UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE MATEMÁTICA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LÉON AUGUSTO DE ARAÚJO PEREIRA

BOAS PRÁTICAS PARA A MELHORIA DE DESEMPENHO DE PÁGINAS WEB: UM ESTUDO SOBRE HTTP/1.1 E HTTP/2

RIO DE JANEIRO

2018

LÉON AUGUSTO DE ARAÚJO PEREIRA

BOAS PRÁTICAS PARA A MELHORIA DE DESEMPENHO DE PÁGINAS WEB: UM ESTUDO SOBRE HTTP/1.1 E HTTP/2

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.

RIO DE JANEIRO

2018

CIP - Catalogação na Publicação

PP436b

Pereira, Léon Augusto de Araújo
Boas práticas para a melhoria de desempenho de páginas Web: Um estudo sobre HTTP/1.1 e HTTP/2 / Léon Augusto de Araújo Pereira. -- Rio de Janeiro, 2018.
106 f.

Orientador: Paulo Henrique de Aguiar Rodrigues.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Performance. 2. Web. 3. HTTP. I. Rodrigues, Paulo Henrique de Aguiar , orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

LÉON AUGUSTO DE ARAÚJO PEREIRA

BOAS PRÁTICAS PARA A MELHORIA DE DESEMPENHO DE PÁGINAS WEB: UM ESTUDO SOBRE HTTP/1.1 E HTTP/2

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em _____ de _____ de _____ .

BANCA EXAMINADORA:

_____

Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.

_____

Prof. Valeria Menezes Bastos, D.Sc.

_____

Prof. Claudio Miceli de Farias, D.Sc.

AGRADECIMENTOS

Agradeço aos meus pais que batalharam muito para me oferecer uma educação de qualidade. Meu eterno agradecimento a todos os meus amigos, que deram uma contribuição valiosa para a minha jornada acadêmica. Obrigada pelos conselhos, palavras de apoio, puxões de orelha e risadas. Só tenho a agradecer e dizer que se hoje esse TCC existe é graças ao apoio de todos.

Agradeço aos professores do departamento que acompanharam a minha jornada acadêmica de perto. Obrigado pelo apoio e confiança em todos os projetos que participei, parte desse aprendizado está refletido nesse TCC. Sou grato principalmente ao professor Aguiar pelas longas conversas, puxões de orelha e, principalmente, pela paciência. Por fim, agradeço à UFRJ, que me proporcionou a chance de expandir os meus horizontes. Obrigado pelo ambiente livre, amigável e repleto de oportunidades.

"It’s up to us as developers to guard our users’ interests. At your site, evangelize performance. Implement these techniques. Fight for a faster user experience. If your company doesn’t have someone focused on performance, appoint yourself to that role. Performance vigilante—I like the sound of that."

Steve Souders

RESUMO

A qualidade da experiência de navegação de um usuário está ligada diretamente à velocidade com a qual páginas da web são baixadas a partir dos servidores de hospedagem e, em seguida, exibidas no navegador. Mesmo com os principais sites, cada vez mais rápidos, ainda é comum encontrar sites menores com páginas que carregam de forma extremamente lenta, com imagens pesadas, elementos e layout não otimizados para web Os desenvolvedores não seguem recomendações básicas para o desenvolvimento de páginas web e configurações de ambiente, deixando de proporcionar uma melhor experiência para o usuário. A partir desse problema, o projeto se foca no desempenho web, abordando tanto HTTP/1.1 como HTTP/2, além de identificar as 10 principais técnicas e boas práticas para a melhora do desempenho web, além de fornecer uma análise do site do Departamento de Ciência da Computação da UFRJ.

Palavras-chave: Web. HTTP. Performance.

ABSTRACT

The quality of a user browsing experience is directly linked to the speed at which web pages are downloaded from hosting servers and then displayed at the browser. Checking the fastest growing web sites, it is common to find smaller sites with pages that load extremely slowly, with heavy images, non-web-optimized elements and layout. It is clear that site developers have not observed basic recommendations for webpage development, causing performance degradation in page access. This work identifies 10 main techniques and good practices for improving HTTP/1 and HTTP/2 web performance. As a case study, UFRJ Department of Computer Sciencet website is analysed and guidelines for improving its performance is obtained.

Keywords: Web. HTTP. Performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

UFRJ	Universidade Federal do Rio de Janeiro
DCC	Departamento de Ciência da Computação
QoE	Quality of Experience
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
TCP	Transmission Control Protocol
RFC	Request for Comments
RTT	Round-Trip time
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
MIME	Multipurpose Internet Mail Extensions
GIF	Graphics Interchange Format
JPG	Joint Photographic Experts Group
CDN	Content Delivery Network
IP	Internet Protocol
DNS	Domain Name System
JSON	JavaScript Object Notation

SUMÁRIO

1 INTRODUÇÃO

A qualidade da experiência de navegação de um usuário está ligada diretamente à velocidade com a qual páginas da web são baixadas a partir dos servidores de hospedagem e, em seguida, exibidas no navegador. Mesmo com os principais sites, cada vez mais rápidos, ainda é comum encontrar sites menores com páginas que carregam de forma extremamente lenta, com imagens pesadas, elementos e layout não otimizados para web e para o dispositivo em que está a ser visualizado.

O motivo da falta de desempenho, para a maioria dos sites, não reside em limitações de recurso, mas em seu código (SOUDERS, 2006). Os desenvolvedores não seguem recomendações básicas para o desenvolvimento de páginas web e configurações de ambiente, deixando de proporcionar um melhor QoE para o usuário. Fica claro que os desenvolvedores não possuem uma visão mais aprofundada das questões que afetam o desempenho da web e é necessário um conhecimento mais detalhado do protocolo, suas características e seu funcionamento, fato já apontado por outros estudos (SAXCÉ; OPRESCU; CHEN, 2015).

Este trabalho teve como principal objetivo identificar boas práticas e técnicas para desenvolvedores diminuírem o tempo de carregamento das páginas nos navegadores do usuário, através de modificação de código e de manuseio de componentes estáticos da página. Foram estudadas e identificadas 10 boas práticas:

- Técnicas básicas de redução de requisições como Image Maps, CSS Sprites, Inline Images e Otimização de imagens.
- Boas práticas para o uso de CDN.
- Adicionar cabeçalho de expiração aos componentes.
- Técnica de compressão
- Técnica de minificação e ofuscação
- Boas práticas para organização de componentes estáticos na página.

- Uso de componentes estáticos externos.
- Redução de DNS Lookup
- Boa práticas para redução de redirecionamentos.
- Configuração de ETag.

Para exemplificar a importância e eficácia das boas práticas identificadas, a análise da página em produção do próprio Departamento de Ciência da Computação da UFRJ foi realizada e facilmente identificadas melhorias para seu desempenho. No processo de análise , foram desenvolvidos scripts e usadas as ferramentas PageSpeed do Google e YSlow do Yahoo. Este estudo de caso contempla uma metodologia que pode ser replicada na análise de outros sites e páginas.

Como o uso da Internet tem se tornado lugar comum e muitas pessoas estão até criando suas próprias páginas pessoais além das páginas corporativas, a melhoria de acesso à web tem impacto e significância para todos. Evidentemente o desempenho de acesso a sites dependerá também de congestionamentos e da disponibilidade de banda entre o servidor e o usuário, mas a construção de uma página com critérios objetivando o seu carregamento rápido no navegador do usuário deve ser perseguida, independentemente de outros fatores. Este documento pode ser um importante referencial para disciplinas voltadas para o desenvolvimento de páginas web.

Ao utilizar as técnicas e boas práticas relatadas será necessária uma compreensão aprofundada do HTTP, seu contexto histórico e as atualizações que o protocolo recebeu. Cada nova versão do protocolo trouxe muitas otimizações em relação à performance, mas não retirou totalmente da web versões anteriores. Mesmo para alguns dos principais sites da web, a atualização do protocolo é visto como uma solução para problemas bem específicos, não uma melhoria essencial para todas suas páginas.

A monografia está subdividida em cinco capítulos. No Capítulo 2, é realizada uma revisão do HTTP/1.1, na qual são abordados os conceitos básicos do protocolo, como a importância do GET condicional para o desempenho do protocolo; uma breve história sobre suas versões, e suas atualizações focadas no desempenho nos últimos anos, como

conexão persistente, pipelining e múltiplas conexões TCP.

No Capítulo 3, são apresentadas em cada seção técnicas e boas práticas para melhorar o desempenho do HTTP/1.1, que focam na importância de se reduzir a quantidade de requisições e utilizar técnicas para redução de bytes enviados. Para cada técnica e boa prática, serão apresentados exemplos os quais elucidarão a falta de desempenho e às soluções para a melhora.

O Capítulo 4 apresenta a nova versão do protocolo, o HTTP/2, em que são discutidas as limitações do HTTP/1.1 que culminaram com a implementação de um novo protocolo e todas as novas funcionalidades que o HTTP/2 traz para melhorar desempenho de uma página web. Capítulo em que é realizada uma reflexão sobre o futuro da web que está a ser dividida entre as duas versões do protocolo.

O Capítulo 5 evidencia um estudo de caso realizado no site do Departamento de Ciência Computação da UFRJ onde é feita uma análise da performance da página, utilizando todas as técnicas mostradas no Capítulo 3 como base. Em formas de anexo, é entregue um relatório completo do que pode ser otimizado na página web do departamento e, por fim, uma discussão acerca das vantagens do uso do HTTP/2 para o site do departamento.

O Capítulo 6 trata das considerações finais e trabalhos futuros. Ao final do trabalho são adicionados anexos relativos ao estudo de caso. O anexo A possui um modelo cascata das requisições à página web do departamento, mostrando o tempo de carregamento da mesma em relação a cada componente, o anexo B até o anexo G são componentes que devem ser otimizados na página do departamento.

2 O PROTOCOLO HTTP

Este capítulo trará uma visão geral acerca do HTTP, com o intuito de evidenciar seu modo de funcionamento, além de conhecer alguns detalhes que tornaram o protocolo tão importante para a web. Os navegadores e os servidores, como todos os aplicativos baseados na web, comunicam-se utilizando como base o protocolo Hypertext Transfer Protocol (HTTP), o qual tem como função se comunicar com navegadores e servidores pela internet. A RFC 2616 (FIELDING et al., 1999) descreve a versão 1.1, mais comum hoje, tendo sido especificada pelo World Wide Web Consortium (W3C) e a Internet Engineering Task Force (IETF). Uma nova versão foi desenvolvida, todavia, essa última versão ainda não tem sido amplamente utilizada, por questões que serão abordadas e realçadas no capítulo 4.

O protocolo é do tipo cliente/servidor e permite acessar recursos pelo localizador uniforme de recursos (URL) no qual é possível encontrar recursos em um ou mais servidores de acordo com esse identificador. O acesso a esses recursos é baseado em um identificador de recurso uniforme (URI), logo, cada recurso do servidor tem um nome para que os clientes possam acessar, portanto, esse registro é único como endereços postais. Cada arquivo possui um caminho único no servidor para ser acessado pelo mundo. Uma requisição HTTP, especificando uma determinada URL, é enviada ao servidor que hospeda essa URL. O servidor, por sua vez, gera uma resposta HTTP de modo a finalizar a interação. O protocolo foi criado com a intenção de ser leve e utilizar um formato de texto simples (SOUDERS, 2006).

Atualmente o HTTP move a maior parte das informações contidas na web, sejam imagens JPEG, arquivos de filmes, arquivos de áudio, scripts baseados em diversas linguagens e arquivos de texto. O uso do HTTP permitiu o desenvolvimento de aplicativos baseados unicamente nesse protocolo. A comunicação HTTP ocorre por meio de sessões confiáveis do protocolo Transmission Control Protocol (TCP) que foi originalmente definido na RFC 793 de 1981, garantindo ao desenvolvedor que não será necessário se preocupar com as comunicações (HTTP) destruídas, duplicadas ou distorcidas.

2.1 CONCEITOS BÁSICOS

Os servidores web utilizam o protocolo HTTP, chamados de servidores HTTP, que armazenam dados os quais podem ser acessados por clientes HTTP. O cliente mais comum é o navegador web (Google Chrome, Firefox, IE etc.). Esses navegadores solicitam arquivos enviados via HTTP de servidores para clientes (GOURLEY et al., 2002). Abaixo modelo na Figura 1.

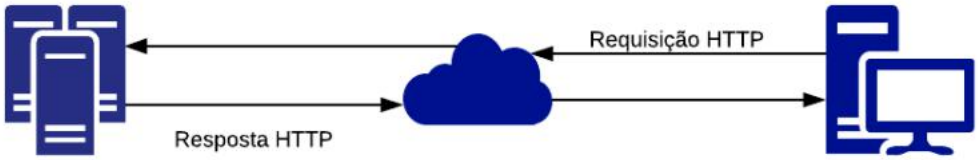


Figura 1: Modelo de requisição/resposta HTTP

Os recursos são hospedados em servidores web, dentre os quais arquivos estáticos, como mencionado acima, que não dependam do usuário para ser modificado em tempo de execução pelo navegador. Por outro lado, os recursos HTTP poderão ser também dinâmicos, como software que gera conteúdo sob demanda, com base na identidade do usuário ou informações solicitadas em uma data e horário. Busca em bases de dados ou compra em lojas on-line que modificam seus dados de acordo com a época do ano e a quantidade de itens são bons exemplos de conteúdo dinâmico (GOURLEY et al., 2002).

As requisições HTTP são formatadas em blocos e denominadas de métodos, estes que são suportados, dentre os quais: GET, POST, HEAD, PUT, DELETE, OPTIONS e TRACE (FIELDING et al., 1999).

O método GET permite recuperar um recurso existente no servidor e é bastante utilizado para testes de performance pela fácil visualização, pois facilita a análise uma vez que os componentes estão em seu URI e tem a possibilidade do uso de requisições condicionais que serão mencionadas posteriormente. Uma requisição do tipo GET, como mostrado na Figura 2, inclui uma URI seguida dos campos de cabeçalho. A resposta contém código de status dela, campos de cabeçalho e os dados renderizados em tela (FIELDING et al., 1999).

```
GET /index.php?jat3action=gzip&jat3type=js&jat3file=t3-assets%2Fjs_310ac.js HTTP/1.1

Accept:*/*
Accept-Encoding:gzip, deflate
Accept-Language:pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control:no-cache
Connection:keep-alive
Cookie:__utmz=263428936.1510337697.1.1.utmcsr=(direct)
Host:www.dcc.ufrj.br
Pragma:no-cache
Referer:http://www.dcc.ufrj.br/
User-Agent:Mozilla/5.0 (X11; Fedora; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
```

Figura 2: Resposta HTTP

Toda mensagem de resposta de uma requisição HTTP contém um código de status de três dígitos que informa ao cliente se a requisição teve sucesso, se falhou, foi redirecionada ou outra ação. Para a análise de performance, o feedback ou o retorno do estado é valioso, permitindo identificar a geração de outras requisições subsequentes, para acesso ao recurso, o que poderá impactar desfavoravelmente o desempenho, com aumento do tempo de resposta. Abaixo alguns dos principais códigos de status.

- 1xx - Informativo
- 2xx - Sucesso
- 3xx - Redirecionamento
- 4xx - Erro de cliente
- 5xx - Erro de servidor

O protocolo HTTP utiliza o paradigma do cliente/servidor sem uma conexão persistente até sua versão 1.0. Nesse tipo de conexão, cada recurso ou arquivo requisitado é transportado por uma conexão TCP, encerrada imediatamente após o retorno da resposta. Logo, novas conexões TCP deverão ser criadas para transferências de múltiplos recursos. Em um cenário com conexões não persistentes, múltiplas conexões paralelas (comum até 10) podem ser abertas para diminuir o tempo de resposta de acesso a uma página.

2.2 GET CONDICIONAL

Cada vez que são feitas múltiplas requisições a uma URI, ainda que pelo mesmo cliente, ocorre a transferência de todo conteúdo novamente. Para evitar essa ineficiência, caso o recurso não tenha sido modificado na versão 1.1, alguns campos de cabeçalho foram adicionados nas versões mais atuais do protocolo, criando o conceito de requisições condicionais (SOUDERS, 2006). Se o navegador tem uma cópia do componente na memória, mas não tem a certeza se este ainda é válido, então é feita uma requisição condicional.

Os componentes recebidos, na última requisição, ficam armazenados em memória cache, cujo objetivo é eliminar a necessidade de enviar requisições ou respostas completas nos casos em que o navegador já possui os dados atualizados de acordo com o servidor.

Com o GET condicional, o navegador tem a possibilidade de informar, no campo "If modified since"do cabeçalho, a data em que foi acessado no servidor. O servidor, por sua vez, informa se essa realmente é a versão mais recente.

Caso a cópia em cache ainda seja válida, o navegador usará essa cópia, que resultará em menos transferências de dados, caso contrário, será novamente transferido para atualizar a versão que está em cache no navegador. A Figura 3, a seguir, mostra dois casos. No primeiro, o dado não foi modificado, logo o GET é respondido com código 304, uma resposta padrão mínima sem qualquer conteúdo no corpo da resposta. No segundo caso, foi modificado, dessa forma, a resposta leva o arquivo para ser armazenado em cache (FIELDING; RESCHKE, 2014).

O uso de requisições condicionais auxilia a diminuir a taxa de transferência entre servidores e clientes, contudo, não elimina a necessidade de, pelo menos, mais uma requisição para executar a verificação de validade. Para evitar esse teste de validade, criou-se o campo expiração no cabeçalho, o qual indica a data até o momento em que o conteúdo transferido poderá ser utilizado sem checar sua validade, de modo a evitar requisições desnecessárias. Em contrapartida, a técnica possui riscos, visto que o servidor deverá se comprometer a não atualizar o dado pelo tempo definido (FIELDING; RESCHKE, 2014).

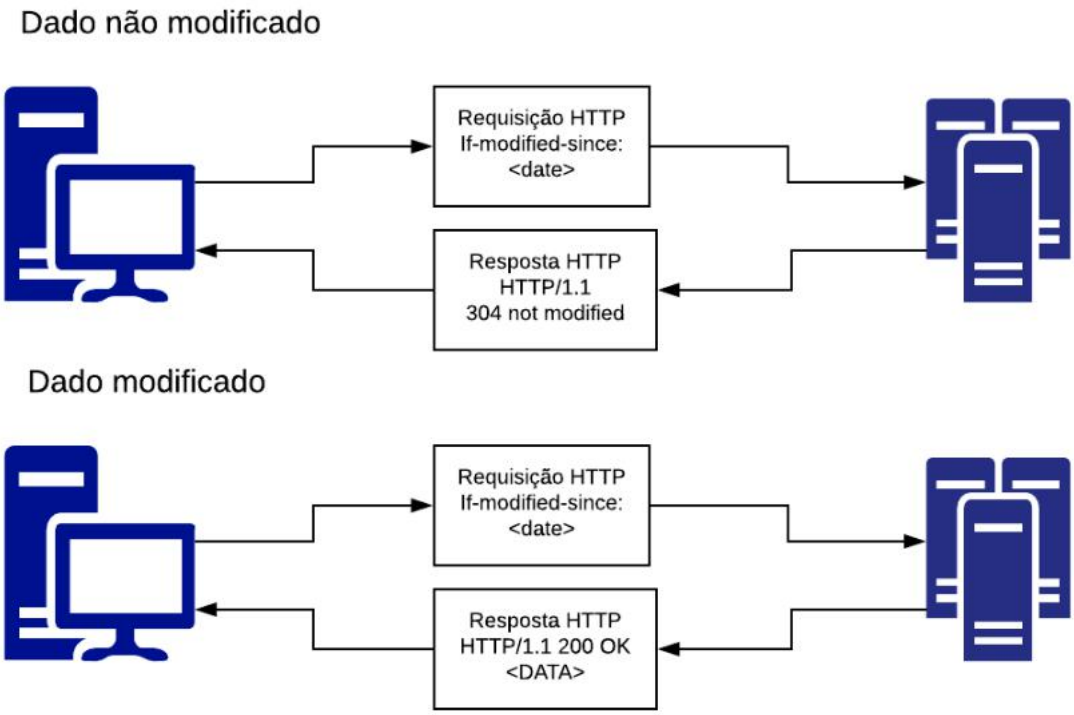


Figura 3: GET Condicional

2.3 BREVE HISTÓRIA

O HTTP possui inúmeras versões, algumas delas ainda em uso, o que dificulta o suporte dos aplicativos. A seguir, algumas dessas versões.

HTTP 0.9: versão de protótipo com falhas graves no projeto. Suporta tão somente o método GET, sem conteúdo de multimídia. Deve ser utilizada apenas para interoperabilidade entre serviços legados (GOURLEY et al., 2002).

HTTP 1.0: responsável por promover o uso em larga escala do World Wide Web. Foi a primeira versão implementada de forma ampla. Adicionou números de versão, cabeçalhos, métodos além do GET e multimídia. Essa versão foi responsável pelos conceitos de boas práticas para web (GOURLEY et al., 2002).

HTTP 1.0+: conforme mencionou-se anteriormente, o HTTP encontrou alguns problemas de performance em seu início; e, nessa versão, foram incorporadas conexões persistentes, ou keep-alive, suporte de hospedagem virtual e suporte para conexão

via proxy. Essa versão não é oficial e é vista como uma extensão do HTTP 1.0 cujo intuito é o de atender algumas necessidades do mercado, além de expandir o uso da web (GOURLEY et al., 2002).

HTTP/1.1: considerada a versão atual do HTTP, focou-se em corrigir algumas falhas arquitetônicas do projeto inicial, que representou melhorias de desempenho e de consumo (GOURLEY et al., 2002).

HTTP/2.0: concentra-se nas otimizações de performance, com compressão automática e uso de criptografia por padrão (BELSHE; THOMSON, 2015). Desde 2015, é implementada na maioria dos navegadores.

2.4 CONEXÃO PERSISTENTE

A conexão HTTP persistente, ou keep-alive, foi uma das principais melhorias inseridas na versão 1.1 do protocolo (FIELDING et al., 1999), logo, deve ser observada mais detalhadamente. Em uma conexão HTTP persistente, a sessão TCP, estabelecida entre o cliente e o servidor, é mantida aberta, podendo ser utilizada por todas as interações HTTP entre os dois pontos.

O estabelecimento de uma conexão TCP envolve uma negociação inicial chamada three-way handshake, em que as primitivas iniciais envolvem uma troca de três mensagens: pedido de estabelecimento de conexão, resposta ao pedido e confirmação do recebimento da resposta. Após o estabelecimento da conexão TCP, o tempo entre o envio de uma requisição HTTP e o recebimento da resposta HTTP haverá outro tempo de espera o qual envolve o tempo de propagação ida e volta, mais aquele para processamento no servidor, além do tempo de transmissão do recurso da URL. O tempo médio de transmissão de um arquivo dependerá da taxa média de transmissão TCP, a qual dependerá de vários fatores como o congestionameno da rede, perda de pacotes e velocidade do enlace (WANG et al., 2013).

Finalmente, um procedimento simples para melhorar o desempenho será o de reutilizar a conexão já existente com keep-alive, visto que permite eliminar o segundo three-way handshake e evitará outro início de estabelecimento de conexão TCP (Slow

Start). A conexão sempre iniciará com uma taxa baixa e atingirá somente uma taxa média para conexão após vários tempos de ida e volta (Congestion Avoidance). O tempo de transferência dos objetos dependerá da taxa média da conexão TCP a ser usada, a qual não é considerada na latência. A latência total para N requisições é (N-1) x RTT quando a conexão é persistente. Em grandes sites (LUNDIN; GARZA, 2017), o número de componentes por página é de cerca de 90, de forma que o valor de N médio será dessa ordem. O HTTP ter se tornado persistente foi uma otimização imprescidível para a melhoria de desempenho do protocolo.

2.5 PIPELINING

O HTTP persistente proporcionou a reutilização de uma única conexão para enviar várias requisições, contudo, em sua implementação isso implicou na criação de uma fila que se refere ao First In, First Out (FIFO), em que o primeiro a entrar é o primeiro a sair, ou seja, tratamento sequencial em ordem. O protocolo segue o fluxo pelo qual envia a requisição, aguarda a resposta completa e envia a próxima requisição, como visto na Figura 4. O pipelining se tornou uma otimização para gerenciar essa fila de requisições e por consequência o fluxo de trabalho do HTTP.

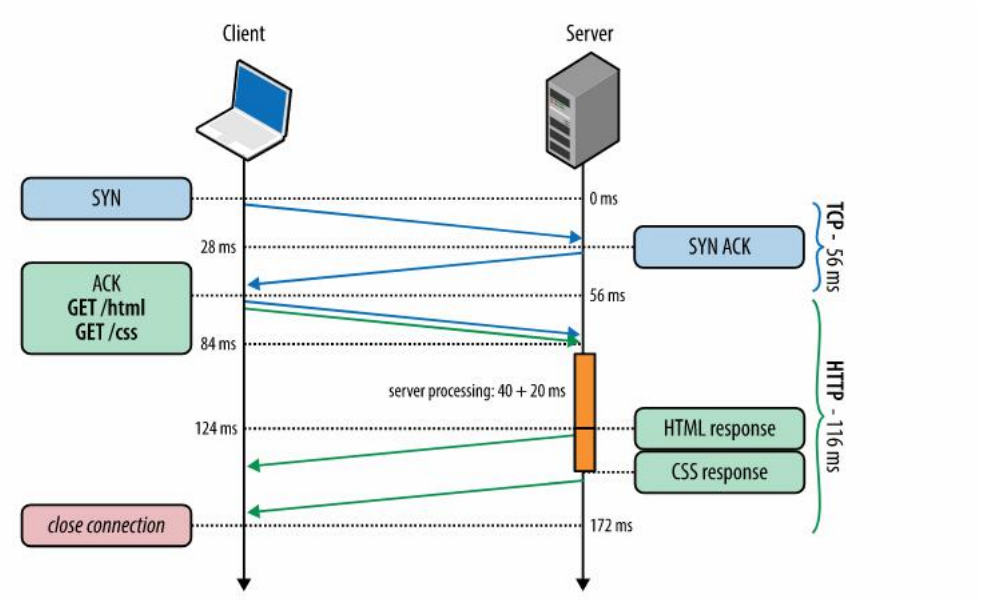


Figura 4: Processamento HTTP em FIFO devido ao Pipelining (GRIGORIK, 2013)

Na Figura 4, é possível observar que dois componentes são requisitados pelo cliente (HTML e CSS), o componente 1 (HTML) é processado e enviado para que apenas depois o componente 2 (CSS) seja processado e enviado. Talvez, de acordo com as necessidades do servidor, seja viável processar ambos paralelamente (GRIGORIK, 2013).

Se o servidor pode trabalhar em duas requisições em paralelo, então, pode-se enviar os dois componentes ao mesmo tempo, como mostrado na Figura 5. O pipelining reduziu a ida e a volta e removeu o tempo ocioso do servidor.

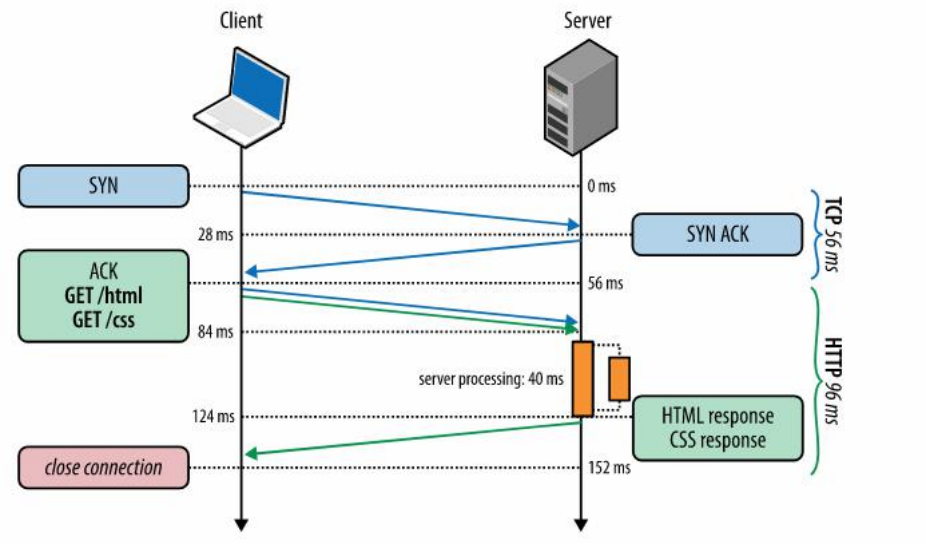


Figura 5: Processamento HTTP em paralelo (GRIGORIK, 2013)

Em teoria, não há motivo para que servidores não processem as requisições entregues (em pipelining) de forma paralela aumentando os ganhos, relacionados a essas otimizações, certamente, serão enormes, contudo, entram em conflito com uma grande desvantagem da versão 1.x do HTTP: serialização rigorosa das respostas (LUNDIN; GARZA, 2017). O HTTP 1.x não permite que dados de várias respostas sejam multiplexados na mesma conexão, de modo a forçar cada resposta a retornar, na íntegra, antes que a próxima resposta seja enviada (GRIGORIK, 2013). Em consonância com a Figura 5, o componente 2 (CSS) foi processado mais rápido que o componente 1 (HTML), entretanto, deve-se esperar obrigatoriamente o fim do componente 1 para ser enviado.

Esse cenário comumente conhecido como "head-of-line"há links de rede subutilizados, aumento de uso de memória de servidores para armazenar dados já processados e atrasos na latência da rede (LUNDIN; GARZA, 2017). Em uma página web, o atraso

de um componente bloqueia o restante, prejudicando a experiência do usuário.

Outros problemas foram encontrados, tais como respostas com falhas, as quais forçam o encerramento da conexão TCP, métodos padronizados de pipeline entre componentes da arquitetura e, por fim, esgotamento de recursos do servidor com requisições muito grandes.

Devido a esses problemas, o pipelining nunca foi muito utilizado em servidores web tradicionais, HTTP/1.1, tornando-se uma configuração avançada tanto para servidores como navegadores, além de conter limitações em sua configuração. Dessa maneira, no caso de aplicações web, não se pode contar com o pipelining para auxiliar no desempenho (GRIGORIK, 2013).

2.6 MÚLTIPLAS CONEXÕES TCP

Na ausência de uma multiplexação no HTTP 1.x, o uso de pipeline se tornou inviável para a quantidade de componentes que uma página genérica poderá ter, de modo que o navegador poderia criar uma fila FIFO com todas as requisições em uma conexão persistente. No entanto, isso removeria a possibilidade de trabalhar as requisições de forma paralela e aumentaria o tempo de ociosidade do servidor. Na prática, criar uma única conexão sequencial será mais lenta que várias conexões em paralelo (GRIGORIK, 2013).

Os navegadores optam por abrir várias conexões entre cliente e servidor, em média de 5 a 10 conexões por host, variando de acordo com o navegador. A solução aumenta o número de conexões abertas, que ampliará o congestionamento do TCP. Isso permite que o navegador não seja limitado no início/reinício de uma conexão TCP, de modo que a taxa de transferência será baixa, slow start, até aumentar o tempo e atingir uma taxa média, congestion avoidance, a depender da versão do TCP.

Os problemas agregados se referem, por exemplo, ao aumento de sockets no cliente que consomem recursos, além de maior concorrência para aproveitar a banda disponível na rede e a limitação de paralelismo da aplicação, as quais têm de acompanhar o fluxo paralelo do TCP (GRIGORIK, 2013).

Utilizar o HTTP força os desenvolvedores a sempre experimentarem melhores configurações baseadas no padrão de formatação de sua página e na quantidade de acesso de seus usuários. A limitação de 5 a 10 conexões TCP paralelas é resultado de experimentação e tornou-se um meio-termo seguro para o desempenho do protocolo. No caso de alguns sites, esse meio-termo poderá fornecer excelentes resultados, em contrapartida, para outros será insuficiente.

O protocolo HTTP é largamente utilizado, sua versão 1.1 está presente em grande parte dos servidores web, entretanto, o protocolo não fora criado para a quantidade de arquivos estáticos utilizados atualmente, dessa forma, ao trabalhar com performance web, serão necessários alguns cuidados e preparativos para que a página não se torne lenta. A seguir, no Capítulo 3, a otimização das requisições serão tratadas mais detalhadamente, junto com cache, conexões paralelas e outras tecnologias/técnicas.

3 TÉCNICAS E BOAS PRÁTICAS PARA MELHORIA DE DESEMPENHO EM HTTP/1.1

Uma página web é formada, sobretudo, por Linguagem de Marcação de Hipertexto (HTML), Folha de Estilo (CSS), Linguagem de programação (Javascript) e Imagens, elementos os quais são compostos por um conjunto de tags atreladas ao conteúdo do site.

O modelo básico de uma página HTML é visto no Código 3.1, no qual a linha 1 é aberta a tag <html> que informa ao navegador que essa é uma página html. Na linha 2, é definida a tag <head> na qual ficará o cabeçalho da página. A linha 3 possui a tag <title> que deverá sempre estar dentro da tag <head> que define o título da página, e é imprescindível ressaltar que logo após deverá ser fechada. Na linha 4, é carregada a folha de estilo, CSS, que definirá o layout da página para garantir a melhor experiência do usuário. Na linha 5, é carregado um script que será executado e modificará, desse modo, variáveis do navegador e adicionará funcionalidades à página. A linha 7 é aberta para o campo <body> que carregará todo conteúdo da página. À linha 8 é adicionado o texto da página pela tag de parágrafo <p>. Na linha 9, é adicionada uma imagem para a página. Por fim, nas linhas 6, 10 e 11, são finalizadas as tags criadas anteriormente em ordem.

Código 3.1: Exemplo de código HTML

```
1  <html>
2  <head>
3      <title>Titulo da pagina</title>
4      <link rel="stylesheet" type="text/css" href="/sidecode/style.
           css">
5      <script src="/sidecode/scripts.js"></script>
6  </head>
7  <body>
8      <p>Ola Mundo!</p>
9      <img alt="Mundo" src="/img/mundo.png">
10 </body>
11 </html>
```

O tempo de carregamento e a renderização de um arquivo HTML puro gasta, em

média, entre 10% e 20%, enquanto o restante do tempo, 80% a 90%, para as requisições de componentes extras na página como imagens, scripts, css etc. (SOUDERS, 2006), os quais são usados para design, fundamentais para sucesso do produto final e, consequentemente, melhorar a usabilidade do sistema.

A tática para os desenvolvedores é diminuir o tempo de carregamento da página, para tanto é necessário diminuir a quantidade de requisições sem que o design seja afetado. Em razão disso, algumas técnicas foram criadas com o intuito de evitar ou de reduzir a performance em detrimento do design. Técnicas as quais definem a redução da quantidade de componentes na página e o melhor uso da memória pelos desenvolvedores.

3.1 TÉCNICAS BÁSICAS DE REDUÇÃO DE REQUISIÇÕES

Nesta seção serão descritas cinco técnicas básicas para redução do número de requisições geradas por uma página, a saber: Image Maps, CSS Sprites, Inline Images, combinação de Scripts e CSS e Otimização de Imagens. Cada técnica será discutida em uma subseção própria.

3.1.1 Image Maps

As Image Maps tratam-se de um modo simples de reduzir hipertextos associados a certo destino no corpo do HTML. Como no caso de um menu com imagens em que não há necessidade de referenciar uma tag para cada imagem, com intuito de redirecionamento, mas sim criar diferentes áreas de uma imagem (polígonos) para que possam ser clicadas. Em seguida, haverá uma lista de coordenadas associadas aos links (SOUDERS, 2006).

A técnica de Image Maps é uma solução eficiente uma vez que, em apenas uma requisição (HTTP), combinam-se todos os dados para ser mais performático de modo a enviar inúmeras requisições para cada imagem na aplicação. A técnica reduz a quantidade de requisições que serão enviadas pelo navegador.

Código 3.2: Image Maps

```
1 <img usemap="#map1" border=0 src="/images/imagemap.gif">
2 <map name="map1">
3 <area shape="rect" coords="0,0,31,31" href="image1.html" title="
    Image1">
4 <area shape="rect" coords="36,0,66,31" href="image2.html" title="
    Image2">
5 <area shape="rect" coords="71,0,101,31" href="Image3.html" title="
    Image3">
6 <area shape="rect" coords="106,0,136,31" href="Image4.html" title="
    Image4">
7 <area shape="rect" coords="141,0,171,31" href="Image5.html" title="
    Image5">
8 </map>
```

O Código 3.2 se trata de um exemplo da técnica de Image Maps a ser utilizada. A linha 1 define que somente uma imagem deverá ser carregada na página (imagemap.gif) e todo mapeamento será feito via usemap. A linha 2 inicia o mapeamento da imagem. Quanto às linhas 3, 4, 5, 6 e 7, estas definem áreas na imagem que poderão ser clicadas, as quais redirecionam para suas respectivas páginas pela tag <href>. Ao utilizar o item "shape"com valor "rect", o desenvolvedor definirá que a área clicável será um retângulo com proporções espaciais definidas em "coords".

Figura 6: Imagem de exemplo - Image Maps

Quanto ao Código 3.2, refere-se a um exemplo da maneira como foi utilizado para um experimento em rede interna com um servidor de HTTP/1.1. O primeiro experimento consistiu em utilizar uma única imagem de 23KB contendo todos os ícones da página, como mostra a Figura 6. O segundo experimento utilizou a mesma imagem, porém, cortada em 6 menores, que, somadas, possuem também 23KB carregadas em sequência. O resultado está na Tabela 1 na qual se comprova que a técnica diminuiu em 3,5 vezes o tempo de carregamento da imagem.

Tabela 1: Tempo de Resposta - Image Maps

	Experimento 1	Experimento 2
Tempo de Resposta (ms)	28	112

3.1.2 CSS Sprites

Diferentemente do uso de Image Maps, cuja desvantagem é a de que, em um mapa, as imagens precisarão estar contíguas, o que implica que a técnica só poderá ser utilizada com restrições relacionadas ao layout da página (SOUDERS, 2006), a técnica de CSS Sprites permite combinar as imagens de maneira mais flexível inserindo-as em um plano de fundo da página. Dessa maneira outros componentes dela poderão se sobrepor a esse plano de fundo de forma a permitir maior liberdade para o desenvolvedor.

O Código 3.3 evidencia o uso da técnica em que a linha 1 é iniciada com a tag de <style> a qual definirá ao navegador como ele deverá renderizar a página. Entre as linhas 2 e 8, são definidas a imagem a ser carregada e as proporções utilizadas. As linhas 9 a 19 definem a posição de cada imagem, de acordo com o estilo da página. As linhas 23 até 29 correspondem à ação que será tomada após o clique do usuário.

A técnica de CSS Sprites segue o mesmo principio que a Image Maps, contudo, sua principal diferença está na liberdade de se trabalhar com as imagens sobre a página. Técnica a qual reduz o tempo de resposta consideravelmente como a Image Maps, além de ser uma técnica elegante para reduzir a quantidade de componentes requisitados.

Código 3.3: Exemplo de CSS Sprites

```
<style>
#navbar span {
    width:31px;
    height:31px;
    display:inline;
    float:left;
    background-image:url(/image.png);
}
.image1
{ background-position:0 0; margin-right:4px; margin-left: 4px;}
.image2
{ background-position:-32px 0; margin-right:4px;}
.image3
{ background-position:-64px 0; margin-right:4px;}
.image4 { background-position:-96px 0; margin-right:4px;}
.image5
{ background-position:-128px 0; margin-right:0px;}
.image6
{ background-position:-160px 0; margin-right:0px;}
</style>

<div id="navbar" style="background-color: #F4F5EB; border: 2px
    ridge #333; width:
180px; height: 32px; padding: 4px 0 4px 0;">
<a href="javascript:alert('image1')"><span class="image1"></span></
    a>
<a href="javascript:alert('image2')"><span class="image2"></span></
    a>
<a href="javascript:alert('image3')"><span class="image3"></span></
    a>
<a href="javascript:alert('image4')"><span class="image4"></span></
    a>
<a href="javascript:alert('image5')"><span class="image5"></span></
    a>
<a href="javascript:alert('image6')"><span class="image6"></span></
    a>
</div>
```

3.1.3 Inline Images

O RFC 2397 (MASINTER, 1998) define que URIs poderão incorporar uma pequena quantidade de dados diretamente, ou seja, pode-se inserir uma imagem a uma página sem a necessidade de adicionar o arquivo estático a ela.

Código 3.4: Exemplo de Inline Images

```
1  <img src="<BASE 64 CODE>" width="16" height="
       14" alt="image1">
```

O Código 3.4 acima elucida como é possível adicionar dados a uma página. Na linha 1, há uma imagem a ser carregada a partir de um base64 (codificação MIME para transferência de conteúdo), na qual o navegador poderá traduzir essa codificação em imagem e renderizar em tela.

Em contrapartida, a técnica impede que arquivos, incorporados ao HTML, não sejam adicionados ao cache do navegador, assim, não são arquivos estáticos. Em razão disso, nesses casos, a solução é utilizar CSS. Todo navegador possui um espaço de memória definido como cache, cuja função é a de salvar os dados estáticos, posto isso, a próxima vez que retornar à página, será renderizada mais rapidamente por conter uma versão previamente carregada em memória, sem a necessidade de requisitá-la novamente (FILHO, 2017).

Diferente do caso acima, no CSS o base64 será armazenado em cache pelo navegador por estar em um arquivo estático, logo pode-se incorporar imagens de segundo plano em arquivos CSS.

O uso de base64 aumenta de 39% a 45% o tamanho da imagem, já que essa representação textual ocupa mais bytes, é preciso a compactação de imagens (SCHECHTER; KRISHNAN; SMITH, 1998). É imprescindível salientar que a limitação do navegador é outra desvantagem do uso de Inline Images, uma vez que, por norma, navegadores têm necessidade de suportar até 1024 bytes por URL. Essa regra muda entre os navegadores,

assim, o uso dessa técnica é aconselhável para imagens pequenas e decorativas para o site (SOUDERS, 2006).

Ao utilizar uma imagem que possui 23KB e traduzir ela para base 64, resultou em um arquivo de 30KB; em rede interna, esse aumento resultou em um acréscimo de 40ms em média no tempo de carregamento da página. A ideia é diminuir o número de requisições extras para se evitar a abertura de conexões TCP para arquivos muito pequenos.

3.1.4 Combinando Scripts e CSS

Atualmente, todos os sites desenvolvidos utilizam CSS e Javascript, momento em que paira a dúvida quanto ao que será mais performático: inserir o CSS e Javascript no corpo do HTML ou em arquivos separados para que sejam referenciados. De forma geral, separar esses arquivos melhorará o desempenho, embora essa técnica de modularização deverá ser feita com cuidado. A separação, em muitos arquivos, poderá prejudicar o desempenho, uma vez que cada arquivo será uma requisição a mais para a página.

Uma solução plausível para sanar esse problema, talvez, seja a de unir todo CSS (ou Javascript) da página em um único arquivo para que não possua mais do que um script e um arquivo de estilo por página. A combinação deverá ser feita de acordo com os conteúdos da página e com as regras de downloads paralelos feitos pelo navegador.

Outra solução se refere ao uso de cache na frente de servidores de conteúdo estático. Segundo Glassman (1994), foi encontrada uma melhora entre 30% e 50% para todas as requisições feitas a partir do cache em uma rede isolada, com redução de 4 até 6 vezes na taxa de transferência, assim como a redução, na carga da rede, foi o equivalente à taxa de acerto do cache.

Matsushita, Nishimine e Ueda (2015) propuseram uma solução distribuída para o cache, no qual se pode melhorar o tempo de carregamento da página, caso esteja concentrada em regiões de maior acesso da aplicação. Para Raza et al. (2015), o uso de uma técnica de pré-busca inteligente, para predição da taxa de troca de componentes em uma página web, conseguiu alcançar uma taxa de 90% de acerto das requisições utilizando tão somente o cache como base.

3.1.5 Otimização de Imagens

Em muitos casos, uma imagem é salva em um formato que facilita sua manipulação, consequentemente, será maior em bytes. Adicionar uma imagem dessa forma, no servidor web, tornará o carregamento mais lento e diminuirá a qualidade da experiência do usuário. Embora imperceptível, as imagens contêm dados a mais além dos pixels vistos na tela, por isso, é necessário compilá-la e salvá-la em um formato que reduza os bytes desnecessários.

Em termos simples, otimizar imagem significa remover todos os dados desnecessários, salvos dentro dela, para reduzir o tamanho do arquivo. Essa otimização agregará uma diminuição da qualidade da imagem, mas serão próximas o suficiente para que, em dimensões reduzidas, passem sem nenhuma aparente perda.

Abaixo foram elencadas algumas técnicas básicas para auxiliar no desenvolvimento de uma página web para reduzir bytes e facilitar a troca de dados.

- Reduzir o espaço em branco ao redor das imagens. É necessário recortar para removê-lo.
- Utilizar formatos de arquivo adequados. Caso haja ícones, marcadores ou gráficos, que não tenham muitas cores, é preferível usar o formato GIF e salvar o arquivo com menos cores, de forma que os bytes serão reduzidos. Ou se houver gráficos mais detalhados, é aconselhável o formato de arquivo JPG para salvar as imagens com as cores originais, o que reduzirá a qualidade.
- Salvar as imagens nas dimensões adequadas à tela. O HTML ou o CSS não servem para redimensionar uma imagem desnecessariamente. Caso o sistema seja responsivo, é preciso adequar a dimensão máxima da página web como a da imagem.

Embora apenas essas técnicas não sejam suficientes para melhorar a performance da aplicação na totalidade, dado que nem sempre será possível reduzir a qualidade da imagem ou adicionar imagens em inline. Uma solução plausível que distribui arquivos estáticos por geolocalização poderá ser a solução para páginas que têm muitas imagens.

3.2 CONTENT DELIVERY NETWORK

A proximidade dos usuários com o servidor web gera um impacto significativo no tempo de resposta das páginas a serem acessadas. Com o aumento da quantidade de acessos em escala mundial, as empresas enfrentam o problema de performance em suas páginas, que se deve à localização em um único lugar de seus servidores, e para corrigi-lo é necessária a implantação do conteúdo em vários servidores geograficamente dispersos.

Para adaptar uma aplicação a vários ambientes espalhados ao redor do mundo, é fundamental que seja usada uma arquitetura distribuída, e, para esse caso especificamente, será necessário rever toda a aplicação, como controle de sessão, gerenciamento de requisições ao banco de dados etc. (SOUDERS, 2006).

Uma mudança de arquitetura de um projeto web não é algo simples de ser feito e irá gerar um alto custo, portanto, esse custo pode não ser justificado pela incerteza de sucesso do projeto. Entender como uma aplicação web genérica funciona pode dar mais clareza ao assunto.

Os componentes estáticos são os mais pesados em uma requisição HTTP na Web atual. Os arquivos estáticos são fáceis de hospedar e para os quais há pouca dependência. A melhor solução, portanto, é dispersar os servidores web com componentes estáticos em uma arquitetura distribuída. O conceito de Content Delivery Network (CDN) nasceu nesse contexto.

CDN é uma coleção de servidores web espalhados pela rede em vários locais do mundo, na qual o conteúdo estático é replicado entre eles. O objetivo principal de uma CDN é entregar o conteúdo de forma transparente e efetiva aos usuários finais, de modo que evolui cada vez mais para superar limitações inerentes à internet. Fornecem serviços para melhorar o desempenho da rede, aumentar a largura de banda, melhorar a acessibilidade e, por fim, auxiliar a manutenção de dados replicados (PATHAN; BUYYA; VAKALI, 2008).

Como consequência da expansão do tráfego da rede, os serviços de CDN cresceram na mesma proporção, já que há cada vez mais aparelhos ligados consumindo banda.

Populares serviços web começaram a sofrer constantes congestionamentos devido ao aumento de acessos, especialmente, em momentos de picos não esperados (flash crowds), os quais indisponibilizam o serviço. À CDN não compete apenas a preocupação com a performance de uma aplicação, mas também a segurança contra ataques que poderão gerar indisponibilidade e escalabilidade automática de servidores em momentos de pico (GOEL; WITTIE; STEINER, 2015).

Existem formas diferentes de implementar uma CDN, dentre as quais implementá-la em forma de proxy no cliente, em que os usuários finais deverão configurar um proxy para redirecionar as requisições de conteúdo estático para um servidor específico. A implementação mais comum encontrada é a que exige que os desenvolvedores alterem a URL de seus componentes para apontar para um DNS específico, o qual faz parte da CDN, um host externo ao que está a ser acessado (GOEL; WITTIE; STEINER, 2015). Poderá, então, surgir o questionamento quanto a utilizar uma CDN em vez de somente um cache web compartilhado. No caso do cache web compartilhado, se um objeto solicitado existe nele, os clientes já detêm uma cópia de um dos servidores proxy que fornecem o serviço.

Os benefícios do cache web são a redução do consumo de banda e do congestionamento da rede, por guardar os itens mais solicitados, e, finalmente, melhora a confiabilidade, porque os clientes poderão acessar o recurso ainda que o servidor não esteja disponível (cópia em cache)(VAKALI et al., 2003). Por outro lado, o uso de cache web compartilhado tem duas desvantagens; a primeira, em relação, à atualização, caso o proxy não seja atualizado corretamente, clientes poderão receber dados antigos e defasados. Quanto ao segundo, a medida que o número de clientes cresce, como não existe um balanceamento de carga, alguns proxies ficarão congestionados e haverá indisponibilidade da aplicação (VAKALI et al., 2003). Ao usar CDN, a escalabilidade, com os mesmos dados, é garantida aos clientes, atrelada a todas as outras vantagens de se utilizar um web cache (KUENZER et al., 2017).

Por definição, proxies de cache são utilizados para armazenar conteúdos solicitados mais frequentemente ou os mais recentes, mas em um contexto local. Enquanto que a CDN armazena conteúdo especificado pelo administrador de forma a melhorar o

acesso normalmente não presente no cache de acordo com a geolocalização (VAKALI et al., 2003).

Em Souders (2006), é possível observar a melhora de 18% no carregamento da página em uma CDN, comparado com a situação com todos os componentes hospedados em um único servidor web. Para Goel, Wittie e Steiner (2015), um arquivo estático único teve, em média, 43% de seu carregamento mais rápido. Por conseguinte, o uso da CDN é valioso e é aconselhável utilizá-la em projetos web.

É importante salientar que, ao realizar testes em uma CDN, deve-se ter consciência de que a localização é uma informação útil, cujo impacto nos resultados será grande. Para entender o impacto real da CDN, é preciso medir os tempos de resposta de vários locais geográficos diferentes. Os testes de desempenho poderão ser enganosos, caso esses cuidados não sejam tomados.

Uma CDN pode redirecionar o cliente para uma página de várias maneiras, visto que o redirecionamento, por si só, poderá ser um problema para o desempenho da página, assunto sobre o qual trataremos mais a diante, afinal, nesse momento, é necessário, contudo, entender quais os tipos de redirecionamento existentes e como decidir qual utilizar.

- Unicast: o cliente será redirecionado de acordo com o nome do host na rede. O DNS será resolvido localmente pelo cliente, há uma relação de um para um entre um endereço IP e seu servidor de DNS, porque este sempre resolverá para aquele endereço IP. Apesar de ser uma implementação simples, não existe um controle sobre qual servidor DNS o cliente acessará. Um usuário dos EUA, por exemplo, pode ser atendido por um servidor de DNS da China, sendo assim não seria redirecionado para um servidor mais próximo e de menor latência (CALDER et al., 2015).

- Anycast: em redes Anycast, diferentemente do Unicast, não existe mais a limitação de um para um, assim ao traduzir o DNS para o IP, prevalecerá o de menor latência e mais próximo à geolocalização do usuário. Outra vantagem é a redundância, porque o conteúdo é refletido em vários servidores DNS para inúmeros IPs. Com o uso de vários servidores, não é possível ter conhecimento do desempenho da rede, sendo, desse modo, dificil redirecionar o tráfego para fora de um nó quando este

está sobrecarregado. Na prática, isso não parece problema já que a maioria dos grandes CDNs utilizam essa técnica (CALDER et al., 2015).

O conteúdo também produz uma grande diferença, dado que cada vez mais o tráfego mundial se foca em streaming de vídeo, até 2019 se tornará 90% do total da web, com a maior parte do conteúdo entregue por CDNs (CISCO, 2012). Em Liu et al. (2012), foi realizada uma medição de 200 milhões de streamings de vídeo em inúmeros sites conhecidos pelo serviço. Os autores afirmam que 20% das sessões apresentaram um recarregamento do arquivo de vídeo em 10% das páginas e, em mais de 14% dos casos, o vídeo teve um tempo de inicialização de até 10 segundos.

Para tanto, inúmeras CDNs estão a trabalhar na melhora do QoE do vídeo e a reduzir os custos desse provisionamento, portanto, ao escolher uma CDN, é necessário que seja de forma inteligente em consonância com as necessidades da página, sobretudo, pelo conteúdo dela por se tratar do principal fator nessa escolha.

Analisar como a aplicação se comporta também é importante antes de contratar a CDN, muitas vezes, o serviço poderá ser ideal para um projeto, no entanto, não funcionar para outro. Alguns fatores como a reação da CDN a mudanças de carga, visto que a melhora da qualidade da entrega e a redução de custos deverão ser consideradas, quando se busca esse serviço.

Com a utilização da CDN, é necessário estabelecer uma política de cache para a troca desses dados estáticos armazenados.

3.3 ADICIONANDO CABEÇALHO DE EXPIRAÇÃO

As páginas web incluem cada vez mais componentes e, ao acessar uma página pela primeira vez, o usuário terá que fazer várias requisições HTTP. Para evitar tais requisições, a solução é utilizar o máximo do cache do navegador. Outra solução é o uso de cabeçalhos de expiração para evitar requisições desnecessárias, normalmente, é usado com mais frequência em imagens, embora seja indicado em todos os componentes estáticos da página.

O cabeçalho de expiração é utilizado para informar ao cliente web, nesse caso navegadores ou proxies, que é possível utilizar a cópia de seu cache de um determinado componente até o tempo especificado no cabeçalho, cujo valor é inserido no cabeçalho HTTP e enviado ao cliente junto com a resposta (HTTP).

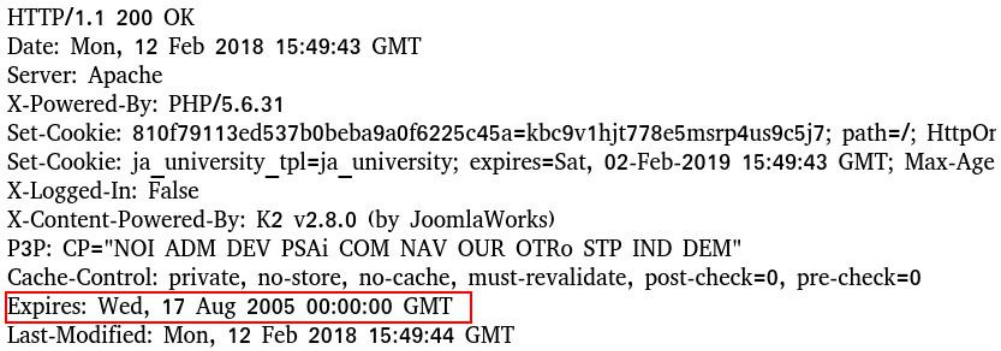


Figura 7: Resposta HTTP - Expires

A Figura 7 se trata de um exemplo do mau uso desse recurso, mostra uma resposta recebida pelo cliente no ano de 2018, no entanto, o campo expires do cabeçalho define uma data de 2005, e forçará a atualização do componente a cada nova requisição do usuário.

Para a versão HTTP/1.1, foi inserido um campo no cabeçalho do protocolo, Cache-Control, para resolver as limitações do uso de expires. O expires tem a necessidade de determinar uma data específica, de forma que a sincronização deverá ser mais rígida entre servidor e cliente (FIELDING et al., 1999).

O método de Cache-Control utiliza o campo max-age para especificar o tempo, em segundos, no qual o componente estará armazenado em cache. Se tiver passado menos que o período previsto em segundos, o navegador usará a versão de cache, evitando uma requisição HTTP como mostra a Figura 8

Por questão de portabilidade, é aconselhável inserir os dois itens no cabeçalho HTTP, já que nem todos os clientes poderão suportar a versão HTTP/1.1. Caso o cliente suporte a nova versão do HTTP, o Cache-Control sobreporá o expires. Diversos proxies fornecem esse serviço automaticamente, por exemplo, o Apache, que utiliza o mod_expires para gerar os dois itens na resposta, ou seja, um algoritmo baseado em TTL,

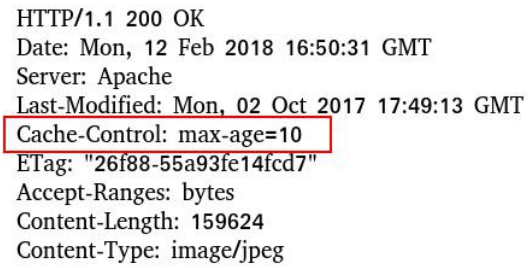


Figura 8: Resposta HTTP - Cache Control

o qual se apoia no cabeçalho de expiração do HTTP. Caso um documento em cache com um TTL expirado seja referenciado, de modo que uma solicitação GET condicional seja enviada ao servidor para validar o documento e os componentes, que possuem menor valor de TTL, serão substituídos primeiro (APACHE, 2018). Outros caches do mercado empregam a mesma técnica baseada em TTL, embora com algumas diferenças como algoritmos de substituição, mapeamento da memória e políticas de escrita (FILHO, 2017).

O cache, como a CDN, também tem problemas de performance para grandes quantidades de acessos. Em razão disso, o uso de cache distribuído pode ser a solução para limitações de recursos, com cada nó provendo um cache confiável e escalável. Esta solução, permite aumentar a performance da página, embora adicione outros problemas, como a distribuição do conteúdo pelos nós, cujo resultado será o aumento da complexidade na busca pelos dados. Muitas vezes, não é viável possuir uma cópia completa de todos os dados em todos os nós (FILHO, 2017).

O cuidado com o que está inserido no cache do cliente também é importante, no entanto, a aplicação não pode depender do cache do usuário, visto que alguns problemas poderão suceder, como segue:

- O usuário pode limpar o cache em um período curto de tempo ou acessar o servidor web em navegação anônima.
- O usuário poderá ter acessado tantos sites diariamente que não possui mais o cache do servidor web.
- Softwares como antivírus oferecem serviço de limpeza de cache cada vez que o navegador é fechado.

Em Souders (2006), um estudo no site Yahoo! indicou que o número de usuários que entraram pelo menos uma vez por dia sem cache variou entre 40-60%. Esse mesmo estudo revelou que o número de visualizações a páginas com cache era de 75-85%. A porcentagem de visualizações, nessa página, com uso de cache é maior que a porcentagem de usuários que entraram sem cache, indicando que os usuários acessam a aplicação sem cache, mas continuam navegando no site com o cache já carregado.

Para dispositivos móveis, a realidade não é a mesma. Em Agababov et al. (2015), foi produzido um proxy externo, em que o usuário poderia utilizá-lo na frente de seu navegador como cache. A taxa de acerto foi entre 22% e 32% para clientes desktop, enquanto a melhora em sistemas móveis foi somente de 2%. O resultado se deve ao poder computacional limitado, principalmente, memória dos dispositivos.

Em Vesuna et al. (2016), os resultados não são diferentes, enquanto páginas carregadas em desktop têm uma taxa de acerto em até 34%, no cache em plataformas móveis, chega no máximo em 13%. Em Wan; Xiao Sophia et al. (2013), é indicado que a velocidade lenta da CPU atrapalha o cache das páginas, em razão disso, é necessário atentar-se a cada componente e escolher cuidadosamente os arquivos mais críticos que entrarão no cache do celular.

Utilizar o expires e o Cache-Control reduz, significativamente, o tempo de resposta do cliente, pois os navegadores leem diretamente do cache sem gerar uma única requisição a mais. No caso da atualização de algum componente antes do tempo de expiração, quando não é gerada uma requisição, o usuário final, mesmo que os arquivos ainda estejam em seu cache, não receberá a atualização dos componentes. Para garantir que o usuário tenha a versão mais recente do componente, o desenvolvedor da aplicação deverá indicar a versão no nome do arquivo, gerando assim, um versionamento para auxiliar o cache.

A configuração de expires e Cache-Control influencia diretamente no desempenho do cache, portanto, uma boa configuração, de acordo com as características da página, refletirá em um ganho expressivo de desempenho em uma página web.

Como foi definido anteriormente, a CDN armazenará componentes estáticos, os

quais, em grande parte, tratam-se de componetes como imagens, CSS e JS. A otimização de imagens já foi relatada, embora não seja a única técnica para redução do tamanho em bytes.

3.4 COMPRESSÃO

Em Ihm e Pai (2011), após 5 anos de análise do tráfego da web real, com a captura de mais de 70.000 usuários diariamente em 187 países, mostrou que 50% de todo tráfego da rede são arquivos multimídias. Por outro lado, 45% do tráfego são conteúdos baseados em texto (HTML, CSS e Javascript 5%, 25% e 15%, respectivamente) para os quais também existem métodos eficientes para a redução do conteúdo como, por exemplo, a compressão.

A regra geral define que, para se melhorar o tempo de resposta no acesso a uma página, basta diminuir as requisições enviadas para o servidor. Quando não é possível reduzir a quantidade de requisições enviadas, então, é necessário diminuir o tamanho da resposta na totalidade, visto que uma resposta com tamanho diminuto reduzirá o tempo de transferência. Para a compressão das respostas HTTP, é utilizada a técnica de codificação por GZIP, deflate ou ZLIB. Focaremos, então, no GZIP já que todos os outros são variações pequenas deste.

A técnica de compressão de arquivos já é utilizada há décadas para reduzir arquivos em mensagens de e-mail e transferência de arquivos. Implementado na versão HTTP/1.1, os clientes web indicam que possuem suporte para compactação pelo cabeçalho da requisição. O servidor, por sua vez, verifica o cabeçalho da requisição e comprime a resposta de acordo com os métodos listados pelo cliente (SINGH; KRISHNA; KUMAR, 2017).

A compressão é feita pelos cabeçalhos Accept-Encoding e Content-Encoding, de modo que o navegador envia o cabeçalho contendo o campo Accept-Encoding o qual informa os métodos de compressão que aceita. O servidor, por outro lado, envia uma resposta em que fornece o formato de compressão utilizado no campo Content-Encoding, caso não possua compressão, o campo não é inserido no cabeçalho (FIELDING et al.,

1999). A Figura 9 mostra uma requisição com o campo Accept-Enconding informando quais métodos de compactação o cliente aceita.

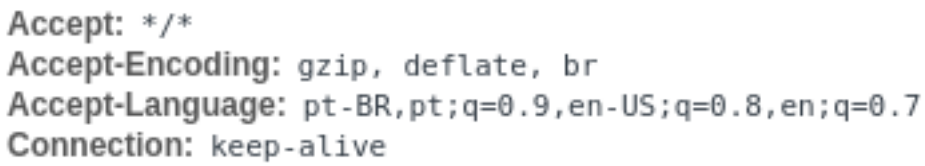


Figura 9: Requisição HTTP - Encoding

O Gzip foi padronizado pelo RFC 1952 (DEUTSCH et al., 1996) e tornou-se o método de compressão mais popular e eficaz. Os servidores escolhem o que irá ser compactado com base no tipo de arquivo. Regra geral, vale a pena comprimir qualquer resposta de texto, seja HTML, JSON, Scripts ou folhas de estilo. Arquivos como imagens ou PDF não deverão entrar no método, já que são formatos compactados, por isso, compactá-los será um desperdício de recursos da CPU e poderá aumentar o tamanho dos arquivos (SOUDERS, 2006).

Embora o HTTP/1.1 tenha uma alta capacidade de compressão, é desabilitado por padrão em servidores. A técnica de compactação de arquivos gera um custo para CPU, tanto para o servidor, que deve compactar os arquivos antes de enviar, como para o cliente, que o descompactará para leitura. Em teoria, para definir se vale a pena ou não compactar certo arquivo, seria necessário considerar o tamanho da resposta, largura de banda e a distância da conexão. Informações complexas de se ter, sobretudo, em relação à distância do cliente e nos casos em que não utilizam CDN (SAKAMOTO et al., 2015).

O formato de compressão GZIP é genérico e poderá ser aplicado a qualquer fluxo de bytes, o método, dessa maneira, transforma qualquer arquivo de texto em outro arquivo menor. O algoritmo utiliza uma técnica de associação para lembrar de fragmentos previamente vistos removendo as partes duplicadas. Por esse motivo, possui grande desempenho em arquivos de texto, no entanto, para formatos de arquivo já compactados, não produz quase nenhuma melhora (SINGH; KRISHNA; KUMAR, 2017).

Todos os navegadores suportam o GZIP como compactação padrão e, automaticamente, geram suas requisições HTTP utilizando o método, em seguida, o servidor deverá estar configurado para servir o conteúdo compactado quando solicitado. A ativa-

Tabela 2: Compressão GZIP

Arquivo	Taxa de Compressão (%)	Tamanho Original (Bytes)	Tamanho Comprimido (Bytes)
jquery-1.8.2.js	336	265217	78931
esprima.js	574	193593	33741
jquery-3.3.1.js	337	271751	80669
jquery-3.3.1.min.js	286	86926	30348

ção desse recurso é uma tarefa simples e possui resultados significativos.

Em alguns casos, o GZIP aumentará o tamanho do arquivo de texto a ser enviado, apesar de ocorrer apenas em arquivos muito pequenos. Para realizar a compressão, o GZIP utiliza um dicionário, entretanto, se este adicionar uma quantia maior de dados, que a economia de compressão pode fornecer, não irá ter uma melhora, talvez, o contrário, podendo criar um arquivo maior que o anterior.

De modo geral, é indicado compactar antes de enviar qualquer arquivo maior do que 1 ou 2 KB. No Apache, a diretiva mod_gzip_minimum_file_size, que controla o tamanho mínimo de arquivos que deverão ser compactados, segue como valor padrão de 500 bytes[1].

A Tabela 2 mostra alguns experimentos com arquivos estáticos bem conhecidos por desenvolvedores. O experimento leva somente em consideração o tamanho em bytes reduzidos do arquivo texto, pois a redução do tamanho influenciará diretamente no tempo de transferência do arquivo. O uso de GZIP, para compactação da resposta, reduziu o tamanho dos arquivos em média 70% do tamanho original, e mostra o quão impactante poderá ser essa técnica.

A técnica já é consolidada nos principais sites, por exemplo, em um ranking dos top 500 sites de 2015, 92% utilizava a técnica em arquivos Javascript e a média de economia era maior que 50% do tráfego em bytes (SAKAMOTO et al., 2015). Apesar de a técnica não poder ser usada para arquivos, que não são textos, ainda assim é válida,

[1] https://httpd.apache.org/docs/2.4/mod/mod_deflate.html

além de seus resultados bem expressivos para o desempenho de uma página web, embora não seja a única técnica eficiente para reduzir bytes de arquivos estáticos de texto.

3.5 MINIFICAÇÃO E OFUSCAÇÃO

O Javascript é responsabilidade total do engenheiro do sistema. O componente não passa por uma etapa de compilação, assim, outras técnicas deverão ser empregadas para otimização desse recurso. A técnica de compressão já foi mencionada e auxilia na transferência dos arquivos diminuindo o tamanho destes. A minificação é uma técnica que possui o mesmo objetivo, mas também auxilia na etapa de compressão.

A minificação é a prática de reduzir todos os componentes do Javascript, CSS ou HTML os quais não são necessários para o processamento, como remoção de caracteres desnecessários ou de espaço, de comentários ou quebra de linha, de maneira a reduzir o total de bytes do arquivo (SHAILESH; SURESH, 2017). O Código 3.5 mostra um CSS não minificado, enquanto o Código 3.6 mostra o mesmo CSS minificado.

Código 3.5: Exemplo de CSS não minificado

```
1  .entry-content p {
2  font-size: 16px;
3  }
4
5  .entry-content ul li {
6  font-size: 16px;
7  }
8
9  .product_item p a {
10   color: #000;
11   padding: 15px 0px 0px 0;
12   margin-bottom: 5px;
13   border-bottom: none;
14 }
```

Código 3.6: Exemplo de CSS minificado

```
1  .entry-content p,.entry-content ul li{font-size:16px}.product_item
       p a{color:#000;padding:15px 0 0;margin-bottom:5px;border-bottom:
       none}
```

Outra técnica menos utilizada e mais agressiva é a ofuscação, muitas vezes, confundida com minificação por realizar as mesmas operações, entretanto, a redução é menos considerável. O objetivo da técnica é deixar o código ilegível e irreversível, porque, ao mesmo tempo que retira bytes como a minificação, também incorpora inúmeros "lixos"desnecessários para o funcionamento da página (SOUDERS, 2006). No final do processo, ele diminui o tamanho do arquivo, no entanto, o desempenho é bem inferior comparado à minificação. O Código 3.7 é a ofuscação do Código 3.5, como foi visto anteriormente.

Código 3.7: Exemplo de CSS ofuscado

```
1  [class~=entry-content] ul li,[class~=entry-content] p{font-size:16
       px;}[class~=product_item] p a{color:#000;}[class~=product_item]
       p a{padding-left:0;}[class~=product_item] p a{padding-bottom:0px
       ;}[class~=product_item] p a{padding-right:0pc;}[class~=
       product_item] p a{padding-top:.9375pc;}[class~=product_item] p a
       {margin-bottom:.052083333in;}[class~=product_item] p a{border-
       bottom-width:medium;}[class~=product_item] p a{border-bottom-
       style:none;}[class~=product_item] p a{border-bottom-color:
       currentColor;}[class~=product_item] p a{border-image:none;}
```

Eis as desvantagens principais da técnica de ofuscação:

- Inclusão de Bugs: por ser uma técnica mais complexa, existe maior probabilidade de inserção de erros no código no meio do processo e será de difícil depuração.
- Manutenção: a técnica não se resume a modificar nomes de variáveis, mas sim ter o controle total de chamadas do código, por exemplo, as URLs externas que o código acessa não podem ser alteradas, logo não pode ser ofuscada.
- Debugging: o código fica ilegível para humanos. Em razão de existir grande difi-

Tabela 3: Minificação vs Ofuscação

Arquivo	Tamanho Original (KB)	Minificação (KB)	Ofuscação (KB)
jquery-1.8.2.js	260	92.4	212
esprima.js	190	71.2	140.8
jquery-3.3.1.js	266	86.4	205.1
processing-1.4.1.js	785	228.9	514.2

culdade na leitura do código, muitas vezes, é necessário, para solucionar a correção do código, abandonar o trabalho já feito e reiniciá-lo.

A decisão na escolha da técnica a ser utilizada está relacionada ao próposito do desenvolvedor, pois, se o objetivo está na segurança do conteúdo disponibilizado, a ofuscação será a melhor opção. Ou caso a performance seja o principal objetivo, a melhor opção será a minificação.

A Tabela 3 compara as técnicas mediante o tamanho em bytes final dos arquivos, demonstrando um desempenho superior da técnica de minificação em relação à redução de bytes. Os itens escolhidos são arquivos facilmente encontrados na web.

Ao adicionar Javascript em inline no HTML, é necessário que menos bytes sejam adicionados para melhorar o desempenho, assim, o uso de minificação se torna importante. A compressão do software Gzip também se beneficia da técnica de minificação, dado que arquivos minificados tendem a melhorar os resultados dos algoritmos de compressão (SHAILESH; SURESH, 2017). Isso ocorre devido à redução de elementos do arquivo pela minificação, tornando o código menor em bytes e facilitando o algoritmo de compressão.

A Tabela 4 mostra o desempenho de arquivos minificados com compressão em bytes. A junção das duas técnicas mostrou uma redução de mais de 85% em bytes de todos os arquivos, por outro lado, a compressão do software Gzip tem o maior impacto. Com o aumento de scripts em páginas web, as duas técnicas se tornam indispensáveis.

A minificação não é exclusividade somente de scripts, por ser cada vez mais

Tabela 4: Compressão com minificação

Arquivo	Minificado (KB)	GZIP + Minificado (KB)
jquery-1.8.2.js	92.4	33
esprima.js	71.2	20.1
jquery-3.3.1.js	86.4	30.3
processing-1.4.1.js	228.9	65.3

comum projetos utilizarem as mesmas técnicas no HTML [2] ou CSS [3]. Assim, é sempre aconselhável a minificação de todos os arquivos, quando possível.

O caso de arquivos estáticos de texto como CSS e JS é ainda mais complexo que de uma imagem, devido às funções que aqueles desempenham. Para tanto, é necessário mais que uma redução de bytes para otimizar a renderização da página.

3.6 CSS NO TOPO E JS NO RODAPÉ

A estrutura de uma página web contém inúmeros elementos e o carregamento destes é feito de forma estrutural, de sorte que os componentes são baixados na ordem em que aparecem na página. Consciente dessa informação, é natural que o desenvolvedor da página inicie com os principais componentes dela, beneficiando os mais críticos, baixados primeiro.

Seguindo essa linha de raciocínio, componentes, como folhas de estilo ou CSS, deveriam ser inseridos no final da página, já que definem tão somente a estética. Apesar desse raciocínio seguir uma lógica, não se trata de um verdade para a Web. Em Souders (2006), observou-se que, enquanto se procurava acelerar o carregamento, a priorização desses componentes não permitia uma renderização da página mais rapidamente.

O resultado desses experimentos mostrou que atrasar o carregamento dos componentes mais críticos, em favorecimento da folha de estilo, ajudaria na percepção de mais velocidade na página. Tal evento ocorre, porque o CSS é um componente bloqueante para

[2] http://kangax.github.io/html-minifier/

[3] http://cssnano.co/

a renderização de um site, visto que o navegador só renderiza a página após obter todos os recursos CSS. Logo não está relacionado à latência da rede, de modo que os componentes podem até ser enviados mais rapidamente, mas não será essa a percepção do usuário enquanto não for priorizado o CSS (JOVANOVSKI; ZAYTSEV, 2016).

O conceito de Carregamento Progressivo nasceu a partir desses experimentos e define que o importante é fornecer um feedback visual ao usuário. O usuário não quer esperar em frente a uma tela branca enquanto seus principais componentes são baixados, sobretudo, os que possuem uma conexão lenta.

Inserir as folhas de estilo no fim da página impossibilitará o Carregamento Progressivo, já que esses componentes serão baixados por último, por isso, o navegador atrasa a exibição desses componentes visíveis, enquanto o usuário aguarda a folha de estilo na parte inferior.

Para Card, Robertson e Mackinlay (1991), até hoje é considerado o conselho básico sobre o tempo de resposta de uma página web, mostrando o feedback que deve ser dado para cada instante. As métricas não se relacionam somente com o tempo de transferência dos componentes ao cliente, mas, principalmente, com a renderização deles na página. A seguir alguns padrões de tempo definidos:

- Para os períodos de até 0,1 segundo: tempo necessário para que o usuário perceba que o sistema está a processar a requisição, nenhum feedback visual é necessário.
- Entre 0,1 e 1 segundo: ideal para o carregamento da página, mesmo que o usuário perceba a lentidão no acesso, ainda é um tempo aceitável para inércia do usuário.
- Entre 1 e 10 segundos: limite para manter a atenção do usuário focado em um diálogo, logo, é necessário um feedback visual em que o usuário já tenha uma interação com a página.

Em Domes (2013), é mostrado o modelo RAIL de performance, projetado para identificar as necessidades dos usuários em relação ao tempo de transferência de dados mais a renderização em tela. O foco do modelo é voltado totalmente para o usuário, cujo objetivo final não é definir o tempo em plataformas específicas, mas, sim um tempo genérico

relacionado ao conforto do usuário como o conselho apontado por Card, Robertson e Mackinlay (1991). O modelo fornece tempo previsto para carregamento da página e para executar uma animação. Eis os intervalos de tempo:

- Entre 0 e 0,16s: necessário para rodar cada quadro de uma animação. Para se manter a fluidez, 60 novos quadros devem ser renderizados por segundo. Essa métrica influencia, especialmente, os serviços de streaming.
- Entre 0 e 0,1s: caso a resposta para o usuário estiver nesse período de tempo, perceberão como imediata.
- Entre 0,1 e 0,3s: os usuários notam atraso leve na resposta.
- Entre 0,3 e 1s: tempo aceitável e natural para resposta, apesar de o atraso ser percebido.
- Mais que 1s: o usuário já perde o foco na página.
- Mais que 10s: frustração do usuário e provável abandono da página.

Mesmo após anos, as métricas não se modificaram muito. Ainda que o tempo de resposta diminua, é imprescindível sempre dar feedback visual ao usuário, visto que esse é um indicador de progresso visual, como o carregamento da página, de forma progressiva, o cabeçalho, a barra de navegação, o conteúdo principal etc., para que o usuário tenha a sensação de um carregamento mais rápido da página.

A técnica mostrada não está atrelada ao tempo real para carregar os componentes da página, dado que esses valores provavelmente não mudarão, mas sim com a percepção de acesso do usuário, influenciada pela ordem com que o navegador carregará os componentes da página ao longo do tempo.

No Código 3.8, é possível visualizar a implementação correta de CSS em uma página, na qual toda renderização se inicia junto com a tag <body> (linhas 8 até 10). Ao carregar todos os componentes CSS na tag <head> (linhas 1 até 7), o desenvolvedor garante que nenhum componente CSS bloqueará a renderização do usuário.

Código 3.8: Exemplo de CSS no Topo

```
1  <head>
2    <link rel="stylesheet" href="/site-header.css">
3    <link rel="stylesheet" href="/article.css">
4    <link rel="stylesheet" href="/comment.css">
5    <link rel="stylesheet" href="/about-me.css">
6    <link rel="stylesheet" href="/site-footer.css">
7  </head>
8  <body>
9      DATA
10 </body>
```

No caso de scripts, quando executados, há uma interrupção no download dos demais itens da página, além de um bloqueio da renderização até que a execução seja finalizada (MACCAW, 2011), uma vez que, conforme mencionado anteriormente, o carregamento de uma página é concomitante ao dos componentes. Mover todos os scripts para o fim da página evitará que bloqueiem qualquer download de componentes. E permanecer no fim da página significará que ela já foi renderizada para o usuário. Utilizar essa sugestão facilita a sensação de Carregamento Progressivo e uma melhor paralelização de downloads (MACCAW, 2011).

O bloqueio do navegador, quando um script é executado, é simples de se justificar por dois motivos plausíveis; no primeiro, o script pode modificar o documento, então, o navegador aguarda o fim de sua execução para continuar bloqueando. Quanto ao segundo motivo, trata-se do bloqueio de downloads paralelos, que garante que os scripts serão executados na ordem correta; se vários scripts fossem baixados simultaneamente, não haveria a garantia dessa ordem (QI et al., 2008). Por exemplo, se dois scripts são baixados simultaneamente, o primeiro a finalizar o download também seria primeiro a ser executado, não levaria em consideração a ordem.

Para clarificar o processo de carregamento de uma página, é preciso entender como o download paralelo é implementado nos navegadores. O navegador define quantos componentes são baixados por vez, mas, de acordo com o RFC 2616 (FIELDING et al., 1999), o HTTP/1.1 sugere que sejam feitas até duas requisições simultâneas por host

gerando um padrão de escada, no qual os componentes são baixados em pares, a imagem é uma simplificação em que cada componente possui o mesmo tamanho e leva o mesmo tempo para ser enviado.

Limitar os downloads a dois componentes por vez é uma diretriz, embora seja customizável, atualmente, nenhum navegador adota tal norma ditada pela RFC 2616 (FIELDING et al., 1999). Com a evolução da internet, os navegadores se adaptaram e a quantidade de downloads paralelos que podem fazer também. No Firefox, por exemplo, esse valor pode ser modificado na flag network.http.maxpersistent-connections-per-server que, presentemente, recebe valor 6, como padrão [4]. O navegador Edge também suporta o limite de 6 conexões TCP simultâneas que não podem ser modificadas [5]. O navegador Chrome se diferencia dos demais por permitir até 10 conexões simultâneas, o que pode ser comprovado em testes realizados no próprio navegador, já que a empresa não fornece tais dados.

3.7 JS E CSS EXTERNOS

Quando a arquitetura CDN foi abordada, mencionou-se que alguns componentes podem ser adicionados como links externos a uma página, componentes estáticos como imagens e CSS são uns desses exemplos. Recomendações de organização de CSS e scripts, em uma página, para melhoria da performance foram abordadas, mas onde são adicionados o Javascript e o CSS? Inline junto ao código HTML ou em uma CDN como arquivo externo?

Em um experimento simples, é possível identificar que arquivos com scripts e CSSs inline são mais rápidos. Como teste, duas páginas foram criadas, a primeira, com um arquivo HTML de 91 KB, e a segunda com um arquivo HTML de 8 KB, um CSS de 62 KB e três scripts (2, 11 e 9 KB), também num total de 91 KB. Em uma rede interna, a página, que contém o arquivo HTML com arquivos estáticos em inline, foi carregada em tempo médio de 80ms, enquanto a página com múltiplos componentes externos levou uma média de 106ms.

[4] http://kb.mozillazine.org/Network.http.max-persistent-connections-per-server

[5] https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/network

A página que utiliza inline é mais rápida, porque a segunda usa links externos e sofre uma sobrecarga de requisições HTTP. É natural que o mais rápido seja o que possui apenas uma requisição, pois, ao abrir uma única conexão TCP, o desempenho da rede será melhor, uma vez que evita o redirecionamento para hosts externos ao da aplicação.

O uso de arquivos externos é um benefício para a performance, visto que, no mundo real, os arquivos estáticos são armazenados em cahce pelos navegadores (ZAKAS, 2010). Todavia, arquivos HTML com conteúdo dinâmico não podem ser armazenados em cache. Assim, componentes CSS e Javascript incorporados ao HTML dinâmico em inline também não poderão ser armazenados em cache. Por isso, é preferível que o HTML com conteúdo dinâmico tenha poucos dados para que possam ser transferidos mais rapidamente. Para os casos onde o HTML é estático, pode-se armazenar em cache sem problemas.

Redirecionar todos seus arquivos estáticos para links externos não causa uma melhora para o primeiro acesso à página, mas para as próximas visitas em um curto período de tempo (ZAKAS, 2010). No nosso exemplo anterior, a média para o primeiro acesso ainda será a mesma, mas as próximas visitas ao site será transferido somente o arquivo HTML de 8 KB em vez de um arquivo HTML de 91 KB como no exemplo com código estático em inline. Então, como é difícil calcular quando arquivos CSS e Javascript deverão estar no código ou fora dele, algumas métricas foram sugeridas (SOUDERS, 2006).

- Page Views: métrica definida de acordo com as visitas na página web. Por exemplo, é possível imaginar que o comportamento padrão do usuário seja acessar a página uma vez por mês, nesse caso, o conteúdo estático já terá sido removido do cache do navegador. Uma opção seria anexar o CSS e Javascript em Inline, pois o benefício de inserir links externos é proporcional ao número de visitas repetidas ao site. Para os casos onde a página é frequentemente acessada se justifica o uso de arquivos externos devido ao cache.

- Uso de Cache: conhecer a capacidade de uso de cache dos usuários também poderá ser usado como métrica. A maioria dos usuários acessa vários links do mesmo site, os quais aparecem pela primeira vez com cache vazio, por outro lado, a continuidade da navegação pelo site fará com que os arquivos estáticos sejam armazenados

em cache e garantindo a melhora de desempenho com o uso de links externos.

- Reuso de Componentes: se cada página do site utiliza o mesmo CSS e Javascript, o uso de arquivos externos resultará em uma alta taxa de reutilização, ajudando o acerto e a melhoria de performance da página.

Bons desenvolvedores de front-end criam aplicativos de web com componentes estáticos externos com características modulares e cada arquivo é inserido de forma independente no projeto. Apesar de ser uma prática comum em Engenharia de Sistemas, pode ocasionar um aumento drástico no tempo de carregamento da página, em que aqueles que visitam a página pela primeira vez, e não possuem os arquivos em seu cache, serão altamente prejudicados (NAGY, 2013).

Algumas técnicas foram criadas para aproveitar o cache do usuário, mesmo que não sejam renderizadas na página inicial, como por exemplo, a técnica de Post-Onload Download, que foca nas páginas secundárias que serão acessadas. Ela se baseia em adicionar todos os Javascripts e o CSS inline no arquivo HTML da página inicial, permitindo que este seja baixado mais rapidamente. Após concluída essa etapa, inicia-se o script de download dos arquivos de Javascript e CSS externos, porque, ao acessar outras páginas, os usuários já devem ter esses arquivos em seu cache (KOECHLEY, 2007). Apesar de aumentar o tempo de carregamento da página, esses componentes não serão renderizados em tela e não prejudicarão a experiência do usuário.

A técnica de Dynamic Inlining pode ser vista como um avanço da anterior. O servidor adiciona um cookie na resposta, fornecendo ao servidor uma visão do que o cliente possui em seu cache. A regra é simples: se possui um cookie válido, então, não é necessário reenviar os arquivos estáticos, caso não possuir, será feito o download dos arquivos estáticos (KOECHLEY, 2007). O servidor detém total controle dos componentes que o cliente possui em seu navegador, diminuindo a quantidade de bytes a serem enviados.

Essas técnicas, no entanto, podem não ser efetivas, dependendo do comportamento do usuário. Caso o usuário feche o navegador, perderá seus cookies e, ao acessar a página mais uma vez, os arquivos serão baixados novamente, mesmo que estejam em cache. A sincronização da duração do cookie, em relação ao tempo de um componente

no cache, pode ser um problema, visto que cada navegador trabalha de forma diferente. Essas técnicas poderão ser uma boa saída comparada às decisões simples de adicionar ou não um arquivo inline em seu HTML, porém devem ser feitas com cautela para evitar problemas como sobrecarga de componentes na rede e lentidão no carregamento das páginas.

Os arquivos estáticos não são os únicos vilões na performance de uma página web. Ao acessar uma URL, o usuário terám um nome associado a um endereço IP, desse jeito, é necessária a tradução desse nome, que também custa tempo.

3.8 REDUÇÃO DNS LOOKUP

Sistemas de Nomes de Domínio (DNS) possuem a função de mapear os nomes de hosts para o seu endereço IP específico (BEERTEMA, 1993). Funciona como uma lista telefônica, mapeando o nome de pessoas de acordo com seus números. Na visão do navegador, quando o usuário acessa uma URL como www.dcc.ufrj.br, o navegador consulta um servidor de DNS que resolve esse nome para o IP específico.

Utilizar DNS pode ser uma vantagem em se tratando de performance, porque um sistema pode ter um único DNS, mas inúmeros IPs diferentes. A infraestrutura servida por balanceadores de carga é um exemplo desse recurso, visto que essa arquitetura cria um alto grau de redundância para aplicação, escalando o projeto em vários nós (KUMAR; VENKATESAN, 2016). Se a quantidade de usuários cresce, a aplicação pode ser escalada entre vários nós para suportar o novo volume de carga.

O uso de DNS também influencia de forma negativa na performance, porquanto, toda vez que o navegador faz a requisição a um DNS, necessita traduzir esse nome para um IP, e dura entre 0,2 e 12s. Enquanto o cliente está traduzindo o nome, o servidor não pode fazer nenhum download de arquivos e ficará inerte. O tempo de tradução do DNS depende diretamente do Internet service provider (ISP) [6] a ser utilizado, a carga sobre ele, a proximidade e a largura de banda influenciam.

O sistema operacional (SO) guarda os DNSs traduzidos em forma de cache, e o

[6] https://en.wikipedia.org/wiki/Internet_service_provider

navegador também possui um cache semelhante (KUMAR; VENKATESAN, 2016). Primeiro, o navegador verifica se possui o DNS em seu cache e, em caso negativo, consulta o SO. Se não tiver sucesso, é feita uma requisição ao servidor de DNS registrado em sua configuração de rede, o que impacta negativamente o desempenho.

Nesse caso, temos dois caches que precisam ser sincronizados, mas que funcionam de forma bem diferente. Toda consulta de DNS adiciona à resposta do servidor um tempo de vida (Time-To-Live ou TTL), indicando ao usuário quanto tempo esse DNS pode ser armazenado no cache. Eis a primeira diferença entre os caches. O cache do Sistema Operacional respeita esse valor, enquanto o do navegador o ignora, por outro lado, junto a esses dois valores distintos de cache, o protocolo HTTP possui o recurso Keep Alive que mantém a conexão com servidor (XU; HUANG; BHUYAN, 2004). Se existe uma conexão TCP entre o cliente e o servidor, não há a necessidade de refazer a pesquisa de DNS e ignora-se o tempo de cache.

O desenvolvedor também deve se preocupar em definir o valor do TTL, pois o RFC 1537 (BEERTEMA, 1993) define como valor recomendado 24 horas; período em que o cache deveria armazenar os dados. Para Souders (2006), todavia, não é dessa maneira que os principais sites funcionam, os valores são bastante distintos, como por exemplo, o yahoo.com que possui um TTL de um minuto, enquanto ebay.com, de uma hora. O valor do TTL dependerá da arquitetura do site. O uso de load balancer permite traduções de vários IPs a um mesmo DNS, por isso o valor de TTL tem de ser reduzido para que os novos hosts, na infraestrutura, sejam conhecidos e o tráfego redirecionado (KUMAR; VENKATESAN, 2016).

A diferença entre os tempos de TTL é uma mistura de fatores, tanto intencionais como históricos. Sites, que têm muitos acessos, possuem uma redundância alta e um grande poder de escalabilidade, razão pela qual possuem vários IPs, evitando o downtime, porém, com a necessidade de incessantes consultas ao DNS. Quanto maior a necessidade de consultas maior será o número de requisições, embora, nesse caso, nem sempre seja gerada uma queda de performance em um escopo macro.

É natural imaginar que a redução de consultas ao DNS diminuam o tempo de carregamento da página, em contrapartida, diminuir essas consultas implicaria em reduzir

a quantidade de hosts. Limitar o número de hosts também limitaria a escalabilidade do projeto e os downloads paralelos, portanto, somente restringir a quantidade de hosts não é uma tática realmente útil.

A solução para o problema de traduções de DNS pode ser suavizada com uma distribuição igualitária de componentes entre os hosts, de modo que a consulta aos servidores sempre será feita, reduzindo a necessidade de consultas ao servidor de DNS externo. O uso de Keep Alive evita a sobrecarga no TCP, mantendo a conexão persistente, reduzindo a necessidade de consultas de DNS. O uso de CDN é novamente recomendada para esses casos, porque a distribuição desses servidores, pelo mundo, reduz o tempo de resposta, caso o SO necessite perguntar ao servidor de DNS externo mais próximo.

3.9 EVITAR REDIRECIONAMENTOS

Quando se possui mais de um host e balanceam-se os componentes em uma distribuição igualitária, cria-se um problema com redirecionamentos, porque forçará acesso a um componente externo, o que criará mais uma requisição.

O redirecionamento tem a função, basicamente, de redirecionar de uma URL a outra, para tanto os motivos, para utilizar essa técnica, são muitos: uso de proxy para redirecionar requisições; requisições de arquivos estáticos da página; análise do fluxo de tráfego por um proxy; contagem de números de views na página por analisadores; URL mais simples ou mais curta para que o usuário possa se lembrar etc (VISCOMI; DAVIES; DURAN, 2015).

Os redirecionamentos são definidos no protocolo HTTP com código 3xx na resposta da requisição. O RFC 2616 (FIELDING et al., 1999) define os conceitos básicos do HTTP para as versões posteriores e, nessa versão, são definidos todos os códigos de resposta do protocolo, inclusive, os de redirecionamento. O redirecionamento indica que existe uma ação a mais e, muitas vezes, resume-se em uma requisição extra a ser feita pelo navegador.

Os códigos de status 3xx são:

- 300 Múltiplas escolhas
- 301 Movido permanentemente
- 302 Movido temporariamente
- 303 See Other (302)
- 304 Não modificado
- 305 Use Proxy
- 306 (não utilizado)
- 307 Temporary Redirect (302)

Os códigos 303 e 307 são especificações do código 302. O código 303 é utilizado para o redirecionamento que força o uso do método GET e o código 307 para forçar redirecionamentos que utilizam a mesma URI. Essa tática foi criada na versão 1.1 do protocolo, contudo, não foi adotada pelos desenvolvedores que se mantiveram o uso do código 302, como na versão 1.0 do protocolo. O status 306 não é mais utilizado e o status 304 não é um redirecionamento, mas uma resposta a GET condicionais como já mencionado.

Os códigos mais encontrados serão o 301 e o 302, porque o redirecionamento do navegador tem a função de enviar o usuário automaticamente para a URL especificada e todas as informações necessárias normalmente estão presentes no cabeçalho. Isso torna a resposta inexistente e nenhuma informação é adicionada em cache (VISCOMI; DAVIES; DURAN, 2015).

O redirecionamento é caótico para páginas web, pois, enquanto não for finalizado, nada poderá ser renderizado na página. Caso ocorra no início da requisição, por exemplo, encurtamento de URL, a página se manterá em branco até o fim do redirecionamento. Ou se ocorrer no meio da página, interferirá mais ainda no carregamento dos outros componentes. O redirecionamento, portanto, entrega um documento HTML e, sem esse documento, não é possível carregar nenhum componente estático. O resultado desse

impasse é o bloqueio de todos os downloads paralelos (VISCOMI; DAVIES; DURAN, 2015).

Redirecionamentos são mais comuns em situações como troca de back-end, quando este é atualizado e suas URLs são modificadas. O redirecionamento é uma solução fácil para resolver tal problema, uma vez que servirá para integrar duas rotas diferentes de dois códigos diferentes. Nesse caso, o redirecionamento facilita o trabalho dos desenvolvedores, embora o excesso desse recurso degrade a performance do site. Um exemplo benéfico do recurso é a função do proxy Apache chamada de Alias [7], pois seu uso permite a integração de certa URI a um arquivo físico, sem utilizar redirecionamentos por URL. O Código 3.9 mostra o uso do recurso.

Código 3.9: Alias Apache

```
1 Alias /computer /usr/local/apache/htdocs/site/index.html
```

O Código 3.9 mostra a URI apontando diretamente para um arquivo (nesse caso, um index.html), evitando futuros redirecionamentos externos. Nesse caso, o proxy conhece de antemão o arquivo referenciado, desse modo, essa técnica funciona para os casos em que redirecionamentos são feitos de URLs que estão presentes no mesmo servidor.

A integração entre back-ends é uma técnica possível de ser explorada, sobretudo, para os casos em quem existam versões diferentes do back-end na mesma máquina. Os casos em que existam back-ends diferentes para servidores diferentes devem ser abolidos, já que forçam redirecionamentos externos desnecessários.

Para os casos nos quais o servidor migrou de DNS, essa configuração poderá ser feita no próprio servidor de DNS. Diante disso, é adicionado um redirecionamento pelo proxy, ou é realizada a criação de uma página simples com um redirecionamento pela tag href. Todas essas técnicas podem ser substituídas por um CNAME, que se trata de um registro DNS que cria um Alias a partir de um domínio para outro (MOCKAPETRIS, 2003). De forma que não ocorre o redirecionamento via HTTP, mas sim uma mudança de DNS na tradução.

Em alguns casos, são inevitáveis os redirecionamentos, por exemplo, encurtado-

[7] https://httpd.apache.org/docs/2.4/mod/mod_alias.html

res de URL. O Google possui esse serviço [8], pelo qual se pode enviar a URL de seu site e aquele aponta para um DNS menor, que poderá ser lembrado mais facilmente pelo usuário. No caso do domínio dcc.ufrj.br, este poderá ser traduzido como https://goo.gl/gTW3LZ. Embora esse exemplo não tenha trazido clareza, imagine o envio de um formulário com URI com mais de 15 caracteres que poderá encuratá-lo com essa ferramenta. Para esse exemplo, o redirecionamento vale a queda de performance.

Questões relacionadas à Segurança da Informação também influenciam nos redirecionamentos. Cada vez mais os navegadores forçam o uso do SSL nos sites, por isso, muitos sites migram do HTTP para HTTPS (Versão com SSL do HTTP) e, nesse processo, utilizam o redirecionamento para forçar usuários antigos a usar a nova URL. Note, como exemplificado acima, no site do portal do aluno da UFRJ, emprega-se HTTPS e via proxy força tal uso. No Código 3.10, é possível visualizar uma requisição via Curl, a página com HTTPS; já no Código 3.11, a mesma requisição sem HTTPS possuindo um redirecionamento.

Código 3.10: HTTPS sem redirecionamento

```
1  $ curl -k -I https://portal.ufrj.br/
2  HTTP/1.1 200 OK
3  Date: Thu, 01 Feb 2018 16:30:50 GMT
4  Server: Apache
5  Last-Modified: Mon, 23 Jan 2017 14:31:55 GMT
6  ETag: "105-546c3dc7b0ca0"
7  Accept-Ranges: bytes
8  Content-Length: 261
9  Vary: Accept-Encoding
10 Content-Type: text/html
```

[8] https://goo.gl/

Código 3.11: HTTP com redirecionamento

```
1 $ curl -k -I http://portal.ufrj.br/
2 HTTP/1.1 301 Moved Permanently
3 Date: Thu, 01 Feb 2018 16:30:54 GMT
4 Server: Apache
5 Location: https://portal.ufrj.br/
6 Content-Type: text/html; charset=iso-8859-1
```

Apesar de ser um redirecionamento, essa prática não deverá ser mal vista, visto que tão somente usuários antigos acessarão a URL e terão de ser redirecionados. A documentação Apache [9] recomendada o emprego dessa técnica para esses casos. Embora seja possível evitar redirecionamentos com técnicas simples, é necessário definir se é realmente inevitável antes que seja implementado.

3.10 CONFIGURE ETAGS

A primeira recomendação de desempenho identificada expressa, que para uma página ser mais performática, é necessário reduzir o número de requisições HTTP feitas ao servidor, e foi verificado o impacto do cache no número de requisições. Quanto maior for a capacidade do navegador armazenar em seu cache, maior será a performance para o usuário.

Ao fazer uma requisição para o servidor, o usuário faz o download de todos os recursos necessários para a página e armazena em cache. Antes de entrar no conceito de ETag, é necessário revisar como os conteúdos são armazenados em cache, porque essa técnica pode frustrar completamente o cache de um site hospedado em mais de um servidor.

Um dos mecanismos para armazenar arquivos no cache é utilizar o cabeçalho Expires, como demonstrado anteriormente, é definida uma data de validade para um componente do site, evitando requisições HTTP a mais. Caso esse componente esteja expirado no cache do navegador, é feita uma requisição GET condicional para verificar se ainda é

[9] http://httpd.apache.org/docs/current/rewrite/avoid.html#redirect

válido, não é performático, todavia, é melhor que fazer o download de todo o componente mais uma vez. Se o componente se mantiver válido no servidor, haverá o status 304 (Não modificado) na resposta como mencionado há pouco.

Existem duas maneiras de verificar se o componente é válido no servidor. A primeira é uma comparação com o campo Last-Modified na resposta do HTTP, porque este define quando o arquivo foi modificado pela última vez no servidor. A segunda maneira utiliza ETag (SOUDERS, 2006).

ETag foi introduzida no protocolo HTTP/1.1 (SHIM; SCHEUERMANN; VINGRALEK, 1999) e atualizada posteriormente (FIELDING; RESCHKE, 2014), definida como uma sequência aleatória de caracteres que identifica de forma exclusiva a versão de um componente requisitado. A cada requisição, feita ao servidor por algum componente, é gerado um valor de ETag e este é enviado na resposta do HTTP, como pode ser visto no Código 3.12. A ETag foi adicionada no protocolo com o objetivo de flexibilizar a verificação de conteúdo guardado em cache, portanto, diferente da estrutura de data do cabeçalho Expires.

Código 3.12: Resposta com ETag

```
1 HTTP/1.1 200 OK
2 Content-Type: image/png
3 Content-Length: 61404
4 ETag: "a5ea8bbcc437de9787e9a87ef6ef690a"
```

Em um momento futuro, o usuário requisitará o componente de novo e poderá usar esse valor para "questionar"se a versão em cache do seu navegador ainda é válida. Para verificar esse dado, o cliente deverá adicionar o campo If-None-Match em sua requisição com o valor recebido anteriormente, como pode ser visto no Código 3.13.

Código 3.13: Requisição ETag

```
1 GET /imagem.png HTTP/1.1
2 Host: dcc.ufrj.br
3 If-None-Match: "a5ea8bbcc437de9787e9a87ef6ef690a"
```

Caso a versão ainda seja válida, o servidor retornará com uma resposta com

código 304 para indicar que é seguro para o cliente usar a versão em cache. Por outro lado, se a imagem tiver mudado, o servidor responderá com código 200, portanto, será preciso uma nova versão da imagem e uma nova ETag (FIELDING; RESCHKE, 2014).

A ETag é um mecanismo de validação de cache baseado em conteúdo, portanto, diferente do par de cabeçalhos Cache-Control e Expires, que configuram um mecanismo de validação baseado em tempo.

O problema do uso de ETags está na maneira com a qual são produzidos e na arquitetura de infraestrutura que utiliza. Normalmente, são construídos usando atributos, muitas vezes, exclusivos de um servidor específico, inviabilizando um balanceamento de carga.

Por exemplo, o Apache utiliza três atributos para geração de uma ETag, o tamanho do componente, data em que o componente foi modificado e o inode do arquivo armazenado no disco [10]. O último item é específico e único para cada servidor, desse modo, caso o site esteja em um cluster de servidores, o valor será diferente para cada servidor. Então, se o balanceamento de carga está sendo feito com cinco máquinas, a probabilidade da requisição GET condicional acessar o mesmo servidor é de 1/5, a considerar que o algoritmo de round robin [11] está a ser utilizado para o balanceamento.

A baixa probabilidade atrapalha o balanceamento do servidor, porque muitos sistemas possuem cache externos como Squid[12] e Varnish[13], mas também utilizam de CDN para realizar essa tarefa. Mesmo utilizando o campo Expires em conjunto com o campo ETag, é necessário que os valores de ambos estejam corretos, caso contrário, o componente é reenviado.

Uma solução é simplesmente retirar da ETag o atributo que não permite o balanceamento de carga, no caso do Apache, o campo inode. Nas configurações de Virtual Host do Apache, é possível escolher quais componentes estarão presentes na formação da ETag, o Código 3.14 mostra uma possível configuração.

[10] http://httpd.apache.org/docs/2.2/mod/core.html#fileetag

[11] https://pt.wikipedia.org/wiki/Round-robin

[12] https://squidproxy.org/

[13] https://varnish-cache.org/

Código 3.14: Modificação de geração de ETag

```
1  FileETag MTime Size
```

No Código 3.14, é definido que campo de ETag é formado apenas pela data de criação e tamanho do componente. É importante lembrar que, nesse caso, o horário entre os servidores deverão ser sincronizados. Com essa mudança, o campo ETag poderá ser utilizado sem a preocupação de gerar falsos positivos e reenviar todos os componentes a cada requisição.

Outras soluções, para o ambiente distribuído abordadas, são o uso de sistemas intermediários, que gerenciam a validade desse campo (KHANAL; KRISHNA; ANNAMALAISAMI, 2018), e serviços de cache específicos para o campo ETag, que trabalham em conjunto com proxies (HAYTON et al., 2016).

Em um ambiente distribuído, é recomendado o uso proxies para balanceamento com capacidade de configuração de Etags, como o caso de HAProxy[14], Ngnix [15] e Apache[16].

3.11 CONSIDERAÇÕES FINAIS

Todas as técnicas mostradas, neste capítulo, têm o objetivo de melhorar o tempo de respostas do protocolo HTTP/1.1, presente em grande parte da web, no entanto, atualmente o HTTP/2 está se mostrando uma solução viável e entrando em conflito com sua versão anterior. No Capítulo 4, serão abordados o HTTP/2 e as principais melhorias com relação ao desempenho.

[14]http://www.haproxy.org/

[15]https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/

[16]https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

4 HTTP/2

O HTTP/2 foi concebido para adicionar melhorias em performance, as quais HTTP/1.1 não poderia fornecer, abrindo novas perspectivas para otimizar o acesso à página web. Neste capítulo, serão apontadas as principais diferenças entre esta versão 2, mais recente do HTTP, e a versão 1.1, além de apresentar as técnicas e as boas práticas de forma a garantir uma melhoria na experiência do usuário no acesso web com HTTP/2.

Os principais avanços do HTTP/2 são reduzir a latência em requisições com a multiplexação das respostas, minimizar a sobrecarga de dados na rede por meio de um novo método de compressão de campos de cabeçalho, uma nova tática de priorização de dados e o uso de um mecanismo de Server Push (BELSHE; PEON; THOMSON, 2015). Precisamente, são novas vantagens que deverão ser conhecidas pelos desenvolvedores para que utilizem melhor o protocolo.

O novo protocolo foi criado com a mesma semântica de seu antecessor. Todos os conceitos básicos estão presentes, como métodos, código de status, URIs e todos os campos de cabeçalho. A principal modificação no protocolo está na maneira como seus dados são formatados e transportados pela rede do servidor ao cliente, todas essas modificações são transparentes à primeira vista.

Neste capítulo, serão primeiramente abordadas as limitações do HTTP/1.1 e a história do HTTP/2 nas seções 4.1 e 4.2, seguidos pelas melhorias introduzidas pelo HTTP/2 na seção 4.3 e aquelas com foco em desempenho, explicitamente, na seção 4.4. Finalmente, na seção 4.5, será tratado o fim do HTTP/1.1.

4.1 LIMITAÇÕES DO HTTP/1.1

O HTTP/1.1 é um protocolo ASCII sequencial, logo produz requisições sequenciais, impedindo o paralelismo da requisição. No HTTP/1.1, quando cliente e servidor estabelecem uma conexão TCP, o cliente terá de fazer múltiplas requisições GET ao servidor para ter múltiplos arquivos, contudo, a resposta é sempre sequencial, de acordo com as requisições.

Os arquivos são necessários na mesma página, por outro lado, seus pedidos são, obrigatoriamente, sequenciais, ignorando uma possível ação simultânea na mesma requisição. Sendo assim, o tempo para a página carregar é equivalente aos números de pedidos multiplicados pela latência entre cliente e servidor, como pode ser visualizado do lado esquerdo da Figura 10 na qual uma resposta é enviada a cada requisição GET recebida pelo servidor (SAXCÉ; OPRESCU; CHEN, 2015).

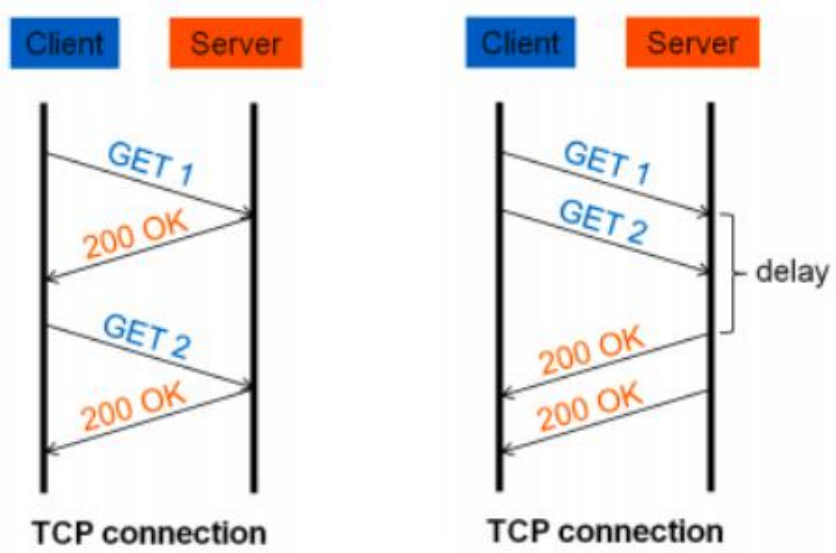


Figura 10: Requisições via conexões TCP, extraído de Saxcé; Oprescu; Chen (2015)

A solução para esse problema no HTTP/1.1 é adicionar um pipelining ao protocolo. O lado direito da Figura 10 mostra o envio de duas requisições GET sem aguardar a resposta da primeira. Apesar de melhorar o desempenho, a leitura das respostas deverá ser feita de forma sequencial, assim, caso a resposta do primeiro GET solicitar um arquivo muito grande, as outras estarão presas até o fim da primeira requisição (SAXCÉ; OPRESCU; CHEN, 2015).

Para contornar esse problema, os navegadores implementam o uso de conexões paralelas para cada domínio, mesmo assim a tática de bloqueio de requisições é um dos principais gargalos do protocolo.

Outro problema do protocolo é a duplicação de dados de cabeçalho entre as requisições como cookies, paths etc., visto que ao enviar muitas requisições, muitos dados são reenviados, ocorrendo pela natureza não persistente de uma comunicação HTTP, que

obriga cada requisição enviar todos os campos de cabeçalho novamente; que, por sua vez, fará com que desenvolvedores tenham de criar um arquivo único para enviar em uma única requisição em detrimento de múltiplas requisições.

4.2 HISTÓRIA DO HTTP/2

Apesar de, em um primeiro momento, não se notar grandes diferenças entre as versões 1.1 e 2 do protocolo, elas seguem paradigmas totalmente diferentes. Embora os protocolos possuam os mesmos campos, são diferentes e incompatíveis. Logo, um proxy executando a versão 2 do protocolo não pode responder requisições feitas por um cliente com a versão 1.1 e isso é inédito na história do protocolo, que sempre se preocupou com a portabilidade.

A ausência de retrocompatibilidade entre as versões de HTTP mostra que a versão não é uma atualização da versão 1.1, mas sim um protocolo totalmente novo que se utiliza da mesma semântica. Devido ao enquadramento binário que segue definido no protocolo SPDY do Google, antecessor do HTTP/2 (GRIGORIK, 2013).

Apesar dessa falta de retrocompatibilidade, o desenvolvedor só perceberá a mudança se programar com sockets TCP puro. Fora esse cenário, a única diferença perceptível será no melhor desempenho em alguns casos e novas funcionalidades que o protocolo fornece.

O SPDY é um protocolo experimental desenvolvido pelo Google e liberado para testes em 2009 com objetivos bem específicos, como (GRIGORIK, 2013):

- Redução de 50% no tempo de carregamento da página.
- Evitar a necessidade de mudança de conteúdo dos autores do site.
- Menor complexidade de implementação e sem necessidade de alteração na arquitetura de redes, devido ao encapsulamento do protocolo em pacotes binários.
- Desenvolvido em código aberto.
- Possuir um alto grau de experimentação para validação do protocolo experimental.

Em pouco tempo, os engenheiros da Google compartilharam os primeiros resultados do projeto e uma versão para experimentos. O estudo relatava também uma melhora de 55% em 25 sites em uma rede simulada (GRIGORIK, 2013).

Em 2012, o protocolo já era suportado pelos principais navegadores do mercado para experimentação, assim, o novo modelo de protocolo para web já estava tomando forma e seguidores. Percebendo o sucesso do novo protocolo, o HTTP-WG (equipe de manutenção de modificações do HTTP) abriu chamadas de propostas para uma nova versão do protocolo, e após muita discussão, foi decidido que o novo protocolo se basearia na proposta do Google, com a necessidade apenas de uma padronização mais rígida.

Após três anos de discussões e testes, o novo protocolo estava finalizado e sua versão experimental (SPDY) foi finalizada pela Google. Essa parceria para criação do novo protocolo permitiu que uma única implementação do protocolo em padronização fosse experimentada por qualquer pessoa, permitindo levar em consideração os comentários de usuários com experiência direta com o protocolo e conhecimento de seus bugs.

4.3 MELHORIAS HTTP/2

A versão 2 do protocolo possui inúmeras vantagens que serão discutidas nessa seção como enquadramento binário, novo algoritmo de compressão, multiplexação de pacotes, prioridade de pacotes e a técnica de Server Push.

4.3.1 Enquadramento Binário

O HTTP/1.1, como visto anteriormente, é baseado em texto ASCII, o que cria uma natureza sequencial ao protocolo. Nele, a fronteira entre o cabeçalho e o corpo de uma requisição é identificada pela quebra de linha entre os dois campos. Protocolos em modo texto são simples, de fácil leitura e depuração, e foram populares no início da internet.

A nova versão adota um formato binário com dois quadros para cada requisição (BELSHE; PEON; THOMSON, 2015), um contendo apenas o cabeçalho e o outro

contendo somente o corpo da requisição, conforme evidencia a Figura 11. Sendo que o formato binário inviabiliza, por exemplo, testes via telnet, que operam em modo texto. Todo cabeçalho estará contido em um quadro de cabeçalho, por sua vez, o corpo da requisição estará em um quadro de dados.

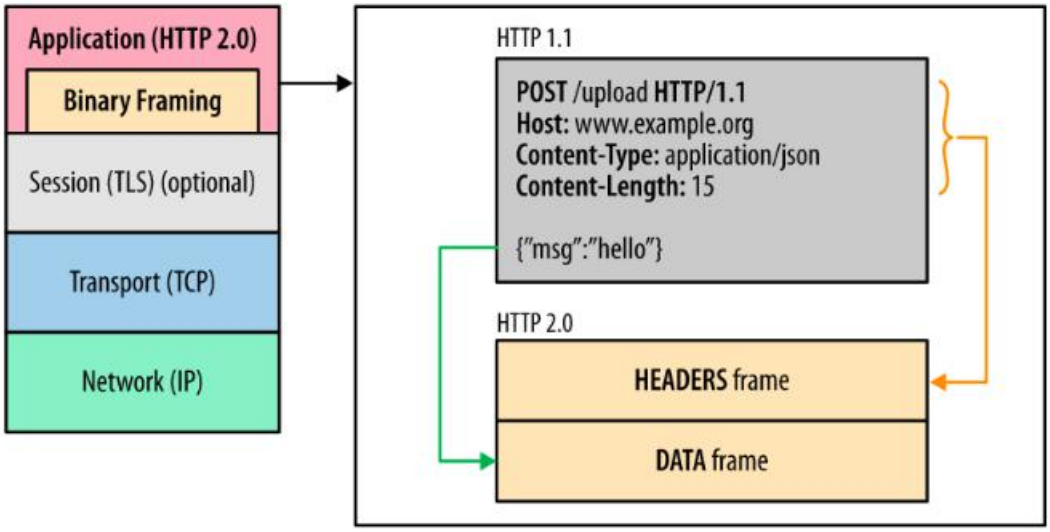


Figura 11: Enquadramento Binário (GRIGORIK, 2013)

Esse formato de enquadramento é o motivo da falta de compatibilidade entre as versões dos protocolos. Os pacotes esperados são totalmente distintos, o que inviabiliza a leitura dos dois da mesma forma. Assim, o servidor de origem necessitaria de dois servidores dedicados com configurações distintas de performance para fornecer o serviço aos dois tipos de protocolos (GRIGORIK, 2013).

O novo formato de enquadramento foi produzido para um novo modelo de fluxo, que realizar toda comunicação via uma única conexão TCP em fluxo bidirecional entre o servidor e o cliente. Assim, toda comunicação poderá ser multiplexada numa mesma conexão TCP, removendo a característica sequencial dos protocolos anteriores (GRIGORIK, 2013). Na Figura 12, é mostrado o fluxo de uma única conexão com HTTP/2, mostrando que o quadro de dados não necessita ser enviado com o quadro de cabeçalhos, uma das vantagens da multiplexação em enquadramento binário, cada mensagem HTTP poderá ser composta de vários quadros (sejam de cabeçalho ou de dados) simultâneos e sem uma ordem pré-definida. Várias requisições poderão ser enviadas numa mesma conexão TCP e as respostas poderão ser recebidas em qualquer ordem, tornando a multiplexação uma

característica inerente ao HTTP/2, cujo novo fluxo se tornou a base para todas as outras otimizações da nova versão do HTTP.

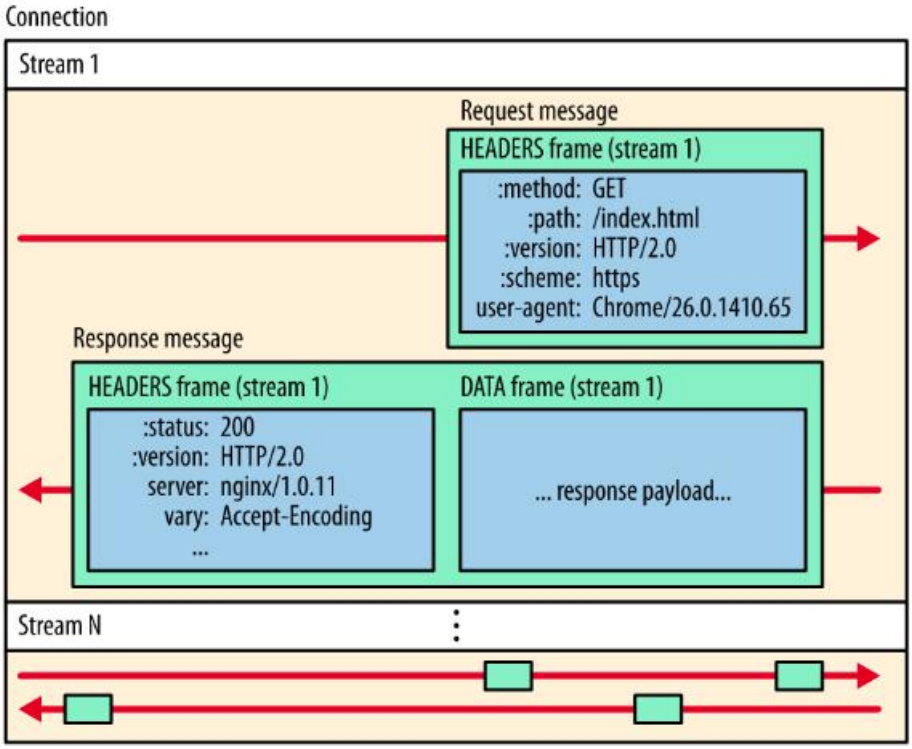


Figura 12: Conexão Persistente HTTP/2 (GRIGORIK, 2013)

4.3.2 Compressão

O HTTP/2 herdou a mesma sintaxe da versão 1.1, mas não o mesmo problema de duplicação de dados, sendo assim, essa redundância é eliminada e utiliza o algoritmo HPACK para isso. Na RFC 7541 (PEON; RUELLAN, 2015), o mecanismo HPACK utiliza uma codificação Huffman com auxílio de duas tabelas para identificar referências sempre utilizadas nos campos dos cabeçalhos, exigindo que cliente e servidor mantenham tabelas atualizadas com os campos de cabeçalho (DARWISH; ABDELWAHAB, 2016).

A Figura 13 mostra dois cabeçalhos. O cabeçalho da primeira requisição é enviado por completo, armazenado no cliente e, ao enviar a segunda requisição, os dados iguais no cabeçalho vão de forma implícita. E reduz significativamente o número de bytes enviados entre cliente e servidor a partir da segunda requisição, com isso o tamanho do quadro diminui e informações duplicadas não são trafegadas na rede de forma desneces-

sária (LUDIN; GARZA, 2017).

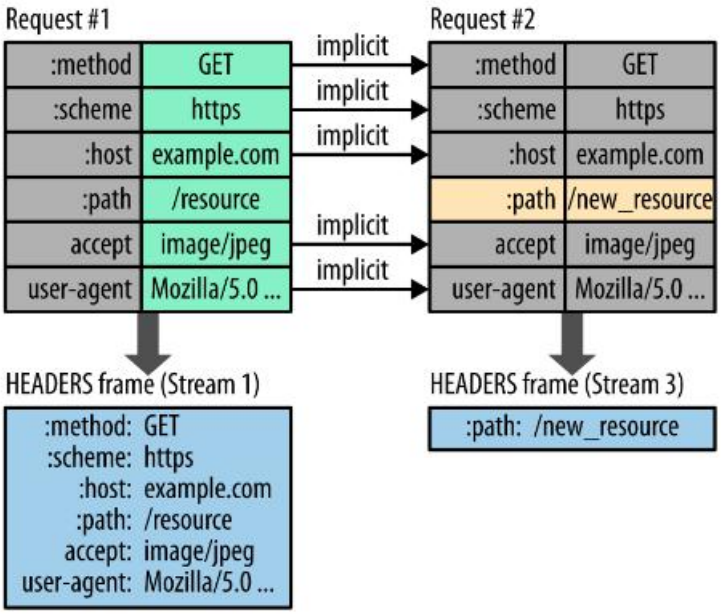


Figura 13: Compressão de Cabeçalhos (GRIGORIK, 2013)

4.3.3 Multiplexação

No novo protocolo, cada solicitação do cliente é marcada por um fluxo específico e todos esses fluxos relacionados às requisições separadas são multiplexados em uma única conexão TCP. O que permite que os fluxos não influenciem uns aos outros e que possam ser respondidos de forma simultânea pelo servidor (SAXCÉ; OPRESCU; CHEN, 2015).

A multiplexação permite que os quadros sejam totalmente independentes, podendo ter uma ordem intercalada e, ainda assim, serem reagrupados corretamente no destino. De forma a garantir ser possível intercalar várias requisições sem a necessidade de bloquear nenhuma delas (LUDIN; GARZA, 2017). Em razão disso, vale mencionar que não existe mais a necessidade de agrupar vários arquivos em um único para evitar conexões TCP múltiplas, e, nesse caso, quanto mais granular forem os arquivos, melhor para o fuxo e para a capacidade da rede que usará apenas uma única conexão TCP.

4.3.4 Prioridade

O sistema de prioridade poderá ser implementado especificamente para o caso do site que está a ser fornecido. A técnica prioriza certos componentes da página em detrimento de outros, de acordo com a necessidade, então, pode ser interessante marcar uma prioridade maior para imagens em oposição a conteúdos estáticos, caso seu site seja focado em imagens (DARWISH; ABDELWAHAB, 2016).

De acordo com o RFC 7540 (BELSHE; PEON; THOMSON, 2015), os fluxos multiplexados podem conter pesos (variando de 1 a 256) e dependências diretas entre eles. O cliente ou o próprio servidor pode criar um fluxo através de uma árvore de priorização que expressa como quer receber suas requisições, desse modo, o servidor aloca melhor os seus recursos. Esse processo pode ser realizado a partir de vários fluxos de prioridade que são associados a arquivos presentes no servidor.

4.3.5 Server Push

Server Push é uma funcionalidade que permite o envio de arquivos que ainda não foram solicitados e deve ser, possivelmente, uma das funcionalidades mais exploradas na nova versão do HTTP. No HTTP/1.1, a leitura do HTML era feita de forma estrutural e um componente só poderia ser requisitado na ordem em que aparecesse na página (GRIGORIK, 2013). Na versão 2, o servidor tem controle total do que pode ser enviado, podendo, inclusive, antecipar envios ainda não requisitados. Assim, se um site utiliza a mesma folha de estilo para todas as suas páginas, o servidor poderá enviar essa folha de estilo junto com a requisição inicial para que o conteúdo já seja armazenado no cache do cliente.

O servidor, desse modo, antecipa os recursos que o cliente pedirá e envia-os antecipadamente, diminuindo a latência de carregamento da página. Essa otimização deverá ser feita diretamente no servidor do site e é configurada de acordo com as necessidades dele, implicando em custo e manutenção maiores. Todavia, à medida que o usuário navegar pelo site, tempos menores de carregamento da página serão usufruídos, com amplo benefício para a qualidade de experiência vivenciada (DARWISH; ABDELWAHHAB,

2016).

4.4 ESTRATÉGIAS DE PERFORMANCE HTTP/2

No Capítulo 3, foram discutidas técnicas e boas práticas para melhoria da qualidade de experiência com o protocolo HTTP/1.1, no entanto, muitas dessas práticas se tornaram obsoletas com o advento do HTTP/2. Faremos, a seguir, uma revisão das recomendações apresentadas no Capítulo 3, levando em consideração as novas funcionalidades do HTTP/2.

Na versão 1.1 do protocolo, a regra de ouro era reduzir a quantidade de requisições, esperando, com isso, reduzir a necessidade de abertura de conexões TCP, o que poderia ser impedido ou limitado pelas configurações do sistema operacional ou do próprio navegador. Com HTTP/2, essa questão é superada, pois é feita apenas uma conexão TCP persistente para requisição e recebimento dos componentes da página.

Minimizar requisições implicará em minimizar o envio de quadros no HTTP/2, todavia, o carregamento da página será facilitado pela redução do envio de dados duplicados no cabeçalho, uma vez que foram enviados em algum outro quadro.

O uso de técnicas como Image Maps, CSS sprites e Imagens no inline do código HTML são totalmente descartáveis, pois no HTTP/2 a concatenação de arquivos inline deve ser evitada, pois o recurso inline impede o cacheamento dos arquivos, obrigando um reenvio de dados.

Com HTTP/2 é necessária uma estratégia diferente para obtenção de desempenho. Não é mais necessário se preocupar com o número de requisições enviadas, em vez disso, o desenvolvedor deverá deixar sua página web mais granular, adicionando atomicidade aos seus arquivos, deixando-os totalmente independentes. Ao reutilizar um mesmo arquivo em mais de uma página, o cliente web usará a informação no cache e diminuirá, consequentemente, a quantidade de bytes a serem transferidos com o servidor. Será recomendável adicionar um tratamento de prioridade aos arquivos, já que nem todos estarão em cache. Aliás, alguns arquivos poderão ser grandes o suficiente e necessitarão de um cuidado especial.

A possibilidade de usar o método de Server Push também dispensa a organização de CSS no topo da página, pois, se o CSS é essencial em sua página, é possível enviar todos os arquivos necessários sem o usuário sequer pedir. O cuidado com Javascript continua, dado que utilizá-lo, no meio da página, causará o bloqueio do carregamento, removendo as vantagens do novo protocolo. Logo a regra de Javascript no rodapé da página continua.

O uso de CDN continua válido para a nova versão do protocolo, porque o intuito da CDN é diminuir a distância entre usuário e o conteúdo estático que está a ser entregue, diminuindo assim a latência da rede. No caso do HTTP/2, o uso do host extra, para as decisões de encaminhamento da CDN, adiciona uma nova conexão TCP e uma nova tradução de DNS, no entanto, a redução do tempo de resposta, com uso da CDN, compensa esses pontos negativos. Técnicas para evitar traduções de DNS e redirecionamentos desnecessários continuam valendo e deverão ser observadas.

Também é válido o uso de cabeçalhos de expiração, porque, se o cache é vital para o funcionamento do HTTP/1.1, torna-se mais ainda para o HTTP/2, que opera, em geral, com maior granularidade de arquivos. Definir o que será armazenado mais tempo em cache se torna uma tarefa mais simples, pois arquivos não precisam mais ser concatenados.

Compactação de arquivos em Gzip e minificação dos arquivos continuam a ser recomendados, pois, quanto menor for o quadro enviado, mais rápida será a resposta para a tela do usuário.

4.5 SOBRE O FIM DO HTTP/1.1

A nova versão do protocolo trouxe muitas otimizações que beneficiam a performance da aplicação, porém, não necessariamente exclui o uso do HTTP/1.1. A versão 1.1 do protocolo ainda está presente, mesmo após três anos do nascimento da versão 2. De acordo com W3techs (2018), somente 24% de todos os sites utilizam HTTP/2 atualmente, conforme Figura 3.5.

Apesar do crescimento de aproximadamente 12% em um ano, como evidenciado

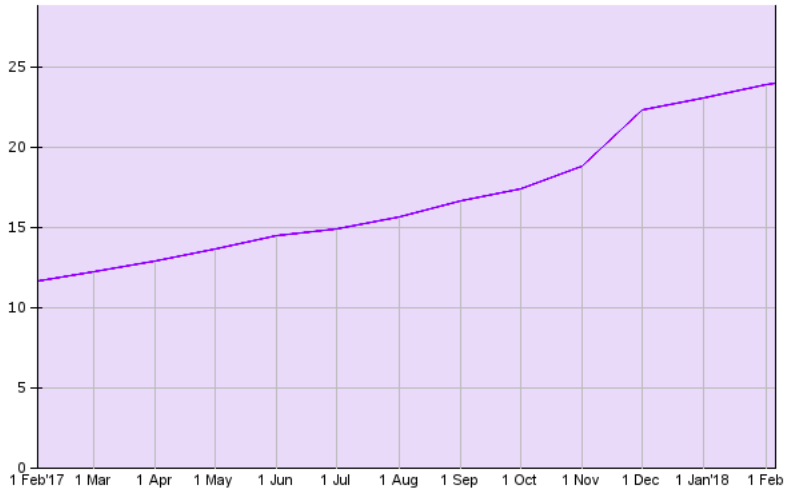


Figura 14: Crescimento do HTTP/2 em servidores no último ano 2017/2018

na Figura 15, o HTTP/2 ainda é implementado de forma tímida e está presente, sobretudo, em grandes portais de empresas que apostam nos benefícios da nova tecnologia. A Tabela 5 mostra o uso das versões HTTP nos principais sites do planeta.

Ao olhar o ranking dos principais sites da Moz (2018), pode-se verificar o uso do protocolo pelos 15 maiores sites da web, como mostra a tabela 3.1.

O uso do protocolo, ainda que para os principais sites, não é bem definido, mostrando que, mesmo com todos os "problemas" do HTTP/1.1, ainda é bastante performático para os dias atuais. O uso da versão 2 do protocolo é visto como solução para problemas específicos de grandes sites como, por exemplo, o Instagram que trabalha com bastante conteúdo estático como imagens. Entretanto, o Pinterest, que tem o mesmo objetivo do Instagram, utiliza a versão 1.1, como mostra a Tabela 5.

Tal constatação acerca do HTTP/2 é ainda fixada pelos proxies mais utilizados do mercado como Apache, Ngninx, Haproxy etc. Em todos os proxies, o protocolo padrão, para servir a página web, ainda é o HTTP/1.1, embora todos tenham portabilidade para a versão 2. Então existe uma migração a nova versão do protocolo, mas ainda timida e lenta.

O uso obrigatório de criptografia poderia ser uma motivação para que alguns sites não migrem para o protocolo. Os principais navegadores do mercado suportam criptografia de forma transparente, mas a maioria dos acesso atuais na web não é feita

Tabela 5: Ranking da versão HTTP que os principais sites utilizam (Moz, 2018)

Domínio	HTTP 1.1	HTTP/2
Facebook.com		X
Twitter.com		X
Google.com		X
Youtube.com		X
Instagram.com		X
Linkedin.com		X
Wordpress.org		X
Pinterest.com	X	
Wikipedia.org		X
Wordpress.com	X	X
Blogspot.com		X
Apple.com	X	
Adobe.com	X	
Tumblr.com	X	
Amazon.com	X	

via HTTPS. Embora a RFC 7540 (BELSHE; PEON; THOMSON, 2015) não obrigue o uso de TLS, implementações como h2[1], presentes nos principais navegadores, executam o protocolo somente via TLS, obrigando ao servidor a ter um certificado.

O certificado poderá ser obtido a partir de compra on-line e deverá ser renovado entre 1 ou 3 anos, ou via o serviço Let's Encrypt, totalmente gratuito, entretanto, necessita de uma renovação de certificados a cada 3 meses. O uso de certificados TLS ainda não é natural para grande parte da web, além de atrasar a resposta da requisição, porquanto necessita criptografar tudo que passa pelo canal de comunicação.

O objetivo do protocolo é que os usuários padrões do HTTP/1.1 possam migrar para a nova versão sem dificuldades e com benefícios. Em contrapartida, nem todos os sites conseguem visualizar esses benefícios ou não querem dar-se ao trabalho de atualizar o protocolo usado. Não é possível forçar todos a migrarem, em consequência da forma a qual são utilizadas as técnicas de proxy e cache, por isso, o HTTP/1.1 provavelmente ainda estará em uso por um longo tempo.

[1] https://http2.github.io/faq/

5 ESTUDO DE CASO

Neste capítulo será analisada uma página em produção e será identificada as melhorias de desempenho que as boas práticas discutidas no Capítulo 3 podem fornecer. Será analisada a página do Departamento de Ciência da Computação UFRJ[1] hospedada em um servidor HTTP/1.1. Ao final do Capítulo, será feita uma discussão sobre as vantagens que o HTTP/2 pode adicionar à performance da página e como atualizar para a nova versão.

Os testes foram realizados utilizando um servidor Amazon AWS[2] hospedado em São Paulo com uma banda de 5000 Kbps e 10 hops de distância da página hospedada no DCC. Nesse servidor, foi utilizado o navegador do Google Chrome[3] com as ferramentas PageSpeed (versão 1.15-gt1)[4] da Google e YSlow da Yahoo (versão 3.1.8)[5]. Tanto PageSpeed como YSlow são ferramentas que geram uma métrica, de acordo com as técnicas de performance, apontadas no Capítulo 3, o tempo de carregamento da página e as quantidades de componentes da página. Também foi utilizado um script[6] que foi adaptado para identificar algumas métricas de tempo para o carregamento da página.

O objetivo de ferramentas como PageSpeed e YSlow é criar um ranking geral em que qualquer site pode comparar sua performance com a de outros e obter dicas de melhoria, portanto, essas dicas fazem parte das métricas utilizadas pelas próprias empresas criadoras das ferramentas. Apesar de as dicas agregarem bastante aos desenvolvedores, seu método de ranking é desproporcional e não pode ser aplicado em qualquer site, devido à metodologia de design adotada pelas empresas que o produzem.

No PageSpeed, ferramenta da Google, é penalizado o excesso do uso de imagens, dessa maneira, sites que têm como regra de negócio o uso de bastantes imagens, nunca estarão no topo do ranking, mesmo que o tempo de carregamento seja aceitável para os padrões atuais. Assim, a pontuação utilizada por essas ferramentas não será considerada, mas sim as boas práticas que as ferramentas indicam para cada componente.

[1] http://www.dcc.ufrj.br/

[2] https://aws.amazon.com/pt/

[3] https://www.google.com/chrome/

[4] https://developers.google.com/speed/pagespeed/insights/?hl=pt-br

[5] http://yslow.org/

[6] https://github.com/micmro/performance-bookmarklet

TCT (s)	PB (s)	IR (s)	PIU (s)	# Requisições	Total de Bytes (KB)
9.632	0.836	2.266	2.312	88	4.902

Tabela 6: Tabela com média de experimemtos

A análise, realizada em 14 de agosto de 2018, levou em consideração três testes de performance em horários distintos (no periodo da manhã, tarde e noite), para retirar a possibilidade de gargalos ocasionais na rede. As métricas de interesse são o tempo médio de carregamento da página (TCT); o tempo médio de chegada do primeiro byte (PB) que chegou ao cliente; o tempo médio para o início de renderização (IR) da tela para o usuário; o tempo médio para o momento em que o usuário poderá inciar a primeira interação (PIU); o número de requisições que a página faz ao servidor; e o total de bytes trafegados para o carregamento total da página. As métricas TCT, PB IR e PIU são obtidas com uso de script, enquanto as duas últimas são obtidas diretamente das ferramentas.

A Tabela 6 mostra o resultado das métricas, obtidas com três medições. Não houve necessidade de um número maior de medições, pois os intervalos de confiança de 90% obtidos já foram plenamente satisfatórios, demonstrando pouca variação entre as medidas. O fato das medições terem acontecido antes do início do segundo semestre de 2017 pode ter contribuído para isso.

As medidas de tempo apresentadas na Tabela 6 estão em segundos e os intervalos de confiança obtidos foram [9.554 ; 9.710] para TCT, [0.774 ; 0.898] para PB, [2.196 ; 2.336] para IR, e [2.265 ; 2.359] para PIU. Evidentemente, o número de requisições e o total de bytes trafegados não mostraram variação, como esperado.

Para a análise da página, é necessário primeiro verificar a saída do script e os tempos médios que ele fornece. O anexo A mostra o modelo Waterfall ou Cascata de carregamento da página gerado pelo script. Somente o layout da página é carregado em 2,2s, ultrapassando a regra do 1 segundo para captar a atenção do usuário. A página está visualmente pronta para interação em 4,7s como mostra a Figura 15, inferior aos 10s (limite para abandono e frustração do usuário). Apesar disso, a página só é finalizada realmente após 9,6s de carregamento.

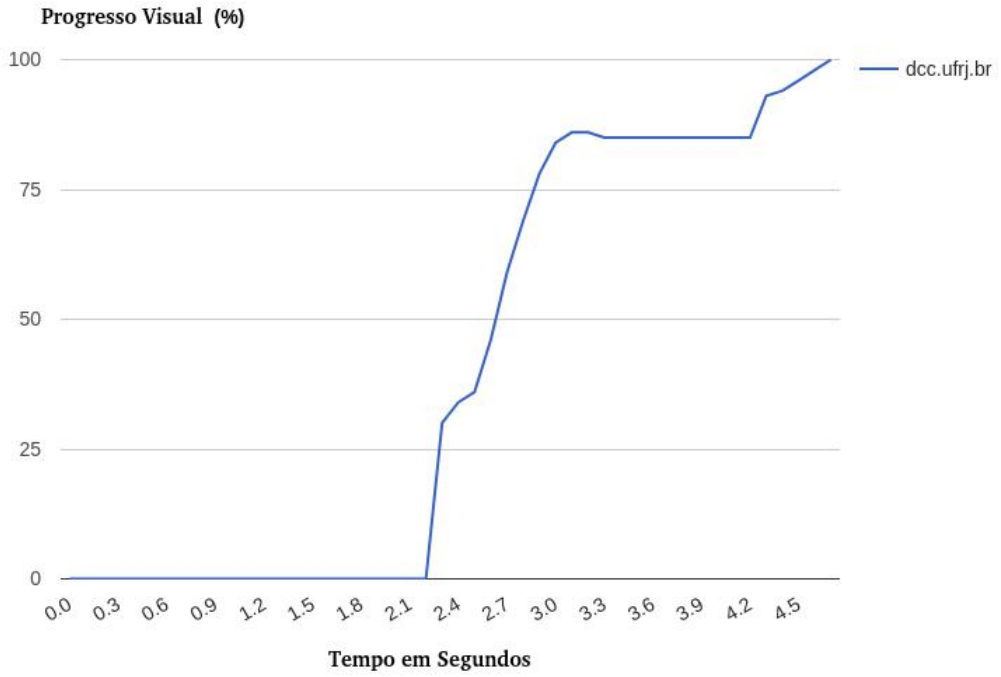


Figura 15: Progressão visual durante o carregamento da página

A Figura 15 mostra o progresso visual da página que se inicia em 2,2s e cresce até os 4,7s quando tem 100% de visualização. Esse progresso poderá ser observado melhor na Figura 16, que mostra quadro a quadro o carregamento da página.

De acordo com o PageSpeed/YSlow, a média do tamanho de uma página com desempenho aceitável, de acordo com o uso de banda mundial, está em 2.97 MB, a página inicial do DCC tem tamanho total de 4.9 MB. Sobre a quantidade de requisições feitas à página, são 89 requisições (HTTP), portanto, o site do DCC está de acordo com a média de 90 requisições por página já mencionada.

5.1 MELHORIAS DE PERFORMANCE

A seguir serão listadas todas as possíveis melhorias de performance de acordo com a análise de cada componente da página fornecidas pelo PageSpeed e Yslow.

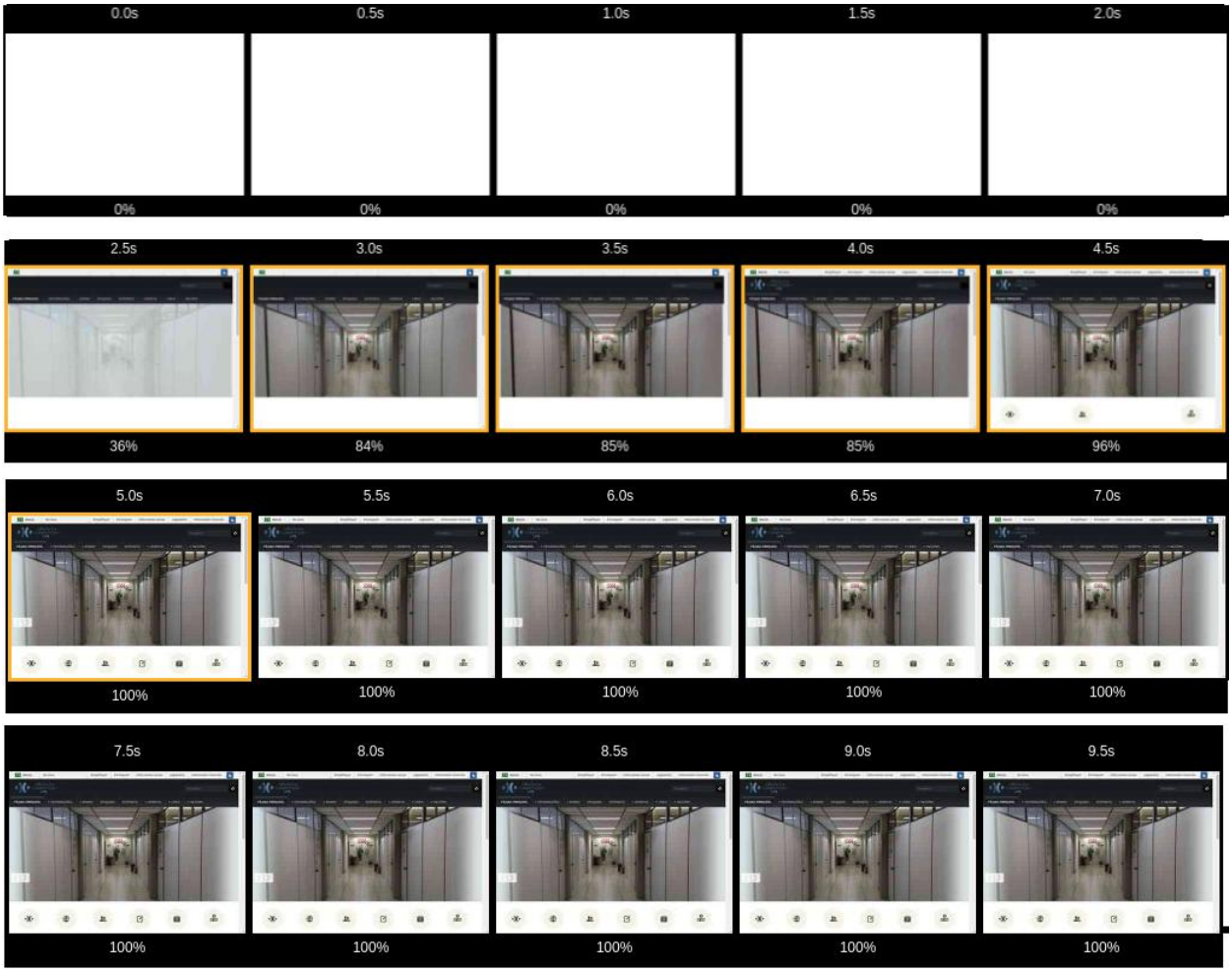


Figura 16: Visão da página durante o carregamento quadro a quadro

5.1.1 Dimensão das imagens

O maior problema da página do DCC reside em suas imagens, a maioria está fora das dimensões recomendadas. O site utiliza espaços destinados às imagens de resolução 160x120, porém, a imagem foi carregada com resolução 800x600, de forma que ocupa mais bytes no envio. A figura 17 mostra as imagens que estão dimensionadas de forma errada, portanto, as imagens só devem carregar na dimensão 800x600 quando forem acessadas diretamente pelo cliente. No total, pode-se salvar 3.7 MB (75% de todos os dados transferidos) da página somente redimensionando essas imagens. Os arquivos são listados no Anexo B.

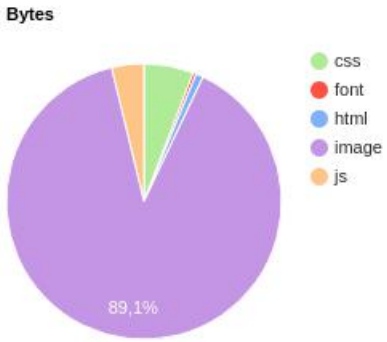
Figura 17: representação das imagens que devem ser redimensionadas na página

5.1.2 Recursos sem data de expiração

Os arquivos estáticos foram inseridos no cache baseando-se somente no campo ETag. Todas as imagens e os arquivos CSS não puderam ser armazenados com uma data de expiração, forçando uma requisição condicional para cada componente da página. O Anexo C lista todos os arquivos que devem ser configurados para estarem no cache com campo de expiração.

5.1.3 Recursos sem compressão

Os recursos não foram enviados pela rede com compressão GZIP e consumiram mais tempo para carregar a página. A Figura 18 gerada pelo script mostra essa relação em que, praticamente, não existe uma compressão (o JS é externo à aplicação). No total, 248.2 KB podem ser reduzidos na transferência para o cliente. O anexo D lista todos os arquivos que devem ser enviados ao cliente com compressão.



MIME Type	Bytes	Uncompressed
image	4.469.308	4.455.456
css	291.971	291.982
js	190.513	583.068
html	43.248	42.493
font	18.857	18.476
flash	0	0
other	0	0
video	0	0

Figura 18: Comparação de compressão em Bytes

5.1.4 Recursos sem minificação

A maioria dos componentes da página está minificada, contudo, ainda existem alguns arquivos sem a técnica, ocasionando mais bytes desnecessários no envio. Esses recursos, no total, podem salvar 66.8 KB de arquivos estáticos de texto. O anexo E lista todos os arquivos que devem ser minificados.

5.1.5 Uso de CSS Sprites

Alguns arquivos são apenas sprites que precisam ser concatenados para diminuir o número de requisições. O Anexo F lista esses arquivos que podem ser concatenados.

5.1.6 Uso de CSS Inline

Alguns arquivos de folha de estilo são pequenos demais para serem externos, portanto, adicioná-los diretamente à página poderá reduzir o tempo de renderização. O Anexo G lista os arquivos CSS que podem ser adicionados ao HTML.

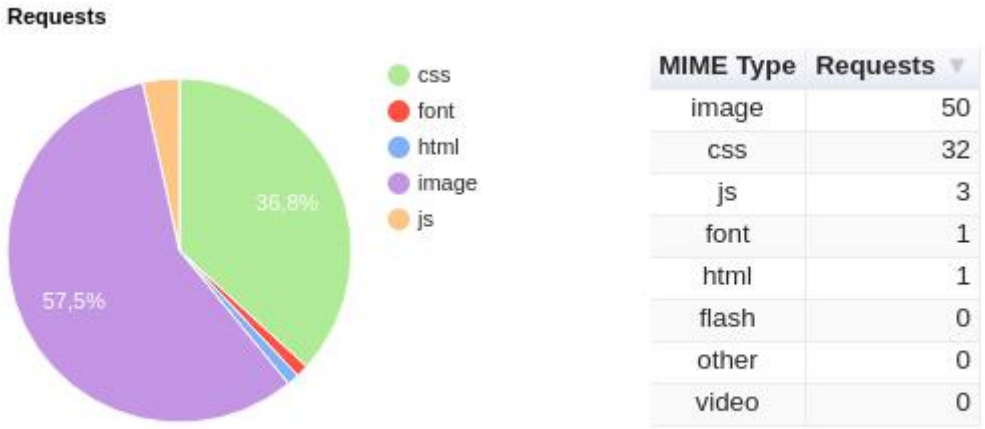
5.1.7 Uso de CDN

O site do DCC não possui uma CDN dedicada, conforme a Figura 18 que foi gerada pelo script, 89,1% da página em bytes é composta por imagens. De acordo com a Figura 19, 94,7% das requisições são feitas com imagens e CSS, de forma que o uso de CDN é indicado para melhorar o tempo de resposta da página.

5.2 PORTABILIDADE HTTP/2

O site do DCC é totalmente portável para a versão HTTP/2, sendo necessário apenas habilitar esse módulo no Apache pelo CMS Joomla[7], que hospeda a aplicação. As vantagens, apresentadas no Capítulo 4, mantêm-se aqui, como a compressão de cabeça-

[7] https://www.joomla.org/



MIME Type	Requests
image	50
css	32
js	3
font	1
html	1
flash	0
other	0
video	0

Figura 19: Comparação entre as requisições

lhos, multiplexação e a fila de prioridades. Quanto ao Server Push, pode ser implementado no Joomla com plugins como HTTP/2 Push da Bluewall[8] e HTTP/2 Server Push da Morza[9]. Com o uso desses plugins, é possível enviar arquivos de CSS e imagens de outras páginas, além de deixá-los no cache do cliente para futuras consultas.

[8]https://extensions.joomla.org/extension/http-2-push/

[9]https://extensions.joomla.org/extension/http-2-server-push/

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O desenvolvimento do presente estudo possibilitou o aprofundamento do principal protocolo web e de todas as suas principais versões com vistas para melhorar o desempenho de suas páginas. O desempenho de uma página web é importante para a experiência do usuário e a garantia de sucesso para um site.

O estudo acerca de HTTP é um tema recorrente na literatura acadêmica, em contrapartida, mediante pesquisa bibliográfica, a melhoria, na performance desse protocolo para a web atual, mostrou-se não ser o cerne de pesquisas. Ainda assim, trata-se de um tema explorado, sobretudo, por grandes empresas, e a evolução dela foi realizada por meio de experimentos cujo intuito era se adaptar às limitações impostas pelo próprio protocolo e pela gestão de recursos, tanto em cliente como em servidor.

Na maioria dos casos, a falta de desempenho de um site não está na limitação de recursos de hardware, mas na falta de conhecimento de desenvolvedores que não seguem recomendações básicas para o desenvolvimento de uma página web. O trabalho identificou técnicas e boas práticas levando em consideração as limitações conhecidas do HTTP que, na maioria dos casos, são fáceis de implementar e trazem grandes resultados. Outros pontos importantes que afetam o desempenho foram abordadas, como a importância do cache para armazenar dados no cliente, organização de componentes estáticos no HTML, uso de CDN, traduções de DNS e o problema dos redirecionamentos. Foram também identificadas as seguintes técnicas : image maps, css sprites, inline images, combinação scripts e CSS, além da otimização de imagens. Todas essas técnicas favorecem a redução do número de requisições.

O trabalho também abordou a discussão sobre a mais nova versão do protocolo se preocupando em apresentar as principais funcionalidades para reforçar as deficiências da versão 1.1, as quais, como foram constatadas, ainda estão muito longe de serem abolidas da web atual.

O trabalho ainda realizou a análise de performance do site do Departamento de Ciência da Computação da UFRJ. A análise da página do DCC foi realizada sob o ponto de vista do usuário, observando a progressão visual durante o seu carregamento. Além do

script para obter a representação em cascata dos tempos para a obtenção dos componentes da página, foram utilizadas as ferramentas PageSpeed e YSlow para obtenção do número de bytes trafegados, total de requisições e possíveis melhorias de performance para cada componente. O aprofundamento do estudo do HTTP e a análise de performance das páginas web pode ser visto como complementar ao conhecimento atualmente coberto nas disciplinas do Bacharelado em Ciência da Computação.

Como orientação para trabalhos futuros, o foco deve estar no HTTP/2 , visto que está cada vez mais aceito e, certamente, será o futuro da Web. Tendo funcionalidades bem diferentes às do HTTP/1.1, um campo extenso para trabalhos de P&D se abre e deve ser de grande interesse para os próximos anos. Técnicas de push server e análise automática de filas de prioridade podem melhorar o desempenho dos sistemas web. Em paralelo, o estudo de novos algoritmos para distribuição de conteúdos estáticos por geolocalização e novas técnicas de cache para páginas web podem ser produtivos. E, finalmente, o estudo de novos protocolos feitos para web, como o protocolo QUIC, possível substituto do TCP para a Web, aderem aos temas a serem prospectados para os trabalhos futuros.

Referências

AGABABOV, V. et al. **Flywheel:** google's data compression proxy for the mobile web. In: *NSDI*. [S.l.: s.n.], 2015. v. 15, p. 367–380.

APACHE. ***Caching Guide***. Disponível em: <https://httpd.apache.org/docs/2.4/caching.html>. Acessado em: 30 Jul. 2018.

BEERTEMA, P. ***Common DNS Data File Configuration Errors***. [S.l.], 1993.

BELSHE, M.; PEON, R.; THOMSON, M. ***Hypertext Transfer Protocol Version 2 (HTTP/2)***. [S.l.], 2015. http://www.rfc-editor.org/rfc/rfc7540.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc7540.txt>.

CALDER, M. et al. **Analyzing the Performance of an Anycast CDN**. In: ACM. *Proceedings of the 2015 Internet Measurement Conference*. [S.l.], 2015. p. 531–537.

CARD, S. K.; ROBERTSON, G. G.; MACKINLAY, J. D. **The information visualizer, an information workspace**. In: ACM. *Proceedings of the SIGCHI Conference on Human factors in computing systems*. [S.l.], 1991. p. 181–186.

CISCO, I. **Cisco visual networking index:** forecast and methodology, 2011–2016. *CISCO White paper*, v. 518, 2012.

DARWISH, N. R.; ABDELWAHAB, I. M. **Impact of Implementing HTTP/2 in Web Services**. *International Journal of Computer Applications*, Foundation of Computer Science, v. 147, n. 9, 2016.

DEUTSCH, L. P. et al. ***GZIP file format specification version 4.3***. [S.l.], 1996. http://www.rfc-editor.org/rfc/rfc1952.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc1952.txt>.

DOMES, S. ***Progressive Web Apps with React:*** *Create lightning fast web apps with native power using React and Firebase*. [S.l.]: "Packt Publishing Ltd", 2013.

FIELDING, R.; RESCHKE, J. ***Hypertext Transfer Protocol (HTTP/1.1):*** *Conditional Requests*. [S.l.], 2014. http://www.rfc-editor.org/rfc/rfc7232.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc7232.txt>.

FIELDING, R. T. et al. ***Hypertext Transfer Protocol – HTTP/1.1***. [S.l.], 1999. http://www.rfc-editor.org/rfc/rfc2616.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc2616.txt>.

FILHO, C. B. d. P. **Estratégia de web cache utilizando redes P2P de clientes sobre WebRTC**. In: . [S.l.: s.n.], 2017.

GLASSMAN, S. **A caching relay for the World Wide Web**. *Computer Networks and ISDN systems*, Citeseer, v. 27, n. 2, p. 165–173, 1994.

GOEL, U.; WITTIE, M. P.; STEINER, M. **Faster Web through Client-assisted CDN Server Selection**. In: IEEE. *Computer Communication and Networks (ICCCN), 2015 24th International Conference on*. [S.l.], 2015. p. 1–10.

GOURLEY, D. et al. ***HTTP: the definitive guide***. [S.l.]: "O'Reilly Media, Inc.", 2002.

GRIGORIK, I. ***High Performance Browser Networking:*** *What every web developer should know about networking and web performance*. [S.l.]: "O'Reilly Media, Inc.", 2013.

HAYTON, S. J. et al. ***Using entity tags (ETags) in a hierarchical HTTP proxy cache to reduce network traffic***. [S.l.]: Google Patents, 2016. US Patent 9,253,278.

IHM, S.; PAI, V. S. **Towards understanding modern web traffic**. In: ACM. *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. [S.l.], 2011. p. 295–312.

JOVANOVSKI, G.; ZAYTSEV, V. **Critical CSS Rules—Decreasing time to first render by inlining CSS rules for over-the-fold elements**. In: *Postproceedings of 2016 Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*. [S.l.: s.n.], 2016. p. 353–356.

KHANAL, K.; JAGADISH, A.; ANNAMALAISAMI, S. ***Systems and methods for ETAG persistency***. [S.l.]: Google Patents, 2018. US Patent 9,866,529.

KOECHLEY, N. ***High Performance Web Sites***. Disponível em: <http://nate.koechley.com/talks/2007/atmedia-london/high-performance-web-sites.pdf>. Accessado em: 29 Jul. 2018.

KUENZER, S. et al. **Unikernels Everywhere:** the case for elastic cdns. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2017. v. 52, n. 7, p. 15–29.

KUMAR, K. S.; VENKATESAN, G. P. **Certain Investigation in DNS Performance by Using Accelerator and Stub Network**. In: IEEE. *Computational Intelligence and Communication Networks (CICN), 2016 8th International Conference on*. [S.l.], 2016. p. 172–176.

LIU, X. et al. **A case for a coordinated internet video control plane**. *ACM SIGCOMM Computer Communication Review*, ACM, v. 42, n. 4, p. 359–370, 2012.

LUDIN, S.; GARZA, J. ***Learning HTTP/2:*** *A Practical Guide for Beginners*. [S.l.]: "O'Reilly Media, Inc.", 2017.

MACCAW, A. ***JavaScript Web Applications:*** *JQuery Developers' Guide to Moving State to the Client*. [S.l.]: "O'Reilly Media, Inc.", 2011.

MASINTER, L. ***The "data"URL scheme***. [S.l.], 1998. http://www.rfc-editor.org/rfc/rfc2397.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc2397.txt>.

MATSUSHITA, K.; NISHIMINE, M.; UEDA, K. **Cooperative Cache Distribution System for Virtual P2P Web Proxy**. In: IEEE. *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*. [S.l.], 2015. v. 3, p. 646–647.

MOCKAPETRIS, P. **RFC 1034:** domain names: concepts and facilities (november 1987). *Status: Standard*, v. 6, 2003.

MOZ. ***7 tips for faster http2 performance***. Disponível em: <https://www.nginx.com/blog/7-tips-for-faster-http2-performance/>. Acessado em: 29 Jul. 2018.

MOZ. ***Usage of HTTP/2 for websites***. Disponível em: <https://moz.com/top500>. Acessado em: 29 Jul. 2018.

NAGY, Z. **Improved speed on intelligent web sites**. *Recent Advances in Computer Science*, v. 1, n. 14, p. 215–220, 2013.

PATHAN, M.; BUYYA, R.; VAKALI, A. **Content delivery networks:** state of the art, insights, and imperatives. In: *Content Delivery Networks*. [S.l.]: Springer, 2008. p. 3–32.

PEON, R.; RUELLAN, H. ***HPACK:*** *Header Compression for HTTP/2*. [S.l.], 2015.

QI, S. et al. **A study of information richness and downloading time for hotel websites in Hong Kong**. *Information and Communication Technologies in Tourism 2008*, Springer, p. 267–278, 2008.

RAZA, A. et al. **Extreme web caching for faster web browsing**. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2015. v. 45, n. 4, p. 111–112.

SAKAMOTO, Y. et al. **Empirical study on effects of script minification and http compression for traffic reduction**. In: IEEE. *Digital Information, Networking, and Wireless Communications (DINWC), 2015 Third International Conference on*. [S.l.], 2015. p. 127–132.

SAXCÉ, H. de; OPRESCU, I.; CHEN, Y. **Is HTTP/2 really faster than HTTP/1.1?** In: IEEE. *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*. [S.l.], 2015. p. 293–299.

SCHECHTER, S.; KRISHNAN, M.; SMITH, M. D. **Using path profiles to predict HTTP requests**. *Computer Networks and ISDN Systems*, Elsevier, v. 30, n. 1-7, p. 457–467, 1998.

SHAILESH, K.; SURESH, P. **An Analysis Of Techniques And Quality Assessment For Web Performance Optimization**. *Indian Journal of Computer Science and Engineering (IJCSE)*, v. 8, p. 61–69, 2017.

SHIM, J.; SCHEUERMANN, P.; VINGRALEK, R. **Proxy cache algorithms:** design, implementation, and performance. *IEEE Transactions on Knowledge & Data Engineering*, IEEE, n. 4, p. 549–562, 1999.

SINGH, K.; KRISHNA, C.; KUMAR, S. **A New Way of Accelerating Web by Compressing Data with Back Reference-Prefer Geflochtener.** *International Arab Journal of Information Technology (IAJIT)*, v. 14, n. 4, 2017.

SOUDERS, S. ***High Performance Web Sites:*** *Essential Knowledge for Front-End Engineers. sl*. [S.l.]: O'Reilly Media, 2006.

VAKALI, A.; PALLIS, G. **Content delivery networks:** status and trends. *IEEE Internet Computing*, IEEE, v. 7, n. 6, p. 68–74, 2003.

VESUNA, J. et al. **Caching Doesn't Improve Mobile Web Performance (Much).** In: *USENIX Annual Technical Conference*. [S.l.: s.n.], 2016. p. 159–165.

VISCOMI, R.; DAVIES, A.; DURAN, M. ***Using WebPageTest:*** *Web Performance Testing for Novices and Power Users*. [S.l.]: "O'Reilly Media, Inc.", 2015.

W3TECHS. ***Usage of HTTP/2 for websites***. Disponível em: <https://w3techs.com/technologies/details/ce-http2/all/all>. Accessado em: 30 Jul. 2018.

WANG, X. S. et al. **Demystifying Page Load Performance with WProf.** In: *NSDI*. [S.l.: s.n.], 2013. p. 473–485.

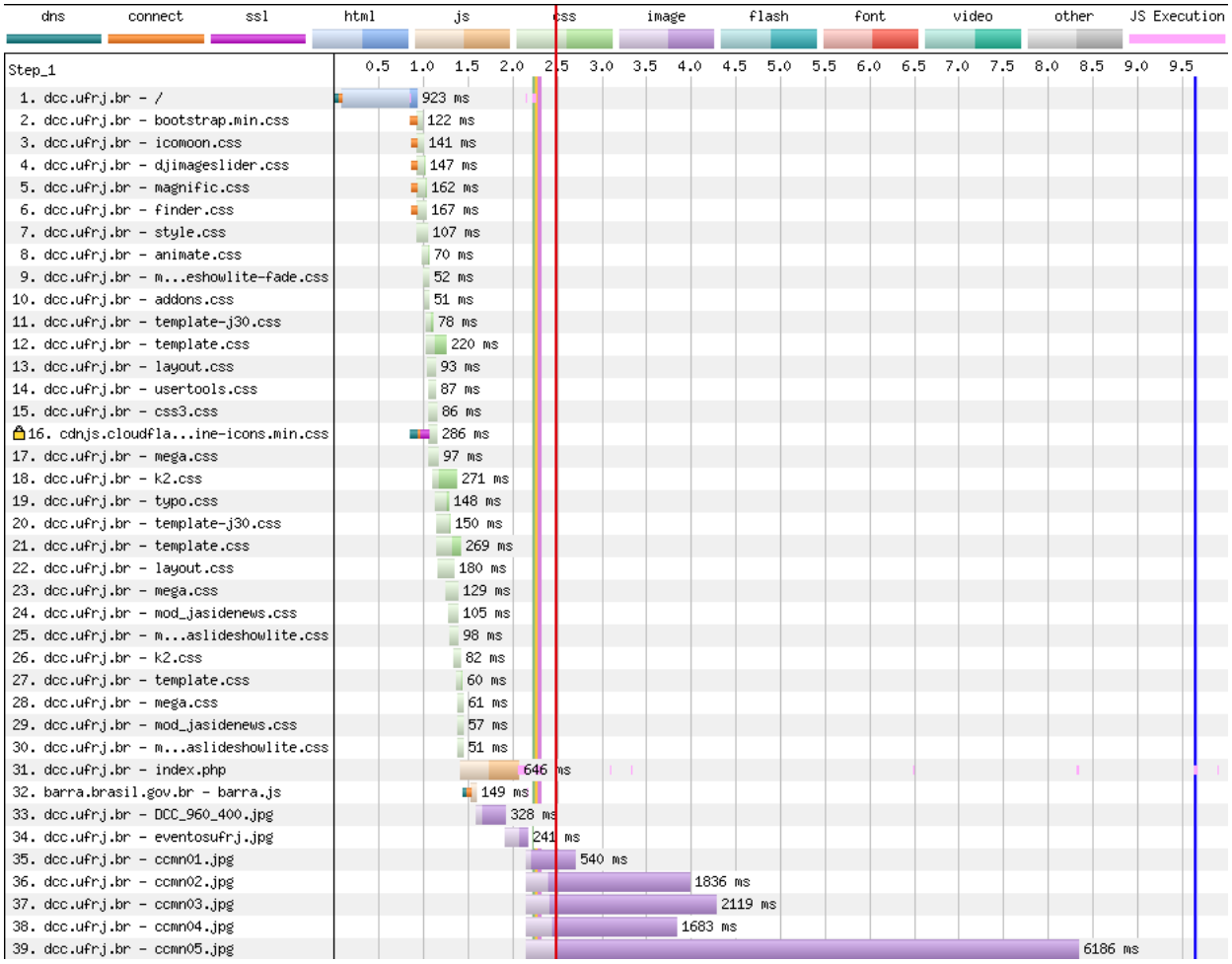
XU, Z.; HUANG, R.; BHUYAN, L. N. **Load balancing of dns-based distributed web server systems with page caching**. In: IEEE. *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*. [S.l.], 2004. p. 587–594.

ZAKAS, N. C. ***High Performance JavaScript:*** *Build Faster Web Application Interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2010.

ANEXOS

ANEXO A – Requisição ao DCC

O modelo cascata fornece o tempo de carregamento de cada componente. Na parte superior do modelo existe uma legenda com as respectivas cores de cada componente da página, além de uma representação por intensidade de cores para identificar quando ocorre a renderização do componente em página (registrado pela cor mais intensa). Na parte inferior são representados dados como o uso de CPU para a renderização de acordo com o tempo, uso de banda com o tempo, uso de thread de acordo com o tempo e o momento de inicio de interação da página.



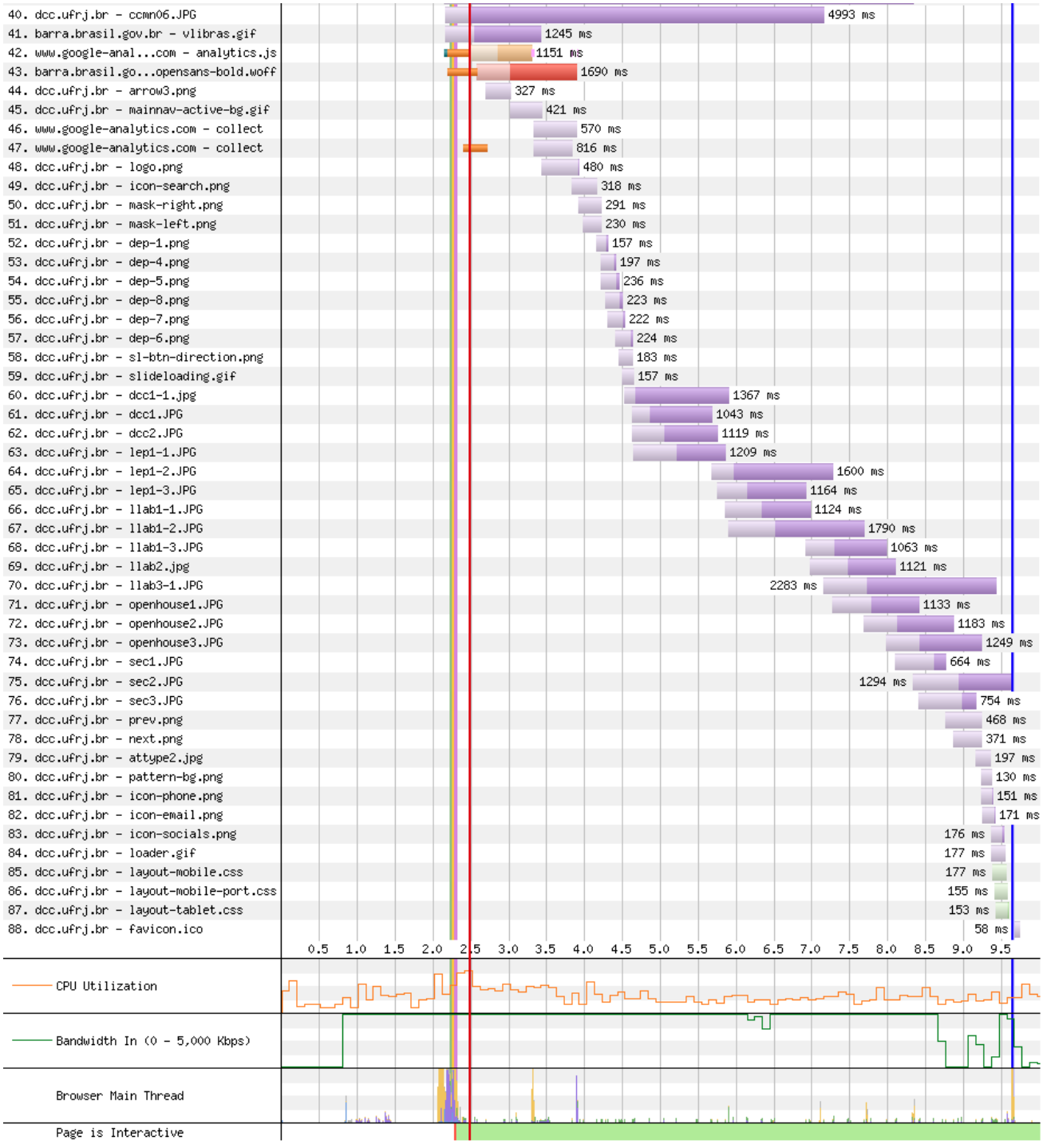


Figura 20: Modelo Waterfall de requisições a página do DCC

ANEXO B – Imagens não redimensionadas

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn03.jpg é redimensionada

em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 288,1 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-2.JPG é redimensionado em HTML ou CSS de 800x600 a 160x120. A exibição de uma imagem dimensionada pode economizar 261,8 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn06.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 233,6 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn02.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 215,2 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn05.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 192,1 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-2.JPG é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 190,4 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc1-1.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 179,4 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn04.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 170,3 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn01.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 169,6 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//llab2.jpg é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 169,5 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 164,5 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 149,6 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 146,9 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse2.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 146,6 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse3.JPG é redimensionada em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 141,5 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc2.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 135,7 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 129,3 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-3.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 118,8 KB (redução de 96%).

- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-3.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 115,1 KB (redução de 96%).
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec2.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 112,9 KB (redução de 96%).
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab3-1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 111,6 KB (redução de 96%).
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec3.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 109,5 KB (redução de 96%).
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec1.JPG é redimensionado em HTML ou CSS de 800x600 para 160x120. A exibição de uma imagem dimensionada pode economizar 98,4 KB (redução de 96%).

ANEXO C – Arquivos sem data de expiração

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn01.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn02.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn03.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn04.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn05.jpg (expiração não especificada)

- http://www.dcc.ufrj.br/images/JA_University/DCC//ccmn06.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc1-1.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc1.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//dcc2.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-1.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-2.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//lep1-3.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-1.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-2.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab1-3.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab2.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//llab3-1.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse1.JPG (expiração não especificada)

- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse2.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//openhouse3.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec1.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec2.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/DCC//sec3.JPG (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/home/attype2.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/JA_University/home/eventosufrj.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/images/resized/images/JA_University/slides/DCC_960_400.jpg (expiração não especificada)
- http://www.dcc.ufrj.br/media/com_finder/css/finder.css (expiração não especificada)
- http://www.dcc.ufrj.br/media/djextensions/magnific/magnific.css (expiração não especificada)
- http://www.dcc.ufrj.br/media/jui/css/icomoon.css (expiração não especificada)
- http://www.dcc.ufrj.br/media/modals/css/bootstrap.min.css (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/css/djimageslider.css (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/images/loader.gif (expiração não especificada)

- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/images/next.png (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/images/prev.png (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_jasidenews/asset/style.css (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/css/animate.css (expiração não especificada)
- http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/css/mod_jaslideshowlite-fade.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/addons.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/css3.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/layout.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/menu/mega.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/template-j30.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/template.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/usertools.css (expiração não especificada)
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/images/arrow3.png (expiração não especificada)

- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/images/favicon.ico (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/k2.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile-port.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/layout-tablet.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/layout.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/menu/mega.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/mod_jasidenews.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/mod_jaslideshowlite.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/template-j30.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/template.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/css/typo.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-1.png (expiração não especificada)

- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-4.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-5.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-6.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-7.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/icons/dep-8.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/mask-left.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/mask-right.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/sl-btn-direction.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/images/slideloading.gif (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/k2.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/menu/mega.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/mod_jasidenews.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/mod_jaslideshowlite.css (expiração não especificada)

- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/template.css (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/icon-email.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/icon-phone.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/icon-search.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/icon-socials.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/logo.png (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/mainnav-active-bg.gif (expiração não especificada)
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/pattern-bg.png (expiração não especificada)

ANEXO D – Folhas de Estilo sem compressão

- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/k2.css pode economizar 51.6 KB (redução de 85%).
- A compactação de http://www.dcc.ufrj.br/ pode poupar 33,5 KB (redução de 80%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/template.css pode poupar 28,8 KB (redução de 80%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/template.css pode poupar 20,4 KB (redução de 76%).

- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/template-j30.css pode poupar 14.2 KB (redução de 77%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/typo.css pode poupar 13,0 KB (redução de 76%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile.css pode poupar 10.6 KB (redução de 78%).
- A compactação de http://www.dcc.ufrj.br/media/jui/css/icomoon.css pode poupar 9,3 KB (redução de 80%).
- A compactação de http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/ css/animate.css poderia salvar 7.7KB (redução de 89%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/themes/iron/ css/template.css pode economizar 5.9KB (redução de 76%).
- A compactação de http://www.dcc.ufrj.br/media/djextensions/magnific/magnific.css pode poupar 5,7 KB (redução de 74%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/template-j30.css pode economizar 5,4 KB (redução de 71%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/menu/mega.css pode poupar 4,8 KB (redução de 79%).
- A compactação do http://www.dcc.ufrj.br/templates/ja_university/css/layout-tablet.css pode poupar 4,8 KB (redução de 73%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile-port.css pode poupar 3,5 KB (redução de 74%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/k2.css pode poupar 3,0 KB (redução de 77%).
- A compactação do http://www.dcc.ufrj.br/media/modals/css/bootstrap.min.css pode economizar 2.9 KB (redução de 73%).

- A compactação do http://www.dcc.ufrj.br/templates/ ja_university/css/mod_jaslideshowlite.css pode economizar 2.7 KB (redução de 69%).
- A compactação de http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/ css/djimageslider.css pode poupar 2,5 KB (redução de 73%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/ css/usertools.css pode poupar 2,4 KB (redução de 69%).
- A compactação do http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/ menu/mega.css pode economizar 2.3 KB (redução de 73%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/ css/layout.css pode economizar 2.2 KB (redução de 70%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/menu/mega.css pode poupar 1,6 KB (redução de 66%).
- A compactação de http://www.dcc.ufrj.br/media/com_finder/css/finder.css pode poupar 1,6 KB (redução de 64%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/ default/css/addons.css pode poupar 1,5 KB (redução de 57%).
- A compactação de http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/ default/css/css3.css pode poupar 1,5 KB (redução de 71%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/mod_jasidenews.css pode economizar 1.2 KB (redução de 62%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/css/layout.css pode poupar 1,1 KB (redução de 65%).
- A compactação de http://www.dcc.ufrj.br/modules/mod_jasidenews/asset/style.css pode poupar 1,0 KB (redução de 60%).
- A compactação http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/ css/mod_jaslideshowlite fade.css pode economizar 517 B (redução de 60%).

- A compactação do http://www.dcc.ufrj.br/templates/ja_university/themes/iron/ css/mod_jasidenews.css pode economizar 334B (redução de 53%).
- A compactação de http://www.dcc.ufrj.br/templates/ja_university/themes/iron/ css/mod_jaslideshowlite.css pode economizar 329 B (redução de 52%).

ANEXO E – Componentes sem minimificação

- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/k2.css pouparia 12,2 KB (redução de 21%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/template.css pouparia 7,3 KB (redução de 21%).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/jat3/ base-themes/default/css/template.css pouparia 5,7 KB (22% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile.css pouparia 4,4 KB (33% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/typo.css pouparia 4,1 KB (25% de redução).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/ jat3/base-themes/default/ css/template-j30.css pouparia 2,7 KB (redução de 15%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/template.css pouparia 2,5 KB (redução de 33%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/layout-tablet.css pouparia 2,4 KB (37% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/k2.css pouparia 2,1 KB (redução de 54%).

- Minificar http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/css/animate.css pouparia 1,8 KB (22% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/template-j30.css pouparia 1,8 KB (24% de redução).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/jat3/ base-themes/default/css/layout.css pouparia 1,8 KB (redução de 55%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/layout-mobile-port.css pouparia 1,7 KB (37% de redução).
- Minificar http://www.dcc.ufrj.br/media/djextensions/magnific/magnific.css pouparia 1,6 KB (22% de redução).
- Minificar http://www.dcc.ufrj.br/plugins/system/ jat3/jat3/base-themes/ default/css/menu/mega.css pouparia 1,5 KB (24% de redução).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/ jat3/base-themes/ default/css/addons.css pouparia 1,4 KB (redução de 53%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/mod_jaslideshowlite.css pouparia 1,2 KB (32% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/menu/mega.css pouparia 1,1 KB (36% de redução).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/ jat3/base-themes/default/ css/usertools.css pouparia 1,1 KB (32% de redução).
- Minificar http://www.dcc.ufrj.br/media/jui/css/icomoon.css pouparia 1,1 KB (redução de 10%).
- Minificar http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/css/css3.css pouparia 1,1 KB (52% de redução).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/layout.css pouparia 1,1 KB (62% de redução).

- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/menu/mega.css pouparia 966 B (redução de 40%).
- Minificar http://www.dcc.ufrj.br/templates/ja_university/css/mod_jasidenews.css pouparia 863 B (redução de 43%).
- Minificar http://www.dcc.ufrj.br/modules/mod_jasidenews/asset/style.css pouparia 807 B (redução de 47%).
- Minificar http://www.dcc.ufrj.br/modules/ mod_djimageslider/themes/default/ css/djimageslider.css pouparia 630 B (19% de redução).
- Minificar http://www.dcc.ufrj.br/modules/ mod_jaslideshowlite/assets/css/ mod_jaslideshowlite-fade.css pouparia 598 B (redução de 70%).
- Minificar http://www.dcc.ufrj.br/templates/ ja_university/themes/iron/ css/mod_jasidenews.css pouparia 579 B (redução de 92%).
- Minificar http://www.dcc.ufrj.br/templates/ ja_university/themes/iron/ css/mod_jaslideshowlite.css poderia economizar 571 B (redução de 91%).
- Minificar http://www.dcc.ufrj.br/media/com_finder/css/finder.css pouparia 215 B (redução de 9%).
- Minificar https://cdnjs.cloudflare.com/ajax/libs/simple-line-icons/2.4.1/css/simple-line-icons.min.css poderia economizar 37 B (redução de 2%).
- Minificar http://www.dcc.ufrj.br/ pouparia 7.8 KB (19% de redução) [1]

ANEXO F – Componentes que podem ser usados em CSS Sprites

- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/images/next.png
- http://www.dcc.ufrj.br/modules/mod_djimageslider/themes/default/images/prev.png
- http://www.dcc.ufrj.br/plugins/system/jat3/jat3/base-themes/default/images/arrow3.png

[1] Todas as páginas HTML devem ser minificadas também

- http://www.dcc.ufrj.br/templates/ja_university/images/sl-btn-direction.png
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/icon-search.png
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/mainnav-active-bg.gif
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/images/pattern-bg.png

ANEXO G – Folhas de estilo que podem ser adicionados no código

- http://www.dcc.ufrj.br/modules/mod_jaslideshowlite/assets/css/mod_jaslideshowlite-fade.css
- http://www.dcc.ufrj.br/templates/ja_university/css/layout.css
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/mod_jasidenews.css
- http://www.dcc.ufrj.br/templates/ja_university/themes/iron/css/mod_jaslideshowlite.css