Análise e Implementação de Paralelização Estrutural utilizando BigPipe para Alta Performance de Páginas Web Dinâmicas

Autor

Afiliação

Resumo

Com o constante aumento e a diversidade de serviços oferecidos na Internet, tais como, redes sociais, comércio eletrônico, entre outros, as páginas web tornaram-se cada vez mais dinâmicas e interativas, exigindo novos estudos e implementação de novas soluções para suprir demandas não previstas. 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. Este problema torna-se presente quando observada a troca de informações entre navegador e servidor web, que segue gastando seus ciclos - sem fazer nada -, aguardando o processamento e renderização das informações em cada um dos lados. Neste contexto, este artigo aborda uma alternativa para melhoria de desempenho no carregamento de páginas web, denominada BigPipe. BigPipe é um redesenho do sistema de carregamento de páginas web dinâmicas. A ideia geral é decompor páginas web em pequenas partes chamadas pagelets e paralelizá-los em diversos estágios de execução dentro dos servidores web e dos navegadores. O presente artigo teve como objetivo implementar uma página de maneira tradicional e outra utilizando BigPipe, realizando testes de desempenho (uso de CPU e memória) entre alguns navegadores e hardwares distintos. Os resultados demonstram o ganho na utilização do BigPipe.

Keywords: bigpipe; alta performance; paralelização estrutural; páginas web dinâmicas

Análise e Implementação de Paralelização Estrutural utilizando BigPipe para Alta Performance de 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, das redes sociais, entre outros, a Internet teve a sua utilização significativamente aumentada (Cisco, 2013; Begen, Akgul & Baugher, 2011).

As páginas web estão cada vez mais dinâmicas e interativas. 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, nele empregadas, 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 das limitações (Mai, Tang, King, Cascaval & Montesinos, 2012). 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. Os mesmos, 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 entre a montagem da página realizada pelo servidor web e a renderização da mesma 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 aborda uma alternativa para melhoria de desempenho no carregamento de páginas web, denominada BigPipe (Jiang, 2010). BigPipe é uma tecnologia utilizada pelo Facebook [1] (Barrigas, Barrigas, Barata, Furtado & Bernardino, 2014) que possui como ideia geral, fazer uma quebra das páginas web em múltiplas partes chamadas pagelets e, assim, paralelizá-los através de vários estágios de execução, dentro dos servidores web e dos navegadores.

Neste contexto, o presente artigo apresenta a implementação e análise de uma página web desenvolvida de maneira tradicional e outra utilizando BigPipe. O objetivo 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 de determinados processos para a montagem da página. 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 é apresentado um resumo sobre a técnica BigPipe. Na Seção é apresentada a modelagem e desenvolvimento dos sites para teste. Na Seção é descrita a metodologia para realização dos experimentos. Na Seção são discutidos os experimentos realizados. Por fim, na Seção são apresentadas as conclusões.

[1] http://www.facebook.com

[2] Disponível no endereço http://tools.pingdom.com/fpt/

BigPipe

O BigPipe é uma técnica de desempenho para páginas web dinâmicas, implementada pelo Facebook para servir as páginas web e melhorar a velocidade de carregamento perceptiva ao usuário. A ideia principal do BigPipe, é quebrar as páginas web em pequenos pedaços chamados *pagelets*, e realizar a paralelização do carregamento destes, assim como no pipeline, dentro dos estágios de execução que ocorrem entre os servidores web e os navegadores.

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.

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 o envia ao navegador. Então, 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*.

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 pela internet e de forma sequencial. Enquanto o servidor web está ocupado gerando a página requisitada até o envio a 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, e neste caso, o servidor já não tem interferência e não pode mais ajudar em nada.

Durante o tempo gasto entre o envio, análise e processamento de resposta, e 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 (Jiang, 2010).

A Figura 1 apresenta um exemplo de como uma página web é quebrada em *pagelets*. Cada quadrado de linha azul representa um *pagelet*, e estes são independentes entre si, pois enquanto um deles é mostrado na tela, outro ainda está sendo gerado pelo servidor web (Jiang, 2010).

Análises de performance foram realizadas, coletando dados ao carregar a página inicial do Facebook 50 vezes, utilizando navegadores com o cache limpo. A Figura 2

apresenta os tempos de latência, do modelo tradicional e com o BigPipe, obtidos através destes testes, onde nota-se que em um deles o ganho de desempenho chega a 2x, utilizando o BigPipe, em comparação ao carregamento tradicional (Jiang, 2010).

Modelagem e Implementação

Nesta seção são descritas as etapas de desenvolvimento adotados para a criação dos sites. As linguagens de programação utilizadas para o desenvolvimento da aplicação 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 3 apresenta a esquerda a página inicial do site e a direita a fragmentação da mesma.

Na Figura 3, pode-se destacar alguns dos fragmentos -, denominados *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 *scroll* e assim visualizar outras 6 notícias que estão ocultas.

Seguindo a ideia de fragmentação da página, chegou-se a um total de 13 *pagelets*, que, até então, seria o máximo de fragmentos que poderiam ser gerados para este site. É importante que 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. Nestes testes foram utilizados os três navegadores mais utilizados atualmente: o Google Chrome, o Internet Explorer e o Mozilla Firefox.

Os testes de utilização de CPU visam também, demonstrar se há ganhos de desempenho no carregamento de páginas 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, o segundo possui um Pentium Dual Core com dois núcleos e 4GB de memória RAM.

Experimentos e Discussões

Para os primeiros testes foi utilizada a ferramenta Pingdom Tool, uma página que executa testes de performance em páginas web informadas pelos usuários. Estes testes são executados em navegadores reais, em múltiplas instâncias do navegador Google Chrome, a fim de corresponder à experiência real do usuário.

A Figura 4, 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 5, 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, pode-se observar que na maioria dos testes, o site com a implementação do BigPipe obteve melhor desempenho no quesito velocidade de carregamento. Nestes testes com o BigPipe, os piores desempenhos ultrapassaram a linha dos 4 segundos 4 vezes, já no site normal, os piores desempenhos ultrapassaram esta linha 8 vezes. Ao verificar os melhores desempenhos, o site com BigPipe obteve resultados inferiores que 3 segundos em 8 testes e o site normal em apenas 3 destes testes. Em última análise dessas informações, ao realizar uma média geral dos testes executados entre os sites, nota-se que o ganho de performance não demonstra grande variação. Esta média é apresentada pela Figura 6, no qual pode-se verificar que o ganho obtido com a implementação do BigPipe é de apenas 0,176 segundos.

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, tem 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. No Figura 7, apresenta-se a maior utilização de CPU feita pelos navegadores no carregamento de ambos os sites, com e sem o BigPipe.

Na Figura 7, pode-se verificar 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 normal, 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 8, no navegador Firefox a utilização foi menor no site desenvolvido normalmente.

Analisando os resultados apresentados na Figura 8, 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 8 com os resultados demonstrados no Figura 7, obtemos então as Figuras **??** e 10 que demonstram suas variações.

Os resultados dos testes apresentados pelos gráficos 8 e 9 demonstram a utilização de memória neste primeiro computador. Ao realizar-se uma análise destes dois gráficos, pode-se identificar que a utilização de memória foi maior em todos os navegadores, nos quais o site com a implementação do BigPipe foi testado.

Dizer que a implementação do BigPipe no site é o gerador do aumento de utilização de memória, é de certa forma analisar a questão de forma errônea, pois os resultados obtidos nas Figuras 11 e 12 apenas demonstram a utilização de memória dos processos relativos a cada navegador. Contudo, nos resultados obtidos, a utilização de memória mostrou ser maior quando em carregamento o site com o BigPipe, isso pode estar ligado ao fato das requisições dos pagelets serem processadas após o retorno da resposta inicial dada pelo servidor.

Sendo 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 os *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 demonstrada nos gráficos.

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 13 e 14, 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.

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 15 e 16 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.

Conclusão

Esse trabalho 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.

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 nesse mesmo site, demonstrando que apesar do consumo de recursos ser elevado a resposta obtida é mais rápida.

Referências

Barrigas, H., Barrigas, D., Barata, M., Furtado, P. & Bernardino, J. (2014). Overview of facebook scalable architecture. Em *Proceedings of the international conference on information systems and design of communication* (pp. 173–176). ISDOC '14. Lisbon, Portugal: ACM. doi:10.1145/2618168.2618198

Begen, A., Akgul, T. & Baugher, M. (2011). Watching video over the web: part 1: streaming protocols. *Internet Computing, IEEE*, *15*(2), 54–63.

Cisco. (2013). Cisco visual networking index: forecast and methodology (2012-2017).

Jiang, C. (2010). Bigpipe: pipelining web pages for high performance.

Mai, H., Tang, S., King, S. T., Cascaval, C. & Montesinos, P. (2012). A case for parallelizing web pages. Em *Proceedings of the 4th usenix conference on hot topics in parallelism* (pp. 2–2). USENIX Association.

Figura 1. *Pagelets* na página inicial do facebook

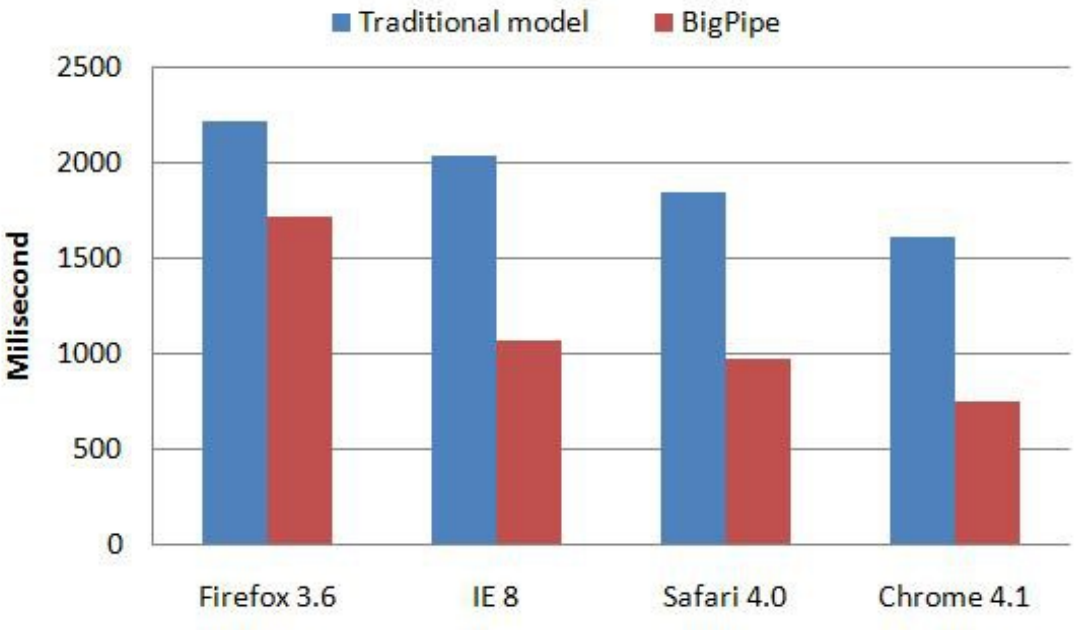

Figura 2. Tempo de execução da página inicial do facebook

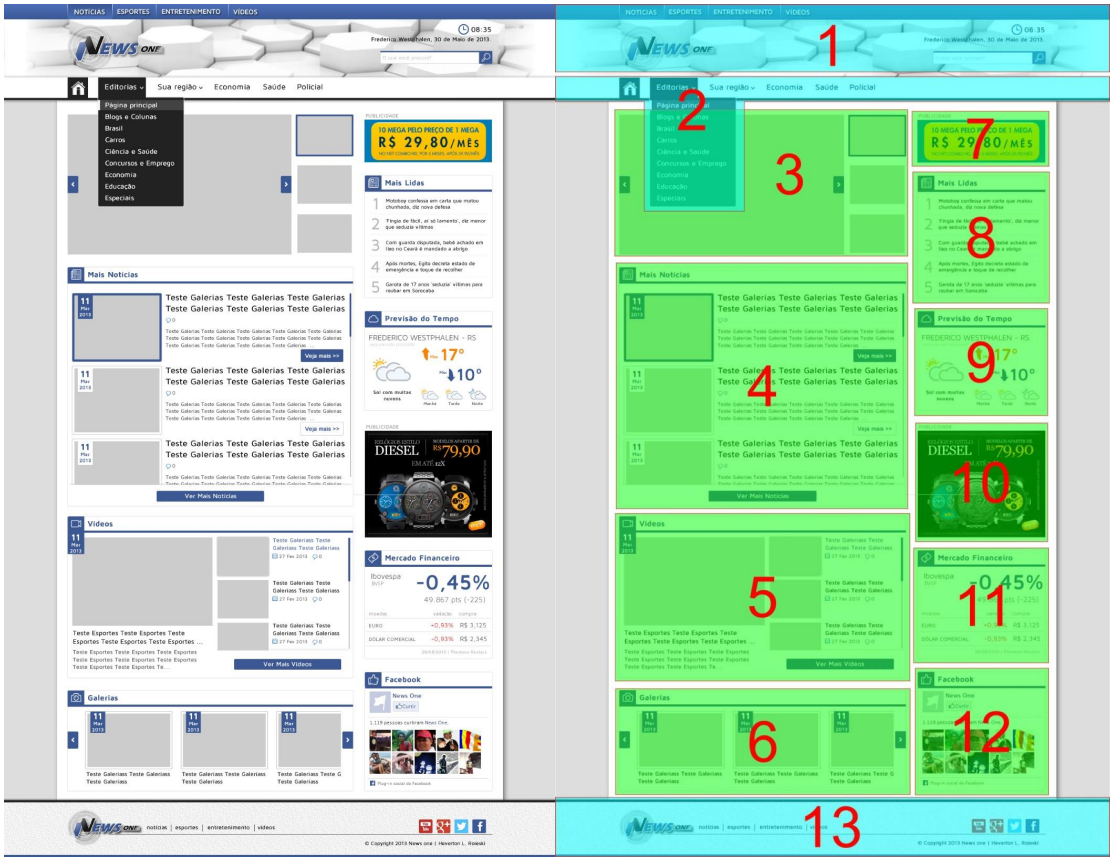

Figura 3. Página inicial do site desenvolvido e sua fragmentação

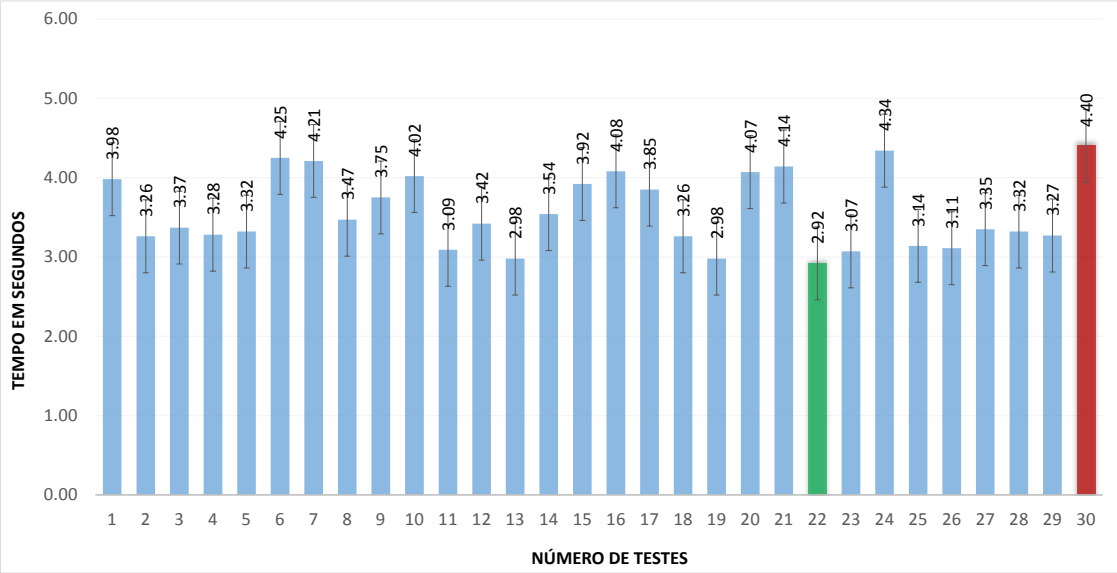

Figura 4. Tempo de execução da página de forma tradicional.

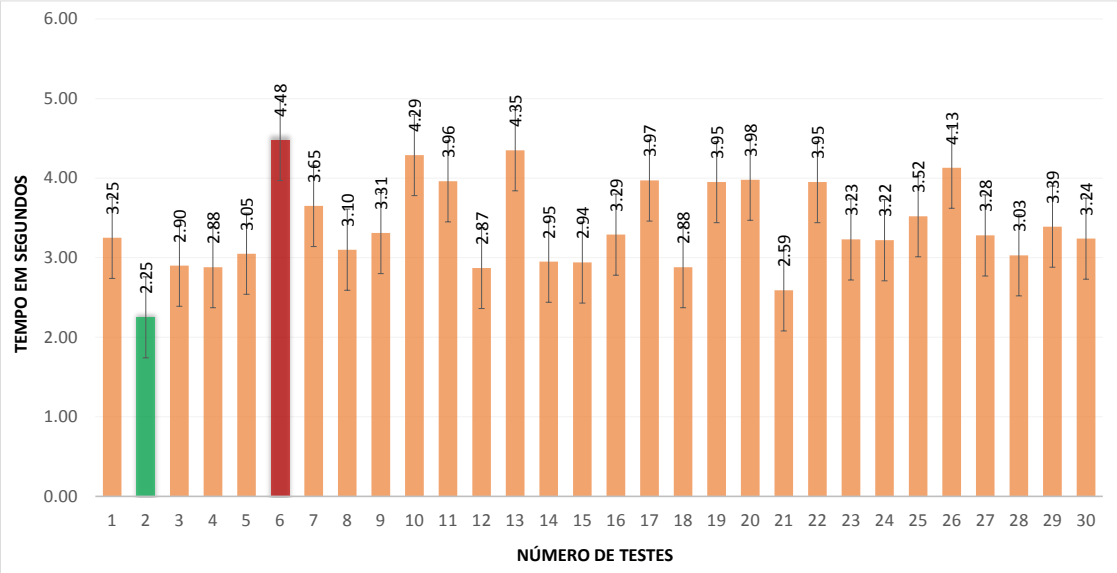

Figura 5. Tempo de execução da página utilizando BigPipe.

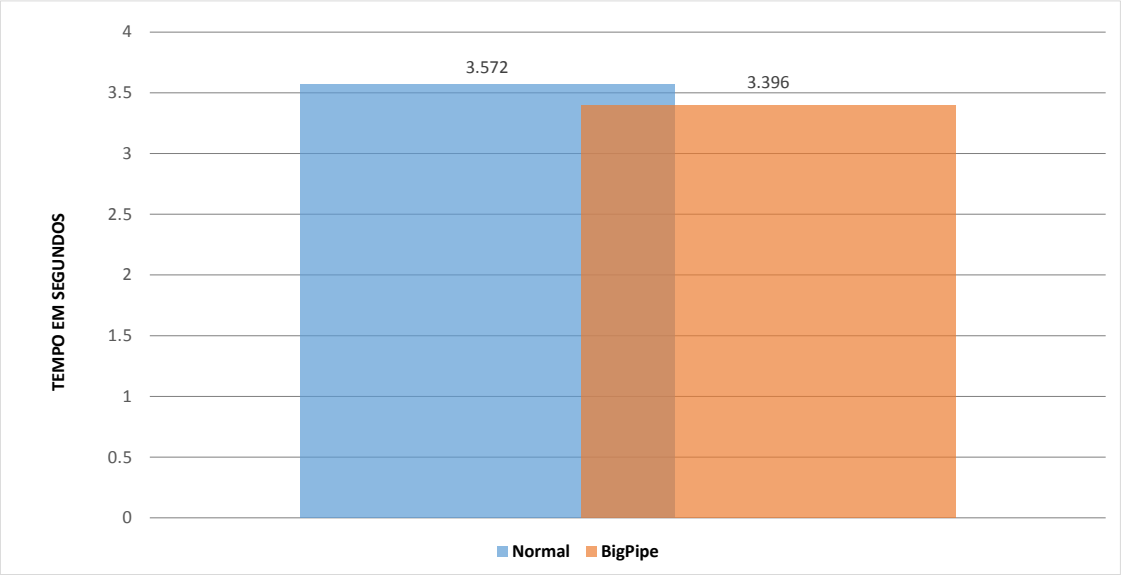

Figura 6. Comparação de tempo de execução da página tradicional vs página utilizando BigPipe.

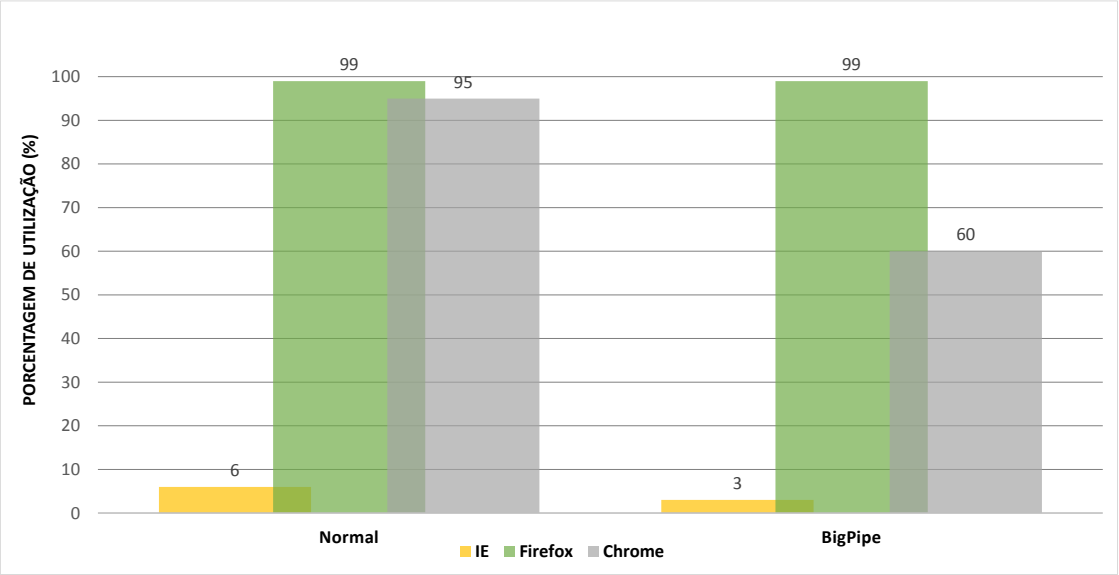

Figura 7. Maior utilização de CPU em um computador monoprocessado.

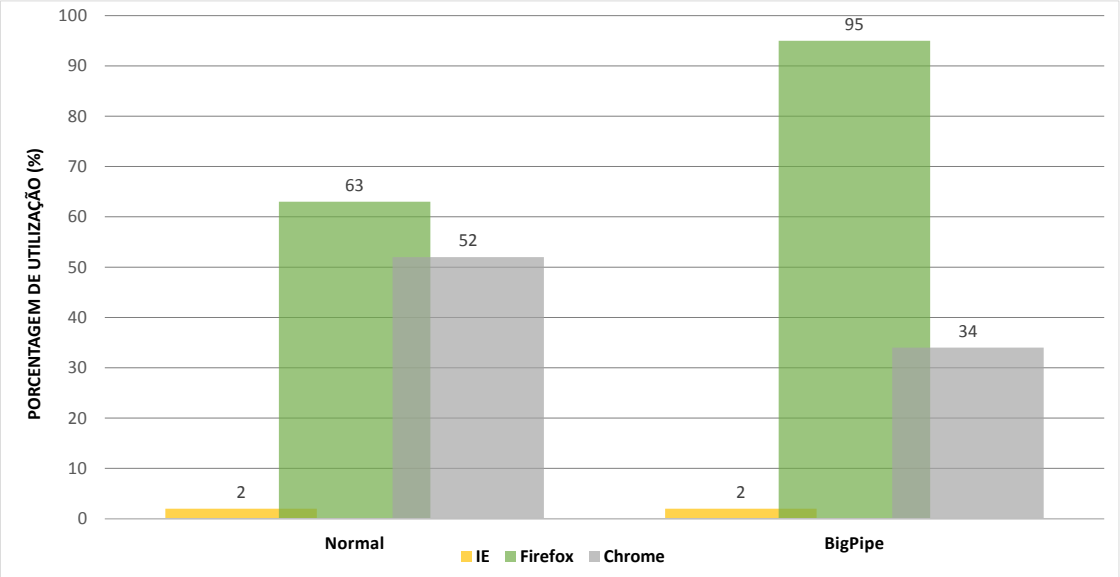

Figura 8. Menor utilização de CPU em um computador monoprocessado.

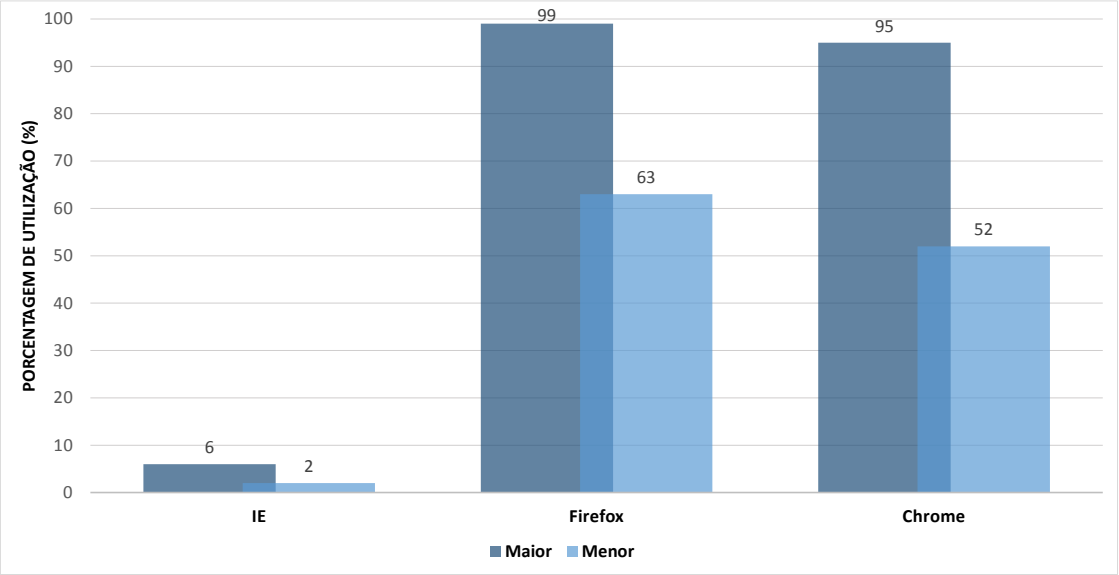

Figura 9. Comparativo de menor e maior tempo nas execuções monoprocessadas na forma normal.

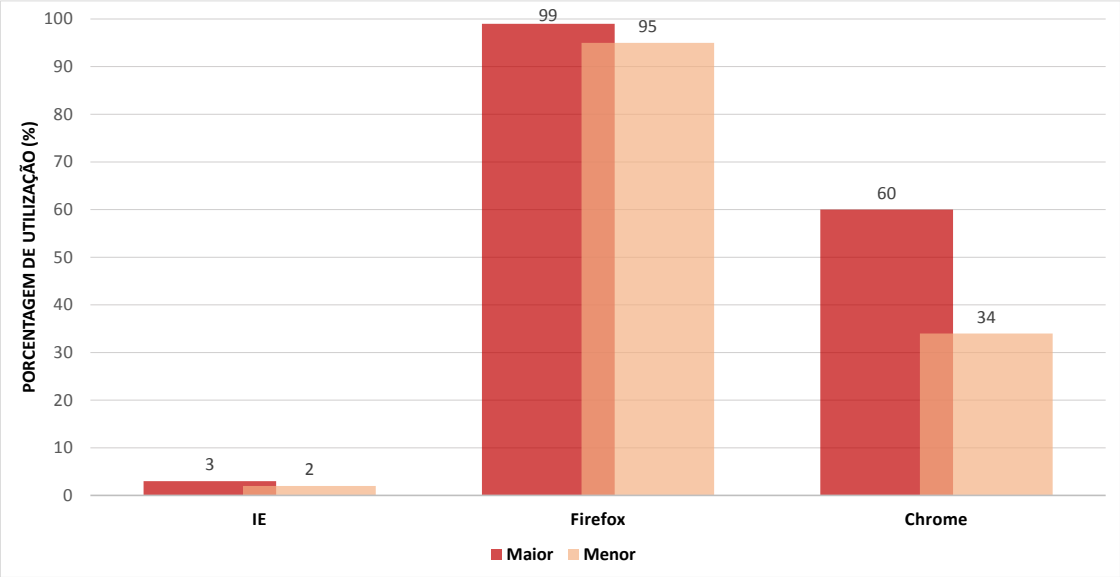

Figura 10. Comparativo de menor e maior tempo nas execuções monoprocessadas utilizando BigPipe.

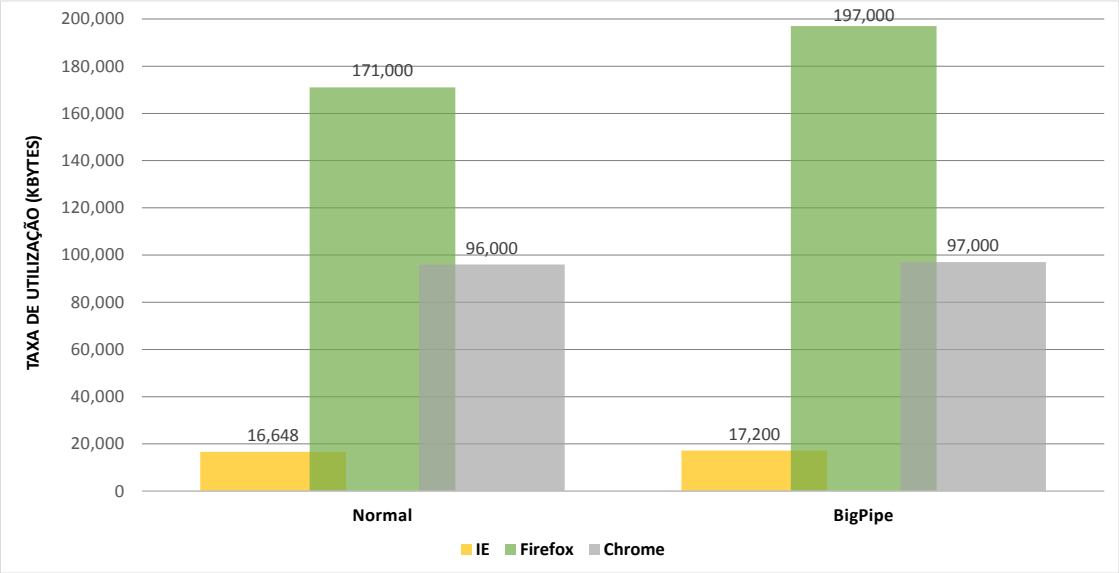

Figura 11. Maior utilização de memória em um computador monoprocessado.

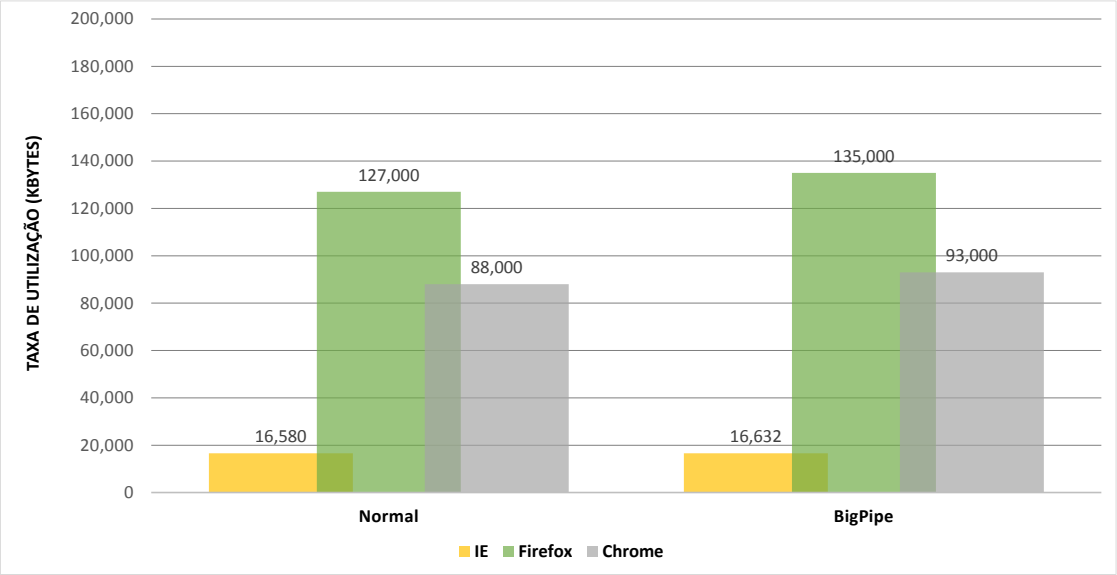

Figura 12. Menor utilização de memória em um computador monoprocessado.

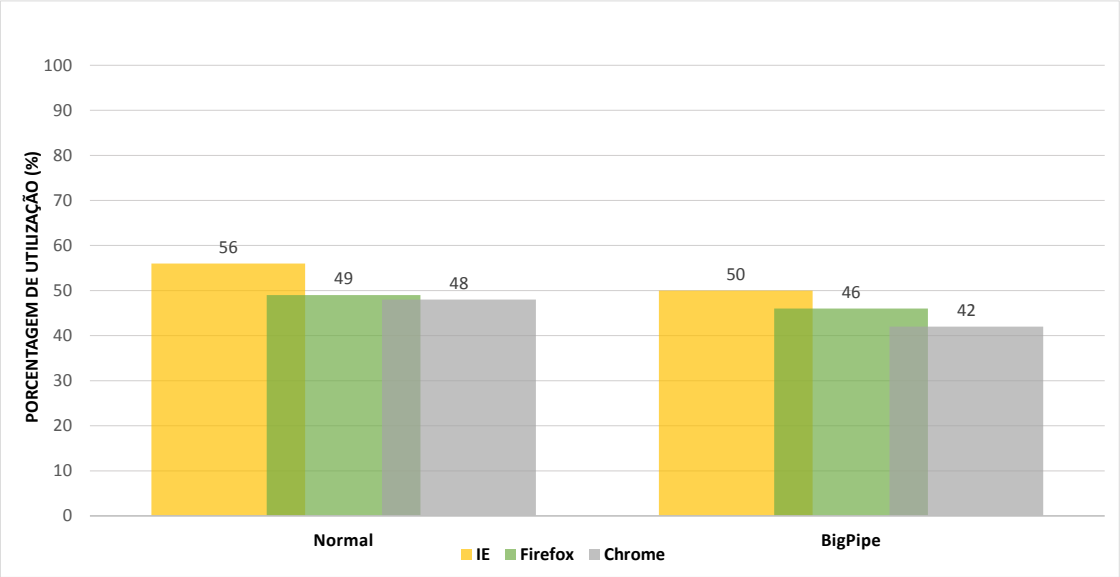

Figura 13. Maior utilização de CPU em um computador Dual Core.

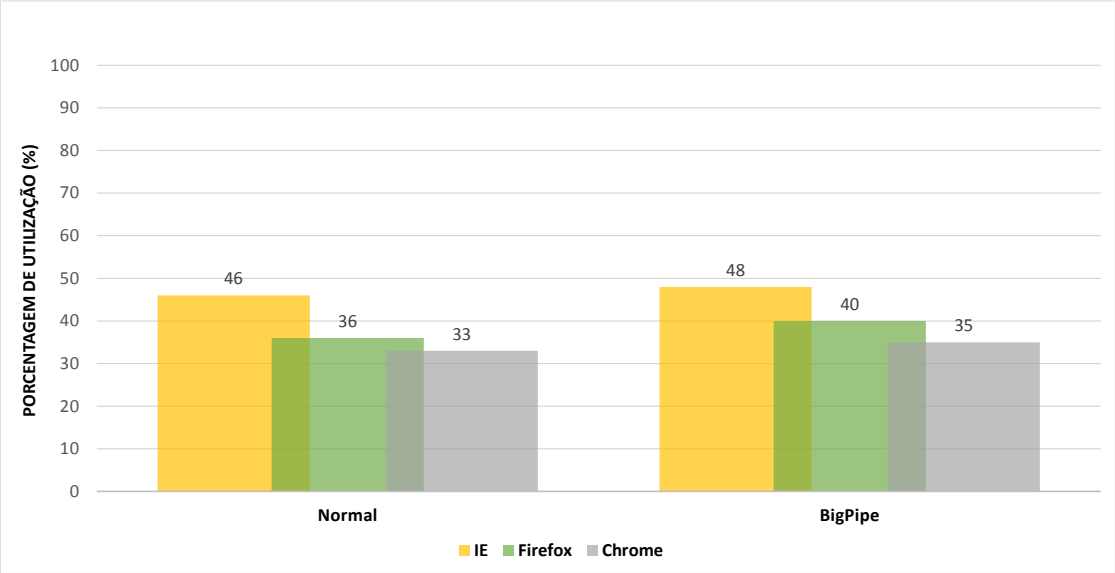

Figura 14. Menor utilização de CPU em um computador Dual Core.

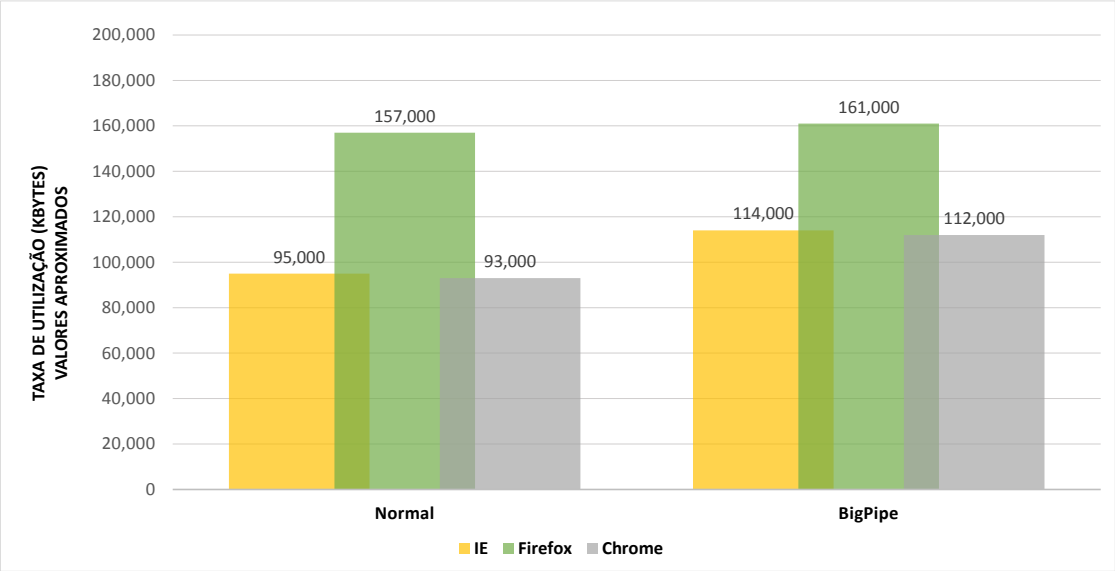

Figura 15. Maior utilização de memória em um computador Dual Core.

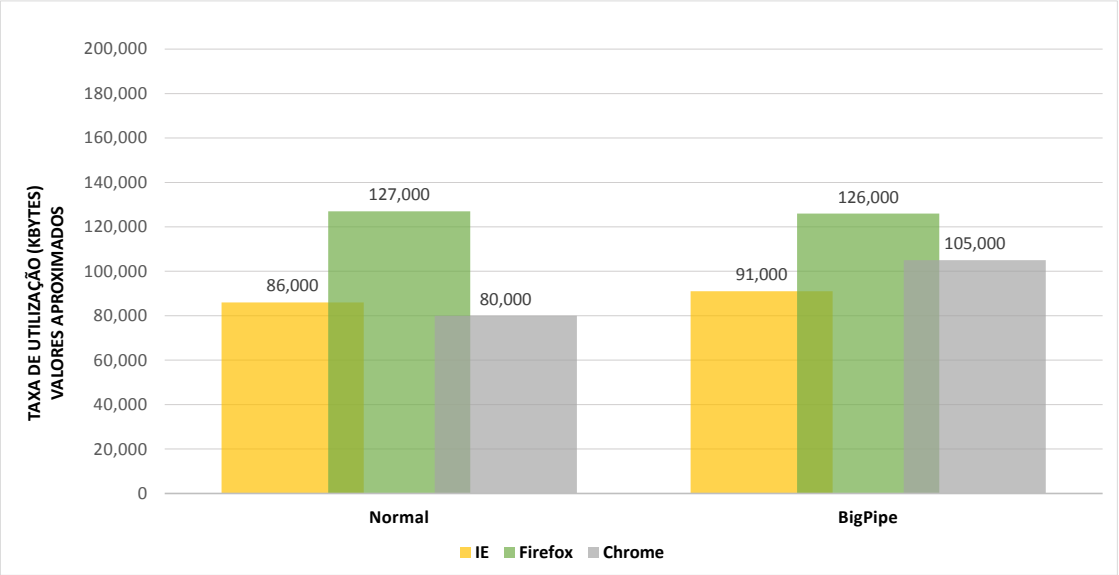

Figura 16. Menor utilização de memória em um computador Dual Core.