SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura : _____

Uso de caches na web – Influência das políticas de substituição de objetos

Jacqueline Augusto de Oliveira

Orientador: *Prof. Dr. Marcos José Santana*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências, na área de Ciências de Computação e Matemática Computacional.

USP-São Carlos
Abril de 2004

Uso de caches na web — Influência das políticas de substituição de objetos

Jacqueline Augusto de Oliveira

Agradecimentos

À Universidade de São Paulo, USP - Campus São Carlos, pela oportunidade de realizar o curso de Pós-Graduação.

A meu orientador, Marcos José Santana, pela valiosa orientação, confiança e amizade durante o curso e execução do trabalho, o qual não poupou dedicação ao meu amadurecimento e formação profissional, além de ter aberto as portas do LaSDPC no momento oportuno.

À professora Regina Helena Carlucci Santana pelo auxílio, amizade e ensinamentos transmitidos.

A todos os amigos do grupo de pesquisa do LaSDPC do ICMC-USP.

Aos amigos do laboratório 160, André, Bruno, Jean, Leonardo, Luis Marcelo, Marcelo e Waldo pela amizade e incentivo em todos os momentos.

Aos amigos Angelo, Fernando e Paulinho.

E a todos que, direta ou indiretamente, contribuíram de alguma forma para a realização deste trabalho.

Este documento foi elaborado com o formatador de textos L^AT_EX.
©Copyright 2004 por Jacqueline Augusto de Oliveira
jac@icmc.usp.br
jac-oliveira@uol.com.br
Todos os direitos reservados.

À minha família
É na educação dos filhos que se revelam as virtudes dos pais.

Resumo

Este trabalho tem como objetivo analisar a influência provocada pelas políticas de substituição de objetos em caches na Web. Isso é feito por meio da investigação das políticas existentes na literatura, considerando um estudo de caracterização de carga, de avaliação de desempenho e de comparação do uso dessas políticas. Para realizar a avaliação das políticas é utilizado um simulador de caches para a Web.

Durante a pesquisa, foi desenvolvida uma nova política, denominada MeMoExp. Ela utiliza os conceitos de Média Móvel para otimizar HR e BHR. As simulações realizadas mostraram que a MeMoExp segue a mesma tendência da política FBR, tida como eficiente na literatura.

Finalmente, são expostas algumas ponderações sobre as idéias apresentadas nos capítulos componentes desta dissertação, além de serem apresentadas as contribuições provenientes desta pesquisa e alguns trabalhos futuros propostos a partir desta dissertação.

Abstract

This work aims to analyse the influence of the replacement policies on web caches. This is carried out by investigating the policies found at the literature, considering an study of load characterization and performance assessment as well as a comparison between the policies' usage. All the experiments are done using a web cache simulator.

During the research, it was developed a new policie, called MeMoExp. It utilizes the concept of Moving Exponential Average to optimize the HR and BHR metrics. The simulation studies showed that the MeMoExp policie follows the same tendency than the FBR metric, which is considered efficient in the literature.

Finally, some considerations about the ideas presented in the dissertation are exposed. The contributions of this research work are presented and some future works are proposed.

Sumário

Lista de Abreviaturas	vi
1 Introdução	1
1.1 Contextualização	1
1.2 Motivação e relevância	2
1.3 Objetivos	3
1.4 Estrutura	3
2 World-Wide Web	5
2.1 Considerações Iniciais	5
2.2 Histórico	5
2.2.1 Hipertexto	6
2.3 Protocolos da <i>Web</i>	6
2.3.1 Protocolo de Comunicação - HTTP	6
2.4 Cliente/Servidor	8
2.5 Desempenho em Servidores Web	10
2.5.1 Requisitos de desempenho	11
2.5.2 Uso de <i>Caches</i>	11
2.6 Aplicações <i>Web</i>	12
2.7 Servidor <i>Proxy</i>	14
2.8 Considerações Finais	14
3 Cache na Web	15
3.1 Considerações Iniciais	15
3.2 Histórico	15
3.3 Caches Tradicionais	18
3.4 Cache na Web	19
3.4.1 Dispositivo de Armazenamento em Cache	21
3.4.2 Modelos de Implementação de um Cache	22
3.4.3 Armazenamento em Cache	24
3.5 Importância do uso de cache na Web	25
3.5.1 Vantagens do uso de cache	25
3.5.2 Desvantagens no uso de cache	26
3.6 Problemas no uso de caches	26

3.7	Consistência do Cache	27
3.7.1	Coerência forte e fraca	29
3.8	Conteúdos que não podem ser mantidos em Cache	30
3.9	Uso de Hierarquia	31
3.10	Cache Distribuído e Hierárquico	32
3.11	Roteamento de Proxy na Web	32
3.12	Particionamento do Cache	33
3.13	Substituição de cache	34
3.14	Políticas de Substituição	34
3.15	Considerações Finais	34
4	Políticas de Substituição	36
4.1	Considerações Iniciais	36
4.2	Avaliação das Políticas de Substituição	36
4.3	Políticas que usam um Parâmetro na Substituição	37
4.3.1	<i>Least Recently Used</i> (LRU)	37
4.3.2	<i>Least Frequently Used</i> (LFU)	37
4.3.3	LFU-Aging	38
4.3.4	LFU*	38
4.3.5	LFU*-Aging	38
4.3.6	Size	39
4.3.7	<i>Least Dynamic Frequency Rule</i> (LDRF)	39
4.3.8	<i>Localized Least Dynamic Frequency</i> (LLDR)	39
4.4	Políticas que Usam dois Parâmetros na Substituição	40
4.4.1	<i>Segmented LRU</i> (SLRU)	40
4.4.2	LRU-K	40
4.4.3	<i>Frequency-Based Replacement</i> (FBR)	40
4.4.4	<i>Greedy Dual-Size</i> (GD-Size)	41
4.4.5	<i>Pyramidal Selection Scheme</i> (PSS)	41
4.4.6	<i>Lowest Relative Value</i> (LRV)	42
4.5	Políticas que usam Múltiplos Parâmetros na Substituição	42
4.5.1	Algoritmo baseado no tempo de recuperação dos objetos	42
4.5.2	<i>Hybrid</i>	43
4.5.3	Dinâmica	43
4.5.4	Peso	44
4.6	Vantagens e Desvantagens das Principais Políticas	45
4.7	Fatores que influenciam a substituição	46
4.8	Classificação das políticas	47
4.9	Vantagens e Desvantagens da Classificação	48
4.10	Limitações das Políticas de Substituição	51
4.11	Considerações Finais	52
5	Simulação na Web	53
5.1	Considerações Iniciais	53
5.2	Probabilidade e Estatística para Simulação	53
5.2.1	Variável Aleatória	53
5.2.2	Eventos Independentes	54

5.2.3	Distribuição de Probabilidade e Função densidade	54
5.2.4	Média ou Valor Esperado	55
5.2.5	Variância e Desvio Padrão	55
5.2.6	Amostragem de dados e estimativa de parâmetros	56
5.2.7	Intervalo de confiança	56
5.3	Média Móvel	56
5.3.1	Média Móvel Simples	57
5.3.2	Média Móvel Exponencial	57
5.4	Uso da estatística para análise dos dados	58
5.5	Simulação de Sistemas	59
5.5.1	Tipos de Simulação	61
5.6	Linguagens de Simulação	62
5.7	Planejamento da Simulação	63
5.7.1	Definição dos Objetivos	63
5.7.2	Escolha das Métricas Adequadas	63
5.7.3	Escolha de Parâmetros	64
5.7.4	Quantidade de replicações	65
5.7.5	Análise dos Resultados	65
5.8	Ambiente de Simulação	66
5.9	Considerações Finais	68
6	Avaliação de Desempenho utilizando uma nova política	70
6.1	Considerações Iniciais	70
6.2	Introdução	70
6.3	MeMoExp	71
6.4	Resultados da Pesquisa	72
6.5	Avaliação das Políticas de substituição	74
6.6	Medindo a eficiência do cache	77
6.7	Considerações Finais	78
7	Conclusão	81
7.1	Análise crítica da revisão bibliográfica	81
7.2	Contribuições e Conclusões	82
7.3	Trabalhos Futuros	82
	Bibliografia	85

Lista de Figuras

2.1	Intervalos de Complexidade e níveis de orientações das aplicações Web	13
3.1	Memória cache tradicional	18
3.2	Funcionamento de um cache para Web	19
3.3	Uso de cache	19
3.4	Fluxograma - Requisição de uma página web	20
3.5	Diferenças entre Caches Tradicionais e Caches Web	21
3.6	Arquitetura de caches na web	23
3.7	Roteamento de <i>proxy</i> na Web	33
4.1	Seções usadas na política FBR	41
5.1	Linha de um arquivo de registro de traces	59
5.2	Estrutura básica do simulador	66
5.3	Principais estruturas de dados do simulador	67
5.4	Arquivos que compõe o simulador	67
5.5	Tipos de arquivos	68
6.1	CISC - Comparação das políticas de substituição	76
6.2	BO2 - Comparação das políticas de substituição	76
6.3	PA - Comparação das políticas de substituição	76
6.4	SV - Comparação das políticas de substituição	77

Lista de Tabelas

2.1	Estrutura de Protocolos em Camadas	7
2.2	Códigos de status geralmente retornados por um servidor Web	9
2.3	HTTP - Códigos de status	10
4.1	Vantagens e desvantagens das políticas de substituição analisadas	46
4.2	Classificação das Políticas estudadas	48
4.3	Vantagens e desvantagens das estratégias usadas	51
6.1	Resumo das cargas de trabalho	72
6.2	Classes de respostas do protocolo HTTP e ICP	73

Lista de Abreviaturas

ANSI American National Standards Institute

BHR Byte Hit Ratio

BO2 Bolder - Universidade do Colorado

CGI Common Gateway Interface

CISC Centro de Informática de São Carlos

DNS Domain Name System

HR Hit Ratio

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

ICP Internet Cache Protocol

MeMoExP Média Móvel Exponencial no Período

MIME Multipurpose Internet Mail Extensions

PA Palo Alto - Califórnia

RTT Round Trip Time

SV Vale do Silício - Califórnia

SMP Symmetric Multiprocessor

TCP Transmission Control Protocol

TTL Time to Live

URL Uniform Resource Locator

WWW World Wide Web

Introdução

1.1 Contextualização

A World Wide Web (WWW) (*WWW* ou *Web*) pode ser vista como o maior e mais acessado sistema de informação distribuído da *Internet*. Essa característica leva a uma considerável sobrecarga na *Internet* e nos servidores, o que gera um tempo de resposta, às vezes, inaceitável a requisições feitas pelo usuário. Conseqüentemente, em função da evolução da *Web*, requerem-se redes com maior velocidade de transmissão de dados. Com o crescimento da utilização da *Internet* e o surgimento de novas aplicações, aumenta a cada dia a quantidade de dados a ser transmitida, o que torna os meios de comunicação disponíveis cada vez mais saturados. O problema é que grande parte desse tráfego é formado, muitas vezes, pela passagem de várias cópias de uma mesma informação. É justamente nesse ponto em que entra cache, que tem por finalidade diminuir tal problema ao fazer uso de dados previamente consultados e armazenados, ao invés de buscá-los na origem sempre que forem novamente requisitados.

O uso adequado de sistemas de *cache* na *Web* traz benefícios e proporciona o uso eficiente da largura de banda e a redução da latência para recuperação dos objetos. Embora herdem características dos sistemas de *caches* tradicionais, divergem em relação à operação de objetos, que têm tamanhos muito variáveis.

Os *caches* empregados na *Web* podem ser implementados em diversos lugares, sendo classificados de acordo com a sua localização: *cache* de cliente, implementado no *browser*;

cache de servidor, implementado no próprio servidor e *cache* de rede, implementados em pontos estratégicos da rede.

Os *caches* de cliente de rede operam como cliente e servidor. Esses *caches* recebem requisições de páginas endereçadas a um servidor e verifica se uma cópia da página requisitada encontra-se em seu estado de armazenamento. Caso afirmativo, o *cache* envia a página respondendo ao pedido do cliente (semelhante a um servidor). Se a página não estiver no *cache*, este atuará como cliente, enviando um pedido ao servidor original. Ao ser atendido, o *cache* responderá ao cliente servindo a página requisitada e guardará uma cópia da mesma.

As políticas de substituição decidem quais objetos irão permanecer no *cache* e quais serão retirados para ceder espaço a novos objetos. A escolha de uma política tem influência no consumo da largura de banda pois, sendo ela adequada, leva à diminuição de tráfego na rede e aumenta a quantidade de vezes que o documento é encontrado no *cache*.

1.2 Motivação e relevância

O rápido crescimento, aliado ao aumento da competição econômica e a proliferação de novas aplicações têm mudado a característica da *Internet* nos últimos anos. As empresas de hospedagem da *Web* pagam pela largura de banda utilizada e usam o *caching* como forma de redução de custos, o que faz com que todos os que participam na troca de mensagens se beneficiem com o *caching* [Krishnamurthy e Rexford, 2001]:

- Os usuários finais têm diversos benefícios com o *caching*, pois a latência percebida na obtenção de uma resposta é reduzida. Isso é muito bom, visto que uma fração significativa de cancelamentos que ocorrem durante uma sessão do usuário normalmente decorre por frustrações da não obtenção de respostas num intervalo de tempo razoável.
- A largura de banda desperdiçada pela transmissão repetida de dados é muito grande. Como o congestionamento é visto em vários pontos na *Internet*, a redução do tráfego ou sua mudança para a beira da rede e longe do *backbone*¹ é vista de forma benéfica.

Assim, somente os dados necessários atravessam a rede e existe largura de banda disponível para outros dados trafegarem. A distância do *cache* ao usuário também é um fator significativo na medição do benefício de um *cache* para o usuário. Um *cache* muito próximo do usuário poderia reduzir significativamente a latência percebida pelo usuário, se comparado com um *cache* mais perto do servidor de origem.

Em resumo, o uso de *cache* na *Web* faz com que os seguintes benefícios fiquem evidentes [Davison, 2001]:

¹Backbone - Espinha dorsal. Estrutura de nível mais alto em uma rede composta por várias sub-redes.

- Economia na largura de banda da rede;
- Menor latência;
- Redução de tráfego;
- Redução na carga dos servidores origem;
- Maior disponibilidade.

1.3 Objetivos

O objetivo principal desta dissertação foi dar continuidade e ampliar o estudo iniciado por Pinheiro [2001], que apresentou a dissertação “Avaliação de Políticas de Substituição de Objetos em Caches na Web”, sob orientação da Prof^a Dr^a Regina Helena Carlucci Santana, no programa de mestrado do ICMC - USP.

Para ampliação desse estudo, foi analisado o desempenho de diversas políticas de substituição de *caches* na *Web* existentes e apresentada uma nova política de substituição, denominada Média Móvel Exponencial no Período (MeMoExp).

Para tanto, foi realizada inicialmente uma atualização e expansão da revisão bibliográfica apresentada por Pinheiro [2001].

Uma próxima etapa da pesquisa se concentrou na escolha de métricas adequadas para avaliação de *caches* na *Web*. Para a realização desta etapa, foi necessário um estudo sobre as métricas existentes e a definição de novas métricas.

As métricas definidas na etapa anterior foram utilizadas para avaliar diferentes políticas de substituição. Para que essa avaliação fosse possível, foi necessário fazer uma adaptação no simulador utilizado por Pinheiro [2001].

1.4 Estrutura

Este capítulo apresentou o contexto no qual se insere este trabalho, a motivação para o seu desenvolvimento e os objetivos a serem alcançados.

O Capítulo 2 introduz conceitos relacionados à *Web* que cobrem desde aspectos históricos até detalhes de alguns componentes básicos da *Web*.

O Capítulo 3 apresenta uma visão geral sobre *cache* e o seu uso na *Web*, enfocando o uso de armazenamento em *cache* na *Web* para acelerar a disponibilidade de conteúdo e melhorar o uso do sistema de comunicação.

São apresentadas, no Capítulo 4, as políticas de substituição existentes.

O Capítulo 5 apresenta alguns conceitos básicos de probabilidade e estatística, que ajudam a avaliar adequadamente os resultados de uma simulação. Também são apresentados os principais conceitos de simulação e mostrados os parâmetros e o ambiente de simulação que será utilizados. O conceito matemático a respeito de médias móveis será mostrado, para que a política MeMoExp proposta no Capítulo 6 possa ser entendida.

É no Capítulo 6 que estão os resultados obtidos com a utilização da nova política.

O Capítulo 7 apresenta as conclusões finais e trabalhos futuros.

World-Wide Web

2.1 Considerações Iniciais

Esta pesquisa está voltada à avaliação de políticas de substituição de objetos em *caches* na *Web*. Assim, torna-se fundamental fazer uma revisão sobre os conceitos gerais envolvidos com a *Web*. Tal revisão é apresentada neste capítulo que cobre desde aspectos históricos até detalhes de alguns componentes básicos da *Web*.

2.2 Histórico

WWW é o formato do método mais popular para compartilhar informações na Internet. A WWW foi desenvolvida em 1991 no CERN (*Conseil Européen pour la Recherche Nucleaire*) ou (*European Center for Nuclear Research*), mas tudo começou em 1989 quando Tim Berners-Lee propôs um sistema baseado em hipertexto, no qual apontar para uma palavra ou frase destacada levava um usuário para uma nova página na mesma máquina ou para uma máquina remota na rede. A idéia principal de tal sistema era encontrar uma forma rápida de compartilhar as pesquisas realizadas pelos físicos do CERN com outros pesquisadores ao redor do mundo.

O termo *World Wide Web* significa “teia de alcance mundial” e é também conhecido por WWW, *W3* ou simplesmente *Web*. A *Web* é uma aplicação da *Internet* que combina

hipertexto e multimídia. Por isso, diz-se que ela é uma aplicação hiperfídia. Textos, imagens, animações, sons e vídeos podem ser exibidos em “páginas”, chamadas de “páginas *Web*”. Para deslocar-se a outro texto em outra página, basta selecionar-se com o *mouse* uma palavra ou uma imagem da página e pressionar-se o botão de acionamento. As palavras ou imagens que fazem as ligações entre as páginas são chamadas de ligações ou ponteiros (*links*). Assim, o hipertexto interliga milhares de páginas *Web* através de ponteiros e a *Web* pode ser definida como uma vasta coleção de páginas interligadas por hipertexto que utiliza os mais diversos recursos de multimídia.

2.2.1 Hipertexto

O termo hipertexto foi criado em 1965 para diferenciar um texto normal e linear de um texto não-linear, sendo que um texto não-linear pode ser definido como um texto que contém referências a informações extras. O exemplo mais conhecido de hipertexto atualmente é uma página da *Web*, onde seguem-se as referências entre as páginas sem necessariamente seguir um roteiro linear. O hipertexto é um texto não-linear, sem ponto fixo de entrada e de saída, sem uma hierarquia pré-determinada, sempre expansível e literalmente sem limites, que tem como componentes básicos as ligações, os nós e as âncoras.

Pierre Levy [Levy, 1993] definiu hipertexto como um conjunto de nós ligados por conexões. Os nós podem ser palavras, páginas, imagens, seqüências sonoras ou documentos complexos. Os itens de informação não são obrigatoriamente ligados linearmente. Desta forma, cada nó pode conter uma rede inteira.

2.3 Protocolos da Web

Para visualizar um documento de hipertexto na *Internet*, é necessário um protocolo para fazer a comunicação entre um computador e o servidor. Um protocolo é uma convenção ou linguagem que permite a intercomunicação entre dois *softwares* e *hardwares* através da rede e define a sintaxe e a semântica das mensagens trocadas entre emissores e receptores [Krishnamurthy e Rexford, 2001].

2.3.1 Protocolo de Comunicação - HTTP

Os protocolos de rede normalmente são desenvolvidos em camadas, onde cada uma trata de um aspecto específico da comunicação. O conjunto de protocolos para a *Internet* consiste em quatro camadas principais, mostradas na Tabela 2.1.

Camada	Protocolo
Aplicação Contém os protocolos de alto nível que são diretamente utilizados pelos programas que interagem com os usuários. UBI*: Mensagem	DNS – <i>Domain Name System</i> – protocolo que controla a tradução de nomes de hosts para endereços IP e vice-versa.
	Telnet – Programa usado para conectar um usuário a um computador remoto pela rede Internet.
	FTP – <i>File Transfer Protocol</i> – protocolo de transferência remota de arquivos entre um computador servidor e um cliente.
	SMTP – <i>Simple Mail Transfer Protocol</i> – protocolo simples de transferência de correspondência – usado na transferência de e-mail.
	SNMP – <i>Simple Network Management Protocol</i> – protocolo usado para monitorar e controlar serviços e dispositivos de uma rede TCP/IP.
	HTTP – <i>HyperText Transfer Protocol</i> – protocolo de transferência das páginas WWW pela rede Internet, requerendo um programa cliente na origem e no destino da conexão.
Transporte Controla a comunicação entre as aplicações envolvidas em uma comunicação inter-redes. UBI*: Segmento	TCP – Protocolo orientado à conexão. Assegura que os dados transmitidos pela rede que alcançarão seu destino, detectando e reenviando pacotes perdidos, adulterados ou duplicados, além de oferecer a correção de erro. Estabelece um circuito virtual durante a conexão.
	UDP – <i>User Datagram Protocol</i> (Protocolo de datagramas) – Protocolo não orientado à conexão; não confiável. Não possui controle de fluxo e nem critérios de recuperação de erros. O UDP apenas multiplexa e demultiplexa datagramas IP. Ambos os protocolos podem servir a mais de uma aplicação simultaneamente.
Rede Determina a interconexão entre as redes da Internet. Ela é a responsável pelo roteamento dos pacotes entre os hosts, com a função de encontrar o caminho mais curto e confiável entre os computadores. UBI*: Datagrama	IP – <i>Internet Protocol</i> – o protocolo comum na rede de computadores Internet. Fornece os endereços que os roteadores necessitam para transferir pacotes através de redes até o seu destino. Para isso, coordena a remessa de pacotes individuais de um host para outro, com base no endereço IP do host de destino.
	ICMP – <i>Internet Control Message Protocol</i> – protocolo utilizado na rede Internet que define mensagens de erro e o funcionamento de pacotes de teste.
	ARP – <i>Address Resolution Protocol</i> – protocolo que relaciona endereços IP a endereços físicos de uma interface de rede, fazendo a conversão de endereço físico em endereço lógico.
	RARP – <i>Reverse Address Resolution Protocol</i> – protocolo de função inversa ao ARP. Utilizado para conversão de endereços lógicos em endereços físicos.
Interface Função principal: interface do modelo TCP/IP com os vários tipos de redes UBI*: Frame	ATM Ethernet X.25

Tabela 2.1: Estrutura de Protocolos em Camadas

A *Web* é um sistema distribuído de informação onde os dados não estão guardados num computador central, mas espalhados em servidores em todo o mundo. Para funcionar, a *Web* faz uso de um protocolo de comunicação e grande parte de seus documentos são páginas escritas em uma linguagem chamada HyperText Markup Language (HTML), que fornece aos navegadores instruções de como a página deve ser exibida.

O protocolo utilizado é o HyperText Transfer Protocol (HTTP), definido no nível de aplicação e usado para executar todas as comunicações entre os navegadores e o servidor da *Web*, cujo objetivo é especificar a ação a ser tomada por eles em resposta a um determinado comando.

Pode-se dizer que o protocolo HTTP evoluiu juntamente com a *Web* e pode ser dividido em três fases: [Krishnamurthy e Rexford, 2001]:

- HTTP 0.9 até HTTP 1.0: protocolo simples para transferência de dados não formatados na *Internet*.
- HTTP 1.0 (RFC 1945 - Maio 1996):
 - introdução de formatação de dados do tipo Multipurpose Internet Mail Extensions (MIME)
 - meta-informação sobre os dados transferidos
 - modificações na semântica de requisições/respostas
 - não levava em consideração o efeito de *proxies* hierárquicos, cacheamento, a necessidade de conexões persistentes ou *hosts* virtuais
- HTTP 1.1 (RFC 2068 - Janeiro 1997/ RFC 2616 - Junho 1999)

O *caching* começou a ser um desafio desde o surgimento do HTTP 1.0 [Krishnamurthy e Rexford, 2001].

A *Web* é basicamente um sistema cliente-servidor [Tanenbaum, 1997]. Do ponto de vista do usuário (lado do cliente), a *Web* é uma grande coleção de objetos (páginas) que podem conter ponteiros para outras páginas. Já do lado do servidor, ele é um processo que envia resposta a solicitações feitas pelos clientes, depois de estabelecida uma conexão.

2.4 Cliente/Servidor

Um servidor *web* é um computador permanentemente conectado à rede que fornece dados e recursos através dela. O servidor atende requisições de arquivos ou serviços de outros computadores ligados à rede. O servidor *Web* é um programa que executa em um computador

remoto, cuja finalidade é fornecer os objetos solicitados pelos clientes. Para funcionar como servidor, um computador deve possuir um software servidor HTTP próprio para essa função, que simplesmente transmite as páginas para o browser do cliente, sem nenhuma alteração.

Após o recebimento de uma requisição, o servidor *Web* responde, utilizando uma estrutura de três partes:

- Código de retorno: Código seguido de uma mensagem (geralmente OK se a operação de requisição do arquivo foi bem sucedida, ou uma descrição do erro, quando for o caso). É retornado também um texto identificando a versão do protocolo HTTP utilizado. A Tabela 2.2 mostra os códigos de *status* geralmente retornados por um servidor Web.

Tabela 2.2: Códigos de status geralmente retornados por um servidor Web

Código	Descrição
200 OK	Um objeto válido foi encontrado no servidor e retornado ao cliente.
202 Accepted	A solicitação foi aceita para processamento, mas essa tarefa ainda não foi terminada.
204 (No content)	O servidor atendeu à solicitação, mas não há novas informações a serem retornadas.
302 Moved Temporarily	O recurso solicitado utiliza temporariamente uma URL diferente
304 Not-Modified	O cliente executou uma solicitação condicional com GET e o acesso foi permitido, mas o objeto não foi modificado desde a data e hora especificados no campo If-Modified-Since.
400 Bad Request	O pedido não pôde ser entendido pelo servidor devido à sintaxe inválida.
403 Forbidden	O servidor entendeu a requisição, mas recusou completá-la.
404 Not Found	O servidor não achou nada compatível com a URL solicitada. Este código é comumente usado quando o servidor não deseja revelar exatamente por que o pedido foi recusado ou quando nenhuma outra resposta é aplicável.
500 Internal Server Error	O servidor encontrou uma condição inesperada que impediu o atendimento da solicitação.
504 Gateway timeout	O servidor não recebeu uma resposta do gateway ou do servidor superior dentro do prazo permitido, quando feito um acesso para tentar completar a solicitação.
505 HTTP Version Not Supported	Versão HTTP não suportada pelo servidor.

O primeiro dígito do código define a classe da resposta. Existem 5 valores possíveis para o primeiro dígito, conforme especificado na Tabela 2.3 [Fielding et al., 1999].

- Cabeçalho: Contém diversas informações (data da última modificação do arquivo, tamanho, tipo) sobre o arquivo sendo enviado, como tamanho e informações sobre o próprio servidor. Essas informações podem ser utilizadas pelo programa cliente para decidir o que fazer com determinado arquivo.
- Arquivo: Após o cabeçalho, uma linha em branco é enviada pelo servidor indicando que, a partir daquele ponto, tudo o que for mandado pelo mesmo será parte do arquivo requisitado. Um programa cliente pode controlar o progresso da transmissão

Tabela 2.3: HTTP - Códigos de status

Código	Informação	Descrição
1XX	Informativo	Requisição recebida, processamento em andamento.
2XX	Sucesso	A ação foi recebida com sucesso, entendida e aceita.
3XX	Redirecionamento	O Cliente deve executar outras ações para que a requisição seja completada (por exemplo, requisitar uma outra URI).
4XX	Erro do Cliente	A requisição contém erro de sintaxe ou não pode ser atendida.
5XX	Erro do Servidor	O Servidor não conseguiu atender a uma requisição aparentemente válida.

monitorando quanto já foi transmitido e comparando essa informação com o tamanho informado no cabeçalho.

2.5 Desempenho em Servidores Web

Quando um servidor *Web* está sobrecarregado, seu desempenho fica comprometido, fato que resulta em requisições com altos tempos de respostas e inúmeras conexões recusadas. Assim, é necessário saber se um servidor *Web* é capaz de suportar um aumento de carga e preservar um tempo de resposta adequado. Caso isso não seja possível, deve ser necessário medir até que ponto a capacidade estará saturada. Isso é importante de ser analisado pois pode dar a medida de quanto dinheiro pode ser perdido diariamente se o servidor *Web* saturar quando a carga aumentar.

O desempenho do servidor pode ser acompanhado através de ferramentas que analisam os *logs*¹ e permitem identificar endereço IP, data e hora, Uniform Resource Locator (URL), *status* da requisição. Com isso, pode ser possível monitorar dados como: porcentagem de acertos no *cache*, *sites* mais acessados, clientes que mais acessam, requisições satisfeitas diretamente, requisições feitas a servidores da hierarquia e tipos de documentos armazenados.

¹Logs são registros de atividades gerados por programas de computador.

2.5.1 Requisitos de desempenho

Parte do planejamento de uma rede envolve determinar um nível realista de desempenho. São dois os fatores que afetam o desempenho que um usuário vê [Tanenbaum, 1997]:

- Quanto mais segmentada for a rede, mais roteadores são necessários para fazer o tráfego fluir. Ou seja, quanto mais *hosts*² forem colocados num segmento de rede, mais baixo será o desempenho.
- Um outro fator diz respeito ao tipo de tráfego que os *hosts* geram e a quantidade real de tempo que os mesmos gastam para se comunicar. No caso do cliente se comunicar constantemente é preciso restringir o número de *hosts* colocados na rede.

2.5.2 Uso de Caches

Ao acessar uma página da *Web*, uma requisição parte da máquina solicitante até o servidor; só então os dados são transmitidos de volta para essa máquina. Como muitas vezes a distância entre o servidor e a máquina é muito grande e a qualidade das linhas de transmissão irregulares, esse processo acaba por tornar-se bastante lento.

Além disso, grande parte dos dados requisitados é estática: não muda com o tempo. Os logotipos que as empresas colocam em suas páginas, por exemplo, não tendem a mudar embora sejam, muitas vezes, bastante grandes, o que gera um enorme desperdício de recursos da rede, além de tempo.

Uma solução encontrada para melhorar o desempenho é o uso de *caches*. Isto é, sempre que é feita uma requisição de algum objeto da *Internet*, o servidor consulta o *cache* para verificar se este objeto não foi requisitado previamente. Se ele foi, então o servidor pode responder à requisição utilizando sua própria cópia local do objeto. Isso acelera significativamente as operações na *Internet*, visto que grande parte dos objetos acaba trafegando apenas localmente.

O servidor verifica se sua cópia está atualizada com o objeto original. Caso não esteja, atualiza sua cópia. Naturalmente, um servidor não pode guardar em *cache* todos os objetos acessados para sempre, pois isso tenderia rapidamente à saturação. Uma solução simples é manter apenas os arquivos utilizados mais recentemente. Isso garante, de uma forma indireta, que os objetos mais freqüentemente utilizados sempre estejam no *cache*.

A teoria sobre *caches* na *Web* será estudada com maiores detalhes no Capítulo 3.

²host é o nome dado ao computador principal de uma rede que comanda e controla a ação de outros computadores

2.6 Aplicações Web

Essencialmente, uma aplicação *Web* realiza as seguintes tarefas:

- Disponibiliza uma interface para a entrada de dados;
- Transmite os dados informados pelo usuário para o servidor *Web*;
- Recebe os dados enviados utilizando algum conjunto de *middlewares*³
- Realiza o processamento no servidor;
- Transmite os resultados de volta ao cliente;
- Realiza o processamento dos dados enviados no cliente, mostrando-os ao usuário.

A Figura 2.1 mostra uma representação de aplicações *Web*, considerando intervalos de complexidade (estática x dinâmica) e graus de orientação (orientados a documentos e orientados a aplicações). De acordo com a Figura 2.1, os *sites* da *Web* podem ser agrupados em [Powell et al., 1998]:

- Estático: Quando em sua forma mais simples, um site da *Web* é uma coleção de páginas estáticas, que podem ser documentos ou informações escritas e publicadas no formato HTML. Do ponto de vista da funcionalidade, a mesma é dada basicamente pelos *links* que permitem a navegação, sejam *links* previstos pelos controles principais e de navegação, ou pelos *links* estruturais (como um índice, por exemplo) e semânticos. Nesses tipos de *sites*, a ênfase no projeto (incluindo o esboço) é dada pela organização da estrutura e do conteúdo e pela facilidade de navegação. Entretanto, podem existir problemas de usabilidade, eficiência, confiabilidade e manutenibilidade.
- Estático com Formulários de Entrada: além do oferecido pelos *sites* estáticos, existe um nível básico de interação implementado por meio de formulários de entrada. Isso favorece a usabilidade do *site* ao permitir mecanismos de realimentação ao cliente, tais como questionários, livros de visita, comentários e sugestões, que favorecem a comunicação *on-line*.
- Estático com Acesso a Dados Dinâmicos: Além das características comentadas, o usuário pode ter acesso a dados armazenados em bases de dados remotas através de consultas e buscas. Os dados retornados são gerados dinamicamente e apresentados no formato HTML.

³Middleware é uma camada de *software* entre a aplicação e a infraestrutura de rede. Uma das suas finalidades é isolar o desenvolvedor das complexidades da rede.

Figura 2.1: Intervalos de Complexidade e níveis de orientações das aplicações Web

- **Dinâmicos:** Além de todas as características comentadas anteriormente, criar um *site* dinâmico deve-se à necessidade de um mesmo *site* ter que prover requerimentos semelhantes, ainda que personalizados em consideração ao conteúdo das páginas para cada instância de usuário. Ou ainda quando, por aspectos de compatibilidade tecnológica, necessita-se construir *sites* dinamicamente em conformidade com o ambiente do navegador do usuário. Para isso, os documentos estáticos devem ser movidos dinamicamente embora, do lado do cliente, não provenham nenhuma interatividade.
- **Aplicação de Software baseado na Web:** Esse tipo pode ser o mais completo e com maior orientação a aplicação, como visto na Figura 2.1. Esse tipo de *site* pode ser

um sistema de, por exemplo, educação a distância, que fornecem funcionalidades que mais parecem uma implementação cliente/servidor tradicional do que um site estático. No entanto, todas as características dos tipos anteriormente expostos podem estar incorporadas à aplicação.

Como mostrado anteriormente, os artefatos *Web* podem tornar-se muito complexos. E, equivalente ao desenvolvimento de *software* tradicional, o desenvolvimento e evolução de aplicações centralizadas na *Web* pode ser um processo com muitos desafios, principalmente se as mesmas são implementadas com estratégias *ad hoc*⁴.

2.7 Servidor Proxy

Um servidor *proxy* é um programa de aplicação que recebe requisições de objetos de um conjunto de clientes, repassa essas requisições ao servidor apropriado e envia o objeto requisitado de volta ao cliente [Fielding et al., 1999].

Como ele faz isso de forma transparente ao cliente que fez a requisição, pode-se dizer que um *proxy* é um computador que se faz passar por um outro computador. Os *proxies*, além de funcionar como tradutores para protocolos desconhecidos [Tanenbaum, 1997], podem executar as funções de autenticação, filtragem de conteúdo e cache.

2.8 Considerações Finais

Neste capítulo foi apresentada uma breve revisão sobre a *Web*, serviço disponível através da Internet que permite a transmissão e visualização de documentos multimídia. Foram descritos alguns dos componentes da *Web*: o cliente, o servidor e o *proxy*. No próximo capítulo será detalhado o *proxy*, agindo como um repositório local de objetos, isto é, fazendo *caching* de objetos. Será mostrada também uma revisão bibliográfica geral sobre *cache* e *cache* na *Web*.

⁴ad hoc - não leva em consideração teorias ou métodos científicos

Cache na Web

3.1 Considerações Iniciais

Neste capítulo são apresentados os conceitos gerais sobre *cache* e os conceitos relacionados a caches na Web. Além da revisão dos conceitos básicos, será dada ênfase às principais diferenças entre caches tradicionais e caches na web.

O capítulo enfoca o uso de armazenamento em *cache* na web para acelerar a disponibilidade de conteúdo e melhorar o uso do sistema de comunicação.

3.2 Histórico

De acordo com as pesquisas na área [Intel, 2003], o tempo médio que um usuário passa esperando a recuperação de uma página em um Web site é menor do que dez segundos. Isso é muito pouco tempo e o desempenho da rede, de velocidade a consistência, pode ser a diferença.

O atraso na busca por determinado recurso, do ponto de vista do cliente, pode ser causado pelos seguintes fatores [Krishnamurthy e Rexford, 2001]:

- Conectividade de rede do usuário ao seu provedor e conexão entre o provedor e a Internet.

- Tempo de pesquisa do Domain Name System (DNS) para localizar o servidor a ser contatado.
- Congestionamento na rede e a largura de banda disponível no percurso entre o usuário e o servidor de origem
- Carga sobre o servidor de origem
- Tempo para a geração de resposta
- Tempo para a apresentação da resposta pelo *browser*

Os fatores apresentados anteriormente traçam o percurso de um pedido enviado de um *browser* para o servidor de origem, até que a resposta seja apresentada de volta no *browser* [Krishnamurthy e Rexford, 2001].

Para este trabalho, é interessante analisar o impacto do *caching* sobre alguns dos fatores apresentados anteriormente [Krishnamurthy e Rexford, 2001]:

- Conectividade da rede: A conectividade, velocidade de conexão entre o provedor do usuário e a Internet, entre um cliente e um provedor está sujeita a menos variabilidade do que os atrasos na Internet. Com o conteúdo trazido para mais perto do provedor de serviços, pode haver um certo grau de previsibilidade na latência experimentada pelo usuário. Se a conectividade for alta, as respostas são trazidas rapidamente para a extremidade da rede do provedor. Caso contrário, a mudança da resposta para a extremidade da rede do provedor pode dominar a latência geral experimentada pelo usuário. Vale lembrar também que se a conexão entre o provedor do usuário e o servidor de origem cair no momento do pedido, um cache localizado no provedor pode ser capaz de atender ao pedido.
- Atraso relacionado ao DNS: aumentar o valor do tempo de vida (Time to Live (TTL)) das pesquisas de DNS pode ajudar a reduzir a latência geral, se for assumido que os vínculos entre nomes e endereços não mudem com frequência. Algumas soluções de *caching* podem aumentar o tempo de pesquisa geral do DNS, ao localizar o conteúdo mais próximo do usuário. Estar mais próximo significa que a distância da rede percorrida pela resposta da Web é significativamente pequena. O cache de DNS de um *proxy* com *caching* provavelmente terá domínios acessados com frequência e poderá ajudar a diminuir futuros atrasos relacionados ao DNS.
- Congestionamento na rede e largura de banda disponível: a redução do número de hops pode melhorar o *throughput* geral entre o cliente e o servidor onde está o conteúdo. Se o Transmission Control Protocol (TCP) for o mecanismo de transporte de suporte para

o HTTP, então as reduções no tempo de ida e volta Round Trip Time (RTT) melhoram o *throughput*. O *throughput* do TCP é inversamente proporcional ao RTT. Vale citar que menos hops reduzem a probabilidade de que a transferência esteja subordinada às inconstâncias de atrasos ocorridos no congestionamento ponta a ponta. Quanto maior o número de pacotes na rede, maior a probabilidade de um congestionamento, o que pode levar a perdas de pacotes. Quando isso ocorre, os mesmos precisam ser retransmitidos. Se a utilização da rede for menor devido ao caching, o restante dos dados na rede pode ser transmitido mais rapidamente. O emissor do TCP pode aumentar o tamanho da janela de congestionamento para usar a rede em sua capacidade plena.

- Carga sobre o servidor de origem: a carga no servidor de origem pode ser reduzida se o *caching* dos recursos solicitados com mais frequência resultar em menos pedidos para o servidor de origem. O servidor de origem transfere a tarefa de atender aos pedidos para diversos sites vizinhos - os *proxies* com *caching*. Da mesma forma, o servidor de origem pode manter conexões persistentes com menos clientes de entrada o que faz com que tais conexões possam permanecer abertas por mais tempo. Isso faz com que o tempo para se gerar respostas redundantes seja eliminado e o servidor tenha mais tempo para realizar outras tarefas.
- Tempo para gerar a resposta: O tempo para gerar resposta a um pedido pode ser reduzido pois, se menos recursos do servidor forem gastos para tratar os pedidos recebidos, uma quantidade maior de recursos pode ser dedicada à geração da resposta.
- Apresentação da resposta pelo *browser*: Um *browser* precisa analisar e exibir a resposta. O tempo para se apresentar o recurso pelo *browser* não pode ser reduzido pelo *caching*, exceto se a versão inteira da página estiver na memória volátil do *browser* ou se a mesma estiver armazenada no *cache* do usuário. Nesta dissertação, não será considerado o tópico dirigido a *cache* no cliente.

Grandes investimentos são feitos por prestadores e provedores de serviços com o objetivo de aumentar a largura de banda disponível para a comunicação na Internet. Entretanto, sozinho, o aumento dessa largura não pode solucionar o problema da latência da rede ou acelerar servidores lentos. Surge o armazenamento em *cache*, que move o conteúdo para mais perto dos usuários e traz benefícios não apenas para o usuário final, mas também para provedores de serviços de Internet e provedores de conteúdo. Além disso, em um futuro em que cada negócio será um *e-Business*, isso pode oferecer a qualquer *site* uma grande vantagem competitiva.

3.3 Caches Tradicionais

O uso de memória *cache* visa obter uma velocidade de acesso à memória próxima da velocidade das memórias mais rápidas e, ao mesmo tempo, disponibilizar no sistema uma memória de grande capacidade, a um custo equivalente ao das memórias de semicondutor mais baratas [Stallings, 2003]. A Figura 3.1 ilustra o conceito de memória cache.

Figura 3.1: Memória cache tradicional

Embora exista uma semelhança funcional, os sistemas de *cache* tradicionais e de *cache* na Web têm características um pouco diferentes. Enquanto nos *caches* tradicionais os dados são movidos em blocos de mesmo tamanho, isto é, o número de bytes encontrados no *cache* em relação ao número de bytes requisitados é exatamente a taxa de hits, o mesmo não ocorre nos caches da Web porque os documentos são transmitidos e armazenados na sua forma integral pois o conceito de bloco não existe.

O armazenamento em *cache* é uma tecnologia comum em vários aplicativos. Diversos dispositivos de hardware armazenam em *cache* as instruções e os dados que são usados freqüentemente para acelerar as tarefas de processamento. Por exemplo, os dados usados com freqüência pela CPU são armazenados em memória extremamente rápida, às vezes diretamente no chip da CPU, reduzindo a necessidade da CPU ler dados de uma unidade de disco mais lenta.

Para armazenamento em *cache* da web, o mesmo conceito é aplicado, embora de maneira mais ampla, com o uso de um servidor ou dispositivo especializado. O conteúdo da web é colocado próximo aos usuários na forma de um *cache* de rede, reduzindo o número de saltos de roteamento necessários para recuperar o conteúdo de um site remoto. Em outras palavras, os documentos requisitados são copiados em posições mais próximas do usuário com o intuito de diminuir o tempo do próximo acesso ao documento.

3.4 Cache na Web

Utilizar *caches* na web significa armazenar cópias de documentos muito acessados em servidores mais próximos aos clientes, o que diminui tanto a carga na rede como as requisições feitas a servidores que possuem documentos muito populares. Como resultado, o usuário experimenta uma diminuição do tempo de resposta a requisições feitas [Malpani et al., 1995], [Abrans et al., 1995].

A Figura 3.2 mostra como um *cache* de Web funciona: as informações acessadas frequentemente são armazenadas onde um usuário possa ter acesso a elas de forma rápida e fácil. O *cache* monitora as requisições da Web, acessa os objetos no servidor remoto e os armazena localmente para uso futuro. Os usuários que solicitam o mesmo conteúdo ou aplicações em seguida são atendidos pelo *cache* local e não pelo servidor origem.

Figura 3.2: Funcionamento de um cache para Web

No caso de uma versão atualizada do documento ser achada no *cache*, nenhuma conexão ao servidor remoto é necessária, conforme mostra a Figura 3.3.

Figura 3.3: Uso de cache

O uso de *cache* elimina, assim, a necessidade de múltiplas e demoradas idas e voltas pela Internet por parte do pedido do usuário até o servidor origem.

O fluxograma a seguir ilustra o que acontece quando um usuário faz uma requisição de uma página da web. As linhas mais grossas representam normalmente conexões locais de

velocidade alta entre o cliente e o cache, enquanto as linhas mais finas representam conexões mais lentas.

Figura 3.4: Fluxograma - Requisição de uma página web

Assim como os *caches* de memória e disco, que mantêm os dados mais acessados em uma área específica para posterior recuperação, o *cache* Web armazena os objetos (páginas HTML, imagens, arquivos, etc) acessados na Internet. Utilizar *caches* na Web significa colocar cópias de documentos muito acessados em servidores mais próximos aos clientes, diminuindo assim a carga de requisições feitas à rede e a servidores de documentos muito populares. Dessa forma, como resultado final, o usuário dos serviços prestados pela Web experimenta uma diminuição do tempo de resposta a requisições. Vários parâmetros estão relacionados ao desempenho de caches para Web. Dentre os principais, pode-se citar o tamanho do espaço de armazenamento, pois em *caches* Web trabalha-se com objetos maiores, a capacidade de processamento do processador no servidor de *cache* e a política de substituição utilizada.

As políticas tradicionais para uso de *cache* presentes em sistemas de arquivos, não apresentam bom desempenho quando usadas para objetos Web pelas seguintes razões [Markatos, 1996]: a granulosidade, medida da razão entre a quantidade de computação realizada e a quantidade de comunicação necessária, é diferente, pois as políticas tradicionais colocam blocos de arquivos no *cache*, enquanto o *cache* de objetos Web deve guardar todo o documento; além disso, os documentos Web são acessados em um modo somente-leitura, diferente dos

sistemas de arquivos que têm que lidar com tráfego somente-leitura e de leitura-escrita, o que introduz uma complexidade adicional para esses sistemas. Partindo do mesmo princípio, isto é, o aumento da performance, o *cache* na Web é uma técnica utilizada para reduzir na rede o tráfego originado da Internet e diminuir o tempo de resposta para os usuários finais.

A Tabela 3.5 mostra um resumo com as principais diferenças apresentadas por um *cache* tradicional e *cache* para a Web.

Característica	Cache tradicional (Memória)	Cache na Web
Tipos de objetos	Blocos, linhas de dados ou instruções	Documentos, imagens, vídeos, arquivos executáveis
Tamanho dos objetos	Dividido em pequenas unidades de mesmo tamanho, denominadas linhas ou blocos	Apresenta grande variabilidade
Tamanho da cache	Ordem de kilobytes (KB)	Ordem de gigabytes (GB)
Políticas de substituição	Mais simples	Mais complexas
Atualização	Existem operações de escrita na cache: a memória deve ser atualizada para não ficar inconsistente	As cópias nas caches podem ser atualizadas somente quando o objeto original for atualizado

Figura 3.5: Diferenças entre Caches Tradicionais e Caches Web

3.4.1 Dispositivo de Armazenamento em Cache

Sem um dispositivo de armazenamento em *cache*, as solicitações de conteúdo de um browser e o conteúdo entregue pelo servidor de origem devem constantemente fazer a mesma viagem longa, do computador que solicita para o computador que tem o conteúdo, e a volta.

Quando um dispositivo de armazenamento é usado, o processo é bem mais eficiente, pois o conteúdo acessado frequentemente não tem que percorrer o caminho, por vezes longo, do servidor de origem ao destino constantemente. Quanto maior a frequência com que um *cache* pode atender às solicitações dos usuários, maior a taxa de acertos e melhor o desempenho percebido pelos usuários.

O requisito mais importante de uma solução de armazenamento em *cache* é a capacidade de fornecer um desempenho otimizado. O desempenho do *cache* tem duas partes:

- Capacidade operacional: é resolvida pela arquitetura e pela implementação do dispositivo de armazenamento em *cache*. Juntamente com a capacidade bruta do *cache*, as questões de arquitetura incluem o modo como o servidor usa diversas linhas de execução e da eficiência com que realiza o balanceamento de carga em uma implantação de vários servidores de *cache*.
- Agilidade para atender às solicitações do usuário: determinada pelas diversas técnicas que o servidor de *cache* usa para maximizar a taxa de acertos, onde se inclui a estrutura de hierarquias e as otimizações de conteúdo. A taxa de acertos do *cache* é uma função que possui fatores como o tamanho do *cache* e a carga sobre ele.

Há dois tipos de modelos de armazenamento em *cache* da Web:

- Modelo *edge-services*: as empresas aceitam os serviços de um fornecedor terceirizado para armazenar seu conteúdo em *cache*.
- Modelo aberto: os provedores de serviços instalam seu próprio *cache* em seus servidores e podem oferecer o armazenamento como um serviço de valor agregado aos seus clientes.

O primeiro modelo tem sérias desvantagens, pois o provedor de acesso à Internet não controla a infra-estrutura e esse tipo de armazenamento em *cache* não armazena necessariamente o conteúdo visualizado com mais frequência.

O modelo aberto tem diversas vantagens: possuindo o hardware de *cache*, o provedor de acesso à Internet investe em seu próprio destino, não no de uma empresa de terceiros, podendo lucrar com esse serviço. Um sistema de dispositivo de armazenamento em *cache* projetado adequadamente armazena automaticamente os sites que os usuários mais acessam.

3.4.2 Modelos de Implementação de um Cache

Há diversas abordagens, ou modelos, para a implementação de um *cache*. O modelo escolhido depende de algumas características tais como, onde o *cache* é implementado, a finalidade principal do *cache* e a natureza do tráfego. O caching na web, como mostrado na Figura 3.6 pode ser executado por diferentes componentes da rede e ser classificado de acordo com os seguintes tipos [Barish e Obraczka, 2000]:

Figura 3.6: Arquitetura de caches na web

- **Browser cache:** A maioria dos *browsers* possui seu próprio sistema para *caching* (cache local), pois é bastante provável que um usuário acesse as mesmas páginas frequentemente, ou muitas vezes num mesmo dia. O tamanho desse cache é pequeno e a maioria dos algoritmos de substituição utilizados ou é FIFO ou LRU, mais simples de serem implementados, porém de eficiência duvidosa. Considerando que é o cache mais próximo do usuário final, pode prover um tempo de resposta mais curto se o objeto pedido já estiver armazenado. No entanto, os conteúdos armazenados nesse cache local não são compartilhados entre usuários de uma rede local.
- **Proxy cache:** Um servidor *proxy* está normalmente situado à extremidade de uma rede, de modo a servir um grande número de usuários internos[Barish e Obraczka, 2000]. Um servidor *proxy* intercepta pedidos de HTTP de clientes e, se achar o objeto pedido em seu cache, devolve o objeto ao usuário. Se o objeto não é achado, o cache vai para o servidor do objeto, o servidor origem e, em nome do usuário, adquire o objeto, deposita ou não o conteúdo em seu cache e finalmente retorna o objeto para o usuário. O uso de proxy caches resulta em economia da largura de banda, melhora o tempo de resposta e aumenta a disponibilidade de dados e objetos estáticos além de minimizar o espaço utilizado em disco, uma vez que é feita apenas uma cópia do objeto para vários usuários.
- **Proxy cache transparente:** É assim chamado porque trabalha interceptando o tráfego da rede de forma transparente ao browser. São usados especialmente por provedores de acesso à Internet, porque não é necessário nenhuma configuração no *browser* do usuário.
- **Proxy Avançado:** o *cache* de proxy avançado é definido por sua natureza reativa. Com ele, as requisições do usuário passam pelo *cache* no caminho para o servidor da web de destino. Se o *cache* contiver o documento solicitado, o fornecerá diretamente. Caso contrário, o servidor agirá como um *proxy*.

- Proxy Reverso: o sistema de armazenamento em *cache* fica à frente de um ou mais servidores da web, interceptando solicitações e agindo como um *proxy*. Os documentos armazenados em *cache* são fornecidos a uma maior velocidade, enquanto os que não estiverem em *cache* - normalmente conteúdo dinâmico e outros objetos de curta duração - são solicitados aos servidores da web de origem quando necessário.
- Armazenamento em cache Transparente: os caches de proxy avançados podem ser configurados como transparentes ou não transparentes. Um *cache* transparente fica no fluxo da rede e funciona de forma invisível para um *browser*. Os benefícios do armazenamento em *cache* são fornecidos automaticamente aos clientes, sem a necessidade de reconfiguração dos *browsers*.

3.4.3 Armazenamento em Cache

A escolha da combinação certa de software e hardware de dispositivo de armazenamento em *cache* é crucial para o sucesso da implementação. Alguns dos fatores a serem considerados são:

- Custo/benefício: um dispositivo de armazenamento em *cache* fornece um número limitado de funções dedicadas e deve ser capaz de fornecer essas funções com melhor relação custo/benefício.
- Facilidade de Instalação e Uso: como solução totalmente pronta e integrada composta por todo o hardware e software necessários, um dispositivo de armazenamento em *cache* é muito fácil de ser instalado e configurado. Assistentes e outras ferramentas automatizadas são uma vantagem.
- Flexibilidade: como é projetado para um único propósito especializado, um dispositivo geralmente oferece um alto grau de flexibilidade de implantação.
- Desempenho: o desempenho depende da capacidade, incluindo a eficiência com que o servidor usa várias linhas e a capacidade de responder rapidamente às solicitações do usuário.
- Facilidade de administração: uma combinação de interfaces de gerenciamento baseadas em browser e em linha de comando oferece flexibilidade, enquanto o controle eficaz de desempenho permite o ajuste do *cache*.
- Capacidade de expansão e confiabilidade: as principais características de um dispositivo de armazenamento em *cache* que afetam essas necessidades são Symmetric Multiprocessor (SMP), recursos de *clustering* com tolerância a falhas e hierarquia de *cache* flexível.

Segundo Brandão e Anido [2001], para quantificar o desempenho de caches para Web, normalmente adota-se o Hit Ratio (HR) e o Byte Hit Ratio (BHR). HR é o percentual de requisições que foram satisfeitas pelo *cache* enquanto que o BHR é o percentual de bytes requisitados que foram enviados pelo *cache*. Assim, quando um documento requisitado foi encontrado em um *cache*, diz-se que houve um Hit. Ao contrário, quando o documento não está no *cache* diz-se que ocorreu um *miss*. Além do HR e do BHR, pode ser usada uma outra métrica muito importante: o tempo de resposta ao usuário, que indica qual o impacto no tempo de espera do usuário das variações de parâmetros de configuração.

3.5 Importância do uso de cache na Web

A importância do uso de *cache* web, tanto em âmbito local, como estadual, nacional e até internacional se evidencia pelo fato de que, para o usuário final, a informação solicitada vai chegar com maior rapidez, sem passar por congestionamentos nos elos nacionais e/ou internacionais, somado a um risco reduzido de possíveis perdas de informação no decorrer das transmissões.

Como uma parte das informações é extremamente dinâmica, é possível que o usuário obtenha informações ultrapassadas ou obsoletas. Esses problemas são resolvidos ou minimizados definindo-se um tempo de validade razoável para a permanência das informações nos servidores proxies ou então implantando-se mecanismos de verificação de mudanças de links e páginas. Contudo, a maior parte das informações é estática e, para esses casos, o modelo apresentado para a utilização de *cache* web é bastante adequado e fortemente recomendado.

3.5.1 Vantagens do uso de cache

O uso de *cache* traz algumas vantagens. Uma delas é o fato de o servidor *cache* ficar mais próximo da máquina que faz a requisição. Isto pode trazer um ganho considerável de desempenho para o usuário, mesmo em caso de linhas pouco congestionadas. Outra vantagem é quando um site, por algum motivo, fica inacessível. Usuários que utilizam um servidor de *cache* continuarão a navegar, desde que os objetos tenham sido previamente armazenados em *cache*. Segundo Jonack e Murta [2002], o uso de sistemas de cache na Web traz vários benefícios:

- Economia no consumo de largura de banda devido à diminuição do tráfego na rede: um documento é enviado uma vez do servidor para o cache e então requisições sucessivas para este documento são servidas a partir desse cache. Um documento encontrado no cache significa que a requisição e a resposta vão percorrer o caminho entre o cliente e o

cache, o que elimina o tráfego redundante na rede. A diminuição do tráfego contribui para diminuir a ocorrência de congestionamentos na rede.

- Diminuição da latência: quando ocorre um *hit*, os usuários podem observar uma redução no tempo decorrido entre o pedido da página e a exibição da resposta. Esta redução ocorre porque o documento está no cache e, portanto, está em um caminho intermediário entre o cliente e o servidor origem, resultando em uma resposta mais rápida. Mesmo em caso de *miss*, o usuário pode ser beneficiado pela utilização generalizada de caches porque seu pedido percorrerá uma rede menos congestionada e poderá encontrar um servidor menos sobrecarregado, o que contribuirá para que a resposta seja mais rápida.
- Redução na carga dos servidores: cada *hit* significa uma requisição que deixa de ser enviada ao servidor origem. Isso beneficia o desempenho dos próprios servidores com a redução da carga, especialmente com a redução nos picos de demanda.
- Aumento da disponibilidade do documento: se o servidor origem do documento está indisponível ou se há algum problema na rede que impede a conexão entre cliente e servidor, o cliente pode acessar a cópia no cache mesmo se esta não está atualizada. Em geral pode-se assumir que uma cópia desatualizada é melhor do que nenhuma cópia.

3.5.2 Desvantagens no uso de cache

Ainda segundo Jonack e Murta [2002], o uso de sistemas de cache na Web traz também algumas desvantagens:

- O usuário pode receber um documento desatualizado.
- O servidor origem recebe menos *hits* e fica impossibilitado de ter uma estimativa real do número de acessos às suas páginas.
- O cache pode se tornar um ponto de estrangulamento, com conseqüente degradação no desempenho.

3.6 Problemas no uso de caches

Existem, porém, muitos problemas que necessitam ser resolvidos quando o uso do *cache* é introduzido. Por exemplo, quanto tempo é possível manter um documento no *cache* e ainda estar seguro de que ele está atualizado? Como decidir quais documentos devem ser colocados no *cache* e por quanto tempo?

A expiração de validade de documentos foi prevista pelo protocolo HTTP, que contém um objeto especificando a data em que o documento já não será válido. Entretanto, existem muito poucos servidores que atualmente fornecem essa informação. Até que ela seja comum (quando a maioria dos servidores Web retornarão a data de expiração) têm-se que confiar em estimativas heurísticas, que fornecem tão somente uma estimativa grosseira do tempo de vida do objeto.

Autores como [Tauscher e Greenberg, 1997, Gwertzman e Seltzer, 1996, Douglass et al., 1997] mostram que, em média, a vida de uma página da Web é curta e não tem mais que dois meses de vida. Isso faz com que todo o registro de log capturado perca seu valor rapidamente pois as referências dentro dele se tornam inválidas depois de um certo tempo.

Uma vez que muitos documentos Web passam por constantes mudanças, especificar um tempo de vida para eles é geralmente uma tarefa difícil. Um documento pode permanecer um bom período de tempo intacto e, repentinamente, ser completamente mudado. Essa mudança pode não ter sido prevista pelo autor do documento e não estar bem especificada na data de expiração.

3.7 Consistência do Cache

O uso de *cache* pode gerar inconsistências no sistema se o objeto original for modificado depois que as cópias forem recuperadas, o que as torna imediatamente desatualizadas. Esse problema é conhecido como inconsistência ou incoerência do *cache*.

O servidor de origem é quem decide sobre o tempo de duração em que uma resposta em *cache* permanece válida. Um *cache* pode ter que garantir que a resposta armazenada em cache ainda é válida antes que a mesma retorne ao cliente que a solicitou [Krishnamurthy e Rexford, 2001]

Vários algoritmos para consistência de *cache* foram propostos para os *caches* na web nos últimos anos. A necessidade de coerência de cache pode depender dos recursos e das diretrizes incluídas no *cache*. Os *caches* podem simplesmente retornar um valor de cache mais antigo, junto com o motivo para a invalidez de uma resposta, que pode ser uma conexão interrompida com o servidor de origem ou o cache muito ocupado [Krishnamurthy e Rexford, 2001]. Dos mecanismos de consistência do cache proposto na literatura, pode-se destacar [Gwertzman e Seltzer, 1996]:

- Time-To-Live (TTL): é uma estimativa do tempo de vida de um objeto, utilizada para determinar quanto tempo um objeto permanecerá atualizado no cache. Para cada objeto é designado um TTL, tal como dois dias ou doze horas. Se um cliente requisitar um objeto que ainda não expirou no cache, então o objeto é fornecido pelo

cache sem precisar contatar o servidor origem. Contudo, se um cliente requisitar um objeto que tem seu TTL expirado, uma requisição condicional (*If-Modified-Since*) é feita ao servidor origem, que verifica se o objeto foi alterado. Se o objeto tiver sido alterado, é retornado o novo objeto com o referente código de *status*; caso contrário, o servidor origem retorna um código de *status*, que informa ao cache que o objeto não foi modificado. TTLs são muito simples de implementar no protocolo HTTP, bastando usar o campo de cabeçalho *expires*, que é opcional.

Vale lembrar que durante o período do TTL, o cache não revalida a resposta e, assim, economiza largura de banda. O valor do TTL pode variar com a resposta e pode ser baseado nos seguintes fatores [Krishnamurthy e Rexford, 2001]:

- Hora de expiração especificada no campo de cabeçalho de resposta: O cache pode adotar diretamente o valor ou modificá-lo com base em sua diretriz. Se não houver uma hora de expiração especificada, o cache pode atribuir um valor de TTL uniforme para os recursos que possuem o mesmo tipo de conteúdo.
 - Frequência do pedido por um recurso em cache: Os recursos solicitados com frequência podem receber um valor de TTL maior.
 - Hora da última modificação do recurso: Um algoritmo TTL adaptativo pode considerar que um recurso modificado recentemente provavelmente mudará novamente. Assim, um cache pode atribuir um TTL menor para um recurso modificado recentemente. Uma resposta em cache poderia receber um valor de TTL maior se um recurso não tiver mudado no servidor de origem por um longo tempo. A informação de hora de modificação é colhida do cabeçalho de resposta *Last-Modified*.
- *Polling* de cliente: É uma técnica em que os clientes verificam periodicamente se um objeto no cache ainda está válido. É baseada nas suposições de que os objetos novos são modificados com mais frequência do que os objetos antigos. O cache associa um campo TTL que é expresso por uma porcentagem, conhecida como *threshold* ou limiar. Um objeto se torna inválido quando o tempo desde a última validação excede os tempos do limiar de atualização do objeto. Por exemplo, se for considerado um objeto cuja idade são 20 dias, com validade checada um dia atrás e que o limiar de atualização é de 10%, então o objeto deve ser marcado como inválido depois de dois dias. Quando for feita uma solicitação a este objeto e o mesmo se tornar inválido, o cache enviará um GET Condicional; caso o objeto tenha sido alterado, será copiada a nova versão do objeto no servidor de origem. Essa estratégia possui a desvantagem de gerar muitas requisições condicionais ao servidor.

- Protocolo de invalidação pelo servidor: Esse protocolo deve ser usado quando for necessário garantir uma consistência mais precisa. Nessa abordagem, cada vez que um objeto é modificado, o servidor notifica os caches de que suas cópias se tornaram inválidas. Um problema com esse protocolo é que ele é computacionalmente caro. Servidores devem guardar um registro dos caches que armazenam seus objetos. Também deve lidar com clientes (caches) indisponíveis, pois se um cache não conseguiu ser notificado, o servidor deve continuar tentando notificá-lo, uma vez que o cache não saberá invalidar o objeto, a menos que seja notificado pelo servidor. O protocolo de invalidação requer modificação no servidor, enquanto que as outras abordagens TTL e *polling* de cliente podem ser implementadas em nível de *proxy*.

O protocolo HTTP/1.1 oferece várias formas para a coerência no cache ser mantida. Se um servidor de origem define uma hora de expiração específica para um recurso, o *proxy* que oferece o *caching* é obrigado a utilizar essa hora. Uma única exceção é uma restrição colocada pelo pedido do cliente por meio de um Cache-Control: only-if-cached, que força o *proxy* a retornar uma resposta em cache sem fazer uma validação no servidor de origem. Se o servidor de origem não definir uma hora de expiração, o *proxy* pode usar uma hora de expiração heurística, geralmente combinada com verificações de coerência [Krishnamurthy e Rexford, 2001].

A técnica mais comum para verificar a coerência é enviar um pedido GET ou HEAD com um cabeçalho de pedido *If-Modified-Since*, que leva consigo a hora da última modificação do recurso, conforme indicado pelo servidor origem. Em alguns casos, a hora em que a resposta foi gerada pode ser a hora da última modificação. As tags de entidade do HTTP/1.1, em conjunto com o cabeçalho *If-Match*, podem ser usadas para realizar verificações de coerência contra variantes específicas de um recurso. O servidor de origem poderia responder com uma cópia completa e um código de resposta 200 ou 304, sem um corpo de resposta. No entanto, uma verificação de coerência exige uma troca pedido-resposta HTTP completa [Krishnamurthy e Rexford, 2001].

3.7.1 Coerência forte e fraca

Se um *proxy* que realiza *caching* envia um pedido de revalidação toda vez que ocorre um acerto do *cache*, a diretriz é chamada de coerência forte. Se o *cache* usa uma heurística para decidir se a resposta em *cache* ainda é válida, sem consultar o servidor de origem toda vez que ocorre um acerto de *cache*, tal diretriz é chamada coerência fraca. Dependendo do *cache* e do conjunto típico de respostas em cache, uma das técnicas pode ser mais apropriada. Uma heurística baseada em locação e as variações na heurística baseada em tempo (TTL) estão entre as técnicas de coerência fraca. Essas duas técnicas diferem no componente que

possui a responsabilidade principal de decidir sobre o intervalo de atualização de um recurso [Krishnamurthy e Rexford, 2001].

A técnica baseada em tempo foi anteriormente explicada. Já a técnica baseada em locação sugere que um *cache* concorda em armazenar uma resposta por um tempo fixo, chamado de período de locação, sem revalidar. No entanto, o servidor origem promete notificar o *cache* se o recurso mudar dentro do período de locação. Se o período de locação se esgotar, o *cache* pode revalidar o recurso e decidir renovar a locação. Essa técnica passa o *overhead* da revalidação para o servidor, que precisa notificar todos os *proxies* que fazem *cache*. Essa técnica não é adequada se um servidor tivesse que notificar milhares de *proxies*. A cooperação entre servidor origem e os *proxies* que fazem o *caching* é necessária para que os *caches* estejam corretos em relação à expiração da locação.

3.8 Conteúdos que não podem ser mantidos em Cache

É necessário verificar sempre quais objetos podem ser armazenados nos servidores de *cache* para que a segurança de conteúdos privativos, tal como senhas de contas bancárias, por exemplo, seja garantida.

Alguns desses objetos nunca podem ser mantidos em *cache*, como os programas Common Gateway Interface (CGI) - que são usados para consulta à base de dados e processamento para formulários; páginas que registram datas atuais e contadores de acesso; objetos protegidos por senha; resultados de *scripts* executados em servidores remotos, entre outros.

De acordo com a RFC 2616, que define comandos e formatos de mensagens HTTP, por padrão, a resposta a uma requisição é armazenada em *cache* se os métodos e campos do cabeçalho da requisição indicam se esta pode ser armazenada. Esses controles podem ser obtidos através de diretivas internas ao objeto e são listados a seguir [Fielding et al., 1999]:

- *public*: indica que a resposta pode ser mantida em *cache* por qualquer *cache*;
- *private*: indica que toda ou parte da resposta é de interesse de um único usuário e não pode ser mantida em *cache* por um servidor que compartilha seu conteúdo. Um *cache* privado (não compartilhado) pode armazenar o objeto;
- *no-cache*: O *cache* não pode usar a resposta para atender requisições subseqüentes, sem executar uma revalidação no servidor de origem. Isto é implementado em servidores que querem prevenir que *caches* configurados para retornar respostas desatualizadas, sejam forçados a acessar a página na origem.

3.9 Uso de Hierarquia

No contexto de *cache* de objetos na *Internet*, os benefícios do *cache* hierárquico foram bem documentados [Douglis et al., 1997].

Para se ter um aproveitamento ainda maior dos recursos de máquinas que executam servidores de *cache*, é possível que elas estejam conectadas em uma hierarquia. Os servidores se comunicam de modo que haja um compartilhamento dos objetos armazenados.

A comunicação entre esses servidores pode se dar de duas formas: através de um protocolo próprio, o Internet Cache Protocol (ICP) ou por *multicast*¹. O ICP é baseado em UDP, e seu objetivo principal é verificar se um outro servidor de *cache* possui uma determinada página. No caso do *Squid*², para se utilizar o ICP é necessário configurar alguns servidores vizinhos. Assim, o servidor ao receber uma requisição de uma determinada página envia pacotes ICP para seus vizinhos, para saber se eles têm esta página armazenada em seu disco local. Se algum deles responder que a possui, o servidor a requisita deste, em vez de recuperar a página original, economizando tempo num acesso mais demorado. Para evitar que vizinhos com problemas atrasem o processo, existe um tempo máximo (2s, em média), em que o servidor espera pela resposta antes de recuperar a página direto na fonte.

Para flexibilizar mais essa hierarquia, existe uma maneira de configurar o que o servidor deve fazer se um vizinho requisitar uma página que ele não possui. Se ele for configurado para apenas responder negativamente caso não a possua, diz-se que ele é irmão do outro servidor. Se, ao invés disso, ele tiver que recuperar a página da fonte original e depois a repassar ao servidor que pediu, diz-se que ele é pai desse servidor. Nesse caso a página estará no banco de dados tanto do servidor a quem foi requisitado inicialmente, quanto no de seu pai. É possível também configurar diversos irmãos e pais para um servidor. Nesse caso, ele faz o pedido a todos eles e pega a página daquele cuja resposta chegar primeiro.

A utilização de hierarquias aumenta muito o HR combinado de diversos servidores. Se três máquinas forem configuradas apenas para responder às requisições de usuários finais, sem comunicação entre si, haverá uma grande probabilidade de uma delas estar buscando um objeto diretamente da origem, que outro servidor acabou de buscar e que poderia simplesmente ser repassado de um para outro. Se um dos três for colocado como pai dos outros dois, o pai, com o passar do tempo, terá em seu banco de dados um grande número de páginas que serão aproveitadas pelos dois filhos. Se, além disso, o pai for irmão de outros servidores

¹transmissão simultânea para várias estações de trabalho

²O *Squid* é um *proxy* da *Web*, considerado um dos mais populares. O *Squid* é uma excelente ferramenta para liberar o acesso de uma rede interna à *Web* e a servidores de FTP. Se bem configurado e controlado, ele permite que seja definido exatamente quem pode acessar quais serviços e bloqueie sites indesejáveis. Com o controle por horários, evita-se que o acesso à *Internet* comprometa a produtividade, sem ter que bloquear completamente o acesso à *Internet*, apenas com o corte dos *sites* desnecessários.

cache, em outros pontos da rede, esses servidores poderão trocar informações entre si, sem ter que recorrer aos sites originais das páginas.

3.10 Cache Distribuído e Hierárquico

É possível distribuir o *cache* por uma cadeia de computadores usados como servidores. Ao distribuir a carga de objetos armazenados, obtém-se aumento no desempenho do *cache* e também tolerância a falhas, caso algum servidor torne-se indisponível.

Os pedidos dos clientes são mandados para um servidor localizado em uma etapa anterior na cadeia, até o objeto ser encontrado. Quando o objeto é localizado em um servidor mais à frente, ele é armazenado no *cache* de todos os servidores até ser encaminhado ao cliente. O encadeamento é uma maneira efetiva de distribuição da carga e de tolerância a falhas. É possível distribuir os computadores servidores em hierarquia para uso em um escritório de uma filial ou um departamento.

Por exemplo, em um pedido de um objeto da *Internet* por cliente em um escritório de uma filial, o mesmo é enviado para o servidor da filial, depois para o regional ou central antes de ir para a *Internet*. Após a obtenção do objeto na *Internet*, o servidor no escritório principal armazena o objeto requisitado e o devolve ao servidor da filial. Na filial, o servidor armazena o objeto e o encaminha ao cliente. O *cache* hierárquico é importante em uma situação assim, pois os objetos são inicialmente roteados pelo escritório principal enquanto pedidos posteriores são atendidos pelas cadeias de servidores da filial.

O *cache* em cadeia é importante para negócios na *Web*, porque permite uma aproximação entre o *cache* e os usuários. Por exemplo, em uma empresa, o *cache* pode manobrar para além de uma única localidade central na extremidade da rede de uma empresa e deslocar-se para os escritórios e grupos de trabalho locais. Com um provedor de *Internet*, o *cache* pode direcionar-se a um provedor regional, em vez de um provedor central. Além disso, o *cache* em cadeia possibilita a tolerância a falhas, visto que fornece uma rota alternativa para quando a rota principal não estiver operando.

3.11 Roteamento de Proxy na Web

As regras de roteamento de *proxy* na *Web* elevam este conceito mais além, permitindo que pedidos sejam roteados condicionalmente, dependendo do destino. Por exemplo, uma organização com uma filial no Reino Unido pode ter servidor instalado no escritório da filial. A cadeia de servidores da filial pode estar conectada a um provedor de *Internet* do Reino Unido. Uma regra de roteamento no servidor na filial pode direcionar pedidos para *hosts* de

Internet localizados no Reino Unido para o provedor local e todos os outros pedidos para a cadeia servidores escritório central. Dessa forma o computador mais à frente na cadeia, nesse caso a filial no Reino Unido, se beneficia do *cache* do servidor no escritório central. Isso também comporta o benefício do *cache* adicional de objetos recebidos do provedor de acesso à *Internet* local no servidor da filial.

Figura 3.7: Roteamento de *proxy* na *Web*

A Figura 3.7 ilustra como os pedidos dos *proxies* na *Web* podem ser encaminhados para servidores distintos, dependendo do destino dos pedidos. Aqui o servidor na filial encaminha para a *Internet* todos os pedidos dos clientes para páginas do mesmo país. Os pedidos dos clientes para as páginas são encaminhados para o servidor que se encontra no começo da cadeia, no escritório central.

3.12 Particionamento do Cache

O particionamento do *cache* permite uma melhor adequação do espaço de armazenamento às diferentes classes de tamanhos dos arquivos o que permite evitar que muitos arquivos pequenos sejam removidos para que um único arquivo grande seja armazenado [Murta et al., 1998]. Com uma configuração correta das partições é possível aumentar o HR de um cache mantendo-se praticamente inalterado o valor do BHR. O particionamento possui a desvantagem de necessitar de um estudo aprofundado das classes de tamanhos dos arquivos de forma a definir uma correta divisão do espaço de armazenamento. Apresenta também o problema de fragmentação do espaço de armazenagem. A fragmentação ocorre quando o espaço livre total do cache é suficiente para armazenar um arquivo porém, o espaço livre está fragmentado entre as partições. Assim, arquivos deverão ser eliminados do cache para a entrada de um novo.

3.13 Substituição de cache

Quando o *cache* estiver completo, alguns objetos deverão ser removidos para dar lugar para o *caching* de novas respostas. Várias técnicas para substituição foram examinadas nos últimos anos. Algumas delas foram alcançadas a partir das técnicas de *caching* tradicionais e outras elaboradas especialmente para os *caches* da Web [Krishnamurthy e Rexford, 2001].

Os objetivos do *caching*, já especificados anteriormente, causam decisões complexas sobre a substituição no *cache*. A vantagem da continuidade de uma resposta no *cache* pode ser medida por vários fatores, tais como [Krishnamurthy e Rexford, 2001]:

- Custo da busca do recurso
- Custo do armazenamento do recurso
- Número de acessos ao recurso no passado
- Probabilidade do recurso ser acessado num futuro próximo
- Tempo desde a última modificação do recurso
- Heurística da hora de expiração

3.14 Políticas de Substituição

Políticas de substituição são usadas para escolher um ou mais documentos a serem removidos do *cache* para abrir espaço para o armazenamento de um novo documento, quando o espaço livre é insuficiente para armazenar esse novo documento [Williams et al., 1996]. As políticas de substituição podem ou não usar parâmetros e, para descobrir qual a melhor configuração de parâmetros relativos a um determinado sistema de caches para Web, deve-se testar várias combinações de valores para os parâmetros de forma a determinar qual delas possibilita o melhor desempenho [Brandão e Anido, 2001]. As políticas de substituição serão vistas em detalhes no próximo capítulo.

3.15 Considerações Finais

Este capítulo mostrou que o armazenamento de informações em *cache* na Web acelera a disponibilidade de conteúdo, pelo fato de o servidor *cache* ficar mais próximo da máquina que fez a requisição, trazendo um ganho considerável de desempenho para o usuário. Verificou-se também que, quando alguma informação torna-se inacessível por algum motivo, os usuários

podem continuar navegando, desde que os objetos tenham sido previamente armazenados no *cache*.

Os objetivos do *caching*, como a redução dos bits transferidos pela rede e redução da latência percebida pelo usuário acabam por ocasionar decisões e técnicas complexas sobre a substituição ou não do conteúdo armazenado no *cache*. Tais técnicas consistem em uma combinação de um conjunto de medições que incluem o tamanho dos objetos em *cache*, seu tipo de conteúdo e ainda uma noção de distância da rede ao servidor de origem.

No próximo capítulo serão discutidas algumas políticas de substituição, que são algoritmos propostos e em uso para a substituição do cache, que têm impacto direto no desempenho global obtido no sistema.

Políticas de Substituição

4.1 Considerações Iniciais

Neste capítulo serão apresentadas várias técnicas para substituição de objetos que foram estudadas nos últimos anos. Algumas delas originaram-se das técnicas tradicionais de caching, na área de sistemas de arquivos e outras foram elaboradas especificamente para a Web.

4.2 Avaliação das Políticas de Substituição

Uma questão muito importante é determinar quando os objetos serão atualizados ou removidos, de modo a garantir a sua consistência. Isso é fundamental porque alguns objetos podem permanecer estáveis por um longo tempo e de repente mudar; já outros mudam mais freqüentemente.

Se um objeto expirar, o servidor *Web* original será consultado para fazer uma revalidação. Se o objeto armazenado no *cache* contém no cabeçalho o campo que indica quando ele sofreu a última modificação, o servidor *proxy* pode usá-lo para fazer uma requisição ao servidor *Web* e, a partir da comparação das datas, saber se o objeto foi alterado. Se o documento não foi modificado, não será retornado. O cliente receberá como resposta informações como, por exemplo, a nova data de expiração. Se o objeto foi modificado, ele será retornado atualizado.

O problema ocorre quando os diretórios de *cache* estão cheios: alguns objetos devem ser removidos para que novos objetos sejam armazenados. A escolha é feita usando algoritmos de substituição, em conjunto com algumas regras denominadas Políticas de Substituição.

As políticas de substituição devem selecionar, de maneira apropriada, qual ou quais objetos devem ser removidos do *cache*, pois uma seleção inadequada pode comprometer a eficácia do uso de caches. Para que tal seleção seja feita de forma adequada, métricas como recenticidade, frequência de acesso, tamanho dos objetos, período de validade do objeto no *cache* e tempo de transferência devem ser consideradas.

4.3 Políticas que usam um Parâmetro na Substituição

4.3.1 Least Recently Used (LRU)

É uma das políticas mais tradicionais e substitui os objetos que não foram requisitados há mais tempo pois remove o objeto mais antigo em termos da hora em que ele foi acessado por último do *cache*. É uma das mais simples, pois não requer parametrização. Essa política trabalha bem quando há uma elevada localidade temporal de referência na carga, isto é, quando os objetos acessados recentemente são mais prováveis de serem referenciados novamente em um futuro próximo. Essa política é implementada usando uma lista ordenada pelo último tempo de acesso. Adicionar ou remover elementos nessa lista é feito em tempo constante $O(1)$, tempo necessário somente para acessar a cauda da lista. Possui a desvantagem de não considerar o tamanho do objeto e o tempo decorrido para recuperá-lo, mas garante de forma direta que os objetos com acessos mais recentes estejam sempre no *cache* [Dilley et al., 1999].

4.3.2 Least Frequently Used (LFU)

Também considerada uma política tradicional, é um algoritmo simples que avalia os objetos em termos de frequência de acesso e remove o objeto que é usado com menos frequência. Para isso, mantém um contador de referências para cada objeto no *cache*. O objeto que possui o menor contador é selecionado para ser substituído, isto é, o objeto que foi acessado menos vezes. Se mais de um objeto têm o mesmo valor do contador, uma segunda política pode ser usada (por exemplo, a LRU) para resolver o problema. Um inconveniente para a LFU é que alguns objetos podem acumular contadores com valores altos e nunca serem candidatos a ser substituídos do *cache*. Isto acontece com objetos muito referenciados no passado e que não serão mais referenciados no futuro e acabam por poder permanecer no *cache* por muito tempo. Ao contrário da política LRU, a política LFU não pode ser implementada como uma lista encadeada simples, cada remoção ou inserção pode levar um tempo de ordem linear

$O(n)$. Pode-se usar uma estrutura de dados pilha para implementar essa política e, neste caso, as operações de inserção e remoção são de ordem logarítmica $O(\log N)$ [Dilley et al., 1999].

4.3.3 LFU-Aging

Para amenizar o problema da política LFU, uma técnica de envelhecimento (*aging*) pode ser implementada. Isso requer dois parâmetros: o primeiro é A_{max} , que coloca um limite superior para o valor médio do contador de referência para todos os objetos do cache; o segundo é M_{Refs} , que impõe um limite superior para o contador de referência que pode ser obtido para um único objeto. Sempre que a média do contador de referência para os objetos no *cache* ultrapassar A_{max} , o contador de cada objeto é reduzido por um fator de dois [Arlitt et al., 2000].

4.3.4 LFU*

Com o objetivo de minimizar o impacto dos objetos acessados somente uma vez, uma técnica chamada LFU* foi proposta [Arlitt e Williamson, 1996]. Assim como na política LFU, é mantido um contador de referências para cada objeto presente no *cache*. Os objetos candidatos à remoção são somente os que tiveram um único acesso. No caso de ocorrer uma falha, o objeto só será inserido no *cache* se o somatório do tamanho dos objetos que tiverem somente um acesso for maior ou igual ao objeto que pretende-se inserir. Independente de o novo objeto ser inserido ou não no *cache*, serão descartados os objetos que tiverem registrados somente um acesso. A quantidade de espaço dos objetos que tiveram somente um acesso é mantida dinamicamente. Não será adicionado mais nenhum objeto no *cache* caso todos os objetos presentes no *cache* tiverem dois ou mais acessos e não houver mais espaço disponível no *cache*. Isto pode não ser um problema no caso de os objetos que estão presentes no *cache* continuarem populares; mas pode ser desastroso se os objetos presentes no *cache* não forem mais acessados.

4.3.5 LFU*-Aging

Para amenizar o problema apresentado pela política LFU*, é utilizada uma técnica de envelhecimento que reduz objetos com poucos acessos a somente um, tornando-os novamente candidatos à remoção [Arlitt e Williamson, 1996].

4.3.6 Size

Essa política, proposta especificamente para *caches* na *Web*, remove os objetos maiores do *cache* quando for necessário espaço para armazenar novos objetos [Williams et al., 1996]. Como o objeto de maior tamanho atualmente no *cache* é excluído, é possível criar espaço para muitos objetos menores. Assim como a política LRU, essa política possui a vantagem de não necessitar de parametrização [Arlitt et al., 2000]. Essa política, que tende a reter os objetos menores, responsáveis pelo maior número de acerto no *cache*, tem a desvantagem de ser sensível à variações na relação entre tamanho e popularidade, além de gerar a possibilidade de reter objetos pequenos indefinidamente que podem não ser mais referenciados, se nenhum mecanismo de expiração for utilizado.

4.3.7 Least Dynamic Frequency Rule (LDRF)

Essa política usa a frequência de acesso de um objeto como parâmetro para remover um objeto do *cache*. Assim, sempre que um novo objeto for inserido no *cache*, o conjunto de objetos adjacentes com menor frequência de acesso é escolhido para ser retirado. Esse esquema é chamado de LDRF e é uma generalização direta da política LFU para o caso de objetos *Web*. Deve-se lembrar que esse esquema pode criar fragmentos de espaços vazios, que são usados sempre que um novo objeto de tamanho inferior ao fragmento for adicionado no *cache*. A desvantagem apresentada por essa política é que exige muito tempo para verificação da frequência de cada objeto no *cache* [Aggarwal e Yu, 1997].

4.3.8 Localized Least Dynamic Frequency (LLDR)

Para minimizar o tempo gasto para verificação da frequência de cada objeto no *cache*, é proposta uma implementação mais rápida que utiliza um *cache* auxiliar, onde são armazenados os objetos mais recentemente acessados (seguindo uma política LRU). O conteúdo do *cache* auxiliar é usado para reduzir a complexidade na escolha do grupo de objetos que devem ser excluídos do *cache* principal. Aggarwal e Yu (1997) propõem ainda uma política de admissão que usa um *cache* auxiliar, utilizando-se da frequência de acesso aos objetos no *cache*. Tal frequência é comparada com a frequência de referência do objeto a ser admitido.

4.4 Políticas que Usam dois Parâmetros na Substituição

4.4.1 Segmented LRU (SLRU)

Essa política foi originalmente projetada para ser usada em *caches* de disco. Ela considera tanto a frequência quanto quão recentemente um objeto foi requisitado para realizar a substituição. Essa política divide o *cache* em dois segmentos: um protegido (para armazenar objetos que são acessados com mais frequência) e outro não-protegido. Quando um objeto é requisitado pela primeira vez, o mesmo é adicionado no segmento não protegido. Quando ocorre um acerto no *cache*, o mesmo é movido para o segmento protegido. Os dois segmentos são gerenciados pela política LRU. No entanto, somente os objetos que estão no segmento não-protegido podem ser eleitos para uma substituição. Quando for necessário espaço para adicionar novos objetos, aqueles que foram menos usados no segmento não-protegido são removidos. Objetos removidos do segmento protegido são adicionados no segmento não protegido como os que tiveram acesso mais recente. Essa política requer um parâmetro, que determina a porcentagem do *cache* que é alocada para o segmento protegido [Arlitt et al., 2000].

4.4.2 LRU-K

Esta política considera na escolha de um objeto para substituição, tanto a frequência de atualização, quanto quão recente ocorreu sua requisição. Na tentativa de melhorar o desempenho, essa política requer dois parâmetros [Arlitt et al., 2000]:

- K: um contador que indica o número de vezes que um objeto foi requisitado. A política mantém as K referências mais novas para cada objeto, mesmo que ele tenha sido removido do *cache*. Objetos com poucas referências são os primeiros candidatos para substituição;
- RP: usado para limitar o tempo em que serão mantidas as informações dos objetos que não estão mais presentes no *cache*. É necessário para evitar o acúmulo de muitos dados históricos dos objetos que foram removidos e não serão mais referenciados ou referenciados em um futuro próximo.

4.4.3 Frequency-Based Replacement (FBR)

Essa política apresentada em [Arlitt e Williamson, 1996] usa uma combinação das políticas LRU e LFU na decisão de quais objetos serão removidos. O *cache* é dividido em três seções,

conforme mostrado na Figura 4.1. O tamanho de cada seção é definido por dois parâmetros, $F_{(new)}$ e $F_{(old)}$ que especificam a quantidade de espaço dedicada a cada seção. $F_{(new)}$ especifica a quantidade de espaço disponível para a seção *new*; já $F_{(old)}$ delimita o espaço para a seção *old*. O espaço remanescente é dedicado à seção *middle*.

Figura 4.1: Seções usadas na política FBR

Os objetos são inicialmente adicionados na seção *new*; conforme vão sendo inseridos novos objetos nesta seção, os objetos mais antigos, através da política LRU, são movidos para a seção *middle*; se o objeto continuar a “envelhecer” sem ser referenciado será transportado para a seção *old*. Somente os objetos na seção *old* podem ser removidos do *cache* e, neste caso, o objeto será selecionado de acordo com a política LFU. Se o objeto referenciado encontra-se na seção *middle* ou *old*, o seu contador de referência é incrementado, enquanto que na seção *new*, se o objeto for referenciado o contador de referência não é incrementado. Isso acontece porque é usada a política LRU.

4.4.4 Greedy Dual-Size (GD-Size)

Essa política considera o tamanho dos objetos e quão recente cada um deles foi requisitado para fazer a decisão de substituição. Os objetos são ordenados conforme um valor H definido por $H = custo/tamanho$. O objeto que tiver o menor valor de H (min_H) é selecionado para substituição. Quando o objeto é acessado, o valor de H é calculado e somado com o valor anterior, privilegiando os objetos acessados mais recentes. Essa política não considera o tempo decorrido para recuperar o objeto e a frequência que o objeto foi referenciado [Cao e Aware, 1997].

4.4.5 Pyramidal Selection Scheme (PSS)

Essa política trata de objetos com tamanhos diferentes e tem como idéia principal uma classificação piramidal dos objetos no *cache*, baseada em seus tamanhos. Os objetos em cada grupo i dessa classificação têm seus tamanhos limitados entre 2^{i-1} e $2^i - 1$ e são ordenados

pela política LRU. Sempre que um objeto tiver de ser excluído do *cache*, os objetos usados menos recentemente em cada grupo são comparados, considerando-se o seu tamanho e a sua frequência de acesso. Os objetos com maior tamanho e menor frequência de acesso são selecionados para serem removidos do *cache*. Os objetos são removidos até que haja espaço para o novo objeto que está sendo inserido no *cache*. Essa política também implementa um controle de admissão, onde o objeto acessado pela primeira vez é armazenado no *cache* somente se houver espaço suficiente. Caso não haja, a URL e o tempo do último acesso (timestamp) são armazenados em um pequeno *cache* auxiliar. Quando o objeto for acessado pela segunda vez, torna-se candidato a ser armazenado no *cache* [Aggarwal e Yu, 1997].

4.4.6 Lowest Relative Value (LRV)

Essa política inclui o custo e o tamanho de um objeto no cálculo de um valor que estima a utilidade de guardar um objeto no *cache*. O objeto com o menor valor é removido do *cache*. Esse cálculo foi baseado em análise empírica de arquivos de *logs*, onde foram encontradas funções matemáticas que são aproximações da distribuição dos tempos entre acessos ao mesmo objeto e da probabilidade de mais acessos dado que o objeto foi acessado previamente i vezes. Para um dado valor i , P_i denota a probabilidade de um objeto ser requisitado $i + 1$ vezes, dado que o objeto foi requisitado i vezes. P_i é calculado de uma maneira *on-line*, tomando-se a taxa $D_i + 1/D_i$, onde D_i é o número total de objetos atendidos até que tenha sido requisitado pelo menos i vezes. $P_i(s)$ é o mesmo que P_i , exceto que o valor é determinado para somente um objeto de tamanho s . Sendo que $1 - D(t)$ é a probabilidade de o objeto ser requisitado novamente com uma função do tempo em segundos. $D(t)$ é obtido através da seguinte fórmula [Rizzo e Vicisano, 2000]:

$$D(t) = 0.035 \log(t + 1) + 0.45(1 - e^{-t/2e^6})$$

Essa política possui a desvantagem de possuir parametrização com funções logarítmicas e exponenciais, que são consideradas caras computacionalmente.

4.5 Políticas que usam Múltiplos Parâmetros na Substituição

4.5.1 Algoritmo baseado no tempo de recuperação dos objetos

Bolot e Hoschka (1996) apresentaram um algoritmo de substituição em *cache* que objetiva minimizar o tempo de recuperação de objetos, considerando, explicitamente, o tamanho dos

objetos e o tráfego da rede. Esse algoritmo tenta minimizar a probabilidade de remoção de objetos com alto retardo de recuperação. Assim, a política armazena, para cada objeto, os seguintes parâmetros: t_i , tempo em que o objeto foi referenciado pela última vez; S_i , tamanho do objeto; RTT_i , tempo levado para recuperar o objeto e TTL_i , tempo de validade do objeto. Com essas variáveis de estado, é possível modelar a maioria dos algoritmos existentes. Por exemplo, a função de peso $W(t_i, S_i, RTT_i, TTL_i) = 1/t_i$ modela a política LRU, pois os objetos têm o peso inversamente proporcional ao tempo de último acesso ao objeto. O algoritmo proposto por Bolot e Hoschka [Bolot e Hoschka, 1996] usa a seguinte função para medir o valor de cada objeto:

$$W(t_i, S_i, RTT_i, TTL_i) = (W_1 RTT_i + W_2 S_i) / TTL_i + (W_3 + W_4) / t_i$$

onde os valores de W_1 , W_2 , W_3 e W_4 são constantes, cujos valores devem ser determinados dependendo do objetivo.

O algoritmo proposto obteve melhor desempenho que o algoritmo LRU, mostrando a vantagem de se usar um algoritmo de substituição baseado no tempo de recuperação dos objetos [Bolot e Hoschka, 1996].

4.5.2 Hybrid

Essa política proposta por Wooster e Abrams [Wooster e Abrams, 1997] associa para cada objeto i um custo dado pela seguinte fórmula:

$$\frac{NRef_i^{C_2} * (RTT_s + \frac{C_1}{bw_i})}{size_i}$$

Onde C_1 e C_2 são constantes fornecidas (os valores padrão são 8196 e 0.9, respectivamente), o índice s representa o servidor em que o objeto i foi recuperado; $Nref_i$ corresponde ao número de referências feitas ao objeto i , RTT_s são baseadas no tempo gasto para trazer o objeto do servidor ao *cache* em um passado recente, e bw_i corresponde à largura de banda disponível para recuperar o objeto. Essa política tem a vantagem de manter informações sobre a latência e a largura de banda disponível para recuperar cada objeto. Porém, essa abordagem requer muitas informações adicionais e torna o seu custo de implementação proibitivo.

4.5.3 Dinâmica

Essa política utiliza três filas que contêm os mesmos objetos, porém ordenados de maneiras diferente. A primeira fila é ordenada utilizando-se a política LRU; a segunda, usando-se a

LFU, e a última é ordenada pela política SIZE. Dessa maneira, é possível considerar tanto o tamanho dos objetos quanto a frequência com que foram acessados e o tempo desde seu último acesso [Brandão e Anido, 2001].

4.5.4 Peso

Essa política de substituição foi proposta por João Carlos Pinheiro [Pinheiro, 2001]. Para cada objeto, consideram-se cinco parâmetros que são utilizados na função Peso na escolha dos objetos que serão substituídos:

- T_i : tempo em que o objeto foi referenciado pela última vez;
- S_i : tamanho do objeto;
- $Nref_i$: o total de referências;
- LAT_i : tempo decorrido para recuperar o objeto;
- $Class_i$: a classe a que o objeto pertence, sendo que, para este estudo foram utilizadas três classes: “HTML/XML/TEXT”, “IMAGENS” e “OUTROS”.

Essa política utiliza a seguinte função para atribuir um peso para cada objeto presente no *cache*.

$$Peso_i = \frac{Nref_i * (LAT_i/S_i)}{T_i} * Class_i$$

Os objetos que tiverem o menor peso são selecionados para substituição. Caso mais de um objeto tenha o mesmo peso, é aplicada uma segunda política: a LRU. Também, pode ser determinado um limite ao total de referências (MAX_{Nref}), para evitar o problema apresentado pela política LFU. O tempo gasto para recuperar um objeto foi normalizado apenas pelo seu tamanho (LAT_i/S_i), pois o tempo decorrido para recuperar um objeto apesar de estar diretamente relacionado ao tamanho do objeto, também é influenciado por outros fatores, como o congestionamento experimentado pela rede e pelo servidor no momento da requisição, a largura de banda disponível, a localização do servidor, dentre outros, que não foram considerados no estudo devido à falta de informações nas cargas de trabalho analisadas. Outro parâmetro importante é o tempo em que o objeto foi referenciado desde a última vez (T_i), sendo que quanto maior for T_i , menor será o peso (P_i) do objeto. A utilização do parâmetro $Class_i$ foi motivada devido a diferentes tipos de objetos apresentarem taxas de atualização diferentes; por exemplo, documentos hipertextos são atualizados com mais frequência que arquivos de audio ou arquivos binários. O objetivo desse parâmetro

é fazer com que objetos populares e com uma baixa taxa de atualização permaneçam no *cache*. No decorrer do estudo os valores assumidos por $Class_i$ são: 0.10 para objetos da classe “HTML/XML/TEXTOS”; 0.15 aos objetos da classe “IMAGENS”; e 0.20 aos objetos remanescentes.

4.6 Vantagens e Desvantagens das Principais Políticas

A Tabela 4.1 mostra um resumo com as principais vantagens e desvantagens apresentadas por cada política de substituição analisada:

Política	Descrição	Vantagem	Desvantagem
LRU	Substitui objetos que não foram requisitados a mais tempo.	Garante que os objetos com acessos mais recentes estejam sempre no cache.	Não considera tamanho do objeto nem o tempo para recuperá-lo.
LFU	Substitui objetos acessados menos vezes (contador de referências).	Privilegia documentos populares.	Objetos com contadores altos podem nunca serem candidatos a substituição.
LFU-Aging		Ameniza a desvantagem da política LFU com uma técnica de envelhecimento.	
LFU*		Minimiza o impacto dos objetos acessados apenas uma vez.	
LFU*-Aging		Reduz objetos com poucos acessos a somente um.	
Size	Especifica para caches na Web, remove os objetos maiores.	Não necessita parametrização e retém os objetos menores em cache.	Retém objetos pequenos que podem não ser mais referenciados, se não for usado algum mecanismo de expiração.
LDRF - Least Dynamic Frequency Rule.	Generalização direta da LRU para o caso de objetos Web.	Retira o conjunto de objetos adjacentes com menor frequência.	Exige muito tempo para verificação da frequência de cada objeto no cache.
LLDR - Localized Least Dynamic Frequency		Usa um cache auxiliar para reduzir a complexidade na escolha do grupo de objetos que devem ser excluídos do cache principal.	
SLRU- Segmented LRU	Objetos devem estar no segmento não protegido para serem substituídos.	Considera a frequência e o quão recentemente um objeto foi requisitado para fazer a substituição.	
LRU-K	Considera tanto a frequência de atualização quanto quão recente ocorreu sua requisição.	Evita o acúmulo dos dados dos objetos removidos (e não mais referenciados). Objetos com menos referências são os primeiros.	
FBR - Frequency-Based Replacement	Combinação de LRU e LFU na decisão de quais objetos serão removidos.		
GD-Size – Greedy Dual-Size	Objetos ordenados conforme um valor (custo/tamanho). É substituído o menor.	Considera o tamanho dos objetos e quão recente cada um deles foi requisitado para fazer a requisição de substituição.	Não considera o tempo decorrido para recuperar o objeto nem a frequência.
PPS – Pyramidal Selection Scheme	Objetos com maior tamanho e menor frequência de acesso são removidos.	Implementa um controle de admissão(objeto acessado pela 1a. vez é armazenado só se houver lugar.	
LVR – Lowest Relative Value	O objeto com menor valor de utilidade é removido.	Inclui o custo e o tamanho de um objeto no cálculo de um valor que estima a utilidade de guardar um objeto no cache.	
Algoritmo baseado no tempo de recuperação dos objetos.	Diminui o tempo de recuperação de objetos considerando o tamanho dos objetos e o tráfego da rede.	Tenta minimizar a probabilidade de remoção de objetos com alto retardo de recuperação.	
Hybrid	Associa a cada objeto um custo	Mantém informações sobre a latência e a largura de banda disponível para recuperar o objeto.	Requer muitas informações adicionais e torna o custo de implementação inviável.
Dinâmica	Usa três filas com os mesmos objetos ordenados de maneira diferente.	Considera o tamanho dos objetos, frequência com que foram acessados e o tempo desde o último acesso.	
Peso	Atribui um peso a cada objeto presente no cache.	Considera: $T_i, S_i, Nref_i, LAT_i, Class_i$	

Tabela 4.1: Vantagens e desvantagens das políticas de substituição analisadas

4.7 Fatores que influenciam a substituição

Como visto, existem importantes fatores ou características que podem influenciar o processo de substituição de objetos em um *cache*. Tais características podem ser resumidas como mostrado a seguir [Krishnamurthy e Rexford, 2001]:

- recenticidade: tempo decorrido desde a última referência para o objeto;
- frequência: número de pedidos para um objeto;
- tamanho: tamanho do objeto;
- custo: custo para ir buscar um objeto no seu servidor de origem;
- tempo de modificação: tempo decorrido desde a última modificação;
- tempo de vencimento: tempo em que um objeto caduca e pode ser substituído imediatamente.

4.8 Classificação das políticas

Os fatores citados na seção anterior podem ser incorporados na decisão de substituição dos objetos. A maioria das propostas existentes na literatura usa os primeiros quatro fatores [Podlipnig e Böszörményi, 2003]. Existem estratégias que usam uma combinação de dois ou mais fatores. Assim, a classificação mais adequada para as estratégias ou políticas de substituição existentes é encontrada em Jin e Bestavros [2000]:

- Estratégias baseadas na recenticidade: estratégias que incorporam recenticidade (e tamanho e/ou custo) no processo de substituição.
- Estratégias baseadas na frequência: estratégias que incorporam frequência (e tamanho e/ou custo) no processo de substituição.
- Estratégias baseadas na recenticidade/frequência: estratégias que consideram recenticidade e frequência sujeitos a suposições de custo/tamanho fixos ou variáveis.

Esta classificação tem a vantagem de distinguir entre recenticidade e frequência, que são os fatores importantes, além de representarem de formas distintas a importância de um objeto web. Mas, para a classificação ficar completa, [Podlipnig e Böszörményi, 2003] acrescentaram dois itens:

- estratégias baseadas em função: Algumas estratégias avaliam uma função específica que incorpora fatores diferentes. Na maioria das vezes usam parâmetros de ponderação para esses fatores.
- estratégias baseadas em randomização: Estratégias baseadas em randomização constituem uma metodologia não determinística para substituição de objetos em cache e, por isso, devem ser tratadas em uma classe separada.

A Tabela 4.2 ilustra as políticas de substituição citadas, de acordo com as classificações expostas anteriormente.

Tabela 4.2: Classificação das Políticas estudadas

Recenticidade	Frequência	Recenticidade/ Frequência	Função	Randomização
LRU	LFU	SLRU	GD-Size	RAND
LRU-Threshold	LFU-Aging	Generational	GDSF	HARMONIC
Pitkow/Reckers	LFU-DA	LRU*	GD*	LRU-C
Size	α -Aging	LRU-Hot	Server-assisted	LRU-S
LRU-Min	swLFU	HYPER-G	TSP	Randomizada com funções de valores gerais
EXP 1		CSS	Bolot/Hoscka's	
Value-Aging		LRU-SP	MIX	
HLRU			M-Metric	
PSS			Hybrid	
LRU-LSC			LNC-R-W3	
Caching Particionado			LRV	
			LUV	
			LR	

4.9 Vantagens e Desvantagens da Classificação

As estratégias que usam a recenticidade como fator principal são, na maioria delas, extensões da conhecida LRU. A política LRU está baseada na localidade de referência, que caracteriza a habilidade para prever futuros acessos a objetos baseado em acessos passados. Há dois tipos principais de localidade: temporal e espacial. Localidade temporal recorre a acessos repetidos ao mesmo objeto dentro de curtos intervalos de tempo. Isso significa que objetos acessados recentemente têm probabilidade de ser novamente acessados no futuro. Localidade espacial parte do princípio que os acessos feitos a alguns objetos implicam acessos a determinados outros objetos. Isso significa que referências para alguns objetos podem ser uma forma de prever referências futuras a outros objetos. As estratégias baseadas em recenticidade exploram o princípio de localidade temporal.

- **Vantagens:** Consideram a localidade temporal como um fator principal. Como os fluxos de requisições web geralmente apresentam algum tipo de localidade temporal, esse é um procedimento vantajoso. Além disso, tais estratégias são particularmente adaptáveis às mudanças da carga de trabalho como, por exemplo, muitos objetos populares novos. Uma outra vantagem é que são simples de implementar e rápidas. A maioria destas estratégias usa uma lista LRU. Os novos objetos requisitados são introduzidos na

cabeça da lista. Quando ocorre um hit, o objeto é removido de sua posição atual e introduzido na cabeça da lista. A recolocação ocorre na extremidade da lista. Assim, tanto inserção como remoção possuem uma baixa complexidade. Além disso, a procura pode ser auxiliada por técnicas de hashing.

- Desvantagens: As variantes mais simples da LRU não combinam recenticidade e tamanho de uma maneira útil e balanceada. Como os objetos da web são geralmente de diferentes tamanhos, esse tamanho deve ser considerado em cada substituição. SIZE faz isso; entretanto, é demasiado agressivo, pois coloca uma ênfase demasiada no tamanho dos objetos. LRU-Min é também focalizado mais no tamanho. A estratégia de PSS é uma exceção, pois é simples de executar, é rápida e combina tamanho e recenticidade de uma maneira mais equilibrada. Adicionalmente, uma estratégia caching LRU particionada, corretamente parametrizada, pode ser muito simples e rápida. Uma outra desvantagem é que essas estratégias não consideram a informação da frequência, que podia ser um importante indicador em ambientes mais estáticos.

Não obstante, as estratégias baseadas em recenticidade são usadas freqüentemente em caches do proxy. Muitos fornecedores de cache usam uma estratégia LRU simples ou variantes dela [Podlipnig e Böszörményi, 2003].

As estratégias que usam a frequência como fator principal são mais ou menos extensões da LFU e são baseadas no fato de que os diferentes objetos da web têm diferentes valores de popularidade e que esses diferentes valores de popularidade resultam em diferentes valores de frequência. As estratégias baseadas em frequência seguem esses valores e usam-nos para as decisões futuras.

- Vantagens: Consideram a frequência do acesso. Isto é importante nos ambientes estáticos onde a popularidade dos objetos não muda muito sobre um período específico de tempo como, por exemplo, dia ou semana.
- Desvantagens: A primeira desvantagem é que as estratégias baseadas na LFU requerem um gerenciamento mais complexo do cache. LFU pode ser executado, por exemplo, com uma fila de prioridade. Uma outra desvantagem é a poluição do cache. As contagens da frequência são estática para mudanças dinâmicas no workload. Conseqüentemente, uma técnica de envelhecimento foi introduzida. Mas o envelhecimento não é nada mais que uma técnica baseada na recenticidade. É questionável se as técnicas de envelhecimento sofisticadas são melhores do que as técnicas simples baseadas em recenticidade em ambientes dinamicamente em mudança. Além disso, as técnicas de envelhecimento adicionam complexidade ao processo da substituição. E, por fim, existe o problema de valores similares. Muitos objetos podem ter a mesma contagem da frequência. Neste caso, um fator para um desempate é necessário.

Podlipnig e Böszörményi [2003] registram que a LFU e suas extensões podem ser implementadas de duas formas diferentes:

- LFU Perfeito. LFU perfeito conta todas as requisições a um objeto i . As contagens do pedido persistem através das recolocações. De um lado, isso assegura que a contagem das requisições representam todas as requisições do passado. Por outro lado, essas estatísticas têm que ser mantidas para todos os objetos vistos no passado. (*overhead* de espaço)
- LFU Em-Cache. As contagens são definidas somente para objetos em cache. Embora isto não represente todas as requisições no passado, assegura uma gerência mais simples (menos *overhead* de espaço).

As estratégias que usam recenticidade e frequência podem usar outros fatores para encontrar um objeto para substituição.

- Vantagens: Combinam a recenticidade e a frequência. Se projetadas corretamente, tais estratégias podem evitar os problemas das estratégias baseadas apenas na recenticidade ou frequência descritas anteriormente.
- Desvantagens: devido aos procedimentos especiais, a maioria destas estratégias introduz uma complexidade adicional. Somente a *Generational Replacement* e a LRU* tentam combinar uma implementação simples da LRU com contagem da frequência. Entretanto, não consideram o tamanho.

As estratégias baseadas em função usam uma função geral para calcular o valor de um objeto. Dependendo da função, a estratégia escolhe o objeto com o menor ou maior valor.

- Vantagens: Não adotam uma combinação fixa de parâmetros. Com a escolha apropriada de parâmetros, um deles pode tentar otimizar alguma métrica de desempenho. Consideram o número de parâmetros para lidar com diferentes situações de carga de trabalho.
- Desvantagens: Escolher pesos apropriados é uma tarefa difícil. Algumas propostas assumem que os pesos são derivados dos estudos de traces da web. Esta é um método simples mas passível de erro. As cargas de trabalho da web mudam em relação ao tempo e requerem algum ajuste capaz de se adaptar aos parâmetros. Essa capacidade de adaptação adiciona mais complexidade ao processo de substituição. Segundo Podlipnig e Böszörményi [2003], não existe nenhum estudo na literatura no que diz respeito a essa capacidade de adaptação. Uma outra desvantagem é que o uso da latência no cálculo do valor pode introduzir alguns problemas. Recenticidade e frequência podem

ser alcançadas facilmente pelo fluxo de requisições feitas no passado. Já a latência é influenciada por muitos componentes no trajeto entre o cliente e servidor, o que leva a uma estimativa inexata da mesma. Conseqüentemente, usar a latência pode conduzir às decisões de substituição inadequadas.

As estratégias baseadas em randomização usam decisões aleatórias para encontrar um objeto para substituição e apresentam um método diferente para substituição em cache, que tenta reduzir a complexidade do processo de substituição sem degradar a qualidade.

- Vantagens: Estratégias baseadas em randomização não precisam de estrutura de dados especial para inserção e remoção de objetos, além de serem simples de implementar.
- Desvantagens: Estratégias baseadas em randomização são mais difíceis para avaliar pois diferentes simulações, rodando com o mesmo trace de requisições, acarretarão resultados ligeiramente diferentes.

A Tabela 4.3 mostra um resumo com as principais vantagens e desvantagens das políticas que usam as estratégias descritas neste capítulo.

Tabela 4.3: Vantagens e desvantagens das estratégias usadas

Substituições baseadas em	Vantagens	Desvantagens
Recenticidade	Consideram a localidade temporal. Simples de implementar e rápidas.	As variantes mais simples da LRU não combinam recenticidade e tamanho. Não consideram a informação da frequência.
Frequência	Consideram a frequência do acesso.	Complexidade. Poluição do cache. Valores similares.
Recenticidade/Frequência	Combinam a recenticidade e a frequência. Podem evitar os problemas das estratégias baseadas apenas na recenticidade ou frequência.	Devido a procedimentos especiais, a maioria destas estratégias introduz uma complexidade adicional.
Função	Não adotam uma combinação fixa de parâmetros. Consideram os parâmetros para lidar com diferentes situações de carga de trabalho.	As cargas de trabalho da web mudam em relação ao tempo e requerem algum ajuste capaz de se adaptar aos parâmetros.
Randomização	Não precisam de estrutura de dados especial para inserção e remoção de objetos. São simples de implementar.	Difíceis para avaliar. Diferentes simulações acarretarão resultados ligeiramente diferentes.

4.10 Limitações das Políticas de Substituição

Existem algumas limitações nas políticas de substituição analisadas neste capítulo. A primeira delas é que, embora seja considerado o tamanho do documento como parâmetro re-

levante à solução do problema, nenhum dos autores citados anteriormente neste capítulo considerou e analisou as implicações trazidas devido a grande diferença do tamanho dos documentos no desempenho da política proposta e, quando a diferença de tamanho foi considerada, apenas estabeleceu-se um limite para o tamanho dos documentos que poderiam ser armazenados no *cache* [Markatos, 1996].

4.11 Considerações Finais

Este capítulo apresentou um conjunto de políticas de substituição de objetos em *caches* na *Web*. Como pôde ser observado, não há uma política que possa ser apontada como geral e que traga sempre os melhores resultados. A escolha da melhor política, dado um domínio de aplicação *Web*, não é uma tarefa trivial, bem como analisar os ganhos e perdas oriundos da escolha de uma dada política também constitui uma tarefa difícil de ser realizada.

O próximo capítulo apresentará alguns conceitos básicos sobre estatística, probabilidade, principais aspectos envolvidos com a simulação de sistemas em geral e, principalmente, aspectos específicos de simulação na *Web*, mostrando, como exemplo, o funcionamento do simulador proposto inicialmente por Arlitt e Williamson [1996], modificado por Pinheiro [2001] e que foi utilizado e expandido por mim para adequar-se à uma nova política.

Simulação na Web

5.1 Considerações Iniciais

Neste capítulo são apresentados os principais conceitos sobre simulação, que é uma técnica utilizada com muita frequência para ajudar a compreender o funcionamento da Web e realizar estudos de avaliação de desempenho baseados em experimentos reais. Para que os resultados da simulação possam ser avaliados adequadamente, são mostrados alguns conceitos básicos de probabilidade e estatística, incluindo conceitos de média móvel exponencial, que será utilizado na determinação de uma nova política de substituição.

5.2 Probabilidade e Estatística para Simulação

Para a realização e avaliação adequadas de simulação é necessário o conhecimento prévio de alguns conceitos de probabilidade e estatística.

5.2.1 Variável Aleatória

Uma variável aleatória é um mapeamento do resultado de um evento em um número. O estado de um servidor, por exemplo, pode ser modelado como uma variável aleatória que pode assumir o valor 1 quando o servidor estiver funcionando ou zero caso contrário. Neste exemplo, a variável mapeia o evento estado do servidor (desligado ou funcionando) em um

número (0 ou 1). Uma variável aleatória pode ser discreta ou contínua. Uma variável é discreta quando o conjunto de possíveis valores é enumerável, como no exemplo acima. Outro exemplo de variável discreta é o número de pacotes IP descartados por um roteador em um intervalo de tempo. Já uma variável contínua pode assumir qualquer valor dentro de um determinado intervalo, ou seja, o conjunto dos possíveis valores não é enumerável. O atraso observado no estabelecimento de uma conexão entre um usuário e um servidor em uma rede pode ser modelado como uma variável contínua [Kamienski et al., 2002].

5.2.2 Eventos Independentes

Dois eventos são chamados independentes se a ocorrência de um evento não afeta a probabilidade de ocorrência do outro. Durante a realização de uma simulação é extremamente importante identificar a relação de dependência entre as variáveis envolvidas [Kamienski et al., 2002]. O número de pacotes descartados por um roteador e o número de pacotes que chegam em um intervalo de tempo são variáveis dependentes. Já o número de chamadas telefônicas que chegam a uma central num intervalo de tempo e o tempo de duração das chamadas são variáveis independentes.

5.2.3 Distribuição de Probabilidade e Função densidade

Considere o exemplo da variável X que modela o estado de um servidor, e que $P[X = 1] = p_1$. O $P[X = 1] = p_2$. O conjunto $\{p_1, p_2\}$ é chamado de distribuição de probabilidade da variável discreta X . A distribuição de probabilidade determina completamente o comportamento da variável, uma vez que todos os possíveis valores da variável têm suas probabilidades especificadas [Kamienski et al., 2002].

No caso de uma variável contínua X , define-se uma função de distribuição acumulada $F_X(x)$ que determina a probabilidade da variável assumir um valor menor ou igual a um determinado valor x :

$$F_X(x) = P(X \leq x) = \int_{-\infty}^x f_X(u) du$$

onde $f_X(x)$ é a função densidade de probabilidade (ou *probability density function* - PDF) de X . Assim, como a distribuição de probabilidade define completamente uma variável discreta, a função densidade define o comportamento de uma variável contínua [Kamienski et al., 2002].

5.2.4 Média ou Valor Esperado

A média ou valor esperado de uma variável aleatória X é dado por:

$$E(\chi) = \sum_{i=1}^n p_i x_i = \int_{-\infty}^{+\infty} x f_{\chi}(x) dx$$

O somatório é utilizado para o caso de uma variável discreta. A média é um parâmetro muito utilizado na prática para obter estimativas do comportamento de um sistema. Por exemplo, em uma rede de computadores, os parâmetros usualmente utilizados para avaliação de desempenho são vazão média e atraso médio. Aspectos de como obter esse parâmetro e qual a significância do seu valor são abordados mais adiante na seção sobre estimativa de parâmetros [Kamiński et al., 2002].

5.2.5 Variância e Desvio Padrão

Em determinadas situações a descrição do comportamento do sistema utilizando apenas a média não é suficiente. Na realidade, a média $E[X]$, não contém nenhuma informação sobre o espalhamento dos dados, ou seja, quão distante as amostras estão do valor médio. No caso de aplicações multimídia na Internet, por exemplo, medir o atraso médio em uma conexão não é suficiente para avaliar a qualidade do serviço, pois um valor médio razoável pode ser resultado da combinação de atrasos bastante elevados e atrasos pequenos. Essa grande variação do atraso pode ser mais prejudicial para a aplicação do que um atraso médio elevado. Nesse caso, um parâmetro para medir a dispersão dos dados em relação ao valor médio é extremamente importante [Kamiński et al., 2002].

A variância e o desvio padrão são parâmetros que medem a dispersão dos dados em relação à média. A quantidade $(X - E(X))^2$ representa o quadrado da distância entre X e sua média e o valor médio dessa quantidade é chamado variância de X [Kamiński et al., 2002]:

$$Var(\chi) = E[(\chi - E(\chi))^2] = \sum_{i=1}^n p_i (x_i - E(\chi))^2 = \int_{-\infty}^{+\infty} (x_i - E(\chi))^2 f_{\chi}(x) dx$$

O quadrado da distância é utilizado para poder computar a dispersão tanto de valores acima, como abaixo da média. A variância é tradicionalmente denotada por σ^2 . A raiz quadrada da variância é chamada de desvio padrão e é representado por σ [Kamiński et al., 2002].

5.2.6 Amostragem de dados e estimativa de parâmetros

As características do conjunto de dados obtidos através das ocorrências de uma variável aleatória são representadas por parâmetros como valor esperado, variância e desvio padrão, por exemplo. Na maioria dos problemas reais, estes parâmetros são desconhecidos e para obtê-los, seria preciso repetir o experimento um número infinitamente grande de vezes, o que é impossível. Na prática, estes parâmetros são aproximados por valores obtidos através da amostragem de alguns valores entre o conjunto das ocorrências da variável. Estas estimativas são chamadas de estatísticas. A média aritmética (média amostral) de um conjunto $\{x_1, x_2, \dots, x_n\}$ de ocorrências de uma variável X , representada por $\bar{x}$, é uma estimativa (estatística) para o valor esperado $E[T]$. É importante a distinção entre um parâmetro e uma estatística, porque o parâmetro é um valor fixo, enquanto uma estatística é uma variável aleatória [Kamiński et al., 2002].

5.2.7 Intervalo de confiança

Na realidade não é possível encontrar uma estimativa perfeita para a média a partir de um número finito de amostras de tamanho finito. Neste caso, a melhor opção é obter limites probabilísticos, por exemplo, c_1 e c_2 , de forma que seja alta a probabilidade $(1 - \alpha)$ de a média exata pertencer ao intervalo (c_1, c_2) [Kamiński et al., 2002]:

$$P(c_1 \leq E(X) \leq c_2) = 1 - \alpha$$

O intervalo (c_1, c_2) é chamado intervalo de confiança para a média, α é chamado nível de significância e $1 - \alpha$ é chamado coeficiente de confiança. Tipicamente, o intervalo de confiança é expresso em termo de um percentual próximo de 100%, por exemplo, 90% ou 95%; enquanto o nível de significância α é expresso como uma fração e é geralmente próximo de zero, por exemplo, 0,05 ou 0,1 [Kamiński et al., 2002].

5.3 Média Móvel

A média móvel é um conceito matemático largamente usado em vários ramos da ciência. No caso da Análise Técnica bolsista, a média móvel de 14 dias, por exemplo, é definida, em cada dia, como a média das cotações dos últimos 14 dias, inclusive. Quanto à forma de a média ser calculada, ela pode ser simples, pesada, exponencial ou triangular.

A idéia fundamental de investir com uma média móvel, no caso da bolsa de valores, é detectar o início da subida de um título e entrar logo nele, bem como detectar o início da descida e sair logo dele. Quando o método detecta o início de uma subida diz-se que gerou

um sinal de compra. Há vários critérios que podem servir de sinal de compra mas um dos mais interessantes consiste no cruzamento da média móvel pela curva das cotações no sentido ascendente.

Uma forma de refinar o método é considerar duas médias móveis, uma de período mais curto e outra de período mais longo. Desenharam-se as três curvas no gráfico (cotações, média móvel curta e média móvel longa) e considera-se sinal de compra a ultrapassagem de ambas as médias móveis no sentido ascendente. Similarmente considera-se sinal de venda a ultrapassagem de ambas as médias móveis no sentido descendente.

5.3.1 Média Móvel Simples

A Média Móvel Simples (*Moving Average*) pode ser entendida através do seguinte conceito:

Sendo $P(d)$ o preço de fechamento no dia d , a média móvel simples para, por exemplo, 5 dias, será:

$$MS = [P(d) + P(d - 1) + P(d - 2) + P(d - 3) + P(d - 4)]/5.$$

Os seguintes prazos podem ser levados em consideração na construção de médias móveis, tanto simples quanto exponenciais:

- Curtíssimo prazo: 5 a 13 dias;
- Curto prazo: 14 a 25 dias;
- Médio prazo: 26 a 74 dias;
- Longo prazo: 75 a 200 dias.

Os investidores tipicamente compram um ativo quando o preço deste sobe acima de sua média móvel, e vendem-no quando o preço cai abaixo desta mesma média.

5.3.2 Média Móvel Exponencial

A Média Móvel Exponencial (*Moving Exponential Average*) pode ser entendida através do seguinte conceito:

Sendo $P(d)$ o preço de fechamento no dia d , a média móvel exponencial será:

$$ME(d) = ME(d - 1) + [2/(N + 1) * (P(d) - ME(d - 1))]$$

onde N é o número de dias para os quais se quer o cálculo.

5.4 Uso da estatística para análise dos dados

De acordo com os motivos expostos anteriormente, deve-se lançar mão da estatística como maneira de analisar os dados de forma mais científica e, assim, tentar determinar a periodicidade com que são realizadas as atualizações e colocar essas atualizações em função do tamanho do arquivo. Além disso, verificar o tempo que foi necessário para recuperar o objeto também torna-se uma tarefa de grande importância.

Existe disponível uma grande quantidade de ferramentas que, através da análise do arquivo de *log* dos servidores *Web*, fornecem informações estatísticas sobre os acessos às páginas [Access Log Analyzers, 2002] como, por exemplo, número de acessos por páginas, tráfego total e origem das requisições. Porém, devido às características dos hiperdocumentos disponibilizadas na rede, estas informações não são suficientes para desvendar o comportamento que é desejado para analisar determinado parâmetro.

Em um primeiro momento, os arquivos de *log* de acessos podem apresentar-se como a solução ideal para a análise do acesso a *sites* da *Web*. Contudo, é importante mencionar que os arquivos de *log* de acesso oferecem recursos para que sejam realizadas análises apenas de cunho estritamente quantitativo, facilitando a identificação de questões relativas a “o quê”, “quando” e “por quem”. De acordo com Haigh [Haigh e Megarity, 1998], os dados contidos em um arquivo de *log* de acessos podem ser processados para gerar relatórios, tais como:

- total de arquivos e kbytes servidos com sucesso;
- número distinto de endereços IP servidos e número de requisições associadas a cada endereço;
- número de requisições feitas por sufixos de domínios;
- número de requisições para arquivos específicos ou diretórios;
- totalizações e médias por períodos específicos de tempo (horas, dias, semanas, meses, anos);
- URLs visitadas anteriormente pelo usuário.

Para obterem-se relatórios extraídos a partir de um arquivo de *log* de acesso, é recomendável o uso de uma ferramenta automatizada adequada para tal fim. É possível, contudo, analisar o conteúdo de um arquivo de *log* de acesso de forma manual embora tal procedimento não seja recomendável, pois o tamanho do referido arquivo freqüentemente possui milhares de linhas.

A análise cautelosa e a compreensão das características de carga na Web são fundamentais para um bom planejamento e implementação de caches na Web, pois proporcionam técnicas que melhoram o uso da largura de banda e reduzem a latência. A caracterização da carga de trabalho é feita através da análise de dados obtidos de servidores Web configurados para armazenar informações sobre as solicitações dos usuários ou através de servidores proxy. A escolha entre coletar dados nas máquinas clientes, servidores Web ou servidores proxy depende do tipo de estudo que se deseja realizar. As cargas obtidas nos servidores Web e proxy são mais apropriadas para avaliar medidas de desempenho no nível de sistema, como a demanda de largura de banda, enquanto as cargas obtidas nos clientes são mais apropriadas para caracterizar o comportamento dos usuários [Pinheiro, 2001].

As cargas de trabalho utilizadas foram obtidas a partir do registro em ordem cronológica de logs de acesso do Squid, onde cada linha do log corresponde a uma URL solicitada pelo usuário. Para fazer uma análise, deve-se levar em conta os dados que estão contidos em cada linha do arquivo de registro dos traces. Esses dados estão mostrados e explicados na Figura 5.1:

96951.129	23	109.77.92.124	TCP_HIT/200	1048	GET	http://www.foofoo.gif	-	NONE/-	image/gif
1°	2°	3°	4°	5°	6°	7°	8°	9°	10°

1° - tempo no qual a conexão com o cliente foi encerrada.
2° - tempo de duração da conexão em mili-segundos.
3° - endereço IP do cliente.
4° - resposta do cache (*hit* ou *miss*) e o código de resposta do http.
5° - tamanho em bytes transferido para esta requisição.
6° - método do HTTP
7° - URL.
8° - desabilitado (identificação do usuário).
9° - como e onde a URL foi buscada.
10° - indica o tipo do conteúdo da URL (imagem, texto, áudio, vídeo, código executável, etc).

Figura 5.1: Linha de um arquivo de registro de traces

5.5 Simulação de Sistemas

Simulação é uma ferramenta importante e amplamente utilizada para testar o comportamento de componentes de redes. É uma técnica muito utilizada para avaliação de desempenho de sistemas em geral. Quando um sistema a ser avaliado não está disponível, o que é o caso em muitas situações, uma simulação é o caminho mais fácil para prever o desempenho ou comparar alternativas. No caso de redes de computadores, como a Internet, apesar do sistema já estar implementado e funcionando, uma alteração em um ponto crítico pode

gerar resultados indesejados, além de muitos transtornos. Logo, a realização de simulações é o melhor caminho para obter boas estimativas do comportamento do sistema após uma possível modificação [Kamiński et al., 2002].

O comportamento da Internet pode ser avaliado e compreendido através de quatro técnicas, que são: medição, experimentação, análise (modelagem analítica) e simulação [Kamiński et al., 2002].

Medição é uma técnica fundamental para compreender o comportamento atual da Internet, com relação a uma série de diferentes pontos de vista. Medições podem ser usadas, por exemplo, para tentar inferir o tamanho da Internet (redes e usuários), assim como padrões de tráfego, protocolos mais utilizados e volume de tráfego. É possível também obter informações sobre o tamanho dos pacotes, quantidade de roteadores entre pontos distintos na Internet, além de parâmetros de desempenho, como atraso fim a fim, perda de pacotes e vazão [Kamiński et al., 2002].

A realização de experimentos também é uma técnica importante para conhecer e avaliar questões de implementação de aplicações e protocolos. A experimentação pode ser realizada com novos protocolos/aplicações na Internet atual, ou então em ambientes de teste, onde pode-se ter um maior controle sobre os elementos a serem estudados. No entanto, medição e experimentação são técnicas limitadas somente a explorar a Internet atual. Elas podem ser usadas para construir ambientes que tentam imitar algum possível cenário futuro, mas certamente de maneira muito limitada. Além disso, nem sempre é possível ter acesso a redes e equipamentos, e essas experiências práticas geralmente têm um custo alto [Kamiński et al., 2002].

A técnica de análise possibilita explorar um modelo da Internet sobre o qual se tem controle completo. Através de modelos matemáticos imensamente simplificados, podem-se obter resultados rapidamente. Por isso, é também uma técnica barata, pois em princípio não necessita de equipamentos especiais. O risco de confiar em resultados analíticos é que os pressupostos e simplificações são tantos que o comportamento original da Internet pode ser perdido. Obviamente, trabalhar com análise requer um forte embasamento matemático [Kamiński et al., 2002].

Enquanto medição e experimentação permitem descobrir o “mundo real”, simulação e análise estão restritas a explorar um modelo abstrato do mundo. Simulação se diferencia de medição e experimentação por permitir maior flexibilidade ao estudo e de análise por permitir a construção de modelos mais complexos e representativos do mundo real. Um importante papel da simulação é desenvolver a intuição das pessoas sobre o comportamento de aplicações, protocolos e tecnologias de rede. Isso é muito útil para pesquisadores, professores, alunos e até para profissionais que trabalham com o projeto de novas redes (podem convencer seus clientes, mostrando a eles como a nova rede vai se comportar) [Kamiński et al., 2002].

Como simulação é uma técnica que se baseia na construção e execução de programas, ela geralmente é a preferida pelo pessoal da área de computação. Vários projetos de pesquisa são baseados inteiramente em resultados de simulação, o que pode trazer alguns riscos. Costuma-se fazer extrapolações à partir de resultados de simulação que freqüentemente se mostram incorretas. Um erro comum e muito freqüente é escolher as situações em que a proposta que está sendo defendida apresenta um desempenho melhor que as outras propostas e omitir as outras situações. Por isso, é importante fazer um bom planejamento da simulação e realizar análises estatísticas sólidas [Kamienski et al., 2002].

5.5.1 Tipos de Simulação

Antes de iniciar uma simulação, deve-se decidir qual o tipo mais adequado para o problema. Dentre os vários tipos de simulação que podem ser encontrados na literatura, os mais importantes para aplicações em redes de computadores são: simulação de Monte Carlo, simulação baseada em traces, e simulação baseada em eventos discretos.

- **Simulação de Monte Carlo:** É uma poderosa técnica para obtenção de soluções aproximadas para problemas complexos que envolvem componentes aleatórios. Este é um tipo de simulação que serve para modelar fenômenos probabilísticos invariantes no tempo. O comportamento de um sistema pode ser caracterizado por distribuições de probabilidades e seus parâmetros. No entanto, na maioria das vezes os parâmetros são desconhecidos e simulações de Monte Carlo podem ser aplicadas na obtenção de estimativas através da realização de várias replicações de um experimento [Kamienski et al., 2002].
- **Simulação baseada em Traces:** Uma simulação baseada em *traces* é a que tem como entrada um registro que contém eventos, ordenados no tempo, observados em um sistema real. Esses registros são chamados de *traces*. Este tipo de simulação é geralmente utilizado na análise de algoritmos de alocação de recursos. Neste caso, um *trace*, contendo a demanda por um determinado recurso, é utilizado como entrada da simulação, a qual pode incluir diferentes algoritmos para serem avaliados sob as mesmas condições de demanda. Uma característica importante é a credibilidade. Um *trace* contendo os acessos feitos a um determinado serviço na Internet, tem uma maior credibilidade do que informações geradas aleatoriamente através de alguma distribuição. Um dos principais problemas dos *traces* é o tamanho. Os *traces* são geralmente seqüências longas e exigem um considerável tempo computacional para serem processados. Outro problema é a dificuldade de variação da carga de trabalho aplicada [Kamienski et al., 2002].

- **Simulação discreta baseada em eventos:** Uma simulação discreta baseada em eventos utiliza um modelo de estados discretos para o sistema. Diferente de um modelo contínuo, no qual os estados que o sistema pode assumir variam continuamente, em um modelo discreto o sistema só pode assumir um número discreto de valores. É importante notar que, mesmo em uma simulação discreta, o tempo da simulação pode assumir valores discretos ou contínuos. Qualquer simulação discreta, independente de aplicação, deve conter os seguintes componentes: um escalonador de eventos; um mecanismo de controle do tempo (*clock*); variáveis globais que descrevem os estados do sistema; rotinas para simular os eventos; rotinas para entrada de parâmetros; rotinas para coletar resultados; rotinas de inicialização; rotinas para gerenciamento dinâmico de memória e um programa principal [Kamienski et al., 2002].

O tipo de simulação escolhido foi a simulação baseada em traces pelo fato de ela ter como entrada um registro que contém eventos, ordenados no tempo, observados em um sistema real. Além disso, oferece credibilidade além de poder incluir diversos algoritmos para serem avaliados sob as mesmas condições de demanda.

Para evitar o grande tempo de processamento devido aos traces serem longos, somente uma pequena quantidade de informação presente em cada linha das cargas de trabalho é utilizada. Para tanto, foi realizado um pré-processamento das cargas de trabalho, reduzindo drasticamente seus tamanhos.

Com a etapa de pré-processamento, foi gerado um arquivo bem mais compacto composto apenas por quatro atributos, de forma a permitir um ganho de desempenho. Os atributos são: um identificador único para cada objeto; o tamanho do objeto, em *bytes*; a classe à qual o objeto pertence e o tempo decorrido para recuperar o objeto no servidor de origem.

5.6 Linguagens de Simulação

A escolha de uma linguagem adequada é um dos passos mais importantes no desenvolvimento de uma simulação, pois uma escolha inadequada pode gerar problemas relacionados com o tempo de desenvolvimento e até falhas no modelo de simulação. Algumas diferenças entre linguagens são: interface amigável; curva de aprendizagem; tempo de desenvolvimento; desempenho (tempo de simulação); escalabilidade para um grande número de eventos (simulação distribuída e paralela). De um modo geral, duas alternativas podem ser consideradas: linguagem de propósito geral e linguagem de simulação [Kamienski et al., 2002].

Um dos primeiros motivos para a escolha de uma linguagem de propósito geral como Pascal ou C é a familiaridade do analista com essas linguagens mais gerais. Por outro lado, escolhendo uma linguagem geral, o usuário terá que desenvolver ou adaptar rotinas

para tratamento de eventos ou geração de números aleatórios que já estão disponíveis em linguagens específicas para simulação. No entanto, alguns fatores mostram que nem sempre uma linguagem de simulação é a melhor escolha. Um modelo desenvolvido em uma linguagem de propósito geral é normalmente mais eficiente pois requer um menor tempo de CPU. Outro fator é a portabilidade das linguagens de propósito geral. Um modelo desenvolvido nesse tipo de linguagem pode facilmente ser convertido para execução em outros sistemas de computação [Kamienski et al., 2002].

No simulador inicialmente proposto por Arlitt e Williamson [1996] foi utilizada uma linguagem de programação de propósito geral: a linguagem C padrão American National Standards Institute (ANSI).

5.7 Planejamento da Simulação

Todo projeto de simulação possui características próprias, mas existem alguns passos que devem ser seguidos no planejamento de qualquer projeto a fim de evitar erros [Kamienski et al., 2002].

5.7.1 Definição dos Objetivos

O primeiro passo em qualquer estudo de avaliação de desempenho, de um modo geral, é definir claramente os objetivos e o sistema que será modelado. Cada sistema possui geralmente uma lista de serviços disponíveis. Todos os serviços disponíveis por um determinado sistema devem ser identificados no início do projeto. Dessa forma, os serviços mais importantes podem ser incluídos na simulação enquanto outros, menos importantes, podem ser desconsiderados [Kamienski et al., 2002].

5.7.2 Escolha das Métricas Adequadas

As métricas são os critérios a serem utilizados para avaliar o desempenho do sistema. Avaliar o desempenho de um sistema é avaliar a capacidade desse sistema no atendimento de sua demanda de serviço. Para essa avaliação definem-se métricas de desempenho de interesse e observa-se o comportamento dessas métricas diante de diferentes situações para o sistema em estudo. Para selecionar as métricas de interesse adequadas é preciso definir todos os serviços disponíveis.

Algumas métricas são usadas para medir a eficiência de caches na Web: a HR, a BHR e a latência.

A HR é a proporção de acertos para todas as requisições. Já a BHR é medida a partir do tamanho dos objetos, ou seja, é a razão entre os *bytes* atendidos pelo cache e o somatório dos *bytes* de todas as requisições. O aumento da HR pode levar à diminuição da latência na recuperação de objetos a partir de uma rede local, enquanto que a BHR melhora o uso de largura de banda, evitando requisições redundantes ao mesmo objeto [Pinheiro, 2001].

A latência é uma métrica importante para avaliação de desempenho, porque esta representa o tempo esperado por requisições Web. Mas não é tarefa fácil medi-la. O cache armazena o tempo gasto para recuperar uma requisição num determinado momento, mas não há como saber exatamente quando o primeiro pacote requisitado foi transmitido e quando o último pacote de resposta foi recebido. Além disso, os valores dependem de muitos fatores, tais como a velocidade da conexão Internet, hora do dia, dia da semana, localização geográfica, velocidade entre os caches e seus usuários e a sobrecarga experimentada pelo servidor no momento da requisição [Pinheiro, 2001].

5.7.3 Escolha de Parâmetros

Uma listagem de todos os parâmetros que têm influência no desempenho do sistema deve ser feita. Durante o desenvolvimento da simulação, outros parâmetros podem surgir os quais devem ser incorporados à listagem inicial. A lista de parâmetros pode ser dividida em duas partes: parâmetros que serão variados durante a simulação e parâmetros que serão fixos. Os parâmetros variados são chamados de fatores e seus possíveis valores são chamados de níveis. Na maioria dos modelos, a quantidade de fatores e níveis de variação é maior do que os recursos disponíveis para considerá-los. A melhor opção é iniciar com uma lista reduzida e adicionar outros fatores gradualmente de acordo com a disponibilidade de recursos. Os fatores devem ser escolhidos dentre os parâmetros que exercem um maior impacto sobre o desempenho do sistema [Kamienski et al., 2002].

Os parâmetros da simulação usados estão listados a seguir e são os mesmos utilizados por Pinheiro [2001] em seu estudo:

- **Tamanho do cache:** Indica a quantidade de espaço definido para armazenar os objetos no cache. No estudo feito Pinheiro [2001] foram considerados 14 tamanhos de cache: 1, 2, 4, 8, 16, 32, 64, 126, 512 Mbytes, 1, 2, 4 e 16 Gbytes. Como 16 Gbytes pode armazenar todos os objetos solicitados em um intervalo de tempo, será referenciado como um cache de tamanho infinito, isto é, nunca serão realizadas substituições de objetos. Com caches de tamanho elevado pode-se chegar a valores máximos de HR e BHR, assim como observar até que ponto é vantajoso aumentar o tamanho do cache.
- **Políticas de Substituição de Objetos:** Das políticas de substituição de objetos encontradas na literatura, foram selecionadas as políticas LRU, LFU, LFU-Aging, LFU*,

LFU*-Aging, FBR, SIZE e a política MeMoExp. As políticas LRU e LFU foram escolhidas por serem muito utilizadas em sistemas de caches tradicionais, além de serem bastante utilizadas em implementações de caches na Web. Foram escolhidas mais três variações da política LFU (LFU-Aging, LFU* e LFU*-Aging), sendo que LFU* e LFU*-Aging atacam de forma direta os objetos com um único acesso, selecionando-os para substituição. Já a política SIZE seleciona os objetos maiores para substituição enquanto que a política FBR tem como objetivo isolar os objetos com poucos acessos e que não foram referenciados em um certo intervalo de tempo.

- **Período de Aquecimento do Cache:** Determina a quantidade de requisições em que o cache não coletará nenhuma estatística de HR e BHR pois, devido ao cache estar inicialmente vazio, haverá uma quantidade significativa de falhas, o que pode não retratar a situação de um cache real em produção. O objetivo do período de aquecimento é fazer com que o cache passe de um estado transiente a um estado estável de operação. Segundo [Arlitt et al., 2000], o cache se estabiliza com aproximadamente 8% do total de requisições. Para este trabalho, foi considerado que o cache se estabiliza com 10% do total de requisições.

5.7.4 Quantidade de replicações

Uma vez que os fatores e seus níveis estão especificados, é preciso definir a sequência de experimentos a ser realizada, que gera a quantidade de informação necessária com o menor esforço possível. Ou seja, é necessário identificar quantas replicações da simulação são necessárias para a obtenção de resultados confiáveis. A resposta a esta questão vai depender da precisão desejada e pode ser respondida utilizando os conceitos de amostragem de dados e intervalo de confiança. O nível de confiança das conclusões baseadas em um conjunto de dados depende do tamanho deste conjunto de dados. Quanto maior o conjunto, maior a precisão. Dessa forma, o número de replicações exigidas pode ser calculado utilizando-se a relação entre intervalo de confiança e tamanho da amostra [Kamiński et al., 2002].

5.7.5 Análise dos Resultados

O valor dos resultados obtidos depende da implementação correta da simulação e da representatividade desses resultados em relação ao sistema real. Estes dois aspectos podem ser garantidos através da validação e verificação do modelo, do estudo das condições iniciais e do tempo de simulação utilizado [Kamiński et al., 2002]. Tais resultados serão apresentados detalhadamente no próximo capítulo.

5.8 Ambiente de Simulação

O simulador utilizado neste estudo foi projetado Arlitt e Williamson, 1996b, para avaliar o desempenho de caches em servidores Web. Pinheiro [2001] realizou diversas modificações no simulador, para poder trabalhar com cargas de trabalho de servidores proxy. As diversas funcionalidades implementadas em relação ao simulador original para o estudo de avaliação de desempenho foram [Pinheiro, 2001]:

- atualização das classes de objetos (HTML, Imagens, Vídeos, etc.) para novos objetos, como: xml, php, asp, etc;
- implementação de particionamento do cache por classes de tamanho;
- tratamento para cada partição como um cache individual;
- implementação da políticas SIZE, PESO e MeMoExP;

A estrutura básica do simulador é apresentada na Figura 5.2.

Figura 5.2: Estrutura básica do simulador

A Figura 5.3 mostra as duas principais estruturas de dados utilizadas no simulador. A estrutura lnode é composta por uma lista duplamente encadeada com N posições, onde N corresponde ao total de objetos. A informação armazenada para cada objeto é constituída pelo do tamanho do objeto, o tempo para recuperá-lo no servidor de origem, o tempo do último acesso, no número de referências, status (indica se o objeto está ou não no cache) e ponteiros para os objetos seguintes e anteriores. A estrutura de dados Caches é composta por um vetor, sendo cada posição referente a um cache, cada cache possui uma lista duplamente encadeada para todos os objetos, um ponteiro para o primeiro e último objeto, o tamanho em bytes do cache, o total de bytes utilizados atualmente pelo cache, o número de objetos,

o total de acertos, o total de falhas, o total de bytes atendidos pelo cache e o total de bytes não atendidos [Pinheiro, 2001].

Figura 5.3: Principais estruturas de dados do simulador

O simulador determina a efetividade de políticas de substituição de objetos na redução de largura de banda e na porcentagem de objetos atendidos pelo cache [Pinheiro, 2001].

No simulador proposto por Arlitt e Williamson [1996], alguns dos parâmetros de configuração são definidos no próprio código, como o número e tamanho do cache. Isso significa que é necessário editar o código fonte do programa para alterar os parâmetros de configuração.

Esse simulador é composto pelos seguintes arquivos, mostrados na Figura 5.4:

Arquivo	Descrição
convert.c	Cria um arquivo de entrada para os simuladores (LRU, LFU, etc.) Requer dois arquivos: (1) Um a <i>timeseries</i> de pedidos feitos ao servidor Web (Corresponde à saída do programa de análise de carga "Step1") (2) Um a lista de um único documento na <i>timeseries</i> , com um único identificador para cada documento (references, saída do programa "Step2" de análise de carga)
LRU.c	Simula uma cache usando a política Least Recently Used
LFU.c	Simula uma cache usando a política Least Frequently Used
LFU_star.c	Simula uma cache usando a política modifica Least Frequently Used (LFU*)
FBR.c	Simula uma cache usando a política Frequency Based Replacement

Figura 5.4: Arquivos que compõe o simulador

O programa Step1.c percorre um arquivo de *log* e guarda estatísticas com diferentes tipos de respostas e diferentes tipos de arquivos que foram requisitados. As solicitações que obtiveram sucesso são armazenadas numa série temporal, que será usada por um outro analisador. Os arquivos de saída são:

- Valid: uma série temporal das requisições HTTP válidas.
- Invalid: requisições HTTP que não foram formadas adequadamente.
- Methods: métodos usados diferentes do método GET, como PUT ou GET.
- Filetypes: guarda qualquer arquivo diferente dos arquivos que foram predefinidos. Estes arquivos podem ser analisados para atualizar as estatísticas para tipos de arquivos diferentes.

Estatísticas são mantidas para vários tipos de arquivos diferentes. Essas estatísticas compreendem: o número de arquivos de um determinado tipo que foi pedido, os totais de bytes para cada classe de arquivos, o tamanho médio dos arquivos, assim como o menor e o maior arquivo. Os tipos de arquivos são organizados da conforme mostrado na figura 5.5:

Tipo de Arquivo	Extensões possíveis
HTML	html, htm, shtml map, xml
IMAGES	gif, xmb, jpg, jpeg, gif, tif, tiff, bmp
SOUND	au, wav, snd, lha, mid
VIDEO	mpg, mov, qt, mp3, wav
DYNAMIC	pl, cgi, count, urls contendo cgi-bin, ?
TEXT	txt, tet, .bib, .readme, .desc, faq
COMPRESSED	Z, gz, zip, tar
SOURCE CODE	c, h, pas, cc, cpp
BINARIES	exe, class

Figura 5.5: Tipos de arquivos

O programa Step2.c percorre através de uma série temporal de requisições HTTP e determina o funcionamento (número de acessos) de cada arquivo. Desses dados podem ser determinadas estatísticas de várias ocorrências, incluindo o número de bytes únicos, total de bytes transferidos, quais arquivos são referenciados mais frequentemente, quais arquivos são responsáveis pela maioria dos bytes transferidos, entre outras.

5.9 Considerações Finais

Este capítulo apresentou alguns conceitos básicos de probabilidade e estatística, que ajudam a avaliar adequadamente os resultados de uma simulação. Também foram apresentados os principais conceitos de simulação e mostrados os parâmetros e ambiente de simulação que

serão utilizados. O conceito matemático a respeito de médias móveis foi mostrado, para que a política MeMoExp proposta no próximo capítulo possa ser entendida.

O próximo capítulo apresenta também os resultados do estudo de caracterização de carga de servidores proxy com aplicação em caches na Web, que foram encontrados por [Pinheiro, 2001] e usadas para validação das cargas que são utilizadas para avaliação de desempenho de políticas de substituição de objetos em caches na Web. O uso de tais resultados tem por finalidade uma comparação dos resultados encontrados, para mostrar se os mesmos são similares às caracterizações já publicadas na literatura.

Avaliação de Desempenho utilizando uma nova política

6.1 Considerações Iniciais

Neste capítulo é apresentada a política MeMoExp proposta, os resultados obtidos e é feita uma comparação com os resultados alcançados por outros autores, além de mostrar os dados obtidos com o estudo de caracterização de carga para servidores proxies realizado.

6.2 Introdução

A variabilidade nos tamanhos das requisições da Web tem influência significativa no desempenho de sistemas de caches desses objetos [Murta e Almeida, 1999].

Ainda segundo Murta e Almeida [1999], as políticas de reposição utilizadas nos caches da Web, a maioria adaptações das políticas encontradas em caches tradicionais, não são suficientes para tratar as limitações impostas pela variabilidade. Assim, não existem políticas que permitam escolher entre otimizar HR ou BHR ou encontrar um ponto de equilíbrio no desempenho de ambas as métricas.

Foi visto no desenvolvimento desta dissertação que o objetivo principal de um cache é responder a requisições do cliente sem necessitar do servidor origem, o que reduz o tempo

de resposta para o cliente, reduz o uso de banda da Internet pois existe o uso da rede local e economia em transmissões redundantes.

O problema é que esses objetivos podem estar em conflito pois um *miss* pode aumentar o tempo de resposta e, para reduzir o uso de banda deve-se armazenar arquivos grandes assim como, para reduzir o tempo de resposta, deve-se armazenar arquivos pequenos.

6.3 MeMoExP

Esta seção apresenta uma nova política de substituição denominada MeMoExP. Essa política foi criada levando em conta os conceitos sobre média móvel exponencial.

Para tanto foram inicialmente propostas faixas de atualização dos objetos, isto é, foi determinada a periodicidade com que são realizadas as atualizações. Essa faixas encontram-se nos seguintes intervalos:

- objetos atualizados minuto a minuto no servidor origem.
- objetos atualizados hora a hora no servidor origem.
- objetos atualizados dia a dia no servidor origem.
- objetos atualizados semana a semana no servidor origem.
- objetos atualizados mês a mês no servidor origem.

Para que os resultados pudessem ser usados de forma científica, a porcentagem de objetos que pertencem a cada classe definida deveria ser real. Na impossibilidade de obter tais dados, foi estipulado o seguinte critério para as faixas:

- faixa 0: objetos atualizados minuto a minuto. (20%)
- faixa 1: objetos atualizados hora a hora. (30%)
- faixa 2: objetos atualizados dia a dia. (20%)
- faixa 3: objetos atualizados semana a semana. (20%)
- faixa 4: objetos atualizados mês a mês. (10%)

Tais valores puderam ser considerados pois os mesmos podem ser alterados conforme seja necessário. Isso irá depender de onde está o servidor proxy. As porcentagens das faixas podem ser alteradas de acordo com a instituição a qual o proxy pertence.

Essas faixas de atualização são geradas durante a execução do programa.

Pelo fato de minuto, hora, semana e mês serem medidas de tempo diferentes, foi realizada uma conversão para padronização das unidades de medida de tempo. Adotou-se o segundo como padrão.

$$M_{e(p)} = M_{e(p-1)} + [2/(N + 1) * (Q_{(p)} - M_{e(p-1)})]$$

onde:

- $M_{e(p)}$: Média exponencial no período.
- $M_{e(p-1)}$: Média exponencial no período anterior.
- $Q_{(p)}$: Quantidade de requisições em um dado período.
- N : nº de períodos para o qual se quer o cálculo.

Assim, se a quantidade de requisições a um determinado objeto no período é menor que a média exponencial no período, o mesmo é selecionado para substituição.

6.4 Resultados da Pesquisa

As cargas de trabalho utilizadas foram obtidas de quatro servidores proxies: Centro de Informática de São Carlos (CISC), Bolder - Universidade do Colorado (BO2), Palo Alto - Califórnia (PA) e Vale do Silício - Califórnia (SV).

Algumas características gerais são apresentadas na Tabela 6.1.

Tabela 6.1: Resumo das cargas de trabalho

Item	CISC	NLANR - BO2	NLANR - SV	NLANR - PA
Duração do log	10 dias	4 dias	1 dia	3 dias
Data de início	22/05/2001	11/06/2001	11/06/2001	13/06/2001
Data de término	01/06/2001	14/06/2001	11/06/2001	15/06/2001
Total de requisições	838.532	808.748	1.239.504	1.062.534
Média diária de requisições	83.853	202.187	1.239.504	354.178
Total de bytes transferidos (Gbytes)	2,73	7,88	7,72	9,22
Bytes transferidos por dia (Mbytes)	273,1	1.972	7.726	3.074

Um estudo de caracterização de carga para servidores proxies foi desenvolvido. Foram estudadas as cargas de quatro servidores *proxy* que implementam *caches* na Web: (i) CISC; (ii) BO2; (iii) PA e SV.

Algumas características para esse conjunto de cargas foram observadas [Pinheiro, 2001], tais como:

- Verificou-se que o log de acesso do CISC registrou 38,88% das requisições para o ICP e 34,13% foram para classes 2XX, totalizando 73% das requisições. Os proxies que fazem parte do NLANR tiveram entre 52,62% (SV) e 62,89% (BO2) dos códigos de status referentes à classe 2XX, o que denota que a solicitação foi recebida, compreendida e aceita.
- A classe de resposta 3XX variou de 21 a 30% das respostas em todas as quatro cargas analisadas, o que significa que outras ações precisam ser tomadas para completar a solicitação.
- As classes 4XX e 5XX correspondem a erros no cliente e no servidor respectivamente. O servidor proxy do CISC apresentou a maior taxa de erros no cliente, sendo que 4,28% das solicitações possuíam sintaxe inválida ou não podiam ser atendidas, enquanto que NLANR-SV registrou 16%, a maior taxa de erros para a classe 5XX, o que significa que o servidor não conseguiu atender a uma solicitação aparentemente válida; isto ocorreu devido ao grande volume de requisições diárias, em torno de 1,2 milhão de requisições.

A Tabela 6.2 apresenta a utilização das classes de resposta retornadas pelos protocolos HTTP e ICP.

Tabela 6.2: Classes de respostas do protocolo HTTP e ICP

CLASSES	CISC	NLANR – BO2	NLANR – SV	NLANR – PA
2XX	34,13	62,89	52,62	59,83
3XX	21,53	28,30	29,59	30,96
4XX	04,28	02,22	01,08	01,31
5XX	01,11	05,96	16,07	07,65
ICP	38,88	-	-	-
Outros	0,65	2,88	0,46	0,61

- a metade dos objetos acessados é menor que 3 Kbytes, bem abaixo da média que variou de 8,9 a 13,4 Kbytes, devido ao fato de que a média é influenciada por poucos objetos de tamanho muito elevado;
- 82,8% das requisições abrangem objetos menores que 10 Kbytes;

- 99,3% das requisições abrangem objetos com até 100 Kbytes;
- objetos maiores que 100 Kbytes correspondem a uma fração muito pequena do total de acessos;
- entre 97,11% e 99,54% das solicitações foram para o método GET, o que comprova que a Web ainda é utilizada maciçamente para leitura;
- os códigos de status 200, 304 e 000, que resultam no sucesso para obtenção do objeto solicitado pelo cliente, ficaram em torno de 83%;
- o valor máximo encontrado para a HR variou de 40,56% (BO2) a 67,11% (CISC) enquanto que a BHR variou de 27,92% (BO2) a 56,03% (PA);
- a fração dos objetos acessados somente uma vez corresponde a uma fração considerável da cardinalidade do conjunto de objetos, ficando acima de 70%, estes resultados são semelhantes aos apresentados em Mahanti e Williamson (1999);
- observou-se um aumento significativo para as requisições dinâmicas, entre 8% (SV) e 25,8% (BO2) em relação aos resultados de Mahanti e Williamson (1999) que registraram valores bem inferiores (menos de 1% das requisições);
- imagens correspondem à maioria das requisições entre 52,23% (BO2) e 74,37% (SV), resultados semelhantes aos estudos de Mahanti e Williamson (1999) e Abdulla et al. (1997), porém observou-se uma diminuição nos bytes transferidos;
- objetos acima de 1 Mbyte, apesar de terem, em média, somente 0,07% das referências, são responsáveis por 33,22% dos bytes transferidos;
- 20% dos objetos correspondem de 50% a 65% das requisições;
- 10% dos objetos correspondem a 80% dos bytes transferidos.

6.5 Avaliação das Políticas de substituição

Com relação a caches particionados, alguns resultados observados por [Pinheiro, 2001] em sua dissertação de mestrado e comprovados pela realização de novos testes foram:

- a atribuição de um limiar superior para os objetos aceitos no cache pode melhorar a HR, apesar de degradar a BHR;

- constatou-se que a política SIZE em caches particionados tanto por classes de objetos quanto por classes de tamanho, não apresentou desempenho satisfatório. As políticas PESO e LFU apresentaram resultados intermediários, tanto na HR quanto na BHR;
- caches com cinco partições apresentaram HR melhor, enquanto a BHR se manteve praticamente inalterada em relação aos caches não particionados; enquanto caches com três partições apresentaram um aumento menor na HR e uma melhora moderada na BHR;
- a utilização de particionamento por classes de objetos não foi vantajosa em relação aos caches não particionados;
- em todos os modelos de partição as políticas que tiveram melhor desempenho foram baseadas em caches tradicionais (LRU e FBR);

Um dos itens verificados por [Pinheiro, 2001] foi que a utilização de particionamento por classes de objetos não é vantajosa em relação a caches não particionados. Assim, serão descritos a seguir os resultados da avaliação de desempenho de diferentes políticas de substituição de objetos em um cache não particionado. As políticas avaliadas são: MeMoExp, Peso, Size, LRU, LRU*-Aging e FBR. Os principais resultados obtidos com a avaliação dessas políticas foram:

- HR e BHR aumentam conforme aumenta o tamanho do cache.
- caches de tamanho infinito têm desempenho semelhante para todas as políticas tratadas, pois não existe a substituição de objetos, vez que o tamanho do cache é infinito.
- caches pequenos apresentam uma maior discrepância nos resultados. Isso ocorre pois ocorre um grande número de substituições.
- a política MeMoExp teve resultados similares à política FBR, considerada eficiente na literatura.
- em caches sem particionamento a política SIZE apresentou a melhor HR e a pior BHR, enquanto que a política LFU*-Aging apresentou desempenho satisfatório em todas as cargas analisadas, tanto para HR quanto para a BHR, pois essa política trata diretamente o problema de objetos acessados somente uma vez;

As figuras 6.1, 6.2, 6.3 e 6.4 mostram os valores de HR e BHR para as cargas do CISC, BO2, PA e SV respectivamente, utilizando as políticas MeMoExp, Peso, Size, LRU, LRU*-Aging e FBR.

Figura 6.1: CISC - Comparação das políticas de substituição

Figura 6.2: BO2 - Comparação das políticas de substituição

Figura 6.3: PA - Comparação das políticas de substituição

Figura 6.4: SV - Comparação das políticas de substituição

Hoje em dia existe uma infinidade de estratégias de substituição de objetos. E existem alguns algoritmos para tais estratégias que dão bons resultados em diferentes avaliações. Tais algoritmos são considerados como *good-enough* para substituição em caches web mas é questionável se eles são bons o bastante para exigências futuras (como, por exemplo, multimídia, QoS). Uma classificação clara das diferentes estratégias de substituição não é possível assim como não existe uma melhor estratégia para diferentes cargas de trabalho. Além disso, estratégias de substituição dão resultados diferentes para diferentes métricas, conforme mostrado a seguir.

- HR. Sem dúvida, a métrica mais freqüentemente usada para avaliar desempenho de cache. As estratégias que favorecem objetos menores, tais como SIZE, LRU-Min ou estratégias baseadas em função com grande relevância ao tamanho do objeto são eficazes em relação ao HR. Essa métrica é interessante para usuários que querem ter uma porcentagem alta de hits locais.
- BHR. Estratégias que tendem a remover objetos maiores melhoram o HR e diminuem o BHR. Essa métrica é interessante para ISPs quando eles tentam minimizar a quantidade de transferências na Internet.

6.6 Medindo a eficiência do cache

Caches Web são desenvolvidos em vários locais da rede e melhoram a latência quando encontram documentos Web enquanto reduzem o tráfego global na rede. Administradores de caches gastam algum esforço configurando o *software* de *caching* e determinando sua localização satisfatória para obter a melhor qualidade de serviço no que diz respeito a demandas de usuário. Porém, apesar deste esforço, a eficiência dos *caches* pode decair com o passar do

tempo. Realmente, a eficiência de uma determinada infra-estrutura de *caching* depende pesadamente do uso atual dos caches: número de usuários, tipo de documentos pedidos, carga, de máquinas que executam o cache. Desde que estas características são coisas que mudam, é improvável que qualquer configuração e desenvolvimento particulares permaneçam efetivos durante todo o ciclo de vida do *cache*. Então, para maximizar os benefícios oferecidos por *caches* Web, os administradores devem reconfigurar a infra-estrutura existente à medida que as características de tráfico evoluem; isto implica em medidas freqüentes de eficiência do cache [Patarin e Makpangou, 2003].

Isto leva a dois assuntos diferentes. Primeiro, com que freqüência deveria ser executada essas medidas? Com respeito à freqüência de avaliações, é importante poder reagir a modificações súbitas (aumento muito grande de usuários, falhas na rede ou servidores), como também mais lentas com tendências fortes (comunidade de usuário crescente, uso crescente de documentos multimídia). A freqüência apropriada depende principalmente do impacto da eficiência do cache na qualidade atual do serviço. Conseqüentemente a melhor escolha de freqüência é particular para cada sistema. Cada administrador deveria ter uma forma de adaptar a medida da freqüência de avaliação de seus caches, levando em conta os objetivos em termos de melhoria do serviço global e o efeito da degradação sobre o desempenho. Segundo, como enfrentar a subjetividade inerente da idéia de eficiência? A métrica eficiência satisfatória depende das metas definidas pelos administradores do cache; o mesmo nível de desempenho pode conduzir a experiências diferentes por diferentes pessoas em ambientes diferentes. Isso deve ser personalizado. Exemplos de métricas que podem ser usadas incluem a consistência dos documentos devolvidos, economia de largura de banda e economia de latência [Patarin e Makpangou, 2003]. É importante apresentar as propostas diferentes de uma forma estruturada, com uma classificação das estratégias de substituição.

6.7 Considerações Finais

Este capítulo mostrou a política MeMoExp e os resultados obtidos com seu uso, bem como os dados obtidos com o estudo de caracterização de carga para servidores proxies realizado.

Como visto no decorrer desta dissertação, não é possível identificar a melhor estratégia de substituição para caches usados na web. Primeiro, diferentes estratégias de substituição são mais eficazes para diferentes métricas de desempenho. Segundo, diferentes situações de carga de trabalho podem resultar em diferentes classificações de desempenho para um conjunto específico de estratégias de substituição. É esta sensibilidade da carga de trabalho que produz os resultados diferentes descritos na literatura. Há fatores de carga de trabalho típicos que influenciam o desempenho de estratégias de substituição. Os fatores seguintes são importantes em um fluxo de requisições de objetos para um cache:

- tamanho do objeto: é o principal motivo de grande parte das pesquisas existentes para substituição de objetos em caches na web. O tamanho dos objetos da Web pode variar significativamente e é geralmente modelado com uma distribuição de cauda-pesada, uma combinação de carga-leve (corpo da distribuição) e cauda-pesada (cauda da distribuição) [Barford et al., 1998] ou com uma distribuição log-normal [Downey, 2001].
- localidade temporal: descreve a característica que apenas um objeto já referenciado tem uma probabilidade maior de ser referenciado num próximo futuro. Localidade temporal tem dois aspectos: popularidade e correlação. Popularidade descreve a probabilidade a longo prazo de um objeto encontrar-se em requisições futuras. A distribuição inclinada da popularidade de um conjunto de objetos é caracterizada freqüentemente por uma distribuição de Zipf [Breslau et al., 1999] ou, recentemente, pela entropia empírica do fluxo de requisição [Fonseca et al., 2003]. Correlação temporal concentra-se no modo no qual referências para um determinado objeto estão separadas através de referências para outros objetos. É caracterizado freqüentemente pelo modelo de distância LRUS-tack [Mahanti et al., 2000] ou, recentemente, pelo coeficiente de variação do número de referências para outros objetos entre dois pedidos sucessivos para um certo objeto [Fonseca et al., 2003].

Estes fatores têm uma influência decisiva no desempenho das estratégias de substituição. Então, qualquer avaliação deveria considerar a influência mútua entre carga de trabalho e estratégia de substituição. Considerando tais fatores e os resultados descritos na literatura, as seguintes regras podem ser descritas para escolher uma política de substituição adequada [Podlipnig e Böszörmenyi, 2003]:

- Estratégias de substituição deveriam conter alguma diferenciação de tamanho. Esta diferenciação de tamanho pode ser incorporada em um valor numa função, como é feito na maioria das estratégias baseadas em função. Isto complica a administração de uma lista ordenada de objetos colocados no cache. Então, se o proxy tiver menos recursos de CPU, uma tal diferenciação também poderia ter muita influência. Uma alternativa simples é separar o cache em um número pequeno de caches. Cada cache controla os objetos que têm o tamanho situado em um intervalo específico. Os objetos são inseridos nos seus caches correspondentes e administrados com uma estratégia simples de substituição. Exemplos para este procedimento são PSS e suas variações ou caching particionado.
- Como descrito anteriormente, a complexidade da avaliação de valores do objeto, não apenas nas políticas baseadas em tamanho, pode ser um fator decisivo em relação a

uma estratégia de substituição. Proxies que são do tipo CPU-bound não deveriam integrar estratégias complexas, como substituições baseadas em função. Alternativas menos complexas para isso são a LRU e certas variações dela.

- Se o espaço em cache é muito pequeno comparado ao tamanho total de todo objeto Web visto em uma carga de trabalho, a estratégia de substituição deveria favorecer a recenticidade em detrimento da frequência. Isto deve-se ao fato que correlações temporais entre pedidos para o mesmo objeto existem dentro de um curto período.
- Se o espaço do cache é grande o bastante, com poucos Gigabytes para cargas de trabalho atuais, a frequência deveria ser incorporada a longo prazo na decisão de caching porque a popularidade do documento é a principal contribuinte para a localidade temporal. A frequência sempre deveria ser combinada com um mecanismo de envelhecimento para reduzir o efeito da poluição do cache.

Estas regras podem ser vistas como diretrizes gerais. Realmente, o desempenho de uma estratégia de substituição em cache depende intensamente da carga de trabalho vista. Interessante seria um estudo a respeito de caches adaptáveis para cargas de trabalho diferentes.

Conclusão

7.1 Análise crítica da revisão bibliográfica

A World Wide Web (WWW ou Web) pode ser vista como o maior e mais acessado sistema de informação distribuído da Internet. Essa característica leva a uma considerável sobrecarga na Internet e nos servidores de página, o que gera um tempo de resposta, as vezes, inaceitável a requisições feitas pelo usuário. Conseqüentemente, em função da evolução da Web, requerem-se redes com maior velocidade de transmissão de dados. Por esse motivo, o Capítulo 2 fez uma introdução sobre o que é a Web. Com o crescimento da utilização da Internet e o surgimento de novas aplicações, cresce a cada dia a quantidade de dados a ser transmitida, tornando os meios de comunicação disponíveis cada vez mais saturados, sendo que grande parte desse tráfego é formado, muitas vezes, pela passagem de várias cópias de uma mesma informação. É justamente nesse ponto em que entra o cache, discutido no Capítulo 3, que tem por finalidade diminuir tal problema ao fazer uso de dados previamente consultados ao invés de buscá-los na origem sempre que forem requisitados. Já no Capítulo 4 foram mostradas as políticas de substituição existentes para transferência de objetos em cache, onde pôde ser observado que não existe uma política que possa ser apontada como geral e que traga sempre os melhores resultados. O Capítulo 5 apresentou alguns conceitos básicos de probabilidade e estatística, que ajudam a avaliar adequadamente os resultados de uma simulação. Também foram apresentados os principais conceitos de simulação e mostrados os parâmetros e o ambiente de simulação que será utilizados. O conceito matemático a

respeito de médias móveis foi mostrado, para que a política MeMoExp proposta no Capítulo 6 pudesse ser entendida. É no Capítulo 6 que estão os resultados obtidos com a utilização da nova política.

7.2 Contribuições e Conclusões

Este trabalho propôs uma nova política usando os conceitos de Média Móvel Exponencial. Com o uso de tais conceitos, pôde-se utilizar como política de substituição um algoritmo que usou as quantidades de requisições feitas a um objeto em um determinado período.

A política proposta, pelas suas características, também poderia ser utilizada para fazer prefetching. Prefetching, ou busca antecipada, é uma técnica para se buscar informações antes que elas sejam necessárias ou solicitadas. A principal motivação é reduzir a latência percebida pelo usuário. Como a busca antecipada é mais eficaz quando a informação tiver chances de ser usada e de permanecer atualizada, deve-se focalizar os recursos presentes em servidores mais populares como também aqueles que não mudam com muita frequência.

Vale destacar que a política proposta teve resultados similares à política FBR, política considerada eficiente na literatura.

Apesar de existirem algoritmos considerados como *good-enough* para substituição em caches web, é questionável se eles serão bons o bastante para exigências futuras como, por exemplo, multimídia e QoS.

7.3 Trabalhos Futuros

- Caches Web são desenvolvidos em quase todos os pontos da rede, pois melhoram a latência quando encontram documentos Web documenta enquanto reduzem o tráfego global na rede. Para maximizar os benefícios oferecidos por caches Web, medidas freqüentes de eficiência do cache devem ser feitas. Sem se levar em conta qual métrica é considerada, a medida de eficiência de um cache requer três passos distintos. Primeiro, é necessário traces registrando o comportamento do sistema de caches. Dois diferentes métodos podem ser usados para obter esses traces. O mais simples é usar os arquivos de log que todo software de caching gera. Porém, esta técnica é demais simples porque nesses logs freqüentemente faltam informações e, o que é pior, às vezes as informações são inexatas [Davison, 1999]. Para evitar essas desvantagens, deve-se instrumentar o software; porém, isso nem sempre é possível: o código fonte pode não estar disponível ou pode requerer uma grande quantidade de trabalho. Outra aproximação para construir tais *traces* é usar monitoramento de tráfego passivo. Várias

ferramentas executam a extração do tráfego de HTTP, mas nenhum deles considera as tendências introduzidas pelos caches. Conseqüentemente, as características do tráfego que tais ferramentas vão capturar na presença de *caches* são corrompidas, conduzindo a uma medida inadequada da eficiência do *cache*. A segunda fase consiste da análise dos *traces* para extrair as métricas básicas apropriadas. O passo final combina previamente as métricas básicas extraídas para produzir a atual medida de eficiência. Ferramentas de simulação *trace-driven* [Shim et al., 1999], [Wooster e Abrams, 1997] são largamente utilizadas. Porém, tais ferramentas normalmente são dedicadas para o avaliação de um número limitado de métricas. Isto faz com que seja difícil estende-las a fim de levar em consideração métricas não pensadas anteriormente. O problema é que não existe uma única ferramenta que integra todos estas três fases [Patarin e Makpangou, 2003]. Um possível trabalho futuro seria a implementação de uma ferramenta capaz de realizar essas três fases, sem que fosse necessário a utilização de diversos softwares diferentes para cada uma delas.

- Um outro problema são as estratégias de cache que consideram o custo, que têm sido muito estudadas por vários autores. A técnica baseada em locação, usada como forma de manutenção da coerência de cache, vista no capítulo dedicado a cache na web, sugere que os algoritmos baseados em locação não sejam muito utilizados nos produtos de cache. Um motivo, é que as técnicas baseadas em locação exigem a cooperação entre caches e servidores. Um outro trabalho futuro seria um estudo mais aprofundado sobre caches cooperativos, de modo a fazer uma análise mais minuciosa da cooperação entre caches como forma de reduzir o tempo de conexão e transmissão.
- Um outro trabalho a ser feito é realizar um estudo do uso da política proposta como forma de fazer busca antecipada.
- Para complementar, existem alguns problemas associados ao uso dos dados contidos nos arquivos de *logs* de acesso. Com certa frequência, encontra-se publicado em periódicos especializados o número de *hits* que um *site* obteve em um determinado período de tempo, com o intuito de quantificar o número de acessos a determinado *site*. A medição do número de acessos de um site, baseada no número de hits, não fornece um indicador confiável, pois uma página consultada uma só vez pode gerar mais hits do que uma página que seja consultada várias vezes, mas que gere uma quantidade menor de hits. Conseqüentemente, não é recomendável utilizar o número de hits como medida para quantificar o acesso a sites da web. O processo de contagem e identificação de sessões de usuários não é preciso, pois não se pode associar com total segurança um endereço IP a um único usuário. Contar e identificar sessões de usuários fornece, apenas, uma estatística aproximada do número de usuários distintos e do número de vezes que os

respectivos acessaram um determinado *site* da *web*. Um evento que também pode afetar o processo de determinar uma sessão, bem como aumentar o número de *hits*, é a visita de um *software* do tipo *robot* (espécie de navegador automático) que faz a varredura completa de um *site*. Como se fosse um usuário comum, um *robot* também tem suas atividades registradas no arquivo de *log* de acesso. Esse tipo de *software* está normalmente associado a *sites* que disponibilizam ferramentas de busca. Alguns *softwares* para a análise de arquivos de *log* permitem que seja isolado o uso gerado por *robots* [Haigh e Megarity, 1998], contribuindo, portanto, para reduzir a incidência de erro quando da análise dos arquivos de *log* de acesso. Para complementar, existem os problemas trazidos pelo uso de *cache*. Como visto no decorrer desta dissertação, a função primordial de um *cache* de dados é permitir que usuários tenham acesso à informação de maneira otimizada. A utilização do *cache* de dados permite que as informações solicitadas pelos usuários sejam recuperadas de maneira mais veloz. Mas, em compensação, pode reduzir a significância das informações contidas nos arquivos de *log* de acesso, pois um usuário pode recuperar um determinado objeto e esta ação pode não ficar registrada no arquivo de *log* de acesso do servidor *web*.

Levando-se em conta essas considerações relativas ao uso dos arquivos de *log* de dados, torna-se evidente a necessidade de ter-se bastante cuidado sempre que for necessário gerar análises baseadas nestas informações. Um assunto importante que deve ser considerado diz respeito à escolha de uma ferramenta automatizada para a análise do *log* de acesso. No momento da escolha de uma ferramenta, é fundamental levantar alguns questionamentos como, por exemplo, se a ferramenta leva em consideração as linhas do *log* geradas por *robots* e como a ferramenta determina uma sessão de usuário. A correta escolha de uma ferramenta automatizada para a análise de *log* é decisiva para ter-se uma idéia próxima da realidade das dinâmicas de acesso a um objeto. Mediante análise dos arquivos de *log* de acesso, não se pode ter um perfil completamente preciso do acesso a *sites* da *web*, mas apenas um modelo aproximado do que acontece na realidade, pois esta é uma abordagem quantitativa que não fornece subsídios para endereçar questões de caráter qualitativo.

Um possível trabalho futuro seria a elaboração de uma ferramenta automatizada para a análise dos *logs* de acesso, capaz de satisfazer as necessidades dos pesquisadores que estudam o comportamento das cargas de trabalho.

Referências Bibliográficas

- Abrans, M., C.R.Standridge, Abdulla, G., Willians, S. e Fox, E. [1995], ‘Caching proxies: Limitations and potentials’, in *Proc. 4 th Int. World Wide Web Conference* pp. 119–133.
- Access Log Analyzers* [2002], Disponível por WWW em <http://www.uu.se/Software/Analyzers/Access-analyzers.html>. Último acesso em 12 Novembro 2002 12:35.
- Aggarwal, C. C. e Yu, P. [1997], ‘On disk caching of web objects in proxy servers’, *6th International Conference on Information and Knowledge Management* .
- Arlitt, M., Friedrich, R. e Jin, T. [2000], ‘Performance evaluation of web proxy cache replacement policies’, *Performance Evaluation* **39**, 149–164.
- Arlitt, M. e Williamson, C. [1996], ‘Web server workload characterization’, *Proc of 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* .
- Barford, P., Bestavros, A., Bradley, A. e Crovella, M. [1998], ‘Changes in web client access patterns: Characteristics and caching implications’, *World Wide Web Journal: Special Issue on World Wide Web Characterization and Performance Evaluation* pp. 15–18.
- Barish, G. e Obraczka, K. [2000], ‘World wide web caching: Trends and techniques’, *Communications Magazine, IEEE* **38**(5), 178–184.
- Bolot, J. e Hoschka, P. [1996], ‘Performance engineering of the world wide web: Application to dimensioning and cache design’, *Proc of the 5th WWW Conference* .

- Brandão, R. F. e Anido, R. D. O. [2001], ‘Um simulador paralelo para sistemas de caches para web’, *19º Simpósio Brasileiro de Redes de Computadores* pp. 480–495.
- Breslau, L., Cao, P., Fan, L., Phillips, G. e Shenker, S. [1999], ‘Web caching and zipf-like distributions: Evidence and implications’, *In Proceedings of the IEEE INFOCOM* pp. 126–134.
- Cao, P. e Aware, S. C. [1997], ‘Www proxy caching algorithms’, *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)* pp. 193–206.
- Davison, B. D. [1999], Web traffic logs: An imperfect resource for evaluation, *in* ‘Proceedings of the INET’99 Conference’.
- URL:** citeseer.nj.nec.com/davison99web.html
- Davison, B. D. [2001], ‘A web caching primer’, *IEEE Internet Comput.* **5**(4), 38–45.
- Dilley, J., Arlitt, M. e Perret, S. [1999], ‘Enhancement and validation of squid’s cache replacement policy’, *HP Laboratories* .
- Douglis, F., Feldmann, A., Krishnamurthy, B. e Mogul, J. C. [1997], ‘Rate of change and other metrics: a live study of the world wide web’, *In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS ’97)* .
- Downey, A. B. [2001], ‘The structural cause of file size distributions’, *SIGMETRICS/Performance* pp. 328–329.
- URL:** citeseer.ist.psu.edu/downey01structural.html
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. e Berners-Lee, T. [1999], ‘Hypertext transfer protocol - http 1.1’, *Network Working Group* .
- URL:** <http://www.ietf.org/rfc/rfc2616.txt>
- Fonseca, R., Almeida, V., Crovella, M. e Abrahao, B. [2003], ‘On the intrinsic locality properties of web reference streams’, *In Proceedings of the IEEE INFOCOM* .
- Gwertzman, J. e Seltzer, M. [1996], ‘World-wide web cache consistency’, *In Proceedings of the USENIX Technical Conference San Diego CA* pp. 141–152.
- Haigh, S. e Megarity, J. [1998], ‘Measuring web site usage: log file analysis’, *Network Notes* n. 57. Disponível em: <http://www.nlc-bnc.ca/publications/1/p1-256-e.html>.
- Intel [2003], ‘Intel’.
- URL:** www.intel.com.br/portugues/eBusiness/technology/manage/1/hi20010.htm

- Jin, S. e Bestavros, A. [2000], ‘Greedydual web caching algorithm: Exploiting the two sources of temporal locality in web request streams’, *In Proceedings of the 5th International Web Caching and Content Delivery Workshop* .
URL: citeseer.nj.nec.com/jin00greedydual.html
- Jonack, M. A. e Murta, C. D. [2002], ‘Caracterização de carga de caches da www’, *Sociedade Brasileira de Computação - Revista Eletrônica de Iniciação Científica* .
- Kamienski, C. A., Sadok, D., Cavalcanti, D. A. T., Souza, D. M. T. e Dias, K. L. [2002], ‘Simulando a internet: Aplicações na pesquisa e no ensino’, *21ª JAI - Jornada de Atualização em Informática* pp. 97–138.
- Krishnamurthy, B. e Rexford, J. [2001], *Web Protocols and Practice*, Editora Campus.
- Levy, P. [1993], *As tecnologias da inteligência - o futuro do pensamento na era da informática.*, São Paulo Editora.
- Mahanti, A., Eager, D. L. e Williamson, C. L. [2000], ‘Temporal locality and its impact on web proxy cache performance’, *Performance Evaluation* **42**(2-3), 187–203.
URL: citeseer.ist.psu.edu/mahanti00temporal.html
- Malpani, R., Lorch, J. e Berger, D. [1995], ‘Making world wide caching servers coooperate’, *in Proc. 4 th Int. World Wide Web Conference* pp. 107–110.
- Markatos, E. P. [1996], ‘Main memory caching of web documents’, *In Proceedings of the Fifth International WWW Conference* .
- Murta, C. D. e Almeida, V. A. F. [1999], ‘Caches na www: Limitações e potencial’, *SBAC-PAD 99 - 11th Symposium on Computer Architecture and High Performance Computing* .
- Murta, C. D., Almeida, V. e Jr, W. M. [1998], ‘Analyzing performance of partitioned caches for the www’, *Proceedings of the 3rd International WWW Caching Workshop* .
- Patarin, S. e Makpangou, M. [2003], ‘Continuous measurement of web proxy cache efficiency’, *WWW2003 - ACM* .
URL: citeseer.nj.nec.com/patarin03continuous.html
- Pinheiro, J. C. [2001], Avaliação de políticas de substituição de objetos em caches na web, Master’s thesis, Instituto de Ciências Matemáticas e de Computação de São Carlos - USP.

- Podlipnig, S. e Böszörményi, L. [2003], ‘A survey of web cache replacement strategies’, *ACM Computing Surveys* **35**(4), 374–398.
- Powell, T., Jones, D. e Cutts, D. [1998], *Web Site Engineering: Beyond Web Page Design*, Prentice Hall PTR.
- Rizzo, L. e Vicisano, L. [2000], ‘Replacement policies for a proxy cache’, *IEEE/ACM Transactions on Networking* **8**(2), 158–170.
- Shim, J., Scheuermann, P. e Vingralek, R. [1999], ‘Proxy cache algorithms: Design, implementation, and performance’, *Knowledge and Data Engineering* **11**(4), 549–562.
URL: citeseer.nj.nec.com/shim99proxy.html
- Stallings, W. [2003], *Arquitetura e Organização de Computadores*, 5a edn, Prentice Hall.
- Tanenbaum, A. S. [1997], *Redes de Computadores*, Editora Campus.
- Tauscher, L. e Greenberg, S. [1997], ‘How people revisit web pages: Empirical findings and implications for the design of history systems’, *International Journal of Human Computer Studies* pp. 97–138.
- Williams, S., Abrams, M., Standbridge, C. R., Abdulla, G. e Fox, E. A. [1996], ‘Removal policies in network caches for world-wide web documents’, *In Proceedings of the ACM SIGCOMM Conference* pp. 293–305.
- Wooster, R. P. e Abrams, M. [1997], ‘Proxy caching that estimates page load delays’, *Proceedings of the 6th World Wide Web Conference*.