds of sleep between runs
Local comparison: Simple ping
Local comparison: Database access
In production comparison

![Graph showing comparison of requests per second between two methods (aiohttp and eventlet) with respect to the number of open connections. The graph indicates a higher request rate for aiohttp compared to eventlet as the number of open connections increases.]
Conclusion

So is asyncio worth the effort?
Thank you!