Asynchronous Network Requests in Web Applications

Lauris Jullien
lauris@yelp.com/@laucia_julljen
Yelp’s Mission
Connecting people with great local businesses.
Yelp Stats
As of Q1 2016

90M
102M
70%
32
What is this talk about?

- Why would you want to do that?
- Why can it be complicated?
- What’s a deployment server (uWSGI)
- How To: Code Examples and ideas
What is the problem we are trying to solve?

High level view
What is the problem we are trying to solve?

With a SOA
What is the problem we are trying to solve?

Async!
ThreadPool Executor
concurrent.futures

Changed in version 3.5: If `max_workers` is None or not given, it will default to the number of processors on the machine, multiplied by 5, assuming that ThreadPoolExecutor is often used to overlap I/O instead of CPU work and the number of workers should be higher than the number of workers for ProcessPoolExecutor.

```python
import concurrent.futures
import urllib.request

URLS = [...]  

def load_url(url, timeout):
    with urllib.request.urlopen(url, timeout=timeout) as conn:
        return conn.read()

with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    future_to_url = {executor.submit(load_url, url, 60): url for url in URLS}
    for future in concurrent.futures.as_completed(future_to_url):
        url = future_to_url[future]
        data = future.result()

https://docs.python.org/dev/library/concurrent.futures.html
```
Deployment

How do I do that efficiently now?

Running a ...

Tornado/Twisted/... app?

WSGI app? (django, pyramid, flask ... )
WSGI Deployment: uwsgi

Why uwsgi?

- Widely used and well tested
- Very configurable: almost every combinations is possible (threads, process, events loop, greenlets, ....)
- Pre-forked (fork abusing) model
Deployment Server/Gateway

The pre-forked model

Master

Worker
Deployment Server/Gateway

Serving requests to your app

Here may be reverse proxies (nginx)

Master

Worker

Worker

Worker
Simple Synchronous App

```python
import time
import requests

def application(env, start_response):
    start_response("200 OK", [("Content-Type", "text/html")])
    start_time = time.time()
    calls = [long_network_call(i/8) for i in range(1,5)]
    end_time = time.time()

    return [
        b"This call lasted %0.3f seconds with synchronous calls.\n"% (end_time - start_time)
    ]

def long_network_call(duration):
    requests.get('http://localhost:7001/?duration={}'.format(duration))
```
Simple Synchronous App configs

# uwsgi_basic.ini
[uwsgi]
http = :5000
wsgi-file = app_sync.py
master = 1

# uwsgi_process.ini
[uwsgi]
http = :5001
wsgi-file = app_sync.py
master = 1
processes = 4

# uwsgi_thread.ini
[uwsgi]
http = :5002
wsgi-file = app_sync.py
master = 1
threads = 4

# uwsgi_mix.ini
[uwsgi]
http = :5003
wsgi-file = app_sync.py
master = 1
processes = 2
threads = 2
Simple Synchronous App

Results!

curl localhost:5000
   This call lasted 1.282 seconds with synchronous calls.

# uwsgi_basic (1 process)
python3 hammer.py --port 5000 --nb_requests 20
   We did 20 requests in 25.425450086593628

# uwsgi_process (4 processes)
python3 hammer.py --port 5001 --nb_requests 20
   We did 20 requests in 6.418

# uwsgi_thread (4 threads)
python3 hammer.py --port 5002 --nb_requests 20
   We did 20 requests in 6.479

# uwsgi_mix (2 process with 2 threads each)
python3 hammer.py --port 5003 --nb_requests 20
   We did 20 requests in 6.415
import asyncio
# ...
from aiohttp import ClientSession

def application(env, start_response):
    # ...
    loop = asyncio.get_event_loop()
    futures = [
        asyncio.ensure_future(long_network_call(i/8))
        for i in range(1,5)
    ]
    loop.run_until_complete(asyncio.wait(futures))
    # ...

async def long_network_call(duration):
    async with ClientSession() as session:
        async with session.get('http://localhost:7001/?duration={}'.format(duration)) as response:
            return await response.read()
Simple Asynchronous App

Event loop

Twisted Network Programming Essentials -
2nd edition - Jessica McKellar and Abe Fettig - O'Reilly 2013
Simple Asynchronous App
Performance and Cavehats

curl localhost:5100
This lasted 0.518 seconds with async calls using asyncio

python3 hammer.py --port 5100 --nb_requests 20
We did 20 requests in 5.010
Making uwsgi threads option work requires changing the get_loop()

```python
def get_loop():
    try:
        loop = asyncio.get_event_loop()
    except RuntimeError as e:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)
    finally:
        return loop
```

Running with --threads 2
Simple Asynchronous App
Performance and Cavehats

aiohttp spawns extra threads for dns resolution (which is kind of what we don’t want)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Mem</th>
<th>Swp</th>
<th>Tasks: 117, 222 thr; 2 running</th>
<th>Load average: 0.31 0.10 0.07</th>
<th>Uptime: 02:58:25</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>PPID USER</td>
<td>S  RES</td>
<td>SHR MEM% CPU% TIME+ Command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16116 16114</td>
<td>lauris</td>
<td>S 22552</td>
<td>5408 0.6 1.4 0:00.42</td>
<td>/home/lauris/.local/bin/uwsqi</td>
<td>uwsgi</td>
</tr>
<tr>
<td>16138 16114</td>
<td>lauris</td>
<td>S 22552</td>
<td>5408 0.6 0.0 0:00.00</td>
<td>uwsgi</td>
<td></td>
</tr>
<tr>
<td>16136 16114</td>
<td>lauris</td>
<td>S 22552</td>
<td>5408 0.6 0.0 0:00.00</td>
<td>uwsgi</td>
<td></td>
</tr>
<tr>
<td>16135 16114</td>
<td>lauris</td>
<td>S 22552</td>
<td>5408 0.6 0.0 0:00.00</td>
<td>uwsgi</td>
<td></td>
</tr>
<tr>
<td>16133 16114</td>
<td>lauris</td>
<td>S 22552</td>
<td>5408 0.6 0.0 0:00.00</td>
<td>uwsgi</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

app_asyncio worker htop

<table>
<thead>
<tr>
<th>CPU</th>
<th>Mem</th>
<th>Swp</th>
<th>Tasks: 116, 211 thr; 2 running</th>
<th>Load average: 0.07 0.07 0.05</th>
<th>Uptime: 03:06:17</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>PPID USER</td>
<td>S  RES</td>
<td>SHR MEM% CPU% TIME+ Command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16239 16236</td>
<td>lauris</td>
<td>S 18648</td>
<td>5836 0.5 0.0 0:00.08</td>
<td>/home/lauris/.local/bin/uwsqi</td>
<td></td>
</tr>
</tbody>
</table>
import time
from functools import partial

import gevent
import requests
from gevent import monkey

# Monkey-patch.
monkey.patch_all(thread=False, select=False)

def application(env, start_response):
    # ...
    jobs = [
        gevent.spawn(partial(long_network_call, i/8))
        for i in range(1,5)
    ]
    gevent.joinall(jobs)
    # ...

def long_network_call(duration):
    requests.get('http://localhost:7001/?duration={}'.format(duration))
**Gevent App**

**Perf**

curl localhost:5200
    This lasted **0.539** seconds with async calls using gevent

python3 hammer.py --port 5200 --nb_requests **50**
    We did 100 requests in **1.255**

python3 hammer.py --port 5200 --nb_requests **100**
    We did 100 requests in **1.373**

python3 hammer.py --port 5200 --nb_requests **200**
    We did 200 requests in **2.546**
Gevent

DNS resolution ... again

app_gevent worker htop: we can see 4 threads, when we expect 1

strace -p 17024
This is doing dns resolution!
import atexit
import functools
from concurrent.futures import Future

from tornado.httpclient import AsyncHTTPClient
from tornado.ioloop import IOLoop

_loop = IOLoop()

def _event_loop():
    _loop.make_current()
    _loop.start()

def setup():
    t = threading.Thread(
        target=_event_loop,
        name="TornadoReactor",
    )
    t.start()

def clean_up():
    _loop.stop()
    _loop.close()
atexit.register(clean_up)

setup()

def long_network_call(duration):
    http_client = AsyncHTTPClient(_loop)

    # this uses the threadsafe loop.add_callback internally
    fetch_future = http_client.fetch(
        'http://localhost:7001/?duration={}'.format(duration)
    )

    result_future = Future()
    def callback(f):
        try:
            result_future.set_result(f.result())
        except BaseException as e:
            result_future.set_exception(e)

    fetch_future.add_done_callback(callback)

    return result_future
def application(env, start_response):
    start_response("200 OK", ["Content-Type","text/html")])
    start_time = time.time()

    futures = [
        long_network_call(i/8) for i in range(1,5)
    ]
    # Let's do something heavy like ... waiting
    time.sleep(1)

    for future in futures:
        future.result()

    end_time = time.time()

    return [b"This call lasted %0.3f seconds with offloaded asynchronous calls.\n" % (end_time - start_time) ]
Offloading in a separate loop thread

curl localhost:5300
This lasted 1.003 seconds with offloaded asynchronous calls.

python3 hammer.py --port 5300 --nb_requests 20
We did 20 requests in 10.097
Offloading Event Loop Ready Made: Crochet

https://github.com/itamarst/crochet

- Uses twisted event loop
- Actually allows to run much more in the reactor than just network requests

- If you are after just the networking: Fido!
  https://github.com/Yelp/fido
Final notes

Use what fit your needs, or what needs to fit

- Tradeoff between speed and concurrency
- Beware of DNS resolutions

All code used for this presentation is available [https://github.com/laucia/europython_2016/](https://github.com/laucia/europython_2016/)
You should probably not use it in production
fb.com/YelpEngineers
@YelpEngineering
engineeringblog.yelp.com
github.com/yelp
QUESTIONS?