uvloop: Python极速网络互连

2 年前 7888

> 本文作者为 Yury Selivanov，译者是 唐晓霆 Jason ，由 EarlGrey 校对。译者简介：唐晓霆，在香港的成都人，城市大学研究助理，会写python，兴趣是深度学习。

asyncio 是Python 标准库里的一个异步 I/O 框架。在本文中，我们将介绍 uvloop : 这是 asyncio 默认事件循环的一个代替品，实现的功能完整，且即插即用。uvloop 是用 Cython 写的，建于 libuv 之上。

uvloop 可以使 asyncio 更快。事实上，它至少比 nodejs、gevent 和其他 Python 异步框架要快 **两倍** 。基于 uvloop 的 asyncio 的速度几乎接近了 Go 程序的速度。

asyncio & uvloop

Asyncio 模块在 PEP 3156 中引入，是一个网络传输、协议和流量抽象化等的集合，带有一个可插换的事件循环。这个事件循环是 asyncio 的核心。它给以下功能提供了 API：

- 安排函数调用，
- 通过网络传输数据，
- 执行 DNS 询问，

推荐阅读 热门文章 随机

- 20天持续压测，云存储性能哪家更强?
- 国内公有云大幅降价后，首份一手云计算产品评测报告
- Python进阶、求职必看的前辈经验分享
- 硅谷码农用Python写了个机器人，租到了让女友满意的房子
- 使用 Python 进行科学计算：NumPy入门
- 十分钟入门Matplotlib
- 从零开发一个小游戏：PyGame 入门
- 好用！在 Notebook 中使用 Sublime Text 快捷键
- 十张GIFs让你弄懂递归等概念

热门标签

IDE PyCon 编译 Flask Codewars Postgresql Django Docker Git 程序员 开发库 漫画 编码风格

经典书籍

《Think Python 2ed》最新中文

- 处理 OS 信号，
- 可创建服务器和连接的方便抽象类
- 异步地处理 subprocess

> 截止目前，uvloop 还智能在 *nix 平台 和 Python 3.5 中使用。

uvloop是一个对 asyncio 默认事件循环的代替品。你可以用 pip 安装它：

```
$ pip install uvloop
```

在 asyncio 代码里面使用 uvloop 也很简单：

```
import asyncio
import uvloop
asyncio.set_event_loop_policy(uvloop.EventLoopPolicy())
```

上面这段代码使得任何对 asyncio.get_event_loop() 的调用都将返回一个 uvloop 的实例。

架构

uvloop是用 Cython 写的，其基础是 libuv。

libuv 是 nodejs 使用的一个高性能、多平台的异步 I/O 库。由于 nodejs 使用很广也很流行，使得 libuv 又快又稳定。

uvloop 实现了所有 asyncio 里面的事件循环 API 。高级 Python 对象封装了底层的 libuv 结构体和函数。为了让代码干净、不重复，并保证手动内存管理都和 libuv 的原语生命周期保持一致，uvloop 使用了子类继承的方法。

基准测试

为了比较 uvloop 实现和其他实现方法的性能区别，我们创建了一个工具平台（tool bench），来对 TCP 和 UNIX 套接字 I/O 和 HTTP 协议的性能进行测试。

基准测试服务器在一个 Docker 容器里面运行，外面有一个负载生成工具(测试 HTTP 协议则使用 wrk)，负责评估请求吞吐量和延迟。

扫码关注编程派

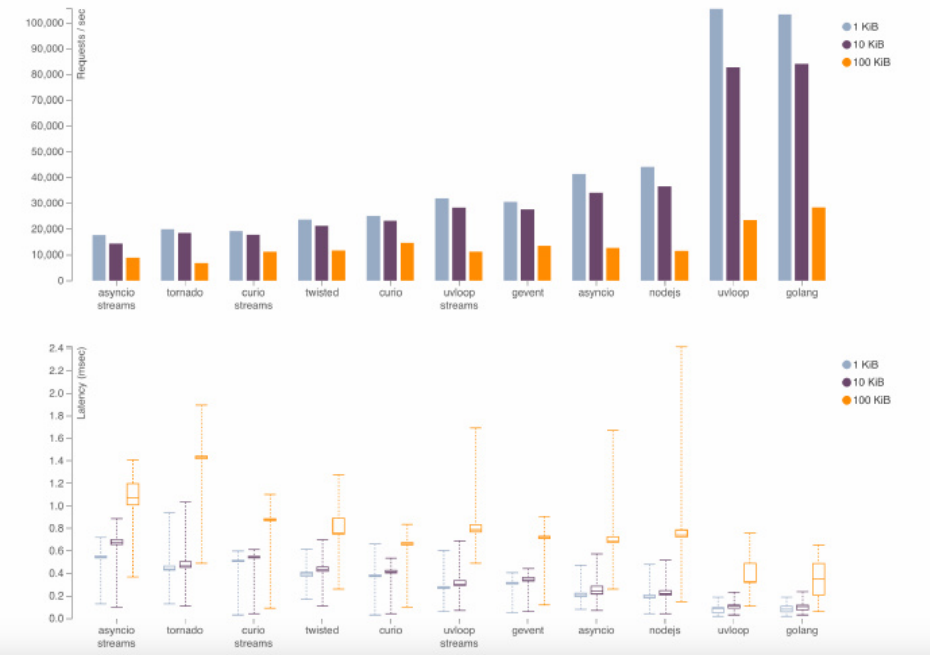
本文提到的所有基准测试都是在一台装了 Ubuntu 的 Linux 机器上运行的。机器搭载了英特尔 Xeon CPU E5-1620 v2 @ 3.70GHz。我们用的是 Python 3.5，并且所有服务器都是单线程的。除此之外，对 Go 代码我们设定 `GOMAXPORCS=1`， nodejs 不使用集群，并且所有 Python 服务器都是单进程的。每个基准测试都设置了 `TCP_NODELAY` 标志。

在Mac OS X上运行的基准测试得到了类似的结果。

TCP

该基准测试通过不同大小的消息，对一个简单的 echo 服务器的性能进行了检测。我们使用了 1、10和 100KB 的包。并发级别设为 10，每个基准测试运行30秒。

参见完整的TCP基准测试报告



> *所有基准测试的代码在这里 也可以看看Unix Socket的基准测试*

简单评论一下每个位置的情况：

1. *asyncio-steams*。 使用内置的纯 Python 事件循环的 asyncio 。在这个基准测试里，我们测试了高级别流抽象的性能。我们用 `asyncio.create_server()` 来创建一个服务器。这个服务器回传一对 `(reader, writer)` 给客户端的协同程序。
2. *tornado*。 这个服务器实现了一个简单的 Tornado 协议，可以立即传回它收到的任何数据。

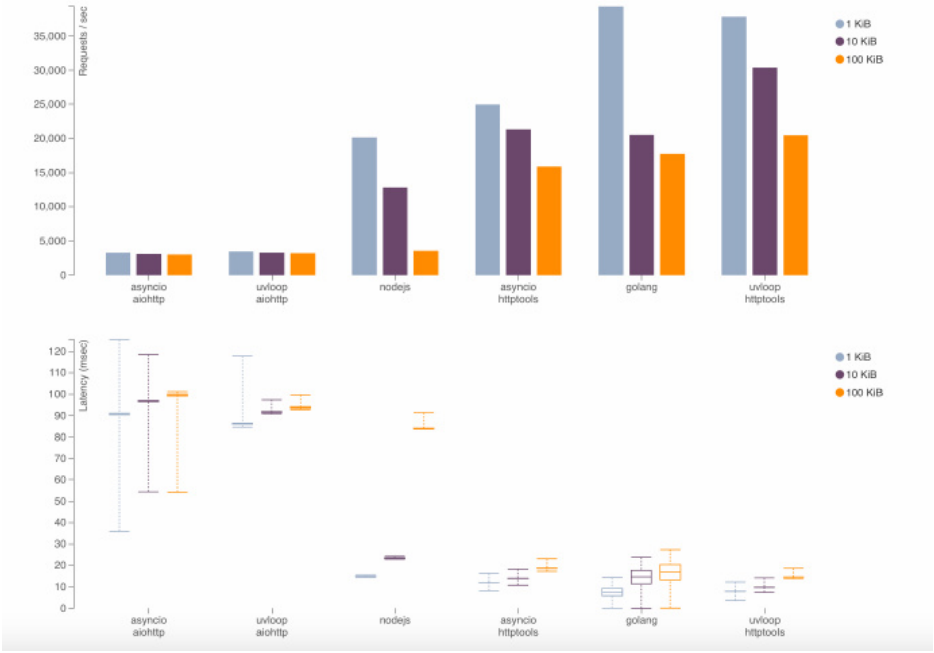
3. *curio-steams*。Curio 是Python 异步库中的新生儿。和 *asyncio-steams* 一样，在本次基准测试里我们打算测试 curio 的数据流。我们使用 curio.make_steams() 来创建一对 (reader, writer) ， 提供了一些高级API，比如 readline() 。
4. *twisted*。和Tornado类似，这里我们测试了一个最简单的 echo 的协议。
5. *curio*。 这个基准测试检验 curio 套接字的性能：一个由 sock.recv() 和 sock.sendall() 协同程序组成的紧凑循环。
6. *uvloop-streams*。 就像第二个(*tornado*)一样，这里我们测试 asyncio 高级数据流的性能，只不过这次我们使用的是 uvloop 。
7. *gevent*。我们用 gevent.SteamServer 和一个 gevent 套接字，在紧凑循环中来发送/接收数据。
8. *asyncio*。看起来原生的 asyncio 也很快！和第 2 个(*tornado*)和第 4 个(*twisted*)类似，这里我们测试一个最简的 echo 协议的性能。这个协议是用纯 Python 的 asyncio 实现的。
9. *nodejs*。我们用 net.createServer API来测试 nodejs v4.2.6 中的数据流性能。
10. *uvloop*。 这个基准测试中，我们用以 uvloop 为基础的 asyncio 实现一个最简单的 echo 协议(像第2、4、8个一样)，并对该协议的性能进行测试。用 1KB 的信息， uvloop 是最快的实现，每秒达到了 105，000 次请求！对于 100KB 的信息来说，uvloop 的传输速度可以做到 2.3 GB/s。
11. *Go*。使用由 net.Conn.Read/Write 调用组成的紧凑循环。Golang 的性能和 uvloop 十分相似，对于 10KB 和 100KB 的信息来说性能稍好一些。

HTTP

一开始，我们想比较搭建在 asyncio 和 uvloop 之上的 aiohttp 与 nodejs、Go 的性能差别。aiohttp 是用 asyncio 搭建异步 HTTP 服务器最流行的框架。

然而，aiohttp 的性能瓶颈竟然是它的 HTTP 解析器。这个解析器的速度非常慢，导致底层 I/O 库再快也没有用。为了让事情更有趣一点，我们为 http-parser (nodejs 中的 HTTP 解析器，用 C 编写，一开始为 Nginx 开发) 创建了一个 Python 绑定。这个库叫作 httptools ，可在 Github 和 PyPI上找到。

对于 HTTP ，所有的基准测试都使用的 wrk 来生成负载。并发级别设置为 300。每次基准测试的时间为30秒。



出人意料的是，有了高性能 HTTP 解析器的帮助，纯 Python 的 asyncio 的速度超过了nodejs，而后者用的也是同一种 HTTP 解析器！

Go在 1KB 的响应上的性能比较快，但是 uvloop+asyncio 的实现在 10/100KB 的表现上明显比较快。对于用 httptools 的 asyncio 和 uvloop 而言，它们的性能非常棒。Go 语言也是一样。

不可否认的是，基于 httptools 的服务器非常的简单，而且比起其他实现方法来，也不包括其他的路由逻辑。然而，这次的基准测试证明了配合一个实现得很有效率的协议，uvloop 可以变得非常之快。

结论

我们可以安全地下结论说，有了 uvloop，我们可以写出每秒每 CPU 核心可以推送上万次请求的 Python 网络互连代码。在一个多核心系统上，用上进程池，也许还可以进一步地提高性能。

在 Python 3.5 中，配合async/await的力量， uvloop 和 asyncio 使得用 Python 写出高性能的网络互连代码比以前任何时候都简单。

试一试 uvloop(github)，跟我们分享你的结果吧！

点此查看原文链接。

Python 翻译组是EarlGrey@编程派发起成立的一个专注于 Python 技术内容翻译的小组，目前已有近 30 名 Python 技术爱好者加入。