36-parallel

April 29, 2016

1 Parallel Processing in Python

- simple threading system
- threads run under one process and share memory
- has some concurrent data structures, like queues, locks, semaphores
- roughly similar to Java concurrency facilities
- doc
- generators are a way to do "manually scheduled threads"

```
In [1]: import time
        import threading

        def counter(n):
            for j in range(n):
                print('count is', j)
                time.sleep(1)

        t = threading.Thread(target=counter, args=[5])
        t.start()
```

```
count is 0
count is 1
count is 2
count is 3
count is 4
```

2 Actor example

- tried and true concurrency scheme
  - generalized producer/consumer
- no global memory, just msg passing btw actors
- easy to understand

```
In [2]: from multiprocessing import Queue
        from threading import Thread, Event

        # Sentinel used for shutdown

        class ActorExit(Exception):
            pass

        class Actor:
```

```
    def __init__(self):
        self._mailbox = Queue()

    def send(self, msg):
        '''
        Send a message to the actor
        '''
        self._mailbox.put(msg)

    def recv(self):
        '''
        Receive an incoming message
        '''
        msg = self._mailbox.get()
        if msg is ActorExit:
            raise ActorExit()
        return msg

    def close(self):
        '''
        Close the actor, thus shutting it down
        '''
        self.send(ActorExit)

    def start(self):
        '''
        Start concurrent execution
        '''
        self._terminated = Event()
        t = Thread(target=self._bootstrap)
        t.daemon = True
        t.start()

    def _bootstrap(self):
        try:
            self.run()
        except ActorExit:
            pass
        finally:
            self._terminated.set()

    def join(self):
        self._terminated.wait()

    def run(self):
        '''
        Run method to be implemented by the user
        '''
        while True:
            msg = self.recv()

# Sample ActorTask
class PrintActor(Actor):
    def run(self):
```

```
                while True:
                    msg = self.recv()
                    print('Print Actor Task Got:', msg)

        # Sample use
        p = PrintActor()
        p.start()
        p.send('Hello')
        p.send('World')
        p.close()
        p.join()
```

```
Print Actor Task Got: Hello
Print Actor Task Got: World
```

3 Global Interpreter Lock(GIL)

- The core of python is NOT concurrent
- The GIL can only be aquired by ONE thread at a time
- No matter how many threads you have, only ONE core will be used
- Really bad for CPU bound tasks
- GIL is released during I/O, so not so bad for I/O bound tasks
- for CPU bound tasks, use can separate processes, instead of threads
- however, processes are more "heavyweight" than threads, and do not share memory
- can move CPU bound tasks into to C - ctypes releases the GIL on a C function call
- Java and C++ do not have this problem

4 multiprocessing module

- run multiple Python processes
- avoids the GIL
- doc

```
In [3]: from multiprocessing import Pool

        def square(x):
            return x*x

        # make a pool of 5 pythons
        # each square call will run in a separate Python executable
        p = Pool(5)
        print(p.map(square, [1, 2, 3]))
```

```
[1, 4, 9]
```

```
In [ ]: p.close()
```