UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

ESCOLA DE INFORMÁTICA APLICADA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

TÉCNICAS PARA MELHORIA DE PERFORMANCE EM APLICAÇÕES WEB NO LADO DO CLIENTE

Zeno Rocha

Mariano Pimentel

Rio de Janeiro
Março / 2013

TÉCNICAS PARA MELHORIA DE PERFORMANCE EM APLICAÇÕES WEB NO LADO DO CLIENTE

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Zeno Rocha

Mariano Pimentel

DEDICATÓRIA

Dedico esse trabalho a minha mãe, meu pai e minha irmã, por todo suporte durante todos esse anos de estudo.

AGRADECIMENTOS

Agradeço aos meus pais Renato e Marjory, pelo apoio e confiança que me deram na construção de toda minha vida escolar. A minha irmã Briza, que me estimulou a concentrar e transmitir de melhor forma minhas ideias nesse trabalho. Agradeço também ao meu orientador Mariano Pimentel, por acreditar em mim não só nessa monografia, mas em todo tempo em que passei na universidade. Agradeço ao meu colega Rúben Jardim, companheiro inseparável de estudo e que me ajudou nos momentos mais difíceis. Por fim, agradeço ao Steve Souders que foi quem me despertou para importância desse assunto através dos seus dois livros.

SUMÁRIO

RESUMO

Nesta monografia é apresentado um estudo sobre a performance no lado do cliente em aplicações web. São mostradas diversas técnicas e ferramentas que auxiliam o desenvolvedor no processo de otimização da performance. Para exemplificar as medidas de performance e a melhoria em decorrência do emprego das técnicas de otimização levantadas nesta monografia, foram realizados estudos com os cinco sites mais acessados do governo brasileiro.

Palavras-chave: Performance; Front-end; Client-side.

CAPÍTULO 1
INTRODUÇÃO

Neste capítulo é discutida a importância da otimização de performance na web. É abordado o impacto que a otimização da performance tem em sites de grande volume de acessos (Seção 1.1) e discutido onde deve-se focar o trabalho de otimização (Seção 1.2). O conteúdo abordado nos demais capítulos desta monografia é apresentado na Seção 1.3.

1.1 O IMPACTO DA OTIMIZAÇÃO DE PERFORMANCE

Desenvolvedores web criam páginas inovadoras e atraentes para melhorar a experiência com o usuário. Com a evolução das tecnologias web, começaram a ser adicionados cada vez mais recursos de mídia. Primeiro veio o uso exagerado de músicas nos sites. Depois veio o excesso de animações em Flash para chamar atenção dos usuários. Mais tarde, com o advento do Youtube, vídeos pesados passaram a ser um recurso muito utilizado nas páginas web. Atualmente, com a chegada da nova especificação HTML5, CSS3 e novas APIs de JavaScript, os desenvolvedores passaram a ter mais poder para desenvolver interfaces web ainda mais cheias de recursos. A sobrecarga de recursos diminui a velocidade de exibição da página, o que degrada a experiência do usuário.

Em contrapartida ao excesso de recursos, alguns websites focam no conteúdo e na rapidez em oferecer os resultados ao usuário, como é o caso do Google, que prioriza a indexação de conteúdo e a rapidez em apresentar o resultado de uma busca em detrimento do uso de recursos de mídia. Esta abordagem representa uma mudança do ponto de vista do design: além do conteúdo e dos aspectos estéticos (não basta construir sites bonitos), a experiência do usuário também é influenciada pela velocidade de exibição das páginas.

Nenhum usuário gosta de perder tempo esperando o carregamento de uma página web. Além de irritante, causa graves consequências com relação ao negócio, como é mostrado nos estudos a seguir:

- O **Yahoo!** descobriu que, para cada 400 milissegundos de melhora na performance, seu tráfego aumenta em 9% (Stefanov, 2008);
- Ao cortar 2,2 segundos da landing page do Firefox, a **Mozilla** aumentou o número de downloads em 15%, totalizando um ganho de mais de 60 milhões de cópias por ano (Cutler, 2010);
- A **Amazon** concluiu que apenas 100 milissegundos de piora no tempo de carregamento, diminui 1% de seu faturamento (Linden, 2008);
- Em um de seus vários experimentos, o **Google** aumentou o número de resultados por página de 10 para 30. Isso aumentou o tempo de carregamento de 0,4 segundos para 0,9 segundos, o que diminuiu em 20% o tráfego das buscas (Linden, 2006);
- A **Microsoft** mostrou que 2 segundos a mais de latência no Bing diminuíam o faturamento em 4,3% (Schurman, 2009);

Esses são alguns dos estudos de empresas poderosas e globais que despertam a atenção para a importância da performance dos sites. Também no Brasil são encontrados estudos semelhantes, como exemplificam as análises sobre o portal Terra e Peixe Urbano:

- Em 60 segundos, o portal Terra recebe cerca de 535 mil autenticações, envia 32 mil e recebe 17 mil e-mails, carrega 150 KB na sua capa e 540 KB em banners, recebe 3.6 milhões de hits e consome 2.7 TB de rede. Logo, economizar 1 KB significa: a cada 60 segundos, uma economia de 3.4 GB; em 1 hora, uma economia de 180 GB; e em 1 dia, 4.2 TB (Gomes, 2012);
- O Peixe Urbano possui mais de 20 milhões de usuários ativos, o que gera mais de 22 milhões de acessos e mais de 75 milhões de visualizações de página por mês. Depois de uma série de otimizações, a página que totalizava 3 MB foi diminuída para 267 KB, o que resulta em uma redução de tempo em seu carregamento de 7.5 segundos para 4.3 segundos. Essa diminuição resultou em uma economia de R$

16.000,00 por dia no custo com servidores e um aumento de receita de mais de R$ 1.000.000,00 por dia (Keppelen, 2012).

1.2 ONDE DEVE-SE FOCAR A OTIMIZAÇÃO

Podemos dividir a programação para web em dois nichos distintos: back-end e front-end. O back-end trata do desenvolvimento no lado do servidor, por meio de linguagens como Java, Ruby, PHP e Python. O front-end lida no lado do cliente, por meio de linguagens como HTML, CSS e JavaScript.

A maioria dos desenvolvedores já investe em melhorias no lado do servidor (por meio de índices de bancos de dados, gerenciamento de memória etc.), mas esquecem de melhorar a performance no lado do cliente, como alerta Steve Souders, engenheiro da Google:

> (...) Há muita atenção e muitos livros dedicados a otimizações de back-end, de modo que é onde a maioria das pessoas gasta tempo procurando melhorias. Na realidade, para maior parte das páginas da web, menos de 10-20% do tempo de resposta do usuário final é gasto buscando o documento HTML de um servidor web até o navegador. Se você quiser reduzir drasticamente o tempo de resposta de suas páginas web, otimize a performance do front-end primeiro, é onde 80-90% do tempo de resposta ao usuário é gasto. (Souders, 2007, p.22)

Este trabalho aborda a otimização no lado do cliente por meio de técnicas para minimizar o tempo de latência em rede, o tamanho e o número das requisições HTTP.

1.3 ORGANIZAÇÃO DA ESCRITA DESTA MONOGRAFIA

Esta monografia é organizada em 9 capítulos. Neste primeiro capítulo é discutido o impacto da performance e onde deve-se o focar esforço em sua otimização. No Capítulo 2 são apresentados as principais ferramentas de diagnóstico de problemas com relação à performance. No Capítulo 3 são apresentados os sites selecionados para exemplificar as técnicas de melhoria de performance, o critério de escolha destes sites e suas avaliações gerais feitas pelas ferramentas de diagnóstico. No Capítulo 4 é analisada a anatomia de uma requisição HTTP ao navegador. No Capítulo 5 é mostrado como diminuir o tempo de latência

na rede através da distribuição geográfica dos arquivos. No Capítulo 6 é mostrado como minimizar o tamanho das requisições HTTP por meio de técnicas de compressão. No Capítulo 7 é mostrado como diminuir o número de requisições HTTP por meio da concatenção de arquivos. No Capítulo 8 é mostrado o guia colaborativo que foi criado com base nessa monografia. A conclusão deste trabalho é apresentada no Capítulo 9.

CAPÍTULO 2
FERRAMENTAS DE DIAGNÓSTICO DE PERFORMANCE

Uma ferramenta de diagnóstico serve para automatizar o processo de detecção de problemas relacionados à performance. Nesse capítulo são abordados YSlow (Seção 2.1), PageSpeed (Seção 2.2) e WebPageTest (Seção 2.3), que são as principais ferramentas de diagnóstico existentes.

2.1 YSLOW

Criado pelo Yahoo! em 2007 como uma extensão para o navegador Mozilla Firefox, o YSlow foi a primeira ferramenta de diagnóstico de performance. A ferramenta avalia uma página web em 23 critérios. Como ilustrado na Figura 1, a ferramenta apresenta um relatório com uma nota geral de A a F, um score geral de 0 a 100, e são avaliados cada um dos 23 critérios com uma nota de A a F.

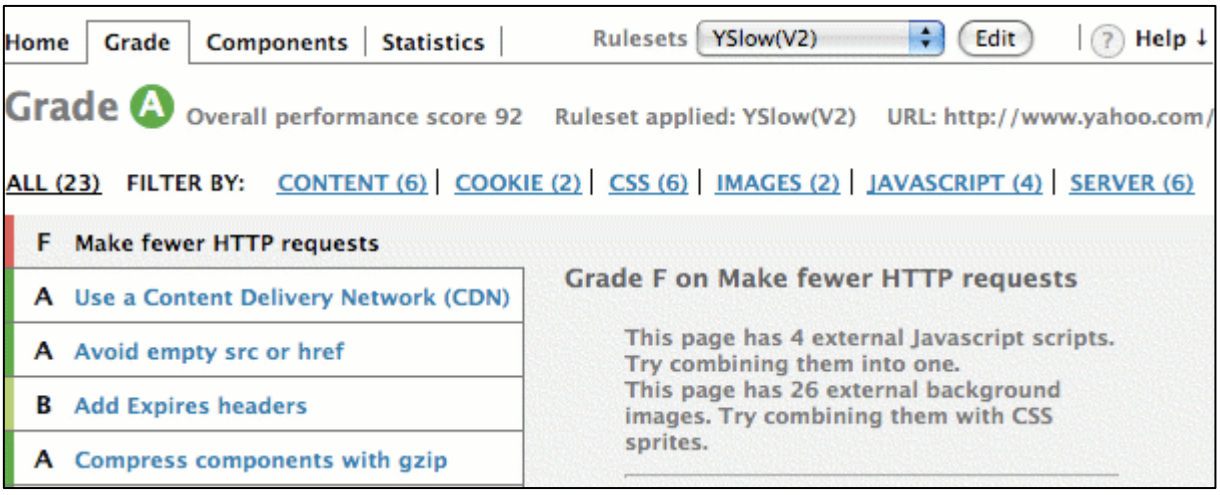

Figura 1. Resultado de um diagnóstico do YSlow.

Pouco antes de sair do Yahoo!, o brasileiro Marcel Duran, conseguiu transformar o projeto YSlow[1] em uma iniciativa de código aberto. Desde então o projeto expandiu para outras plataformas como Google Chrome, Safari e Opera. Além disso, há também um Bookmarklet para testes em dispositivos móveis, integração com PhantomJS e uma ferramenta de linha de comando.

2.2 PAGESPEED

O PageSpeed, ilustrado na Figura 2, é uma ferramenta desenvolvida pela Google. Foi inicialmente implementada como uma extensão do Google Chrome[2] e atualmente está disponível também para Firefox, tem uma versão online (independentemente de navegador), e uma API pública (para extensões da análise). Esta ferramenta produz um relatório com a análise da performance sobre uma página web baseada em 31 critérios, e atribui uma nota geral de 0 a 100.

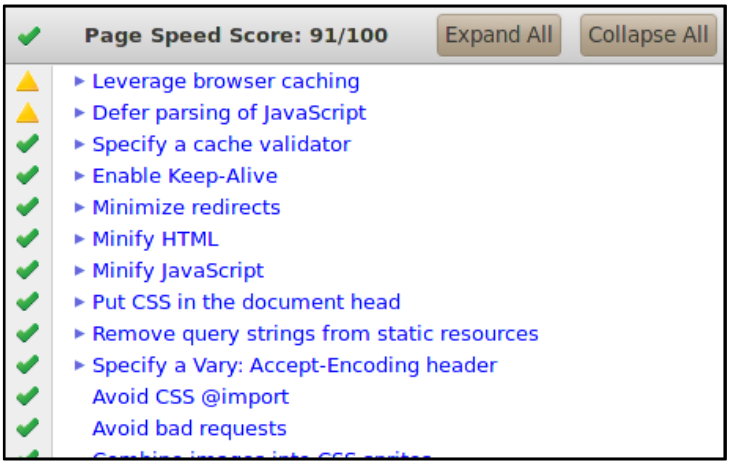

Figura 2. PageSpeed - Extensão para Chrome.

[1] http://yslow.org/
[2] https://chrome.google.com/webstore/detail/gplegfbjlmmehdoakndmohflojccocli/

2.3 WEBPAGETEST

O WebPageTest[3] é uma ferramenta de diagnóstico online. Conforme ilustrado na Figura 3, o usuário digita uma URL, depois escolhe qual navegador e local gostaria que a página fosse carregada. Após executado o teste, vários gráficos são apresentados ao usuário. O diferencial consiste na escolha geográfica de onde será realizada a requisição, o que possibilita testar a perfomance como se alguém da China acessasse determinada página web hospedada em um servidor no Brasil, por exemplo.

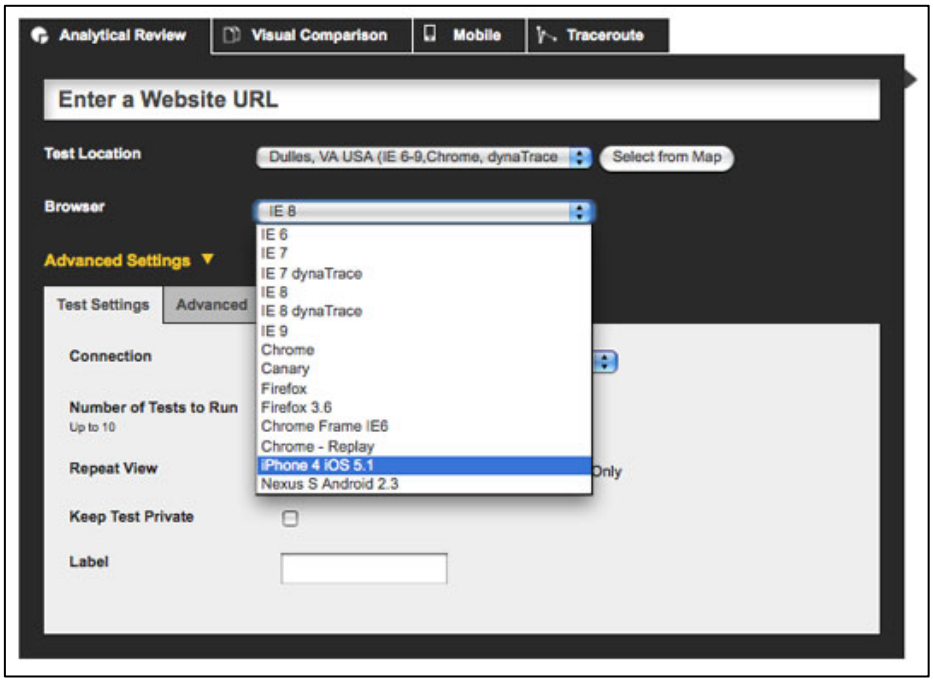

Figura 3. WebPageTest.

[3] http://www.webpagetest.org

CAPÍTULO 3
SITES SELECIONADOS PARA EXEMPLIFICAR AS TÉCNICAS DE MELHORIA DE PERFORMANCE

Neste capítulo são apresentadas cinco páginas web avaliadas com relação à performance em diferentes quesitos ao longo de toda a monografia. Foi definido um critério, apresentado no Seção 3.1, para a seleção de cinco páginas web. Na Seção 3.2 é apresentada a avaliação geral destas páginas, realizada pelas ferramentas de diagnóstico apresentadas no capítulo anterior.

3.1 CRITÉRIO DEFINIDO PARA A SELEÇÃO DE PÁGINAS

Para definir quais seriam as cinco páginas avaliadas em diferentes critérios ao longo da monografia, foi consultado o Alexa[4], que é o ranking das páginas mais acessadas em cada país. Após a consulta pelo país Brasil, foi realizado um filtro para selecionar apenas os sites pertencentes ao governo brasileiro, ou seja aqueles que contém domínio .gov.br. A lista dos cinco primeiros sites desta consulta, realizada em 9 de janeiro de 2013, se encontra na Tabela 1.

[4] <http://www.alexa.com/topsites/countries/BR> Acessado em 9 de janeiro de 2013

Tabela 1: Ranking dos 5 websites do governo mais acessados no Brasil. Dados coletados em 09.jan.2013

Rank	Website	URL	Ranking Alexa
1	Caixa Econômica Federal	caixa.gov.br	24
2	Governo do Estado de São Paulo	www.sp.gov.br	44
3	Ministério da Fazenda	fazenda.gov.br	58
4	Ministério da Educação	mec.gov.br	117
5	Governo do Estado do Rio de Janeiro	www.rj.gov.br	132

3.2 AVALIAÇÕES INICIAIS NO PAGESPEED, YSLOW E WEBPAGETEST

Os sites selecionados (apresentados na Tabela 1) foram testados com as principais ferramentas de diagnóstico (apresentadas no Capítulo 2). As notas obtidas, em uma escala de 0 à 100, estão listadas na Tabela 2.

Tabela 2: Avaliações dos 5 websites do governo mais acessados no Brasil. Dados coletados em 09.jan.2013

Rank	Website	PageSpeed	YSlow	WebPageTest	Média
1	Caixa Econômica Federal	47	Nota C (74)	51	57
2	Governo do Estado de São Paulo	66	Nota C (75)	67	69
3	Ministério da Fazenda	68	Nota A (98)	63	76
4	Ministério da Educação	78	Nota B (83)	80	80
5	Governo do Estado do RJ	52	Nota D (65)	56	57

Nenhum dos portais mais acessados do governo brasileiro obteve um resultado muito bom. A maioria nem sequer apresenta uma performance satisfatória[5], o que é muito preocupante pois serviços públicos deveriam ser os primeiros a oferecer qualidade no acesso de suas páginas web. Como diria David Cheriton: "Se é rápido e feio, as pessoas vão usá-lo e odiá-lo. Se é lento, elas não vão usá-lo" (*apud* Jain, 1991, p.39).

[5] Considera-se a performance "muito boa" quando a página obtém nota acima de 90. Considera-se a nota de 80 a 89 como um indício de que a performance é boa, ainda que possa ser melhorada. Considera-se que a página tem performance insatisfatória quando obtém nota abaixo de 80.

CAPÍTULO 4

A ANATOMIA DE UMA REQUISIÇÃO HTTP

Diversos fatores podem afetar a performance de uma página: o tamanho dos arquivos inclusos, a quantidade de elementos em uma página, a distância entre o cliente e o servidor, a largura de banda etc. Para otimizar o carregamento de páginas web e, consequentemente, melhorar a experiência do usuário, é preciso entender como funcionam as requisições HTTP ao servidor.

Um site realiza dezenas de requisições, já que para cada documento HTML, folha de estilo CSS, arquivo JavaScript e imagem é preciso realizar uma requisição HTTP diferente. Conforme ilustrado na Figura 4, o carregamento de uma requisição HTTP passa por diversos processos diferentes. Neste capítulo é abordada a anatomia de uma requisição HTTP, como DNS Lookup (Seção 4.1), Initial Connection (Seção 4.2), Cabeçalho Keep-Alive (Seção 4.3), Time to First Byte (Seção 4.4), Content Download (Seção 4.5), Start Render e Document Complete. Como o processo de Start Render e Document Complete são praticamente instantâneos, não são alvo de otimização da performance.

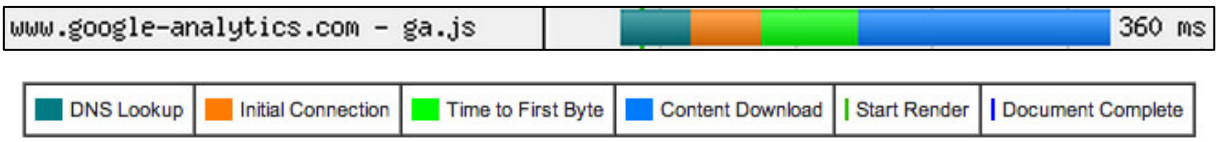

Figura 4. Demonstração de requisição HTTP - WebPageTest.

4.1 DNS LOOKUP

A primeira fase é chamada de DNS Lookup (Domain Name System). É o processo de conversão de nome (domínio) em um IP – por exemplo, a URL *http://google.com* está associada ao IP 173.194.70.100. Só assim o navegador consegue realizar a conexão com o servidor remoto. Na requisição ilustrada na Figura 4, o processo de DNS Lookup durou 50 milissegundos.

Para cada domínio diferente que é solicitado, um novo DNS Lookup é realizado. Isso vale mesmo para subdomínios como *http://imagens.meusite.com.br,* já que diferentes subdomínios podem estar associados a diferentes IP's.

4.2 INITIAL CONNECTION

Toda requisição HTTP que um navegador solicita para um servidor é trafegada através de conexões TCP (Transmission Control Protocol). É preciso ao menos uma conexão TCP ativa para que o download dos componentes da página seja realizado. Na requisição exibida na Figura 4, o processo de Initial Connection durou 51 milissegundos.

Conforme ilustrado na Figura 5, um método chamado *three-way handshake* é feito entre o cliente e o servidor por meio de metadados enviados nos pacotes, só assim é estabelecida uma conexão TCP. Os pacotes do *handshake* são enviados e reconhecidos pelas duas pontas, então a conexão é estabelecida e a transferência do arquivo é iniciada. Esse processo se realiza nos seguintes passos:

1. O navegador envia um pacote para o servidor com o metadado SeN (Sequence Number);
2. O servidor então responde com ACK (Acknowledged) e um outro SeN;
3. O navegador finaliza o cumprimento *(handshake)* com mais um ACK;
4. A conexão é estabelecida.

Figura 5. Processo de handshake entre o navegador e o servidor.

4.3 CABEÇALHO KEEP-ALIVE

Para evitar um sobrecarregamento no estabelecimento de conexões TCP *(handshake)*, um novo cabeçalho foi introduzido na especificação do HTTP 1.1 chamado *Keep-Alive*. Seu objetivo é reaproveitar conexões TCP para diferentes requisições. Assim, ao utilizar uma conexão TCP aberta e que não tenha sido encerrada, a conexão é reaproveitada para trafegar outras requisições.

Na Figura 6, cada conexão TCP está representada por uma linha, e uma mesma conexão pode ser reaproveitada para trafegar diferentes tipos de componentes.

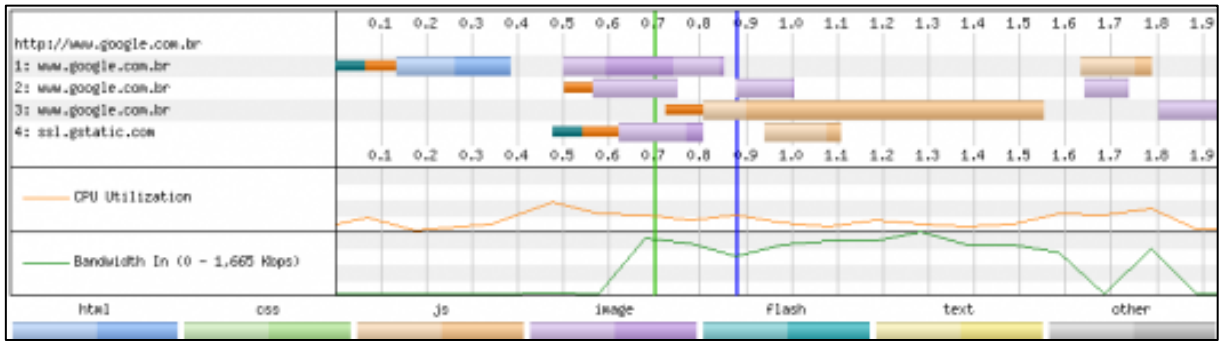

Figura 6. Visualização das múltiplas requisições sendo feitas.

4.4 TIME TO FIRST BYTE

O tempo que o navegador leva para receber o primeiro byte de informação da requisição é chamado de TTFB (Time To First Byte). Na requisição ilustrada na Figura 4, o processo de TTFB durou 72 milissegundos. Esse tempo é influenciado pela demora no

processamento server-side, pela localização do servidor e pela largura de banda da rede.

4.5 CONTENT DOWNLOAD

Depois de receber o primeiro byte, o resto do tempo da requisição depende do tamanho do componente que será carregado. Como ilustra a requisição na Figura 4, o processo de Content Download durou 187 milissegundos, o que corresponde à metade do tempo gasto nessa requisição inteira.

CAPÍTULO 5
O TEMPO DE LATÊNCIA NA REDE

A partir do estudo da anatomia de uma requisição HTTP conseguimos traçar o caminho para o processo de otimização na performance de aplicações web. Resumidamente, quanto maior a latência, número e tamanho das requisições mais lento um website será.

Portanto, nos próximos capítulos é abordado técnicas para minimizar cada um desses aspectos, começando pelo tempo de latência na rede.

5.1 DISTRIBUIÇÃO GEOGRÁFICA DE ARQUIVOS

Suponha que o usuário está na China e o servidor em Los Angeles. Quando ele carrega um arquivo, seu navegador envia uma requisição HTTP que atravessará o globo até chegar no seu servidor, e como é de se imaginar, isso leva tempo.

Entretanto, se os arquivos estão hospedados em uma CDN (Content Delivery Network), será distribuido geograficamente entre diferentes servidores no mundo. Logo, quando o usuário solicitar determinado arquivo, esse será carregado a partir do servidor mais próximo, o que diminui muito o tempo de latência.

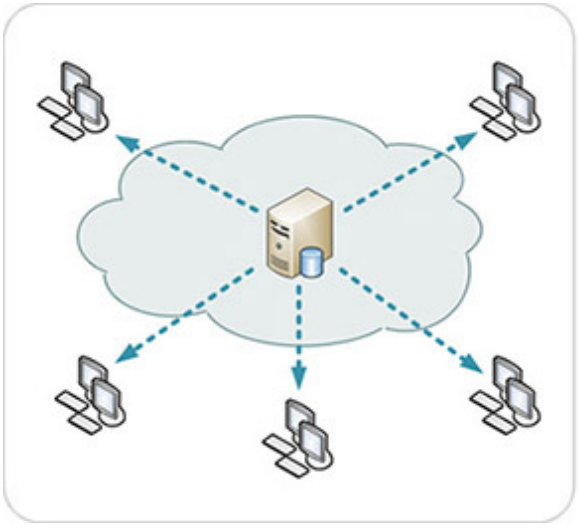

Figura 7. (Esquerda) modelo tradicional de distribuição
(Direita) Modelo CDN de distribuição

Como visto na Figura 8, para analisar os benefícios da melhoria de performance com um caso real, o teste é realizado carregando exatamente o mesmo arquivo, usando e não usando CDN.

Figura 8. Requisição de um arquivo JavaScript sem utilizar uma CDN.

Como visto na Figura 9, para carregar um arquivo JavaScript, que não está hospedado em uma CDN com uma conexão de 10 MB, leva 1.27 segundos com 404 milissegundos de latência.

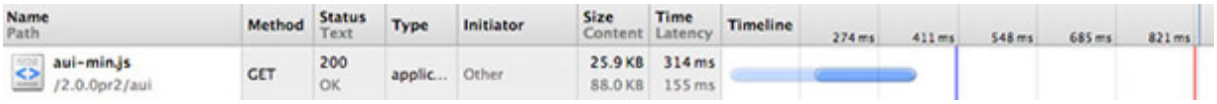

Figura 9. Requisição de um arquivo JavaScript utilizando uma CDN.

Para carregar esse arquivo, que está hospedado em uma CDN com a mesma conexão de internet, leva 314 milissegundos com 155 milissegundos de latência. Uma diminuição de 75%. Isto é apenas um arquivo, o impacto em um portal inteiro é ainda maior.

Outro ponto que contribui muito é o fato de o arquivo estar hospedado em um domínio diferente, já que os navegadores realizam um número limitado de requisições HTTP em paralelo por domínio.

Conforme ilustrado nas Figura 10 e 11, o navegador consegue realizar mais requisições paralelas ao distribuir-se os arquivos em diferentes domínios.

Figura 10. Requisições sendo feitas pelo mesmo domínio.

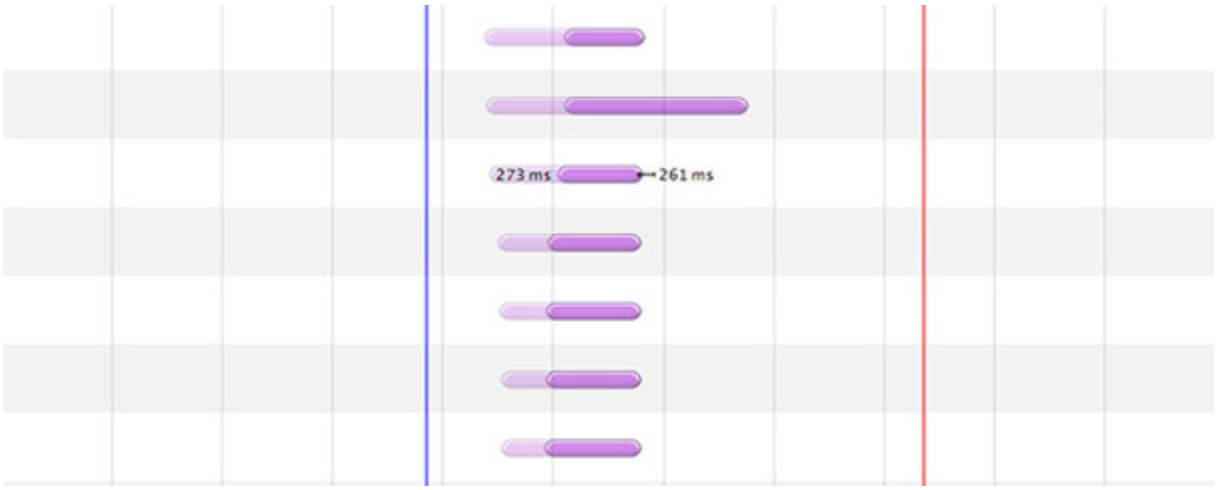

Figura 11. Requisições sendo feitas por domínios diferentes.

Conforme a configuração, esse arquivo também pode ser armazenado em cache. O que acontece é que todo navegador armazena alguns arquivos localmente depois de serem carregados, evitando assim o tráfego do mesmo conteúdo várias vezes pela rede. Por exemplo, um usuário visita um website que carrega um arquivo hospedado em uma CDN. Assim que o usuário carregar a página novamente, o navegador automaticamente armazenará em cache na

sua máquina. Depois, ao visitar outra página que também solicite aquele arquivo, o usuário não precisará fazer o download novamente, já que os armazenou em cache.

Dos cinco sites analisados, apenas um utiliza uma CDN. Desse único site que utiliza, apenas um dos sete arquivos JavaScript é carregado através dela.

Tabela 3: Análise dos sites que utilizam alguma CDN para distribuir seus arquivos.

Rank	Website	CDN
1	Caixa Econômica Federal	Não utiliza
2	Governo do Estado de São Paulo	Utiliza
3	Ministério da Fazenda	Não utiliza
4	Ministério da Educação	Não utiliza
5	Governo do Estado do Rio de Janeiro	Não utiliza

CAPÍTULO 6
O TAMANHO DAS REQUISIÇÕES HTTP

Cada byte que é trafegado via requisições HTTP passa por milhares de quilômetros pela rede até chegar no seu destino para depois ser exibido em uma página web. Por isso, é importante diminuir o *payload*, ou seja, o volume de dados trafegados. Neste capítulo são apresentados técnicas para diminuir o tamanho das requisições HTTP através da compressão de código (Seção 6.1) e da compressão de imagem (Seção 6.2).

Tabela 4: Total de bytes carregados.

Rank	Website	Tamanho
1	Caixa Econômica Federal	1008 kb
2	Governo do Estado de SP	992 kb
3	Ministério da Fazenda	233 kb
4	Ministério da Educação	108 kb
5	Governo do Estado do RJ	1600 kb

Tabela 5: Distribuição dos bytes carregados em uma página entre cada tipo de elemento

Rank	Website	HTML	CSS	JS	Flash	Imagens
1	Caixa Econômica Federal	2%	1.4%	9.7%	0.6%	86.3%
2	Governo do Estado de SP	0.8%	4.1%	6.3%	0%	88.8%
3	Ministério da Fazenda	27.5%	0%	3%	1.6%	67.8%
4	Ministério da Educação	4.5%	1.5%	13.9%	0%	80.1%
5	Governo do Estado do RJ	0.8%	5%	54.4%	17.6%	22.1%

6.1 COMPRESSÃO DE CÓDIGO

Para manter um código legível e bem documentado é preciso escrever comentários e ter cuidado com indentação. Só que nada disso importa na hora da execução e, portanto, trafegar esse volume de *bytes* é simplesmente desperdício. Por isso, é uma boa prática comprimir os arquivos, essa técnica consiste em remover todos os caracteres desnecessários do código fonte, como espaços, quebras de linha e comentários sem alterar sua funcionalidade.

6.1.1 COMPRESSÃO DE JAVASCRIPT

Sendo assim, um arquivo JavaScript com por exemplo 500 linhas de código com 16 KB, uma vez comprimido através da ferramenta UglifyJS, passa a ter 1 linha e 8 KB, uma economia de cerca de 50% de tamanho. É recomendado a integração de uma ferramenta de compressão de JavaScript no seu fluxo de trabalho como YUI Compressor, Google Closure Compiler ou UglifyJS.

Nenhum dos sites analisados comprimiam seu JavaScript.

6.1.2 COMPRESSÃO DE CSS

Comprimir os arquivos CSS é igualmente importante. Para isso, existem ferramentas como YUI Compressor, CSS Minifier ou CSSmin.js. Um arquivo com por exemplo, 1000 linhas de código e 33kb, uma vez comprimido através da ferramenta CSS Minifier, passa a ter 10 kb. Vale lembrar que alguns pré-processadores de CSS como o LESS, SASS ou Stylus possuem opções para comprimir seu arquivo na hora da compilação.

Novamente, nenhum dos sites testados comprimiam seu CSS.

6.1.3 COMPRESSÃO DE HTML

Com HTML não seria diferente, também é importante comprimí-lo, mesmo que a

compressão não tenha o mesmo impacto do que a compressão de JavaScript e CSS. Para realizar esse procedimento existem ferramentas como HTML Compressor e HTML Minifier. Como visto na Tabela 6, apenas um dos cinco sites avaliados comprimiam seu HTML.

Tabela 6: Análise dos sites que realizam compressão dos arquivos

Rank	Website	Comprime HTML	Comprime CSS	Comprime JS
1	Caixa Econômica Federal	Não	Não	Não
2	Governo do Estado de SP	Não	Não	Não
3	Ministério da Fazenda	Não	Não	Não
4	Ministério da Educação	Não	Não	Não
5	Governo do Estado do RJ	Sim	Não	Não

6.2 COMPRESSÃO DE IMAGEM

Websites não são feitos apenas de código, muitas imagens em diferentes formatos são carregadas em aplicações web também. Arquivos de imagens possuem muitos KBs de informação desnecessária no contexto da web. Por exemplo, uma foto no formato JPEG contém uma porção de metadados *Exif* colocados pela câmera como a data da foto, modelo da câmera, local onde a foto foi tirada etc. Já o formato PNG possui uma série de informações sobre cores, metadados e, às vezes, até uma miniatura da imagem embutida. Só que nada disso é importante para a renderização da imagem no navegador, ocorrendo, então, um gasto desnecessário no tráfego desses bytes pela rede.

6.2.1 COMPRESSÃO LOSSLESS

Existem ferramentas de otimização de imagens como Kraken.io, ImageOptim e Riot que removem todas as informações desnecessárias, gerando um novo arquivo muito mais enxuto, porém, visualmente idêntico. Elas realizam uma compressão sem perda de qualidade, chamada de *lossless*. Um exemplo desse tipo de compressão é o formato ZIP.

6.2.2 COMPRESSÃO LOSSY

Outra forma de otimizar imagens é comprimí-las com certas perdas visuais aceitáveis, ou compressão *lossy*. Esse tipo de algoritmo é comumente usado para sons, imagens e vídeo. No lado esquerdo da Figura 12 é possível ver um arquivo PNG sem compressão, com a dimensão de 179x300 pixels e um tamanho de 72 kb.

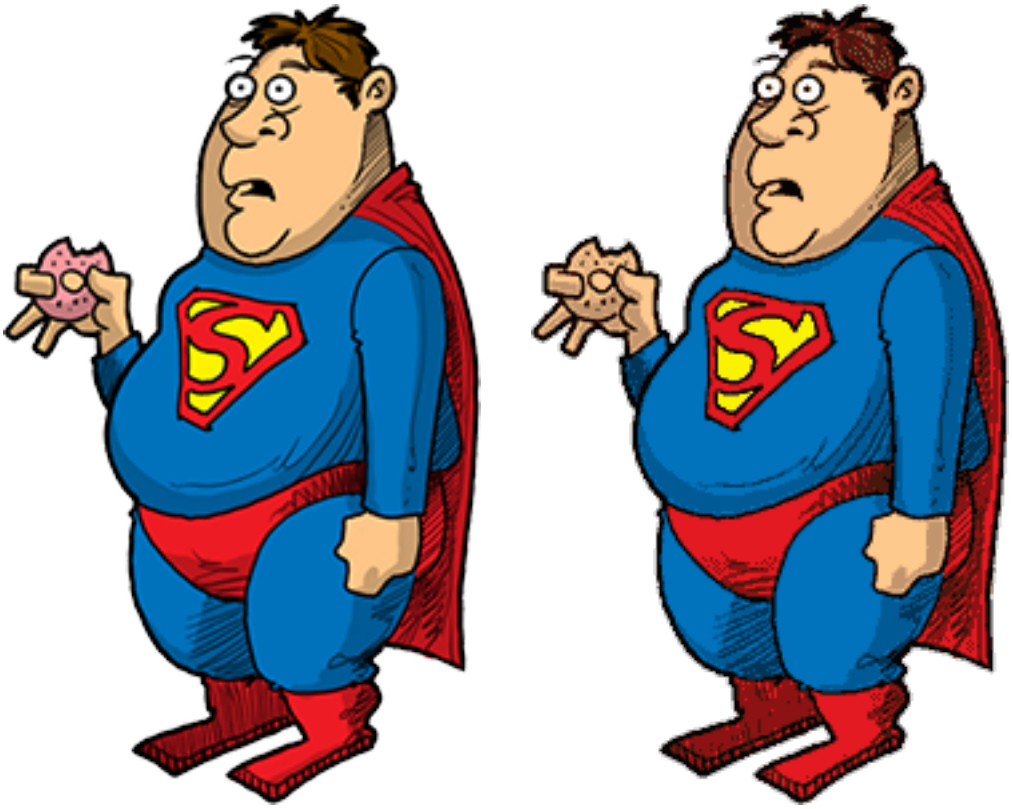

Figura 12. (Esquerda) Imagem no formato PNG sem compressão
(Direita) Imagem no formato PNG com compressão

Através do uso da ferramenta ImageAlpha, é possível reduzir o número de cores desse PNG de 16.777.216 para 16, o que causa uma redução de 90% no tamanho do arquivo, depois de realizada a compressão na imagem da esquerda (Figura 12), a imagem da direita (Figura 12) passa a pesar apenas 7 KB. Note que existe uma perda de qualidade visível, mas o benefício em performance é tão grande que vale a pena. Para otimização *lossy* de outros formatos existem as ferramentas JPEGMini e Smush.it.

CAPÍTULO 7
O NÚMERO DE REQUISIÇÕES HTTP

A maioria dos navegadores realiza apenas 3 requisições simultâneas por domínio, o que significa que quanto menor o número de requisições, melhor. Só que nem por isso é preciso sacrificar a experiência do usuário reduzindo o número de componentes na interface ou simplificando o design. É possível construir páginas web com interfaces ricas e ainda assim conseguir rápidos tempos de resposta. Neste capítulo serão apresentados técnicas para minimizar o número de requisições através da concatenação de código (Seção 7.1) e concatenação de imagem (Seção 7.2).

Tabela 7: Número de requisições

Rank	Website	Requisições
1	Caixa Econômica Federal	120
2	Governo do Estado de SP	65
3	Ministério da Fazenda	37
4	Ministério da Educação	14
5	Governo do Estado do RJ	87

7.1 CONCATENAÇÃO DE CÓDIGO

Um problema recorrente nas aplicações web é o número de requisições JavaScript e CSS que são feitas, na Tabela 8 é mostrado como esse número é elevado em algumas páginas.

Tabela 8: Número de requisições JavaScript e CSS

Rank	Website	Número de JS's	Número de CSS's
1	Caixa Econômica Federal	9	3
2	Governo do Estado de SP	7	6
3	Ministério da Fazenda	1	0
4	Ministério da Educação	1	1
5	Governo do Estado do RJ	21	16

Souders também reforça essa questão:

> JavaScript e CSS são usados na maioria dos sites hoje em dia. Engenheiros de Front-end precisam escolher entre colocar seu código JavaScript e CSS "inline" (incorporado no documento HTML) ou incluí-los de um script e folha de estilo externos. Em geral, utilizar scripts e folhas de estilo externos são melhores para performance. Entretanto, se você seguir a recomendação da engenharia de software e modularizar seu código quebrando-o em diversos arquivos pequenos, você piora a performance porque cada arquivo resulta em uma requisição HTTP adicional (Souders, 2007, p33).

Para resolver esse dilema não é preciso abdicar da modularização ao colocar o código em um só arquivo ou então abdicar da performance ao separar o código em diversos arquivos. Existe uma solução para conseguir o melhor dos dois cenários, basta acrescentar no fluxo de trabalho uma ferramenta de build que concatenará diversos arquivos em um só. Assim, é possível desenvolver com diferentes arquivos e na hora de enviar para o servidor concatenar todos em um só.

7.1.1 CONCATENAÇÃO DE JAVASCRIPT

Para que o processo de concatenação aconteça sem problemas, é importante garantir que o código esteja sem problemas de sintaxe. Isso porque um simples ponto-e-vírgula que esteja faltando pode causar um problema em todos os outros arquivos. Para concatenar arquivos JavaScript existem ferramentas como Grunt e YUI Compressor.

7.1.2 CONCATENAÇÃO DE CSS

No caso do CSS, ocorre o mesmo, ao misturar diversos arquivos em um só, é preciso garantir que eles estejam bem escritos, a fim de evitar que um simples erro em um arquivo se propague por todos os outros. Para concatenar arquivos CSS existem ferramentas como Grunt, Minify e CSS Merge.

7.2 CONCATENAÇÃO DE IMAGEM

O maior de todos os problemas é com relação as imagens, até porque geralmente são os elementos mais frequentes em qualquer página web conforme mostrado na Tabela 9.

Tabela 9: Número de requisições de imagens

Rank	Website	Número de Imagens
1	Caixa Econômica Federal	82
2	Governo do Estado de SP	65
3	Ministério da Fazenda	29
4	Ministério da Educação	8
5	Governo do Estado do RJ	43

7.2.1 CSS SPRITES

O método mais comum para minimizar esse problema é chamado de CSS Sprites, essa antiga técnica que começou a ser difundida na época do videogame Atari, consiste em agrupar diversas imagens em uma só e depois posicioná-las através da propriedade do CSS.

Figura 13. Agrupamento de imagens em um Sprite na homepage do Google.

```
.icon-foo {
  background-image: url('mySprite.png');
  background-position: -10px -10px;
}

.icon-bar {
  background-image: url('mySprite.png');
  background-position: -5px -5px;
}
```

Figura 14. Posicionamento de um Sprite através da propriedade background-position.

7.2.2 DATA URIs

Uma técnica alternativa para o uso de CSS Sprites é a técnica de Data URIs. A fim de reduzir o número de requisições de imagens, é possível transformar uma imagem inteira em texto através de uma codificação base64 como visto na Figura 15.

```
.icon-foo {
  background-image: url('foo.png');
}
```

```
.icon-foo {
  background-image: url('data:image/png;
base64,iVBORw0KGgoAAAANSUhEUgAAAAEAAAABA
QMAAAAl21bKAAAAA1BMVEUAAACnej3aAAAAAXRST
lMAQObYZgAAAApJREFUCNdjYAAAAAIAAeIhvDMAA
AAASUVORK5CYII%3D');
}
```

Figura 15. (Esquerda) Carregamento de uma imagem comum através de CSS (Direita) Carregamento de uma imagem em base64 através de CSS.

Tanto esse método quanto CSS sprites precisam de ferramentas de build para serem de fácil manutenção. Esse método tem a vantagem de não exigir reposicionamento manual do Sprite já que mantém as imagens em arquivos individuais durante o desenvolvimento.

Entretanto, tem a desvantagem de aumentar consideravelmente o tamanho do HTML/CSS se possuir imagens grandes. Não é recomendado utilizar esse método se não estiver utilizando alguma técnica de compressão do HTML/CSS durante as requisições HTTP, já que a sobrecarga de tamanho pode anular os ganhos de velocidade sobre minimizar o número de requisições HTTP.

CAPÍTULO 8

GUIA COLABORATIVO SOBRE PERFORMANCE

As informações apresentadas nos capítulos precedentes desta monografia serviram de base para a elaboração de um projeto de código aberto e sem fins lucrativos denominado "Como Perder Peso (no browser)" <http://browserdiet.com/pt>.

Durante o processo de escrita dessa monografia, foram identificadas quatro referências sobre performance. Dois livros do autor Steve Souders e dois guias online, um da Yahoo! e outro da Google. Essas referências não se encontram disponíveis no idioma português e também não são abertas para colaboração, o que motivou a criação de um guia colaborativo sobre performance visando divulgar esse conhecimento para desenvolvedores e webdesigners.

Figura 16. Capa do site http://browserdiet.com/pt

Para realizar esse projeto, foram estabelecidas algumas parcerias. A primeira pessoa convidada foi a designer Briza Bueno (Americanas.com), que elaborou uma identidade visual atrativa aos usuários em conjunto com o ilustrador Scott Johnson (ExtraLife). Enquanto isso, Iraê Lambert (Yahoo!) auxiliou na criação do site. Após finalizada a estrutura do site, os

desenvolvedores Davidson Fellipe (Globo.com), Giovanni Keppelen (Peixe Urbano) e Jaydson Gomes (Terra) foram convidados a auxiliar na elaboração do conteúdo desse guia colaborativo, no qual foi escrito com base nessa monografia. Finalizado o conteúdo, iniciou-se a revisão de Marcel Duran (Twitter), Renato Mangini (Google) e Sérgio Lopes (Caelum). Por fim, com o auxílio de Mike Taylor (Opera), o conteúdo que estava somente no idioma português foi traduzido para Inglês e lançado na web através do domínio browserdiet.com.

O projeto foi lançado em 11/03/2013. Durante os dez primeiros dias, 83.097 pessoas visitaram o site, sendo estas visitas oriundas de 157 países diferentes. O projeto foi compartilhado por pessoas em todo lugar no mundo como exemplificam os tweets listados na Figura 17. Repercussão do projeto no Twitter Foram feitas mais de 160 contribuições através do Github, sendo uma delas a tradução para o idioma espanhol. Neste período também iniciaram as traduções para italiano e polonês. Esses dados indicam o poder da disseminação da informação na web e da colaboração em projetos de código aberto.

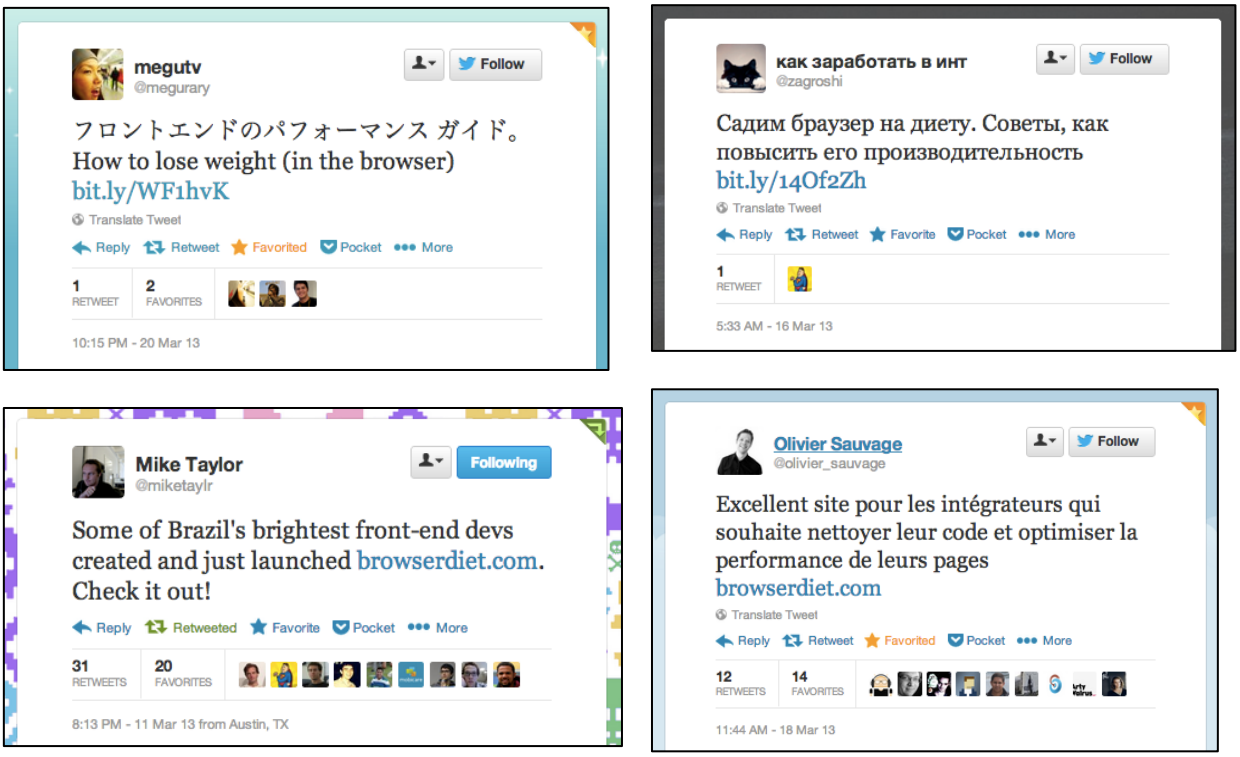

Figura 17. Repercussão do projeto no Twitter

CAPÍTULO 9
CONCLUSÃO

Essa monografia apresentou a importância da performance no lado do cliente em aplicações web. O objetivo foi enumerar as principais técnicas e suas respectivas ferramentas para apoiar a melhoria da performance de páginas web. Foi ilustrado o impacto de cada uma das técnicas no mundo real, exemplificando sua aplicação nos cinco websites do governo brasileiro mais acessados no país.

A principal conclusão é que as técnicas discutidas podem melhorar muito a performance dos sites. Os resultados sobre os sites estudados indicam as deficiências daquelas aplicações, e as soluções para os problemas foram listadas nesta monografia.

REFERÊNCIAS BIBLIOGRÁFICAS

Souders, Steve. *High Performance Web Sites.* O'Reilly Media, 2007.

Souders, Steve. *Even Faster Websites*. O'Reilly Media, 2009.

Jain, Raj. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

SITES:

Yahoo! Best Practices for Speeding Up Your Web Site.
Disponível em: <http://developer.yahoo.com/performance/rules.html>
Acesso em: 22.jan.2013

Google Web Performance Best Practices.
Disponível em: <https://developers.google.com/speed/docs/best-practices/rules_intro>
Acesso em: 22.jan.2013

Stefanov, S. YSlow 2.0. Beijing, 2008. [Slides da Palestra].
Disponível online: <http://www.slideshare.net/stoyan/yslow-20-presentation>. Acessado em: 13.mar.2013.

Cutler, B. Firefox & Page Load Speed - Part II.
Disponível online: <http://blog.mozilla.org/metrics/2010/04/05/firefox-page-load-speed-%E2%80%93-part-ii/>. Acessado em: 13.mar.2013.

Linden, G. Make Data Useful. [Slides da Palestra]
Disponível online: <http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazoncom>. Acessado em: 13.mar.2013.

Linden, G. Marissa Meyer at Web 2.0.
Disponível online: <http://glinden.blogspot.com.br/2006/11/marissa-mayer-at-web-20.html>. Acessado em: 13.mar.2013.

Schurman, E. Performance Related Changes and their User Impact. [Palestra]
Disponível online: < http://blip.tv/oreilly-velocity-conference/velocity-09-eric-schurman-and-jake-brutlag-performance-related-changes-and-their-user-impact-2292767>. Acessado em: 13.mar.2013.

Gomes, J. Extreme Web Performance. [Slides da Palestra].
Disponível online: < http://jaydson.org/talks/x-web-performance/#slide-6>. Acessado em: 13.mar.2013.

Keppelen, G. Performance Front-end. Front In. Maceió, 2012. [Slides da Palestra]. Disponível online: <http://www.slideshare.net/keppelen/performance-frontend-front-in-macei>. Acessado em: 13.mar.2013.

TÉCNICAS PARA MELHORIA DE PERFORMANCE
EM APLICAÇÕES WEB NO LADO DO CLIENTE

Aprovado em ____/_____/_____

BANCA EXAMINADORA

Prof. Marino Pimentel (orientador)

Prof. Márcio Barros

Prof. Gleison Santos

O autor deste Projeto autoriza a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardados os direitos autorais conforme legislação vigente.

Rio de Janeiro, _____de_____ de_____.

Aluno Zeno Rocha