Asynchronous Python?

Yes, it is a reality!

Async Python misnomers

- Python threads are not *real* threads
- They won't speed up my program (GIL anyone?)
- They are difficult (well, threading.Thread is kinda yucky)

Async Options

- Synchronous - for comparison
- Twisted
- Eventlet
- Threading
- Gevent
- Tornado

Simple Program

Retrieve the home page contents and capture its size (bytes) for the following domains:

- http://google.com
- http://yahoo.com
- http://facebook.com
- http://apple.com
- http://python.org
- http://stackoverflow.com

Sync Example Code

```
 1 import common
 2 import requests
 3 import time
 4
 5
 6 @common.timeit
 7 def get_page(ep):
 8     req = requests.get(ep)
 9     return req, ep
10
11 all_time = 0
12 for ep in common.eps:
13     (resp, ep), req_time = get_page(ep)
14     all_time += req_time
15     print ep.ljust(30), str(len(resp.content)).
16                    rjust(10), req_time
17
18 print "-" * 10
19 print all_time
```

Sync Example Results

```
1 > time python sync.py
2 http://google.com                  10920 1.05749297142
3 http://yahoo.com                  392935 1.84565806389
4 http://facebook.com                43634 0.672782897949
5 http://apple.com                   10011 0.357982873917
6 http://python.org                  44275 0.363801956177
7 http://stackoverflow.com          211262 0.350831985474
8 ----------
9 4.64855074883
```

- Says it took 4.6s to do all lookups

Sync Example Results

```
 1 > time python sync.py
 2 http://google.com                  10920 1.05749297142
 3 http://yahoo.com                  392935 1.84565806389
 4 http://facebook.com                43634 0.672782897949
 5 http://apple.com                   10011 0.357982873917
 6 http://python.org                  44275 0.363801956177
 7 http://stackoverflow.com          211262 0.350831985474
 8 ----------
 9 4.64855074883
10 python sync.py  0.14s user 0.08s system 4% cpu 4.846 total
```

- Says it took 4.6s to do all lookups
- time function concurs

Sync Example Results

```
 1 > time python sync.py
 2 http://google.com                    10920 1.05749297142
 3 http://yahoo.com                    392935 1.84565806389
 4 http://facebook.com                  43634 0.672782897949
 5 http://apple.com                     10011 0.357982873917
 6 http://python.org                    44275 0.363801956177
 7 http://stackoverflow.com            211262 0.350831985474
 8 ----------
 9 4.64855074883
10 python sync.py  0.14s user 0.08s system 4% cpu 4.846 total
```

- Says it took 4.6s to do all lookups
- time function concurs
- Some minor overhead for bootstrapping the interpreter

Sync Example Conclusion

PROS:

- Easy to follow logic
- Make a request, get a response, repeat

Sync Example Conclusion

PROS:

- Easy to follow logic
- Make a request, get a response, repeat

CONS:

- Takes sum of individual times
- Not scalable - change from 6 end points to 600?

Async Options

- ~~Synchronous~~
- Twisted
- Eventlet
- Threading
- Gevent
- Tornado

Twisted

Twisted is an event-driven network programming framework written in Python.

Twisted projects variously support TCP, UDP, SSL/TLS, IP Multicast, Unix domain sockets, a large number of protocols (including HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP, and others), and much more. Twisted is based on the event-driven programming paradigm, which means that users of Twisted write short callbacks which are called by the framework.

Twisted

Twisted is an event-driven network programming framework written in Python.

Twisted is event-driven which means callbacks

Twisted Example Code

```
import time
from twisted.internet import reactor, defer
from twisted.web.client import Agent, RedirectAgent
from twisted.web.http_headers import Headers
from twisted.internet.protocol import Protocol
from twisted.internet.ssl import ClientContextFactory
from common import eps

class WebClientContextFactory(ClientContextFactory):
    def getContext(self, hostname, port):
        return ClientContextFactory.getContext(self)


class StreamingParser(Protocol):
    def __init__(self, done):
        self.done = done

    def connectionMade(self):
        self.data = ''

    def dataReceived(self, dbytes):
        self.data += dbytes

    def connectionLost(self, reason):
        self.done.callback(self.data)


def cbResponse(resp, ep, start_time):
    done = defer.Deferred()
    done.addCallback(printLen, ep, start_time)
    handler = StreamingParser(done)
    resp.deliverBody(handler)
    return done

def cbError(resp):
    print "ERROR"
    print resp

def printLen(data, ep, start_time):
    global total_time
    req_time = time.time() - start_time
    total_time += req_time
    print ep.ljust(30), str(len(data)).rjust(10), req_time

def cbGroupResponse(blah):
    print "_" * 10
    print total_time
    reactor.stop()


if __name__ == "__main__":
    contextFactory = WebClientContextFactory()
    agent = RedirectAgent(Agent(reactor, contextFactory))
    total_time = 0

    defList = []
    for ep in eps:
        t1 = time.time()
        d = agent.request('GET', ep, Headers({'User-Agent': ['SocPug Request']
            }), None)
        d.addCallback(cbResponse, ep, t1)
        d.addErrback(cbError)
        defList.append(d)
    deferredList = defer.DeferredList(defList)
    deferredList.addCallback(cbGroupResponse)
```

Twisted Example Code

```
1  import time
2  from twisted.internet import reactor, defer
3  from twisted.web.client import Agent, RedirectAgent
4  from twisted.internet.protocol import Protocol
5  from twisted.internet.ssl import ClientContextFactory
6  from common import eps
7
8  class WebClientContextFactory(ClientContextFactory):
9      def getContext(self, hostname, port): return ClientContextFactory.getContext(self)
10
11 class StreamingParser(Protocol):
12     def __init__(self, done): self.done = done
13     def connectionMade(self): self.data = ''
14     def dataReceived(self, dbytes): self.data += dbytes
15     def connectionLost(self, reason): self.done.callback(self.data)
16
17 def cbResponse(resp, ep, start_time):
18     done = defer.Deferred().addCallback(printLen, ep, start_time)
19     resp.deliverBody(StreamingParser(done))
20     return done
21
22 def printLen(data, ep, start_time):
23     global total_time
24     req_time = time.time() - start_time
25     total_time += req_time
26     print ep.ljust(30), str(len(data)).rjust(10), req_time
27
28 def cbGroupResponse(blah):
29     print "_" * 10, "\n", total_time
30     reactor.stop()
31
32 if __name__ == "__main__":
33     agent = RedirectAgent(Agent(reactor, WebClientContextFactory()))
34     total_time = 0
35     defList = defer.DeferredList([agent.request('GET', ep).addCallback(cbResponse, ep, time.time()) for ep in eps]).addCallback(cbGroupResponse)
36     reactor.run()
```

Twisted Example Code

```
1  import time
2  from twisted.internet import reactor, defer
3  from twisted.web.client import Agent, RedirectAgent
4  from twisted.internet.protocol import Protocol
5  from twisted.internet.ssl import ClientContextFactory
6  from common import eps
7
8  class WebClientContextFactory(ClientContextFactory):
9      def getContext(self, hostname, port): return ClientContextFactory.getContext(self)
10
11 class StreamingParser(Protocol):
12     def __init__(self, done): self.done = done
13     def connectionMade(self): self.data = ''
14     def dataReceived(self, dbytes): self.data += dbytes
15     def connectionLost(self, reason): self.done.callback(self.data)
16
17 def cbResponse(resp, ep, start_time):
18     done = defer.Deferred().addCallback(printLen, ep, start_time)
19     resp.deliverBody(StreamingParser(done))
20     return done
21
22 def printLen(data, ep, start_time):
23     global total_time
24     req_time = time.time() - start_time
25     total_time += req_time
26     print ep.ljust(30), str(len(data)).rjust(10), req_time
27
28 def cbGroupResponse(blah):
29     print "_" * 10, "\n", total_time
30     reactor.stop()
31
32 if __name__ == "__main__":
33     agent = RedirectAgent(Agent(reactor, WebClientContextFactory()))
34     total_time = 0
35     defList = defer.DeferredList([agent.request('GET', ep).addCallback(cbResponse, ep, time.time()) for ep in eps]).addCallback(cbGroupResponse)
36     reactor.run()
```

Don’t do this!
Only done for the slides.

Twisted Example Results

```
1 > time python twis.py
2 http://python.org                    44275 0.463947057724
3 http://google.com                    45146 0.598286151886
4 http://apple.com                     10011 0.61084485054
5 http://stackoverflow.com            212099 0.746867179871
6 http://facebook.com                  20971 0.931021928787
7 http://yahoo.com                     94221 1.28130507469
8 __________
9 4.6322722435
```

- Says it took 4.6s to do all lookups - same as before

Twisted Example Results

```
 1 > time python twis.py
 2 http://python.org                    44275 0.463947057724
 3 http://google.com                    45146 0.598286151886
 4 http://apple.com                     10011 0.61084485054
 5 http://stackoverflow.com            212099 0.746867179871
 6 http://facebook.com                  20971 0.931021928787
 7 http://yahoo.com                     94221 1.28130507469
 8 __________
 9 4.6322722435
10 python twis.py  0.64s user 0.12s system 36% cpu 2.094 total
```

- Says it took 4.6s to do all lookups - same as before
- time function does ***NOT*** concur (2.2x faster)

Twisted Example Results

```
 1 > time python twis.py
 2 http://python.org                    44275 0.463947057724
 3 http://google.com                    45146 0.598286151886
 4 http://apple.com                     10011 0.61084485054
 5 http://stackoverflow.com            212099 0.746867179871
 6 http://facebook.com                  20971 0.931021928787
 7 http://yahoo.com                     94221 1.28130507469
 8 __________
 9 4.6322722435
10 python twis.py  0.64s user 0.12s system 36% cpu 2.094 total
```

- Says it took 4.6s to do all lookups - same as before
- time function does ***NOT*** concur (2.2x faster)
- Q: What happened?

Twisted Example Results

```
 1 > time python twis.py
 2 http://python.org                    44275 0.463947057724
 3 http://google.com                    45146 0.598286151886
 4 http://apple.com                     10011 0.61084485054
 5 http://stackoverflow.com            212099 0.746867179871
 6 http://facebook.com                  20971 0.931021928787
 7 http://yahoo.com                     94221 1.28130507469
 8 __________
 9 4.6322722435
10 python twis.py  0.64s user 0.12s system 36% cpu 2.094 total
```

- Says it took 4.6s to do all lookups - same as before
- time function does ***NOT*** concur (2.2x faster)
- Q: What happened? A: Asynchronous IO

Twisted Example Conclusion

PROS:

- Big speed up - > 2.2x
- Twisted has support for a ton of protocols. Very mature project.
- Very scalable

Twisted Example Conclusion

PROS:

- Big speed up - > 2.2x
- Twisted has support for a ton of protocols. Very mature project.
- Very scalable

CONS:

- Very verbose - addCallback, addErrback, addBoth…
- Callback driven - helped with @defer.inlineCallbacks
- You don’t write Python, you write Twisted.

Async Options

- ~~Synchronous~~
- ~~Twisted~~
- Eventlet
- Threading
- Gevent
- Tornado

Eventlet

Eventlet is built around the concept of green threads (i.e. coroutines, we use the terms interchangeably) that are launched to do network-related work. Green threads differ from normal threads in two main ways:

- Green threads are so cheap they are nearly free. You do not have to conserve green threads like you would normal threads. In general, there will be at least one green thread per network connection.

- Green threads cooperatively yield to each other instead of preemptively being scheduled. The major advantage from this behavior is that shared data structures don’t need locks, because only if a yield is explicitly called can another green thread have access to the data structure. It is also possible to inspect primitives such as queues to see if they have any pending data.

Eventlet Example Code

```
 1 import eventlet
 2 eventlet.monkey_patch()
 3 import requests
 4 import time
 5 import common
 6
 7 @common.timeit
 8 def get_page(ep):
 9     req = requests.get(ep, headers={"User-Agent":"Python eventlet"})
10     return req, ep
11
12 greenlets = []
13 pool = eventlet.GreenPool(len(common.eps))
14 pile = eventlet.GreenPile(pool)
15
16 for ep in common.eps:
17     pile.spawn(get_page, ep)
18 pool.waitall()
19 resps = list(pile)
20
21 all_time = 0
22 for resp in resps:
23     (resp, ep), req_time = resp
24     all_time += req_time
25     print ep.ljust(30), str(len(resp.content)).rjust(10), req_time
26 print "-" * 10
27 print all_time
```

Eventlet Example Results

```
1 > time python evntlt.py
2 http://google.com                   10907 0.671675205231
3 http://yahoo.com                   171831 1.17502689362
4 http://facebook.com                 43606 0.904833078384
5 http://apple.com                    10011 0.651261091232
6 http://python.org                   44275 0.563825845718
7 http://stackoverflow.com           211262 0.50678896904
8 ----------
9 4.47341108322
```

- Says it took 4.5s to do all lookups

Eventlet Example Results

```
 1 > time python evntlt.py
 2 http://google.com                    10907 0.671675205231
 3 http://yahoo.com                    171831 1.17502689362
 4 http://facebook.com                  43606 0.904833078384
 5 http://apple.com                     10011 0.651261091232
 6 http://python.org                    44275 0.563825845718
 7 http://stackoverflow.com            211262 0.50678896904
 8 ----------
 9 4.47341108322
10 python evntlt.py  0.62s user 0.07s system 37% cpu 1.649 total
```

- Says it took 4.5s to do all lookups
- time function does ***NOT*** concur (2.72x faster)

Eventlet Example Conclusion

PROS:

- Big speed up - > 2.7x
- Uses os event loop (epoll, kqueue, poll, select, twisted)
- Very synchronous looking code
- Full featured (db_pool, subprocesses, GreenPool, GreenPile)

Eventlet Example Conclusion

PROS:

- Big speed up - > 2.7x
- Uses os event loop (epoll, kqueue, poll, select, twisted)
- Very synchronous looking code
- Full featured (db_pool, subprocesses, GreenPool, GreenPile)

CONS:

- Support has fallen in favor of Gevent
- No support for libevent or libev.
- Maybe not super efficient?

Async Options

- ~~Synchronous~~
- ~~Twisted~~
- ~~Eventlet~~
- Threading
- Gevent
- Tornado

Threading

thread:
This module provides low-level primitives for working with multiple threads (also called light-weight processes or tasks) — multiple threads of control sharing their global data space. For synchronization, simple locks (also called mutexes or binary semaphores) are provided. The threading module provides an easier to use and higher-level threading API built on top of this module.

threading:
This module constructs higher-level threading interfaces on top of the lower level thread module. See also the mutex and Queue modules.

CPython implementation detail: In CPython, due to the Global Interpreter Lock, only one thread can execute Python code at once (even though certain performance-oriented libraries might overcome this limitation). If you want your application to make better use of the computational resources of multi-core machines, you are advised to use multiprocessing. However, threading is still an appropriate model if you want to run multiple I/O-bound tasks simultaneously.

Threading Example Code

```
from urllib2 import urlopen
from Queue import Queue, Empty
from threading import Thread
import common
import time

results = Queue()
queue = Queue()

def parse():
    try:
        url = queue.get_nowait()
        t_start = time.time()
        content = urlopen(url).read()
        t_end = time.time()
        results.put((url, len(content), t_end - t_start))
    except Empty:
        pass


if __name__ == '__main__':
    for ep in common.eps:
        queue.put(ep)
    workers = []
    for i in range(len(common.eps)):
        worker = Thread(target=parse)
        worker.start()
        workers.append(worker)
    for worker in workers:
        worker.join()
    all_time = 0
    while not results.empty():
        ep, size, req_time = results.get()
        print ep.ljust(30), str(size).rjust(10), req_time
        all_time += req_time

    print "-" * 10
    print all_time
```

Threading Example Results

```
 1 > time python thrd.py
 2 http://python.org                    44275 0.4920399189
 3 http://google.com                    10958 0.624886989594
 4 http://apple.com                     10011 0.626431941986
 5 http://stackoverflow.com            214235 0.666630983353
 6 http://facebook.com                  41992 0.953256845474
 7 http://yahoo.com                    171709 1.42871999741
 8 ----------
 9 4.79196667671
10 python thrd.py  0.06s user 0.03s system 5% cpu 1.478 total
```

- Says it took 4.8 to do all lookups

Threading Example Results

```
 1 > time python thrd.py
 2 http://python.org                    44275 0.4920399189
 3 http://google.com                    10958 0.624886989594
 4 http://apple.com                     10011 0.626431941986
 5 http://stackoverflow.com            214235 0.666630983353
 6 http://facebook.com                  41992 0.953256845474
 7 http://yahoo.com                    171709 1.42871999741
 8 ----------
 9 4.79196667671
10 python thrd.py  0.06s user 0.03s system 5% cpu 1.478 total
```

- Says it took 4.8 to do all lookups
- time function does ***NOT*** concur (3.24x faster)

Threading Example Conclusion

PROS:

- Big speed up - > 3.2x
- Built-in to Python - no libraries
- Lots of examples

Threading Example Conclusion

PROS:

- Big speed up - > 3.2x
- Built-in to Python - no libraries
- Lots of examples

CONS:

- Very verbose - especially class based
- Communicate using Queue
- Locking can be an issue (sample time?)
- Deadlocks

Async Options

- ~~Synchronous~~
- ~~Twisted~~
- ~~Eventlet~~
- ~~Threading~~
- Gevent
- Tornado

Gevent

gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev event loop.

Features include:

- Fast event loop based on libev (epoll on Linux, kqueue on FreeBSD).
- Lightweight execution units based on greenlet.
- API that re-uses concepts from the Python standard library (for example there are Events and Queues).
- Cooperative sockets with SSL support »
- Monkey patching utility to get 3rd party modules to become cooperative »

gevent is inspired by eventlet but features more consistent API, simpler implementation and better performance. Read why others use gevent and check out the list of the open source projects based on gevent.

Gevent Example Code

```
 1 import gevent
 2 from gevent import monkey
 3 monkey.patch_all()
 4 import requests
 5 import time
 6 import common
 7
 8 @common.timeit
 9 def get_page(ep):
10     req = requests.get(ep, headers={"User-Agent":"Python gevent"})
11     return req, ep
12
13 greenlets = []
14 for ep in common.eps:
15     greenlets.append(gevent.spawn(get_page, ep))
16 gevent.joinall(greenlets)
17
18 all_time = 0
19 for greenlet in greenlets:
20     (resp, ep), req_time = greenlet.value
21     all_time += req_time
22     print ep.ljust(30), str(len(resp.content)).rjust(10), req_time
23 print "-" * 10
24 print all_time
```

Gevent Code Compared to Synchronous

```
 1 import common
 2 import requests
 3 import time
 4
 5
 6 @common.timeit
 7 def get_page(ep):
 8     req = requests.get(ep)
 9     return req, ep
10
```

```
11 all_time = 0
12 for ep in common.eps:
13     (resp, ep), req_time = get_page(ep)
14     all_time += req_time
15     print ep.ljust(30), str(len(resp.content)).
16                     rjust(10), req_time
17
18 print "-" * 10
19 print all_time
```

```
 1 import gevent
 2 from gevent import monkey
 3 monkey.patch_all()
 4 import requests
 5 import time
 6 import common
 7
 8 @common.timeit
 9 def get_page(ep):
10     req = requests.get(ep)
11     return req, ep
12
13 greenlets = []
14 for ep in common.eps:
15     greenlets.append(gevent.spawn(get_page, ep))
16 gevent.joinall(greenlets)
17
18 all_time = 0
19 for greenlet in greenlets:
20     (resp, ep), req_time = greenlet.value
21     all_time += req_time
22     print ep.ljust(30), str(len(resp.content)).rjust(10),
req_time
23 print "-" * 10
24 print all_time
```

Gevent Example Results

```
1 > time python gevnt.py
2 http://google.com                  10924 0.193860054016
3 http://yahoo.com                  172370 0.774422168732
4 http://facebook.com                43590 0.632377147675
5 http://apple.com                   10011 0.154708862305
6 http://python.org                  44275 0.376847028732
7 http://stackoverflow.com          216542 0.364218950272
8 ----------
9 2.49643421173
```

- Says it took 2.49 to do all lookups

Gevent Example Results

```
 1 > time python gevnt.py
 2 http://google.com                    10924 0.193860054016
 3 http://yahoo.com                    172370 0.774422168732
 4 http://facebook.com                  43590 0.632377147675
 5 http://apple.com                     10011 0.154708862305
 6 http://python.org                    44275 0.376847028732
 7 http://stackoverflow.com            216542 0.364218950272
 8 ----------
 9 2.49643421173
10 python gevnt.py  0.15s user 0.05s system 22% cpu 0.897 total
```

- Says it took 2.49 to do all lookups
- time function does ***NOT*** concur (2.77x faster)

Async Options

- ~~Synchronous~~
- ~~Twisted~~
- ~~Eventlet~~
- ~~Threading~~
- ~~Gevent~~
- Tornado

Tornado

Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for long polling, WebSockets, and other applications that require a long-lived connection to each user.

Tornado Example

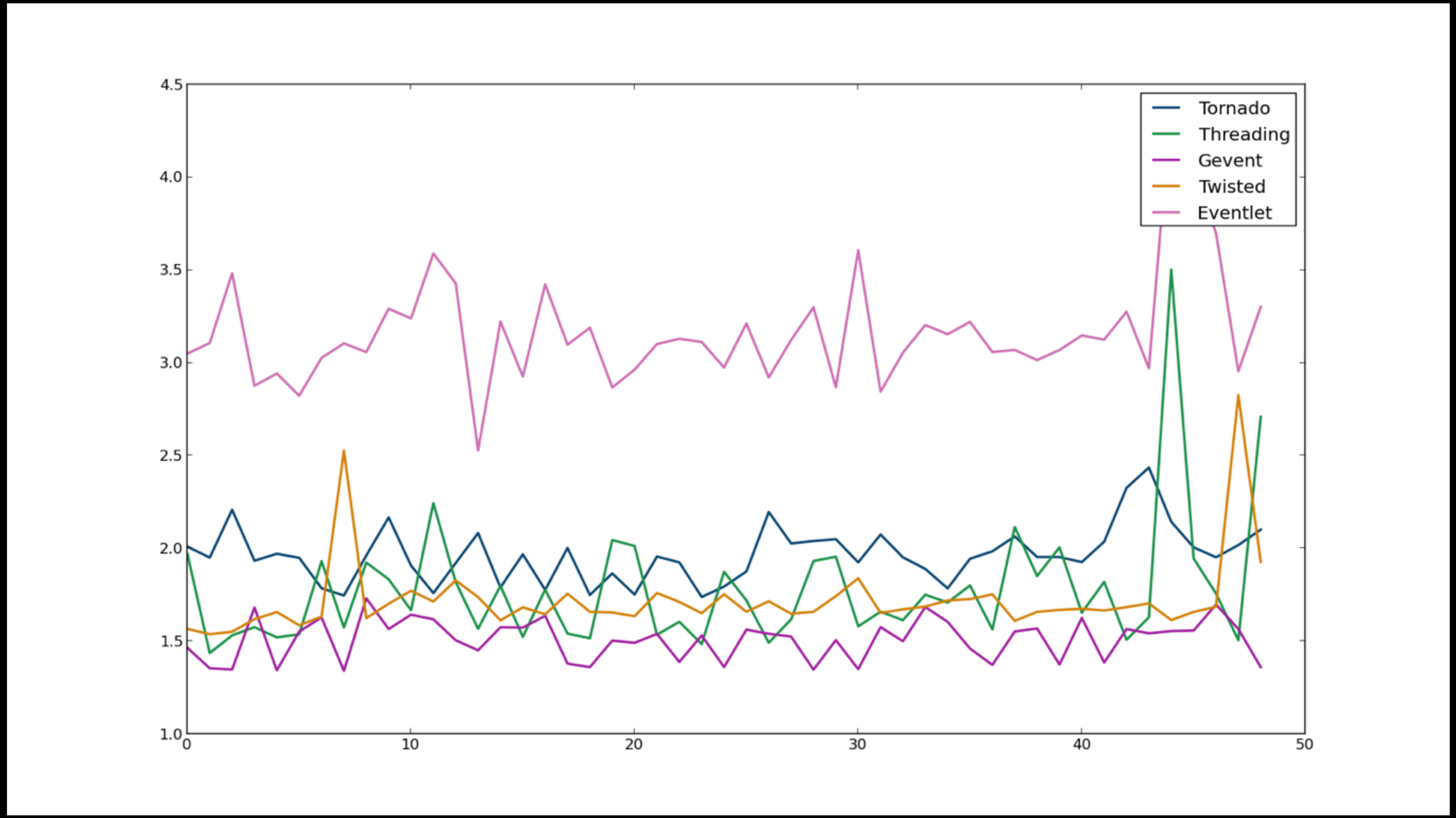
```
import tornado.httpclient
import tornado.ioloop
import tornado.gen
import common


@tornado.gen.engine
def main():
    keys = []
    http_client = tornado.httpclient.AsyncHTTPClient()
    for i, ep in enumerate(common.eps):
        key = 'key-%s' % (i,)
        keys.append(key)
        http_client.fetch(
                tornado.httpclient.HTTPRequest(ep, method="GET", headers={"User-Agent": "Firefox"}),
                callback=(yield tornado.gen.Callback(key)))
    complete = yield tornado.gen.WaitAll(keys)

    time_total = 0
    for item in complete:
        print item.effective_url.ljust(30), str(len(item.body)).rjust(10), item.request_time
        time_total += item.request_time

    print "-" * 10
    print time_total
    tornado.ioloop.IOLoop.instance().stop()

if __name__ == "__main__":
    tornado.ioloop.IOLoop.instance().add_callback(main)
    tornado.ioloop.IOLoop.instance().start()
```

Tornado Example Results

```
 1 > time python torn2.py
 2 http://www.google.com/              30055 0.226093053818
 3 https://www.yahoo.com/            320307 1.28047204018
 4 https://www.facebook.com/          43379 0.478633165359
 5 http://www.apple.com/              11678 0.238075971603
 6 https://www.python.org/            45453 0.345919132233
 7 http://stackoverflow.com          211518 0.347573041916
 8 ----------
 9 2.91676640511
10 python torn2.py  0.19s user 0.09s system 13% cpu 2.043 total
```

Async Options Results

Async Options Results



What about a web server?

Does a yielding approach (gevent, eventlet) work better than threading?