Greenlet-based concurrency

Goran Peretin
@gperetin

Who am I?

- Freelancer
- Interested in concurrent, parallel and distributed systems

What is this about?

- understand what <buzzword> is
- when should you use <buzzword>
- concurrency as execution model (as opposed to composition model)

There will be no...

- Turnkey solutions
- GIL
- Details

Buzzwords ahead!

- concurrent vs parallel execution
- cooperative vs preemptive multitasking
- CPU bound vs IO bound task
- thread-based vs event-based concurrency

Mandatory definitions

Parallel execution

- Simultaneous execution of multiple tasks
- Must have multiple CPUs

Concurrent execution

- Executing multiple tasks in the same time frame
- … but not necessarily at the same time
- Doesn't require multiple CPU cores

Why do we want concurrent execution?

- We need it - more tasks than CPUs
- CPU is much faster than anything else

Thread-based concurrecy

- Executing multiple threads in the same time frame
- OS scheduler decides which thread runs when

How OS scheduler switches tasks?

- When current thread does IO operation
- When current thread used up it's time slice

How OS scheduler switches tasks?

- When current thread does IO operation
- When current thread used up it's time slice

Preemptive multitasking

```
import urllib2

def get_url(url):
    html = urllib2.urlopen(url).read()
    print len(html)

get_url('http://www.python.org')
get_url('http://www.linux.org')
get_url('http://www.google.com')
```

Mandatory GIL slide

- Global Interpreter Lock
- One Python interpreter can run just one thread at any point in time
- Only problem for CPU bound tasks

CPU bound vs IO bound

- CPU bound - time to complete a task is determined by CPU speed
  - calculating Fibonacci sequence, video processing...
- IO bound - does a lot of IO, eg. reading from disk, network requests...
  - URL crawler, most web applications...

Python anyone?

- `import threading`
- Python threads - real OS threads

Houston, we have a...

Problem?

- Lots of threads
- Thousands

Benchmarks!

Sample programs

- Prog 1: spawn some number of threads - each sleeps 200ms
- Prog 2: spawn some number of threads - each sleeps 90s

Prog 1

- Sleep 200ms

# of threads	100	1K	10K	100K
Time	207 ms	327 ms	2.55 s	25.42 s

Prog 2

- Sleep 90s

# of threads	100	1K	10K	100K
RAM	~4.9 GB	~11.8 GB	~82GB	? (256GB)

… and more

- Number of threads is limited
- Preemptive multitasking

We need

- Fast to create
- Low memory footprint
- We decide when to switch

Green threads!

Green threads

- Not managed by OS
- 1:N with OS threads
- User threads, light-weight processes

Greenlets

- "...more primitive notion of micro-thread with no implicit scheduling; coroutines, in other words."
- C extension

Greenlets

- Micro-thread
- No implicit scheduling
- Coroutines

Coroutine

- Function that can suspend it's execution and then later resume
- Can also be implemented in pure Python (PEP 342)
- Coroutines decide when they want to switch

Coroutine

- Function that can suspend it's execution and then later resume
- Can also be implemented in pure Python (PEP 342)
- Coroutines decide when they want to switch

Cooperative multitasking

Cooperative multitasking

- Each task decides when to give others a chance to run
- Ideal for I/O bound tasks
- Not so good for CPU bound tasks

Using greenlets

- We need something that will know which greenlet should run next
- Our calls must not block
- We need something to notify us when our call is done

Using greenlets

- We need something that will know which greenlet should run next

Scheduler

- Our calls must not block
- We need something to notify us when our call is done

Using greenlets

- We need something that will know which greenlet should run next

Scheduler

- Our calls must not block
- We need something to notify us when our call is done

Event loop

Event loop

- Listens for events from OS and notifies your app
- Asynchronous

```python
import urllib2

def callback(html):
    print len(html)

def get_url(url):
    # This is an example call
    urllib2.urlopen(url, callback)

get_url('http://www.python.org')
get_url('http://www.linux.org')
get_url('http://www.google.com')
```

Greenlets + ...

- Scheduler
- Event loop

Gevent

Gevent

- “...coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libevent event loop.”

```
import gevent
from gevent import monkey; monkey.patch_socket()
import urllib2

def get_url(url):
    html = urllib2.urlopen(url).read()
    print len(html)

g1 = gevent.spawn(get_url, 'http://www.python.org')
g2 = gevent.spawn(get_url, 'http://www.linux.org')
g3 = gevent.spawn(get_url, 'http://www.google.com')

gevent.joinall([g1, g2, g3])
```

Prog 1

- Sleep 200ms

# of threads	100	1K	10K	100K
Time	207 ms	327 ms	2.55 s	25.42 s

# of Greenlets	100	1K	10K	100K
Time	204 ms	223 ms	421 ms	3.06 s

Prog 2

- Sleep 90s

# of threads	100	1K	10K	100K
RAM	4.9 GB	11.8 GB	82GB	? (256GB)

# of Greenlets	100	1K	10K	100K
Time	33 MB	41 MB	114 MB	858 MB

Gevent

- Monkey-patching

```
from gevent import monkey; monkey.patch_socket()
import socket
socket.gethostbyname('www.python.org')
```

- Event loop

Disadvantages

- Monkey-patching
- Doesn't work with C extensions
- Greenlet implementation details
- Hard to debug

Alternatives

- Twisted
- Tornado
- Callback based

PEP 3156 & Tulip

- Attempt to standardize event loop API in Python
- Tulip is an implementation

Recap

- Concurrent execution helps with IO bound applications
- Use threads if it works for you
- Use async library if you have lots of connections

Thank you!

- Questions?

Resources

- http://dabeaz.com/coroutines/Coroutines.pdf
- http://www.gevent.org/
- http://greenlet.readthedocs.org/en/latest/