Refas Revista Fatec Zona Sul
1ª. Edição Outubro 2014

Logística
ISSN 2359-182X

ENSAIO SOBRE APLICAÇÃO JUST IN TIME (JIT): UM ESTUDO COMPARATIVO ENTRE INTERPRETADORES PYTHON E PYPY

JUST IN TIME (JIT) APPLICATION TEST: A COMPARATIVE STUDY BETWEEN PYTHON AND PYPY INTERPRETERS

Horácio Dias Baptista Neto[1]
Marcelo Lampkowski[2]
Kleber Rocha de Oliveira[3]

Artigo recebido em outubro de 2018

RESUMO

No cenário computacional, o termo JIT (*Just-in-Time*) refere-se à compilação de programas em tempo de execução, visando a otimização de todo o processo, uma vez que o algoritmo compilado dinamicamente pode levar em consideração as características da máquina real na qual ele está sendo executado e também pelo fato de traduzir blocos de código, em vez de avaliar e executar linha por linha, incrementando assim a sua performance. Este trabalho consiste na realização de um estudo sobre 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 cenário utilizou o interpretador Python padrão, chamado CPython, e, o segundo, uma versão do interpretador que utiliza JIT em seu funcionamento, conhecida por Pypy. Os dados quantitativos obtidos foram analisados e, dessa maneira, pôde-se compreender em quais situações o JIT era vantajoso. Evidenciou-se que o ambiente do Pypy não se mostrou eficiente, pois seu tempo de resposta foi maior quando comparado ao CPython e, diante do número de requisições, apresentou um maior número de falhas nas respostas. Observou-se que a contagem do *trace* e às execuções feitas pelo JIT levam um tempo consideravelmente maior comparado ao CPython.

Palavras-chave: CPython. Desenvolvimento. Programação. Pypy. Sistemas de informação.

ABSTRACT

In the computational scenario, the term JIT (Just-in-Time) refers to the runtime compilation of programs, seeking the optimization of the whole process, since the dynamically compiled algorithm can take into account the characteristics of the real machine in which it is being executed and also by translating blocks of code, instead of evaluating and executing line by line, thus increasing its performance. This work consists in the accomplishment of a study on a web application executed in two different environments, trying to analyze which scenario is more robust, attending the greater number of requests in a shorter time. The first scenario used the standard Python interpreter, called CPython, and the second, a version of the interpreter that uses JIT in its operation, known as Pypy. The quantitative data obtained were analyzed and it was possible to understand in which situations the JIT is advantageous. It was evidenced that the Pypy environment was not efficient, because its response time was higher when compared to CPython and, considering the number of requests, presented a higher number of failures in the responses. It was also observed that the trace count and JIT executions take considerably longer time compared to CPython.

Keywords: CPython. Development. Programming. Pypy. Information systems.

[1] ITE Bauru. E-mail: horacio.dias92@gmail.com.
[2] FATEC Jahu. E-mail: marcelo.lampkowski@fatec.sp.gov.br.
[3] ITE Bauru. E-mail: kleber@ite.edu.br.

1 INTRODUÇÃO

Atualmente, não é possível imaginar a tecnologia desvinculada da internet. *Smartphones*, *tablets*, televisores, relógios e uma infinidade de *gadgets* estão interconectados à rede mundial de computadores, provendo facilidades de acesso às informações e outras comodidades.

A maioria dos aparelhos, ao utilizar a internet, acaba acessando diversos tipos de aplicações, bem como as chamadas aplicações web. Com o crescente aumento de usuários 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.

O objetivo deste trabalho é propor uma comparação entre o interpretador Python padrão e o Pypy, definindo em quais situações a primeira opção é mais vantajosa que a segunda. Isso foi feito por meio de requisições a 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.

Para atingir tal objetivo, fez-se uso de um servidor assíncrono chamado Tornado para, posteriormente, analisar-se o ambiente de execução desse servidor e uma possível interferência na quantidade de requisições atendidas. Por fim, os testes foram realizados em dois ambientes: CPython e Pypy e um comparativo foi realizado.

2 REFERENCIAL TEÓRICO

Com o intuito de fundamentar teoricamente o trabalho, foi realizada uma pesquisa bibliográfica sobre os diversos temas relacionados ao tema da pesquisa. Assim, subitens a seguir procuram demonstrar quais ferramentas e tecnologias foram utilizadas para a elaboração deste estudo. Incluem-se as explicações sobre JIT (*Just-in-Time*), a linguagem Python, o interpretador Pypy, além de outras tecnologias como Tornado, Gunicorn, Apache JMeter e MongoDB.

2.1 Just in Time (JIT)

JIT 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 é executado, 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 sequencial 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 (BOLZ et al., 2012).

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

a) Eliminação de expressões comuns;
b) Eliminação de código inalcançável;
c) Movimentação de código;
d) Dobramento de constantes;
e) Eliminação de desvios desnecessários;
f) Análise de escape de ponteiros;
g) Alocação de registros.

Depois das otimizações, o trace é transformado em código de máquina, que será executada na 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 (RAMALHO, 2015).

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

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. 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 (CRUZ, 2015).

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).

Trata-se de 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, é possível 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 OFFICIAL DOCUMENTATION, 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.

RPython é um framework para a implementação de interpretadores e máquinas virtuais para linguagens de programação, especialmente linguagens dinâmicas. Os seus desenvolvedores enfatizam que o RPython não é um compilador de código Python (RPYTHON OFFICIAL DOCUMENTATION, 2016).

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 OFFICIAL 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 URLs (*Uniform Resource Locator*) que são os endereços das páginas web (MDN WEB DOCS, 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 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 (ELMAN; LAVIN, 2015).

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, *open source* e escrito em linguagem C++ com *drivers* disponíveis para as seguintes linguagens de programação: C, C++, C#, Java, Perl, PHP, Python, Ruby eScala (MONGODB DOCS, 2019).

Inicialmente, foi desenvolvido como um componente de serviço pela empresa 10gen, em outubro de 2007. Apenas em 2009 passou a ser um software open source (SOLAGNA; LAZZARETTI, 2016).

É orientado a documentos, o que significa que em vez de armazenar dados em estruturas rígidas, como as típicas tabelas do Modelo Relacional, eles armazena os dados em documentos vagamente definidos destacando-se por sua estrutura maleável, onde novos atributos podem ser adicionados sem a necessidade de alteração na estrutura do documento (CUNHA, 2011; ARTE DOS DADOS, 2016). Um documento possui uma estrutura comparável a um arquivo XML ou JSON. De fato, o MongoDB baseia os documentos em um formato chamado BSON, o qual é muito similar ao JSON, inclusive com o mesmo suporte a estruturas como *arrays* e *embedded objects*.

O MongoDB é classificado como um banco de dados NoSQL, termo genérico usado para uma classe definida de banco de dados que rompe o paradigma dos bancos de dados baseados nos Modelos Relacionais, apresentando, portanto características fundamentais como escalabilidade horizontal, esquema flexível e suporte nativo à replicação, que o tornam adequado para armazenamento de grandes volumes de dados não estruturados ou semiestruturados (LÓSCIO; OLIVEIRA; PONTES, 2011).

3 MÉTODO

O artigo 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 que, segundo Teixeira e Pacheco (2005), são úteis para identificar conceitos e variáveis relevantes de situações que possam ser comparadas e processadas quantitativamente.

Foi utilizado um servidor assíncrono chamado Tornado que, pelo fato dele não ser bloqueante, consegue atender um grande volume de requisições. Fez-se uso do gunicorn como servidor WSGI (*Web Server Gateway Interface*) com 5 *workers* para rodar o Tornado. 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.

Os testes foram realizados em um notebook Evolute SFX-65 L431, com processador Intel Core i5 2,53 GHz, com 4 GB de memória RAM e 500 GB de armazenamento.

Por fim, justifica-se o uso do MongoDB neste experimento por, no momento do desenvolvimento do trabalho, ser um dos poucos bancos de dados que operava de maneira assíncrona.

4 RESULTADOS E DISCUSSÃO

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 foram utilizadas a *virtualenv* e a *virtualenvwrapper*, que são bibliotecas que encapsulam um determinado ambiente com um determinado interpretador. Isso foi necessário para que uma mesma máquina pudesse 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.

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 1.000 requisições, com o *Timeout* padrão de 120.000 milissegundos.

Não foi realizada 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 1.000 requisições feitas na gravação via REST, porém os dados não foram paginados a fim de testar em um caso com grande volume de dados.

O Quadro 1 apresenta uma compilação dos resultados relacionados às situações estudadas, sendo: requisição de páginas estáticas, gravação de dados via REST, recuperação de dados via REST, requisição de páginas dinâmicas e geração de relatórios. Sempre há a comparação dos ambientes CPython e Pypy. Os dados são referentes à 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; e a porcentagem de erros.

Quadro 1 - Resultados das comparações entre os ambientes CPython e Pypy

Requisição de páginas estáticas				
	Média	Min	Max	% de erro
Pypy	202,00	2,10	1022,00	0
Python	65,10	1,00	499,00	0
Gravação de dados via REST				
Pypy	2203,00	375,00	7542,00	0
Python	236,00	3,00	1041,00	0
Recuperação de dados via REST				
Pypy	4851,79	785,00	15133,00	0
Python	3274,00	140,00	15095,00	0
Requisição de páginas dinâmicas				
Pypy	4819,00	286,00	26299,00	0
Python	5066,00	141,00	16984,00	0
Geração de relatórios				
Pypy	25531,00	1314,00	120103,00	31,4
Python	55564,00	698,00	120130,00	20,4

Fonte: Elaborada pelos autores

Em relação à requisição de páginas estáticas, observou-se que interpretador Python teve o tempo médio, mínimo e máximo mais veloz do que o do interpretador Pypy e que em ambos os testes não houveram falhas nas requisições.

Sobre os resultados de operações de gravação de dados via REST (*Representational State Transfer*), modelo padrão adotado em aplicações web, foi possível notar 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.

Os resultados relativos à recuperação de dados via REST mostrados na Tabela 1 permitem observar que o tempo máximo entre os dois interpretadores foi bem próximo, mas, ainda assim, o interpretador Python foi o mais rápido e em ambos os cenários não houve respostas com falhas.

No que se refere às requisições de páginas dinâmicas, este se mostrou como o único cenário onde o Pypy teve um tempo médio de resposta mais rápido que o interpretador padrão. Novamente, não houve nenhuma requisição falha.

Por fim, em relação aos resultados obtidos após as operações de geração de relatórios, verificou-se, neste caso, um valor médio menor para o Pypy. Porém, nota-se uma porcentagem maior de erros, o que reduz numericamente a proporção dos dados avaliados, impactando no resultado da média. Também se observa que o tempo máximo ficou ligeiramente superior ao limite de 120.000 milissegundos. O tempo mínimo das requisições também foi mais veloz no interpretador padrão.

Notou-se que os resultados obtidos se mostraram contrastantes a outros estudos, como o realizado por Lavrijsen (2012) em que o autor conclui que a utilização do Pypy foi mais eficiente por tornar possível compilar o código escrito em alto nível para linguagem de máquina, resultando em ganho de velocidade e performance, muitas vezes acima de 20 vezes mais em comparação com o CPython.

5 CONSIDERAÇÕES FINAIS

Conforme os dados obtidos neste estudo em particular, evidenciou-se que o ambiente do Pypy não se mostrou eficiente. Seu tempo de resposta foi maior quando comparado ao CPython e, diante do número de requisições, apresentou um maior número de falhas nas respostas. A contagem do *trace* e as execuções feitas pelo JIT levaram um tempo consideravelmente maior comparado ao CPython e que o Pypy não mostrou uma eficiência maior quando usado em ambientes web com baixo nível de complexidade.

O contraste com demais estudos, como o realizado por Lavrijsen (2012) pode ter sido influenciado pela diferença de versões dos interpretadores. No momento da realização dos testes, fez-se uso do interpretadores da versão 2.7 pelo fato de o interpretador que utiliza JIT (Pypy) para a versão 3.X do Python ainda estar em desenvolvimento. Existiam grandes expectativas em relação à nova versão do Pypy ser realmente mais rápida. Portanto, um trabalho futuro utilizando as novas versões de interpretadores poderia trazer resultados que corroboram com demais estudos já realizados.

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

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

6 REFERÊNCIAS

ARTE DOS DADOS. **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.

BOLZ, Carl Friedrich ; CUNI, Antonio ; FIJAŁKOWSKI, Maciej; RIGO, Armin. **Tracing the Meta-Level**: PyPy's Tracing JIT Compiler - ICOOOLPS '09. Anais do 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems. p. 18-25, 21 mar. 2012.

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.

CUNHA, T. M. de A. **Escalabilidade de sistemas com banco de dados NoSQL**: um estudo de caso comparativo com MongoDB e MySQL. Centro Universitário da Bahia – Estácio, Salvador, 2011. 85 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.

LAVRIJSEN, W. TLP. **Optimizing python-based ROOT I/O with PyPy's tracing just-in-time compiler**, Inter. Conf. on Computing in High Energy and Nuclear Physics, 2012.

LÓSCIO, B. F.; OLIVEIRA, H. R. de; PONTES, J. C. de S. **NoSQL no desenvolvimento de aplicações web colaborativas**. In: Simpósio Brasileiro de Sistemas Colaborativos – SBSC, 8., 2011, Paraty (RJ). Paraty: SBC, 2011.

MDN WEB DOCS. **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.

MONGODB DOCS. **Introduction to MongoDB**. Disponível em: <https://docs.mongodb.com/manual/introduction/>. Acesso em: 03 abr. 2019.

PYPY'S OFFICIAL 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 OFFICIAL DOCUMENTATION. **What is RPython?** Disponível em: <http://rpython.readthedocs.io/en/latest/faq.html#what-is-rpython>. Acesso em: 6 set. 2016.

SOLAGNA E. A.; LAZZARETTI, A. T. **Um estudo comparativo entre o MongoDB e o PostgreSQL**. IFSUL, Passo Fundo, 2016. Disponível em <http://painel.passofundo.ifsul.edu.br/uploads/arq/20160711180550101591419 8.pdf>. Acesso em 7 abr. 2019.

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