**УДК 004.438** **Информационные технологии**

***Олимпиев Никита Валерьевич,*** *магистрант, Национальный исследовательский университет ИТМО, Санкт-Петербург, Россия*

# РАЗРАБОТКА МЕТОДИКИ ТЕСТИРОВАНИЯ ИНТЕРПРЕТАТОРОВ ЯЗЫКА PYTHON ДЛЯ ВЕБ-ПРИЛОЖЕНИЙ

**Аннотация:** Работа посвящена исследованию методик тестирования интерпретаторов языка Python. Главным образом рассматривается вопрос важности и необходимости разработки методики тестирования интерпретаторов для веб-приложений. Авторы анализируют научные труды по теме, в результате чего приходят к выводу, в открытых источниках нет ни одной комплексной работы, посвященной одновременно сравнению основных интерпретаторов языка и веб-разработке. Основным проблемным моментом существующих публикаций является вопрос о независимости и актуальности результатов сравнительного анализа. Авторы разрабатывают собственную методику на основе инструмента wrk и основных Python-фреймворков. Предложенный подход позволяет разработчикам выбрать лучший интерпретатор для конкретного веб-приложения с учетом его особенностей.

**Ключевые слова:** интерпретаторы, Python, веб-фреймворки, тестирование, методика

**Annotation:** The paper is devoted to the study of testing methods for Python interpreters. The question of the importance and necessity of developing a methodology for testing interpreters for web applications is mainly considered. The authors analyze scientific papers on the topic, because of which they concluded that there is not a single comprehensive work in open sources dedicated to comparing the main language interpreters and web development. The main problematic aspect of existing publications is the question of the independence and relevance of the results

of comparative analysis. The authors develop their own methodology based on the wrk tool and basic Python frameworks. The proposed approach allows developers to choose the best interpreter for a particular web application, considering its features.

**Keywords:** interpreters, Python, web frameworks, testing, methodology

**Введение**

В современном мире для конкурентоспособной реализации веб-приложения важно сделать свой продукт функциональнее и производительнее, чем у конкурентов. Для веб-разработки все чаще используется один из самых популярных языков программирования Python [1]. Из-за растущего количества имплементаций языка для разработчиков на Python актуальной проблемой является выбор интерпретатора. Поскольку Python – это интерфейс с множеством имплементаций, важно выбирать наиболее эффективный интерпретатор для конкретных задач. Решением проблемы является проведение комплексного независимого исследования интерпретаторов на основе методики, учитывающей особенности веб-разработки. Целью нашей работы является разработка методики тестирования интерпретаторов языка Python для веб-приложений. Для достижения цели необходимо проанализировать существующие труды по теме и выделить используемые авторами подходы к проведению исследований, а также выделить и описать собственную методику по результатам анализа.

**Анализ научных источников**

Выделим лучшие практики из общедоступных исследований, обозначим используемые методы и дополним их для разработки методики тестирования интерпретаторов.

Основной метод, используемый во всех исследуемых авторами научных трудах, это метод бенчмаркинга. В работе [2] также были применен метод профайлинга для анализа кода и выделения малопроизводительных участков. Авторы запускали тесты для измерения времени выполнения каждого интерпретатора. Тесты выполнялись на двух разных компьютерах. Данные,

собранные из тестов, представляют собой результат, демонстрирующий производительность каждого интерпретатора для набора тестов.

В исследовании [3] оценка производительности производилась с помощью тестов из проекта Unladen Swallow, которые включали синтаксический анализ, сериализацию данных и манипуляции с HTML. Также бенчмаркинг авторы осуществляли при помощи pybench. В основу методики [4] легло тестирование start-up и steady-state производительности, то есть замер скорости запуска короткого (short-running) и долгого (long-running) приложения. Авторы замеряли время выполнения процессов, осуществляя 30 повторных замеров, для минимизации ошибок и усреднения аномальных значений.

В работе [5] в большей степени представлено исследование производительности интерпретатора CPython. Остальная часть работы посвящена сравнению интерпретаторов, особенностью которого является подробное представление перечня бенчмарков с разнообразными тестами (Pyston/minibenchmarks и Pyston/microbenchmarks). Авторы сравнивают между собой следующие интерпретаторы: CPython-3.3.0, Jython, IronPython, Pyston и PyPy и приходят к выводу, что на производительность интерпретаторов большое влияние оказывает диспетчеризация байт-кода, которую можно оптимизировать. В зависимости от особенностей тестов полученные результаты кардинально различаются, что еще раз подтверждает отсутствие единственного ответа на вопрос о лучшем интерпретаторе. Недостатком работы является ограниченное и неподробное описание используемых тестов.

R. Power и A. Rubinsteyn в своей работе «How fast can we make interpreted Python?» [6] отмечают, что производительность Python довольно низка по сравнению с современными реализациями таких языков, как Lua и JavaScript. Это происходит из-за того, что те же самые API и библиотеки, которые делают Python мощным языком, также очень затрудняют его эффективное быстрое выполнение. Авторы работы пытаются решить вопрос оптимизации и повышения эффективности базового интерпретатора CPython путем разработки и внедрения совместимого с CPython высокопроизводительного интерпретатора

байт-кода Falcon. В результате это помогло повысить производительность кода до 2.5 раз в определенных случаях. Однако такой подход является крайне трудоемким и требует высокой квалификации разработчиков.

Анализ работ по теме показал, что перечисленные проблемы в достаточной мере исследованы зарубежными авторами, за исключением нерешенных вопросов о независимости и актуальности результатов сравнительного анализа. Также в открытых источниках не было обнаружено ни одной комплексной работы, одновременно посвященной сравнению основных интерпретаторов языка и веб-разработке.

**Разработка и описание методики тестирования**

Для проводимого нами тестирования возьмем результаты CPython, как основного интерпретатора языка, за точку отсчета для сравнения. Основной упор в рамках нашего исследования сделаем на тестировании кода веб-приложений, созданных с помощью различных веб-фреймворков, многогранности результатов тестирования и их пользе для веб-разработчиков. В процессе работы используем следующие методы: тестирование, бенчмаркинг, контейнеризация, анализ и синтез. Методику посмотрим, исходя из цели тестирования – проанализировать разницу между производительностью интерпретаторов при воспроизведении тестов для различных веб-фреймворков с одинаковыми условиями. Обозначенная цель позволяет нам проводить исследование даже без наличия высокопроизводительного железа.

Для проведения тестирования воспользуемся подходом к бенчмаркингу из репозитория «web-frameworks» от The Benchmarker [7], суть которого заключается в выявлении и измерении различий производительности основных фреймворков для популярных языков программирования. Основной нашей методики является замер производительность кода во время выполнения, поскольку различные имплементации показывают различные результаты производительности в зависимости от особенностей фреймворка. Для формирования комплексной картины результатов тестирования нам необходимо определить набор показателей для сравнения выбранных веб-фреймворков.

Возьмем за основу код приложений, созданных с помощью Python-фреймворков, из репозитория [7]. Основными методами выберем наиболее распространенные GET и POST, методика будет включать в себя проведение тестирования для каждого фреймворка с отключенным логированием. Для обработки GET и POST запросов в тестируемом веб-приложении создадим функции с пустым телом ответа. Для запуска веб-приложений будем использовать WSGI HTTP сервер Gunicorn с Meinheld в качестве типа воркера. В общедоступных исследованиях Meinheld показывает себя как высокопроизводительный сервер, требующий минимальное количество ресурсов [8]. Gunicorn основан на pre-fork worker модели, таким образом, в ее основе лежит мастер-процесс, который управляет сетом процессов-воркеров. Тип воркера Meinheld также был выбран создателями репозитория «web-frameworks» [7] из-за легковесности реализации WSGI веб-сервера с поддержкой асинхронного ввода-вывода.

Еще одной основой планируемого тестирования является построение окружения на базе Docker. Для замеров производительности интерпретаторов воспользуемся wrk – HTTP-инструментом для бенчмаркинга [9]. Данный инструмент способен генерировать нагрузку при запуске на одном многоядерном ЦП. wrk взаимодействует со скриптами на языке Lua, благодаря чему мы имеем возможность рассчитать множество метрик, в том числе необходимые нам: количество запросов в секунду, время ожидания и его перцентили. Используем lua-скрипты для замеров результатов производительности GET и POST запросов. Основной функцией для скрипта на Lua является done(summary, latency, requests). Она принимает таблицу, содержащую результаты замеров (summary), и два объекта статистики: время выполнения каждого запроса (latency) и частоту запросов потока (requests).

Таким образом, наша методика построена на инструменте wrk и его применении в изолированной среде Docker-контейнера для веб-приложений на Python, созданных с помощью любых веб-фреймворков. Для каждого фреймворка и интерпретатора необходимо провести замеры с 1 потоком любой длительности от 60 секунд, после чего повторить замеры и рассчитать среднее

арифметическое результатов.

Для получения и обработки результатов тестирования можно воспользоваться любыми инструментами для сбора результатов работы wrk. В качестве примера будем использовать базу данных под управлением СУБД PostgreSQL. Для дальнейшей обработки и визуализации результатов воспользуемся библиотекой matplotlib в рамках экосистемы Python.

Разработанная нами методика использует и актуализирует лучшие исследовательские наработки в области тестирования и сравнительного анализа интерпретаторов. Показатели, полученные в результате работы инструмента wrk, позволят детально оценить производительность и ее перцентиль для каждого интерпретатора и веб-фреймворка в изолированной среде. Такой подход может быть использован веб-разработчиками в качестве основы для проведения тестирования интерпретаторов Python для веб-приложений с учетом требуемых особенностей.

**Библиографический список:**

1. TIOBE Index for December 2020. [Электронный ресурс] URL: https://www.tiobe.com/tiobe-index (дата обращения: 12.12.2021).

2. Roghult A. Benchmarking Python Interpreters: Measuring Performance of CPython, Cython, Jython and PyPy: Degree project in computer science and engineering. Sweden, 2016.

3. Krintz N. M. C., PengWu C. D. P. Understanding the potential of interpreter-based optimizations for Python // UCSB, Tech. Rep. Technical Report. 2010. С. 5-14.

4. Redondo J. M., Ortin F. A comprehensive evaluation of common python implementations // IEEE Software. 2014. Т. 32. №. 4. С. 76–84.

5. Cao H., Gu N., Ren K., Li Y. Performance research and optimization on CPython's interpreter // 2015 Federated Conference on Computer Science and Information Systems (FedCSIS). 2015. С. 435-441.

6. Power R., Rubinsteyn A. How fast can we make interpreted Python? //

CoRR. 2013. T. abs/1306.6047.

7. the-benchmarker/web-frameworks: Which is the fastest web framework? [Электронный ресурс] URL: https://github.com/the-benchmarker/web-frameworks (дата обращения: 09.03.2022).

8. Анализ производительности WSGI-серверов: Часть вторая. [Электронный ресурс] URL: https://habr.com/ru/post/427217/ (дата обращения: 13.01.2022).

9. wg/wrk: Modern HTTP benchmarking tool. [Электронный ресурс] URL: https://github.com/wg/wrk (дата обращения: 09.02.2022).