Análise Comparativa Entre Linguagens de Back End

Angelito M. Goulart[1], Ricardo N. Rodrigues[1]

[1]Instituto de Informática – Universidade Federal de Rio Grande (FURG)
Av. Itália, s/n - Km 8 - Carreiros, Rio Grande - RS - Brazil

{angelitomgoulart,ricardonagel}@gmail.com

Abstract. *This article aims to observe and compare the main languages for backend-focused systems development. The study compares Java, Node.js, and PHP technologies by measuring the average response time, deployment complexity, and resource consumption of each technology using a sample application. The sample application reads and writes to a database. SQLite data and computes the Fibonacci sequence. The application was implemented in each of the technologies to perform the tests.*

Resumo. *Este artigo tem objetivo observar e comparar as principais linguagens para o desenvolvimento de sistemas com foco em backend. O estudo compara as tecnologias Java, Node.js e PHP medindo o tempo médio de resposta, a complexidade de implementação e o consumo de recursos de cada tecnologia utilizando uma aplicação de exemplo. A aplicação de exemplo realiza leitura e escrita em um banco de dados SQLite e calcula a sequência de Fibonacci. A aplicação foi implementada em cada uma das tecnologias para a realização dos testes.*

1. Introdução

O crescimento da Internet, impulsionado pela popularização dos computadores pessoais, pelos serviços oferecidos na Web e pelas várias formas e locais de acesso, tem impactado no dia-a-dia das pessoas na realização de trabalhos, estudos, negócios e entretenimento [Maia 2010].

Uma enorme indústria de software tornou-se fator dominante nas economias do mundo industrializado. Equipes de especialistas em software, cada qual concentrando-se numa parte da tecnologia necessária para distribuir uma aplicação complexa, substituíram o programador solitário de antigamente [Pressman 2011].

Nos dias atuais, estão disponíveis muitos frameworks e tecnologias eficientes e eficazes para a implementação de aplicações web. No entanto, é crucialmente importante comparar e contrastar as tecnologias adequadas para o tipo de aplicativo web que está sendo desenvolvido [Mishra 2014].

O desenvolvimento back end refere-se ao lado do servidor do desenvolvimento de uma aplicação, onde o foco é, principalmente em como uma aplicação web funciona. Este tipo de desenvolvimento usualmente consiste em três partes: um servidor, uma aplicação e um banco de dados. Códigos escritos por desenvolvedores back end são responsáveis por realizar a comunicação entre a base de dados e o navegador [Stewart 2019].

Baseado neste contexto, surge a necessidade de comparar as principais tecnologias para o desenvolvimento back end de aplicações web. O presente trabalho tem como principal objetivo avaliar três principais tecnologias para o desenvolvimento web: Java, PHP e Node.js.

Nas próximas seções são apresentadas a fundamentação teórica, os trabalhos relacionados, uma breve explicação de cada tecnologia e o ambiente utilizado em cada um dos testes. Ao fim do trabalho será possível identificar dados como tempo de resposta de cada tecnologia, consumo de recursos do servidor e complexidade de implementação da aplicação de exemplo utilizada nos testes em cada uma das tecnologias.

2. Fundamentação Teórica

Este capítulo tem como objetivo apresentar os estudos em que o presente artigo se baseia. A fundamentação teórica deste trabalho é baseada em três principais tópicos: trabalhos relacionados, backend e frontend e por fim as tecnologias analisadas no trabalho em questão.

2.1. Trabalhos Relacionados

Segundo Chhetri (2016), com o Node.js, podem ser criados aplicativos complexos de tempo real que podem ser escalados para milhões de conexões de clientes. Em seu estudo, é realizada uma introdução ao Node.js e suas principais vantagens. Um comparativo entre Node.js e PHP também é apresentado, utilizando a sequência de Fibonacci como exemplo.

No comparativo de Chhetri (2016), Node.js foi um pouco mais rápido para atender as requisições, porém utilizou mais processamento, já a linguagem PHP teve uma taxa de resposta um pouco mais alta, porém utilizou menos recursos computacionais.

O estudo de Mishra (2014) compara a linguagem PHP com o framework ASP.NET. Seu estudo conclui que o ASP.NET dá ao desenvolvedor maiores possibilidades e velocidade no desenvolvimento. É algo bastante controverso, já que comparar um framework (ASP.NET) com uma linguagem pura (PHP sem framework) é um tanto quanto desleal. Para ser mais preciso, seu estudo deveria ter utilizado um framework PHP no comparativo e não apenas a linguagem pura.

No estudo de Sathyaseelan e Cordova (2016), é realizada uma análise somente entre entre o ecossistema da linguagem Java, onde as tecnologias Struts e JSF são apresentadas e comparadas. Já o estudo de Kronis e Uhanova (2018), comparou ASP.NET e Java, onde em uma média geral, as duas tecnologias obtiveram resultados similares.

O estudo de Suzumura e Tozawa (2008) descreve a comparação de webservices implementados em PHP com os implementados em Java e C. Dada a arquitetura e a linguagem de programação diferentes, os resultados experimentais mostram que o PHP tem desempenho razoavelmente alto em comparação às implementações baseadas em Java e C, ao mesmo tempo em que oferece aos usuários uma alta produtividade.

2.2. Backend vs Frontend

O front-end é todo o código da aplicação responsável pela apresentação do software (client-side). Em se tratando de aplicações web, é exatamente o código do sistema que roda no navegador. Um desenvolvedor front-end, geralmente, trabalha com linguagens como HTML, CSS e JavaScript, além de frameworks e bibliotecas, como por exemplo Angular, React, Vue.js, etc [Andrade 2018].

O back-end é a parte do software que roda no servidor, por isso também é conhecida como server-side. É o back-end que fornece e garante todas as regras de negócio, acesso a banco de dados, segurança e escalabilidade. Embora o front-end também possa ter algumas regras e validações, é o back-end que deve garantir a integridade dos dados [Andrade 2018].

2.3. Java

Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Em 2008 o Java foi adquirido pela empresa Oracle Corporation. Diferente das linguagens de programação modernas, que são compiladas para código nativo, a linguagem Java é compilada para um bytecode que é interpretado por uma máquina virtual (Java Virtual Machine, mais conhecida pela sua abreviação JVM) [JAVA (LINGUAGEM DE PROGRAMAÇÃO) 2019].

Segundo o ranking Tiobe [Tiobe Index 2019], que mede a popularidade das principais linguagens de programação, Java está em primeiro lugar, sendo considerada a linguagem de programação mais popular mundialmente.

2.4. Node.js

Node.js é um interpretador, com código aberto, de código JavaScript de modo assíncrono e orientado a eventos, focado em migrar a programação do Javascript do lado do cliente para os servidores, criando assim aplicações de alta escalabilidade (como um servidor web), capazes de manipular milhares de conexões/requisições simultâneas em tempo real, numa única máquina física [NODE.JS 2019].

O Node.js é baseado no interpretador V8 JavaScript Engine (interpretador de JavaScript open source implementado pelo Google em C++ e utilizado pelo Chrome). Foi criado por Ryan Dahl em 2009, e seu desenvolvimento é mantido pela fundação Node.js em parceria com a Linux Foundation [NODE.JS 2019].

Segundo uma pesquisa realizada por um popular fórum de desenvolvedores, pelo sétimo ano consecutivo, o JavaScript é a linguagem de programação mais usada [Stack Overflow 2019].

2.5. PHP

PHP (um acrônimo recursivo para "PHP: Hypertext Preprocessor", originalmente Personal Home Page) é uma linguagem interpretada livre, usada originalmente apenas

para o desenvolvimento de aplicações presentes e atuantes no lado do servidor, capazes de gerar conteúdo dinâmico na World Wide Web. Figura entre as primeiras linguagens passíveis de inserção em documentos HTML, dispensando em muitos casos o uso de arquivos externos para eventuais processamentos de dados [PHP 2019].

Em questão de popularidade, o WordPress, um dos principais softwares desenvolvidos utilizando a linguagem PHP, alimenta mais de 34% da web - um número que aumenta todos os dias [WordPress 2019].

3. Metodologia de Pesquisa

Existem muitas razões para o teste de carga em aplicações Web. O tipo mais básico de teste de carga é usado para determinar o comportamento da aplicação Web através de condições normais e altos picos de carga. À medida que se começa o teste de carga, é recomendável se começar com um pequeno número de usuários virtuais(Virtual Users) e então incrementar gradualmente a carga do normal até o pico [Campos 2019].

No presente estudo, testes de carga foram aplicados na aplicação desenvolvida nas três tecnologias apresentadas. Os testes foram realizados de forma individual e em uma mesma janela de tempo, para evitar que fatores externos, como sobrecarga da rede em certos horários, comprometesse os resultados.

3.1. Aplicação de Exemplo

A aplicação de exemplo consiste em um método executado através de uma chamada HTTP do tipo GET, sem a passagem de parâmetros. A funcionalidade básica da aplicação é realizar a consulta a um número fixo no banco de dados (30), onde, baseado neste número, será executado um algoritmo para cálculo da sequência de Fibonacci de forma recursiva. Após a conclusão, um registro será atualizado no banco de dados.

Assim como o estudo de Chhetri (2016), o cálculo da sequência de Fibonacci foi escolhido por envolver alta carga computacional, fornecendo um meio de comparar as tecnologias sob o aspecto de uso extremo de processamento. O acesso a banco de dados foi escolhido pelo fato de aplicações backend serem responsáveis por, como citado por Andrade (2018), segurança, escalabilidade e acesso a banco de dados.

A aplicação foi desenvolvida utilizando o máximo de componentes padrão da linguagem, evitando bibliotecas e frameworks de terceiros, já que o objetivo do estudo foi analisar a tecnologia pura, sem qualquer código extra ou desnecessário. O código fonte das aplicações desenvolvidas pode ser encontrado nos anexos I, II e III. O banco de dados escolhido para uso nas 3 aplicações foi SQLite devido a sua simplicidade e por estar embutido na maioria das tecnologias.

3.2. Sequência de Fibonacci

Na matemática, a Sucessão de Fibonacci (também Sequência de Fibonacci), é uma sequência de números inteiros, começando normalmente por 0 e 1, na qual, cada termo subsequente corresponde à soma dos dois anteriores. A sequência recebeu o nome do matemático italiano Leonardo de Pisa, mais conhecido por Fibonacci, que descreveu, no ano de 1202, o crescimento de uma população de coelhos, a partir desta. Esta sequência já era, no entanto, conhecida na antiguidade [SEQUÊNCIA DE FIBONACCI, 2019].

Segundo Chhetri (2016) o cálculo dos números de Fibonacci é um problema de programação comum no ensino de ciências da computação e matemática.

A principal desvantagem da função Fibonacci recursiva, no entanto, é sua complexidade de tempo exponencial (devido ao número crescente de valores intermediários) em comparação com a complexidade de tempo linear das versões iterativas. O cálculo de Fibonacci foi escolhido por envolver cálculos computacionais pesados, sendo uma forma de testar como as tecnologias se comportam com cargas de cálculo computacional.

3.3. Configuração dos Servidores

Para os testes, foram criadas três máquinas virtuais individuais utilizando o serviço de computação em nuvem da Digital Ocean. As três máquinas foram criadas no mesmo datacenter (São Francisco, CA) e possuem configurações de hardware iguais, Nas tabelas 1, 2, 3 e 4 é possível visualizar a configuração das máquinas utilizadas nos testes.

Tabela 1. Configuração da máquina responsável por aplicar os testes.

Modelo:	Macbook Pro 13” (meados 2012)
Processador:	Intel Core i5 2.5GHz
Memória RAM:	4GB
Armazenamento:	128GB SSD
Software:	macOS Mojave 10.14.6

Tabela 2. Configuração da máquina utilizada nos testes com Java.

Processador:	2 CPUs
Memória RAM:	2GB
Armazenamento:	60GB SSD
Software:	Ubuntu Server 18.04.3 LTS OpenJDK 11.0.4 Apache Tomcat/9.0.16 (Ubuntu)

Tabela 3. Configuração da máquina utilizada nos testes com Node.js.

Processador:	2 CPUs
Memória RAM:	2GB
Armazenamento:	60GB SSD
Software:	Ubuntu Server 18.04.3 LTS Node.js v8.10.0

Tabela 4. Configuração da máquina utilizada nos testes com PHP.

Processador:	2 CPUs
Memória RAM:	2GB
Armazenamento:	60GB SSD
Software:	Ubuntu Server 18.04.3 LTS Apache/2.4.34 (Unix) PHP 7.1.23

3.4. Banco de Dados

Como banco de dados da aplicação, foi utilizado o SQLite, por sua simplicidade, leveza e por todas as tecnologias utilizadas o suportarem. Foi criada uma base de dados com apenas uma tabela, apenas para ilustrar os acessos de leitura e escrita a mesma. A descrição dos campos da tabela pode ser visualizada na tabela 5.

Tabela 5. Esquema da tabela única utilizada na base de dados de teste.

Nome do Campo	Descrição
id	Campo sequencial com a identificação única de cada registro.
number	Campo fixo, onde todos os registros possuem o mesmo valor (30). Este campo é utilizado como base pela aplicação para a execução da sequência de Fibonacci.
date	Campo de data genérico. Criado apenas para testes de escrita na base.

A mesma estrutura de tabelas e registros foi utilizada nos três testes. O banco possuía 10000 registros inseridos inicialmente e todas as aplicações buscaram o registro com ID 7510, escolhido de forma aleatória. Após a leitura do campo number da tabela e

execução da sequência de Fibonacci, a aplicação grava a data atual no registro consultado.

3.5. Aplicação dos Testes

A aplicação dos testes foi realizada com o uso do software Apache JMeter versão 5.1.1. Foram definidas 3 baterias de testes com diferentes números de usuários. O número de usuários foi definido após testes preliminares em cada aplicação.

A bateria 1 foi realizada com 5 usuários virtuais simultâneos, sendo que 5 novos usuários foram adicionados a cada 2 segundos por um período de 30 segundos, totalizando 75 requisições (15 ciclos x 5 usuários por ciclo).

A bateria 2 foi realizada com 10 usuários virtuais simultâneos, sendo que 10 novos usuários foram adicionados a cada 2 segundos por um período de 30 segundos, totalizando 150 requisições (15 ciclos x 10 usuários por ciclo).

A bateria 3 foi realizada com 30 usuários virtuais simultâneos, sendo que 30 novos usuários foram adicionados a cada 2 segundos por um período de 30 segundos, totalizando 450 requisições (15 ciclos x 30 usuários por ciclo).

3.6. Avaliação dos Resultados

A avaliação dos resultados das requisições foi feita utilizando as ferramentas "Ver Árvore de Resultados", "Relatório de Sumário", "Response Time Graph", "Ver Resultados em Tabela", "Gráfico de Resultados" do software Apache JMeter. Para a avaliação do nível de carga e consumo de recursos do servidor foi utilizada a ferramenta de linha de comando htop.

O comparativo da complexidade de implementação é bastante relativo, já que existem diversas formas de desenvolver uma mesma funcionalidade. O número de linhas de código foi uma base utilizada, porém outros fatores como complexidade de implantação e particularidades de cada tecnologia também foram analisados.

4. Apresentação dos Resultados

Esta seção traz os resultados obtidos na aplicação dos testes nas três aplicações desenvolvidas. Os testes foram realizados a partir da cidade de Santo Antônio da Patrulha, RS - Brasil e os servidores estavam localizados na cidade de São Francisco, Califórnia - Estados Unidos. Para evitar tráfego de rede acentuado, os testes foram realizados durante a noite do dia 16/10/2019 (domingo), entre 20 e 23 horas (GMT -3:00).

4.1. Complexidade de Implementação

A complexidade de implementação é algo bastante relativo de se analisar, já que na computação muitas vezes existem diversas formas de chegar a um mesmo resultado. Uma das métricas utilizadas neste estudo foi o número de linhas de código (sem contar comentários e linhas em branco) e algumas peculiaridades de cada tecnologia.

A aplicação desenvolvida em Java foi escrita em 38 linhas de código, a aplicação Node.js foi escrita com 23 linhas de código e por fim, a aplicação PHP foi a que precisou de menos linhas de código, com um total de apenas 18 linhas.

Analisando outras questões como particularidade de cada linguagem, notou-se que por Node.js utilizar JavaScript, é preciso ter cuidado com chamadas a funções assíncronas, já que muitas vezes este fator pode atrapalhar o fluxo principal do programa.

A linguagem Java mostrou-se a mais complexa para iniciar o desenvolvimento, sendo necessário algumas configurações extras para o correto deploy da aplicação no servidor Apache Tomcat.

O desenvolvimento da aplicação PHP foi o mais simples, já que foram necessárias menos linhas de código e menos declarações, facilitando a compreensão para iniciantes no mundo do desenvolvimento de software.

4.1. Desempenho

Na bateria 1, todas as aplicações completaram o teste com sucesso e não ocorreram erros durante as requisições. A aplicação Java gerou um tempo de resposta médio de 425 milissegundos. A aplicação Node.js um tempo de resposta médio de 481 milissegundos e a aplicação PHP uma média de 480 milissegundos. A bateria 1 foi executada 3 vezes e uma média dos resultados foi calculada. Na bateria 1 a tecnologia Java mostrou um tempo de resposta menor, e as tecnologias PHP e Node.js tiveram resultados praticamente iguais.

A bateria 2 foi executada 3 vezes e uma média dos resultados foi calculada. A aplicação Java teve um tempo de resposta médio de 428 milissegundos, Node.js um tempo de resposta médio de 478 milissegundos e PHP teve uma média de 485 milissegundos. Novamente Node.js e PHP mostraram um tempo de resposta bastante

próximo e Java teve um tempo de resposta menor. Todos os testes foram concluídos com êxito na bateria 2.

Por fim, a bateria 3 também foi executada 3 vezes e uma média do tempo de resposta foi calculado. Para a bateria 3, apenas os dados entre as tecnologias PHP e Java puderam ser comparados, pois a aplicação Node.js acabou apresentando uma taxa de erro superior a 80% nas requisições. Os erros foram gerados durante o acesso ao banco de dados, quando muitas requisições foram realizadas. A tecnologia Java apresentou um tempo de resposta médio de 424 milissegundos e aplicação PHP um tempo de resposta médio de 694 milissegundos. O gráfico com os resultados das baterias pode ser visto nas figuras 1, 2 e 3.

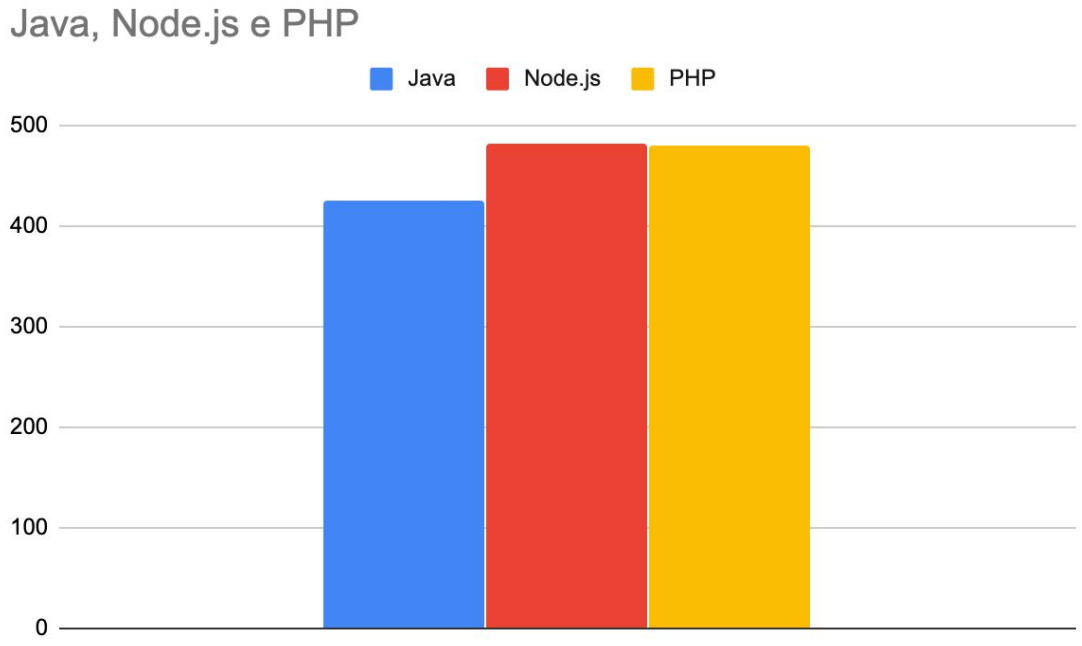

Figura 1. Tempo de resposta das aplicações durante a primeira bateria de testes (75 requisições totais).

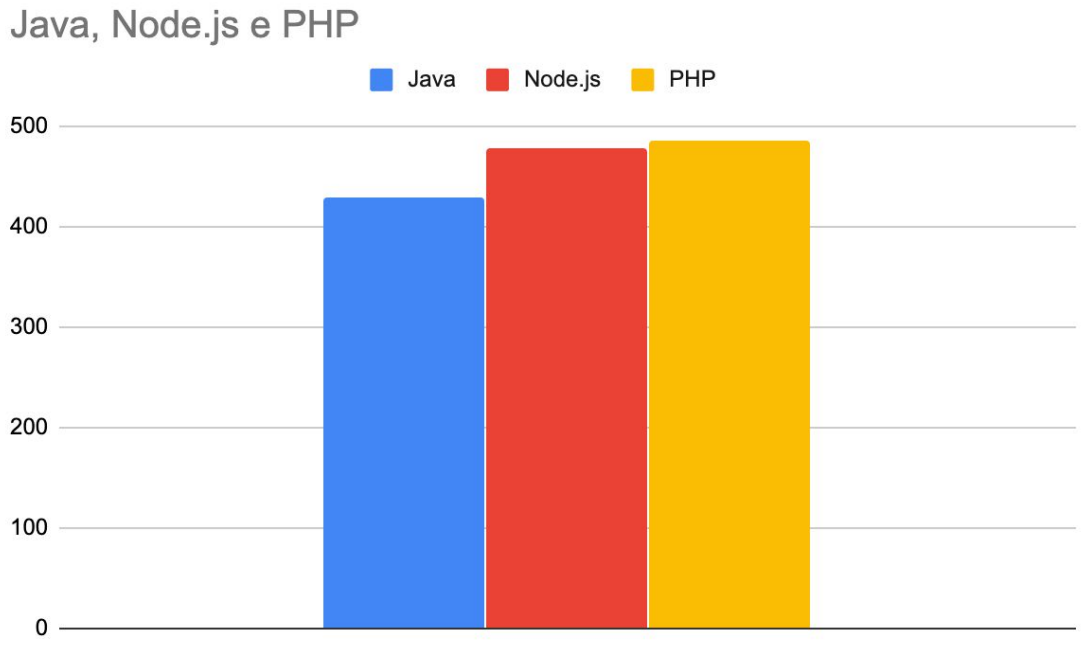

Figura 2. Tempo de resposta das aplicações durante a segunda bateria de testes (150 requisições totais).

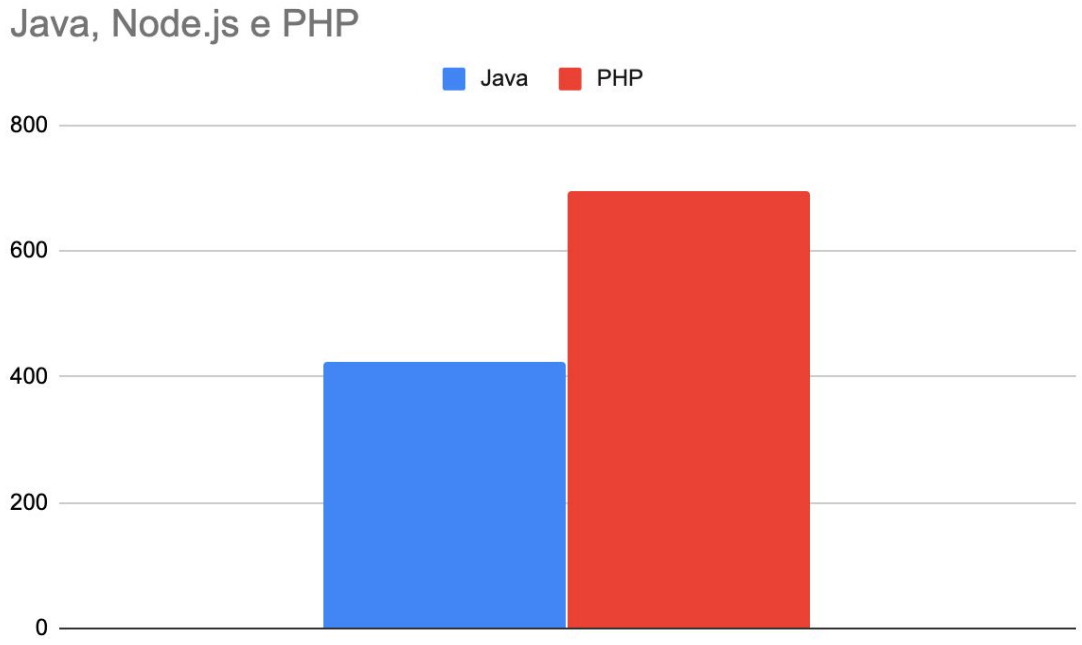

Figura 3. Tempo de resposta das aplicações durante a terceira bateria de testes (450 requisições totais).

4.1. Consumo de Recursos

Sobre o consumo de recursos, o monitoramento foi realizado através da ferramenta htop durante a execução dos testes. Na bateria 1, as aplicações mostraram um consumo de recursos bastante semelhante. Na bateria 2, a aplicação Java continuou com um baixo consumo de CPU, e as aplicações Node.js e PHP mostraram consumo de recursos semelhantes, conforme as figuras 4, 5 e 6.

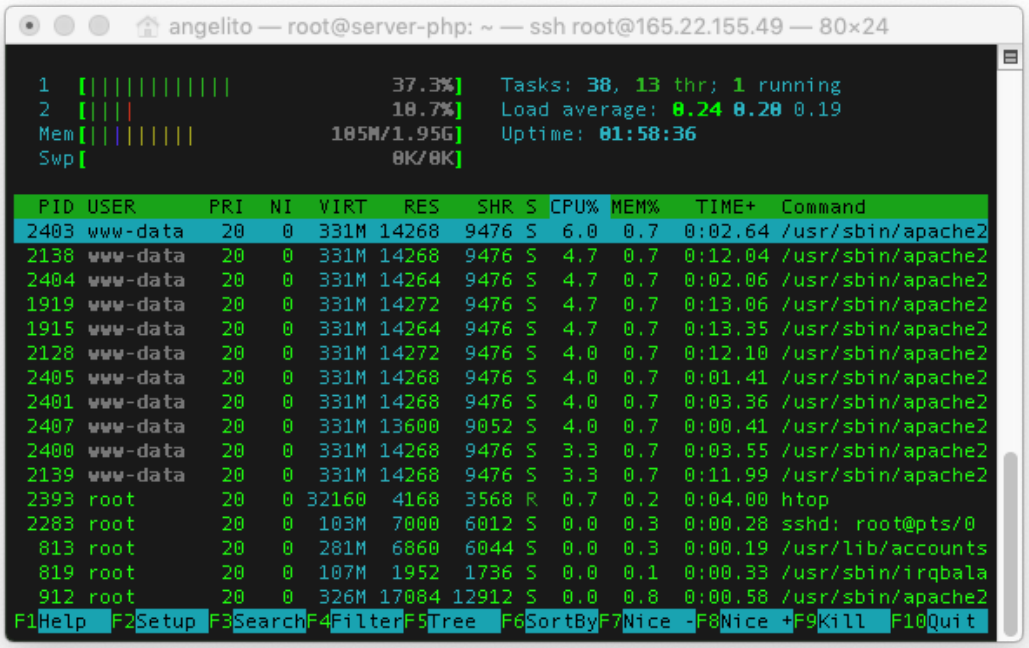

Figura 4. Consumo de recursos da aplicação PHP durante a execução segunda bateria de testes (150 requisições totais). Consumo de memória não atingiu 10% e consumo de processamento não ultrapassou 50%.

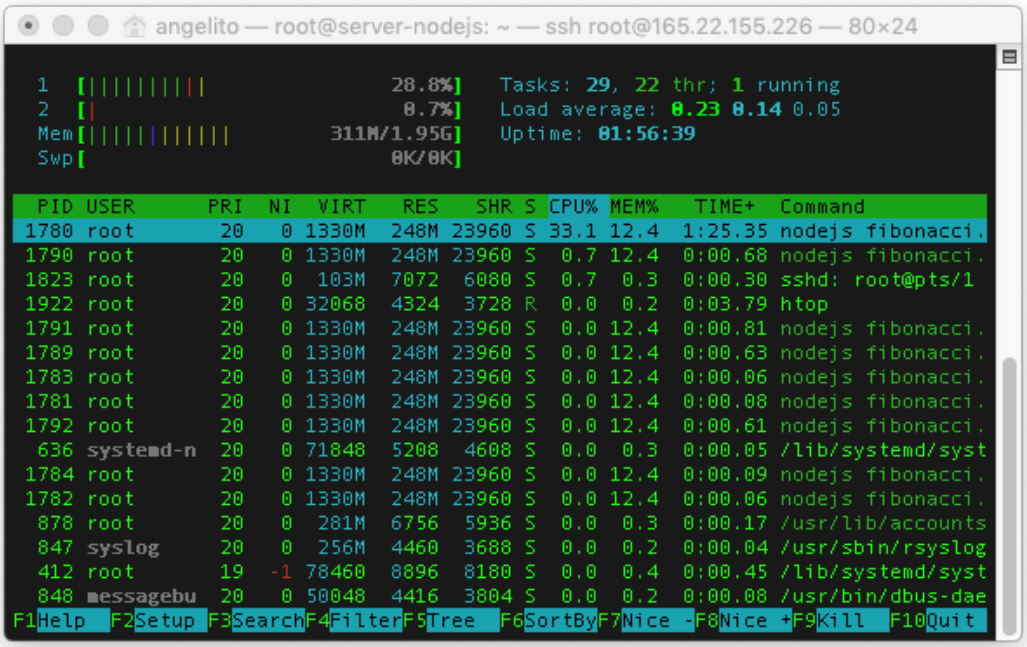

Figura 5. Consumo de recursos da aplicação Node.js durante a execução segunda bateria de testes (150 requisições totais). Consumo de memória não ultrapassou 20% e consumo de processamento não ultrapassou 40%.

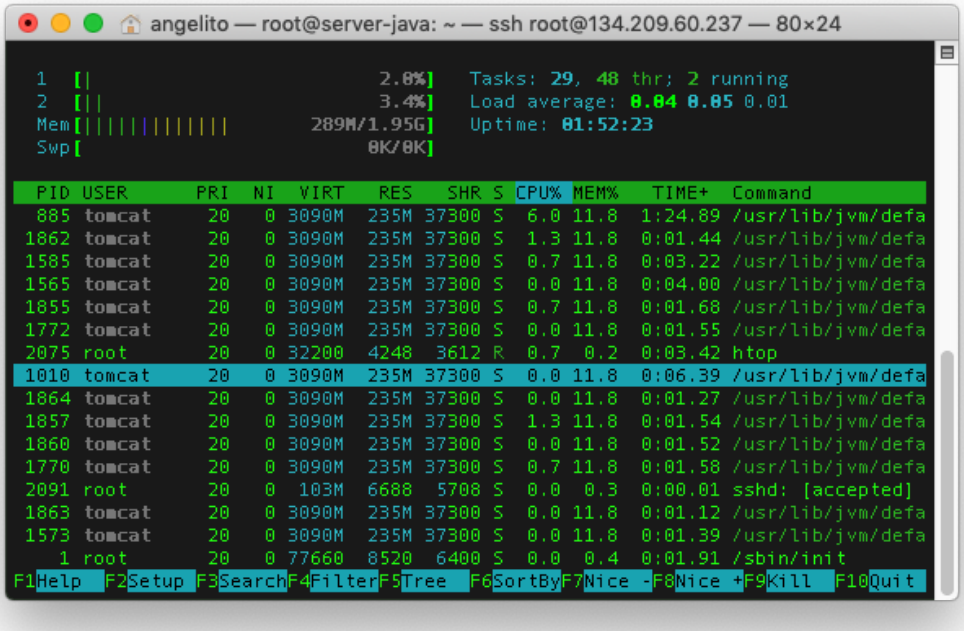

Figura 6. Consumo de recursos da aplicação Java durante a execução da segunda bateria de testes (150 requisições totais). Consumo de memória não ultrapassou 20% e consumo de processamento não ultrapassou 20%.

Para a bateria 3, foram consideradas apenas as aplicações Java e PHP, já que a aplicação Node.js não suportou os testes, devido a problemas ao acesso ao banco de dados. A aplicação Java mostrou um consumo de recursos bastante baixo, bem semelhante ao consumo durante as outras baterias. A aplicação PHP teve um consumo de recursos um pouco mais elevado, porém conseguiu entregar todas as requisições realizadas com sucesso. As figuras 7 e 8 mostram o consumo de recurso das aplicações durante a bateria 3.

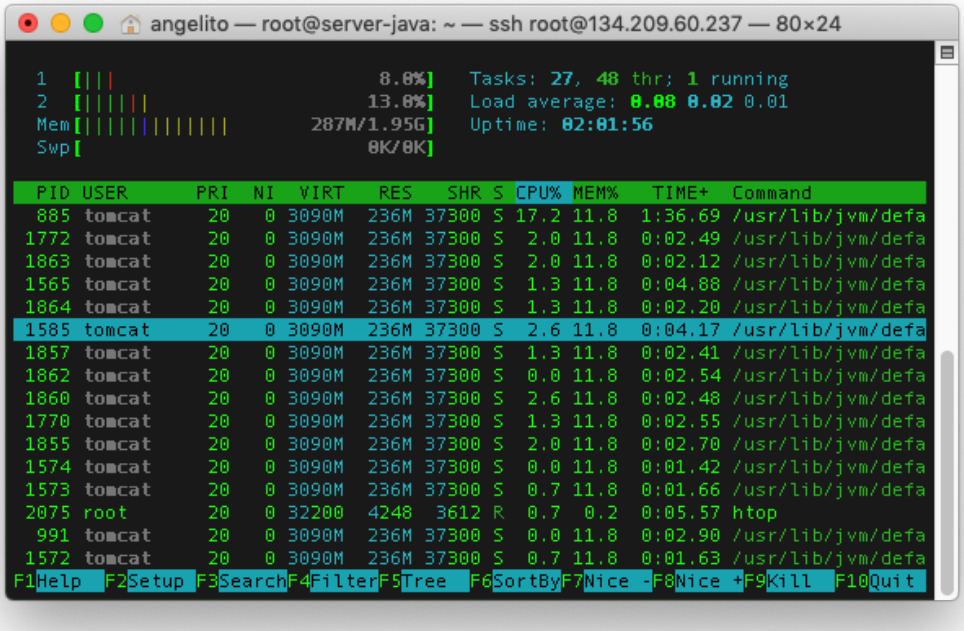

Figura 7. Consumo de recursos da aplicação Java durante a execução da terceira bateria de testes (450 requisições totais). Consumo de memória não ultrapassou 20% e consumo de processamento não ultrapassou 20%.

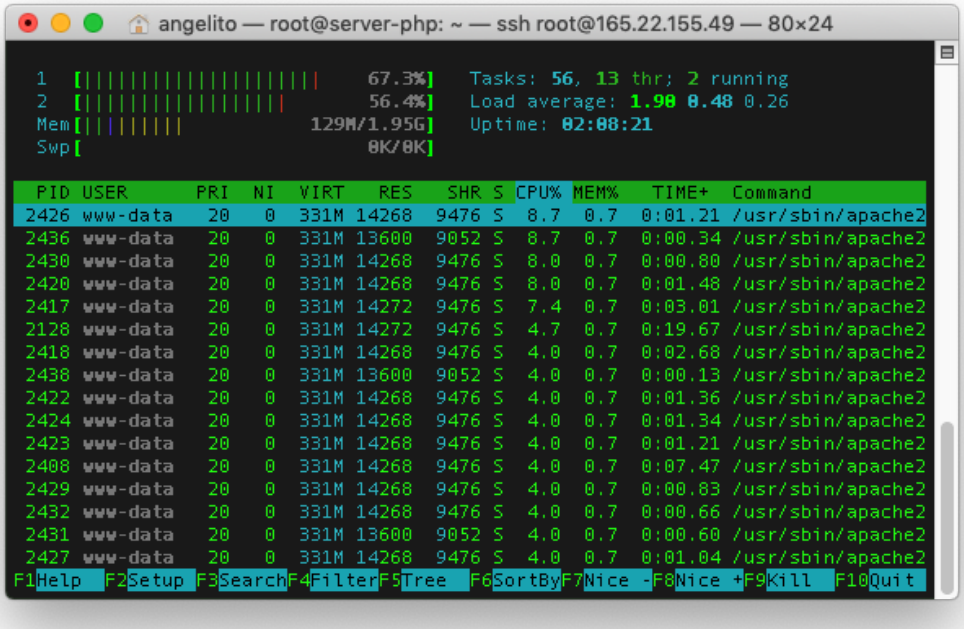

Figura 8. Consumo de recursos da aplicação PHP durante a execução da terceira bateria de testes (450 requisições totais). Consumo de memória não ultrapassou 10% e consumo de processamento ficou em 50% e 70%.

5. Considerações Finais

Analisando o desempenho das aplicações, a tecnologia Java mostrou maior complexidade no desenvolvimento e instalação do ambiente porém mostrou um maior desempenho ao atender diversas requisições consumindo poucos recursos. Isto pode se caracterizar devido ao fato de ser uma linguagem compilada, enquanto as outras tecnologias utilizadas nos testes serem interpretadas.

A tecnologia PHP mostrou uma grande facilidade no desenvolvimento e um desempenho satisfatório ao atender as requisições. A tecnologia Node.js não conseguiu completar a terceira bateria de testes por problemas de concorrência no acesso ao banco de dados. Possivelmente utilizando um outro SGBD esta limitação não ocorreria.

Quanto à validade dos testes, é possível obter resultados diferentes com a customização e otimização de configurações/ajustes no servidor de aplicação de cada tecnologia. Vale lembrar que nenhum ajuste ou tuning foi realizado em qualquer uma das tecnologias. O servidor de aplicação foi instalado de forma default, de acordo com o gerenciador de pacotes do sistema operacional utilizado (Ubuntu Server).

Como trabalhos futuros, poderia ser realizado um comparativo aplicando diversas técnicas de tuning e otimização em cada uma das tecnologias. Um comparativo

com diferentes tipos de aplicações também poderia ser uma ótima escolha para verificar o desempenho das tecnologias sob outros aspectos.

Referências

Andrade, T. (2018). Back-end vs Front-end vs Fullstack: Escolha o seu futuro como programador! Acesso em 21/09/2019.

Campos, F.; Testes de Performance - Testes de Carga, Stress e Virtualização - Parte 3. In: http://www.linhadecodigo.com.br/artigo/3259/testes-de-performance-testes-de-carga-stress-e-virtualizacao-parte-3.aspx. Acesso em 27/09/2019.

Chhetri, N. (2016). A Comparative Analysis of Node.js (Server-Side JavaScript).

Maia, L. (2010). Um Processo Para o Desenvolvimento de Aplicações Web Acessíveis.

Mishra, A. (2014). Critical Comparison Of PHP And ASP.NET For Web Development.

Kronis, K.; Uhanova, M.; (2018). Performance Comparison of Java EE and ASP.NET Core Technologies for Web API Development.

Pressman, R. (2011). Engenharia de Software 7a Edição - Uma Abordagem Profissional.

Sathyaseelan, B.; Cordova, R; (2016), A Comparative Study of Top Web Design Models that are using Java Technologies.

SEQUÊNCIA DE FIBONACCI. In: https://pt.wikipedia.org/w/index.php?title=Sequ%C3%AAncia_de_Fibonacci&oldid=56482987. Acesso em 16/10/2019.

Stack Overflow Developer Survey Results 2019. In: https://insights.stackoverflow.com/survey/2019. Acesso em 21/09/2019.

Stewart, Lauren (2019). Front End vs Back End Development. https://www.coursereport.com/blog/front-end-development-vs-back-end-development-where-to-start. Acesso em 15/09/2019.

Suzumura, T.; Tozawa, A.; (2008). Performance Comparison of Web Service Engines in PHP, Java and C.

Tiobe Index. In: https://www.tiobe.com/tiobe-index/. Acesso em 21/09/2019.

Anexo I - Aplicação Java

```java
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.sql.*;

// Classe principal da aplicação
public class Fibonacci extends HttpServlet {

    // Função principal, responsável por responder a uma requisição GET
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException
    {

        // Definição do tipo de resposta
        res.setContentType("text/html");

        // Objeto responsável por escrever o conteúdo na tela
        PrintWriter out = res.getWriter();

        // Inicialização das variáveis
        Connection conn = null;
        Statement stmt = null;
        int num = 0;
        int result = 0;

        try {

            // Definição das informações para conexão com banco de dados
            Class.forName("org.sqlite.JDBC");
            conn = DriverManager.getConnection("jdbc:sqlite:/var/lib/tomcat9/webapps/ROOT/WEB-
INF/classes/database.db");

            // Criação da consulta
            stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery( "SELECT * FROM numbers WHERE id = 7510;" );

            // Percorre os registros encontrados
            while (rs.next()) {
                // Obtém o número do banco de dados
                num = rs.getInt("number");
            }

            // Cálculo da sequência de Fibonacci
            result = fibonacci(num);

            // Atualização do registro no banco de dados
            stmt.executeUpdate("UPDATE numbers SET date = CURRENT_TIMESTAMP WHERE id = 7510;");

            // Fechamento da conexão
            stmt.close();
            rs.close();
            conn.close();

        } catch (Exception e) {
            e.printStackTrace(out);
        }

        // Exibição do resultado
        out.println("<p>" + result + "</p>");
        out.close();

    }

    // Função para cálculo recursivo da sequência de Fibonacci
    public int fibonacci(int num)
    {
        if (num <= 2) return 1;
        return fibonacci(num - 1) + fibonacci(num - 2);
    }

}
```

Anexo II - Aplicação Node.js

```
// Bibliotecas necessárias
var http = require('http');
var sqlite3 = require('sqlite3').verbose();

// Inicializa o servidor
http.createServer(function (req, res) {

    // Função para cálculo da sequência de Fibonacci
    function fibonacci(n) {
        if (n <= 2) return 1;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    // Função para atualizar o registro no banco de dados
    function updateRow() {
        var stmt = db.prepare("UPDATE numbers SET date = CURRENT_TIMESTAMP WHERE id = ?");
        stmt.run(id);
    }

    // Função para exibição da resposta
    function response(result) {
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end('<p>' + result + '</p>');
    }

    // Abre o banco de dados
    var db = new sqlite3.Database('./database.db', sqlite3.OPEN_READWRITE);

    // Definição do ID que será utilizado
    var id = 7510;

    // Acesso o banco de dados para a consulta do registro
    db.get("SELECT * FROM numbers WHERE id = " + id, (err, row) => {

        // Cálculo da sequência de Fibonacci
        var result = fibonacci(row.number);

        // Gravação no banco de dados
        updateRow();

        // Exibição da resposta
        response(result);

    });

}).listen(8080);
```

Anexo III - Aplicação PHP

```php
<?php

// Função para cálculo da sequência de Fibonacci
function fibonacci($n) {
    if ($n <= 2) return 1;
    return fibonacci($n - 1) + fibonacci($n - 2);
}

// Abertura do banco de dados e consulta do registro
$id = 7510;
$db = new PDO('sqlite:database.db');
$query = $db->query("SELECT * FROM numbers WHERE id = $id");
$result = $query->fetch();
$number = $result['number'];

// Chamada a sequência de Fibonacci
$final = fibonacci($number);

// Atualização do registro
$stmt = $db->prepare('UPDATE numbers SET date = CURRENT_TIMESTAMP WHERE id = $id');
$stmt->execute(array($id));

// Exibição do resultado em HTML
echo '<p>' . $final . '</p>';

?>
```