Fearless Concurrency vs GIL 0:0

Thomas Schorr
info@thomasschorr.de

Plone Tagung Dresden 2020

Übersicht

Noch ein WSGI-Server?!

- https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface: mehr als 30 Web Frameworks, die WSGI unterstützen, u.a. Bjoern, Gunicorn, uWSGI, Waitress, Werkzeug, ...

Noch ein WSGI-Server?!

- https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface: mehr als 30 Web Frameworks, die WSGI unterstützen, u.a. Bjoern, Gunicorn, uWSGI, Waitress, Werkzeug, ...
- `Hello Rust!`
- Erwartung: Rust ist in der Ausführung schneller und effizienter als Python
- Erwartung: Rust ist sicherer als C („Fearless Concurrency“)
- Kontrolle über GIL

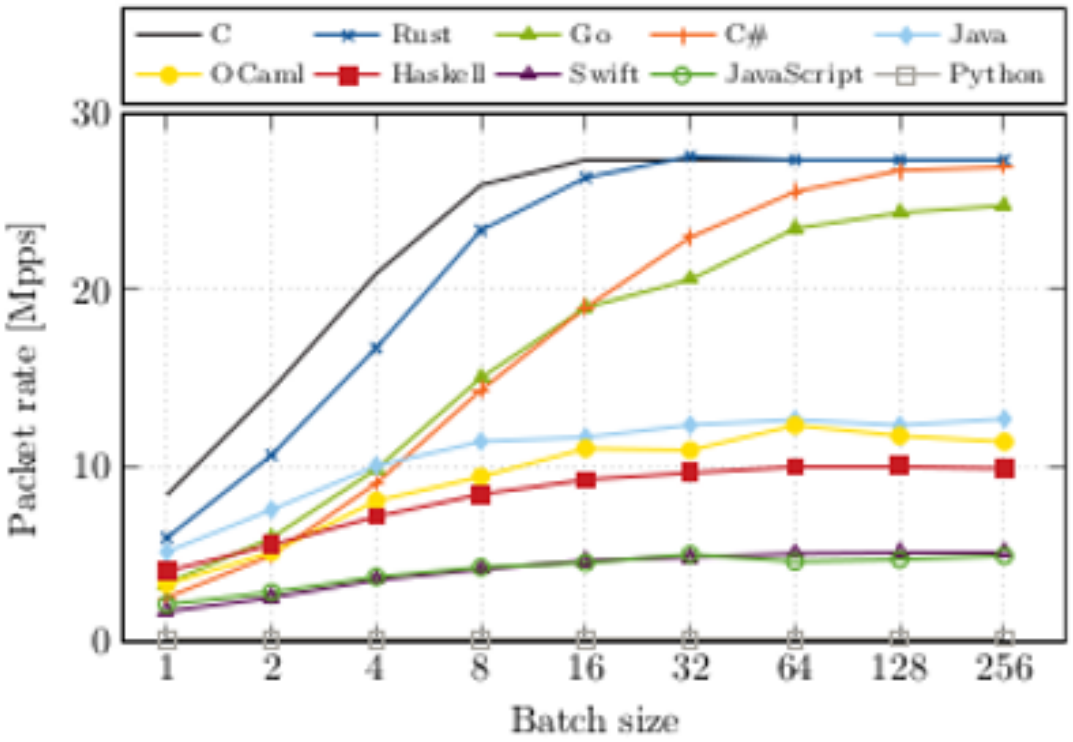
Noch ein WSGI-Server?!

- https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface: mehr als 30 Web Frameworks, die WSGI unterstützen, u.a. Bjoern, Gunicorn, uWSGI, Waitress, Werkzeug, ...
- `Hello Rust!`
- Erwartung: Rust ist in der Ausführung schneller und effizienter als Python
- Erwartung: Rust ist sicherer als C („Fearless Concurrency“)
- Kontrolle über GIL
  - IO
  - in welchen Situationen bringt das Vorteile?

Noch ein WSGI-Server?!

- https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface: mehr als 30 Web Frameworks, die WSGI unterstützen, u.a. Bjoern, Gunicorn, uWSGI, Waitress, Werkzeug, ...
- `Hello Rust!`
- Erwartung: Rust ist in der Ausführung schneller und effizienter als Python
- Erwartung: Rust ist sicherer als C („Fearless Concurrency“)
- Kontrolle über GIL
  - IO
  - in welchen Situationen bringt das Vorteile?
- Plone 5.2 verwendet WSGI
- Python 2 EOL

Warum Rust?



(b) Forwarding rate on 3.3 GHz CPU

Quelle: Emmerich, P. et al (2019): The Case for Writing Network Drivers in High-Level Programming Languages. - https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/the-case-for-writing-network-drivers-in-high-level-languages.pdf.

Ziele

- Nonblocking IO, Eventloop
- mehrere Worker
- Filewrapper mit sendfile Support
- PasteDeploy Entrypoint
- einfach zu verwenden
- performant

Pyruvate Komponenten

- PyO3: Python-Rust-Bindings (https://pyo3.rs)
- IO mit mio (Metal IO, https://github.com/tokio-rs/mio)
- httparse (https://docs.rs/httparse/1.3.4/httparse): Request-Parsing
- threadpool (https://docs.rs/threadpool)

PyO3

- Schnittstelle zwischen Rust und Python
- benötigt Rust nightly
- aktuelle Version 0.8.5 und 0.9.0-alpha1
- mögliche Alternativen:
  https://docs.rs/cpython/0.4.1/cpython/, ...

Speicherverwaltung in Python und Rust

- Python: Refcounts, Garbage Collection, GIL
- Rust: Ownership, Borrowing, Lifetimes, Typen (Sync, Send)
- PyO3 verknüpft diese Konzepte

mio; Metal IO

- „fast, low-level I/O library for Rust focusing on non-blocking APIs and event notification “
- Bestandteil von Tokio (https://tokio.rs/), einer asynchronen Platform in Rust
- je nach OS wird epoll, kqueue oder IOCP verwendet
- aktuelle Version 0.7.0

Non-Blocking IO

- Transportschicht (TCP)
- `fcntl(2)`, `select(2)`, `poll(2)`, `epoll(7)`
- EAGAIN/EWOULDBLOCK
- asyncore, asyncio, libev, mio, ...
- Futures, Promises als weierführende Abstraktionen

Pyruvate Rust Modul als Python Package

- Ziele:
  - Source Tarball (`sdist`) und Binärpakete
  - `pip` und `zc.buildout` Support
  - `mr.developer` Support
  - PasteDeploy Entrypoint
  - py.test Extra

Pyruvate Rust Modul als Python Package

- Ziele:
  - Source Tarball (`sdist`) und Binärpakete
  - `pip` und `zc.buildout` Support
  - `mr.developer` Support
  - PasteDeploy Entrypoint
  - py.test Extra
- Optionen: maturin (https://github.com/pyo3/maturin), setuptools-rust (https://github.com/PyO3/setuptools-rust)

Pyruvate Rust Modul als Python Package

- Ziele:
  - Source Tarball (`sdist`) und Binärpakete
  - `pip` und `zc.buildout` Support
  - `mr.developer` Support
  - PasteDeploy Entrypoint
  - py.test Extra
- Optionen: maturin (https://github.com/pyo3/maturin), setuptools-rust (https://github.com/PyO3/setuptools-rust)
  - pyproject.toml -> build-backend muss angegeben werden
  - maturin: neuer, gedacht als „Zero configuration replacement “für setuptools-rust
  - maturin: keine setuptools Unterstützung
  - Entry-Points, Extras, zc.buildout derzeit nur mit setuptools-rust

Rust Code Coverage messen - Tools

- Tarpaulin https://github.com/xd009642/tarpaulin
  - läuft nicht mit –test-threads=1
  - bei mehreren Test-Threads Test Isolation Probleme (Sockets binden...)
  - Schwierigkeiten mit Gitlab CI bei mehreren Threads
  - Werte waren sehr optimistisch

Rust Code Coverage messen - Tools

- Tarpaulin https://github.com/xd009642/tarpaulin
  - läuft nicht mit –test-threads=1
  - bei mehreren Test-Threads Test Isolation Probleme (Sockets binden...)
  - Schwierigkeiten mit Gitlab CI bei mehreren Threads
  - Werte waren sehr optimistisch
- Grcov https://github.com/mozilla/grcov
  - kommt nicht mit in-module Tests klar (misst Abdeckung des Testcodes mit)

Rust Code Coverage messen - Tools

- Tarpaulin https://github.com/xd009642/tarpaulin
  - läuft nicht mit –test-threads=1
  - bei mehreren Test-Threads Test Isolation Probleme (Sockets binden...)
  - Schwierigkeiten mit Gitlab CI bei mehreren Threads
  - Werte waren sehr optimistisch
- Grcov https://github.com/mozilla/grcov
  - kommt nicht mit in-module Tests klar (misst Abdeckung des Testcodes mit)
- Kcov https://github.com/SimonKagstrom/kcov

Rust Code Coverage messen - Tools

- Tarpaulin https://github.com/xd009642/tarpaulin
  - läuft nicht mit –test-threads=1
  - bei mehreren Test-Threads Test Isolation Probleme (Sockets binden...)
  - Schwierigkeiten mit Gitlab CI bei mehreren Threads
  - Werte waren sehr optimistisch
- Grcov https://github.com/mozilla/grcov
  - kommt nicht mit in-module Tests klar (misst Abdeckung des Testcodes mit)
- Kcov https://github.com/SimonKagstrom/kcov
- https://codecov.io hat offiziellen Rust-Support, https://coveralls.io nicht

Rust Code Coverage messen - Tools

- Tarpaulin https://github.com/xd009642/tarpaulin
  - läuft nicht mit –test-threads=1
  - bei mehreren Test-Threads Test Isolation Probleme (Sockets binden...)
  - Schwierigkeiten mit Gitlab CI bei mehreren Threads
  - Werte waren sehr optimistisch
- Grcov https://github.com/mozilla/grcov
  - kommt nicht mit in-module Tests klar (misst Abdeckung des Testcodes mit)
- Kcov https://github.com/SimonKagstrom/kcov
- https://codecov.io hat offiziellen Rust-Support, https://coveralls.io nicht
- Informationsstand 5.3.2020

Version 0.2 auf PyPI

- nicht produktiv einsetzbar, „Proof of Concept “
- 1:1 Threading für Worker
- asynchroner Eventloop
- sendfile Support
- Gitlab Pipeline

Version 0.2 auf PyPI

- nicht produktiv einsetzbar, „Proof of Concept “
- 1:1 Threading für Worker
- asynchroner Eventloop
- sendfile Support
- Gitlab Pipeline
  - Rust Unittests
  - Coverage (codecov.io)
  - wenige Python Integrationstests (`py.test`, `tox`)
  - Manylinux Wheels für Python 3.6, 3.7, 3.8

Version 0.3 als MVP

- HTTP/1.1
  - Connection Handling, keep-alive
- 85% Test Coverage
- sollte einfachere Anwendungsfälle abdecken

Ausblick

- mehr Konfigurationsmöglichkeiten:
  - maximale Anzahl/Größe an Requestheader, -body
  - Socket Activation
- Logging
  - Structured Logging `slog`
  - Integration mit Python Logging
- Unix Domain Sockets
- Expect/Continue
- HTTP 1.0

Non-Goals

- Windows
- WSGI API
- Legacy Features (`write`-Callable zurückgeben ...)
- (Python 2)

Pyruvate mit plone.recipe.zope2instance installieren

- buildout.cfg

```
[instance]
recipe = plone.recipe.zope2instance
http-address = 127.0.0.1:8080
eggs =
Plone
pyruvate
wsgi-ini-template = ${buildout:directory}/
    templates/pyruvate.ini.in
```

- pyruvate.ini.in Template

```
[server:main]
use = egg:pyruvate#main
socket = %(http_address)s
workers = 4
```

Live Demo

Vielen Dank für Eure Aufmerksamkeit

- Thomas Schorr
- info@thomasschorr.de

- https://gitlab.com/tschorr/pyruvate
- https://pypi.org/project/pyruvate