ПРОГРАММИРОВАНИЕ*, PYTHON*

Да, Python медленный, но меня это не волнует

ПЕРЕВОД

TyVik 2 июня 2017 в 08:11 46,2k

Оригинал: Nick Humrich

Разговоры о снижении производительности ради продуктивности.

Я беру паузу в моём обсуждении asyncio в Python, чтобы поговорить о скорости Python. Позвольте представиться, я — ярый поклонник Python, и использую его везде, где только удаётся. Одна из причин, почему люди выступают против этого языка, — то, что он медленный. Некоторые отказываются даже попробовать на нём поработать лишь из-за того, что «X быстрее». Вот мои мысли на

этот счёт.

Производительность более не важна

Раньше программы выполнялись очень долго. Ресурсы процессора и памяти были дорогими, и время работы было важным показателем. А уж сколько платили за электричество! Однако, те времена давно прошли, потому что главное правило гласит:

> Оптимизируйте использование своих самых дорогих ресурсов.

Исторически самым дорогим было процессорное время. Именно к этому подводят при изучению информатики, фокусируясь на эффективности различных алгоритмов. Увы, это уже не так — железо сейчас дёшево как никогда, а вслед за ним и время выполнения становится всё дешевле. Самым дорогим же становится время сотрудника, то есть вас. Гораздо важнее решить задачу, нежели ускорить выполнение программы. Повторюсь для тех, кто просто пролистывает статью:

> Гораздо важнее решить задачу, нежели ускорить выполнение программы.

Вы можете возразить: «В моей компании заботятся о скорости. Я создаю веб-приложение, в котором все ответы приходят меньше, чем за x миллисекунд» или «от нас уходили клиенты, потому что считали наше приложение слишком медленным». Я не говорю, что скорость работы не важна совсем, я хочу лишь сказать, что она перестала быть самой важной; это уже не самый дорогой ваш ресурс.

Скорость — единственная вещь, которая имеет значение.

Когда вы употребляете слово «скорость» в контексте программирования, скорей всего вы подразумеваете производительность CPU. Но когда ваш CEO употребляет его применительно к программированию, то он подразумевает скорость бизнеса. Самая главная метрика — время выхода на рынок. В конечном счёте неважно насколько быстро работают ваши продукты, не важно на каком языке программирования они написаны или сколько требуется ресурсов для их работы. Единственная вещь, которая заставит вашу компанию выжить или умереть, — это время выхода на рынок. Я говорю больше не о том, насколько быстро идея начнёт приносить доход, а о том, как быстро вы сможете выпустить первую версию продукта. Единственный способ выжить в бизнесе — развиваться быстрее, чем конкуренты. Неважно, сколько у вас идей, если конкуренты

реализуют их быстрее. Вы должны быть первыми на рынке, ну или как минимум не отставать. Чуть затормозитесь — и погибните.

> Единственный способ выжить в бизнесе — развиваться быстрее, чем конкуренты.

Поговорим о микросервисах

Крупные компании, такие как Amazon, Google или Netflix, понимают важность быстрого развития. Они специально строят свой бизнес так, чтобы быстро вводить инновации. Микросервисы помогают им в этом. Эта статья не имеет никакого отношения к тому, следует ли вам строить на них свою архитектуру, но, по крайней мере, согласитесь с тем, что Amazon и Google должны их использовать. Микросервисы медленны по своей сути. Сама их концепция заключается в том,

чтобы использовать сетевые вызовы. Иными словами, вместо вызова функции вы отправляетесь гулять по сети. Что может быть хуже с точки зрения производительности?! Сетевые вызовы очень медленны по сравнению с тактами процессора. Однако, большие компании по-прежнему предпочитают строить архитектуру на основе микросервисов. Я действительно не знаю, что может быть медленнее. Самый большой минус такого подхода — производительность, в то время как плюсом будет время выхода на рынок. Создавая команды вокруг небольших проектов и кодовых баз, компания может проводить итерации и вводить инновации намного быстрее. Мы видим, что даже очень крупные компании заботятся о времени выхода на рынок, а не только стартапы.

Процессор не является вашим узким местом

Если вы пишете сетевое приложение, такое как веб-сервер, скорее всего, процессор не является узким местом вашего приложения. В процессе обработки запроса будет сделано несколько сетевых обращений, например, к базе данных или кешу. Эти сервера могут быть сколь угодно быстрыми, но всё упрётся в скорость передачи данных по сети. Вот действительно классная статья, в которой сравнивается время выполнения каждой операции. В ней автор считает за один такт процессора одну секунду. В таком случае запрос из Калифорнии в Нью-Йорк растянется на 4 года. Вот такая вот сеть медленная. Для примерных расчётов предположим, что передача данных по сети внутри датацентра занимает около 3 мс, что эквивалентно 3 месяцам по нашей шкале отсчёта. Теперь возьмём программу, которая нагружает CPU. Допустим, ей надо

100 000 циклов, чтобы обработать один вызов, это будет эквивалентно 1 дню. Предположим, что мы пишем на языке, который в 5 раз медленнее — это займёт 5 дней. Для тех, кто готов ждать 3 месяца, разница в 4 дня уже не так важна.

В конечном счёте то, что python медленный, уже не имеет значения. Скорость языка (или процессорного времени) почти никогда не является проблемой. Google описал это в своём исследовании. А там, между прочим, говорится о разработке высокопроизводительной системы. В заключении приходят к следующему выводу:

> Решение использовать интерпретируемый язык программирования в высокопроизводительных приложениях может быть парадоксальным, но мы столкнулись с тем, что CPU редко когда является сдерживающим фактором; выразительность языка означает, что большинство программ невелики и большую часть времени тратят на ввод-вывод, а не на собственный код. Более того, гибкость интерпретируемой реализации была полезной, как в простоте экспериментов на лингвистическом уровне, так и в предоставлении нам возможности исследовать способы распределения вычислений на многих машинах.

Другими словами:

> CPU редко когда является сдерживающим фактором.

Что, если мы всё-таки упираемся в CPU?

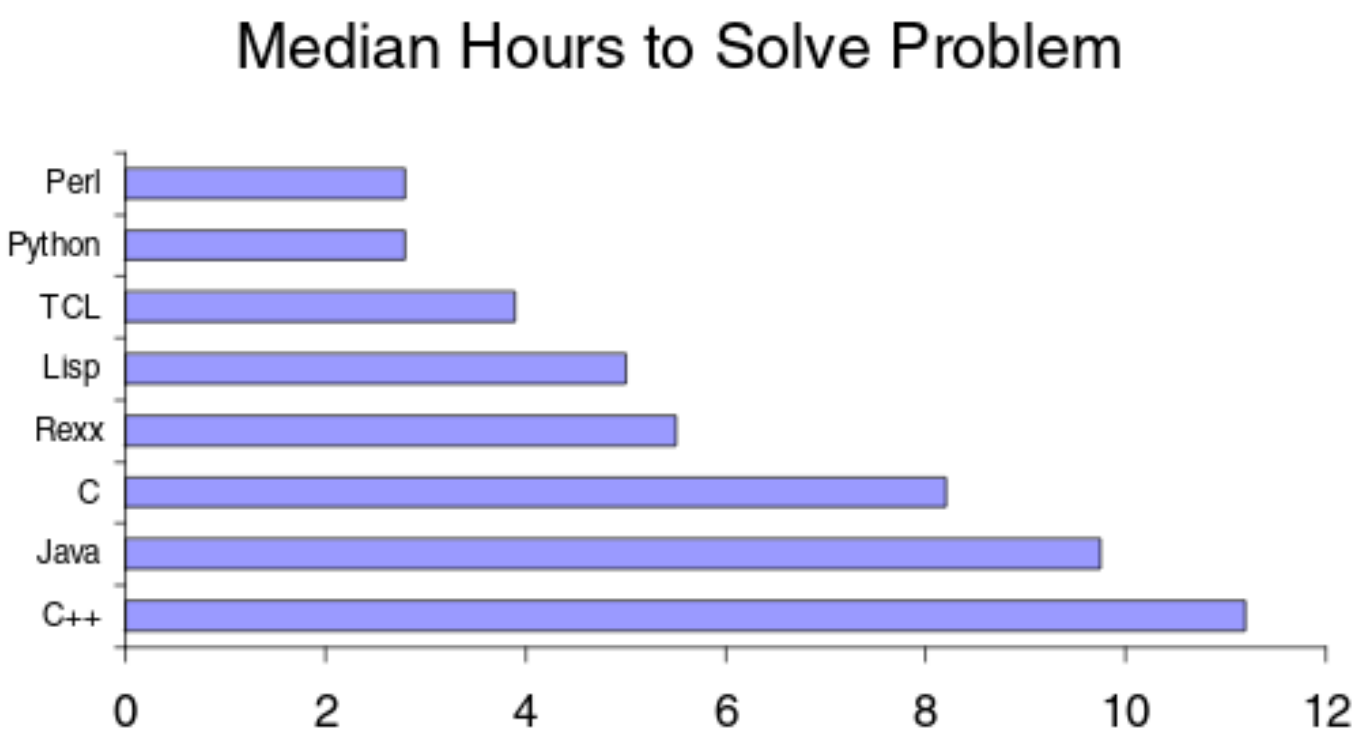
Что насчёт таких аргументов: «Всё это прекрасно, но что, если CPU становится узким местом и это начинает сказываться на производительности?» или «Язык x менее требователен к железу, нежели y»? Они тоже имеют место быть, однако вы можете масштабировать приложение горизонтально бесконечно. Просто добавьте серверов, и сервис будет работать быстрее :) Без сомнения, Python более требователен к железу, чем тот же C, но стоимость нового сервера гораздо меньше стоимости вашего времени. То есть дешевле будет докупить ресурсов, а не бросаться каждый раз что-то оптимизировать.

Так что, Python быстрый?

На протяжении всей статьи я твердил, что самое важное — время

разработчика. Таким образом, остается открытым вопрос: быстрее ли Python языка X во время разработки? Смешно, но я, Google и кое-кто ещё, могут подтвердить насколько продуктивен Python. Он скрывает так много вещей от вас, помогая сосредоточиться на основной вашей задаче и не отвлекаясь на всякие мелочи. Давайте рассмотрим некоторые примеры из жизни.

По большей части все споры вокруг производительности Python скатываются к сравнению динамической и статической типизации. Думаю, все согласятся, что статически типизированные языки менее продуктивны, но на всякий случай вот хорошая статья, объясняющая почему. Что касается непосредственно Python, то есть хороший отчёт об исследовании, в котором рассматривается, сколько времени потребовалось, чтобы написать код для обработки строк на разных языках.



В представленном выше исследовании Python более чем в 2 раза

продуктивнее Java. Многие другие также языки программирования дают похожий результат. Rosetta Code провёл довольно обширное исследование различий изучения языков программирования. В статье они сравнивают python с другими интерпретируемыми языками и заявляют:

> Python, в целом, наиболее краток, даже в сравнении с функциональными языками (в среднем в 1,2-1,6 раза короче).

По-видимому, в реализации на Python будет как правило меньше строк кода, чем на каком-либо другом языке. Это может показаться ужасной метрикой, однако несколько исследований, в том числе и упомянутых выше, показывают, что время, затраченное на каждую строку кода, примерно одинаково на каждом языке. Таким образом, чем меньше строк кода, тем больше продуктивность. Даже сам codinghorror (программист на C#) написал статью о том, что Python более продуктивен.

Я думаю, будет справедливо сказать, что Python более продуктивен, чем многие другие языки программирования. В основном это связано с тем, что он поставляется «с батарейками», а также имеет множество сторонних библиотек на все случаи жизни. Вот небольшая статья, сравнивающая Python и X. Если вы мне не верите, можете сами убедиться насколько этот язык программирования прост и удобен. Вот ваша первая программа:

```
import __hello__
```

Но что, если скорость действительно важна?

Мои рассуждения выше могли сложить мнение, что оптимизация и скорость выполнения программы вообще не имеют значения. Во многих случаях это не так. Например, у вас есть веб-приложение и некоторый endpoint, который уж очень тормозит. У вас могут быть даже определённые требования на этот счёт. Наш пример строится на некоторых предположениях:

1. есть некоторый endpoint, который выполняется медленно
2. есть некоторые метрики, определяющие насколько медленными может быть обработка запросов

Мы не будем заниматься микро-оптимизацией всего приложения. Всё должно быть «достаточно быстро». Ваши пользователи могут заметить, если обработка запроса занимает секунды, но они никогда не заметят разницу между 35 мс и 25 мс. Вам нужно лишь сделать приложение «достаточно хорошим».

Дисклеймер

Чтобы понять как оптимизировать, мы должны сначала понять что именно надо оптимизировать. В конце концов:

> Любые улучшения, сделанные где-либо помимо узкого места, являются иллюзией. — Джин Ким.

Если оптимизация не устраняет узкого места приложения, то вы просто потратите впустую своё время и не решите реальную проблему. Вы не должны продолжать разработку, пока не исправите тормоза. Если вы пытаетесь оптимизировать нечто, не

зная конкретно что, то вряд ли результат вас удовлетворит. Это и называется «преждевременной оптимизацией» — улучшение производительности без каких-либо метрик. Д. Кнуту часто приписывают следующую цитату, хотя он утверждает, что она не его:

> Преждевременная оптимизация — корень всех зол.

Если быть точным, то более полная цитата:

> Мы должны забыть об эффективности, скажем, на 97% времени: преждевременная оптимизация — корень всех зол. Однако мы не должны упускать наши возможности в этих критических 3%.

Другими словами, здесь говорится, что большую часть времени вам не нужно думать об оптимизации. Код и так хорош :) А в случае, когда это не так, нужно переписать не более 3% Вас никто не похвалит, если вы сделаете обработку запроса на несколько наносекунд быстрее. Оптимизируйте то, что поддаётся измерению.

Преждевременная оптимизация, как правило, заключается в вызове более быстрых методов <прим пер. видимо, подразумеваются ассемблерные вставки> или использовании специфичных структур данных из-за их внутренней реализации. В университете нас учили, что если два алгоритма имеют одну асимптотику Big-O, то они эквивалентны. Даже если один из них в 2 раза медленнее. Компьютеры сейчас настолько быстры, что вычислительную сложность пора измерять на большом количестве данных. То есть, если у вас есть две функции O(log n), но одна в два раза медленнее другой, то это не имеет большого значения. По мере увеличения размера данных они обе начинают показывать примерно одно и то же время выполнения. Вот почему

преждевременная оптимизация — это корень всего зла; Это тратит наше время и практически никогда не помогает нашей общей производительности.

В терминах Big-O все языки программирования имеют сложность O(n), где n — кол-во строк кода или инструкций. Не имеет значения, насколько медленным будет язык или его виртуальная машина — все они имеют общую асимптоту. <прим. пер. автор имеет ввиду, что даже если сейчас язык X в два раза медленнее Y, то в будущем после оптимизаций они будут примерно равны по скорости> В соответствии с этим рассуждением можно сказать, что «быстрый» язык программирования всего лишь преждевременно оптимизированный, причём непонятно по каким метрикам.

Оптимизируя Python

Что мне нравится в Python, так это то, что он позволяет оптимизировать небольшой участок кода за раз. Допустим, у вас есть метод на Python, который вы считаете своим узким местом. Вы оптимизировали его несколько раз, возможно, следуя советам отсюда и отсюда, и теперь пришли к выводу, что уже сам Python является узким местом. Но он имеет возможность вызова кода на C, а это означает, что вы можете переписать этот метод на C, чтобы уменьшить проблему с производительностью. Вы без проблем можете использовать этот метод вместе с остальным кодом.

Это позволяет писать хорошо оптимизированные методы узких мест на любом языке, который компилируется в ассемблерный код, то есть вы продолжаете писать на Python большую часть времени и переходите к низкоуровневому программированию лишь вам это действительно нужно.

Существует также язык Cython, который является надмножеством Python. Он представляет из себя смесь с типизированным C. Любой код Python является также валидным кодом и на Cython, который транслируется в C. Вы можете смешивать типы C и утиную типизацию. Используя Cython, вы получаете прирост производительности только в узком месте, не переписывая весь остальной код. Так делает, например, EVE Online. Эта MMoRPG использует только Python и Cython для всего стека, проводя оптимизацию только там, где это требуется. Кроме того, есть и другие способы. Например, PyPy — реализация JIT Python, которая может дать вам значительное ускорение во время

выполнения долгоживущих приложений (например, веб-сервера), просто путем замены CPython (реализация по умолчанию) на PyPy.

Итак, основные моменты:

- оптимизируйте использование самого дорогого ресурса — то есть вас, а не компьютера;
- выбирайте язык/фреймворк/архитектуру, которая позволяет вам разрабатывать продукты как можно быстрее. Не стоит выбирать язык программирования лишь потому, что программы на нём работают быстро;
- если у вас проблемы с производительностью — определите, где именно;
- и, скорее всего, это не ресурсы процессора или Python;
- если это всё же Python (и вы уже оптимизировали алгоритм), реализуйте проблемное место на Cython/C;
- и побыстрее возвращайтесь к основной работе.

Спасибо за внимание!

Проголосовать:

+44

Поделиться:

Сохранить:

Комментарии (213)

Похожие публикации

Сравнение производительности процессоров Intel разных поколений

19

Enigma077 • 2 марта 2017 в 10:03

Почему не нужно учить python первым языком

342

ИЗ ПЕСОЧНИЦЫ

hatman • 21 февраля 2017 в 14:34

Сравнение производительности: curl, php curl, php socket, python pycurl

16

ИЗ ПЕСОЧНИЦЫ

Hukuta • 22 января 2013 в 16:15

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета) 69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов 20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE 24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor 10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle 6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке

31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s

49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

140

marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16