ENSAIO SOBRE APLICAÇÃO JIT USANDO PYTHON E PYPY

Horácio Dias Baptista Neto[1]

Resumo: O objetivo deste trabalho é fazer o estudo de uma aplicação web, executada em dois ambientes diferentes, buscando analisar qual cenário é mais robusto, atendendo o maior número de requisições em menor tempo. O primeiro, utilizando o interpretador Python padrão, chamado CPython, e o segundo, uma versão do interpretador que utiliza JIT em seu funcionamento, conhecida por Pypy. O trabalho analisa os dados quantitativos obtidos. Dessa maneira, podemos compreender em quais situações o JIT é vantajoso.

Palavras-chave: JIT, Python, Tradução Dinâmica, Programação.

1 INTRODUÇÃO

Atualmente, não é possível imaginar a tecnologia desvinculada da internet. S*martphones*, *tablets*, televisores, relógios e uma infinidade de *gadgets* que estão interconectados com a rede, provendo facilidades de acesso às informações e comodidades.

A maioria dos aparelhos, ao utilizar a internet, acaba acessando diversos tipos de aplicação, bem como as chamadas aplicações web. Com o crescente aumento de usuários da internet, as aplicações web tiveram de buscar maneiras de escalar e aumentar quantidade de requisições atendidas. Isto motivou o estudo nesta área e a elaboração deste trabalho.

Existem diversas maneiras de aumentar a quantidade de requisições atendidas por um servidor web, normalmente com a utilização de vários servidores e utilizando recursos físicos para isso. Porém existem alternativas que não necessariamente se aplicam ao *hardware*, mas às tecnologias utilizadas.

Assim, foi utilizado um servidor assíncrono chamado Tornado que, pelo fato dele não ser bloqueante, esse servidor consegue atender um grande volume de

[1]Aluno do 4ª ano do Curso de Sistemas de Informação. E-mail: horacio.dias92@gmail.com. Trabalho orientado pelo Prof. Kleber Oliveira e apresentado como requisito parcial para a formação no Curso de Sistemas de Informação, do Centro Universitário de Bauru no ano de 2016.

requisições. Foi analisado se o ambiente em que esse servidor é executado também interfere na quantidade de requisições atendidas. Por fim, os testes foram realizados em dois ambientes, o primeiro utilizando o interpretador padrão do Python chamado CPython, e um segundo que utiliza uma implementação JIT, chamado Pypy.

1.1 Objetivos

O objetivo deste trabalho é propor uma comparação entre o interpretador Python padrão com o Pypy, definindo em quais situações o Python é mais vantajoso que o Pypy. Isso foi feito por meio de requisições à páginas estáticas, gravação de dados via REST(*Representational State Transfer*), que é um modelo padrão adotado em aplicações web, recuperação de dados via REST, requisição de páginas com conteúdo dinâmico e a elaboração de relatórios.

1.2 Metodologia

Esse trabalho foi desenvolvido por meio de pesquisa a bases de dados de conhecimento, trabalhos e livros publicados sobre as ferramentas utilizadas. Foi utilizado o método quantitativo para aferir os resultados, segundo Pacheco e Teixeira:

> [...] os procedimentos quantitativos são úteis para identificar conceitos e variáveis relevantes de situações que possam ser comparadas e processadas quantitativamente [...] (TEIXEIRA; PACHECO, 2005).

Este trabalho está organizado da seguinte forma:

A partir do capítulo 2.1 o JIT é apresentado e seu funcionamento interno é descrito, exemplificando os passos e fases.

O capítulo 2.2 apresenta a linguagem Python e todo o seu ecossistema, comunidade, características e paradigmas suportados.

O capítulo 2.3 apresenta o interpretador Pypy e suas características básicas, além do subconjunto RPython.

O capítulo 2.4 apresenta os servidores web e como eles trabalham basicamente com HTTP, para disponibilizar páginas web.

O capítulo 2.5 apresenta o web framework Tornado, que é utilizado para a construção de aplicações web não bloqueantes.

O capítulo 2.6 apresenta o Gunicorn que é o servidor HTTP que atende as especificações WSGI utilizadas pelo Python.

O capítulo 2.7 apresenta a ferramenta de testes Apache JMeter, explica sobre a variedade de plugins e funcionalidades do mesmo.

O capítulo 2.8 apresenta o banco de dados não relacional MongoDB, que é baseado em documentos JSON e possui uma estrutura bem flexível e rápida para armazenar os dados.

O capítulo 3 apresenta a comparação entre os interpretadores, e como os testes foram realizados.

O capítulo 4 apresenta os resultados do trabalho na forma de tabelas, explicando os dados e sua importância.

O capítulo 5 apresenta a conclusão deste trabalho, onde evidenciamos o que os dados obtidos representam.

O capítulo 6 apresenta as possíveis maneiras de continuar este trabalho, em conjunto com outras técnicas e abordagens.

2 REFERENCIAL TEÓRICO

Este capítulo do trabalho procura demonstrar quais ferramentas e tecnologias foram utilizadas para a elaboração do mesmo.

2.1 JIT

JIT (*Just-in-Time*) é a compilação de programas em tempo de execução, com uma abordagem voltada à otimização. Um compilador JIT Tracing passa quatro fases em sua execução, sendo elas a Fase de Identificação, a Fase de Rastreamento, a Fase de Otimização e a Fase de Execução. Primeiro os *hotspots*, que são regiões do código onde um alto número de instruções são executadas, são identificados pela Fase de Identificação. Em seguida um tipo especial de chamada registra todas as operações desse *hotspot*. Essa sequência de operações é

chamada *trace*. Quando esse *hotspot* for executado novamente, uma versão compilada de seu *trace* será chamada.

A Fase de Identificação tem como objetivo identificar os *hotspots*. Essa verificação normalmente é feita por uma contagem do número de chamadas e/ou iterações para cada bloco. Após a contagem ultrapassar um determinado limite, o bloco passa a ser considerado um *hotspot*, então a máquina virtual marca este bloco para o modo *trace*, onde ela pode rastrear as chamadas a esse bloco.

Em seguida a Fase de Rastreamento se inicia. Nesta fase o bloco é executado normalmente, porém cada operação executada é registrada no *trace*. As informações são gravadas em uma forma de representação intermediária. Esse registrador segue o fluxo de chamadas e faz seu registro de maneira seqüencial no *trace*. A execução continua até que todas as operações do bloco sejam registradas.

Após a gravação das operações do bloco, o fluxo de execução deste bloco é percorrido à procura de caminhos onde a execução de operações possa divergir. Nesses pontos são inseridos ao *trace* uma instrução especial de guarda, pode-se entender por uma verificação condicional, onde uma rápida verificação ocorre para determinar se a condição original ainda é verdadeira. Caso a verificação falhe a execução do *trace* é abortada. Enquanto essa fase está em execução algumas informações correspondentes à execução deste bloco são armazenadas, como por exemplo, o tipo dos dados desse escopo, pois essas informações são utilizadas na fase de otimização. (WORLD EBOOK LIBRARY, 2016)

Na Fase de Otimização, as otimizações do código são realizadas são simples, pois são referentes a apenas um caminho de execução. Entre elas:

- Eliminação de expressões comuns;
- Eliminação de código inalcançável;
- Movimentação de código;
- Dobramento de constantes;
- Eliminação de desvios desnecessários;
- Análise de escape de ponteiros;
- Alocação de registros;

Depois das otimizações, o *trace* é transformado em código de máquina, que será executado pela Fase de Execução até que a instrução de guarda falhe.

2.2 Python

Python é uma linguagem de programação multiparadigma, conhecida por ser clara e simples. Possui uma característica marcante, que é a obrigação da indentação do código para definir os blocos.

Conforme expõe Eduardo Borges, em Python para Desenvolvedores:

> A linguagem de programação Python surgiu em 1990, foi criada por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI) e tinha originalmente foco em usuários como físicos e engenheiros. Python foi concebido a partir de outra linguagem existente na época, chamada ABC.(BORGES, 2014, p14).

Python é uma linguagem com código-fonte aberto e uso gratuito, que possui uma forte comunidade no Brasil e no mundo, sendo compatível com os sistemas operacionais predominantes no mercado, como por exemplo: Linux, OSX, Windows, BSD, etc. (CRUZ, 2015)

Ainda segundo o autor, a linguagem possui uma vasta biblioteca padrão, em conjunto a uma documentação rica de exemplos, possibilitando que os novos usuários explorem funcionalidades sem a necessidade de instalar dependências adicionais. Python vem sendo aplicada no desenvolvimento de projetos para os mais diversos fins: científicos, educacionais, comerciais, plataformas web, aplicações *desktop*, embarcados, e recentemente para dispositivos móveis.

Python é uma linguagem de baixa complexidade, pois com pouco código é possível realizar muitas tarefas, além de ser simples, com comandos claros escritos na língua inglesa e com pouca utilização de sinais e símbolos, como na definição de blocos, ou a não obrigatoriedade de parênteses em estruturas de decisão, resultando em uma alta produtividade. Seu uso teve um crescente aumento nos últimos anos em diversas áreas de atuação: no meio científico, acadêmico, no desenvolvimento de ferramentas, e no ensino de algoritmos e introdução à programação. (CRUZ, 2015)

Python é uma linguagem interpretada e seu funcionamento é semelhante aos de outras linguagens, como o Java onde o código-fonte é compilado para um *bytecode*, que é posteriormente interpretado por sua máquina virtual (Python Virtual Machine). Dessa forma podemos ter várias implementações do Python, que por sua

vez utilizam mecanismos diferentes para a compilação do código-fonte e interpretação dos *bytecodes*. Existem várias implementações do Python, como Jython (em Java), IronPython(.NET), Stackeless Python, Pypy, bem como a padrão CPython.

2.3 Pypy

Pypy é um projeto que resultou em um novo interpretador Python mais flexível e rápido que o convencional (CPython). Ele possui esse nome em razão de sua implementação utilizar o RPython, que é um dialeto da linguagem Python com algumas restrições. Assim, Pypy significa Python in Python. (PYPY'S... 2016)

Por ser uma linguagem dinamicamente tipada a performance de seu interpretador pode não ser tão eficiente, e é justamente com o foco em eficiência que surgiu o Pypy. Conforme expõe Luciano Ramalho, em Python Fluente:

> Se você estiver executando uma tarefa com uso intensivo de CPU em Python, experimente usar PyPy (http://pypy.org/)[...] (RAMALHO, 2015).

RPython é um framework para a implementação de interpretadores e máquinas virtuais para linguagens de programação, especialmente linguagens dinâmicas. Os autores enfatizam que o RPython não é um compilador de código Python.

O RPython é um subconjunto restrito da linguagem Python, que faz uma restrição na tipagem dinâmica para garantir a inferência de tipos. Normalmente as restrições limitam a capacidade de misturar tipos arbitrários, RPython não permite a ligação de dois tipos diferentes na mesma variável, semelhante as linguagens estaticamente tipadas como o Java. (RPYTHON DOCUMENTATION, 2016)

2.4 Servidor Web

Servidor Web é um computador que armazena arquivos que compõem um site, disponibilizando os mesmos ao usuário final. Os servidores web possuem configurações referentes ao acesso aos arquivos, basicamente utilizam o protocolo HTTP (*HyperText Transfer protocol*), e entendem URL's (*Uniform Resource Locator*) que são os endereços das páginas web. (MDN, 2016)

2.5 Tornado

O Tornado é um Web Framework assíncrono inicialmente desenvolvido pela FriendFeed, empresa que foi incorporada posteriormente pelo Facebook. Trata-se de um *framework* robusto no qual consegue atender mais de dez mil conexões abertas. O Tornado é utilizado no desenvolvimento de *web sockets* e aplicações web onde é necessário uma alta disponibilidade. (TORNADO WEB DOCUMENTATION, 2016)

2.6 Gunicorn

O Gunicorn é um servidor HTTP que serve aplicações que atendem as especificações *Web Server Gateway Interface*. A especificação WSGI é um padrão de como deve ser a comunicação entre o servidor e a aplicação Python. (DIGITALOCEAN, 2016)

Ele é baseado no modelo chamado *prefork*, onde um processo mestre controla outros subprocessos, que efetivamente atendem as requisições.

2.7 Apache JMeter

O Apache JMeter é uma ferramenta de código aberto que é executado na plataforma do Java, é utilizada para realizar testes de carga em sistemas web. O mesmo possui uma gama gigantesca de *plugins* e variadas configurações, possibilitando simular uma carga grande de requisições. O funcionamento dele segue simulando várias requisições simultâneas, que são disparadas para o servidor com os cabeçalhos e dados configurados pelo usuário, conforme as requisições são atendidas os dados colhidos são salvos. O JMeter conta com vários gráficos de saída, métodos de asserção, métodos de confirmação de erro ou determinado status de resposta. (DECOM, 2016)

2.8 MongoDB

O MongoDB é um banco de dados não relacional baseado em documentos no formato JSON(*Javascript Object Notation*). O MongoDB destaca-se por sua

estrutura maleável, onde novos atributos podem ser adicionados sem a necessidade de alteração na estrutura do documento. (ARTE DOS DADOS, 2016)

3 COMPARAÇÃO ENTRE INTERPRETADORES

O trabalho foi desenvolvido em ambiente Linux e os scripts foram executados por interpretadores instalados a partir do pyenv, que é um gerenciador de diferentes versões do interpretador Python. Também foi utilizado o virtualenv e o virtualenvwrapper, que são bibliotecas que encapsulam um determinado ambiente com um determinado interpretador. Isso foi necessário para que uma mesma máquina possa conter vários ambientes e cada qual com suas bibliotecas, possibilitando aos ambientes possuir bibliotecas em diferentes versões.

O Apache JMeter foi escolhido para obter e aferir os resultados sobre a quantidade de requisições e tempo de resposta. O banco de dados utilizado foi o MongoDB em conjunto com a biblioteca PyMongo.

O servidor web assíncrono chamado Tornado foi utilizado pelo fato dele não ser bloqueante, devido a isso esse servidor consegue atender um grande volume de requisições. (ELMAN; LAVIN, 2015)

Os testes foram realizados em dois ambientes, o primeiro utilizando o interpretador padrão do Python chamado de CPython, e um segundo que utiliza JIT em sua implementação, chamado Pypy. A escolha pelo Pypy neste trabalho se deve ao fato do mesmo possuir uma grande capacidade para processamento e uso do CPU se comparado ao interpretador padrão do Python.

Foram definidas algumas configurações para a execução dos testes. Todos os testes receberam 1000 requisições, com o Timeout padrão de 120000 milisegundos.

Não foi feito paginação ou algum tipo de otimização nos testes que utilizam dados dinâmicos, os dados utilizados na recuperação via REST, na página dinâmica e na geração dos relatórios são as 1000 requisições feitas na gravação via REST, porém os dados não foram paginados a fim de testar em um caso com grande volume de dados.

4 RESULTADOS

Na Tabela 1 – Resultado Requisição de páginas estáticas, Tabela 2 – Resultado Gravação de dados via REST, Tabela 3 – Resultado Recuperação de dados via REST, Tabela 4 – Resultado Requisição de páginas dinâmicas e Tabela 5 – Resultado Geração de relatórios, estão os dados referentes a Média em tempo de resposta em milissegundos, Min que é o tempo mínimo de resposta em milissegundos, Max que é o tempo máximo de resposta em milissegundos, a quantidade menor corresponde ao resultado mais rápido, e a porcentagem de erros.

Tabela 1 – Resultado Requisição de páginas estáticas.

% de erro	Max	Min	Média	
0	1022,00	2,10	202,00	Pypy
0	499,00	1,00	65,10	Python

Na Tabela 1 vemos que o interpretador Python teve o tempo médio, mínimo e máximo mais veloz do que o do interpretador Pypy, em ambos os testes não houveram falhas nas requisições.

Tabela 2 – Resultado Gravação de dados via REST.

% de erro	Max	Min	Média	
0	7542,00	375,00	2203,00	Pypy
0	1041,00	3,00	236,00	Python

Na Tabela 2 vemos uma diferença grande entre o tempo decorrido, o interpretador Python foi o mais rápido neste cenário em todos os aspectos e nenhuma requisição falhou para ambos dos interpretadores.

Tabela 3 – Resultado Recuperação de dados via REST.

% de erro	Max	Min	Média	
0	15133,00	785,00	4851,79	Pypy
0	15095,00	140,00	3274,00	Python

Na Tabela 3 vemos que o tempo máximo entre os dois interpretadores foi bem próximo, mas ainda sim o interpretador Python foi o mais rápido e nos dois cenários não houveram respostas com falhas.

Tabela 4 – Resultado Requisição de páginas dinâmicas.

% de erro	Max	Min	Média	
0	26299,00	286,00	4819,00	Pypy
0	16984,00	141,00	5066,00	Python

Na Tabela 4 vemos o único cenário onde o Pypy teve um tempo médio de resposta mais rápido que o interpretador padrão, nos outros casos o Python ainda foi superior. E não houve nenhuma requisição falha.

Tabela 5 – Resultado Geração de relatórios.

% de erro	Max	Min	Média	
31,4	120103,00	1314,00	25531,00	Pypy
20,4	120130,00	698,00	55564,00	Python

Na Tabela 5, vemos um valor médio menor para o Pypy, porém com uma porcentagem maior de erros, o que reduz numericamente a proporção dos dados avaliados, e impactando no resultado da média. Também vemos que o tempo máximo ficou ligeiramente superior ao limite de 120000 milisegundos. E o tempo mínimo das requisições também foi mais veloz no interpretador padrão.

5 CONCLUSÃO

De acordo com os dados obtidos, que o ambiente do Pypy não se mostrou eficiente. Seu tempo de resposta foi maior comparado ao CPython, e diante do número de requisições apresentou um maior número de falhas nas respostas.

Assim, a conclusão de que a contagem do trace e às execuções feitas pelo JIT levam um tempo consideravelmente maior comparado ao CPython, e que o Pypy não mostra uma eficiência maior quando usado em ambientes web com baixo nível de complexidade.

6 TRABALHOS FUTUROS

Como possíveis trabalhos futuros é interessante apontar a elaboração de ensaios com JIT utilizando inteligência artificial ou machine learning, devido a complexidade dos cálculos envolvidos nessas áreas, onde supostamente o uso do CPU seja maior.

Outro tipo de trabalho possível no futuro seria um ensaio com testes contendo processamento de cálculos de geografia espacial, ou testes com *bigdata* que utilizam um volume grande de dados, onde o JIT poderia se mostrar vantajoso.

ESSAY ON JIT APPLICATION USING PYTHON AND PYPY

Abstract: The aim of this work is to study a web application, performed in two different environments, trying to analyze which scenario is more robust, serving the largest numbers of requests in less time. The first using the standard Python interpreter, called CPython, and the second, a version of the interpreter that uses JIT in its operation, known as PyPy. The paper analyzes the quantitative data. In this way we can understand situations in which the JIT is advantageous..

Keywords: JIT, Python, Dynamic Translation, Programming.

REFERÊNCIAS

ARTE dos Dados: Python e MongoDB: porque usar e primeiros passos. Python e MongoDB: porque usar e primeiros passos. Disponível em: <http://artedosdados.blogspot.com.br/2013/07/python-e-mongodb-porque-usar-e.html>. Acesso em: 03 nov. 2016.

BORGES, Luiz Eduardo. **Python para Desenvolvedores.** São Paulo: Novatec, 2014. 320 p.

CRUZ, Felipe. **Python:** Escreva seus primeiros programas. São Paulo: Casa do Código, 2015. 252 p.

DECOM: Tutorial JMeter: Uso Do JMeter Para Testes De Desempenho Na Web. Tutorial JMeter: Uso Do JMeter Para Testes De Desempenho Na Web. Disponível em: <http://www.decom.ufop.br/imobilis/metodologia-de-testes-tutorial-jmeter-para-testes-de-performance-em-plataforma-web/>. Acesso em: 03 nov. 2016.

ELMAN, Julia; LAVIN, Mark. **Django Essencial:** Usando REST, websockets e Backbone. São Paulo: Novatec, 2015. 312 p.

DIGITALOCEAN: How to Deploy Python WSGI Apps Using Gunicorn HTTP Server Behind Nginx. Disponível em: <https://www.digitalocean.com/community/tutorials/how-to-deploy-python-wsgi-apps-using-gunicorn-http-server-behind-nginx>. Acesso em: 18 set. 2016.

MDN: What is a web server?. What is a web server?. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server>. Acesso em: 03 nov. 2016.

PYPY'S Documentation, JIT hooks. Disponível em: <http://doc.pypy.org/en/latest/jit-hooks.html>. Acesso em: 12 out. 2016.

RAMALHO, Luciano. **Python Fluente:** Programação clara, concisa e eficaz. São Paulo: Novatec, 2015. 800 p.

RPYTHON Documentation. Disponível em: <http://rpython.readthedocs.io/en/latest/faq.html#what-is-rpython>. Acesso em: 6 set. 2016.

TEIXEIRA, Rubens de França; PACHECO, Maria Eliza Corrêa. Pesquisa social e a valorização da abordagem qualitativa no curso de administração: a quebra dos

paradigmas científicos. **Cadernos de Pesquisa em Administração,** São Paulo, v. 12, n. 1, p.55-68, jan/mar. 2005.

TORNADO Web Documentation. Disponível em: <http://www.tornadoweb.org/en/stable/>. Acesso em: 7 ago. 2016.

WORLD Ebook Library, Tracing Just-In-Time Compilation. Disponível em: <http://www.ebooklibrary.org/article/WHEBN0035604013/Tracing just-in-time compilation>. Acesso em: 13 ago. 2016.