Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

UTILIZAÇÃO DA TECNOLOGIA NODE JS PARA CONSUMO EFICIENTE DOS RECURSOS DE SERVIDORES WEB.

Recebido: 22.02.2013
Publicado: 30.05.2013

Rafael Freitas Quixabeira, slz.rafael@gmail.com [1],
Will Ribamar Mendes Almeida, will75@gmail.com[1]
Patrício Moreira de Araújo Filho, pmaraujof@yahoo.com.br[2]

1. Faculdade Pitagoras de São Luís, São Luís MA.
2. Coordenação de Pesquisa e Extensão - Faculdade Pitágoras de São Luis - Ma

Abstract. *The need to maintain web applications with good performance depends on factors such as infrastructure and a robust hardware. However, performance may be affected by technology is used to create this application. The way programming languages manage the resources of the web servers, may be the cause of "bottlenecks" that occur at times of peak access and consequently making them very slow (response time) and in some cases off-line, without any access to users. The exponential growth of Internet users has generated a need to handle hundreds or even thousands of connections at the same time, this has caused a need for paradigm shifts in software development for web. In order to harness the full power of the hardware used, has used a new programming model oriented events Input / Output. The idea is to generate a computational savings when compared to conventional models. This model is applied in Node.js, which is a technology that uses javascript in the backend, facilitating the quick creation of applications and potentially scalable. The intention of this article to demonstrate the advantages and disadvantages of this event-driven paradigm, using the Node.js platform as a tool to create and verify by testing the performance superiority of this paradigm in some web applications market..*

Keywords: *web, node.js, scalable systems, javascript, v8.*

Resumo. *A necessidade de manter aplicações web com um bom desempenho, depende de alguns fatores como infraestrutura e um hardware robusto. Porém, o desempenho pode está sendo afetado pela tecnologia utilizada para a criação dessa aplicação. A forma como as linguagens de programação gerenciam os recursos dos servidores web, podem ser a causa dos "gargalos "que ocorrem nos momentos de picos de acesso e consequentemente deixando-os muito lento (tempo de resposta) e em alguns casos off-line, sem qualquer acesso aos usuários. O aumento exponencial de usuários da internet vem gerando uma necessidade de manipular centenas e até milhares de conexões ao mesmo tempo, isto tem causado uma necessidade de mudanças de paradigmas no desenvolvimento de softwares pra web. A fim de aproveitar todo o poder do hardware utilizado, tem-se utilizado um novo modelo de programação orientado à eventos de Input/Output. A ideia é gerar uma economia computacional quando se comparado aos modelos convencionais. Tal modelo é aplicado no node.js, que é uma tecnologia que utiliza javascript no back-end, facilitando a criação de aplicações rápidas e potencialmente escaláveis. Pretende-se com este artigo demonstrar as vantagens e desvantagens deste paradigma orientado a eventos, utilizando a plataforma node.js como ferramenta de criação, e constatar por meio de testes de desempenho a superioridade desse paradigma em algumas aplicações web de mercado.*

Palavras-chave: *web, node.js, sistemas escaláveis, javascript, v8*

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

1 INTRODUÇÃO

A crescente necessidade de manter os aplicativos *web* funcionando 24 horas por dia se torna um problema quando o número de usuários simultâneos que requisitam tal serviço ultrapassam os recursos disponíveis que o servidor web responsável pela aplicação suporta. Este fato se deve ao crescimento que a internet, o acesso à web, teve nos últimos dez anos, onde existiam 606 milhões de usuários e no ano de 2012 chegando à marca de 2,4 bilhões. [1][2] Outro fator importante, diz respeito a quantidade de tempo em que o usuário passa em frente ao computador navegando na *web*, seja no computador ou em um dispositivo móvel. Quanto ao crescimento deste tempo de utilização *on-line* pode-se afirmar sem medo que o Brasil líder nesse quesito com o tempo médio de acesso 50 horas e 30 minutos apenas em sites, somados ao acesso em redes sociais e *instant messengers* tem-se em média de 80 horas e 30 minutos. [3]

Uma solução viável para esse problema de momentos de picos de acesso seria o aumento do poder de processamento dos servidores *web*. Entretanto, do ponto de vista do software é necessário que a aplicação esteja preparada para assimilar uma carga crescente de conexões e processamentos, ou seja, escalabilidade. Essa é uma característica que indica a habilidade de um aplicativo estar preparado para crescer e manipular uma grande porção de trabalho. [4]

Além de escalar o aplicativo é necessário também um alto investimento em infraestrutura, hardware e mão de obra. O que nem sempre é possível devido às altas cifras envolvidas além do risco de gerar um resultado abaixo do esperado. Segundo Leitner, é possível que o gargalo seja ocasionado pelo *software*. [5]

Geralmente aplicações *web* que geram paginas dinâmicas, são constituídas por uma série de componentes, como: servidores de banco de dados, servidores de imagens e etc. Um deles pode se transformar no gargalo da aplicação. A fim de encontrar o local do gargalo, algumas técnicas como a de monitoramento e benchmarks. Através delas são coletados estatísticas e dados, que após analisados, indicam o local à ser aplicados otimizações.[6]

Se constatado que o problema está na forma que a tecnologia, responsável por gerar o conteúdo dinâmico gerencia os recursos do servidor será necessária a sua troca ou um incremento no poder de processamento do servidor. Caso opte pela troca, geralmente grande parte das linguagens e tecnologias de programação trabalha num mesmo modelo de gerenciamento de recursos.

Nesse panorama o *node.js* pode ser uma alternativa eficaz, devido seu modelo de programação incomum e a forma que gerencia seus recursos. [7]

2.O QUE É NODE.JS

Segundo o site oficial da tecnologia, o *node.js* é uma tecnologia construída em cima da *engine Javascript V8*, que é de código aberto e é também utilizada no navegador *Google Chrome*, ideal para a criação rápida de aplicativos e de aplicações em rede escaláveis.[8] Faz uso de um modelo baseado em eventos e de entrada e saída não bloqueante. Este ambiente foi desenvolvido em 2009 por Ryan Dahl utilizando a linguagem de programação *C++* para sua implementação.

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

2.1 Vantagens

Segundo Cantelon, alguns dos benefícios da utilização do *node.js* é o fato de utilizar o *javascript* tanto no *back-end* como front-end de aplicações *web* e ser largamente utilizado para manipulação de dados em bancos de dados *NoSQL*. [9]

Além disso, utiliza o *JSON*, que é um formato de texto para a serialização de dados estruturados. [10] O V8 implementa toda a especificação *ECMAScript* padrão, assim tornando possível a utilização de todo o poder da linguagem *javascript*. [9] O que não ocorre com todos os navegadores.

2.2 Desvantagens

Uma das principais desvantagens reside no fato da linguagem *Javascript* não possui uma biblioteca padrão de desenvolvimento. Além disso, a falta de módulos nativos dificultam o desenvolvimento de aplicativos de grande porte. Contudo, esse problema está sendo resolvido com o desenvolvimento do modulo *CommomJS*.

A utilização da programação orientada a eventos e assíncrona é um conceito relativamente novo para a programação no *back-end*, e pode levar um certo tempo até que esse se torne plenamente produtivo. Existe uma certa resistência quanto à sua utilização principalmente pelo fato de ser uma tecnologia nova e pouco difundida. [11]

3. COMPARATIVO COM OUTRAS TÉCNOLOGIAS

No principio era utilizado o *Commom Gateway Interface* (CGI), que era um método usado para permitir a interação entre um servidor *WWW* e outros programas que executavam no mesmo sistema operacional. [12] Era independente de linguagem, funciona com *Perl, C, Visual Basic* e outras. [13] O CGI para cada requisição do cliente criava um novo processo no sistema operacional, o que com as sucessivas requisições se tornava computacionalmente ineficiente e dificultava a escalabilidade [14] e [15].

Como alternativa foi utilizado o padrão *thread* por requisição, que gerava uma nova *thread* para cada requisição recebida. Esta alaternativa tinha como vantagem o menor consumo de recursos, mesmo não tornando o servidor *web* mais rápido, podia tratar um numero maior de requisições concorrentemente [16]. Em linguagens como *Java*™[17] e *PHP* [18], cada conexão inicia um novo encadeamento que, potencialmente, é acompanhado de 2 MB de memória. Em um sistema que tenha 8 GB de RAM, isto define o número máximo teórico de conexões simultâneas em cerca de 4.000 usuários [19].

Devido aos limites de escalabilidade impostos pelo modelo baseado em threads, muitos desenvolvedores têm optado por um modelo baseado em eventos para gerenciar concorrência. [20] Um modelo baseado em eventos pode proporcionar um melhor desempenho utilizando um único ciclo de loop de eventos [21]. O *node.js* faz uso desse modelo. O ambiente *javascript* utilizado no *back-end* é *single-threaded*. Para que se tenha um I/O baseado em eventos e não bloqueante se fez necessário o uso das bibliotecas *libev* e *libeio* [20]. Para isso, são utilizados *callbacks* que registram um trecho de código que será executado quando um determinado evento ocorrer. Um exemplo do *callback* "*click*" utilizado no *framework jQuery* [22] é apresentado conforme o código da Figura 1. O alerta só vai ser mostrado quando ocorrer um click, no elemento HTML de *id* igual à "botao_marcado".

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

```
$("#botão_marcado").click(function(){
        alert("VOCÊ CLICOU NO BOTÃO DE ID 'BOTÃO'");
});
```

Figura 1: Código *javascript* com *jquery*

Como demonstrado na Figura 2, foi utilizada a linguagem de programação *PHP*, para fazer uma busca no banco de dados e ao depois mostrar os resultados obtidos. Ao final da instrução é apresentado uma mensagem informativa do tipo "TUDO OK". O mesmo procedimento pode ser verificado na Figura 3, utilizando linguagem node.js.

```
<?php
$conexao= mysqli_connect("localhost","root","123","nodejs");
$result = mysqli_query($con, "SELECT * FROM tabela");
mostrar_resultados($result);
echo 'TUDO OK';
?>
```

Figura 2: Código da linguagem de programação em *PHP*

```
var mysql = require('mysql');
var client = mysql.createConnection({
host     : 'localhost',
user     : 'root',
password : 'admin',
database : 'nodejs'
});

client.query('SELECT * FROM tabela',
  function(erro, resultado, linhas){
    console.log(resultado);
});

console.log("TUDO OK");
```

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

Figura 3: Código em linguagem node.js

Esse trecho de código executa o mesmo procedimento desenvolvido em linguagem *PHP*. Porém os resultados recebidos são diferentes. Pois, enquanto no primeiro exemplo os resultados da busca são mostrados primeiro, com o node acontece o contrario. É primeiro mostrado a mensagem "TUDO OK" e depois os resultados da busca.

O que acontece é que no *PHP* ocorre um bloqueio na linha em que se espera o resultado da consulta no banco de dados e só depois que os resultados são salvos é que o programa continua. Como o node faz uso de um *I/O* não bloqueante, ele continua o programa, e quando os resultados chegam, ele dispara o callback e executa o trecho de código registrado para ser executado quando os resultados forem salvos.

4. TESTES DE DESEMPENHO

A fim de demonstrar uma prática demosntsrando as vantagens dessa tecnolgia e de se verificar as diferentes situações do desempenho. Fez-se alguns testes comparativos em relaçao a performance do *node.js* em comparação com a linguagem *PHP*.

4.1 Recursos Utilizados

Para a realizaçao de testes será utilizado um computador com um processador *Intel Core i3* M330@ 2.13*GHz* equipado com 2 GB de memória RAM do tipo DDR3. O sistema operacional utilizado será o *Ubuntu* [23] *Precise Pangolin* 12.04LTS 32 bits. Também será utilizado o *Apache* [24] 2.2.22 com o modulo *PHP* 5.3.10, a versão do *node.js* utilizada será 0.11.0. No caso da simulação para os testes será utilizado o *ApacheBench* [25], que será responsável pelo o envio das requisições concorrentes e pela análise de desempenho de HTTP do servidor *web*.

4.2 Teste (Hello World)

Na Figura 4, é demonstrado a implementaçao do código que será utilizado durante o primeiro teste. A idéia é criar um servidor HTTP e escrever uma mesngem do tipo "Hello World" como resposta em todas as requisições solicitadas.

ISSN: 2317-7403

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
```

Figura 4: Criando um servidor http e escrevendo "Hello World" para respostas.

Na Figura 5 é demonstrado a implementaçao do código em *PHP*, que escreve "Hello World".

```
<?php
        echo "Hello World!";
?>
```

Figura 5: Código PHP, escrevendo "Hello World "

Parâmetros utilizados no *ApacheBench* para os resultados:

-r : ignorar erros
-n : número de requisições
-c : número de requisições concorrentes

node.js

```
ab -r -n 20000 -c 1000 http://localhost:1337/
```

PHP+Apache

```
ab -r -n 20000 -c 1000 http://localhost/php
```

4.2.1 Resultados

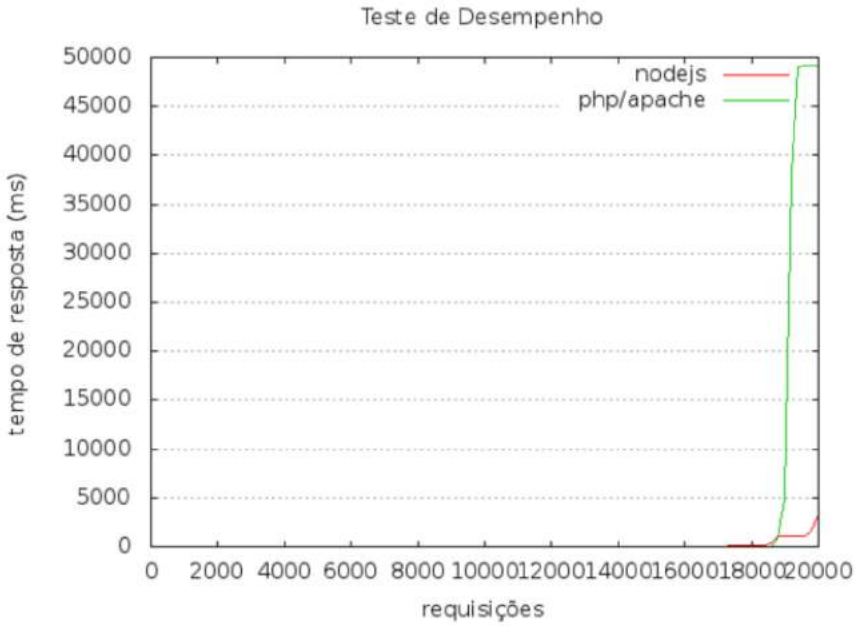

Figura 6: Resultados do teste de desempenho.

Tabela 1: Dados do teste de desempenho

DADOS	**NODE.JS**	**PHP+APACHE**
Nível de Concorrência	1000	1000
Tempo Total	4.061 segundos	49.219 segundos
Requisições Completas	20000	20000
Requisições Falhas	0	2427
Requisições por segundo	4924.91 segundos	406.34 segundos
Tempo por requisição	203.049 milissegundos	2460.973 milissegundos

De acordo com a Figura 6 e a Tabela 1 os resultados demonstraram um resultado favorável ao node.js. O *PHP* chegou a levar 10 vezes mais tempo para responder todas as requisições e ainda assim com algumas falhas.

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

4.3 TESTE (BUSCA BANCO DE DADOS)

O banco de dados utilizado foi o *MySQL Server* 5.5.29 .[26] A tabela criada foi criada com o seguinte código *SQL*:

```
create table tabela (id int AUTO_INCREMENT, PRIMARY KEY(id));
```

Na Figura 7, é mostrado o código de acesso e de busca no banco de dados utilizando o *PHP*. Também é mostrado nesta figura o código que escreve a resposta do número de resultados da busca.

```
<?php
        $conn = new mysqli('localhost', 'root', 'admin', 'nodejs');
        $result = $conn->query('SELECT * FROM tabela');
        echo ("RESULTADOS: ". $result->num_rows);
 ?>
```

Figura 7: código PHP de busca no banco de dados.

O mesmo procedimento é feito utilizando-se o *node.js*. Percebe-se, que na segunda linha de código, demonstrado na Figura 8, o módulo do banco de dados *Mysql* é importado.

ISSN: 2317-7403

```
var http = require('http');
var mysql = require('mysql');
var client = mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : 'admin',
        database : 'nodejs'
});
http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});

        client.query('SELECT * FROM tabela',
        function(erro, linhas, colunas){
                res.write("RESULTADOS: " + linhas.length);
                res.end();
        });

}).listen(1337, '127.0.0.1');
```

Figura 9: Código de acesso ao banco de dados utilizando o modulo *mysql* do node.js.

Parâmetros utilizados no *ApacheBench* para os resultados:

node.js

```
ab -r -n 1000 -c 500 http://localhost:1337/
```

PHP+Apache

```
ab -r -n 1000 -c 500 http://localhost/php
```

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

4.3.1 Resultados

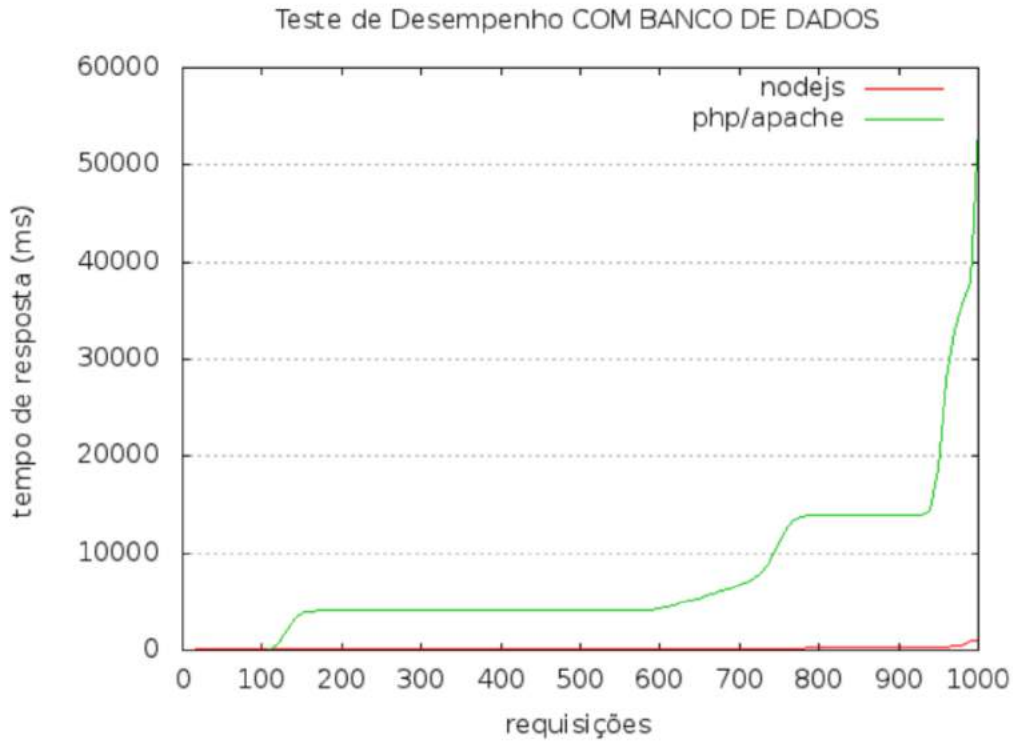

Figura 10: Gráfico de resultados do segundo teste.

Tabela 2: Dados referentes ao segundo teste.

DADOS	**NODE.JS**	**PHP+APACHE**
Nível de Concorrência	500	500
Tempo Total	1.075 segundos	54.202 segundos
Requisições Completas	1000	1000
Requisições Falhas	0	606
Requisições por segundo	930.66	18.45
Tempo por requisição	1.075 milissegundos	54.202 milissegundos

A partir da análise dos gráficos da Figura 9, pode-se verificar que com o aumento do numero de requisições o *PHP* começou a aumentar o seu tempo de resposta, fato que não aconteceu com a plataforma *node.js*, pois este conseguiu tratar a mesma demanda de requisições em um tempo bem pequeno. De acordo com os dados da Tabela 2, pode-se comprovar a eficiência e a confiabilidade de um sistema feito em *node.js*, pois conseguiu tratar todas as requisições sem falhas, o que não ocorreu com o *PHP*.

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

5. CONSIDERAÇÕES FINAIS

Neste artigo, teve-se o intuito de demonstrar alguns dos problemas enfrentados pela maioria dos serviços *web* da atualidade e apresentar os modelos de gerenciamento de recursos utilizados por diferentes tecnologias em diferentes épocas. Diante de tudo isso, verificou-se a eficiência do *node.js* por meio de um simples teste de desempenho. Quando comparado as tecnologias com uma grande progessão no mercado, como é o caso do *PHP*.

Por se tratar de uma ferramenta livre e de código aberto, tem tido grande adoção por parte da comunidade open-source, sendo inclusive o repositório mais famoso no github, ultrapassando tecnologias famosas como o Ruby on Rails. Este tem sido utilizado em grande escala por empresas como a *Microsoft* e a *Yahoo*.

Enfim, ficou claro a simplicidade e o poder do node.js, cabe aos gerentes de projetos a adoção dessa tecnologia, analisando se esta pode ser ou não a solução para seus problemas.

REFERÊNCIAS

EYSENBACH, Gunther. **The Impact of the Internet on Cancer Outcomes**. University of Toronto, CANADA. (2003), disponivel em: http://www.ncbi.nlm.nih.gov/pubmed/15224975.

INTERNET WORLD STATS. **World Internet Users Statistics Usage and World Population Stats**. 30 de Junho de 2012. Disponivel em: http://www.internetworldstats.com/stats.htm.

FOLHA DE SAO PAULO. **Brasil bate recorde em tempo de navegação e número de internautas.** 27 de Agosto de 2012. Disponível em: http://www1.folha.uol.com.br/folha/informatica/ult124u438362.shtml.

GOMES, Diego. **O que é escalabilidade?** 2010. Disponível em: http://escalabilidade.com/2010/01/31/o-que-e-escalabilidade.

LEITNET, Felix von. **Scalable Network Programming**. 2003. Disponível em: http://bulk.fefe.de/scalable-networking.pdf.

MARTINS, Carlos et al. **Otimização de desempenho em ambientes web**, Revista Pensar Tecnologia. (2011). Disponível em: http://www.inforium.com.br/revista/pensar/tecnologia/art/artigo.php?no=03.

NODE.JS, "**Node.js**", http://nodejs.org/, 2013;

GOOGLE. "**V8 javascript Engine**". http://code.google.com/p/v8/, 2013.

CANTELON, Mike et al. **Node.js In Action**, Editora Maning, 1º Ediçao 2012.

FONSECA, Ruben. **Alternativas ao XML: YAML e JSON**, Universidade do Minho, Portugal 2007. Disponível em: http://repositorium.sdum.uminho.pt/handle/1822/6230.

ELOFF, Eric e Daniel Torstensson. **An Investigation into the Applicability of Node.js as a Platform for Web Services**, Universidade de Linköping, Suecia 2012. Disponível em: http://liu.diva-portal.org/smash/get/diva2:550993/FULLTEXT01.

TAROUCO, Liane Margarida Rockenbach. **Redes de Computadores e suas aplicações na Educação**. UFRGS. Disponível em: http://penta.ufrgs.br/.

LENOX, J. et al, **Common Gateway Interface for SIP**, Network Working Group, 2001. Disponível em: http://www.hjp.at/doc/rfc/rfc3050.html.

Acta Brazilian Science
Ano I, vol.01, 1º trimestre, Abril de 2013, São Luis, Ma, Brasil
ISSN: 2317-7403

BLACKWELL, Jr. et al. **Web Aplication Server with Secure Commom Gateway Interface,** United States Patent nº 5,857.191, 1999. Disponível em: http://www.google.com/patents?id=zRYXAAAAEBAJ&printsec=abstract&zoom=4&hl=pt-PT#v=onepage&q&f=false.

WIKIPEDIA, **FastCGI**, Disponível em: http://en.wikipedia.org/wiki/FastCGI.

KUEHNE, Bruno Tardiole. **Servidores Web**, ICMC-USP, 2007. Disponível em: http://lasdpc.icmc.usp.br/disciplinas/pos-graduacao/sistemas-distribuidos/2007/monografias-seminarios/Monografia_WS_2007%20-%20Bruno.pdf?set_language=pt-pt&cl=pt-pt.

Java: "**Write once, run anywhere**", http://www.java.com/en/, 2013
PHP: "**Pre Hypertext Processor**", http://php.net/, 2013

ABERNETHY, Michael. **O que exatamente é o Node.js?** IBM, 24 de junho de 2001. Disponível em: http://www.ibm.com/developerworks/br/library/os-nodejs/

JUNIOR, Francisco de Assis. **Programação Orientada a Eventos no lado do servidor utilizando Node.js,** Ifactory Solutions. Disponível em: http://www.infobrasil.inf.br/userfiles/16-S3-3-97136-Programa%C3%A7%C3%A3o%20Orientada___.pdf

MCCUNE, Robert Ryan. Node**.js Paradigms and Benchmarks**, University of Notre Dame, FRANCA 2011. Disponível em: http://netscale.cse.nd.edu/twiki/pub/Edu/GradOSF11ProjProposal/nodejs_proposal.pdf

JQUERY: "**The Write Less, Do More**", jquery.com, 2013.

UBUNTU, "**Ubuntu**", http://www.ubuntu.com/, 2013.

APACHE, "**HTTP Server Project**", http://httpd.apache.org/, 2013

APACHEBENCH, " **Apache HTTP server benchmarking tool**", http://httpd.apache.org/docs/2.2/programs/ab.html, 2013

MYSQLL, "**The world's most popular open source database**", http://www.mysql.com/