Testando a performance de tecnologias especialistas em *Rest API JSON* uma abordagem em JAVA, PHP, C++, NODEJS, RUBY, PYTHON e GO

Itacir Ferreira Pompeu [1], **Fernando Barbosa Matos** [1] **e Marcel da Silva Melo** [1]

[1]Instituto Federal de Educação, Ciência e Tecnologia Goiano Câmpus Morrinhos. Rodovia BR-153, KM 633 - Zona Rural, Morrinhos - GO, 75650-000 (64) 3413-7900.

itacir@hotmail.com,{fernando.matos, marcel.melo}@ifgoiano.edu.br

Abstract. *This article seeks to demonstrate the power of programming languages and frameworks in operating conditions using competitive and performance tests. The methodology used is the use of APACHE BENCHMARKING and SIEGE tools to generate load and observe the behavior and performance of the REST APIs for each of the following programming languages: Java, PHP, C ++, JavaScript (NodeJS), Ruby, Python and GO . The workload is the generation of HTTP requests to the GET method that returns a JSON with a string.*

Resumo. *Este artigo visa demonstrar o poder das linguagens de programação e frameworks em condições de operação utilizando testes de concorrência e performance. A metodologia utilizada consiste no uso das ferramentas APACHE BENCHMARKING e SIEGE para gerar carga e observar o comportamento e a performance das APIs REST de cada uma das seguintes linguagens de programação: JAVA, PHP, C++, JavaScript(NODEJS), Ruby, Python e GO. A carga de trabalho consiste na geração de requisições HTTP para o método GET que retorna um JSON com uma string.*

1. Introdução

O desenvolvimento de *WEB API* está em ênfase na atualidade e o seu uso vem crescendo a cada ano. Padrões como Estado Representacional de Transferência (*REST, Representational State Transfer*) [Saudate, 2013], Protocolo Simples de Acesso a Objetos (*SOAP, Simple Object Access Protocol*) [Saudate, 2013] são amplamente discutidos no mundo da engenharia de software. Quando observado o crescimento do acesso à internet por meio de dispositivos móveis, verifica-se que o uso de *JSON API* na entrega de dados é comum e usado em larga escala por grandes empresas.

A partir dessa necessidade foi realizado um comparativo entre as principais tecnologias para criação de *Web Service REST*. Este comparativo visa discutir a performance das principais tecnologias da atualidade que utilizam o Protocolo de Transferência de Hipertexto (*HTTP, HyperText Transfer Protocol*) [Saudate, 2013] para criação de rotas usando *REST* para entrega de *JSON (JavaScript Object Notation)*.

Em cenários onde existem limitações dos recursos de banda, o uso do *REST* provê respostas estruturadas criada permitindo que os navegadores possam interpretar e possuindo um formato padrão de chamadas, onde é utilizado os verbos do HTTP: GET, PUT,

POST e DELETE. O REST é normalmente utilizado em conjunto com AJAX *(Asynchronous JavaScript and XML)*, já que os navegadores modernos possuem suporte a estas tecnologias [Rozlog, 2013].

A motivação deste trabalho é necessidade de conhecer a capacidade das principais tecnologias utilizadas atualmente pelo mercado de trabalho e grandes empresas na criação de *Web Services*. Em paralelo a este trabalho estão sendo desenvolvidos alguns projetos de desenvolvimento de *software* pelo mesmo grupo de pesquisa onde estas aplicações necessitam do uso de serviços *REST*, e isso motivou a criação deste *benchmark* para validar as tecnologias utilizadas atualmente.

2. Trabalhos Relacionados

Existem cenários nos quais soluções que seguem arquitetura orientada a serviços, onde cada serviço deve desempenhar com perfeição sua função, a forma de gerar garantia na qualidade dos serviços web e a realização de teste [Correra, Alencar e Schmitz, 2012].

Para avaliar dados de aplicações web e validar seu funcionamento, antes de sua disponibilização aos usuários, emula-se o ambiente de forma que seja passível de verificar a utilização do cliente [Nogueira, 2014].

Essa área que visa pesquisar sobre performance das linguagens de programação e seus *micro-framework* ou bibliotecas específicas para criação ser *webservices JSON*, restringe-se muita ao mundo comercial e de grandes empresas, que fazem uso de alta concorrência em suas aplicações e por esse motivo trabalhos acadêmicos relacionados são raros [Correra, Alencar e Schmitz, 2012].

3. Metodologia

Duas ferramentas foram usadas para gerar a carga de trabalho e, por meio do resultado gerado, realizar o comparativo entre as linguagens de programação e *frameworks*. De acordo com [Fulmer, 2012] o *SIEGE* um utilitário para teste de carga sobre *HTTP* e *benchmarking* projetado para permitir que os desenvolvedores *web* possam medir o seu código sob coação, possibilitando verificar a reação do servidor no carregamento de conteúdo.

O *APACHE BENCHMARKING*(AB) é uma ferramenta de *benchmarking* para servidor *HTTP*. Ele é projetado para verificação de como o servidor executa. Em especial ele mostra o tempo das solicitações *HTTP* que o servidor é capaz de servir. *[*Apache Foundation, 2015].

3.1. Infraestrutura

Os testes se deram usando um microcomputador para servir as aplicações denominado *SERVER* uma máquina com sistema operacional Linux Ubuntu 14.10 64-bit, um processador de 2 núcleos modelo Intel *Core 2 Duo CPU E8400 3.00Ghz x2*, disco de 7200 RPM e 4 *GB* de memória DDR2 800MHZ.

O segundo microcomputador usado nos testes é denominado *CLIENT*, possui um processador de 4 núcleos Core I3 2400GHZ, disco de 5400 RPM e 4 GB de memória DDR3 1066Mhz. O meio de acesso entre os dois foi uma rede local cabeada, por meio de cabos par trançados, um *switch* com largura da banda de 100 Mb/s.

3.2. Ambiente de teste

Para uso do *SIEGE* foram utilizados três padrões de teste com tempo 120 segundos cada e usando 100, 500 e 1000 conexões concorrentes. Foram utilizados os logs gerados ao final da execução para analisar como cada uma das tecnologias comportou-se em alta concorrência de acesso ao recurso. Os dados utilizados do log ao final do teste são 1) número de transações, 2) média do tempo(ms), 3) obtenção de sucesso, 4) maior tempo de transação(ms), para cada uma das abordagens.

Para os testes utilizando o AB foram criados três padrões de carga de trabalho. Foram utilizados como parâmetro o número de requisições para 10.000, 50.000 e 100.000 requisições todas com 200 conexões simultâneas. Para avaliar a melhor tecnologia foram comparados os seguintes dados obtidos do *log* de resposta do AB: 01) quantidade de transações, 2) número de transações por segundo e 3) transferência de dados em mb 4)maior tempo de resposta em cada tenologia(ms).

Na Tabela 1 são apresentados os comandos utilizados para cada um dos experimentos realizados. Os comandos apresentados são referentes a linha de comando do Sistema Operacional *linux*. Para os comandos do *SIEGE*, o parâmetro -c refere-se ao numero de conexões, o parâmetro -t ao tempo em segundos e o último parâmetro refere-se ao endereço de rede. Na configuração do comando do AB o parâmetro -n é quantidade de requisições, o parâmetro -c o número de clientes concorrentes e o último parâmetro refere-se ao endereço de rede.

Tabela 1. *Siege* descrição dos comandos

Ferramenta	**Clientes Concorrentes**	**Tempo de execução**	**URL**
siege	-c100	-t120s	http://.../json
siege	-c500	-t120s	http://.../json
siege	-c1000	-t120s	http://.../json
Ferramenta	**Número de Requisições**	**Tempo de execução**	**URL**
ab	-n 10000	-c 200	http://.../json
ab	-n 50000	-c 200	http://.../json
ab	-n 100000	-c 200	http://.../json

3.3. Configuração das Tecnologias

As configurações de cada uma das tecnologias foram realizadas da forma mais simples possível seguindo o básico indicado na documentação de cada uma delas. No caso do *C++, PHP, RUBY, PYTHON e NODEJS* o uso de *micro-framework* se fez necessário para deixar mais simples a suas implementações, sendo eles *CROW, SLIM, SINATRA, FLASK e RESTIFY* respectivamente. Quanto ao *JAVA e o GO*, a própria tecnologia deu suporte à criação de *API REST* do tipo *JSON* de forma nativa. No *JAVA* em especial fez-se necessário o uso de uma biblioteca extra o *GSON* (criada pela equipe do *Google* sendo esta uma biblioteca livre, focada em manipulação de *JSON*) para auxiliar na resposta padrão, deixando assim todas implementações com a *string* mesma resposta.

Um caso particular fora observado quanto ao caso de uso do *C++, NODEJS e GO*, nestas linguagens não se fez necessário o uso de um servidor extra para as aplicações, a

própria tecnologia provia esse serviço de forma nativa com a criação de um processo no sistema operacional. Em contra partida *JAVA, PHP, RUBY e PYTHON* necessitaram da instalação de um servidor para cada, sendo eles *TOMCAT, APACHE, THIN e GEVENT* respectivamente.

A Quadro 1 apresenta o código desenvolvido pra implementar a rota *HTTP* com verbo *GET* para o *micro-framework CROW* do C++. Para implementação de outras tecnologias foi utilizado a mesma ideia principal da implementação do *CROW*.

Quadro 1. Exemplo de C++ com CROW

```
#include "crow.h"
#include "json.h"//api manipulação
//de json do proprio crow
int main(){
//criando objeto app
crow::SimpleApp app;
//esse objeto gerencia as requisições
//macroque gerencia as rotas
//http no no crow
CROW_ROUTE(app, "/json")
   .methods("GET"_method)([]{ //listando verbos
      crow::json::wvalue x;
      x["Ola"] = "Mundo!";
      return x;
   });
   //definição da porta
   app.port(3000)
   .multithreaded()//deixando paralelizado
   .run();//rodando o server
}
```

4. Resultados

4.1. SIEGE

O resultado dos testes utilizando a ferramenta *SIEGE* são apresentados na Tabela 2 e Figura 1, onde é apresentado o comportamento das tecnologias em um cenário de 100 clientes concorrentes. Não ouve um diferença considerável entre as tecnologias analisadas, e pode-se concluir com esse cenário que com um baixo número de Clientes uma *API REST* tem comportamento semelhante independentemente da tecnologia utilizada.

A Tabela 3 apresenta os resultados do experimento com o número de requisições concorrente em uma quantidade 5 vezes maior. Neste caso houve uma diferença considerável entre o primeiro e último colocado no experimento, cerca 21.207 requisições. As linguagem compiladas, JAVA, C++ e GO, se mostraram com um maior poder de processamento paralelo e capacidade de resposta.

Vale observar que o C++ e GO tem seu maior tempo de resposta muito baixo, cerca 0,07 - 0,84 segundos respectivamente, mostrando que para auto concorrência possui um poder de processamento maior que *PHP, PYTHON e RUBY*. Estas apresentam um

Tabela 2. *Siege* Teste 100 requisições concorrentes

Tecnologia	Transações(qtd)	Transações (s)	Transferência	Maior Tempo
JAVA	23.691	197,56	0,38 mb	0,06 s
C++	23.800	198,52	0,36 mb	0.02 s
GO	23.630	198,50	0,36 mb	0.01 s
RUBY	23.674	197,58	0,36 mb	0.05 s
NODEJS	24.123	201,63	0,37 mb	0.05 s
PHYTON	23.802	199,05	0.48 mb	0.04 s
PHP	24.095	201,63	0,37 mb	0.07 s

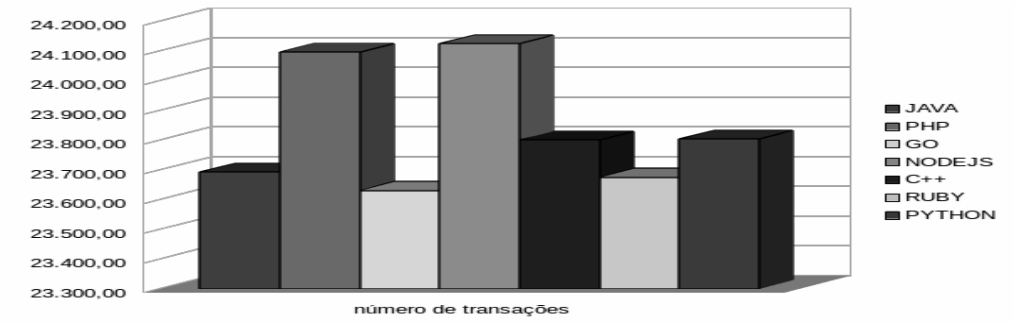

Figura 1. Gráfico de requisições com 100 clientes

resultado na casa dos três segundos, ou seja, aproximadamente cinquenta vezes mais lentas que C++, GO e três vezes perante o *JAVA e NODEJS*. Observa-se que o *NODEJS* é uma linguagem interpretada que usa *JIT (just in time)* compilado em tempo de execução através do V8 (motor *javascript do google chrome*).

Observando os resultados obtidos no terceiro teste, verificados na Tabela 4 e Figura 2, pode-se destacar a performance das tecnologias compiladas, como o C++. Ao verificar a diferença no número de transações do C++ e do PHP, observa-se uma diferença grande entre as tecnologias, aproximadamente 119.649 transações. Assim, verificando o quão poderoso é o *CROW*, apesar de não possuir um servidor para suporte tem boa qualidade. Quando observado o maior tempo de resposta entre as duas tecnologias, a diferença entre C++ (0.16s) e PHP (63.72s) é considerável, com uma diferença de 370.000 vezes.

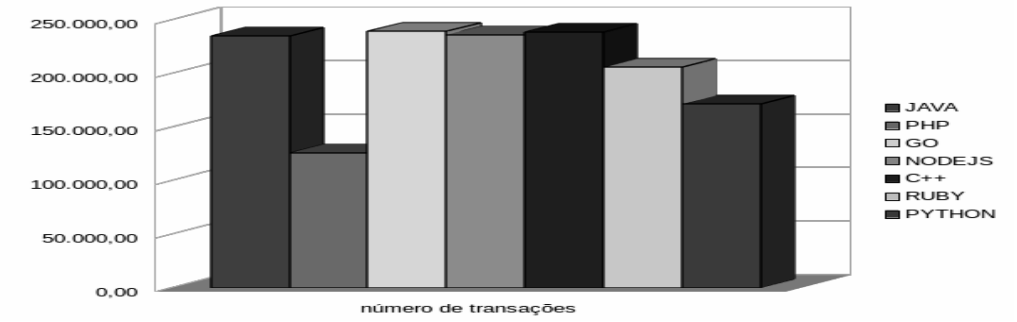

Figura 2. Gráfico de requisições com 1000 clientes

Tabela 3. *Siege* Teste 500 requisições concorrentes

Tecnologia	Transações(qtd)	Transações (s)	Transferência	Maior Tempo
JAVA	118.536	987,88	1,92 mb	1,01 s
C++	119,264	994,03	1,82 mb	0,07 s
GO	120.037	1.000,39	1,83 mb	0,84 s
NODEJS	119.356	994,80	1,82 mb	3,01 s
RUBY	117.717	981,14	1,80 mb	3,10 s
PHYTON	118.516	987,80	2,37 mb	3,02 s
PHP	90.646	755,51	1,38 mb	3,17 s

Tabela 4. *Siege* Teste 1000 requisições concorrentes

Tecnologia	Transações(qtd)	Transações (s)	Transferência	Maior Tempo
C++	238.416	1.986,97	3,64 mb	0,16 s
GO	239.032	1.992,27	3,65 mb	1,03 s
NODEJS	235.959	1.966,65	3,60 mb	3,05 s
JAVA	235.042	1.959,17	3,81 mb	1,06 s
RUBY	205.649	1.713,88	3,14 mb	7,08 s
PHYTON	171.259	1.427,40	3,43 mb	7,13 s
PHP	125.847	1.048,90	1,92 mb	63,72 s

4.2. *APACHE BENCHMARK*

O AB foi usado para verificar o tempo gasto pelo Servidor para responder 10.000, 50.000 e 100.000 requisições realizadas pela máquina Cliente. A Tabela 5, 6, 7 corresponde a esses resultados, respectivamente.

Anteriormente, tecnologias compiladas se mostraram com um comportamento mais performático quando submetidas a teste de exaustão e estresse feitos pelo *SIEGE*. Nesse experimento os resultados foram equivalentes. Eles apresentados nas Tabelas 5, 6, 7 foram obtidos com 200 Clientes concorrentes.

Tabela 5. *AB* 10.000 requisições

Tecnologia	Tempo(ms)	RPS	Desvio Padrão(s)	Maior(ms)	Média(ms)
JAVA	707	14.141,59	1,80	48	5
GO	935	10.695,98	3,10	21	9
C++	1.407	7.105,30	9,10	34	14
NODEJS	3.548	2.818,62	4,80	81	35
RUBY	5.526	1.809,73	9,90	89	54
PHYTON	6.693	1.494,09	4,80	100	66
PHP	13.311	751,28	19,50	297	126

A Tabela 5 possui valores muito distantes das tecnologias com melhor desempenho para com a pior, chegando a uma diferença impactante. O *GO* no caso em questão,

não gastou nem 1 segundo para fazer 10.000 requisições, algo realmente considerável, já o *PHP* demorou cerca de 9 segundos,(quase quinze vezes mais lenta) deixando claro o poder das linguagem compiladas perante as interpretadas.

A Tabela 5 observa-se uma diferença notória na capacidade de processamento entre as tecnologias analisadas. Nela, o *JAVA* e *GO* precisaram de 1 segundo para processar 10.000 requisições. Já o *PHP* usou cerca de 13 segundos. Visivelmente as linguagens compiladas se comportaram melhor nos ambientes dos experimento perante as linguagens interpretadas.

Tabela 6. *AB* 50.000 requisições

Tecnologia	Tempo(ms)	RPS	Desvio Padrão(s)	Maior(ms)	Média(ms)
JAVA	3.692	13.542,32	6,5	156	5
GO	4.798	10.420,23	4,3	26	9
C++	6.895	7.252,02	8,3	39	13
NODEJS	19.757	2.545,52	8,3	116	39
RUBY	27.781	1.799,81	10,6	100	54
PHYTON	33.380	1.497,92	4,5	86	66
PHP	71.512	699,18	6,3	144	142

Conforme a Tabela 7 e Figura 3, no qual o *PHP, RUBY e PYTHON* quando impostos a 100.000 requisições, verificou-se que o comportamento foi abaixo das outras tecnologias que são compiladas. Esse é um dos motivo pelo qual grandes empresas, como *Twiter e Facebook* migraram para tecnologias compiladas em suas *API*, sendo elas o *SCALA e HHVM*.

Tabela 7. *AB* 100.000 requisições

Tecnologia	Tempo(ms)	RPS	Desvio Padrão(s)	Maior(ms)	Média(ms)
JAVA	7.582	13.188,77	7,4	177	5
GO	7.973	12.541.67	3,7	26	7
C++	13.919	7.184.56	8,7	44	13
NODEJS	33.715	2.966,07	6,4	120	51
RUBY	55.584	1.799,08	10,6	104	54
PHYTON	67.223	1.487,58	5,0	92	66
PHP	96.091	1.040,68	1,7	97	96

5. Conclusão

Este trabalho teve como objetivo o teste da performance das principais tecnologias usadas para criação de *API REST JSON*. Fica verificado a melhor performance das linguagens compiladas se comparadas as linguagens interpretadas.

Foram realizados seis testes com diferentes configurações nas sete principais tecnologias e utilizando duas ferramentas distintas, SIEGE e AB. Os testes foram realizados seguindo uma metodologia rígida, utilizando o mesmo ambiente de teste para todas

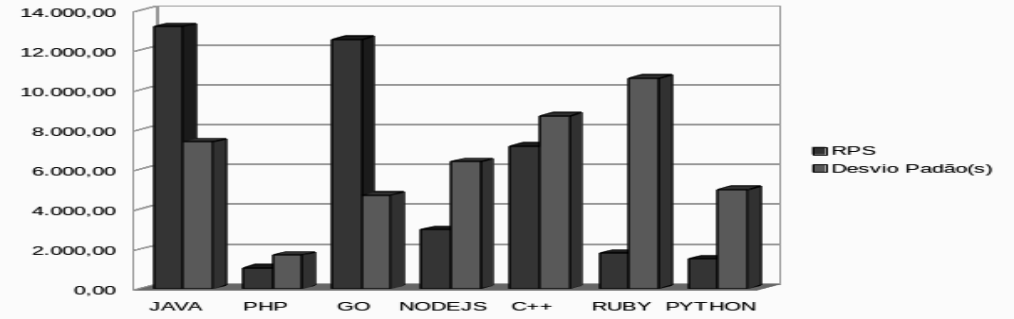

Figura 3. Gráfico de RPS e DS com 1000 clientes

tecnologias. Por meio dos testes foi possível verificar algumas particularidades de cada tecnologia, tais como estabilidade das linguagem compiladas porém com uma maior complexidade acerca da implementação.

Como ameaça a validade do trabalho pode-se destacar a utilização de uma rede local para realização dos testes. Como trabalhos futuros será utilizado uma aplicação real para melhor avaliação das tecnologias aqui testadas.

Conclui-se que para em sistemas com baixa quantidade de clientes, não faz-se necessário o uso de linguagens complexas. É necessário sempre uma boa análise do ambiente onde a *API* será utilizada para uma boa escolha da linguagem a ser utilizada.

Referências

Saudate, Alexandre(2013). *REST*. Construa API's inteligentes de maneira simples, 2013 , Casa do Codigo, 2013.

Silveira, Guilherme e Amaral, Mario (2014). *JAVA SE 7 Programmer 1*. O guia para certificacao *Oracle Certifiqued Associate* 2014, Casa do Código, 2014.

Rozlog, Mike (2013). *Rest e Soap* Usar um dos dois ou ambos?, 2013. Disponível em: <http://www.infoq.com/br/articles/rest-soap-when-to-use-each> Acesso em: 23 fevereiro de 2014.

E. Rescorla (2000) *HTTP over tls*. Disponível em: <http://tools.ietf.org/html/rfc2818> Acesso em: 16 de junho 2015

Fulmer, Jeff (2012). *About Siege 2012*. Disponível em: <https://www.joedog.org/siege-home/> Acesso em: 22 de junho 2015

Apache Foundation (2015). *Apache HTTP server benchmarking tool*. Disponível em: <http://httpd.apache.org/docs/current/programs/ab.html> Acesso em: 22 de junho 2015

Alexandre Luis Correa, Antonio Juarez Alencar , Eber Assis Schmitz (2012). *Uma Abordagem Baseada em Especificações para Testes de Web Services RESTful*. In: SBSI, VIII Simpósio Brasileiro de Sistema de Informações, Anais, 2012, 372-383.

Nogueira, André da Silva (2015). *Profiling de aplicações Web : Estudo comparativo entre aplicações Java Web e aplicações RoR* . Universidade de Minho , Escola de Engenharia, Departamento de Informática, Dissertações.