Django on Gevent

asynchronous i/o in a synchronous world

UMBEL™

About Me

- 5 years professionally as web developer
  - 4 years using Django
  - 2 years at Lawrence Journal-World (birthplace of Django)
- Tech obsessions
  - Systems and operations
  - Performance and scalability
  - Django!

The “Real-time” Web

- Technologies that enable delivery of information to users the instant the servers know about it
- Interesting scaling challenges
- Lots of open connections

The C10K Problem

“It's time for web servers to handle ten thousand clients simultaneously, don't you think? After all, the web is a big place now.”

- Dan Kegel

http://www.kegel.com/c10k.html

High concurrency creates new challenges

- Non-blocking I/O
- Low resource overhead
- Distributed

Building concurrent systems

- Processes
- Threads
- Non-blocking I/O
  - Callbacks (CPS, Reactor Pattern)
  - Coroutines

Threads

- Pre-emptive scheduling (non-deterministic)
  - Race conditions and locks/mutexes
- Memory overhead
- Readable, synchronous interface
- Guaranteed cooperation

Callbacks

- Call stack not preserved
- Simple things are intuitive
- Complex things become confusing

Coroutines

- Call stack preserved
- Synchronous API
- Benefits of threads without the non-determinism

Greenlet

- True coroutines in Python
- Call switching from Stackless Python implemented as a C extension module
- Call stack slicing to preserve context
  - Portions of the stack are copied to the heap and vice-versa

Gevent

- Expands upon greenlet to provide “green threads”
- Provides an event loop (libev with 1.0beta, libevent on current pre-1.0)

Green threads

- Similar programming style to POSIX threads
- POSIX threads are pre-emptive
- Green threads are cooperative
- Many green threads can exist within a single POSIX thread
- Green threads are very lightweight

gevent.spawn

```
>>> import gevent
>>> def add(x, y):
...     print x + y
...
...
>>> greenlet = gevent.spawn(add, 1, 2)
>>> greenlet.join()
3
```

gevent.sleep

```
>>> def func1():
...     gevent.sleep(2)
...     print 'function 1'
...
>>> def func2():
...     gevent.sleep(1)
...     print 'function 2'
...
>>> green1 = gevent.spawn(func1)
>>> green2 = gevent.spawn(func2)
>>> gevent.joinall([green1, green2])
function 2
function 1
```

gevent.Timeout

```
>>> with gevent.Timeout(seconds=1,
exception=False):
...     gevent.sleep(2)
...     data = 'data'
...
>>> if data is None:
...     print 'timeout elapsed'
... else:
...     print data
...
timeout elapsed
```

gevent.event.Event

```
>>> from gevent.event import Event
>>> event = Event()
>>> def func1(event):
...     print 'func1: setting event'
...     event.set()
>>> def func2(event):
...     print 'func2: waiting for event'
...     event.wait()
...     print 'func2: event captured'
>>> gevent.joinall([
...     gevent.spawn(func1, event),
...     gevent.spawn(func2, event)])
func2: waiting for event
func1: setting event
func2: event captured
```

gevent.event.AsyncResult

```
>>> from gevent.event import AsyncResult
>>> result = AsyncResult()
>>> def func1(result):
...     print 'func1: sending data'
...     result.set('nice!')
>>> def func2(result):
...     print 'func2: waiting for result'
...     data = result.get()
...     print 'func2: result captured: ' + data
>>> gevent.joinall([
...     gevent.spawn(func1, result),
...     gevent.spawn(func2, result)])
func2: waiting for result
func1: sending data
func2: result captured: nice!
```

gevent.queue

```
>>> import gevent
>>> from gevent.queue import Queue
>>> queue = Queue()
>>> def func1(queue):
...     queue.put(1)
...     queue.put(2)
>>> def func2(queue):
...     print queue.get()
...     print queue.get()
>>> gevent.joinall([
...     gevent.spawn(func1, queue),
...     gevent.spawn(func2, queue)])
1
2
```

gevent.pool

```
>>> import gevent
>>> pool = gevent.pool.Pool(size=2)
>>> def print_value(value):
...     print value
>>> pool.spawn(print_value, 1)
<Greenlet at 0x10d93d4b0: print_value(1)>
>>> pool.spawn(print_value, 2)
<Greenlet at 0x10d93db90: print_value(2)>
>>> pool.spawn(print_value, 3)
2
<Greenlet at 0x10d93d5f0: print_value(3)>
>>> pool.join()
3
1
```

gevent.threadpool

```
>>> from time import sleep, time
>>> from gevent.threadpool import ThreadPool
>>> start_time = time()
>>> pool = ThreadPool(4)
>>> pool.map(sleep, [1, 1, 1, 1])
[None, None, None, None]
>>> print time() - start_time
1.07812786102
```

gevent.server

```
>>> from gevent.server import StreamServer
>>> def handle(socket, address):
...     socket.send('Your address is %s\n' % address[0])
...
>>> server = StreamServer(('127.0.0.1', 1234), handle)
>>> server.serve_forever()
```

```
$ nc localhost 1234
Your address is 127.0.0.1
```

gevent.socket

- Cooperative socket implementation
- When used with monkey patching, brings asynchronous network I/O to an abundance of third-party libraries:
  - memcached
  - redis-py
  - boto
  - requests

Wait, monkey patching?

- Yes!
- Uncooperative libraries are “patched” to cooperate with event loop
- Creates a large ecosystem of gevent-compatible libraries

gevent.monkey

- from gevent.monkey import patch_all; patch_all()
- Patches standard library:
  - socket
  - ssl
  - os
  - time
  - select
  - thread/threading

Django

- Built around synchronous APIs

obj = Model.objects.get(pk=id)

cache_hit = cache.get(key)

response = view(request)

- Gevent is a perfect fit
- Green libraries and monkey-patching solve blocking

Serving Django

- Use Gunicorn!
- In gunicorn.conf:
  - worker_class = “gevent”
- That’s it! (well, almost)

Green database library

```
# gunicorn.conf


# Postgres
def post_fork(server, worker):
    from psycogreen.gevent.psyco_gevent import make_psycopg_green
    make_psycopg_green()


# MySQL
def post_fork(server, worker):
    import pymysql
    pymysql.install_as_MySQLdb()
```

Celery

- Celery is a distributed background task runner
- ./manage.py celeryd -P gevent
- Useful especially if your tasks are I/O heavy (examples: API calls and web scraping)

Websockets

- gevent-websocket
  - worker_class = geventwebsocket.gunicorn.workers.GeventWebSocketWorker
- gevent-socketio
  - worker_class = socketio.sgunicorn.GeventSocketIOWorker

Scaling

- Messaging (backend)
  - ZeroMQ: zmq.green
  - Redis (monkey-patch compatible)
- Load balancing (frontend)
  - HAProxy
  - Varnish

Thank you!

Questions?