Análise de Desempenho de Tempo de Resposta e Consumo de Recursos no Carregamento de Páginas Web Utilizando BigPipe

Cristian Cleder Machado
Mestrando em Computação pela Universidade Regional Integrada do Alto Uruguai e das Missões.

Heverton Luan Roieski
Graduado em Ciência da computação pela Universidade Regional Integrada do Alto Uruguai e das Missões.

Resumo

Com o constante aumento e a diversidade de serviços oferecidos na Internet, tais como, redes sociais, comércio eletrônico, as páginas Web tornaram-se cada vez mais dinâmicas e interativas. Um dos problemas encontrados nessas mudanças está relacionado ao carregamento do conteúdo destas páginas que ao longo do processo evolutivo computacional manteve-se praticamente inalterado. Neste contexto, este artigo apresenta uma análise de desempenho do tempo de resposta no carregamento de páginas Web utilizando como alternativa à programação tradicional de páginas Web uma técnica denominada BigPipe. O objetivo é apresentar o ganho no tempo de resposta ao utilizar o BigPipe. O BigPipe é um redesenho do sistema de carregamento de páginas Web dinâmicas, onde páginas Web são decompostas em pequenas partes chamadas pagelets que são paralelizadas em diversos estágios de execução dentro dos servidores Web e dos navegadores. Para realização da análise foram desenvolvidos dois cenários, um utilizando uma página Web desenvolvida de maneira tradicional e o outro utilizando BigPipe. Os testes foram realizados em alguns navegadores e *hardwares* distintos com o objetivo de analisar o tempo de resposta no carregamento das páginas e a utilização de recursos computacionais.

Palavras-chave: Bigpipe, Análise de desempenho, Páginas Web dinâmicas.

Introdução

Com a constante presença da computação na vida das pessoas a Internet torna-se hoje peça fundamental para o desenvolvimento da sociedade oferecendo um rápido e - muitas vezes - fácil acesso a uma diversidade de serviços e informações. A Computação passou - e vem passando - por um processo evolutivo intenso. Desde o seu surgimento, foram desenvolvidas diversas tecnologias, tais como, a computação paralela, as redes de computadores, os mecanismos pipeline e a própria Internet. Com a popularização da Internet, o seu conteúdo também teve fortes aprimoramentos e a partir do surgimento dos site de comércio eletrônico, de jogos online, *streaming* de vídeo e das redes sociais a Internet teve sua utilização significativamente aumentada (**cisco-2013VNI**; **begen2011watching**).

As páginas Web estão cada vez mais dinâmicas, ou seja, vem utilizando linguagens de programação, tais como, PHP, ASP, JSP, entre outras, para alterar suas informações baseadas em banco de dados, posições geográficas ou por armazenamento de informações dos usuários nos navegadores. Porém, o carregamento do conteúdo destas páginas ainda é um problema. Isto deve-se ao carregamento tradicional que as páginas seguem

que, ao longo do processo evolutivo computacional, manteve-se praticamente inalterado. Para os sites mais modernos, que fazem a utilização de elementos que exigem mais recursos, tais como, CPU e memória, o modelo tradicional é muito ineficiente, pois as operações de carregamento são sequenciais e não podem ser sobrepostas.

Algumas técnicas de otimização, como o retardamento e a paralelização do *download* dos recursos das páginas, tem sido adotadas como tentativa de solucionar algumas limitações (**mai-2012case**). Porém, mesmo com a utilização destas otimizações, percebe-se que ainda existe um problema presente quando observada a troca de informações entre navegador e servidor Web. Tanto o servidor, quanto o navegador, seguem gastando ciclos de processamento - sem fazer nada - aguardando o processamento e renderização das informações no lado do servidor e o processamento e apresentação destas informações no lado do navegador.

Sendo assim, uma paralelização, ou seja, a execução de tarefas simultâneas, entre a montagem da página, realizada pelo servidor Web, e a renderização da mesma, realizada pelo navegador, pode não só reduzir a latência gerada entre ambos, como também fazer com que o navegador forneça ao usuário uma resposta muito mais rápida. Neste contexto, este artigo apresenta uma análise de desempenho do tempo de resposta no carregamento de páginas Web utilizando como alternativa a programação tradicional de páginas uma técnica denominada BigPipe (**jiang2010facebook**). BigPipe é uma tecnologia utilizada pelo Facebook[1] (**barrigas2014facebook**) que possui como ideia geral fazer uma quebra das páginas Web em múltiplas partes chamadas *pagelets* a fim de paralelizá-los através de vários estágios de execução dentro dos servidores Web e dos navegadores.

O objetivo neste trabalho é identificar melhorias ao substituir métodos tradicionais de carregamento de informações em páginas dinâmicas que possuem uma sequência estática para apresentação das informações por métodos baseados em fragmentação da estrutura que carregam a página conforme cada fragmento é recebido. Esse tipo de técnica faz com que se elimine a dependência de alguns determinados processos para a montagem da página. Para realização da análise foram desenvolvidos dois cenários, um utilizando um página Web desenvolvida de maneira tradicional e o outro utilizando BigPipe. Os testes realizados visaram analisar o desempenho (uso de CPU e memória) entre alguns navegadores e *hardwares* distintos. A metodologia para a execução dos testes consistiu em realizar 30 execuções em cada caso de estudo, nos quais utilizou-se a ferramenta Pingdom Tool[2] para obter os tempos de carregamento total das páginas criadas.

Este artigo está organizado como segue: Na seção II é apresentado um resumo sobre a técnica BigPipe. Na Seção III é apresentada a modelagem e desenvolvimento dos sites para teste. Na Seção IV é descrita a metodologia para realização dos experimentos. Na Seção V são discutidos os experimentos realizados. Por fim, na Seção VI são apresentadas as conclusões.

BigPipe

O BigPipe é uma técnica de desempenho para páginas Web dinâmicas, implementada pelo Facebook para melhorar a velocidade de carregamento das páginas Web. A ideia principal do BigPipe é quebrar as páginas Web em pequenos pedaços chamados *pagelets* e realizar a paralelização do carregamento destes dentro dos estágios de execução que ocorrem entre os servidores Web e os navegadores (**jiang2010facebook**).

Atualmente, a interatividade é o foco principal das páginas Web. Para que esta interatividade aconteça de forma rápida, é necessário que a conexão ofereça uma boa velocidade. Além disso, o modelo de carregamento das páginas Web não foi melhorado de forma a oferecer melhor velocidade, e continua, portanto, sendo executado de forma sequencial (**meyerovich2010fast**).

No modelo tradicional de carregamento, o navegador envia uma requisição HTTP para o servidor Web. Este, por sua vez, analisa a requisição, formula a resposta e a envia ao navegador. Após o recebimento da resposta, o navegador monta a árvore DOM e faz o *download* dos estilos visuais e *scripts* referenciados, aplicando estes estilos na árvore e executando os *scripts* em sequência (**mai2012case**; **meyerovich2010fast**).

As soluções adotadas atualmente para otimização deste processo são ineficazes, pois acabam tendo como uma limitação o gargalo gerado pela transferência feita entre o navegador e o servidor Web, que ocorre devido a Internet e

1 <http://www.facebook.com>.

2 Disponível no endereço: <http://tools.pingdom.com/fpt/>.

da forma de carregamento sequencial. Enquanto o servidor Web está ocupado gerando a página requisitada até o envio da resposta, o navegador segue desperdiçando ciclos à espera de uma resposta do servidor. O mesmo acontece quando o navegador recebe a resposta e renderiza a página. Neste caso, o servidor já não tem interferência e não pode mais ajudar em nada (**mai2012case**; **meyerovich2010fast**).

Durante o tempo gasto entre o envio, análise e processamento de resposta, e também do recebimento e renderização da página, nenhuma resposta inicial é mostrada ao usuário. Portanto, ao paralelizar este processo, ganha-se não só uma redução na latência fim-à-fim, mas também uma resposta inicial muito mais rápida e visível ao usuário.

Para explorar o paralelismo entre o servidor Web e o navegador, o BigPipe realiza a quebra das páginas Web, transformando os pedaços em *pagelets*, e quebra também o processo de geração da página em vários estágios. Estes estágios são numerados da seguinte forma:

1. O servidor Web analisa a requisição HTTP.
2. O servidor Web obtém os dados da camada de armazenamento.
3. O servidor Web gera a resposta HTML.
4. Ocorre a transferência da resposta, através da internet, para o navegador.
5. O navegador realiza o download do CSS requerido pela página.
6. O navegador aplica os estilos visuais na árvore DOM.
7. O navegador realiza o download do JavaScript referenciado.
8. O navegador executa o código na página.

Nestes estágios, os três primeiros são executados pelo servidor, e os últimos quatro pelo navegador. Apesar dos *pagelets* passarem pelos estágios de forma sequencial, o BigPipe permite que vários *pagelets* sejam executados simultaneamente em diferentes estágios (**jiang2010facebook**).

Modelagem e Implementação

Para realização da análise foram desenvolvidos dois cenários, um utilizando um página Web desenvolvida de maneira tradicional e o outro utilizando BigPipe. As linguagens de programação utilizadas para o desenvolvimento das páginas foram o C#, o JavaScript, o *HyperText Markup Language* (HTML) e o *Cascading Style Sheets* (CSS). Dentre as ferramentas, *frameworks* e servidores destacam-se o .NET *Framework*, o ASP.NET, o *Visual Studio*, o Adobe Fireworks, o MySQL e o *Internet Information Services* (IIS).

O passo inicial para a criação do layout e separação das *pagelets* foi analisar o que poderia fornecer maior carga de processamento e uso de memória. Houve também uma preocupação no quesito de utilização do mesmo layout, tanto para o modelo de programação tradicional, ou seja, um site que segue apenas o padrão de desenvolvimento MVC do ASP.NET, quanto para a forma otimizada do site, utilizando o BigPipe. A Figura 1 apresenta a esquerda a página inicial do site e a direita a fragmentação da mesma.

Na Figura 1, pode-se destacar alguns dos *pagelets* utilizados na implementação com o BigPipe. Um exemplo é o segmento intitulado "Mais Notícias", o qual compreende notícias que estão visíveis e outras semi-visíveis, onde o usuário pode optar por fazer o *scrol l* e assim visualizar outras 6 notícias que estão ocultas.

Seguindo a ideia de fragmentação da página, chegou-se a um total de 13 *pagelets* gerados para este site. É importante esclarecer que este número total de fragmentos não tem vínculo com o número total de *pagelets* que podem ser criados para o BigPipe. Não existe uma especificação indicando uma limitação para tal.

Metodologia de testes

A metodologia para a execução dos testes consistiu em realizar 30 execuções de testes em cada caso. Os testes iniciais foram realizados com a ferramenta Pingdom Tool com o objetivo de obter os tempos de carregamento total das páginas criadas. Neste processo, pôde-se obter resultados que foram utilizados para gerar gráficos de desempenho e, assim, analisar os ganhos e perdas na utilização do BigPipe. Na sequência, foram executados testes de utilização de CPU e memória. Estes testes foram realizados com os três navegadores mais utilizados atualmente: o Google Chrome, o Internet Explorer e o Mozilla

Firefox

Além dos testes para análise de tempo de carregamento, foram realizados testes de utilização de CPU e memória RAM. O primeiro

objetivo destes testes era apresentar o consumo de recursos em cada cenário. O segundo objetivo era demonstrar se há ganhos de desempenho no carregamento de páginas Web com a utilização do BigPipe, mesmo se este esteja sendo carregado em um computador monoprocessado, que não oferece suporte ao paralelismo de processos. Sendo assim, dois computadores com configurações diferentes foram utilizados nos quais o primeiro possui um processador Celeron de apenas um núcleo de processamento e 1GB de memória RAM e o segundo possui um processador Pentium Dual Core com dois núcleos e 4GB de memória RAM.

Testes e Discussões

Para os primeiros testes, foi utilizada a ferramenta Pingdom Tool, uma página que executa testes de performance em páginas Web baseado a partir de configurações escolhidas pelo usuário. Estes testes são executados em navegadores reais a fim de corresponder à experiência real do usuário no carregamento das páginas Web.

A Figura 2, apresenta os testes realizados no site desenvolvido de forma tradicional (Normal). Neste gráfico, pode-se observar o desempenho obtido no carregamento total da página, no qual se destaca o menor tempo em 2,92 segundos e o pior desempenho em 4,40 segundos.

A Figura 3, demonstra o desempenho obtido no carregamento total da página com o BigPipe. Neste gráfico estão em destaque o melhor desempenho, 2,25 segundos, e o pior desempenho obtido, 4,48 segundos.

Em análise desses dois gráficos apresentados (Figura 2 e Figura 3) pode-se observar, que na maioria dos testes, o site com a implementação do BigPipe obteve melhor desempenho no quesito tempo de carregamento. Nestes testes com o BigPipe, os piores desempenhos ultrapassaram a linha dos 4 segundos por 4 vezes enquanto no site tradicional os piores desempenhos ultrapassaram esta linha 8 vezes. Ao verificar os melhores desempenhos, o site com BigPipe obteve resultados inferiores a 3 segundos em 8 testes e o site tradicional em apenas 3 destes testes. Essa diferença é obtida pela capacidade que o BigPipe tem em não manter os navegadores osciosos aguardando por resposta para apresentar a página de maneira sequencial, fazendo com que a página fosse montada independente da sequência de respostas das *pagelets*. Em última análise dessas informações, ao realizar uma média geral dos testes executados entre os sites, nota-se que o ganho no tempo de carregamento não demonstra grande variação. Esta média é apresentada pela Figura 4, na qual pode-se verificar que o ganho obtido com a implementação do BigPipe é de 0,176 segundos.

Como mencionado, a média do ganho obtido com o uso do BigPipe é de 0,176 segundos. É importante ressaltar que esse ganho está apontando um valor para o carregamento de uma página. Dessa forma, se for contabilizada a quantidade de vezes que um usuário vai carregar (novas) páginas dentro de um site, esse valor torna-se expressivo num cenário real cuja a navegação é contínua.

Na sequência dos testes, foi analisado o uso de CPU e memória em dois computadores, os quais têm como principal diferença a quantidade de núcleos de processamento. O primeiro computador a ser testado é um monoprocessado, ou seja, possui apenas um núcleo de processamento baseado em um processador Celeron e, sua memória RAM é de apenas 1 GB.

Foram realizados testes nos navegadores Internet Explorer, Firefox e Google Chrome, nos quais os resultados obtidos foram adquiridos através de análise dos logs gerados pelo Windows. A Figura 5 apresenta a maior utilização de CPU feita pelos navegadores no carregamento de ambos os sites, com e sem o BigPipe.

Na Figura 5, pode-se observar que houve maior utilização de CPU no carregamento do site desenvolvido de forma normal. Em ambos os casos de carregamento apresentados, o navegador Firefox teve 99% de utilização de CPU, já o navegador Internet Explorer obteve uma variação de 50% entre os casos, nos quais 6% de utilização de CPU foram empregados ao site normal e apenas 3% ao site com BigPipe.

O navegador Google Chrome é um caso à parte, pois o mesmo ao ser executado gera diversos processos distintos, o que impossibilita a identificação exata do processo utilizado para a renderização de página. Porém, para a análise feita, nesse caso, foi verificado em quais dos processos a variação de utilização de CPU era mais alta e, assim, os testes foram voltados ao processo do navegador com maior variação, quando esse estava renderizando a página.

Portanto, no caso do Google Chrome, a maior utilização de CPU esteve ligada ao site nor-

mal, o qual obteve 95% de utilização de CPU ao contrário do site com BigPipe, que obteve apenas 60% de utilização. No entanto, ao verificar-se a menor utilização de CPU, segundo a Figura 6, no navegador Firefox a utilização foi menor no site desenvolvido normalmente.

Os navegadores Google Chrome e Internet Explorer apresentaram um baixo consumo de recurso de CPU ao utilizar o BigPipe, pois abstraem testes de estruturação do site, preocupando-se somente com a montagem da informação na íntegra. Por exemplo, se uma determinada tag HTML não for fornecida por erro de programação do site, os navegadores Google Chrome e Internet Explorer podem imprimir essa página com erros. Indiferente disso, o Firefox teste tag a tag e no caso de uma falta de tag, ele tenta "adivinhar" qual deveria ser a tag que está faltando e em qual posição ela deveria estar. Além disso, o Firefox tende a montar as informações da página na sequência que elas aparecem, ou seja, quando identificada uma ação externa, por exemplo um include de uma página ou uma função, ele analisa de tempos em tempos se existe a resposta desta ação, afim de apresentar ao usuário as informações na ordem da sequência programada. Por esse motivo, o consumo de CPU do Firefox é superior ao utilizar BigPipe, pois ele tem um gasto (excessivo) em monitorar as respostas das *pagelets*.

Analisando os resultados apresentados na Figura 6, pode-se verificar uma grande variação de utilização nos navegadores Firefox e Google Chrome quanto aos teste obtidos no site normal. O navegador Internet Explorer não forneceu variação dentre os melhores resultados obtidos, mas, se comparados os melhores resultados apresentados pela Figura 6 com os resultados demonstrados na Figura 5, obtemos então as Figuras 7 e 8 que demonstram suas variações.

Os resultados obtidos nas Figuras 9 e 10 demonstram a utilização de memória dos processos relativos a cada navegador. Ao analisar esses gráficos, pode-se observar que a utilização de memória foi maior quando em carregamento do site com o BigPipe. Isso está ligado ao fato de que as requisições das *pagelets* são processadas após o retorno da resposta inicial dada pelo servidor. Assim, a primeira requisição enviada ao servidor retorna o conteúdo da *MasterPage* (estrutura inicial dá página), o que dá sequência ao envio da requisição que retornará a *view* e então as *pagelets*. O número de requisições enviadas ao servidor na implementação com o BigPipe é maior que na implementação sem o mesmo, o que pode explicar a maior utilização inicial de memória.

Os resultados obtidos nas Figuras 9 e 10 demonstram a utilização de memória dos processos relativos a cada navegador. Ao analisar esses gráficos, pode-se observar que a utilização de memória foi maior quando em carregamento do site com o BigPipe. Isso está ligado ao fato de que as requisições das *pagelets* são processadas após o retorno da resposta inicial dada pelo servidor. Assim, a primeira requisição enviada ao servidor retorna o conteúdo da *MasterPage* (estrutura inicial dá página), o que dá sequência ao envio da requisição que retornará a *view* e então as *pagelets*. O número de requisições enviadas ao servidor na implementação com o BigPipe é maior que na implementação sem o mesmo, o que pode explicar a maior utilização inicial de memória.

Nos testes de utilização de CPU em um multiprocessado com 2 núcleos os resultados encontrados demonstram que houve menor variação no site com a utilização do BigPipe. Nas Figuras 11 e 12 percebe-se que a maior variação encontrada é a do navegador Google Chrome, que obteve no site normal 15 pontos percentuais de variação e no site com a utilização do BigPipe 7 pontos percentuais de variação.

A maior utilização de CPU no uso do BigPipe se dá pelo fato de que o BigPipe tenta utilizar as CPUs continuamente para que as mesmas não se tornem osciosas. Para alcançar esse objetivo, cada *core* de cada CPU recebe uma quantidade de tarefas, por exemplo, um conjunto de *pagelets*. Dessa forma, cada *core* de cada CPU fica encarregado de determinadas tarefas, tentando executá-las o mais breve possível.

Quanto ao uso de memória, os testes realizados no computador com 2 núcleos demonstraram que a utilização da mesma é maior quando em carregamento no site com BigPipe. As Figuras 13 e 14 apresentam os resultados obtidos relativos à utilização de memória, onde é possível perceber que a utilização do BigPipe utiliza aproximadamente o mínimo de 91.000 Kbytes de memória, como é o caso do navegador Internet Explorer.

Como mencionado anteriormente, pode-se observar que a utilização de memória foi maior quando em carregamento do site com o BigPipe. Isso está ligado ao fato de o número de requisições enviadas pelo navegador ao servidor na implementação com o BigPipe é maior que na implementação sem o mesmo.

Conclusão

Esse artigo apresentou uma alternativa para diminuição no tempo de carregamento de páginas Web que tenham grande quantidade de itens dinâmicos. O BigPipe fornece um modelo de implementação que visa paralelizar esse carregamento quebrando a página requisitada em pequenos pedaços chamados *pagelets*.

O Facebook utiliza o BigPipe para melhoria de seus serviços e, assim, consegue diminuir em até duas vezes o tempo de resposta visual fornecida ao usuário. Ao contrário do Facebook que utiliza o PHP como linguagem de programação principal, a implementação apresentada nesse trabalho utilizou o C# que é integrante do ASP. NET e seguiu o modelo MVC para implementação do BigPipe.

A implementação dessa alternativa ao carregamento demonstrou fornecer significativa melhora sem necessitar de grandes alterações no código principal da página. Porém, a mesma apresentou algumas limitações quanto à utilização de vínculos JavaScript, o que faz com que o BigPipe nem sempre possa ser utilizado em sua totalidade. d Os testes realizados demonstraram as variações de utilização de CPU e memória em três dos navegadores mais utilizados atualmente e, assim, pode-se verificar que o consumo desses recursos foi maior quando o site com o BigPipe foi carregado. No entanto, os menores tempos de carregamento foram obtidos no site utilizando o BigPipe, demonstrando que apesar do consumo de recursos ser elevado a resposta obtida é mais rápida.

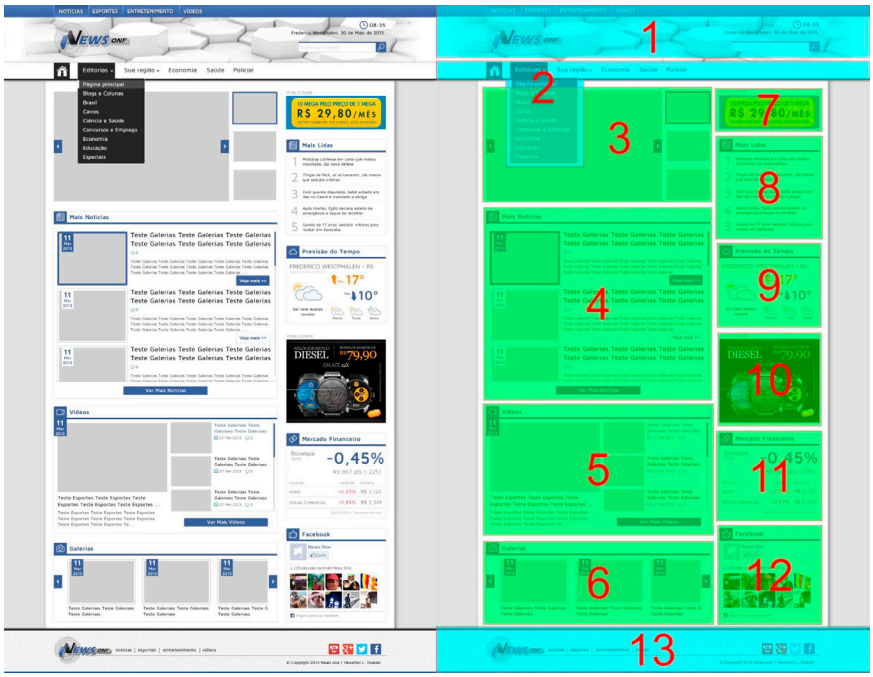

Figura 1. Página inicial do site desenvolvido e sua fragmentação.

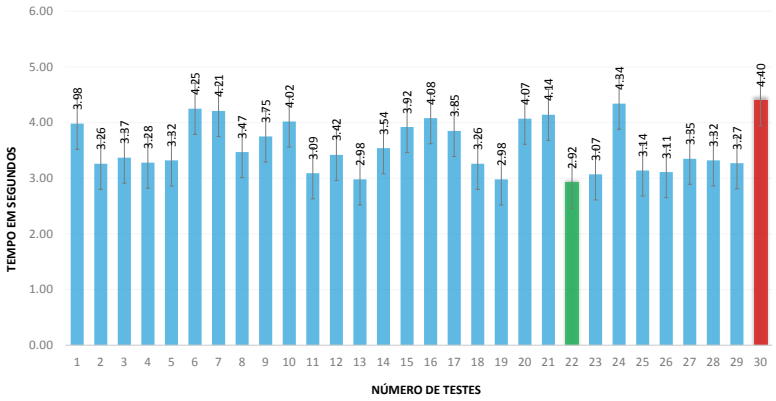

Figura 2. Tempo de execução da página de forma tradicional.

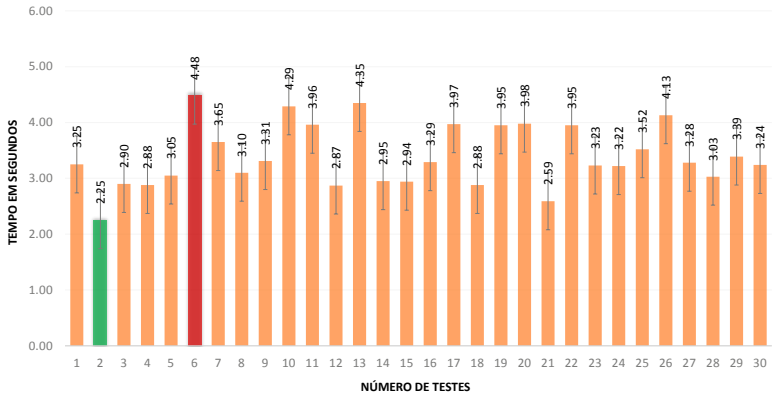

Figura 3. Tempo de execução da página utilizando BigPipe.

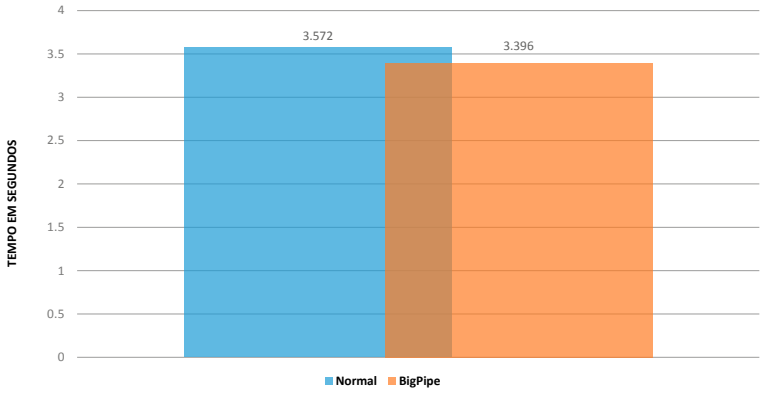

Figura 4. Comparação de tempo de execução da página de forma tradicional vs página utilizando BigPipe.

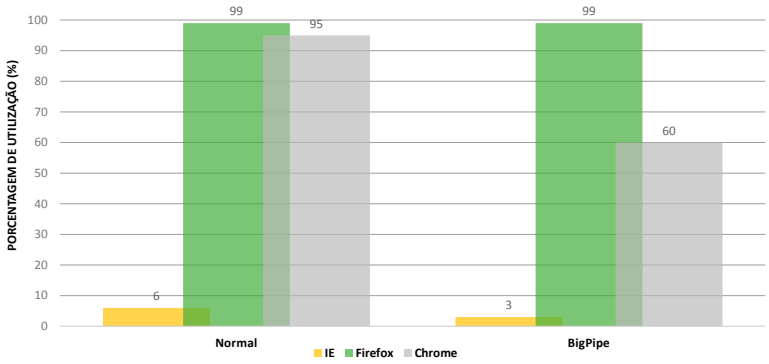

Figura 5. Maior utilização de CPU em um computador monoprocessado.

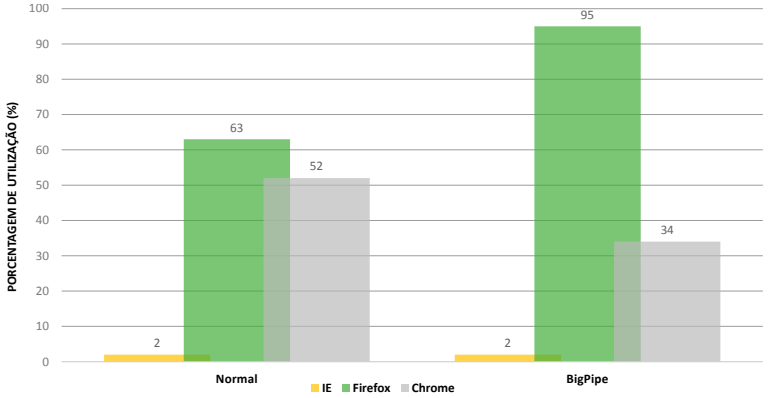

Figura 6. Menor utilização de CPU em um computador monoprocessado.

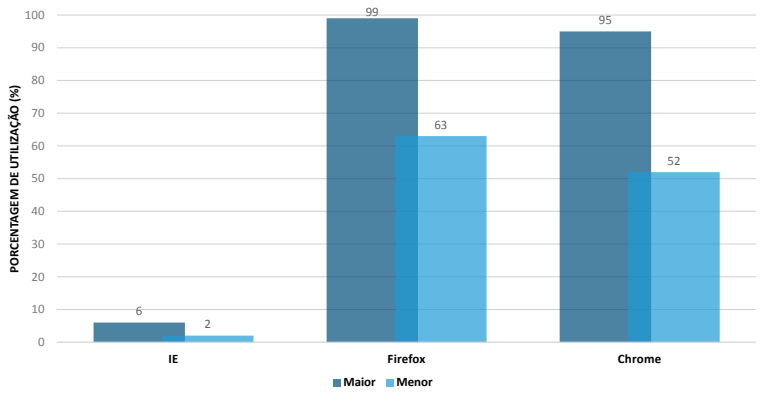

Figura 7. Comparativo de menor e maior tempo nas execuções monoprocessadas na forma normal.

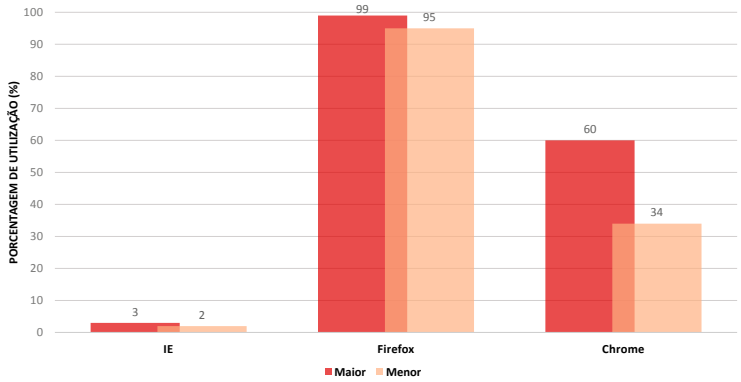

Figura 8. Comparativo de menor e maior tempo nas execuções monoprocessadas utilizando.

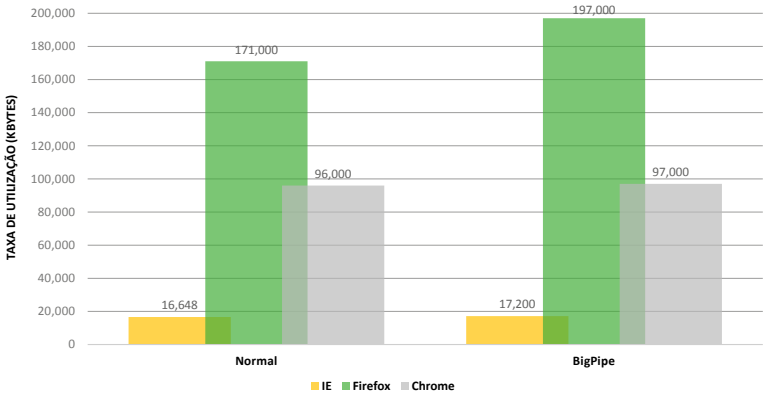

Figura 9. Maior utilização de memória em um computador monoprocessado.

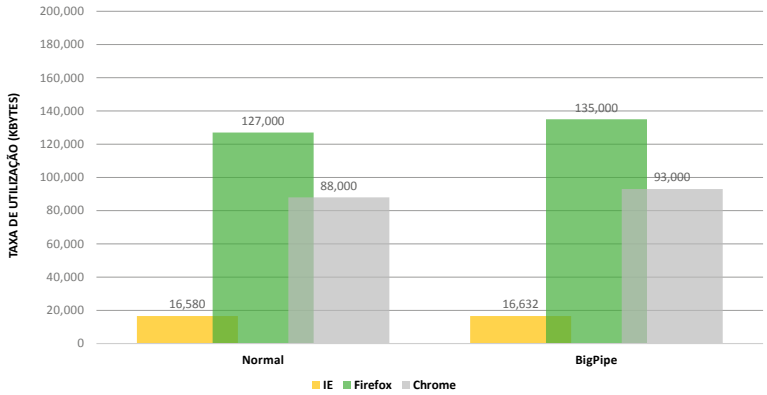

Figura 10. Menor utilização de memória em um computador monoprocessado.

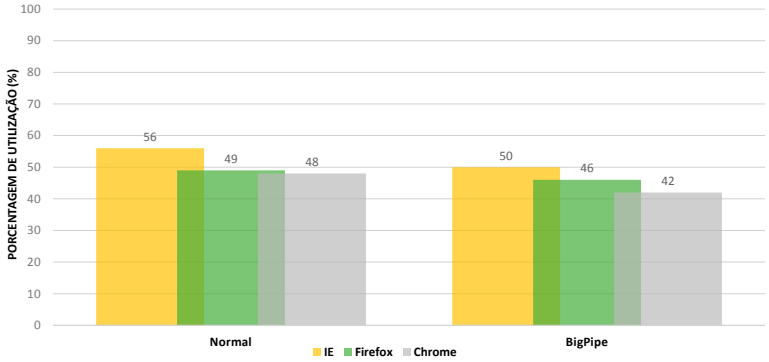

Figura 11. Maior utilização de CPU em um computador Dual Core.

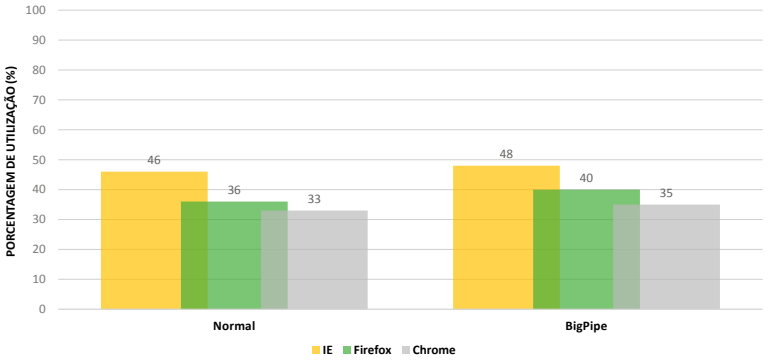

Figura 12. Menor utilização de CPU em um computador Dual Core.

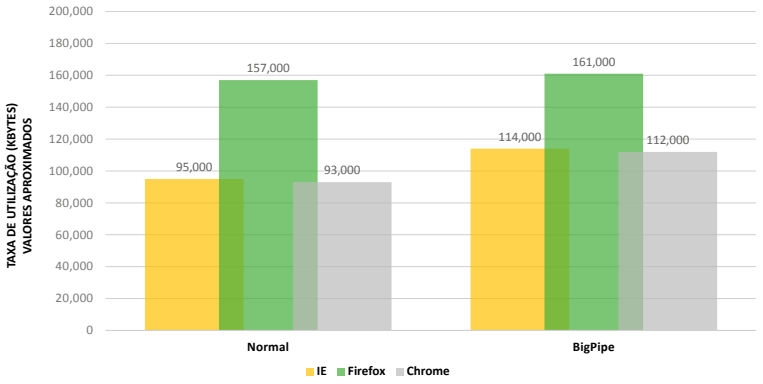

Figura 13. Maior utilização de memória em um computador Dual Core.

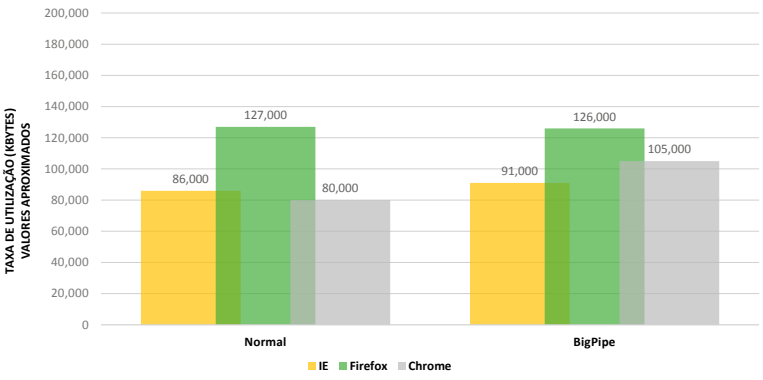

Figura 14. Menor utilização de memória em um computador Dual Core.

Response time and resource consumption performance analysis on web pages loading using BigPipe

Abstract

With the constant increase and diversity of services offered through the Internet, such as, social networks and e-commerce, web pages have become more dynamic and interactive. One of the problems found in such changes is related to the page content loading that during the process of computational evolution remained practically unchanged. In this context, this paper presents a performance analysis of response time on web page loading using as an alternative to traditional programming of web pages a technique called BigPipe. The goal is to show the response time increase when using BigPipe. The BigPipe is a tecnique that redesign the web pages loading system, decomposing the web pages in small chunks called pagelets that are parallelized in many stages of execution inside web servers and browsers. Two scenarios were developed for the analysis, one using a traditionally programmed web page and another with BigPipe technique. The testes were performed on different hardware and browsers to measure response time and computational resource.

Keywords: BigPipe, Performance analysis, Dynamic Web Pages.