UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

Evolução e avaliação do *ViewIt,* uma ferramenta para automação de testes de desempenho de aplicações Web

Relatório Final do Projeto de Iniciação Científica

Bolsista: Marina de Souza Mendes - 15.1.5978
Orientadora: Prof. Dra. Amanda Sávio Nascimento e Silva

Ouro Preto – Minas Gerais – Brasil
Julho de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

Evolução e avaliação do *ViewIt,* uma ferramenta para automação de testes de desempenho de aplicações Web

Relatório Final do Projeto de Iniciação Científica

Bolsista: Marina de Souza Mendes - 15.1.5978
Orientadora: Profª. Drª. Amanda Sávio Nascimento e Silva

Relatório técnico referente ao segundo semestre do projeto de iniciação científica "Evolução e avaliação do *ViewIt*, uma ferramenta para automação de testes de desempenho de aplicações Web", apresentado à Universidade Federal de Ouro Preto, como parte das exigências do programa de iniciação científica desta instituição.

Ouro Preto – Minas Gerais – Brasil
Julho de 2017

Evolução e avaliação do *ViewIt*, uma ferramenta para automação de testes de desempenho de aplicações Web

A ferramenta aqui proposta passou por vários estágios e mudanças em seu desenvolvimento, sendo que em seu estado inicial precisou de melhorias como a implementação de uma interface amigável, sendo este um dos objetivos do projeto. Além disso, o projeto se propôs a estudar a importância da ferramenta para a automação de testes em aplicações *web.* Para tanto, foi feito um levantamento das métricas utilizadas para coletar dados de desempenho, de forma que fosse possível observar a relação dessas métricas com a qualidade do *ViewIt.* Contudo, várias dificuldades foram encontradas no levantamento científico dado que a ferramenta utilizada pelo *ViewIt* para a coleta e medição dos dados, o *Firebug*. Foram realizados diferentes tipos de testes, cada um com uma finalidade. Os testes foram feitos com a página principal da UFOP e o portal Minha Ufop. De acordo com os resultados apresentados, há latência na rede local da UFOP, portanto deveriam ser tomadas medidas que melhorem a infraestrutura da rede.

Lista de tabelas

Imagens

Índice

1 INTRODUÇÃO

Este relatório expõe o processo de desenvolvimento do projeto de iniciação científica do *ViewIt*, uma ferramenta proposta para a automação de testes em aplicações *web*. No estado inicial da ferramenta, os testes eram executados a partir de arquivos XML[1], o que dificultava seu uso. Uma interface gráfica amigável foi proposta e implementada como solução para esta dificuldade, o que facilita o uso da ferramenta e permite um maior aprofundamento na compreensão de automação de testes em aplicações *web*.

O objetivo do projeto é estudar a importância e a qualidade do *ViewIt,* para isto seu desenvolvimento foi dividido em levantamento de estudos sobre as métricas da coleta dos dados e no desenvolvimento de uma interface gráfica amigável. Porém, foram encontradas dificuldades no levantamento de pesquisas científicas sobre métricas usadas em coleta de dados, o que talvez possa ser explicado pelo fato de as métricas serem usadas em ferramentas que geralmente são usadas fora da academia. A interface gráfica está pronta para ser testada por usuários para receber um retorno deles e a partir disso ser ainda mais aperfeiçoada.

Apesar das dificuldades encontradas, pode-se admitir que uma ferramenta gratuita e *open-source* como o *ViewIt,* que possa ser usada de forma relativamente fácil por usuários da área, possui a vantagem de poder ser utilizada em sala de aula sem nenhum custo para a instituição. Esta é uma ideia que se aplicaria a alunos da área de tecnologia, em especial em disciplinas relacionadas ao desempenho ou ao desenvolvimento de aplicações *web.*

Como a maioria dos protótipos, o *ViewIt* ainda pode ser melhorado, mas seu estágio atual já é suficiente para a realização de testes reais com *stakeholders* e, além disso, é possível que exista ainda outras utilidades para ele além da automação de testes. Isto porque as ferramentas utilizadas por ele podem ser utilizadas de outras formas, por exemplo, o *Selenium Webdriver,* responsável pela automação das ações do teste (como abrir a página, clicar em algum botão, digitar algo em algum campo etc), pode ser usado para a mineração de dados. O que diferencia uma funcionalidade de outra é a forma como os dados são coletados e quais dados são coletados.

O escopo do atual projeto é a automação de testes de desempenho em aplicações *web* e, portanto, os dados de interesse são relacionados ao desempenho de uma aplicação do ponto de vista do cliente. A coleta de dados no *ViewIt* é feita pelo *Firebug* (extensão do navegador

[1]Extensible Markup Language

Mozilla Firefox) e se refere aos dados de desempenho, como o tempo de carregamento do HTML, das imagens e dos *scripts* da página.

2 OBJETIVOS

2.1 Objetivo Geral

O objetivo geral do projeto é melhorar a ferramenta proposta e compará-la com outras ferramentas para que seja mais fácil de notar as características do *ViewIt,* como as suas vantagens e desvantagens em relação às outras ferramentas. Além disso, com acesso a um estudo comparativo e metódico é possível alcançar conclusões mais precisas.

2.2 Objetivos específicos

Objetivos específicos para a evolução da ferramenta:

- ➔ Transição do uso da ferramenta por *XML* para seu uso a partir de uma interface gráfica amigável

3 REVISÃO DE LITERATURA

Dada a importância crescente da internet, torna-se necessário que aplicações Web atendam, além dos requisitos funcionais, requisitos de qualidade a fim de evitar a insatisfação de seus usuários, e, até mesmo, prejuízos financeiros.

Para tanto, o W3C criou uma recomendação chamada *Navigation Timing API*, que é implementada por alguns navegadores com a finalidade de obter medidas detalhadas de modo nativo sobre as etapas de processamento. As ferramentas para automação de testes em sua maioria coletam apenas o tempo total de processamento de uma requisição a partir da perspectiva do servidor.

Sendo assim, há uma lacuna por soluções para automação de testes que permitam analisar de forma detalhada o tempo gasto em cada fase de processamento de uma requisição para uma aplicação Web e em intervalos pré-definidos durante um período de tempo. A ferramenta proposta, por utilizar o *Firebug*, é capaz de coletar o tempo dispendido em cada etapa do carregamento da aplicação.

3.1 Firebug

O Firebug é uma extensão para o Mozilla Firefox que adiciona ao navegador diversas ferramentas para facilitar a tarefa de desenvolvimento de paginas Web (Firegug, 2014). De maneira mais relevante para esse trabalho, a extensão NetPanel do Firebug permite ao Desenvolvedor monitorar todo o tráfego iniciado a partir de uma requisição a uma página Web, coletando informações de tempo de processamento da requisição corrente bem como informações sobre o tamanho em bytes do arquivos carregados (HTML, CSS, imagens etc).

As informações coletadas pelo NetPanel são agrupadas de acordo com a etapa de carregamento da página. Na implementação do *ViewIt*, não foi considerado que algumas etapas são assíncronas e suas medições foram adicionadas ao somatório mesmo que ocorram ao mesmo tempo. Isto porque o que importa é o valor total das medições de cada etapa, pois quanto menor for melhor será o desempenho da página. Essas etapas são descritas na documentação do Firebug e NetPanel da seguinte forma:

Métrica	Descrição
Blocking	Tempo gasto na fila do navegador aguardando conexão de rede (Queueing). Para conexões SSL isso inclui Handshake SSL e o passo de validação OCSP
DNS Lookup	Tempo de resolução do DNS
Connecting	Tempo gasto para criar uma conexão TCP
Sending	Envio de cabeçalhos de requisição
Waiting	Aguarda resposta do servidor
Receiving	Tempo para ler a resposta completa do servidor ou do cache
'DOMContentLoaded' (event)	Quando o evento DOMContentLoaded foi disparado (desde o início da requisição, pode ser negativo se a requisição tiver começado após o evento)
'load' (event)	Quando o evento load foi disparado (desde o início da requisição, pode ser negativo se a requisição tiver começado após o evento)

1 – Tabela: Métricas NetPanel/Firebug(adaptado de https://getfirebug.com/wiki/index.php/Net_Panel)

3.2 Selenium Webdriver

O Selenium WebDriver(Webdriver, 2014a) é um dos frameworks mais populares de testes funcionais em sistemas Web (Armeli-Battana, 2012). Ele possui diversas funções para a implementação dos cenários de teste.

As principais funcionalidades para a criação de testes pelo Selenium Webdriver são descritas a seguir:

- ➔ Navigate: abre uma página a partir de seu endereço (URL).
- ➔ Click/clickAndWait: executa a ação d clique (clique em botão, link etc)
- ➔ Submit: permite a submissão de um formulário.
- ➔ Check Box: permite selecionar uma ou mais opções de uma lista de opções.
- ➔ Type: permite digitar um valor em um determinado campo da página.

3.3 HarLib e JFreeChart

O HarLib é uma biblioteca Java cuja API apoia a leitura e escrita de arquivos .har que contém as informações de desempenho coletadas a partir do NetPanel do Firebug. O JFreeChart é um framework Open Source, baseado na linguagem de programação Java, que permite a criação de uma grande variedade de gráficos.

3.4 Trabalhos e Ferramentas Relacionadas

O Apache *JMeter* é um aplicativo desktop desenvolvido na linguagem de programação Java para desenvolvimento e execução de testes de desempenho e estresse em aplições Web. Ele permite a criação de testes de desempenho para diferentes tipos de protocolos e serviços (HTTP, HTTPS, SOAP, FTP e TCP).

Em comparação com o *ViewIt,* o *JMeter* não funciona como um navegador, pois não executa scripts nem renderiza o código HTML, fazendo apenas uma emulação do carregamento das páginas. Por conta na disso, na seção com o comparativo entre as duas ferramentas, o *JMeter* apontou para um melhor desempenho da mesma página testada pelo *ViewIt*, mas na verdade a forma como o *JMeter* efetua o teste não simula a execução real por parte do cliente. O *ViewIt* ao executar os cenários de testes no navegador Web, os executa no mesmo ambientee sob as mesmas condições do que o usuário final. No entanto, é possível adicionar ao *JMeter* o plugin do Selenium WebDriver (JMeter-plugins.org, 2015) para que seus testes sejam executados a partir do navegador Web.

Outra limitação do *JMeter* em relação ao *ViewIt* é que o *JMeter* coleta apenas o tempo total de processamento de uma requisição enquanto o *ViewIt* coleta o tempo total de cada etapa do carregamento (de *blocking*, *waiting*, *receiving* etc).

4 METODOLOGIA

Durante o desenvolvimento do projeto foi necessário o entendimento de como a ferramenta funcionava com o uso do XML. Alguns testes foram realizados com o XML para que fossem compreendidas quais informações eram necessárias em cada cenário de teste.

Além disso, o fluxo do programa deve ser entendido para que se possa reproduzi-lo para uma versão com interface gráfica. Considerando então o fluxo para a realização dos testes e a geração de gráficos, a interface gráfica foi implementada. Apareceram muitas dificuldades relacionadas à integração de um novo código voltado para a interface gráfica com um código já existente e sem suporte para interface gráfica. Por se tratar de um projeto relativamente grande, aspectos de engenharia de software devem ser revisados no código da interface gráfica, de forma a refatorá-lo e melhor adaptá-lo ao código já existente.

Com relação aos testes gerados pelo *ViewIt*, após a implementação da interface gráfica eles continuam funcionando da mesma forma que funcionavam com XML (e como deveriam), mas com maior facilidade de usabilidade.

Foram efetuados três cenários de testes:

➔ Na rede local da UFOP:

URL: http://www.ufop.br

Execuções: 30

Intervalo: 1 min

URL: http://www.minha.ufop.br

Execuções: 30

Intervalo: 1 min

➔ Em rede remota:

URL: http://www.ufop.br

Execuções: 30

Intervalo: 1 min

URL: http://www.minha.ufop.br

Execuções: 30

Intervalo: 1 min

➔ Em rede remota com *JMeter* e *ViewIt* simultaneamente:

URL: http://www.ufop.br

Execuções: 30

Intervalo: 1 min

Na subseção seguinte é demonstrado o passo-a-passo para utilizar a ferramenta.

4.1 Instalação e Configuração do *ViewIt*

Para que seja possível utilizar a ferramenta proposta, é preciso fazer o download o clonar o projeto, disponível no link: https://github.com/marinamsm/ViewIt.git

Além do download ou clone do projeto, deve-se ter instalada a versão 47 do navegador Mozilla Firefox, que pode ser encontrado no link: https://ftp.mozilla.org/pub/firefox/releases/47.0/

Assim que esta versão for instalada deve-se impedir sua atualização, dado que a versão mais recente do navegador não suporta os plug-ins da ferramenta. Para tanto, abra o navegador, clique no ícone de listagem (opções), no canto superior direito, para abrir as opções do navegador. Em seguida, clique em Opções, então em Avançado no menu à esquerda da página que abrir. Nas abas que aparecerem dentro de Avançado, clique em Atualizações e desabilite a instalação automática de atualizações.

Após as instalações, deve-se abrir o projeto, em especial a classe PerformanceTesting.java pelo caminho ViewItGUIRepo\GUIViewIt\src\br\ufop\performance

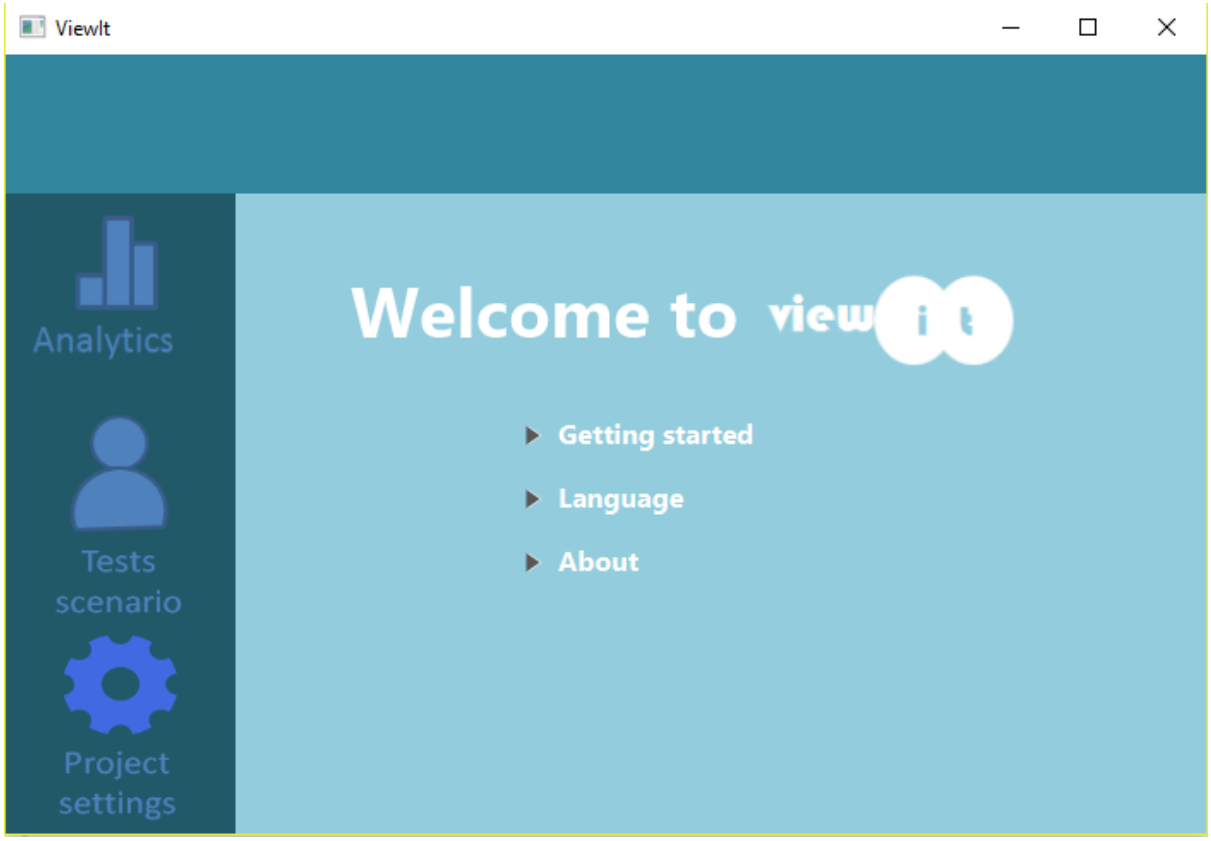
Na segunda linha de código (sem considerar comentários) do método *run*, temos o seguinte:

```
FirefoxBinary binary = new FirefoxBinary(new File("C:\\Program Files\\Mozilla Firefox\\firefox.exe"));
```

Caso o navegador tenha sido instalado em outro local, o caminho deve ser especificado no trecho entre aspas.

4.2 Execução simples de um teste

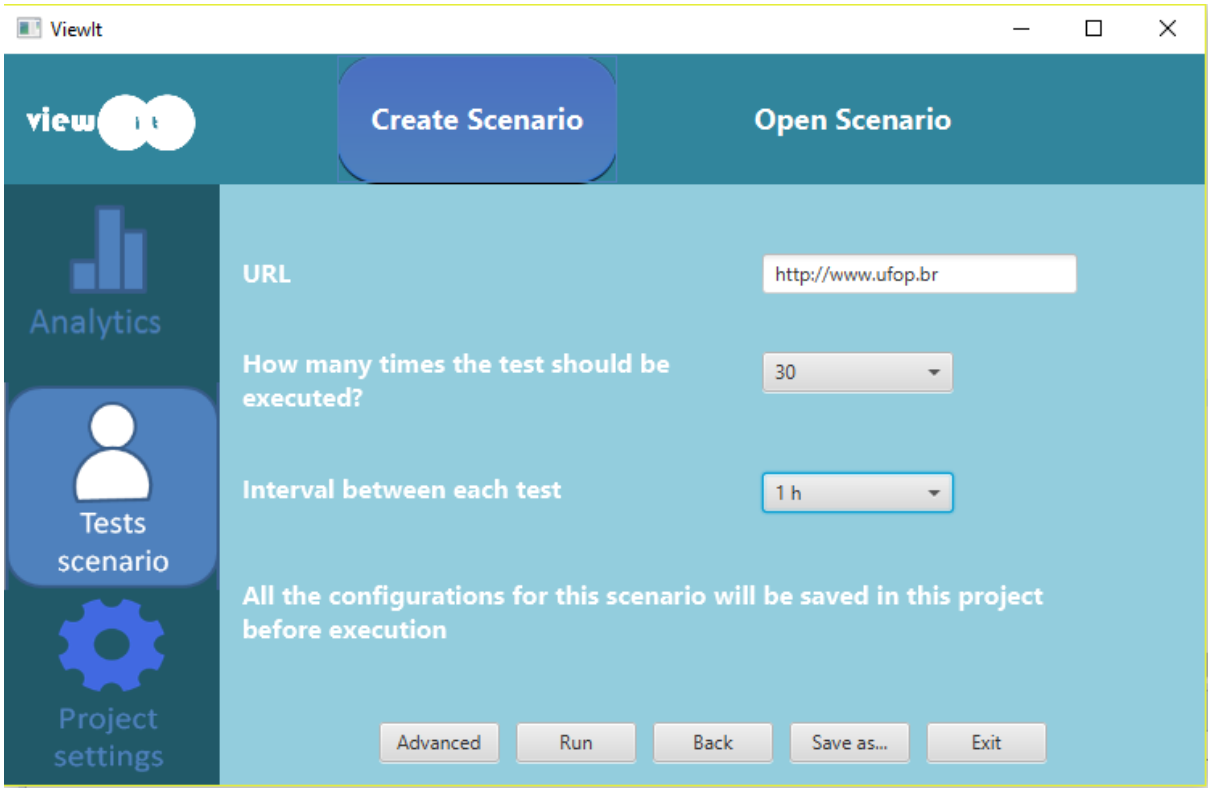
Com as configurações acima feitas, a ferramenta está pronta para execução. A classe Main.java, em ViewItGUIRepo\GUIViewIt\src\br\ufop, já pode ser executada. A tela inicial pode ser vista abaixo:



1 – Figura: Tela inicial do ViewIt

A opção *Getting Started* irá ajudar o usuário através da documentação do código e com exemplos de uso. *Language* irá possuir tanto a opção em inglês quanto em português, apesar de o *ViewIt* ter sido feito a princípio em inglês. As opções ainda estão em desenvolvimento.

Para realizar um teste simples basta clicar em *Tests scenario* e depois em *Create Test* na parte superior da tela:



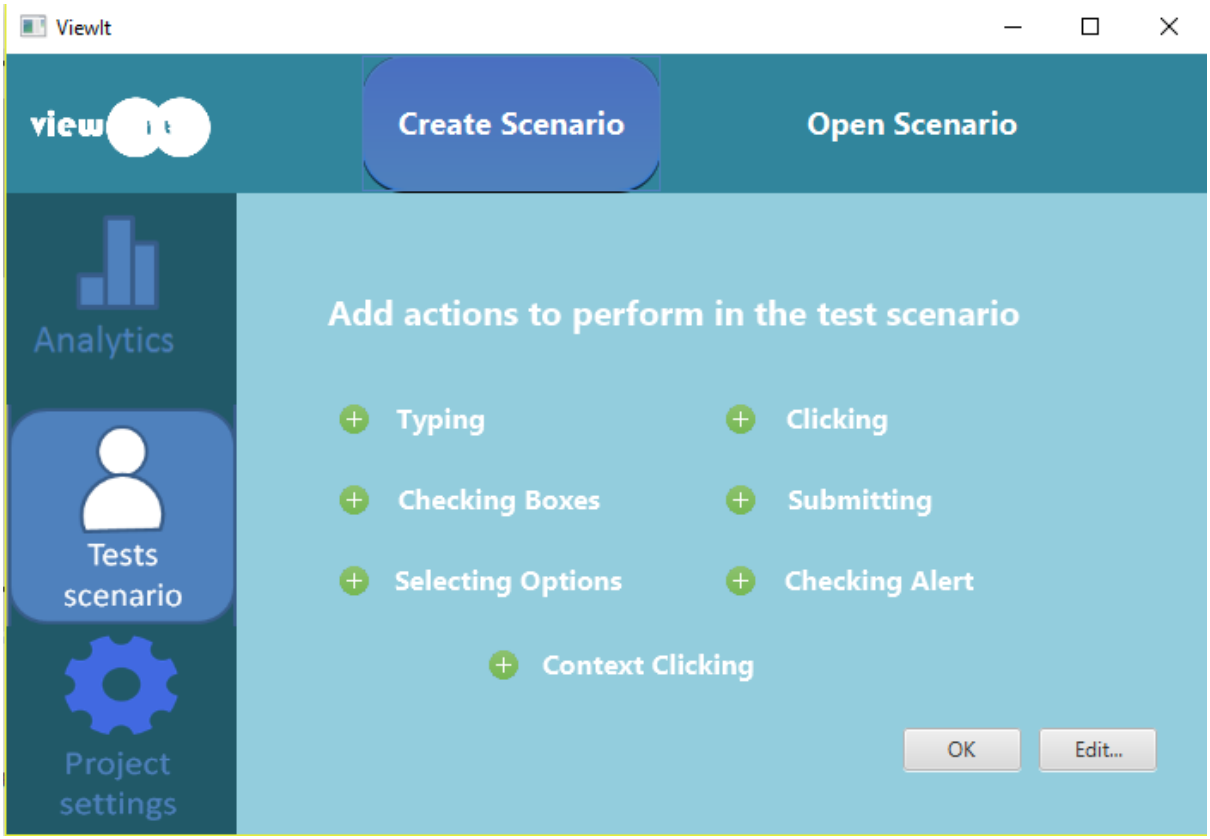
2 – Figura: Tela inicial para criar cenário de teste no *ViewIt*

No campo *URL* deve-se inserir o endereço do site junto com o protocolo (ex: http, https), no segundo campo deve-se selecionar a quantidade de vezes que o teste será executado e no último campo escolhe-se o intervalo entre cada teste. Há ainda opções avançadas (no botão *Advanced)* de ações a serem executadas durante a navegação no site, como cliques, seleção de opção e digitação de texto. Se nenhuma opção avançada for escolhida, o teste apenas abrirá o site em questão de acordo com a quantidade de vezes e intervalo especificados. Neste exemplo, o site da ufop será carregado 30 vezes com intervalo de 1 hora entre cada execução, ou seja, a execução irá durar 30 horas no total.

Caso o usuário deseje salvar o cenário de teste para uso posterior, ele pode clicar em *Save as...* e escolher onde salvar o arquivo (o arquivo é salvo no formato XML). A partir do momento que o arquivo é salvo, tudo que for gerado como resultado dos testes será salvo na mesma pasta onde do XML. Se o usuário não salvar o cenário de teste, os resultados serão salvos na mesma pasta do projeto do *ViewIt* e o cenário de teste não será armazenado, apenas seus resultados. Há ainda a opção de abrir cenário já existente em *Open Scenario* caso o XML já exista. Por fim, ao clicar em *Run* o teste será executado.

4.3 Opções avançadas de teste

Na tela de teste, ao clicar em *Advanced*, as seguintes opções de caso de teste irão aparecer:



3 – Figura: Tela de opções avançadas para o cenário de teste no *ViewIt*

As opções disponíveis são:

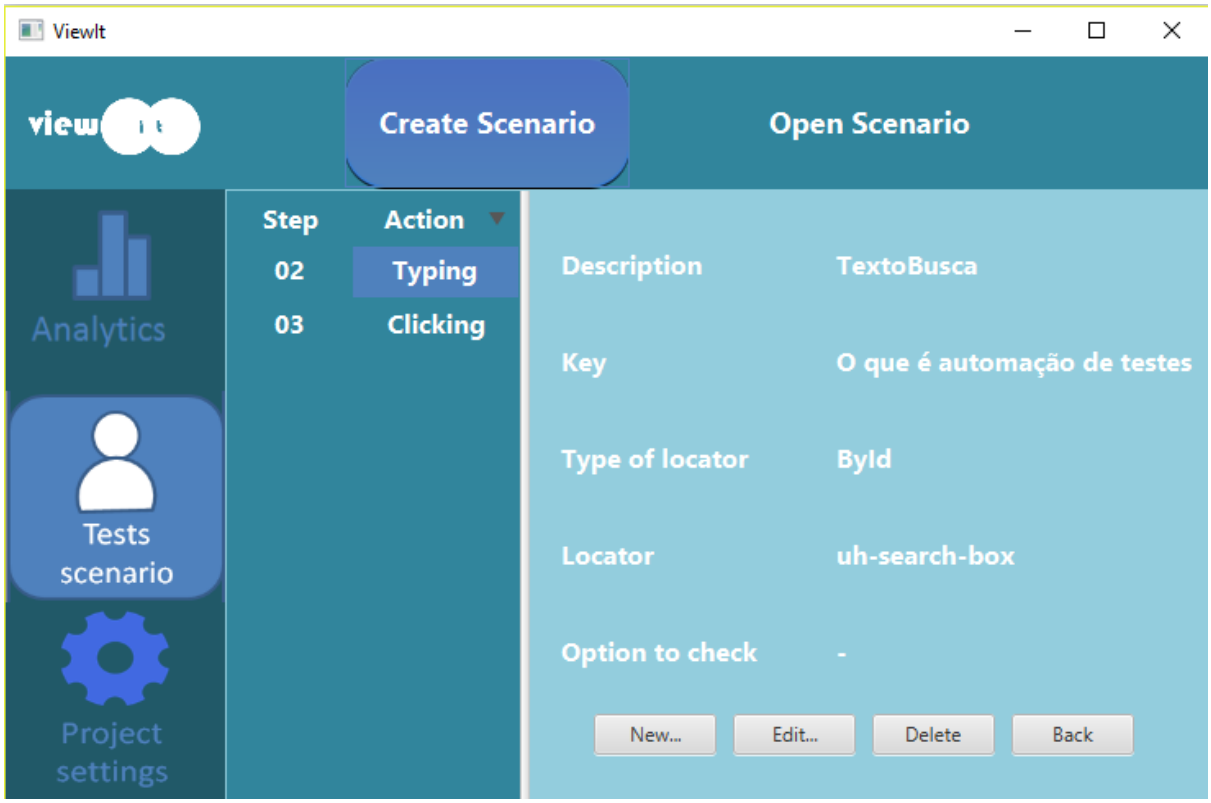
- ➔ *Typing*: com esta opção é possível que durante a execução do teste algo seja digitado em algum campo de texto na página a ser testada.
- ➔ *Clicking*: opção de clicar em algum elemento da página.
- ➔ *Checking Boxes*: seleciona uma ou múltiplas *checkbox.*
- ➔ *Submitting*: submete formulário.

➔ *Selecting Options*: seleciona opção de um conjunto de opções.

➔ *Checking Alert*: confirma/Recusa mensagem de aviso.

➔ *Context Clicking*: clique com botão direito para exibir menu.

O conjunto de todas as opções adicionadas pode ser consultado e editado ao clicar em *Edit...* :



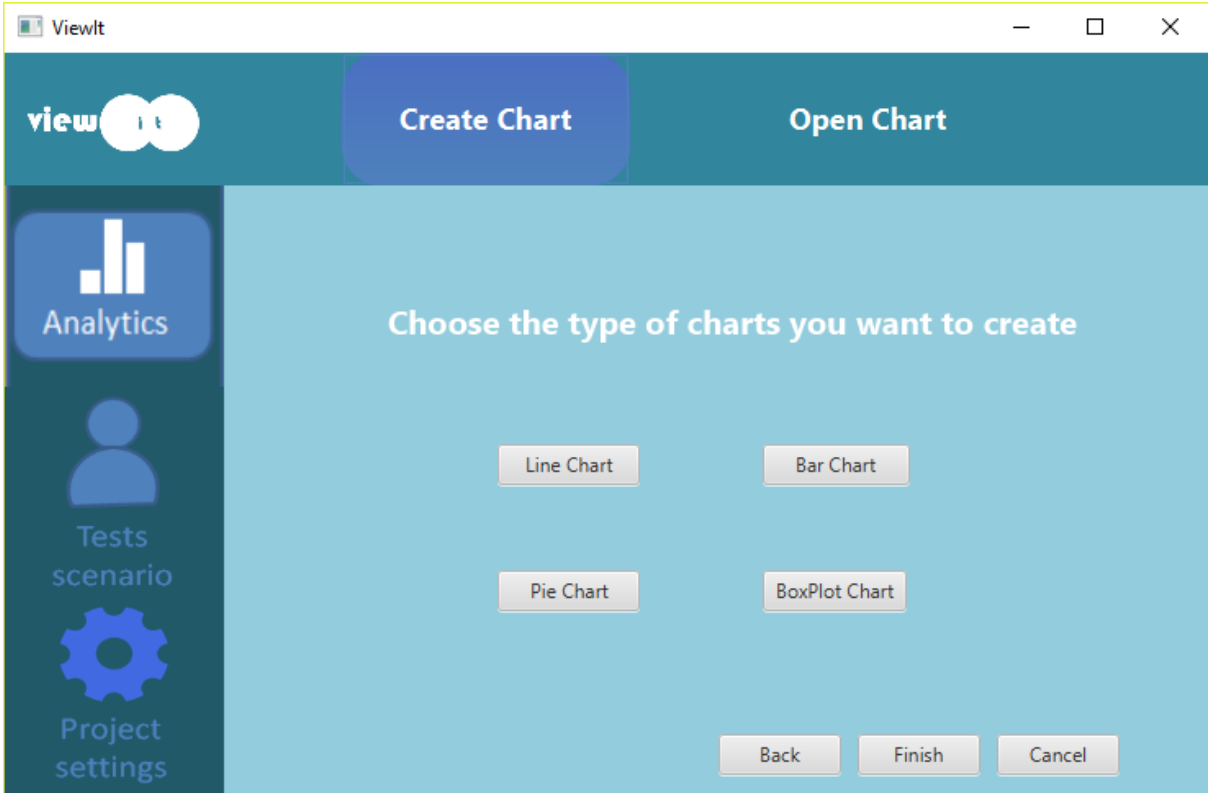
4 – Figura: Tela de consulta e edição de opções avançadas do cenário de teste do ViewIt

Do lado esquerdo está o nome do caso de teste e a ordem de execução, começando do 2, pois o primeiro caso é sempre o *Navigating* (responsável por abrir a página inicial). Ao clicar em uma das ações, seus detalhes irão aparecer no lado direito da tela junto com opções de cadastrar nova ação, editar a ação selecionada ou apagá-la.

4.4 Gerador de gráficos

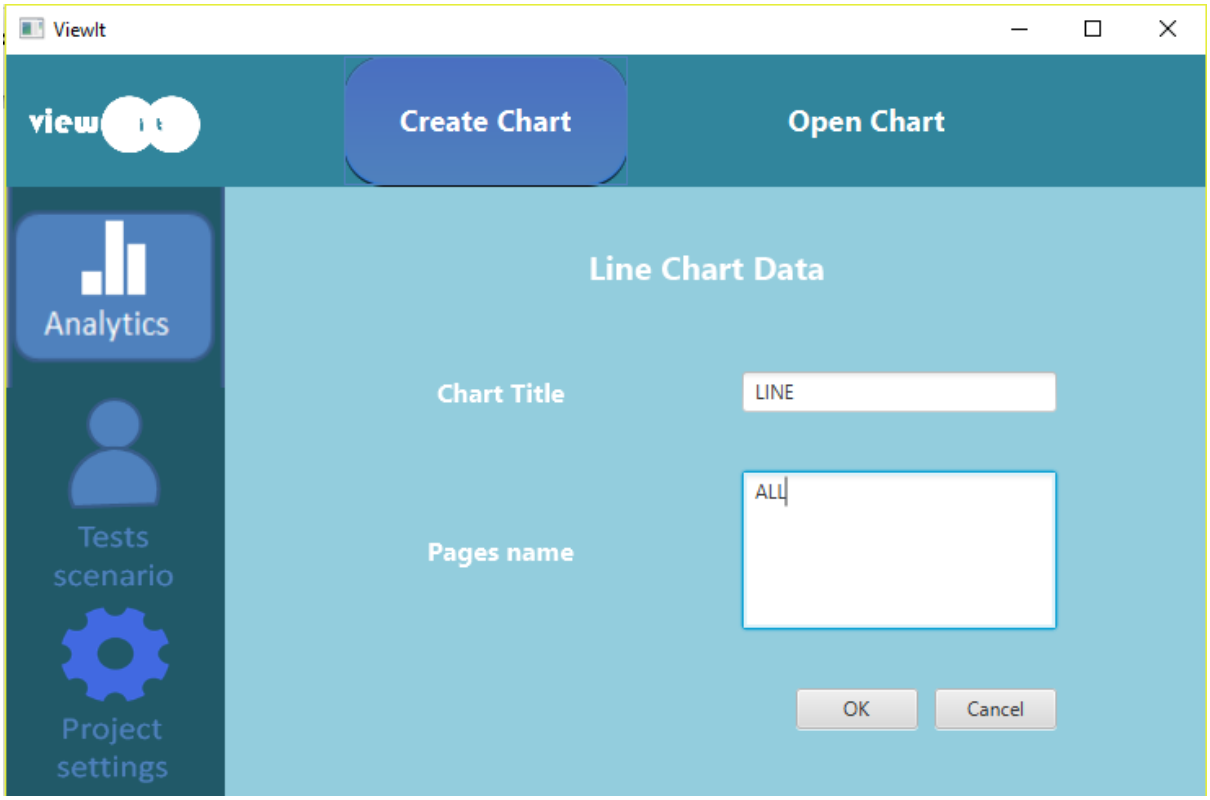
O gerador de gráficos é útil para que os dados coletados na fase de testes possam ser representados por meio de um gráfico. O gerador foi implementado de forma separada do módulo de execução de testes, portanto, fica a critério do usuário escolher o tipo de gráfico a ser gerado e a partir de qual arquivo CSV. Isto porque cada teste gera seu conjunto de arquivos HAR com os resultados e esses arquivos são então convertidos para a extensão .csv. Com esta extensão é possível gerar gráficos com as informações contidas no arquivo .csv.

Há 4 tipos de gráficos, o gráfico linha, pizza, barra e *boxplot.* A tela com as opções de gráficos é a seguinte:



5 – Figura: Tela inicial para gerar gráficos no *ViewIt*

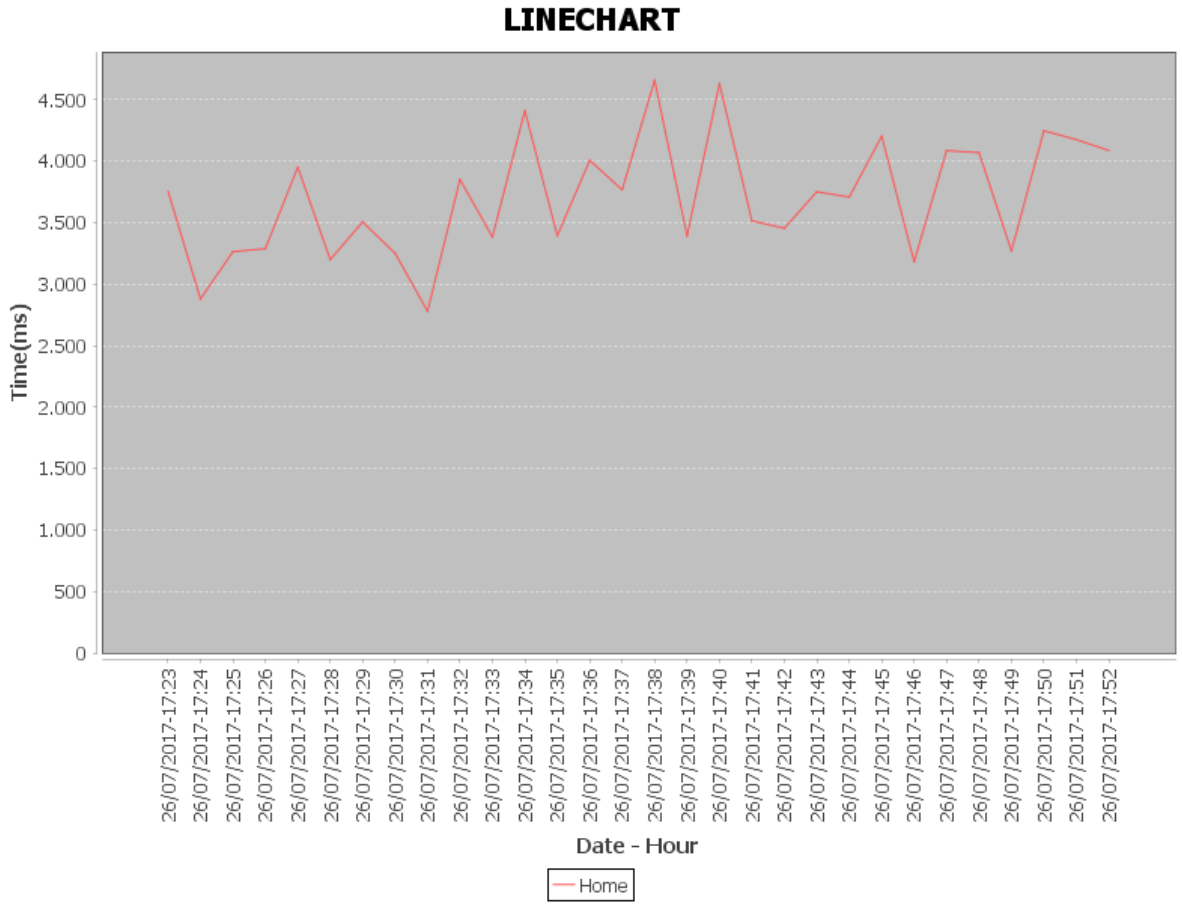
Para acessar a tela acima deve-se clicar em *Analytics* no menu à esquerda e depois em *Create Chart* no menu superior. A tela do gráfico linha pode ser vista em seguida:



6 – Figura: Exemplo – Tela do gráfico linha no *ViewIt*

No caso do gráfico linha é pedido apenas o título a ser dado para identificar o gráfico e o nome das páginas cujos dados devem constar no gráfico. Se for escrito *ALL* então o gráfico será gerado com os dados de todas as páginas que constarem no arquivo .csv.

Exemplo de gráfico linha:



7 – Figura: Exemplo de gráfico linha gerado pelo *JFreeChart* no *ViewIt*

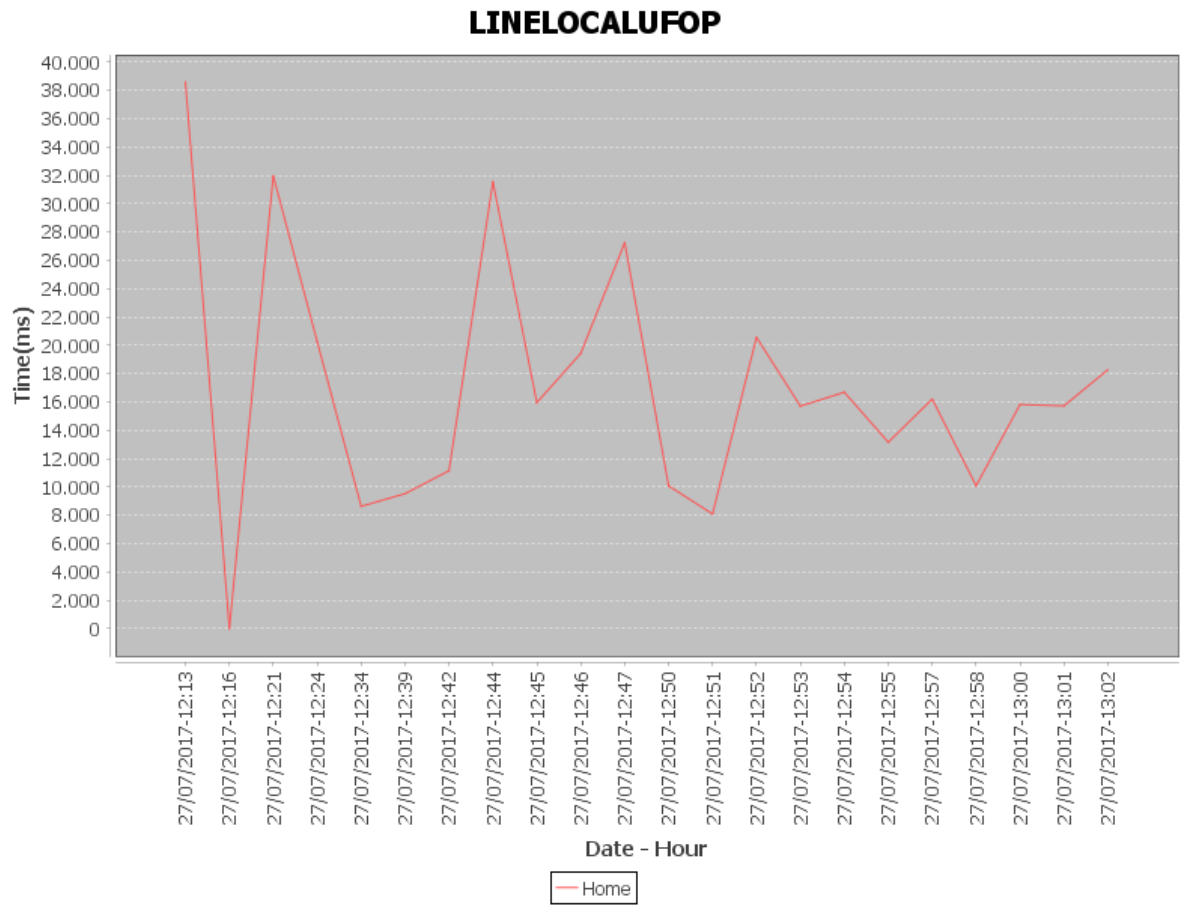
O eixo horizontal refere-se à data do teste e o eixo vertical ao tempo de carregamento da página (desde o início da requisição até a entrega do documento para o usuário) em milissegundos. Mais exemplos de gráficos serão encontrados na próxima seção, pois todos os gráficos com resultados dos testes do *ViewIt* foram gerados por ele próprio (pela ferramenta *JFreeChart*).

5 TESTES E RESULTADOS

5.1 Requisição local à página inicial da UFOP

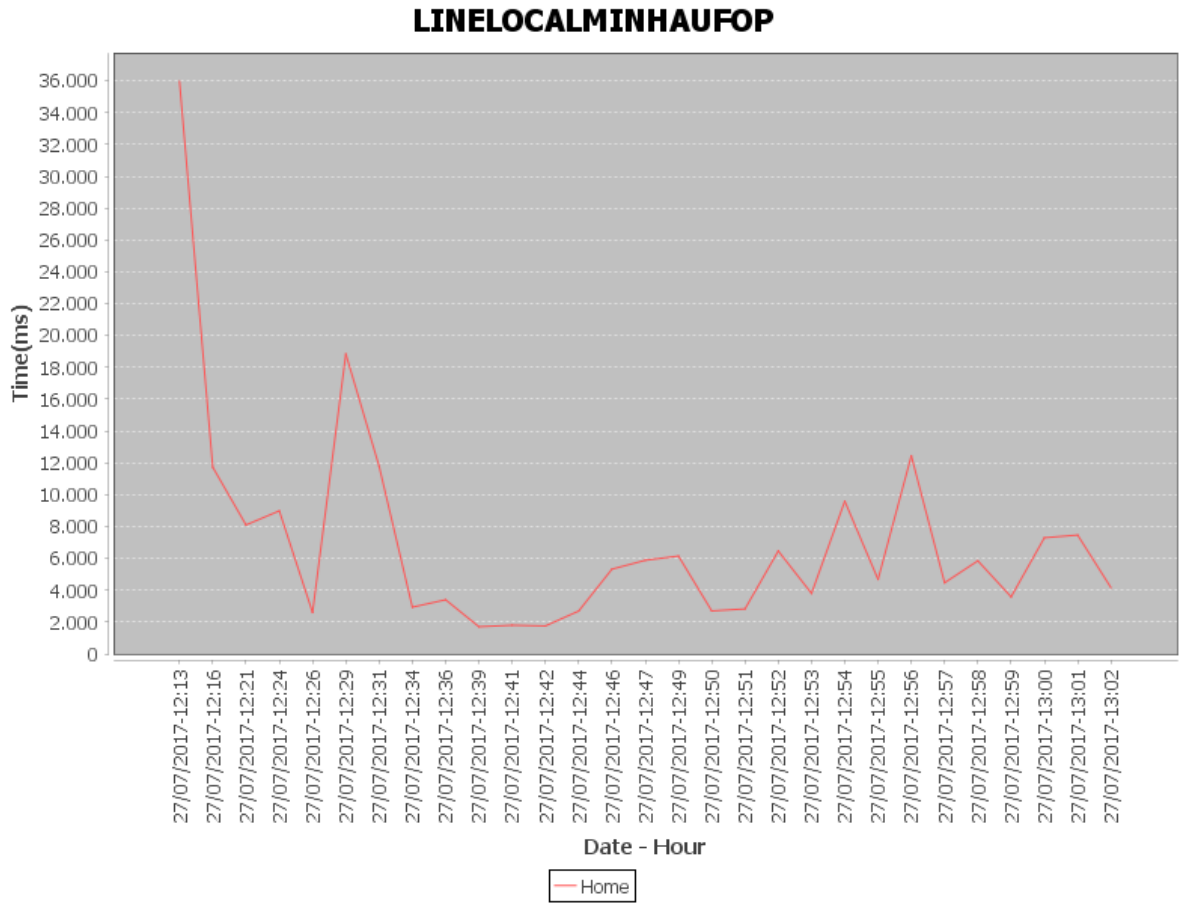
Vários testes foram executados com o *ViewIt* para observar as medições feitas por ele bem como para testá-lo. Alguns testes foram feitos com a página inicial da UFOP de dentro da própria universidade com a finalidade de comparar seus resultados com aqueles gerados por testes fora da universidade. Isto é importante para avaliar como a proximidade com o servidor influencia no desempenho final experimentado pelo usuário.

Os resultados abaixo dizem respeito às medições realizadas na página inicial da UFOP:



8 – Figura: Resultados do desempenho da página inicial da UFOP em rede local (*ViewIt*)

O gráfico linha acima mostra a oscilação do tempo de carregamento da página inicial da ufop no decorrer de quase uma hora. No primeiro teste realizado a página demorou um pouco mais de 38 segundos desde o início da requisição até o completo carregamento dos elementos da página (html, css, imagens, javascript). Vale ressaltar que este número é alto por causa da forma como as métricas são computadas pelo ViewIt. A cada coleta de cada métrica, é feito um somatório para a mesma, tendo assim o valor total do somatório de cada métrica no final da medição. Portanto, quanto menor este número, melhor o desempenho. No fim do teste o tempo já havia estabilizado entre 20 e 10 segundos. Essa estabilização, como observado por Souders(2007, p.64), está relacionada ao uso do cache pelo navegador. Após a primeira requisição, o resolvedor de DNS (Domain Name System) recebe do servidor além da resposta da requisição o tempo de vida do DNS, assim, durante este tempo o DNS fica armazenado na cache do navegador. Nas próximas requisições, se o DNS ainda estiver armazenado no cache (isto dependerá do tempo de vida) não será necessário buscar pelo DNS, diminuindo o tempo de carregamento da página.



9 – Figura: Resultados do desempenho do portal Minha Ufop em rede local (*ViewIt*)

O gráfico acima mostra o tempo de carregamento da página do Minha Ufop durante o mesmo período em que foi medido o desempenho da página inicial da Ufop. Apesar da medição ter sido feita sob as mesmas condições, pode-se observar a diferença no tempo de carregamento máximo e mínimo das duas páginas. O tempo de carregamento da página inicial da Ufop foi de 40 segundos equanto que da página do portal Minha Ufop foi de 36 segundos. Além disso, o portal Minha Ufop obteve menor variação entre as medições e chegou a um mínimo de 4 segundos até o carregamento total da página.

A diferença no desempenho entre as duas páginas é explicada pela estrutura de cada página, pois fatores como a quantidade e a proporção de imagens e scripts usados em comparação com apenas HTML influenciam diretamente no desempenho da página. As imagens abaixo exibem a proporção de cada um desses componentes nas duas páginas medidas:

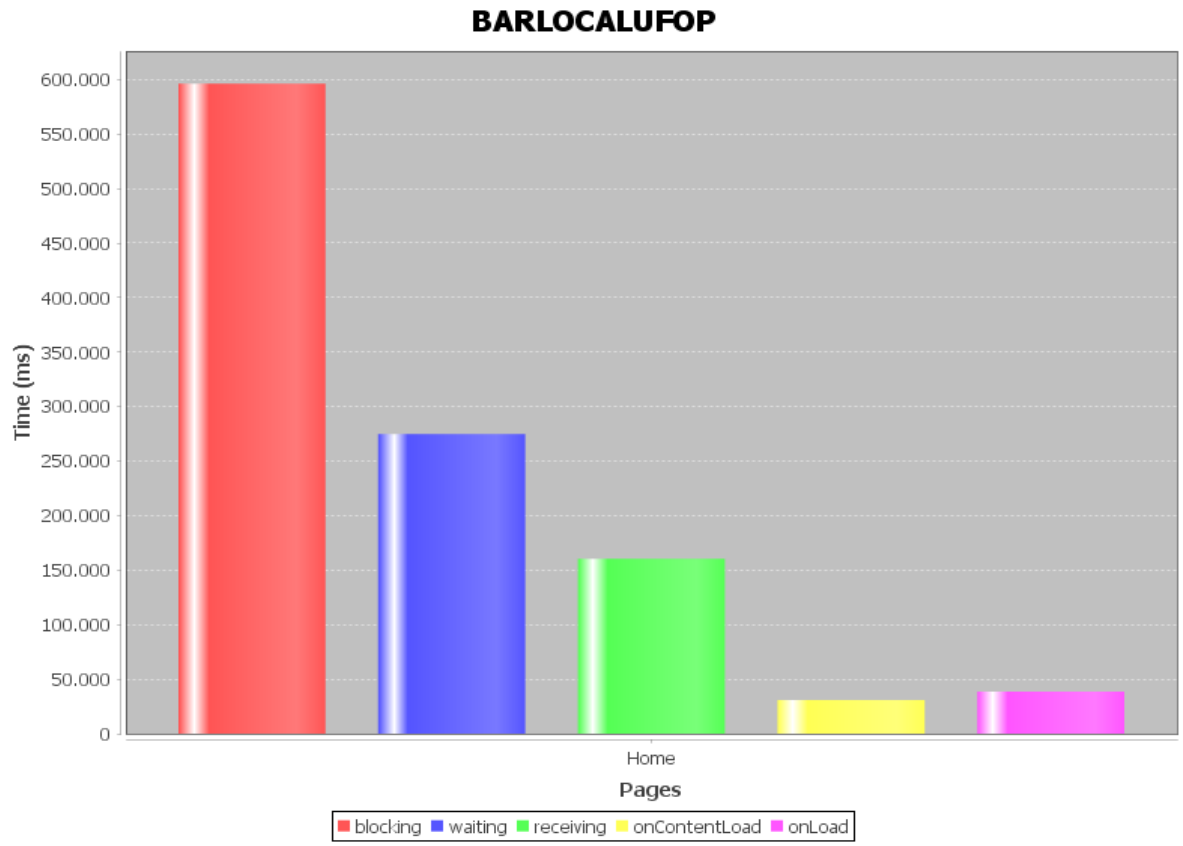
10 – Figura: Quantidade de arquivos por recurso da página inicial da UFOP (*ViewIt*)

O gráfico acima é resultante das medições na página inicial da Ufop. Em relação aos outros elementos da página, o HTML é quase ínfimo. Além disso, os recursos mais usados foram imagens e *scripts*, maiores consumidores de tempo de carregamento da página, principalmente se não estiverem compactados (Souders, 2007).

11 – Figura: Quantidade de arquivos por recurso do portal Minha Ufop (*ViewIt*)

Este gráfico é resultante das medições do portal Minha Ufop. A quantidade de HTML na página é tão ínfima que não aparece no gráfico (o valor numérico foi quase nulo de acordo com as medições). Em uma rápida navegação pelas duas páginas e pode-se observar que a página inicial da Ufop é muito mais dinâmica do que a do portal Minha Ufop, cuja tela principal é a tela de login.

Uma das características mais importantes do *ViewIt*, ao usar o Firebug, é a coleta de cada tempo relacionado ao carregamento de uma página. Cada etiqueta de tempo representa uma etapa do carregamento, como visto na seção 3, e o gráfico abaixo mostra o tempo total (somatório) dispendido em cada uma dessas etapas:



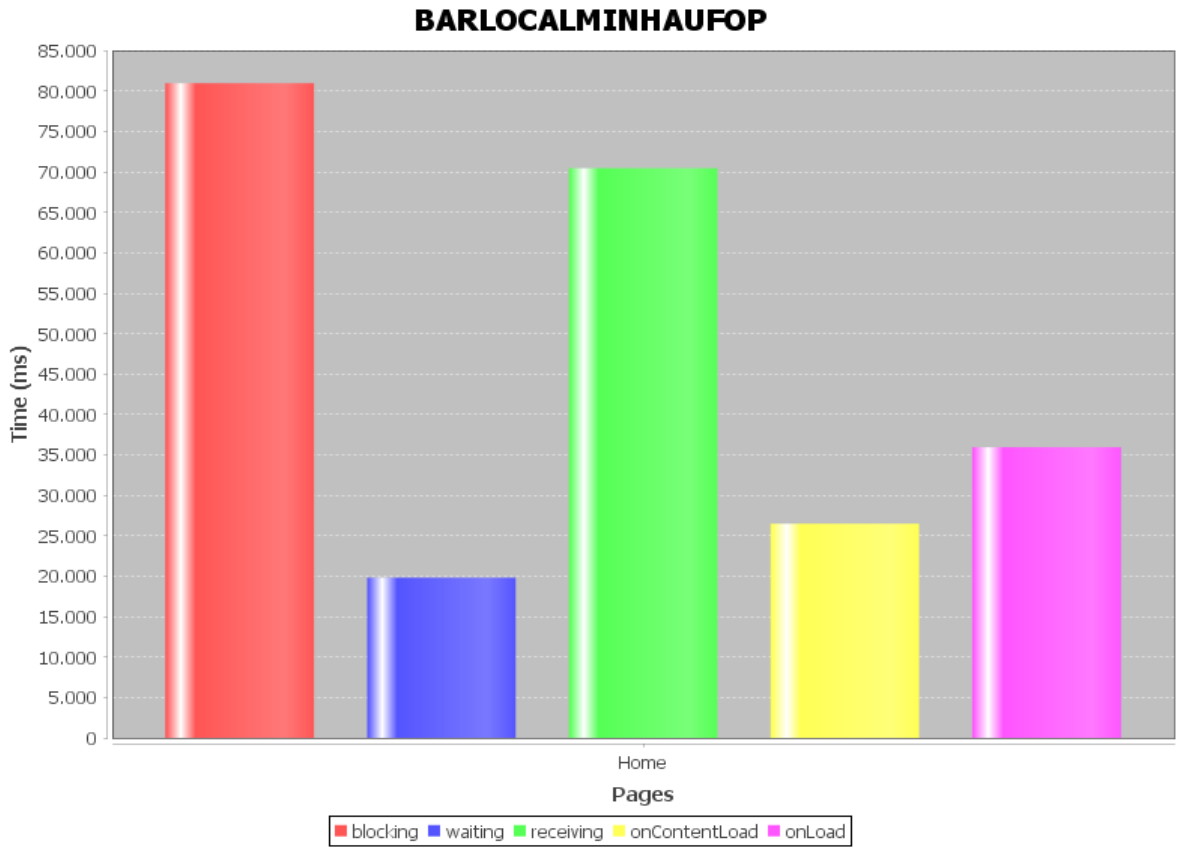
12 – Figura: Resultado de cada etapa da página inicial da UFOP em rede local (*ViewIt*)

Os números estão grandes, mesmo em milissegundos, devido ao somatório de cada etiqueta durante o carregamento da página (ou seja, foi desconsiderado que o carregamento é assíncrono, por isso os números ficam grandes, mas a proporção é a mesma para cada medição). Portanto, deve-se sempre desejar a minimização desses números. Como pode ser visto foi dispendido mais tempo na etapa inicial ao esperar por uma conexão de rede (*blocking*), indicando gargalo na rede. As causas para um gargalo na rede são variadas, como a distância entre servidor e cliente, mas este não é o caso dado que o teste foi feito de dentro da UFOP, onde o servidor se encontra. Outra causa pode ser a quantidade de roteadores intermediários (Grigorick, 2013, p.5), pois se a rede possuir muitos deles ou se não forem bem distribuídos (de forma otimizada) isso pode aumentar o tempo de processamento e atrasar a transmissão de pacotes pela rede. Além disso, o tráfego na rede também é um grande fator de impacto na rede (Grigorick, 2013, p.5).

Apesar da grande diferença entre o tempo de espera da rede e as outras etapas, o tempo dispendido em *waiting* e *receiving* também foi bem maior em relação a *onContentLoad* e *onLoad*. Eles se referem ao tempo de espera do servidor em responder a

requisição e o tempo de leitura da resposta do servidor ou do cache. Esse tempo irá depender da velocidade de transmissão de pacotes entre o servidor e o cliente e também do tráfego entre eles(Grigorick, 2013, p.15).

Abaixo o gráfico do mesmo teste feito no portal Minha Ufop para comparação:



13 – Figura: Resultado de cada etapa do portal Minha Ufop em rede local (*ViewIt*)

Ao comparar as maiores medidas dos dois sites a página inicial da Ufop apresentou 7,5 vezes mais lentidão de rede do que o portal Minha Ufop. Outra diferença está no padrão dos resultados das medições do segundo site, pois o tempo gasto em *waiting* e *receiving* foi inversamente proporcional se comparado com os resultados do primeiro site. Isto quer dizer que não houve demora por parte do servidor em responder a requisição. Por sua vez, para receber a página requisitada demorou bastante (*receiving*) se comparado com o tempo gasto na maioria das etapas. Quer dizer que a requisição é muito pesada para rede e isto pode ser solucionado com a compactação da página durante a fase de desenvolvimento front-end (Souders, 2007, p.29). Uma das técnicas propostas por Souders(2007) e também por Grigorick(2013) é de utilizar o método de compressão Gzip. Para isto a requisição deve

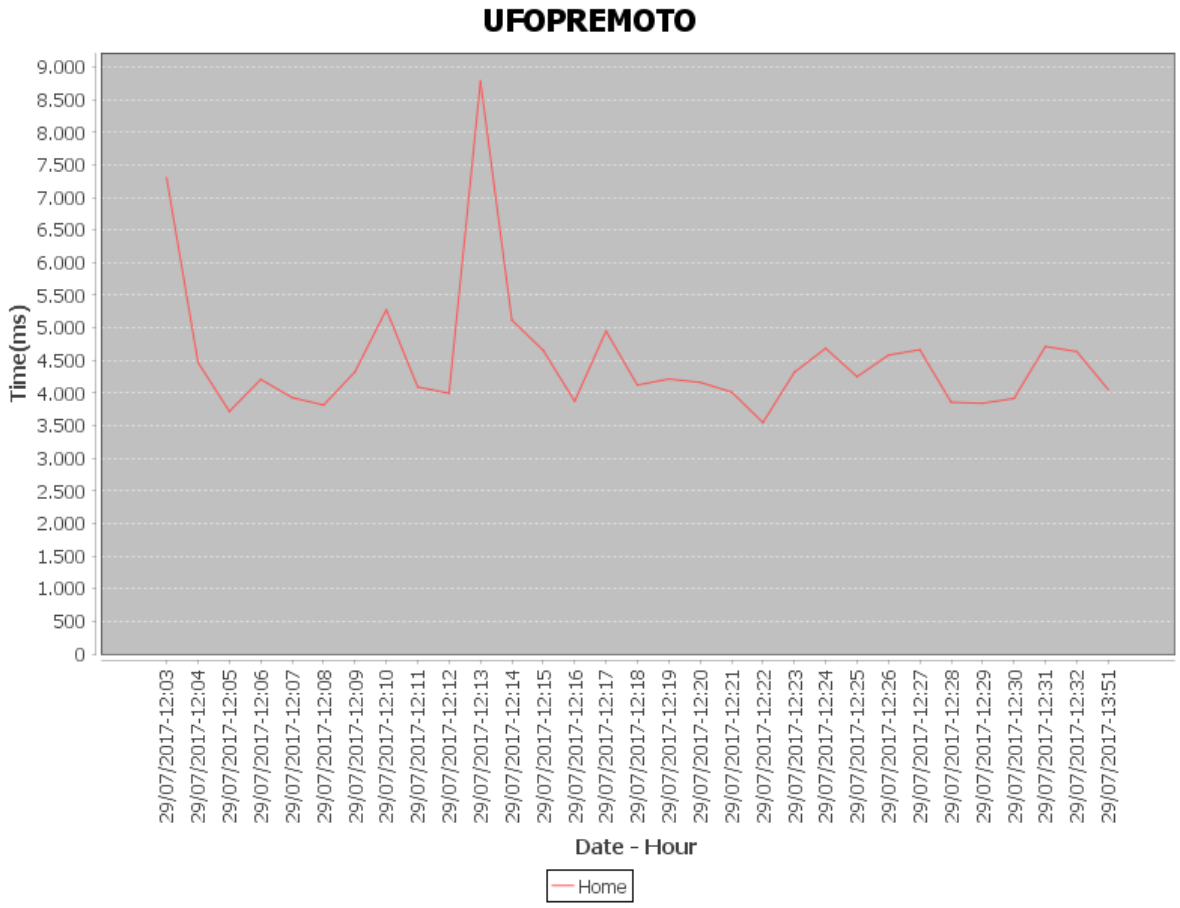
especificar no cabeçalho que deseja uma resposta comprimida, então, ao receber a requisição com este cabeçalho o servidor retornará a resposta de forma comprimida, reduzindo o tempo desta etapa.

As medições feitas apontam para um gargalo na rede da Ufop, mas além disso é possível melhorar também a compressão das páginas desenvolvidas pela instituição, apesar de este não ser o principal responsável pela latência. É mais importante o investimento na infraestrutura da rede, de forma a redistribuir os roteadores de forma otimizada e controlar o tráfego da rede. Contudo, dado que a exposição desses dados é uma amostra de menos de uma hora de testes, para orientações mais precisas e para ter certeza de que o gargalo se encontra da rede, é necessário replicar esta pesquisa de variadas formas, usando medições periódicas e constantes. Não foi possível ainda fazer este teste de longa duração e periodicamente de dentro da UFOP, contudo, nas subseções abaixo há amostras coletadas no decorrer de um dia inteiro.

5.2 Requisição remota à página inicial da UFOP

Como já mencionado, foram feitos testes localmente, de dentro da UFOP, bem como de um lugar mais distante. Os testes remotos foram executados durante 30 minutos com um intervalo de 1 minuto entre cada navegação, da mesma forma que os testes locais apresentados anteriormente. Nesta subseção então, serão comparados os resultados dos testes remotos com os dos testes executados localmente e apresentados na subseção anterior.

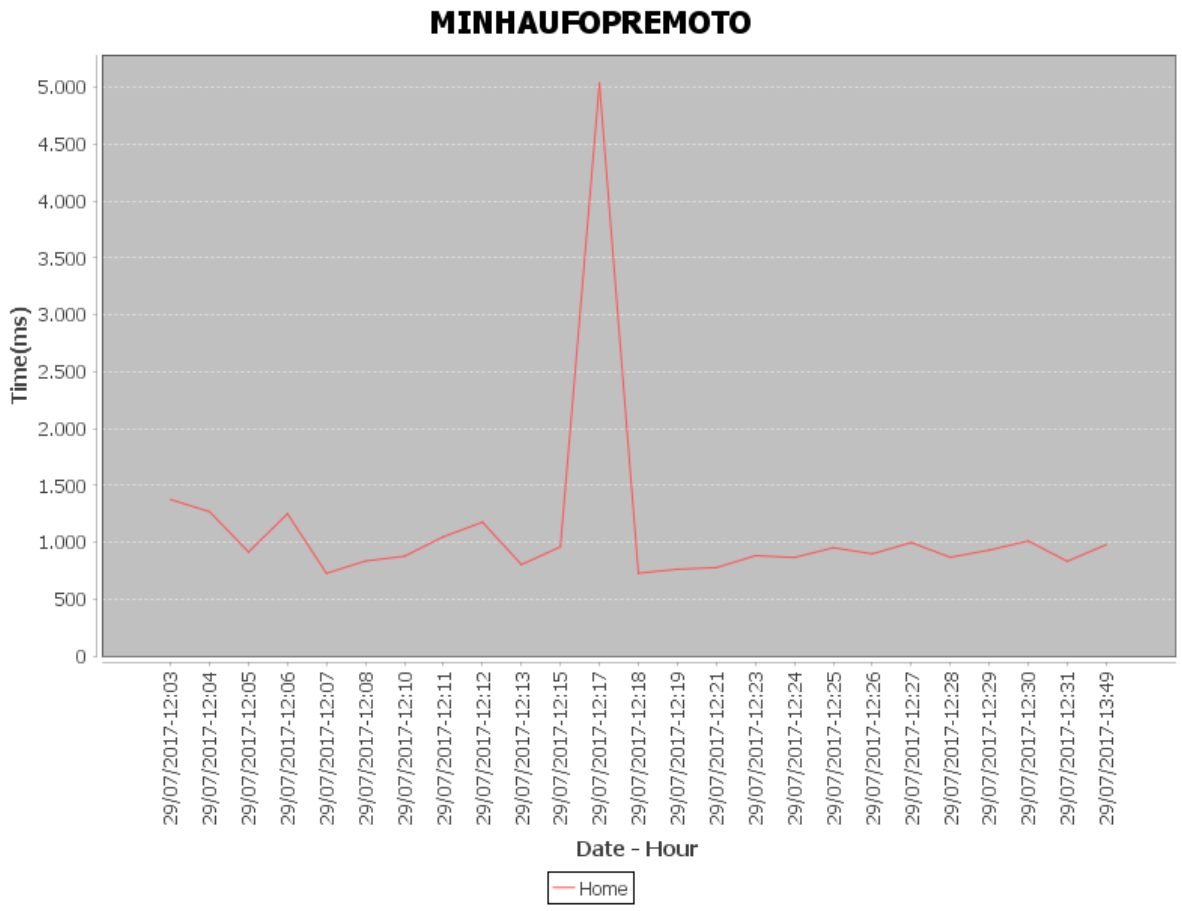
O tempo total (somatório, desconsiderando-se que algumas etapas são assíncronas) de carregamento da página inicial da Ufop estão representados no gráfico abaixo:



14 – Figura: Resultados do desempenho da página inicial da UFOP em rede remota (*ViewIt*)

Os testes foram realizados aproximadamente na mesma faixa de horário (horário de almoço), contudo os testes locais foram feitos durante a semana e pode ser que o dia influencie no tráfego e, portanto, no tempo de carregamento da página. De qualquer forma este gráfico apresenta resultados discrepantes com relação aos testes locais, pois estes resultados mostram que nos testes remotos o maior tempo de carregamento obtido foi quase 5 vezes menor do que o dos testes locais. Além disso, observa-se menor variação de tempos em tempos, ao contrário da rede da Ufop que apresentou alta variação de um teste para outro, o que talvez seja explicado pela grande quantidade de usuários que acessam a rede da Ufop, sempre com tráfego intenso. Contudo, para ter certeza da causa desta discrepância seriam necessários outros tipos de testes, que seriam testes de estresse da rede da Ufop para descobrir o quanto ela suporta e comparar com seu tráfego.

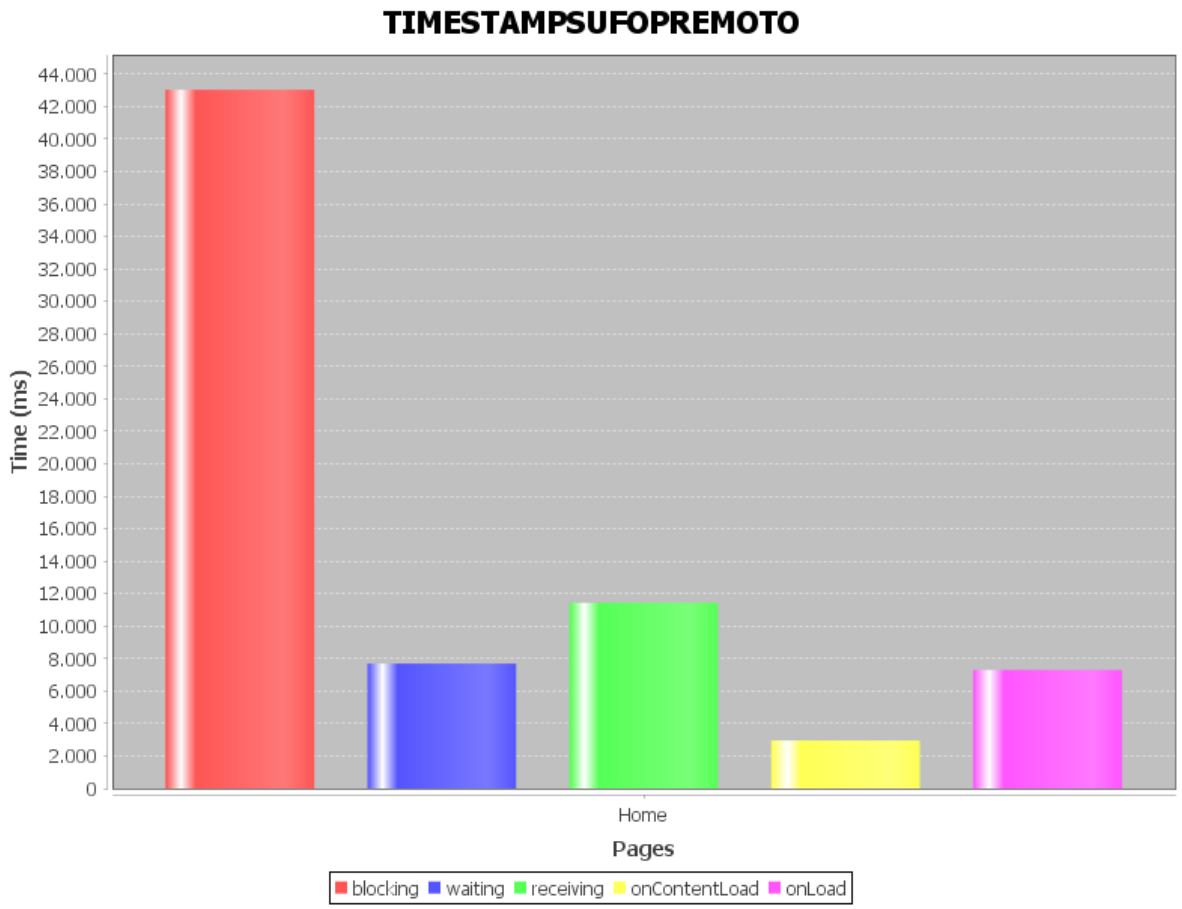
Abaixo está o gráfico do teste remoto no portal Minha Ufop:



15 – Figura: Resultados do desempenho do portal Minha Ufop em rede remota (*ViewIt*)

Assim como nos testes locais, o portal Minha Ufop obteve um desempenho geral melhor do que a página inicial da Ufop. Em comparação com o teste local, o remoto obteve resultados bem melhores, confirmando a latência na rede local da Ufop.

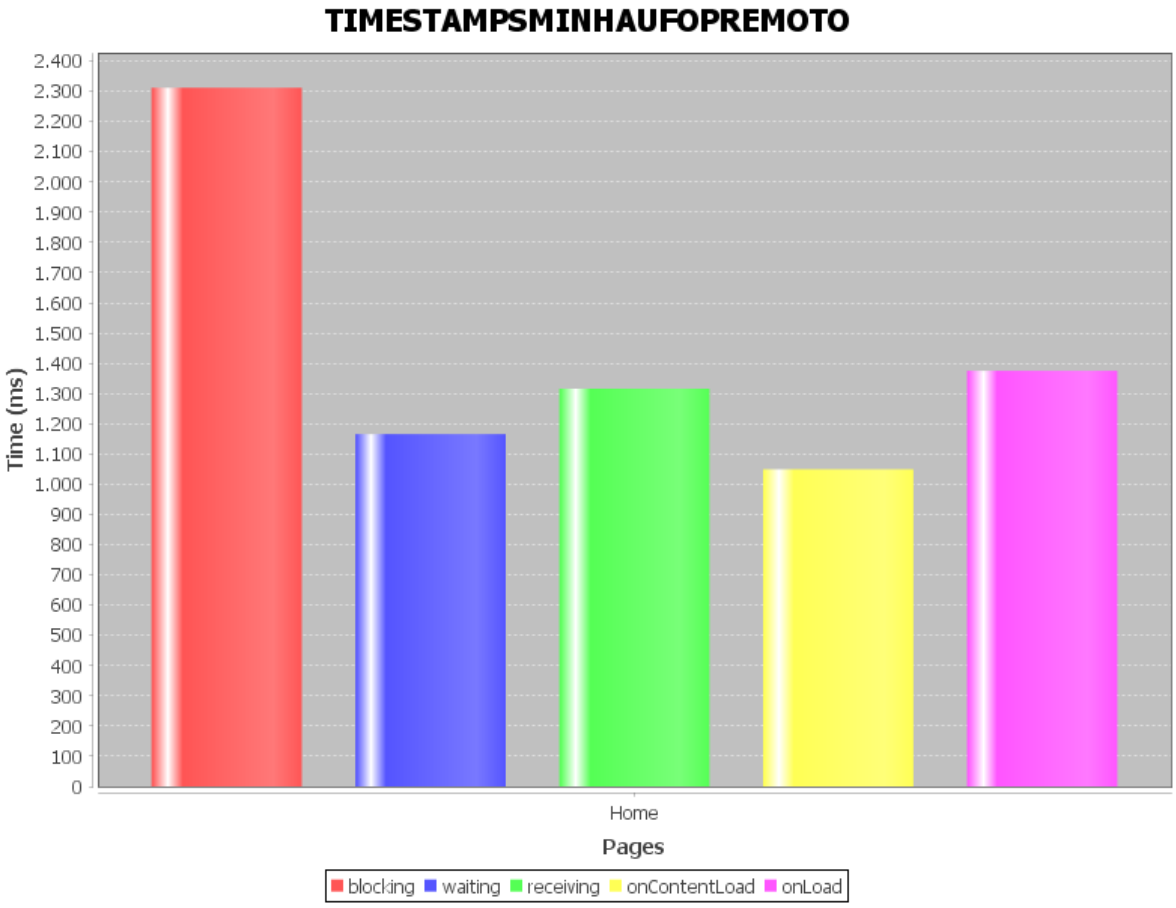
Para comparações mais precisas, abaixo estão os gráficos com cada etiqueta de tempo:



16 – Figura: Resultado de cada etapa da página inicial da UFOP em rede remota (*ViewIt*)

O gráfico acima mostra o tempo total em cada etapa para carregar a página inicial da Ufop de uma rede remota. Mais uma vez é confirmada uma enorme diferença de desempenho entre as requisições remotas e as locais. O tempo de *blocking*, por exemplo, nos testes locais foi de 600 segundos (lembrando que este número só é alto porque soma as medições de etapas assíncronas como se fossem síncronas) e nos remotos foi de aproximadamente 43 segundos, quase 14 vezes mais rápido. Isto só reforça que há um enorme gargalo na rede da Ufop. De qualquer forma, mesmo nos testes remotos, a etapa mais demorada também foi durante a espera de uma conexão (*blocking*), evidenciando que não há nada de errado no desenvolvimento do site, mas sim na rede. Isto porque se o gargalo fosse causado pela forma como o site foi desenvolvido, as medições das etapas *waiting* e *receiving* seriam altas tanto nos testes remotos quanto nos locais, o que não aconteceu.

O próximo gráfico mostra o tempo total em cada etapa para carregar a página inicial do portal Minha Ufop de uma rede remota:



17 – Figura: Resultado de cada etapa do portal Minha Ufop em rede remota (*ViewIt*)

Neste teste não houve muita diferença de uma etapa para outra, apesar de o tempo de espera por conexão (*blocking*) ainda ser o mais longo. Porém, ao comparar com os resultados dos testes locais, todas as etapas foram mais rápidas na rede remota, sendo que o tempo mais longo foi de 2,3 segundos na espera por uma conexão.

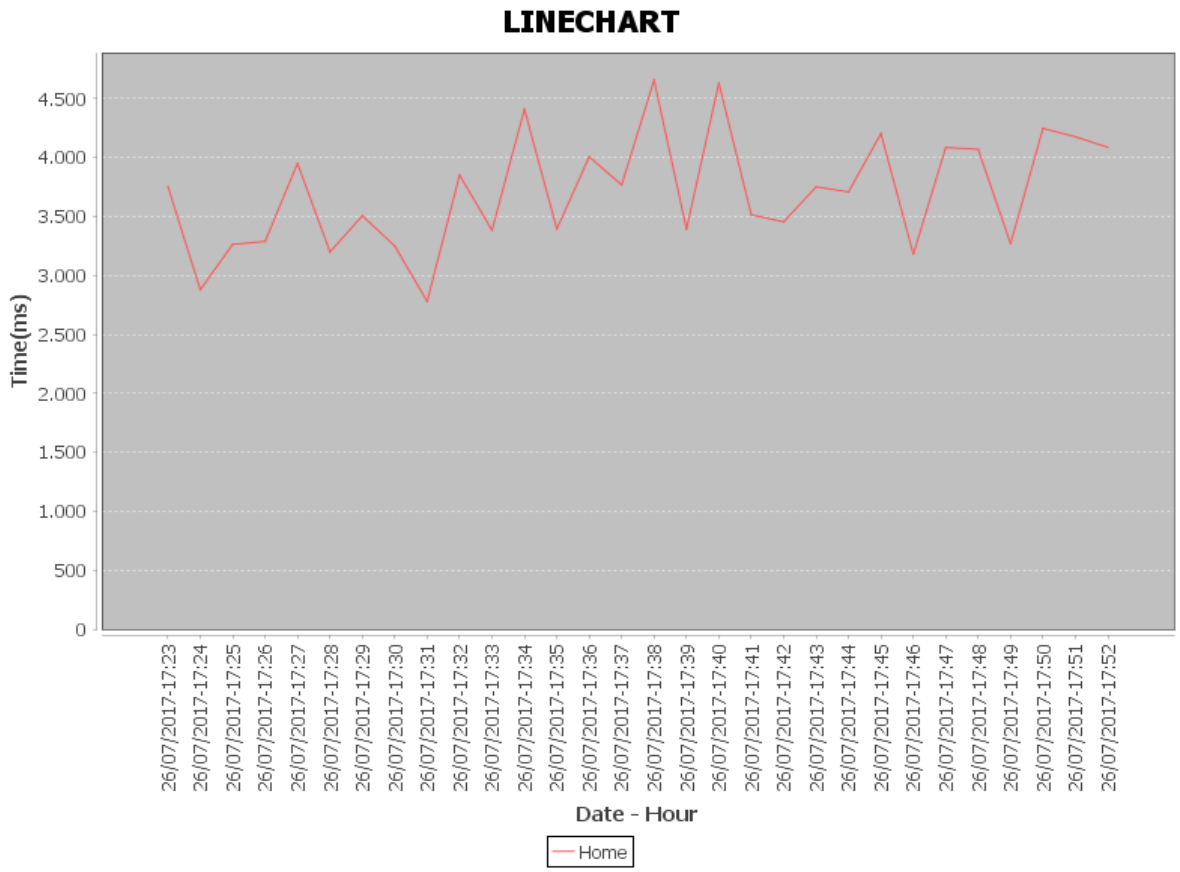
Com a comparação de testes executados em rede local e em rede remota foi possível confirmar que existe latência na rede da UFOP e que, portanto, deve-se buscar soluções viáveis para melhorar sua infraestrutura a longo prazo. Desta forma será possível manter o tráfego da rede sob controle e melhorar seu desempenho.

5.3 Comparativo entre ViewIt e JMeter

Até agora foram apresentados resultados de vários testes executados pelo *ViewIt*, mas sem que isso provasse as qualidade da própria ferramenta. Para tanto, decidiu-se executar os mesmos testes e ao mesmo tempo no *ViewIt* e no *JMeter*, ferramenta amplamente conhecida e utilizada por desenvolvedores web e empresas interessadas em testar suas aplicações web.

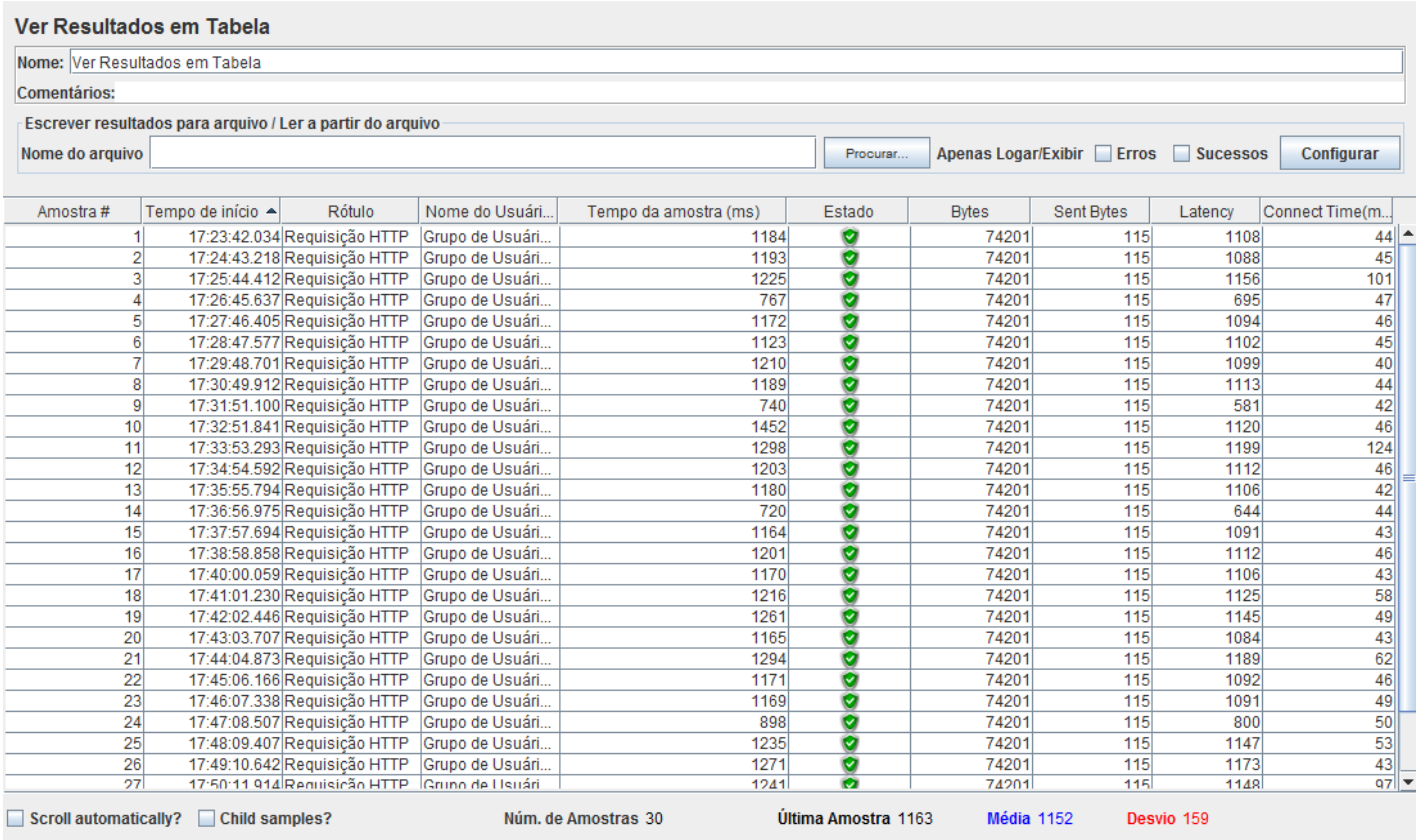
O cenário de teste utilizado foi o carregamento da página inicial da Ufop por 30 minutos, sendo que o teste foi executado a cada 1 minuto tanto pelo *ViewIt* quanto pelo *JMeter*. Vale ressaltar que o *JMeter* não simula o acesso da mesma forma que o usuário faria, portanto seus resultados não são análogos aos resultados que um usuário obteria. O *ViewIt*, por sua vez, simula todos os passos, ele abre o navegador (graças ao Selenium Webdriver, responsável pela automação dos testes) e só fecha após o cliente receber toda a resposta. Isso quer dizer que o *ViewIt* aguarda o carregamento completo de cada componente da página, html, css, imagens, scripts etc, da mesma forma que o usuário aguardaria. Contudo, o *ViewIt* desconsidera o fato de algumas das etapas de carregamento serem assíncronas, somando novamente medições de diferentes etapas que ocorreram ao mesmo tempo. Isto porque o que interessa de fato é o valor total, desde que seja o total de cada etapa. Desta forma, quanto menor o valor em cada etapa, menor o gargalo nesta fase.

Tendo feito as considerações acima, seguem os resultados obtidos pela ferramenta proposta neste projeto:



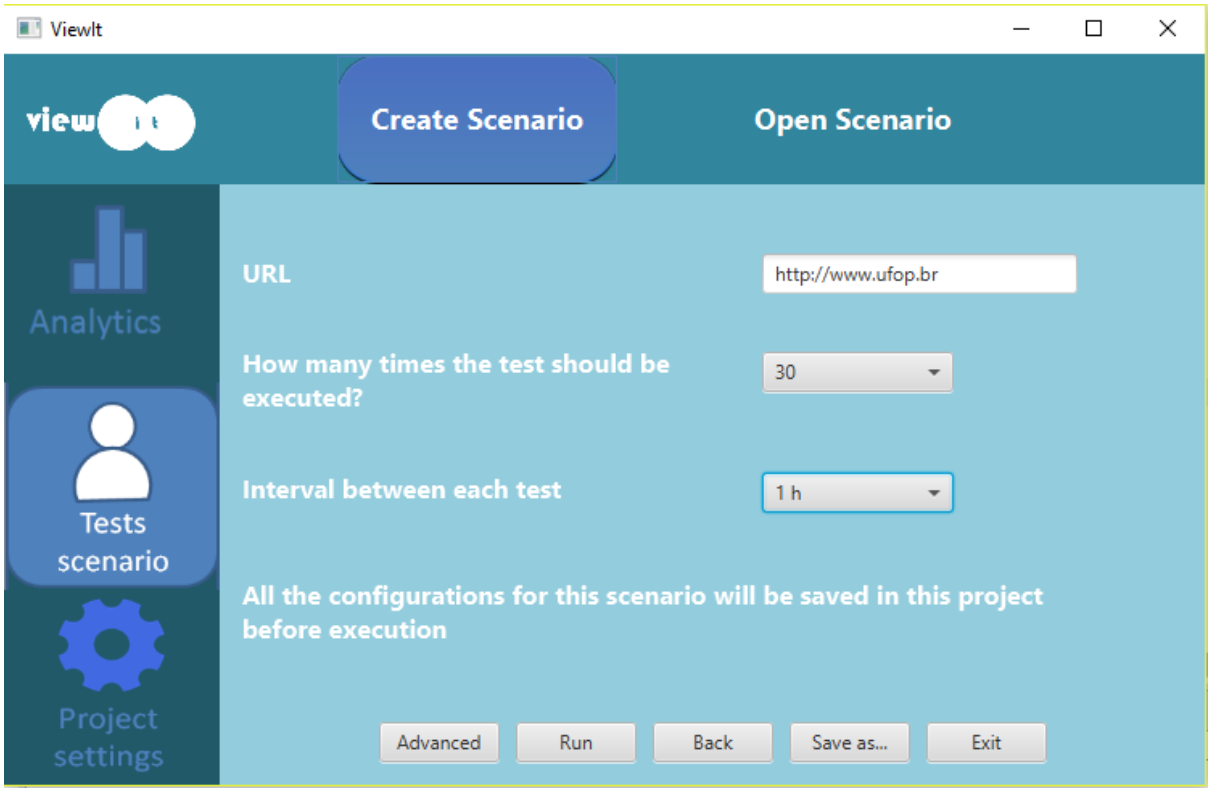
18 – Figura: Resultados da página inicial da UFOP em rede remota pelo *ViewIt*

O pico das medições foi no horário 17:38 e 17:40 com um pouco mais de 4,5 segundos e a menor medição registrada foi às 17:31 com um pouco menos do que 3 segundos. De acordo com as medições do *JMeter*, a maior medição registrada foi de 1,452 segundos às 17:32, lembrando que a ferramenta não executa o teste da mesma forma que o usuário, deixando passar algumas etapas de carregamento e se preocupando mais com o desempenho do ponto de vista do servidor e não do cliente. Sua menor medição foi de 0,720 segundos às 17:36. Para se ter uma noção melhor, às 17:38, quando o *ViewIt* registrou aproximadamente 4,5 segundos, o *JMeter* registrou 1,2 segundos. Os resultados do *JMeter* podem ser vistos abaixo:



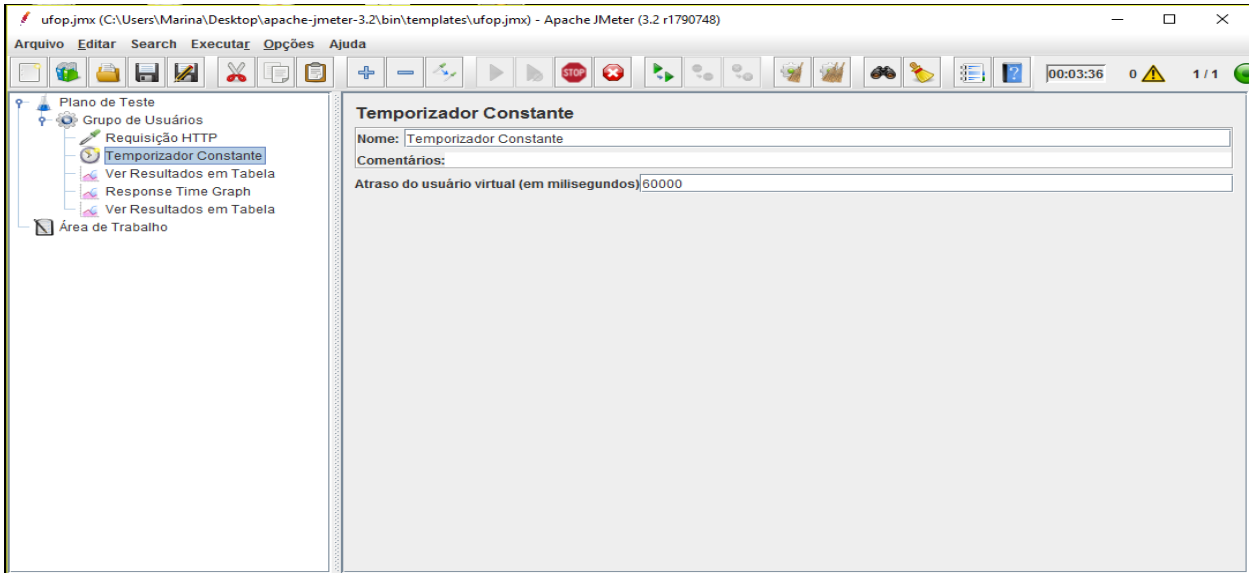
19 – Figura: Resultados da página inicial da UFOP em rede remota pelo *JMeter*

Essas diferenças de resultados mostram como os testes de desempenho do lado do servidor estão longe de representar o desempenho real sofrido pelo usuário. Outras diferenças interessantes entre as duas ferramentas é a forma como uma mesma funcionalidade é apresentada ao usuário. Para definir a quantidade de execuções de um mesmo teste e o intervalo entre cada execução por exemplo, o *ViewIt* pede esses dados como entrada do usuário já na tela principal do cenário de teste, como pode ser visto abaixo:



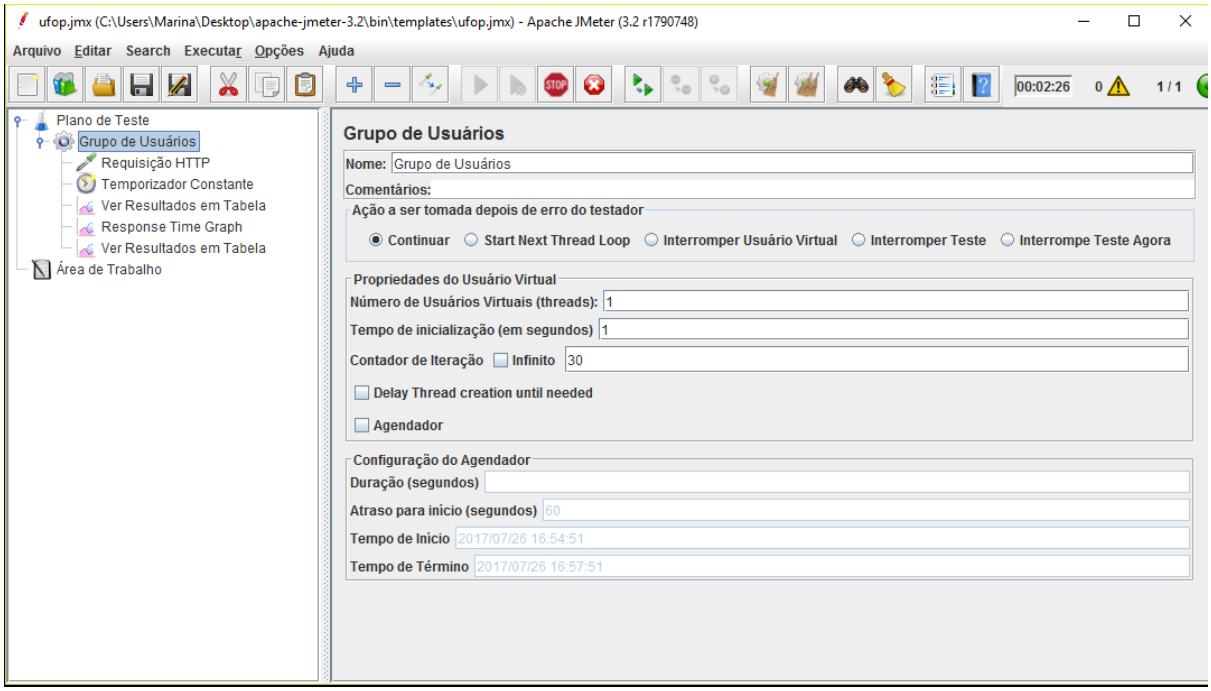
20 – Figura: Tela inicial para criar cenário de teste no *ViewIt*

No *JMeter*, por sua vez, é necessário passar por três telas diferentes para configurar a quantidade de execuções e o intervalo entre execuções. A seguinte imagem representa o Temporizador Constante, componente responsável por definir o intervalo entre as execuções:



21 – Figura: Configuração de intervalo entre execuções no *JMeter*

Na imagem abaixo, em Grupo de Usuários define-se a quantidade de execuções do teste:



22 – Figura: Configuração de quantidade de execuções e usuários no *JMeter*

Portanto, o *JMeter* pode parecer um pouco mais complicado a princípio e, além disso, seu objetivo é medir o desempenho do lado do servidor.

Desta forma, cada uma das ferramentas possui suas vantagens e desvantagens e a escolha de uma em detrimento da outra irá depender do que se deseja medir e como se deseja medir. O *ViewIt* é uma ferramenta que busca ser a mais simples possível e seu objetivo e automatizar testes de desempenho de aplicações web do ponto de vista do usuário e não do servidor.

6 CONCLUSÕES

Os resultados obtidos nos cenários de testes avaliados na seção anterior mostram que a rede local da UFOP precisa ser melhorada. Soluções relativas à otimização da distribuição de roteadores e à melhorias na infraestrutura devem ser buscadas de forma a melhorar a experiência dos usuários da rede.

Sobre as melhorias do *ViewIt*, ao compará-lo com outra ferramenta foi possível notar algumas vantagens como a fácil usabilidade e também o fato de realizar os testes do lado do cliente, pois isto se mostra muito importante quando o objetivo é melhorar a experiência de usuário.

Desta forma, o projeto cumpriu com seus objetivos e, por se tratar de uma ferramenta *open-source* o *ViewIt* sempre estará disponível no *github* para que qualquer pessoa interessada possa fazer mais melhorias.

7 REFERÊNCIAS BIBLIOGRÁFICAS

Apache (2014). Apache JMeter Project. http://jmeter.apache.org/. Accessed: 2016-12-03.

Blazemeter (2015). BlazeMeter offical Web site. http://blazemeter.com/. Accessed: 2016-12-03.

Bolajwar, V. (2013). Measuring perceived performance with navigation timing.

Firebug (2016). Firebug Project. http://getfirebug.com/whatisfirebug. Accessed: 2016-12-03.

Grigorik, I. (2013). High Performance Browser Networking. O'Reilly Media, 1st edition.

JFreeChart (2014). JFreeChart. http://www.jfree.org/jfreechart/. Accessed: 2016-12-03.

JMeter (2015). JMeter-plugins - Documentation. http://jmeter-plugins.org/wiki/WebDriverTutorial/. Accessed: 2016-12-03.

Kyrnin, J. Minimize HTTP Requests to Speed Up Page Load Times. http://webdesign.about.com/od/speed/qt/minimize-http-requests.

htm. Accessed: 2016-12-03.

Margotto, P. R. (2015). Estatística computacional uso do spss.

NeotysUSA (2014). NeoLoad Documentation. http://www.neotys.com/documents/doc/neoload/latest/en/html/#2983.htm. Accessed: 2016-12-03.

Pagani, T. (2014). Medindo performance e latência com a Navigation Timing API. https://tableless.com.br/navigation-timing-api/. Accessed: 2016-12-03.

Richardson, A. (2016). Upgrading to Selenium 3 with My First Selenium Project. http://seleniumsimplified.com/2016/10/

upgrading-to-selenium-3-with-my-first-selenium-project/.

Accessed: 2016-12-03.

Selenium (2016). Selenium Documentation. http://www.seleniumhq.org/docs/. Accessed: 2016-12-03.

Sexton, P. (2015). Web performance. https://varvy.com/performance/. Accessed: 2016-12-03.

Siegel, S. (1956). Nonparametric statistics for the behavioral sciences.

SolarWinds (2014). Solar Winds Documentation. http://www.solarwinds.com/documentation/en/flarehelp/wpm/default.htm#frontmatter/

title.htm\%3FTocPath\%3DWPM\%2520Admin\%2520Guide\%7C___

__0. Accessed: 2016-12-03.

WebDriver (2016). WebDriver Documentation. https://www.w3.org/TR/webdriver/. Accessed: 2016-12-03.