FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE JOSÉ DUTRA GOMES

ESTRATÉGIAS DE IMPLEMENTAÇÃO DE CACHE PARA OTIMIZAÇÃO DE SERVIÇOS WEB

Marília
2013

ALEXANDRE JOSÉ DUTRA GOMES

ESTRATÉGIAS DE IMPLEMENTAÇÃO DE CACHE PARA OTIMIZAÇÃO DE SERVIÇOS WEB

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência Da Computação.
Orientador: Prof. Ms. Ricardo Sabatine

Marília
2013

GOMES, Alexandre José Dutra
Estratégias De Implementação De Cache Para Otimização De Serviços Web / Alexandre José Dutra Gomes; orientador: Profª. Ms. Ricardo José Sabatine. Marília, SP: [s.n.], 2013.
74 f.

Trabalho de Conclusão de Curso (TCC) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha.

1. Web Cache 2. Serviços Web

CDD: 005.2

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Alexandre José Dutra Gomes

Estratégias De Implementação De Cache Para Otimização De Serviços Web

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10 (Dez)

Orientador: Ricardo José Sabatine — Ricardo Sabatine

1°. Examinador: Emerson Alberto Marconato

2°. Examinador: Giulianna Marega Marques

Marília, 04 de dezembro de 2013.

Às minhas mães, por todo amor, dedicação e incentivo.

Aos amigos, pela cobrança.

À minha noiva, companheira e mulher da minha vida,
Danielle Machado, pelo incentivo, apoio, amor e compreensão.

AGRADECIMENTOS

Agradeço à UNIVEM por tudo o que essa graduação proporcionou de aprendizado e novas experiências.

À minha família pelo apoio em toda a minha vida e principalmente durante o desenvolvimento do curso.

Um agradecimento especial à minha primeira mãe que deu muito duro para que eu pudesse estar onde estou hoje e à minha segunda mãe que participou de todos os momentos da minha vida, sempre me apoiando.

À Danielle Machado por tudo o que ela tem feito por mim como pessoa e por todo o amor doado a mim. E também pela compreensão com a coisas que eu deixei de fazer em casa para poder terminar este trabalho.

Aos meus amigos mais próximos, pela cobrança, às vezes implicante, mas necessária; pela horas de descontração e por todo o resto que eles representam na minha vida.

Ao meu orientador Prof. Ms. Ricardo José Sabatine que dedicou muito do seu tempo para me auxiliar e me direcionar quanto ao desenvolvimento deste trabalho que faz parte da conclusão de uma etapa na minha carreira. Sempre me lembrarei disso.

"Be like water making its way through cracks. Do not be assertive, but adjust to the object, and you shall find a way around or through it. If nothing within you stays rigid, outward things will disclose themselves.

Empty your mind, be formless. Shapeless, like water. If you put water into a cup, it becomes the cup. You put water into a bottle and it becomes the bottle. You put it in a teapot, it becomes the teapot. Now, water can flow or it can crash. Be water, my friend."

Bruce Lee

GOMES, Alexandre José Dutra. **Estratégias De Implementação De Cache Para Otimização De Serviços Web**. 2013. 74f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2013.

RESUMO

Tecnologias de cache são utilizadas para reduzir o tempo de resposta, mantendo cópias das respostas (páginas HTML, imagens, e outros arquivos), para que posteriormente, caso seja feita uma nova requisição a resposta seja muito mais rápida que a antecessora. Elas também são utilizadas para reduzir a carga de servidores, fazendo com que o cache responda a boa parte das requisições realizadas e que o servidor responda somente o necessário. Propõe-se neste projeto criar um cenário de testes utilizando diferentes estratégias de cache para avaliar o desempenho das estratégias relacionadas com serviços web e servir como base para o desenvolvimento de aplicações web com suporte a cache.

Palavras-chave: Cache; Web Cache; Serviços Web; HTML.

GOMES, Alexandre José Dutra. **Estratégias De Implementação De Cache Para Otimização De Serviços Web**. 2013. 74f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2013.

ABSTRACT

Web cache technologies are used to reduce the response time, keeping copies of the responses as HTML pages, images and others files, to be used later, in case a new request is made, the delivery will be more quickly than its predecessor. They are also used to reduce the server load, forcing the cache to respond to the major part of the requests made and the server only needs to respond when extremely necessary. In this project, it has been proposed to create a test scenario utilizing different cache strategies to evaluate the performance of the strategies related to web services and to be used as a groundwork to the development of web applications that supports web cache.

Keywords: Cache; Web Cache; Web Services; HTML.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE SIGLAS

CDN	*Content Delivery Network*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
JMETER	Java *Meter*
PHP	*Personal Home Page*
TCP/IP	*Transmission Control Protocol over IP*
TTL	*Time-to-Live*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Location*
VCL	*Varnish Configuration Language*

SUMÁRIO

INTRODUÇÃO

O grande fluxo crescente de tráfego de dados na Internet torna o trabalho sensível ao crescimento, dimensionamento a escalabilidade de sistemas web (WANG, 1999). Planejando corretamente um sistema Web, os caches podem ajudar no tempo de carregamento e na carga utilizada no servidor. A diferença pode ser dramática: uma página que levaria dez segundos para ser carregada, com o uso de cache pode ser carregada quase que instantaneamente.

Já no lado do servidor, a carga dele pode ser reduzida em até 70% (PINHEIRO, 2001), gerando estabilidade para o sistema e também economia, tanto com infraestrutura, quanto em outros aspectos como energia elétrica e mão de obra.

Os serviços web apresentam diversas características, sendo uma das principais, o uso de estratégias de cache. Elas são extremamente importantes e estão diretamente relacionadas com o desempenho do sistema tanto em termos de consumo de banda como em termos de redução de atrasos.

Algumas estratégias de cache utilizadas são: cache local no navegador, controle de cache do HTTP, servidor de proxy reverso (AKAMAI, 2013) (VARNISH, 2013).

Objetivo

O objetivo deste trabalho foi criar um cenário de testes utilizando diferentes estratégias de cache para avaliar o desempenho das mesmas relacionadas com serviços web e diminuir a curva de aprendizado quanto ao manuseio e configuração de sistemas de cache, contribuindo com a melhoria no desenvolvimento da arquitetura dos sistemas web atuais e futuros. Para isso, foi implementado e configurado um servidor de proxy reverso. Neste trabalho foi destacado o uso da tecnologia Varnish.

CAPÍTULO 1 - CACHE

Segundo Colouris, Dollimore e Kindberg(2007) a definição de *cache* é realizar um armazenamento dos dados utilizados recentemente em um local mais próximo do que a origem real do objeto. Se um novo dado é recebido, ele é adicionado ao cache, caso já exista esse objeto, ele é substituído caso haja necessidade.

> "[...]Quando um processo cliente requisita um objeto, o serviço de cache primeiro verifica se possui armazenado uma cópia atualizada desse projeto, caso esteja disponível, o mesmo é entregue ao processo cliente. Se o objeto não estiver armazenado, ou se a cópia está atualizada, o mesmo é acessado diretamente em sua origem.
> As caches podem ser mantidas nos próprios clientes, ou localizadas em um servidor *proxy* que possa ser compartilhado por eles. [..]" (COLOURIS, DOLLIMORE E KINDBERG; 2007)

O uso de memórias cache se tornou praticável devido a dois princípios de acesso à memória: localidade temporal e localidade espacial (TANENBAUM, 2010).

A localidade temporal indica que se um processador acessa uma determinada posição de memória, é muito provável que ele a acessará novamente em um curto período de tempo. A espacial indica que caso um processador acesse uma determinada posição de memória, é muito provável que ele também acessará posições adjacentes em um curto espaço de tempo.

Toda vez que o processador solicitar um dado na memória, este poderá encontrá-lo na memória cache (cache *hit*) ou terá de buscá-lo diretamente na memória principal (cache *miss*).

Para que o conceito de cache seja entendido, é necessário ter conhecimento também sobre a *World Wide Web*. Também conhecida como WWW (Berners-Lee, 1991) é um sistema em evolução para a publicação e para o acesso a recursos e serviços pela Internet.

Por meio de navegadores web, os usuários recuperam e veem documentos de muitos tipos, escutam faixas de áudio, assistem a vídeos e interagem com um vasto conjunto de serviços.

A *World Wide Web* nasceu no centro europeu de pesquisa nuclear (CERN), na Suíça, em 1989, como um veículo para troca de documentos entre uma comunidade de físicos conectados pela Internet. (Berners-Lee, 2000)

"A web é um sistema aberto: ela pode ser ampliada e implementada de novas maneiras, sem perturbar a funcionalidade existente.". (COLOURIS, DOLLIMORE e KINDBERG; 2007)

1.1. URL (Uniform Resource Location)

URLs (Uniform Resource Location) são utilizadas para localizar recursos, fornecendo uma identificação abstrata da localização do recurso. Após o recurso ser localizado é possível executar diversas operações caracterizadas pelas palavras *access*, *update*, *replace*, *find attributes*.

1.1. HTTP (Hypertext Transfer Protocol)

"O Hypertext Transfer Protocol (HTTP) é um protocolo em nível de aplicação para sistemas colaborativos, distribuídos, de informação hipermídia (IETF, 1999).

É um protocolo genérico que pode ser usado para muitas tarefas além do seu uso para hipertexto, como servidores de nome e sistemas de gerenciamento de objetos distribuídos, através de extensões dos seus métodos de requisição, códigos de erro e cabeçalhos. Uma funcionalidade do HTTP é a tipagem e negociação da representação de dados, permitindo aos sistemas serem construídos independentemente dos dados trafegados." (IETF, 1999, p.1, tradução nossa).

O HTTP é um protocolo desenvolvido dentro do framework do conjunto de protocolos de internet, *Internet Protocol Suite*. Foi projetado para permitir elementos de rede intermediários para habilitar ou melhorar a comunicação entre clientes e servidores (IETF, 1999).

Em uso desde 1990, a primeira versão, HTTP/0.9, era um protocolo simples para transferência de dados brutos através da internet. O HTTP/1.0, como definido pelo RFC (Request for Comments) 1945, melhorou-o permitindo que as mensagens fiquem no formato MIME, que contém metainformações sobre os dados transferidos e modificados através da semântica das mesmas IETF, 1999).

Como essa versão do protocolo não levava em consideração os efeitos de proxies hierárquicos, cache, necessidades por persistência de conexão e hosts virtuais, foi criada em 1999 a versão HTTP/1.1, que inclui requisitos mais rigorosos que sua antecessora, a fim de garantir um implementação mais confiável de suas características (IETF, 1999).

O protocolo em questão funciona através de requisição-resposta, em um modelo de computação cliente-servidor. Um browser é um exemplo de cliente, que envia uma requisição HTTP ao servidor, este, responde essa requisição ao cliente fornecendo um determinado recurso

como arquivos HTML ou outros conteúdos, ou apenas executa uma tarefa e retorna uma mensagem de resposta ao cliente (IETF, 1999).

Outros tipos de clientes incluem softwares de indexação, aplicações móveis e outras aplicações que acessam, consumem ou exibem conteúdo web.

Métodos

Os métodos definem a ação a ser executada no recurso identificado pela URL. Os métodos são sensíveis a letras maiúsculas e minúsculas.

- OPTIONS: Retorna os métodos HTTP suportados pelo servidor para uma determinada URL.
- GET: Requisita uma representação de um recurso específico. Requisições utilizando o GET devem retornar somente dados e não devem ter nenhum outro efeito.
- HEAD: Solicita uma resposta idêntica ao método GET, porém sem o corpo da resposta. Este método é útil para obter metainformações contidas nos cabeçalhos das respostas sem que todo o conteúdo necessite ser transferido.
- POST: Envia informações para serem processadas pelo servidor. Essas informações são transferidas no corpo da requisição. No cabeçalho são necessários campos adicionais especificando o tamanho e o formato das informações.
- PUT: Envia dados para serem guardados. Caso o URI refira-se a um recurso já existente, ele é modificado. Caso o recurso não exista, ele é criado.
- DELETE: Apaga o determinado recurso.
- TRACE: Exibe para o cliente a requisição realizada, para que o cliente possa visualizar possíveis alterações ou adições feitas por servidores intermediários.
- CONNECT: Converte a conexão da requisição em um túnel TCP/IP transparente, utilizado frequentemente em comunicações encriptadas por SSL (HTTPS).

Códigos de Status

Os códigos de status são inteiros de 3 dígitos que indicam o resultado de uma requisição.

- **1XX:** Informacional. Utilizada para enviar informação para o cliente indicando que sua requisição foi recebida e está sendo processada.
- **2XX:** Sucesso. Indica que a requisição do cliente foi recebida com sucesso, compreendida e aceita.

- **3XX:** Redirecionamento. Informa que ações adicionais necessitam ser tomadas pelo cliente para que a requisição seja completa. A ação necessária deve ser realizada pelo cliente sem interação com o usuário somente se o método utilizado for GET ou HEAD. Um redirecionamento infinito deve ser detectado pelo cliente, pois um laço desses gera tráfego de rede indesejável nas duas direções.
- **4XX:** Erro no cliente. A requisição possui uma sintaxe incorreta ou não pode ser completada. Indica um erro no lado do cliente. Exceto quando respondendo a um método HEAD, o servidor deve incluir uma resposta explicando a causa da situação do erro.
- **5XX:** Erro no servidor. O servidor falhou em concluir uma requisição aparentemente válida. Respostas iniciadas com o digito 5, indicam casos em o servidor cometeu um erro ou não conseguiu executar a requisição. Excluindo quando o servidor responde a uma requisição HEAD, ele deve incluir uma resposta contendo a explicação da situação do erro, e se é uma condição temporária ou permanente.

1.2. Cache na Web

Assim como a Internet continua a crescer em popularidade e tamanho, a demanda por infraestrutura e escalabilidade também cresce. O crescimento exponencial sem escalabilidade resultará em um tráfego de rede gigante e um tempo de resposta dos serviços web inaceitável.

É conhecido que a principal razão desse crescimento é a popularidade da *World Wide Web*, especificamente a alta porcentagem de tráfego em HTTP.

> "[...] A popularização da Web gerou sobrecarga na Internet e nos servidores, acarretando maiores tempos de resposta às requisições. Quando são feitos pedidos para servidores em links lentos, existe geralmente uma demora considerável na recuperação de objetos remotos. Além disso, a alta taxa de transferência de objetos pela rede leva a um aumento de tráfego que acaba reduzindo a largura de banda disponível e, também, introduzindo atrasos perceptíveis pelo usuário. [...]" (PINHEIRO, 2001)

O aumento de bytes transferidos entre hosts na internet juntamente com a dominância do protocolo HTTP sugere que muito do tráfego pode ser entregue através de cache. Ele se torna uma solução atrativa, pois conforme Wang (1999) e Murta e Almeida (1999) destacaram, o seu uso apresenta alguns benefícios como: economia de largura de banda da rede, menor latência, redução de tráfego, redução na carga de servidores e maior disponibilidade do sistema.

Implementar caches próximos aos clientes pode reduzir o tráfego de *backbone* consideravelmente, pois quando um conteúdo cacheado é usado, o acesso ao servidor de origem é desnecessário.

Para melhoria de disponibilidade de um servidor, é possível implementar um sistema de cache reverso, também chamado de servidor de proxy reverso, onde o cache pode ser, ou não, administrado por provedores de conteúdo, que são preparados para receber um alto volume de tráfego, assim, o acesso é recebido por esse servidor passando para o servidor de origem somente o extremamente necessário, melhorando a escalabilidade ou antecipando uma grande demanda.

Nesses casos, o cache não só aumenta a disponibilidade como também pode atuar como balanceador de carga.

1.2.1. Vantagens

Ao implementar um sistema de cache, além dos objetivos citados acima, é necessário observar e buscar algumas características, que segundo Wang (1999), são necessárias para o bom funcionamento do sistema. Elas são:

- Rapidez de acesso: um sistema de cache deve focar em reduzir a latência de acesso. O acesso realizado através dele deve ter em média uma latência menor do que acessando o mesmo sistema sem cache. Essa característica é de grande importância visto que para o usuário, o tempo para acessar uma página é um dos principais indicadores de qualidade de um site;
- Robustez: Para um usuário, robustez é sinônimo de disponibilidade. Um sistema deve estar disponível sempre que for requisitado;
- Transparência: Deve ser transparente aos olhos do usuário. Rapidez nas respostas e alta disponibilidade do sistema são as únicas características que devem ser notadas pelo usuário;
- Escalabilidade: Devido ao crescimento acelerado da internet é exigido que um o esquema de cache seja escalável para que ele possa acompanhar o crescimento do sistema sem diminuir sua funcionalidade;
- *Eficiência: O sistema de cache deve aumentar minimamente a carga de tráfego na rede, mantendo seu custo-benefício;

- Adaptabilidade: Engloba vários valores como, por exemplo, roteamento e administração de cache e localização de servidor de Proxy. Ao implementar um sistema de cache é preciso fazê-lo adaptar-se às mudanças de demanda do usuário;
- Estabilidade: A distribuição e organização do cache não deve gerar instabilidade no sistema;
- Balanceamento de carga: É esperado que em um esquema de cache o balanceamento de carga seja realizado. Caso exista somente um servidor de Proxy, por exemplo, será gerado nele um gargalo que compromete a performance de todo o sistema;
- Suporte à heterogeneidade: Conforme as redes crescem, elas começam a dispor de uma grande variedade de hardware e software de diversos tipos. Com isso é necessário que o sistema de cache tenha suporte e consiga trabalhar corretamente com essa diversidade arquiteturas de rede;
- Simplicidade: Um sistema de cache ideal é simples de implantar. Sistemas mais simples são mais fáceis de entender e identificar possíveis falhas, tornando-o mais vantajoso.

1.2.2. Desvantagens

- A principal desvantagem é a dificuldade na manutenção da consistência dos dados, que por qualquer configuração ou alteração inadequada pode ocasionar a exibição de conteúdo desatualizado.
- O possível crescimento do tempo de resposta em caso de cache miss, quando o conteúdo não é encontrado no cache. Devido a isso, a taxa de acerto do cache deve ser maximizada e o custo de um cache miss deve ser minimizado quando uma arquitetura de cache é desenvolvida.
- Um cache de proxy é sempre um gargalo. Dessa forma, deve-se levar em consideração limitar o número de clientes que um proxy pode servir, para que ele seja no mínimo tão eficiente quanto a comunicação direta com os servidores raiz onde o conteúdo se encontra. Outro problema em ser um gargalo, é se tornar um único ponto de falha. Caso algo aconteça com ele todos os servidores raiz abaixo dele podem ficar inacessíveis.
- Ao utilizar um servidor de cache o acesso aos servidores de origem será reduzido drasticamente, o que ocasionalmente pode ser prejudicial para a captura de informações sobre a utilização do sistema.

1.2.3. Tipos de Implementação de cache

O conteúdo da Web pode ser armazenado em vários locais diferentes entre o cliente e o servidor de origem. Muitos browsers possuem um cache embutido, os chamados browser caches. Seguindo a cadeia requisição-resposta, pode-se encontrar os proxy caches, que armazenam os objetos de acordo com as requisições de um determinado grupo de clientes.

Um tipo especial destes proxies não exige configurações por parte do cliente, pois interceptam requisições HTTP, sendo chamados proxies de interceptação (ou proxy cache transparente). No outro extremo da cadeia existem os proxies reversos (ou *surrogates*), responsáveis por armazenar as respostas mais comuns dos servidores. A seguir serão detalhados os tipos de *Web caching*.

Cache de Navegador

A maioria dos browsers conhecidos possui um cache embutido. Através deste cache muitos arquivos podem ser reutilizados, quando se visita novamente um mesmo site ou quando páginas Web utilizam os mesmos logos, figuras, banners. Este tipo de *caching* é realizado, pois é comum o acesso a uma mesma página múltiplas vezes em um curto espaço de tempo (por exemplo o uso do botão Voltar do browser) (PINHEIRO, 1999).

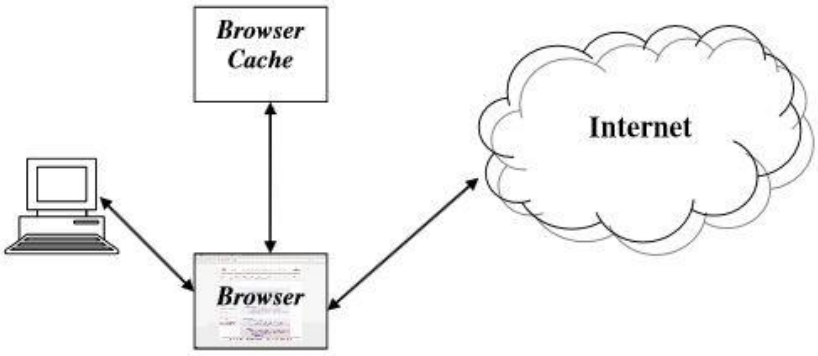
Geralmente os browsers permitem que os usuários definam os parâmetros, tais como quanto espaço se deseja reservar para o cache e frequência com que as informações do cache devem ser invalidadas. Apesar muito úteis, esses caches apresentam alguns problemas.

Os dados armazenados são correspondentes às requisições de apenas um usuário. Isto quer dizer que só haverá um hit no caso de uma página ser revisitada.

Outro problema é a incompatibilidade de caches de diferentes browsers. Este último problema já é alvo de pesquisas. Existem algumas soluções comerciais que são compatíveis com um grande número de browsers.

A Figura 1 ilustra um exemplo da estrutura desse tipo de cache.

Figura 1 Esquema de cache de navegador



Fonte: Própria

Cache de Proxy

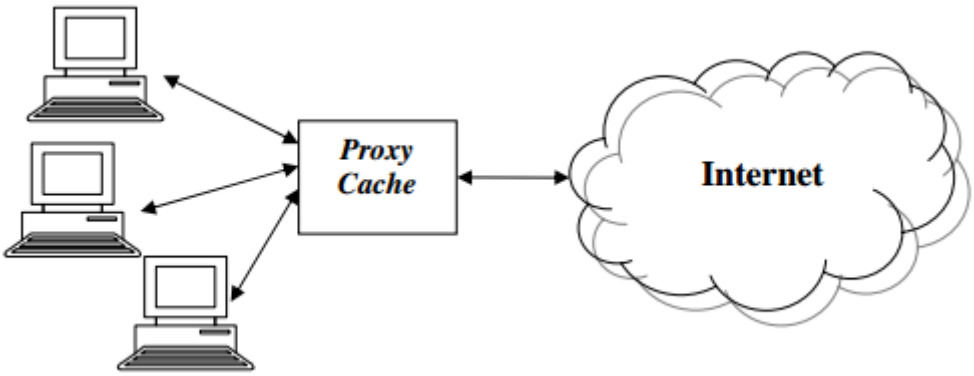
Este tipo de cache pode servir a vários usuários de uma só vez. Uma vez acessado e utilizado por muitos clientes, os acessos ocorrerão em maior número e de forma mais diversa.

Sendo assim, o proxy será mais diversificado, aumentando a sua taxa de acerto. Este tipo de proxy normalmente apresenta uma taxa de acerto mais alta que os browser caches. Ao receber uma requisição, o proxy cache procura pelo objeto localmente.

Se a encontrar (hit), este é prontamente repassado ao usuário. Caso contrário (miss), o proxy faz uma requisição ao servidor, grava a página no disco e a repassa ao usuário. Requisições subsequentes (de qualquer usuário) recuperam a página que está gravada localmente.

Os servidores do tipo proxy cache são utilizados por organizações ou provedores que querem reduzir a quantidade de banda do sistema de comunicação que utiliza.

Figura 2 Esquema de cache de proxy



Fonte: Própria

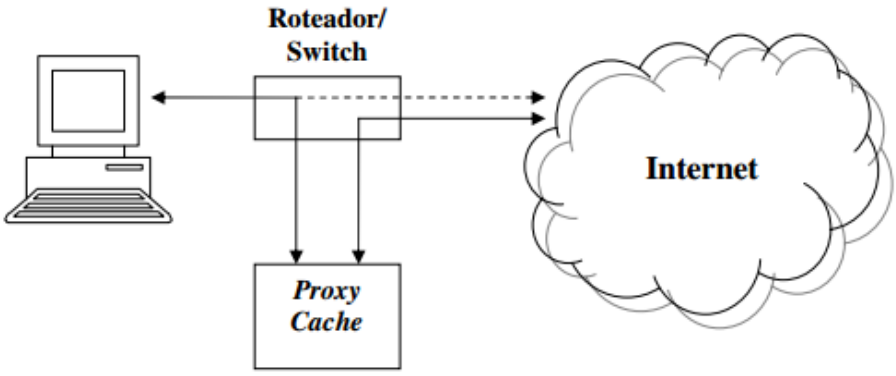
Cache de Proxy de Interceptação

Segundo Wessels (2001) um das maiores dificuldades de operação de um proxy cache é conseguir usuários para usar o serviço. Isto se deve à dificuldade de se configurar os browsers. Usuários podem pensar não estarem configurando seus browsers corretamente e, por isso, acabarem por desabilitar o uso do mesmo.

Outro problema com relação aos usuários é a possível resistência ao uso de proxy devido ao medo de receber informações desatualizadas.

Tendo em vista estes problemas, muitas organizações passaram a usar proxies de intercepção, uma vez que eles diminuem a sobrecarga dos administradores e aumentam o número de clientes utilizando o proxy. A ideia principal deste tipo de proxy é trazer o tráfego para o cache, sem a necessidade de configuração dos clientes. Isto é feito através do reconhecimento das requisições HTTP pelos roteadores, e redirecionamento das mesmas ao proxy (PINHEIRO, 1999).

Figura 3 Esquema de cache de interceptação



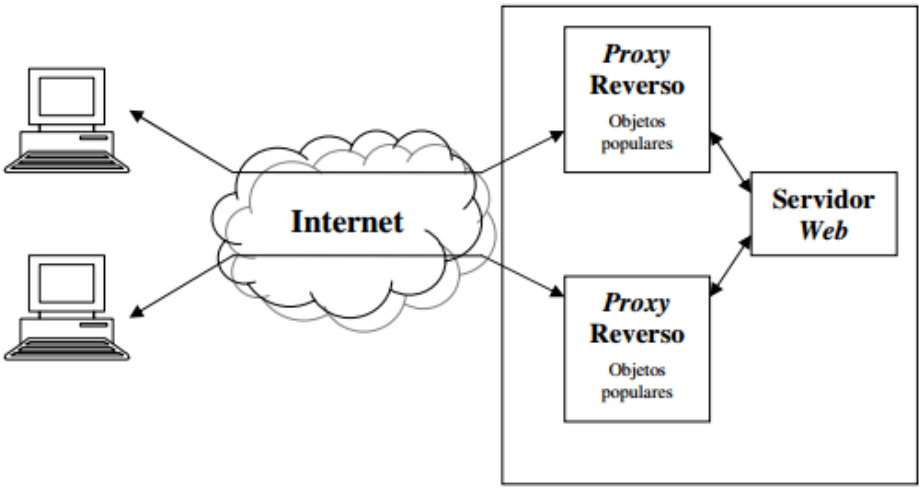
Fonte: Própria

Cache de Proxy Reverso

Surgiu da necessidade de aproximar os proxies dos servidores para reduzir a carga sobre eles. Recebem este nome, pois estão na ponta contrária ao tradicional na cadeia requisição-resposta. Estes proxies são também chamados de aceleradores, pois o sistema de armazenamento em cache fica à frente de um ou mais servidores Web, interceptando solicitações e agindo como um proxy (PINHEIRO, 1999).

Os documentos armazenados em cache são fornecidos a uma maior velocidade, enquanto os que não estiverem em cache (conteúdo Roteador/ Switch Internet Proxy Cache dinâmico/personalizado) são solicitados aos servidores Web de origem quando necessário.

Figura 4 Esquema de cache de proxy reverso



Fonte: Própria

1.2.4. Principais Problemas do Cache na Web

Manter a consistência de um Web cache é um problema muito complexo. A princípio, pode parecer simples manter um cache atualizado, bastaria perguntar ao servidor sobre a validade dos seus dados. Isto acarretaria muitas requisições ao servidor, voltando a ocasionar problema de latência, aumentando a carga no servidor e a utilização de banda.

No caso de aumentar-se o intervalo de tempo entre verificações da validade dos dados, pode-se estar criando o problema de acesso a informações velhas.

A latência pode aumentar no caso da ocorrência de muitos *misses*, uma vez que existe o tempo de busca e o tempo de armazenamento do objeto no cache.

Cache pode trazer complicações aos servidores, pois causam distorções nos arquivos de log. Dados como número de *pageviews*, locais de acesso, frequência com que certos usuários acessam a página ficam distorcidos e passam a não ser analisados corretamente.

Um único proxy é sempre um gargalo. Deve existir um limite de clientes que um proxy pode servir. Este limite deve manter o proxy no mínimo tão eficiente quanto se estivesse sendo usada uma conexão sem proxy.

A "cacheabilidade" dos elementos deve ser considerada sempre. O número de páginas personalizadas e geradas dinamicamente tem crescido e isso pode trazer graves problemas. Uma página personalizada para uma pessoa X pode ser posta em cache, e ser mostrada para uma pessoa Y, quando da requisição da mesma. Existem ainda as páginas que dependem de preenchimento de formulário. Estas normalmente são acessadas uma única vez, e apenas ocuparão espaço no cache.

Têm-se ainda problemas com privacidade, armazenamento de conteúdo ofensivo, integridade das informações contidas no proxy, veracidade das informações, direitos autorais e propagandas indesejadas.

1.2.5. Consistência de Cache

Mecanismos de consistência tentam garantir que as cópias em cache estão atualizadas, refletindo o que acontece no servidor. Existem vários mecanismos de consistência atualmente, nesta seção serão apresentados alguns.

Client-validation: Esta técnica consiste em constantes verificações de inconsistência dos objetos por parte do cliente (cache). O cache admite que todos os objetos armazenados estão

potencialmente desatualizados. O cliente então faz solicitações periódicas ao servidor utilizando GET condicional (*if-modified-since*). Este mecanismo pode levar a muitas respostas "*Not modified*" por parte do servidor, se o documento não foi alterado. Esta abordagem apresenta a desvantagem de gerar muitas requisições, aumentando o tráfego da rede e a carga do servidor.

TTL (Time-to-live): trata-se da estimativa do tempo de vida de um objeto, determinando o tempo que um objeto pode permanecer em cache sem estar desatualizado. Se um objeto requisitado ainda está válido no cache, este último o fornece sem contatar o servidor. Caso o objeto esteja inválido (TTL expirado), faz-se uma requisição condicional ao servidor (*if-modified-since*) para verificar se o objeto foi alterado. O servidor retorna um código de status informando se o objeto foi ou não modificado e, no primeiro caso, retorna o novo objeto. TTLs são de simples implementação no protocolo HTTP, bastando usar o cabeçalho *expires*.

Alex Protocol (TTL Adaptativa): A TTL Adaptativa (CATE, 1992) tenta resolver o problema da geração de muitas requisições ao servidor ajustando os TTLs dos documentos, baseado em observações do tempo de vida. Ele assume, basicamente, que arquivos mais novos mudam mais frequentemente que arquivos antigos.

Se um arquivo está há muito tempo em cache sem modificações, ele tende a ser estático. O algoritmo utiliza um protocolo que se baseia em um limite de atualização. Este limite é expresso como uma porcentagem da idade atual do documento (data atual – data da última modificação). Um objeto se torna inválido quando sua idade exceder o limite de atualização.

A maioria dos servidores de proxy atuais utilizam este mecanismo. Porém, existem alguns problemas relacionados aos mecanismos baseados em tempo de vida. Primeiro, o cache pode retornar dados inválidos se o objeto tiver sido alterado dentro do tempo que a cópia é considerada válida. Segundo, o mecanismo não dá garantias sobre a validade do documento. Por último, os usuários não têm como ajustar o nível de "velhice" tolerável.

Server-Invalidation: O protocolo de invalidação pelo servidor é capaz de garantir uma consistência mais precisa, pois o servidor é responsável por enviar mensagens aos clientes (caches) sempre que houver alguma modificação em seus objetos. Este mecanismo, porém, possui alto custo, uma vez que o servidor deve possuir uma lista dos clientes que armazenam cópias de seus objetos para invalidá-las.

Outro ponto negativo para este mecanismo é a possibilidade da própria lista se tornar desatualizada, fazendo com que o servidor envie mensagens a clientes que não mais armazenam

seus objetos. Existe ainda a possibilidade de haverem clientes indisponíveis, sendo necessário que o servidor continue tentando notificá-los, uma vez que os caches só poderão invalidar os objetos se forem notificados pelo servidor.

1.3. Considerações Finais

A revisão bibliográfica realizada neste capítulo indicou várias formas de web cache disponíveis atualmente.

Dentre todas as tecnologias de cache pesquisadas foram utilizadas as de cache de navegador, cache de proxy reverso e cache de interceptação.

A escolha delas foi devido às características destacadas nos tópicos anteriores se ajustarem às necessidades dos serviços web desse trabalho.

CAPÍTULO 2 - TECNOLOGIAS PARA UTILIZAÇÃO DE CACHE

Neste capítulo serão abordadas algumas das tecnologias que podem ser utilizadas para a adoção de sistemas de cache na web e como cada uma colabora para o desenvolvimento deste projeto.

2.1. Controle de Cache do HTTP

No protocolo HTTP 1.1 (IETF, 1991), foram introduzidas diretrizes especiais de controle de cache para administrar o comportamento dos caches de navegador e de proxy. Essas diretivas foram uma melhoria nos cabeçalhos já utilizados no protocolo HTTP 1.0.

Os cabeçalhos de controle de cache permitem cliente e servidor transmitir uma infinidade de instruções tanto nas requisições quanto nas respostas. Essas instruções normalmente sobrescrevem os algoritmos de cache padrão. Caso exista qualquer conflito entre os valores dos cabeçalhos, a instrução mais restritiva é aplicada (IETF, 1991).

As instruções indicadas nos cabeçalhos de controle de cache, por padrão, devem ser obedecidas por todos os mecanismos de cache ao longo da cadeia requisição/resposta, momento desde a requisição do conteúdo até a entrega da resposta para o solicitante (IETF, 1991).

Essas instruções são unidirecionais, ou seja, uma instrução estar presente em uma requisição não significa que ela será utilizada ou estará presente na resposta. Elas devem sempre ser passadas adiante por proxies ou gateways, independentemente de sua significância para a aplicação.

Algumas instruções são:

- Instruções de requisição:
    - no-cache;
    - no-store;
    - max-age;
    - max-stale;
    - min-fresh;
    - no-transform;
    - only-if-cached;
    - cache-extension;
- Instruções de resposta:
    - public;

- private;
- no-cache;
- no-store;
- no-transform;
- must-revalidate;
- proxy-revalidate;
- max-age;
- s-maxage;
- cache-extension;

As instruções de controle de cache podem ser divididas nas seguintes categorias:

- Restrições sobre o que pode ser cacheado. Determinadas somente pelo servidor de origem.
- Restrições sobre o que pode ser armazenado pelo cache. Determinadas pelo servidor de origem ou pelo usuário.
- Modificações nos mecanismos básicos de vencimento. Determinadas pelo servidor de origem ou pelo usuário.
- Controles sobre revalidação ou recarregamento de cache. Determinadas somente pelo usuário.
- Controles sobre a transformação de entidades.
- Extensões para o sistema de cache.

O que pode ser cacheado

Por padrão, uma resposta pode ser cacheada se os métodos, cabeçalhos e status da resposta indicarem que pode. As instruções de controle de cache abaixo permitem a um servidor de origem sobrescrever o valor padrão desses indicadores:

- public: Indica que a resposta pode ser cacheada por qualquer cache, mesmo que normalmente ela não seja.
- private: Indica que toda ou parte da mensagem da resposta é direcionada para um único usuário e não deve ser cacheada por um cache compartilhado. Isso permite ao servidor de origem constatar que partes específicas da resposta não são válidas para requisições de outros usuários. Um cache privado, não compartilhado, pode cachear a resposta. É válido ressaltar que o uso deste recurso controla somente onde a resposta pode ser salva e não pode garantir a privacidade da mensagem.
- no-cache: Se esta instrução não especificar um nome de campo, então, o cache não deve usar a resposta para satisfazer uma resposta posterior sem uma revalidação no

servidor de origem. Isso permite ao servidor de origem prevenir o armazenamento de cache mesmo por sistemas de cache que forem configurados para retornar conteúdo antigos. Se um ou mais campos são informados, então o cache deve usar a resposta para satisfazer uma requisição posterior, entretanto, os campos especificados não devem ser enviados sem validação. Isso permite a um servidor de origem prevenir o reuso de certos campos de cabeçalho, mesmo permitindo o reuso do restante da resposta.

O que pode ser armazenado no cache

- no-store: O objetivo dessa instrução é prevenir a liberação ou retenção negligente de informação. Ela se aplica à mensagem toda, e pode ser enviada na requisição ou na resposta. Se enviada na requisição, o cache não deve guardar nenhuma parte nem da requisição e nem da resposta. Se enviada na resposta, ele não deve guardar nenhuma parte da resposta e nem da requisição que a gerou. Essa instrução se aplica tanto para caches compartilhados quanto para não compartilhados. Mesmo quando ela é aplicada a uma resposta, o usuário pode explicitamente guardar tal resposta fora do sistema de cache, por exemplo, salvando a página gerada. Buffers de histórico devem armazená-las normalmente seguindo sua operação padrão. Mesmo podendo aumentar a privacidade em alguns casos, é alertado que essa instrução não é um mecanismo seguro e suficiente para garanti-la, pois caches maliciosos ou comprometidos podem não reconhecer ou não aceitar a instrução e a rede em que os dados trafegam pode ser vulnerável a interceptação de informações.

Modificações nos mecanismos de vencimento

O tempo de validade de um objeto pode ser especificado pelo servidor de origem através da utilização do cabeçalho de vencimento, utilizando ainda os recursos do protocolo HTTP/1.0, ou ele pode ser definido usando a instrução *max-age* na resposta. Assim, quando essa instrução é encontrada em uma resposta cacheada, ela está vencida caso o tempo de vida dela, indicado pela tag *Age*, seja maior que o valor passado para o *max-age* no momento em que a requisição foi efetuada.

- **s-maxage:** Se uma resposta incluir essa instrução, então, somente para o cache compartilhado o tempo de vida máximo especificado por essa instrução sobrescreve o especificado na instrução max-age. Essa instrução é totalmente ignorada por caches privados.
- **max-age:** Essa instrução indica que um cliente está disposto a aceitar repostas com tempo de vida menor do que o tempo especificado nela em segundos. A menos que

a instrução *max-stale* também esteja presente, o cliente não aceitará uma resposta vencida.

- **min-fresh:** Indica que um cliente aceitará uma resposta caso o tempo de vida dela mais o valor informado na instrução não seja maior que o tempo de vida máximo dessa resposta. Ou seja, o cliente deseja uma resposta que ainda será válida por pelo menos o tempo informado na instrução.
- **max-stale:** É usada para indicar que o cliente aceitará uma resposta que tenha excedido o tempo de vida limite. Se um valor for atribuído, o tempo excedido não deve ultrapassá-lo. Caso nenhum valor seja informado, então, são aceitas quaisquer respostas vencidas independentemente do tempo de vida delas.

Revalidação de cache e controles de atualização

Há ocasiões em que é necessário forçar que um serviço de cache revalide ou renove seu conteúdo com o servidor de origem. Uma revalidação dessas pode ser necessária caso o servidor de cache ou o servidor de origem tenham superestimado o tempo de validade da resposta cacheada e o conteúdo em cache se tornou corrompido ou inválido por alguma razão.

Essa revalidação pode ser solicitada tanto quando o cliente possui quanto quando não possui uma cópia do conteúdo em cache. É chamada de **Revalidação *end-to-end* indeterminada** quando o cliente não possui a cópia e **Revalidação *end-to-end* determinada**.

O cliente pode utilizar os três tipos de ação abaixo utilizando as instruções de controle de cache:

- **Recarregamento end-to-end:** A requisição inclui a instrução de controle de cache *no-cache* ou a instrução *Pragm: no-cache*, para manter a compatibilidade com o protocolo HTTP/1.0. O servidor não deve utilizar uma cópia em cache para responder a uma requisição dessas.
- **Revalidação end-to-end determinada:** A requisição inclui a instrução de controle de cache *max-age=0*, que obriga todos os servidores de cache ao longo do caminho para o servidor de origem a revalidarem o seu conteúdo com o próximo cache disponível. A requisição inicial inclui dentro de si uma condição de validação de cache, o que indica em qual circunstância o cache revalidado será válido ou não.
- **Revalidação end-to-end indeterminada:** Funciona da mesma forma que a determinada, porém a requisição inicial não inclui a condição de validação de cache, fazendo com que o primeiro cache que possuir o recurso determina a condição da validação do cache.

2.2. CDN (Content Delivery Network)

Content Delivery Network (CDN) é uma coleção de elementos de rede abrangendo a Internet, onde conteúdo é replicado ao longo de muitos servidores web idênticos a fim de realizar uma entrega de conteúdo transparente e efetiva ao usuário final (BUYYA, PATHAN e VAKALI, 2008).

Através disso uma CDN distribui o conteúdo de um servidor de origem para servidores de replicação próximos aos clientes finais. Esses servidores guardam grupos muito seletos de objetos e somente requisições para aqueles grupos de conteúdo são servidos pela CDN fazendo com que a taxa de acerto possa se aproximar a 100%.

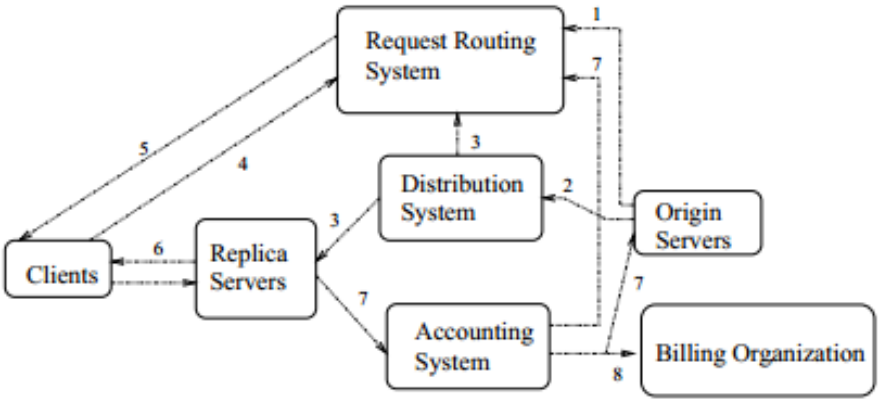
Isso faz com que ao utilizar uma CDN o tempo de resposta apresentado seja muito baixo e o consumo de banda seja menor. Ela também fornece outros benefícios como ajustar o posicionamento de conteúdo, o roteamento de requisições e a capacidade de provisionamento para responder à demanda e às condições da rede. Adicionalmente o fato de que muitos objetos não são cacheados, mas sim replicados e isso inclui objetos dinâmicos com acesso somente de leitura e objetos personalizados (requisições a cookies), tornam uma CDN muito útil (BUYYA, PATHAN e VAKALI, 2008).

A arquitetura de uma CDN é mostrada na Figura 5. Ela é composta por sete componentes: cliente, servidores réplica, servidor de origem, organização de faturamento, sistema de roteamento de requisição, sistema de distribuição e sistema de contas. As relações entre esses componentes são representadas pelas linhas numeradas na Figura 5 e descritas abaixo:

1. O servidor de origem delega o seu endereço para que os objetos sejam distribuídos e entregues pela CDN para o sistema de roteamento de requisição.
2. O servidor de origem publica o conteúdo para que ele seja distribuído e entregue pela CDN ao sistema de distribuição.
3. O sistema de distribuição move o conteúdo para os servidores de replicação. Adicionalmente, esse sistema interage com o sistema de roteamento de requisição através de informações que auxiliam na seleção de qual servidor responderá a solicitação dos clientes.
4. O cliente requisita documentos do que ele acredita ser o servidor de origem. No entanto, devido à delegação de endereço, a requisição é na verdade direcionada para o sistema de roteamento de requisição.

5. O sistema de roteamento de requisição encaminha a requisição para um servidor de réplica adequado na CDN.
6. O servidor réplica entrega o conteúdo requisitado ao cliente. Além disso este servidor envia informações de conta sobre o conteúdo entregue ao sistema de contas.
7. O sistema de contas agrega e refina as informações recebidas em estatísticas e registros detalhados de conteúdo para uso do servidor de origem e da organização de faturamento. Estatísticas também são usadas como feedback pelo sistema de roteamento de requisição.
8. A organização de faturamento utiliza os registros detalhados de conteúdo para tarifar cada uma das partes envolvidas na distribuição do conteúdo.

Figura 5 Componentes da arquitetura de uma CDN



Fonte: CDN: Content Distribution Network

2.2.1. Akamai

Akamai Technologies, Inc. (AKAMAI, 2013) é uma CDN com sede em Cambridge, Estados Unidos. Sua rede é uma das maiores plataformas de redes distribuídas do mundo, responsável por servir cerca de 15 a 20 por cento de todo o tráfego da web (Nygren, Sitamaran, Sun; 2010).

Ela foi fundada em 1998 por Daniel M. Lewin, graduado no MIT, juntamente com o professor de matemática aplicada também no MIT Tom Leighton. O princípio dela foi a criação de um algoritmo que acelerava o acesso à internet.

A empresa opera sua rede no mundo todo e aluga espaço em seus servidores para clientes que desejam que seus sites e recursos funcionem mais rápido por meio da distribuição

do conteúdo de uma localização mais próxima do usuário. Grandes clientes e exemplos dos seus serviços são Facebook, Bing e Twitter.

Quando um usuário acessa a URL de um cliente da Akamai, o navegador dele é redirecionado para uma cópia do site dentro da Akamai quase totalmente transparente para a maioria dos usuários.

2.3. Varnish

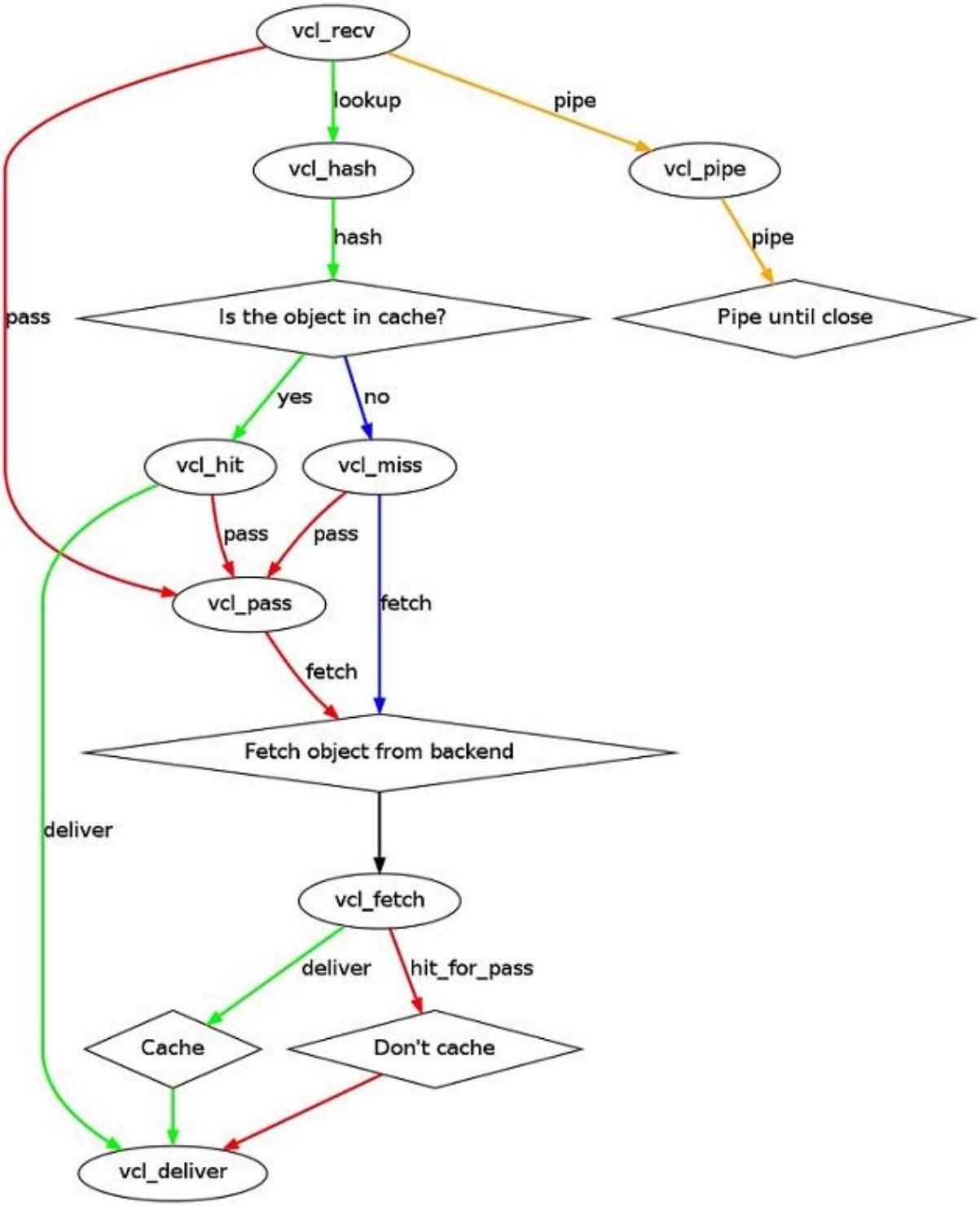
O Varnish (VARNISH, 2013) é um proxy reverso de HTTP também conhecido como acelerador de aplicações web. É um software livre licenciado sob duas cláusulas da licença *FreeBSD*.

O projeto foi iniciado em 2005. Sua primeira versão foi lançada em setembro de 2006 (*Varnish Cache* 1.0), e a última versão estável, *Varnish Cache* 3.0.4 foi lançada em junho de 2013 (VARNISH, 2013).

As principais características do *Varnish* são sua performance e flexibilidade. Através de sua própria linguagem de configuração, VCL (*Varnish Configuration Language*), ele é altamente configurável e pode-se criar políticas de como lidar com vários cenários de tráfego.

2.3.1. Fluxo de Funcionamento e suas Funções

Figura 6 Fluxo de funcionamento do Varnish Cache



Fonte: Varnish Book

vcl_recv: Função de tratamento de requisição. É chamado no início da requisição, após ela toda ser analisada. Seu propósito é decidir se uma requisição será entregue pelo varnish ou não, como fazê-lo e se aplicável, qual backend utilizar. Nessa função é possível alterar a requisição, sendo possível remover cookies e adicionar e remover cabeçalhos.

vcl_hash: Função de criação de hash. Esta função é utilizada para criar o hash do objeto cacheado.

vcl_pipe: Função de abertura do modo pipe. Neste modo a requisição é passada diretamente ao backend e quaisquer dados posteriores tanto do cliente como do backend são entregues sem alteração até que uma ponta encerre a conexão.

vcl_pass: Função de abertura do modo pass. Neste modo a requisição é passada diretamente ao backend e a resposta dele é passada ao cliente porém sem entrar no cache. Requisições subsequentes submetidas pelo mesmo cliente serão manuseadas normalmente.

vcl_hit: Função de acerto. Define o que será realizado quando o objeto é encontrado no cache.

vcl_miss: Função de erro. Define o que será realizado quando o objeto não é encontrado no cache.

vcl_fetch: Função de coleta no backend. Nesta função são definidas as ações a serem realizadas quando o objeto é coletado no backend.

vcl_deliver: Função de entrega de objeto. Essa função é chamada antes do objeto ser entregue.

Visando uma solução sem custos a ferramenta Varnish foi escolhida para ser utilizada neste trabalho. Devido à alta capacidade de personalização e capacidade de controle de cache HTTP da ferramenta é possível que ela seja utilizada e testada nas mais diversas arquiteturas.

CAPÍTULO 3 - CENÁRIO

Neste capítulo serão abordadas as informações referentes ao cenário inicial em que o ambiente utilizado nos testes se encontrava e o cenário modificado visando a otimização através da utilização de web cache.

3.1. Cenário Padrão

3.1.1. Tecnologias Utilizadas

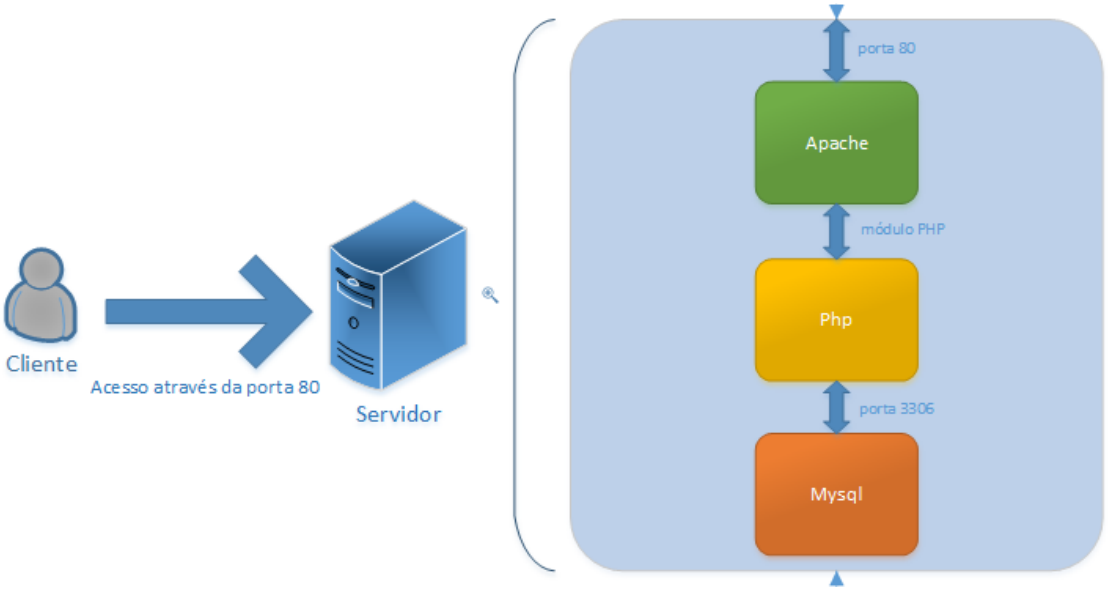
As tecnologias utilizadas para o desenvolvimento da aplicação são Apache, aplicação de servidor web; PHP, linguagem de programação; e MySQL, sistema de gerenciamento de banco de dados relacional.

3.1.2. Estrutura

A estrutura lógica da aplicação em questão é composta basicamente pelos três elementos citados no tópico anterior: Apache, PHP e MySQL.

O apache recebe a requisição da internet na porta 80 que é o padrão HTTP, o código é interpretado pelo módulo PHP e a página é exibida no navegador do cliente. Caso alguma consulta ao banco de dados seja necessária, o módulo PHP faz a conexão com o MySQL na porta 3306 e retorna os dados para a página.

Figura 7 Estrutura do cenário padrão



Fonte: Própria

3.2. Características

3.2.1. Aplicação em Relação ao Cache

A aplicação utilizada para a realização dos testes no servidor é um sistema de e-commerce.

Foi escolhido esse tipo de aplicação devido a sua necessidade de alta disponibilidade e o seu alto número de acesso.

3.2.2. Linguagem

A linguagem utilizada para este trabalho é o PHP (PHPNET, 2013). Ela é uma linguagem de script para aplicações que executam no lado do servidor, originalmente criada em 1995 por Rasmus Lerdorf e atualmente mantida pelo *The PHP Group*. No início, a sigla PHP significava *Personal Home Page* (Página Principal Pessoal), porém agora é o acrônimo recursivo de *Hypertext Preprocessor* (Pré-processador de Hipertexto). O código PHP é interpretado por um servidor web com um modulo processador de PHP, gerando uma página web. Os comandos dessa linguagem podem ser inseridos diretamente no código HTML.

Sua sintaxe é derivada das linguagens C, Java e Perl, tornando-a fácil de aprender. “O objetivo principal da linguagem é permitir a desenvolvedores escreverem

páginas que serão geradas dinamicamente rapidamente, mas você pode fazer muito mais do que isso com PHP.” (PHPNET, 2013).

3.2.3. Banco de Dados

O sistema de banco de dados utilizado para nessa aplicação é o MySQL (MYSQL, 2013). Ele é um sistema gerenciador de banco de dados relacional open-source e foi criado em 1995 por uma empresa sueca chamada MySQL AB que hoje pertence a Oracle.

O MySQL é uma escolha popular para o uso em aplicações web e para projetos de software grátis que necessitam de um sistema de gerenciamento de banco de dados completo.

A sua compatibilidade com o PHP e a facilidade no uso foram alguns dos principais fatores para a escolha deste banco.

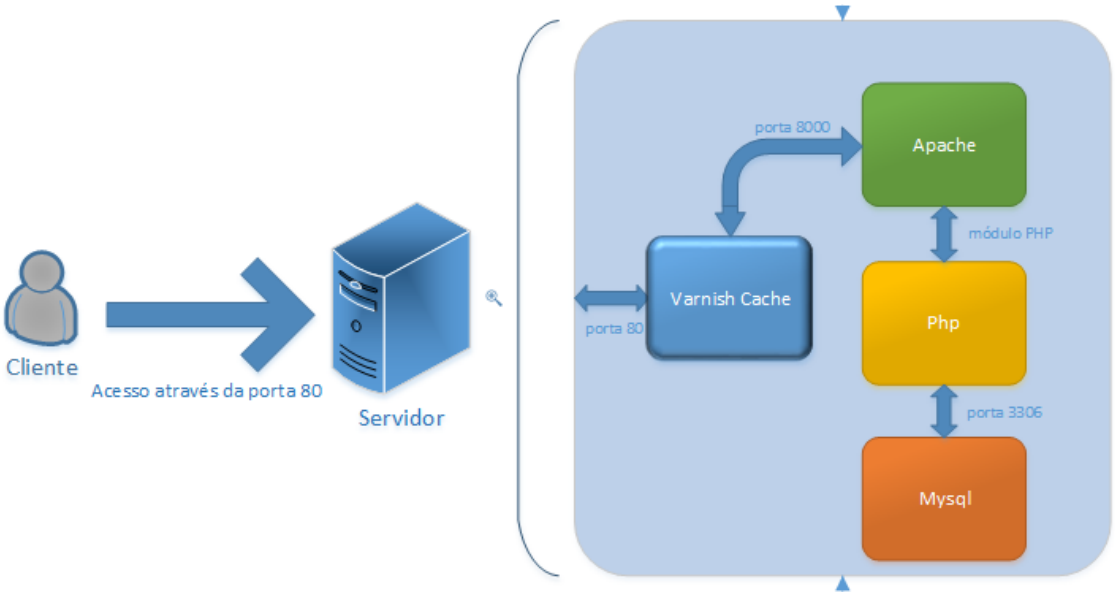
3.3. Cenário com Solução Desenvolvida

3.3.1. Tecnologias Utilizadas

As tecnologias utilizadas para o desenvolvimento da solução são as mesmas do cenário padrão com o acréscimo do Varnish Cache.

3.3.2. Estrutura

Figura 8 Estrutura do cenário com solução desenvolvida



Fonte: Própria

3.4. Instalação do Varnish Cache

3.4.1. Instalação Através do Pacote de Instalação

Esse procedimento de instalação é simples por ser realizado através de um pacote fornecido no repositório varnish-cache.org (VARNISH, 2013).

Os comandos abaixo são utilizados para a instalação em Red Hat/CentOS.

Instalar o repositório na máquina:

Executar:

```
rpm --nosignature -i http://repo.varnish-cache.org/redhat/varnish-3.0/el6/noarch/varnish-release/varnish-release-3.0-1.el6.noarch.rpm
```

Instalar o Varnish Cache:

Executar:

```
yum install varnish
```

Habilitar o início automático da aplicação quando a máquina ligar:

```
chkconfig varnish on
```

3.4.2. Instalação pelo Código Fonte

Caso não exista um pacote de instalação para o sistema operacional que esteja sendo utilizado, ou a compilação do código fonte do Varnish Cache seja necessária, a instalação se dará da maneira como está descrita abaixo.

3.4.3. Requisitos para Instalação a partir do código fonte

- Compilador C;
- Pacote automake;
- Pacote autoconf;
- Pacote libtool;
- Pacote ncurses-devel;
- Pacote groff;
- Pacote pcre-devel;
- Pacote pkgconfig;

Download do código fonte

Faça o download através do seguinte comando:

```
wget http://repo.varnish-cache.org/source/varnish-3.0.4.tar.gz
```

Extração dos arquivos

Extraia os arquivos utilizando o comando:

```
tar xzf varnish-3.0.4.tar.gz
```

Configuração e compilação

Acesse o diretório que foi criado:

```
cd varnish-cache
```

Gere o script de configuração automática:

```
sh configure
```

Execute o script que foi gerado:

```
sh autogen.sh
```

Compile o código fonte através do comando:

```
make
```

Instalação

Finalmente instale o Varnish Cache com o comando.

```
make install
```

Ele será instalado no diretório "/usr/local/varnish".

3.4.4. Configuração do Varnish Cache

O Varnish pode ter a sua configuração totalmente customizada de acordo com o tipo de aplicação, graças aos seus vários recursos (VARNISH, 2013).

Porém, devido à falta de documentação avançada, podem haver dificuldades quanto ao funcionamento deles. Este trabalho procura diminuir essa curva de aprendizado.

A configuração detalhada neste trecho foi obtida depois de vários estudos do comportamento da aplicação com e sem a utilização do cache.

Configuração de um *backend*

Um *backend* é uma origem em que os dados serão buscados. Essa origem pode estar situada local ou remotamente em outro servidor.

Neste trecho de código é configurado um *backend* de nome local, indicando que a conexão será feita através do 127.0.0.1 na porta 8000.

As configurações *between_bytes_timeout* e *first_byte_timeout* e indicam o tempo máximo de espera entre o recebimento de um byte e o próximo, e o máximo de tempo até recebimento do byte inicial.

Figura 9 Configuração de backend

```
backend local {
        .host = "127.0.0.1";
        .port = "8000";
        .between_bytes_timeout = 70s;
        .first_byte_timeout = 300s;
}
```

Configuração de verificação de saúde de um backend

O trecho *probe* foi configurado para que sejam feitas 5 checagens a cada 25 segundos, caso 3 ou menos dessas checagens indiquem erro, esse *backend* é considerado doente e nenhum conteúdo não é buscado nele até que volte ao normal.

Figura 10 Configuração de verificação de saúde de um backend

```
backend local {
...
        .probe = {
                .url = "/check.txt";
                .interval = 5s;
                .timeout = 1s;
                .window = 5;
                .threshold = 3;
}
...
  }
```

Configuração de um balanceador

O *Varnish* também pode ser utilizado como balanceador de carga. As suas configurações permitem 3 tipos de balanceamento: *round-robin*, *random* e *client*.

O *round-robin* é um tipo balanceamento em círculo, que divide o tempo igualmente entre todos os elementos de seu grupo. Segue abaixo um modelo de configuração:

Figura 11 Balanceador round-robin

```
director balance round-robin {
        {
                .backend = web1;
        }
        {
                .backend = web2;
        }
        {
                .backend = web3;
        }
}
```

O *random* funciona balanceando os acessos de uma maneira aleatória de acordo com pesos especificados para cada *backend* como é demonstrado abaixo:

Figura 12 Balanceador random

```
director balance random {
        {
                .backend = web1;
                .weight = 2;
        }
        {
                .backend = web2;
                .weight = 1;
        }
        {
                .backend = web3;
                .weight = 1;
        }
}
```

O *client* funciona de maneira parecida ao *random*, porém ele pode levar em consideração outros fatores como URL ou IP. Nessa configuração é necessário realizar algumas alterações também na função *vcl_recv*, como é exibido no trecho de código abaixo:

Figura 13 Balanceador client

```
director balance client {
        {
                .backend = web1;
                .weight = 2;
        }
        {
                .backend = web2;
                .weight = 1;
        }
}
sub vcl_recv {
         set req.backend = balance;
           /* Balanceamento por URL */
           set client.identity = req.url;
           /* Balanceamento por IP */
           set client.identity = client.ip;
}
```

Esse balanceador permite com que uma requisição de determinado IP ou URL continue sendo respondida pelo mesmo *backend*.

É necessário lembrar que somente uma configuração deve ser utilizada por vez, ou por URL ou por IP. Caso nenhuma seja inserida, o padrão é o balanceamento por IP.

Configuração do tratamento de requisição

A configuração do tratamento de requisição é feita através da função *vcl_recv*. Para selecionar qual *backend* responderá à requisição, a função *set req.backend* é utilizada:

Figura 14 Função de tratamento de requisição

```
sub vcl_recv {
        #Seleciona o backend a ser utilizado
        set req.backend = balance;
        ...
}
```

Uma informação útil que deve ser gravada aos logs do *Varnish* é o IP do cliente. Para essa inserção, o código abaixo é usado:

Figura 15 Inserção de ip

```
...
remove req.http.X-Forwarded-For;
set    req.http.X-Forwarded-For = client.ip;
...
```

Da forma como está sendo realizado no código, o IP do cliente será gravado no cabeçalho da requisição e ficará gravado junto com a requisição nos logs.

Um dos recursos do *Varnish* é o *grace*. Com ele é possível fazer com que um objeto que esteja armazenado por mais tempo que o seu *TTL* (*time-to-live* ou tempo de vida) seja utilizado. Isso é bastante útil caso haja um atraso na resposta ou o *backend* esteja indisponível.

No trecho abaixo o *grace* está sendo configurado para 15 segundos caso o *backend* esteja saudável e caso contrário 5 minutos:

Figura 16 Configuração do grace

```
...
if (! req.backend.healthy) {
        set req.grace = 5m;
    } else {
        set req.grace = 15s;
    }
...
```

Em quase toda aplicação web é provável que existam partes que não devem ser armazenadas em cache em nenhuma ocasião. Na aplicação deste trabalho existem duas, a administração interna do e-commerce e a geração de XMLs de integração.

Nesses casos em que as requisições não serão cacheadas em nenhum momento é utilizado o *pipe*. Quando esse valor é retornado em um arquivo de configuração do *Varnish*, uma conexão é aberta diretamente entre o cliente e o *backend*.

Através dela nenhum objeto é verificado ou armazenado pelo *Varnish*. Essa conexão pode ser fechada pelo navegador do cliente ou diretamente pelo *Varnish* caso o tempo dela exceda o limite indicado nas suas configurações.

O código a seguir é um exemplo para a utilização do *pipe* caso a URL seja da administração do usuário ou da geração de XML:

Figura 17 Configuração do pipe

```
...
if ( (req.url ~ "/adm/") ||  (req.url ~ "/xml")){
        return(pipe);
 }
...
```

Quando um conteúdo não deve ser armazenado, mas que em algumas situações ele pode ser necessitado a partir do cache, ele pode ser utilizado através do uso do *pass*. Através desse retorno, a existência do objeto em cache não é verificada, sendo buscado diretamente na origem.

Isso é útil em situações em que o objeto sempre é atualizado mas em caso de falha, ou atraso, a entrega de um conteúdo antigo é aceitável.

No exemplo abaixo, são indicadas todas as URLs da aplicação que se encaixam nessa regra:

Figura 18 Configuração do pass

Por fim, é necessário verificar se o objeto da requisição está em cache. O *lookup* é o que informa que a requisição deve ser verificada. No trecho a seguir, é

```
        ...
        if (
```

Figura 19 Configuração do lookup

```
            (req.url ~ "/nocache/") || (req.url ~ "/loja/cadastro_layout") ||
            (req.url ~ "/loja/central_anteriores") ||
            (req.url ~ "/loja/central_cliente") ||
            (req.url ~ "/loja/central_comentarios") ||
            (req.url ~ "/loja/central_confirmar_pagamento") ||
            (req.url ~ "/loja/central_dados") ||
            (req.url ~ "/loja/central_detalhe_pedido") ||
            (req.url ~ "/loja/central_detalhe_pedido") ||
            (req.url ~ "/loja/central_lista_espera") ||
            (req.url ~ "/loja/central_rastrear") ||
            (req.url ~ "/loja/central_senha") || (req.url ~
"/loja/central_troca") ||
            (req.url ~ "/loja/comparador") || (req.url ~ "/loja/contato") ||
            (req.url ~ "/loja/duvida_produto") ||
            (req.url ~ "/loja/form_nao_encontrado") ||
            (req.url ~ "/loja/funcoes/dadosCartaoSuperPay") ||
            (req.url ~ "/loja/indicar_produto") || (req.url ~
"/loja/login_layout") ||
            (req.url ~ "/loja/logout") || (req.url ~ "/loja/navegacao-
visitados") ||
            (req.url ~ "/loja/retorno_pago") ||
            (req.url ~ "/api") ||
            (req.url ~ "/loja/finalizar(.*)"))
        {
            return(pass);
        }
        ...
```

solicitado que a requisição seja verificada pelo *Varnish* caso ela for um *POST* ou um *GET*:

Configuração de acerto

Quando o objeto de uma requisição feita é encontrado no cache, chama-se acerto. O método *vcl_hit* do Varnish define o que acontecerá caso isso ocorra.

Um exemplo de utilização desse método é ignorar o cache de um conteúdo quando a requisição vier de um IP determinado.

No exemplo abaixo caso o ip seja 111.111.111.111 o *TTL* do objeto é definido como 0 é retornado *pass*, caso contrário ele retorna *deliver*:

Figura 20 Configuração de acerto

```
sub vcl_hit {
        if (req.http.X-Real-IP == "111.111.111.111") {
                set obj.ttl = 0s;
                return(pass);
        }else {
                return(deliver);
        }
}
```

Configuração de falha

Quando o objeto de uma requisição feita não é encontrado no cache, chama-se erro. O método *vcl_miss* define o procedimento a ser executado caso isso ocorra. O exemplo a seguir faz com que o objeto quando não encontrado em cache seja entregue diretamente do *backend* através do retorno *fetch*:

Figura 21 Configuração de falha

```
sub vcl_miss {
        return(fetch);
}
```

Configuração de erro

Caso ocorra algum erro e o conteúdo não possa ser entregue nem do cache e nem do *backend* a função *vcl_error* é utilizada para indicar o que será realizado. O trecho de código abaixo faz com que uma mensagem indicando o erro seja exibida contendo o erro informado pelo *backend*:

Figura 22 Configuração de erro

```
sub vcl_error {
    set obj.http.Content-Type = "text/html; charset=utf-8";
    synthetic {"
        <?xml version="1.0" encoding="utf-8"?>
        <!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
        <html> <head>
          <title>"} + obj.status + " " + obj.response +
{"</title>
        </head> <body>
          <h1>Error "} + obj.status + " " + obj.response
+ {"</h1>
          <p>XID: "} + req.xid + {"</p>
        </body></html>
    "};
```

Configuração da função de coleta de objeto

Quando um objeto é coletado do servidor de origem, é necessário informar e configurar alguns dados no objeto para que nas próximas requisições, esse objeto em cache possa ser útil.

Para realizar tais procedimentos, é utilizada a função *vcl_fetch*.

Dentro dela é possível também, executar o tratamento de erros retornados pelos servidores de origem. Alguns exemplos serão citados adiante.

Figura 23 Função de coleta

```
sub vcl_fetch {
        ...
}
```

Configuração do tempo que um objeto é mantido em memória

Como citado na configuração de tratamento da requisição, a funcionalidade *grace* permite que um objeto seja utilizado mesmo que seu tempo de vida tenha excedido o limite. Para que isso seja possível é necessário realizar uma configuração dentro da função *vcl_fetch* indicando quanto tempo um objeto deve ser mantido após seu TTL expirar.

Um exemplo seria: um objeto com *TTL* de 20 minutos será armazenado por esse tempo, passado esse período ele será descartado.

Utilizando o *grace* é possível que esse objeto seja armazenado por um tempo maior, assim, caso algo que possa atrasar a renovação do objeto não atrasará a reposta pois o conteúdo antigo será exibido.

No trecho de código a seguir esse tempo está configurado para 15 minutos:

Figura 24 Configuração de utilização do grace

```
sub vcl_fetch {
        ...
        set beresp.grace = 15m;
        ...
}
```

Configuração do TTL de um objeto

A configuração demonstrada na próxima imagem, estabelece um TTL de uma hora para o objeto caso a requisição tenha sido um POST ou um GET.

Figura 25 Configuração do TTL

```
sub vcl_fetch {
        ...
        if ((req.request == "POST" || req.request == "GET"))
{
                set beresp.ttl = 1h;
        }
...
```

Configuração de tratamento do código 404(*Not Found*)

Quando o código 404 é retornado pelo servidor, indica que o objeto requisitado não foi encontrado.

Para que essa busca pelo objeto no *backend* não seja explorada e possa vir a sobrecarregar o servidor de origem, um cache desse erro é mantido e exibido durante uma hora.

Isso faz com que a busca por esse objeto inexistente não seja feita, aliviando a carga do servidor em um possível incidente.

Figura 26 Configuração de tratamento de 404

```
sub vcl_fetch {
        ...
        if (beresp.status == 404) {
                set beresp.ttl = 1h;
        }
...
 }
```

Configuração de tratamento de erros

É provável que um servidor esteja funcionando corretamente porém um requisição específica esteja com algum problema. Assim, se uma requisição por algum motivo não consegue ser respondida pelo *backend*, ele retorna um código de erro. Com o um recurso chamado *Saint Mode* do Varnish é possível reduzir o impacto desse erro ou até mesmo fazer com que ele não seja notado pelo o usuário final.

Ele marca o *backend* como inválido para essa requisição, o que faz com que o Varnish busque o objeto em uma outra origem ou entregue um conteúdo que esteja em cache.

Com o código abaixo, a configuração é realizada para que se qualquer um dos erros 403, 400, 500, 501, 502, 503 e 504 ocorra, o *backend* seja indicado como inválido por 10 segundos e um *restart* seja retornado para que o objeto possa ser procurado em outro *backend*.

Figura 27 Configuração de tratamento de erros

```
sub vcl_fetch {
        ...
        if (beresp.status == 403 ||
          beresp.status == 400 ||
          beresp.status == 500 ||
          beresp.status == 501 ||
          beresp.status == 502 ||
          beresp.status == 503 ||
          beresp.status == 504)
          {
                set beresp.saintmode = 10s;
                return(restart);
        }
...
 }
```

Configuração de entrega

Por fim, antes de um objeto ser entregue é possível configurar ações para serem realizadas através da função *vcl_deliver*. Um dos usos mais comuns é a inserção de cabeçalhos.

No exemplo da próxima figura é inserido um cabeçalho de nome X-Cache que indica se o objeto entregue foi encontrado no cache. Se sim, o valor dele é “HIT”, se não, “MISS”.

Figura 28 Configuração de entrega

```
sub vcl_deliver {
        if (obj.hits > 0) {
                set resp.http.X-Cache = "HIT";
        } else {
                set resp.http.X-Cache = "MISS";
        }
}
```

CAPÍTULO 4 - RESULTADOS

Neste trabalho foi implementado um conjunto de técnicas que visa um melhor aproveitamento dos recursos computacionais através do uso eficiente dos recursos de cache.

Para testar a eficiência da solução proposta foi utilizado o seguinte ambiente de teste:

- Um servidor é uma máquina virtualizada com 2 processadores virtuais Intel® "Meron" Gen. (Xeon® Core™2) com dois núcleos cada, 2GB de memória e disco compartilhado de 302GB.
- O sistema operacional instalado é o CentOS 6 (64-bit).
- Não foi utilizado um ambiente local controlado.

Foram executados testes de carga e volume, ou seja, uma injeção de um determinado tráfego na rede. Os testes foram realizados utilizando a ferramenta JMeter (JMETER, 2013) com a seguinte configuração:

- Foram criados 4 casos de testes no JMeter, dividindo os usuários em grupos.
  - 1 usuário realizando um número infinito de requisições por 5 minuto.
  - 10 usuários realizando um número infinito de requisições por 5 minuto.
  - 100 usuários realizando um número infinito de requisições por 5 minuto.
  - 1000 usuários realizando um número infinito de requisições por 5 minuto.

Todos os grupos foram submetidos a testes por 3 vezes. Os valores apresentados neste capítulo são o resultado de uma média de cada grupo.

Para a coleta a coleta de dados do servidor durante os testes, foi utilizada a ferramenta de análise New Relic (NEWRELIC, 2013).

4.1. Métricas

Foram definidas algumas métricas com o objetivo de comparar os cenários propostos verificando o comportamento das soluções.

Taxa de Transferência (Vazão): Quantidade máxima de requisições que o servidor (aplicação) pode suportar.

Latência (Latência Média): Latência é o tempo gasto por uma requisição desde a origem até o destino. Esse tempo é absoluto e leva em consideração o tempo de processamento da requisição e o tempo de comunicação.

Variação na Latência (Latência mediana, Latência mínima e Latência máxima): O tempo de chegada de uma requisição pode variar devido ao fato de trafegar na rede por diferentes rotas. O que, pode diminuir a qualidade do serviço.

Erros (Taxa de erro): Porcentagem de requisições que não foram respondidas pela aplicação por motivos diversos como: tempo de limite excedido, erros de processamento da requisição pelo servidor devido à alta carga de processamento, etc.

Carga de uso do servidor (Load médio): Em sistemas UNIX a carga é medida de acordo com a quantidade de trabalho computacional que o sistema consegue realizar. Essa carga é relacionada ao número de núcleos de processamento existentes na máquina. No caso do servidor utilizado nesse trabalho, que possui 4 núcleos, o *Load* médio deve ficar abaixo de 4. Qualquer valor acima disso indica que há uma carga além da suportada e que existem processos na fila para serem executados.

4.2. Resultados do cenário sem cache

Os resultados no cenário sem cache foram obtidos após a execução dos casos de testes, na qual o número de usuários é fixo incrementando na proporção de 1, 10, 100 e 1000 executados separadamente.

4.2.1. Resultados de desempenho

Os resultados dos testes sem cache foram organizados nas seguintes tabelas:

Tabela 1 Desempenho sem cache 1

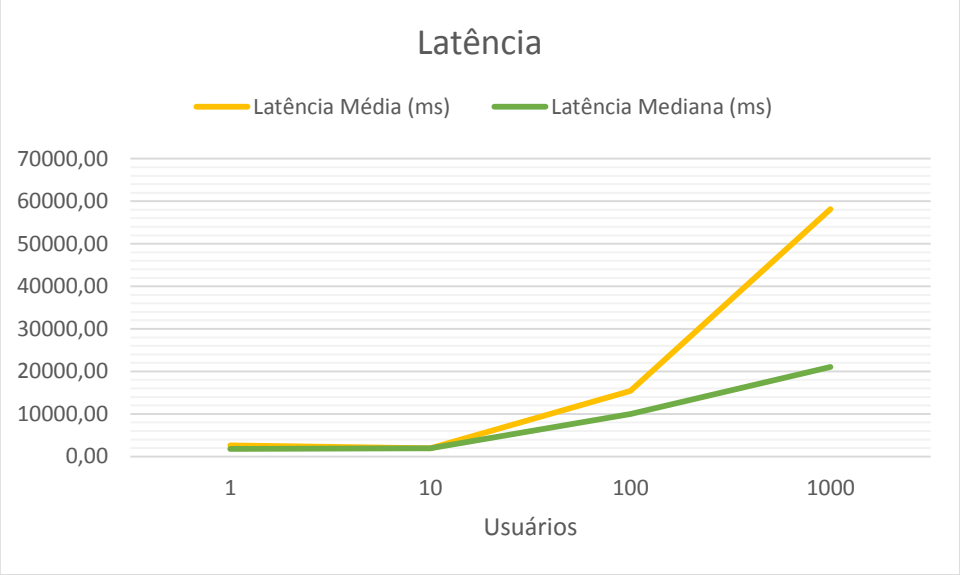
Usuários	Latência Média (ms)	Latência Mediana (ms)	Latência Mín. (ms)	Latência Máx. (ms)
1	2620,33	1747,67	827,33	34329,00

10	1957,00	1953,67	759,33	4818,67
100	15408,33	9972,67	190,67	7803661,33
1000	58151,33	21013,33	3508,67	607530,33

De acordo com o dados da Tabela 1 e do Gráfico 1 é possível notar que com 1 e 10 usuários os valores se mantém estáveis. Já com 100 usuários, o aumento da latência é considerável, quase sete vezes mais, chegando a uma latência máxima de duzentas vezes mais que a latência máxima nos grupos de 1 e 10 usuários.

No último grupo, as latências média e mediana continuam aumentando, chegando a quase 4 vezes os valores do grupo anterior, porém sem uma latência máxima tão alta.

Gráfico 1 Latência sem cache



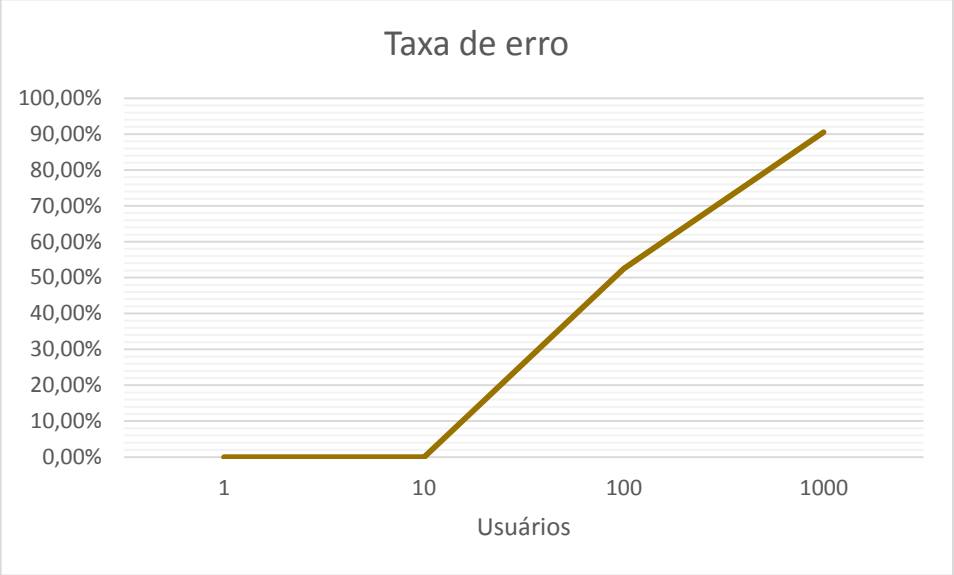
Conforme a Tabela 2 e o Gráfico 2, a taxa de erro se mantém em 0 por cento enquanto os testes com 1 e 10 usuários são realizados. Aumentando para 52,54% com 100 usuários e para 90,53% para 1000 usuários.

Tabela 2 Desempenho sem cache 2

Usuários	Taxa de Erro	Vazão
1	0,00%	0,56
10	0,00%	5,13

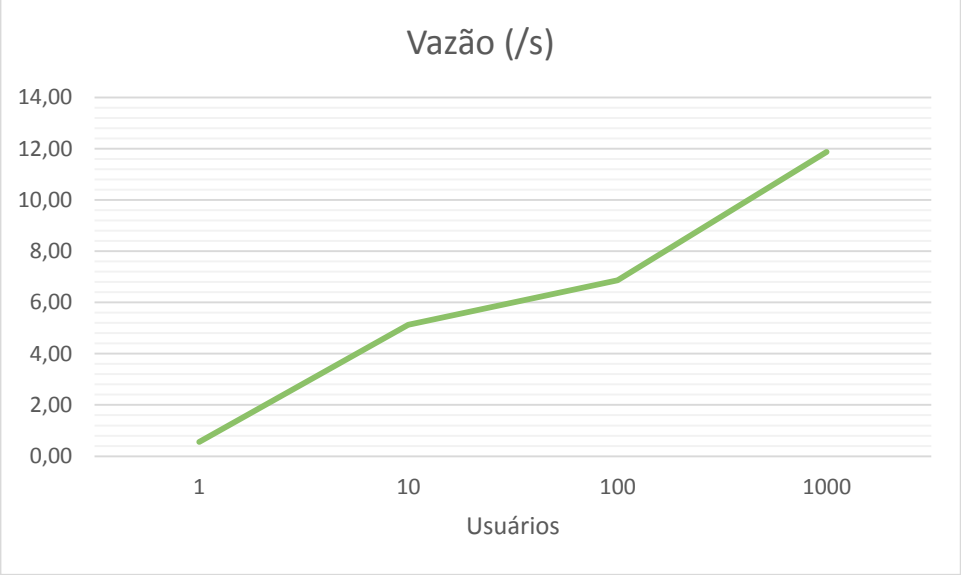
100	52,54%	6,87
1000	90,53%	11,87

Gráfico 2 Taxa de erro sem cache



Os dados indicados na Tabela 3 e no Gráfico 3, mostram que a vazão é crescente mesmo a taxa de erro aumentando.

Gráfico 3 Vazão sem cache



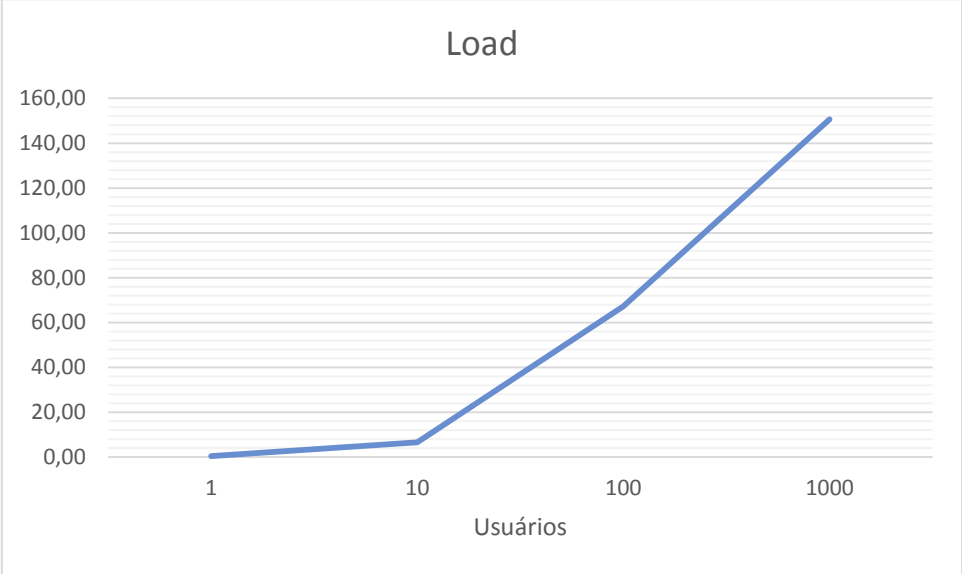
4.2.2. Resultados de impacto no ambiente

Durante os testes, o *load*, que indica a carga de utilização dos recursos no servidor, também foi monitorado. Esse monitoramento gerou a Tabela 3 e o Gráfico 4:

Tabela 3 Load sem cache

Usuários	Load médio
1	0,48
10	6,64
100	67,33
1000	150,67

Gráfico 4 Load sem cache



É possível identificar através destes dados que o *load* médio se mantém baixo com 1 usuário e já acima do padrão com 10. Isso deve-se ao fato de todas as aplicações estarem em um só servidor.

Já com 100 usuários ele sobe quase 10 vezes. Essa carga muito alta no servidor é o principal fator para a taxa de erro ter subido, Gráfico 2 Taxa de erro sem cache.

Com 100 usuários o *load* chega a 150, aproximadamente 37 vezes mais que o ideal de funcionamento desse servidor.

4.3. Resultados do cenário com cache

Os resultados no cenário com cache foram obtidos após a execução dos casos de testes, na qual o número de usuários é fixo incrementando na proporção de 1, 10, 100 e 1000 executados separadamente.

4.3.1. Resultados de desempenho

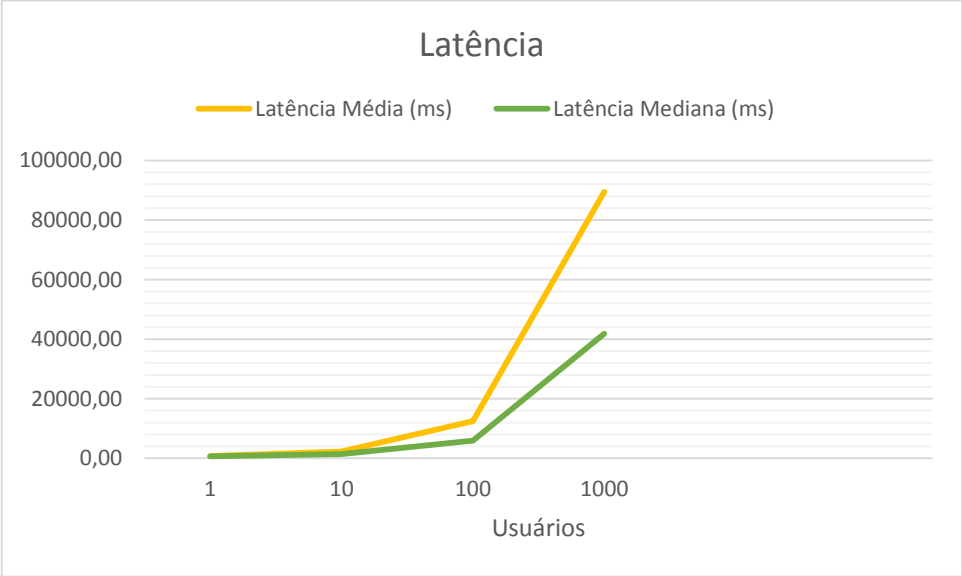
Os resultados dos testes com cache foram organizados nas seguintes tabelas:

Tabela 4 Desempenho com cache 1

Usuários	Latência Média (ms)	Latência Mediana (ms)	Latência Mín. (ms)	Latência Máx. (ms)
1	796,67	689,67	264,67	4823,33
10	2243,33	1408,33	393,33	113043,67
100	12539,33	5894,67	933,00	463541,00
1000	89488,33	41876,33	3255,33	697509,67

De acordo com os dados da Tabela 4Tabela 4 e do Gráfico 5 é possível identificar que a latência é crescente em todas as classificações dela.

Gráfico 5 Latência com cache



Conforme os dados da Tabela 5 e do Gráfico 6 é possível notar que a taxa de erro é crescente durante todo o teste.

Iniciando com 0% de erro para o grupo de 1 usuário. Ficando em 0,12% para 10 usuários. E se mantendo abaixo de 1% também para 100 usuários. Somente com 1000 usuários é que a taxa de erro passa de 1% e sobe para 7,94%.

Tabela 5 Desempenho com cache 2

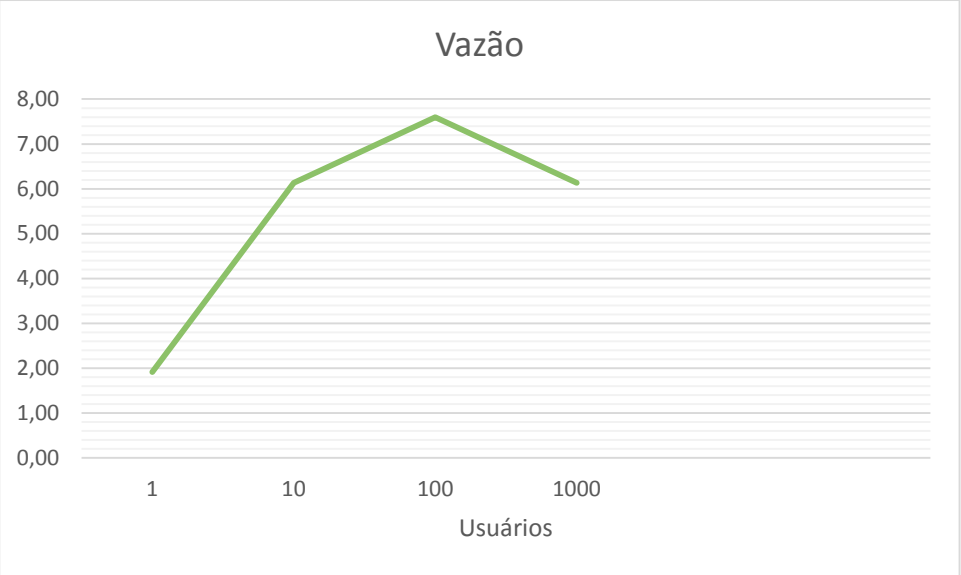
Usuários	Taxa de Erro	Vazão
1	0,00%	1,91
10	0,12%	6,13
100	0,92%	7,60
1000	7,94%	6,13

Gráfico 6 Taxa de erro com cache

Os dados da Tabela 5 e do Gráfico 7 mostram que a vazão se mantém crescente até 100 usuários, onde tem o limite de 7,6 por segundo.

Com 1000 usuários esse valor diminui para 6,13.

Gráfico 7 Vazão com cache



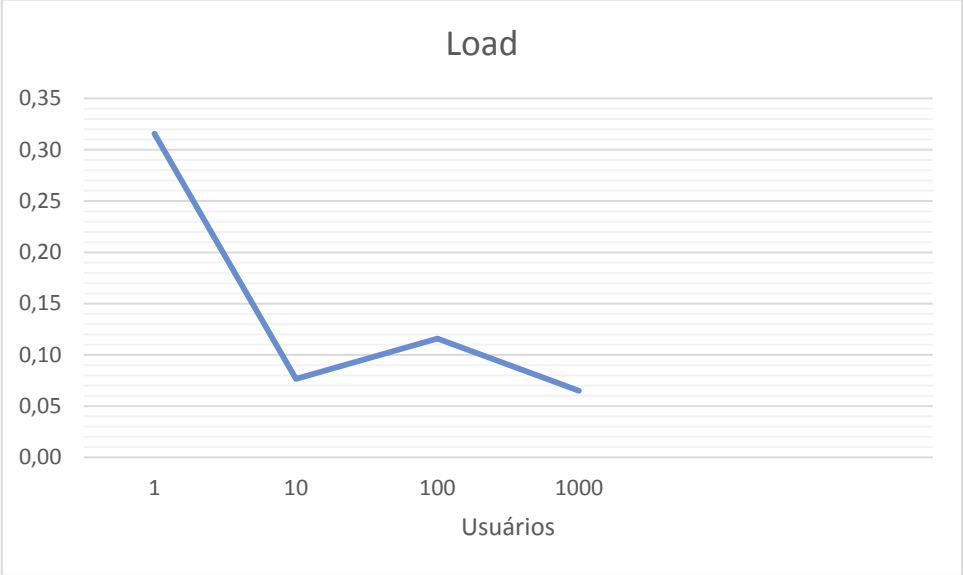
4.3.2. Resultados de impacto no ambiente

Durante os testes, o *load*, que indica a carga de utilização dos recursos no servidor, também foi monitorado. Esse monitoramento gerou a seguinte tabela:

Tabela 6 Load com cache

Usuários	Load médio
1	0,32
10	0,08
100	0,12
1000	0,06

Gráfico 8 Load com cache



É possível identificar através dos dados da Tabela 6 e do Gráfico 8 que o *load* médio se mantém abaixo de 1 em todos os grupos de usuários.

Isso acontece devido ao cache entregar o conteúdo e a utilização dos recursos do servidor ser realizada somente quando extremamente necessária.

4.4. Análise dos resultados

Nesta análise os resultados dos dois cenários foram combinados para uma melhor visualização da evolução no desempenho da aplicação e do servidor em que ela está instalada.

4.4.1. Latência

A latência média e mediana como mostradas nos gráficos de Comparativo de latência média e Comparativo de latência mediana ficaram menores no cenário com cache até 100 usuários.

Com 1000, o cenário sem cache foi mais rápido o outro. Esse fato está diretamente ligado à taxa de erro, pois a latência de uma resposta de erro é menor que a de uma página completa.

Analisando de uma forma geral, mesmo a latência sendo maior, o cenário de cache teve um desempenho melhor.

Gráfico 9 Comparativo de latência média

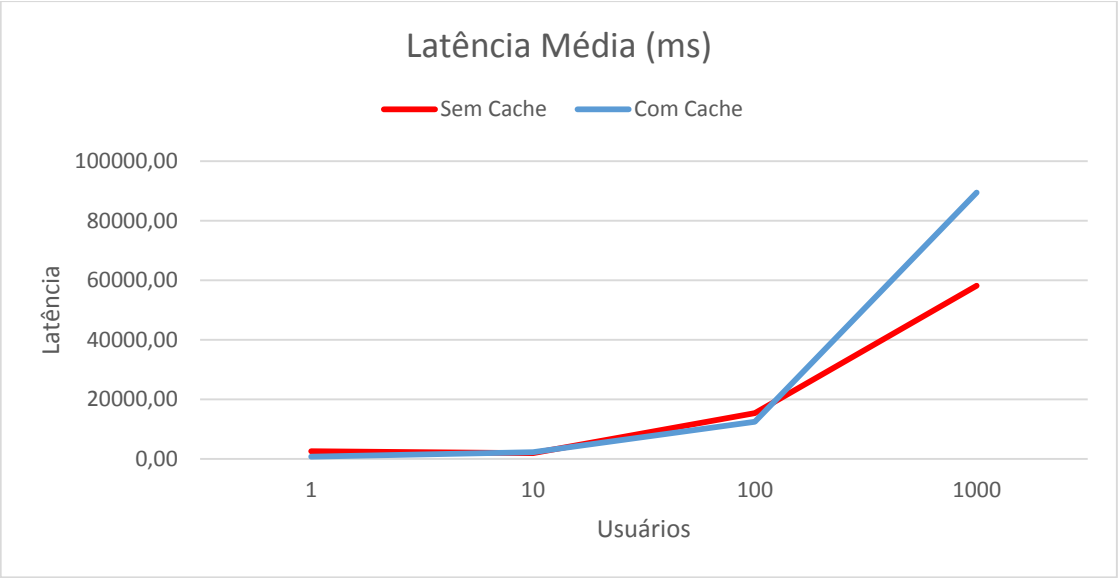
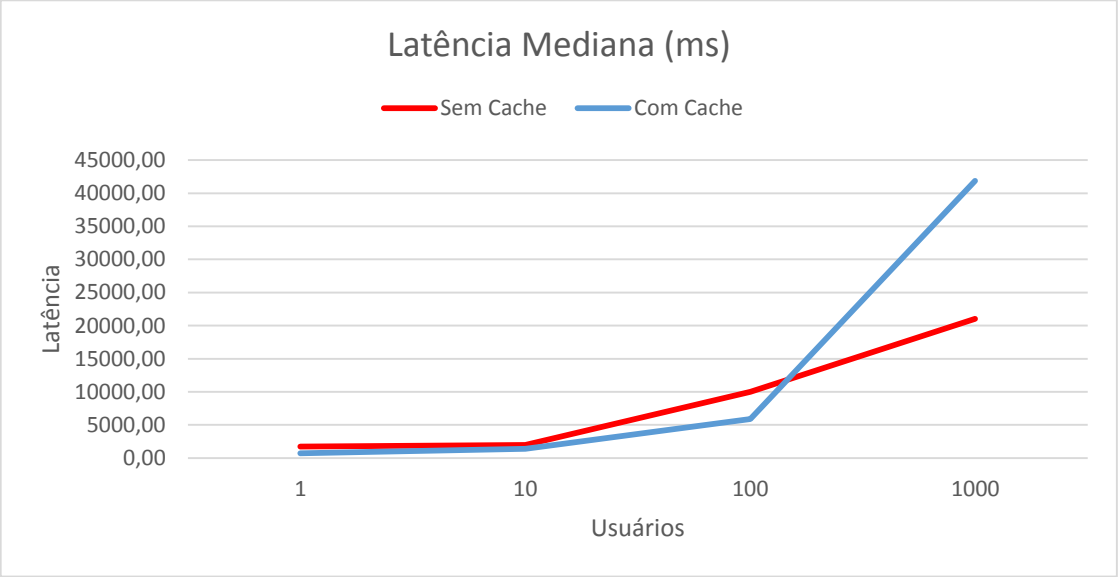


Gráfico 10 Comparativo de latência mediana

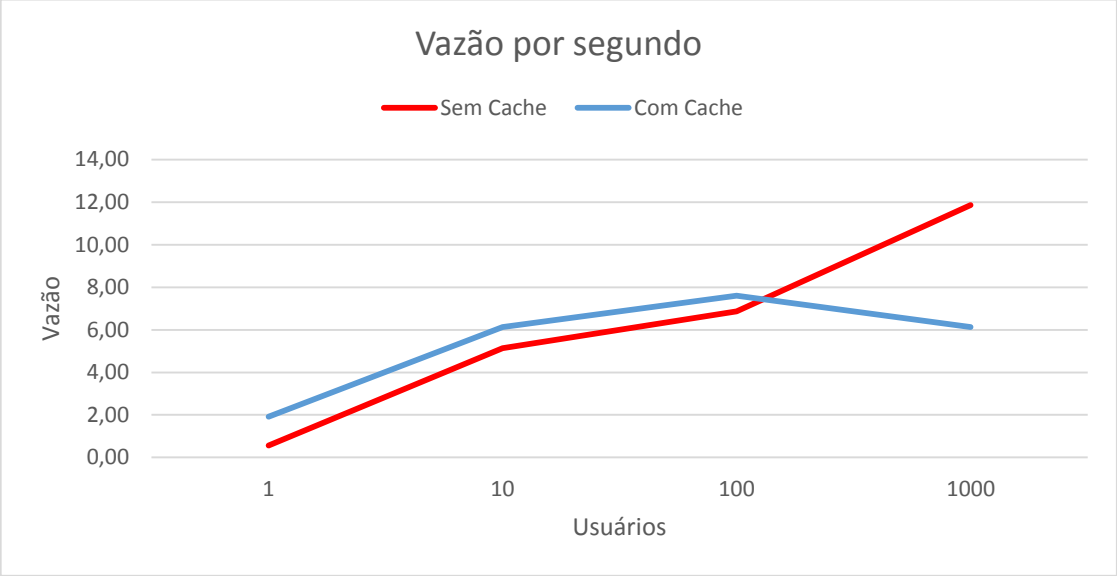


4.4.2. Vazão

Similar ao ocorrido com a latência, a vazão (Gráfico 11 Comparativo de vazão) cenário com cache foi melhor até 100 usuários, reduzindo a sua eficácia no grupo de 1000 usuários.

Também relacionado à taxa de erros, a vazão maior do cenário sem cache se dá devido ao grande número de erros retornados por ele.

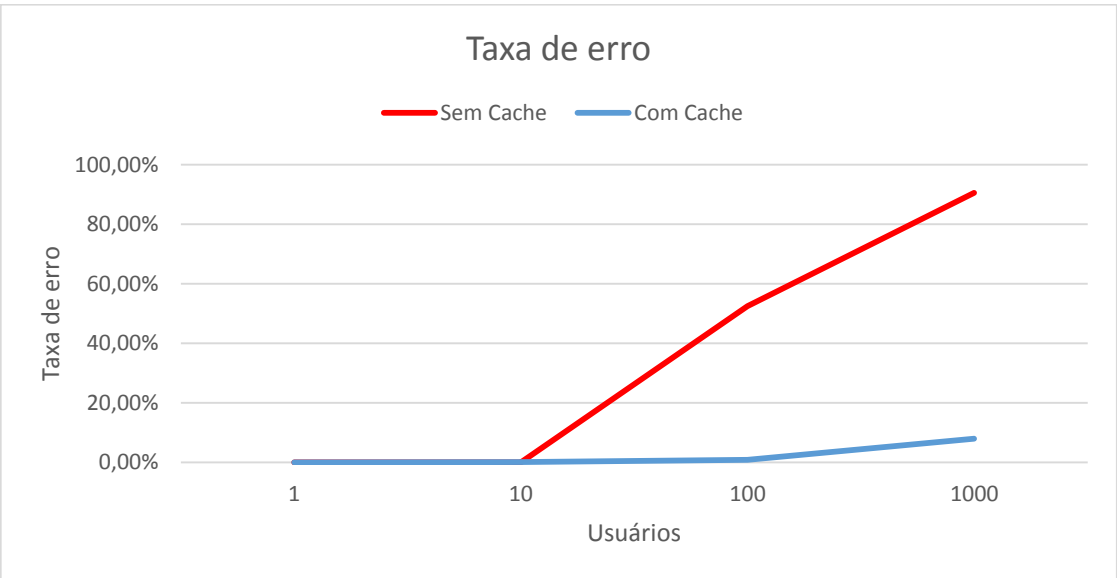
Gráfico 11 Comparativo de vazão



4.4.3. Taxa de erro

Como exibido no Gráfico 12, a taxa de erro apresentou um dos resultados mais expressivos, reduzindo em mais de 80% a taxa de erros nos testes com 1000 usuários.

Gráfico 12 Comparativo de taxa de erro



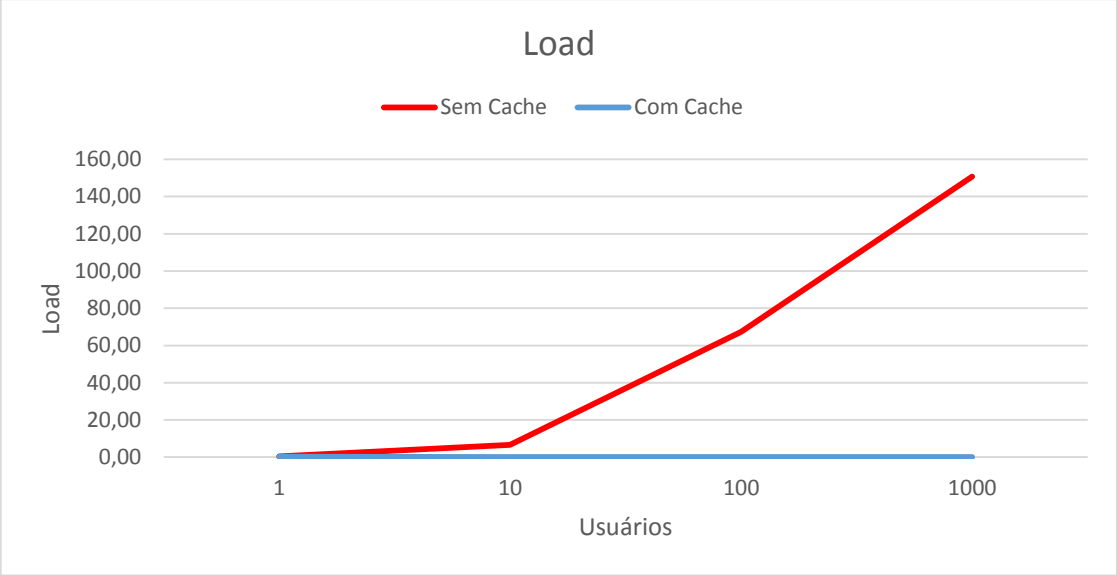
4.4.4. Load médio

O gráfico da Gráfico 13 constata que o impacto das configurações do cenário com cache na carga do servidor foi muito significativo.

Nos testes realizados sem cache, uma carga de processamento muito maior do que o ideal foi exigido, chegando a elevar seu *load* a um valor de 150.

Já no cenário com cache, a carga se manteve em todos os momentos abaixo de 4, sendo assim, dentro do suportado pelo servidor.

Gráfico 13 Comparativo de load



CAPÍTULO 5 - CONCLUSÕES

As estratégias de cache em serviços web de forma geral são extremamente importantes, entretanto, existe uma dificuldade nas empresas em tratar de forma eficiente esse tipo de situação.

Desenvolvedores em especial tem receio de perder o controle de seu sistema, pois um proxy pode tornar difícil identificar quem o está acessando. Entretanto com configurações corretas e utilizando as políticas adequadas pode-se melhorar o desempenho de sistemasp Web.

Partindo-se dos resultados obtidos pode-se verificar que existe na maioria dos testes, um ganho em se fazer uso de cache para sistemas web. Esse ganho está presente tanto na área de qualidade de serviço quanto na de custos.

Na área de qualidade de serviço, melhorando o tempo de carregamento de página, diminuindo o tempo de resposta e aumentando a disponibilidade do serviço mesmo que problemas ocorram com o servidor da aplicação.

Em custos, pode-se reduzir e otimizar a infraestrutura necessária de uma aplicação, tornando a aplicação mais escalável e diminuindo a necessidade real de aquisição de novos servidores.

Acredita-se que esse trabalho possa contribuir significativamente com a diminuição da curva de aprendizado quanto ao manuseio e configuração de sistemas de cache e com o desenvolvimento da arquitetura dos sistemas web atuais e futuros.

5.1. Trabalho futuros

Com base no trabalho produzido e nos resultados apresentados, as possibilidades de trabalhos futuros são:

- Análise de desempenho em um ambiente local controlado ou um ambiente robusto com diversos servidores.
- Implementação e teste em aplicações diferentes, como aplicações webservices e REST.
- Estudo e comparação com outras tecnologias e ferramentas de cache como Squid e Nginx.
- Implementação e teste de um cluster de servidores de cache.

REFERÊNCIAS

AKAMAI**. Site oficial**. Disponível em: <http://www.akamai.com/>. Acesso em: 11 de agosto de 2013.

BERNERS-LEE, T. **World Wide Web Seminar**. 1991. Disponível em <www.w3.org>.

BERNERS-LEE, T. **Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web**. 2ª edição, Harper Paperbacks, 2000.

BERNERS-LEE, T. **Propagation, Replication and Caching on the Web**. Disponível em <http://www.w3.org/pub/WWW/Propagation/Activity.html>.

BRADLEY, Adam D.; BESTRAVOS, Azer. **Basis Token Consistency: Supporting Strong Web Cache Consistency**. Departamento de Ciência da Computação da Universidade de Boston. 2002

BUYYA, R.; PATHAN, M.; VAKALI, A. **Content Delivery Networks**. 1ª Edição, Springer, 2008.

COULORIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos: Conceitos e projeto**. Tradução de João Tortello, 4ª ed., Porto Alegre, Bookman, 2007.

IETF, Internet Engineering Task Force. **Hypertext Transfer Protocol -- HTTP/1.1**. 1991. Disponível em <http://www.ietf.org/rfc/rfc2616.txt>.

JMETER, Apache. **Site oficial**. Disponível em < http://jmeter.apache.org/ >. Acessado em 15 de novembro de 2013.

MYSQL. **Site oficial**. Disponível em: <http://www.mysql.com/>. Acesso em: 11 de agosto de 2013.

MURTA, Cristina D.; ALMEIDA, Virgílio Augusto F. **Modelo de Particionamento de Espaço para Caches da World Wide Web**. Universidade Federal de Minas Gerais, 2000.

NEWRELIC. **Site oficial**. Disponível em < http://newrelic.com/ >. Acessado em 18 de novembro de 2013

NYGREN, Erik; SITAMARAN, Ramesh K.; SUN, Jennifer. **The Akamai Network: A Platform for High-Performance Internet Applications**. ACM SIGOPS Operating Systems Review, Vol. 44, No.3, 2010.

PENG, G. **CDN: Content Distribution Network**. Technical Report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY 2003.

PHPNET. **Manual do PHP**, Disponível em: <http://www.php.net/manual/pt_BR/preface.php>. Acesso em 11 de agosto de 2013.

PINHEIRO, João Carlos. **Avaliação de Políticas de Substituição de Objetos em Caches na Web**. Tese de Doutorado. Dissertação de Mestrado, USP/ICMC, 2001.

VARNISH, Software. Varnish Cache. Disponível em <http://www.varnish-cache.org>. Acessado em 20 de outubro de 2013.

VARNISH, Software. **Varnish Book**. Disponível em <www.varnish-software.com/static/book/>. Acessado em 20 de outubro de 2013.

WANG, Jia. A survey of web caching schemes for the internet. **ACM SIGCOMM Computer Communication Review**, v. 29, n. 5, p. 36-46, 1999.

WESSELS, D.; CLAFFY, K. ICP and the Squid Web Cache. **IEEE Journal on Selected Areas in Comunication**, v.16, n.3, abr 1998.