UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SIMULADOR PARA ANÁLISE DE DESEMPENHO DE POLÍTICAS DE GERENCIAMENTO DE SERVIDORES WEB CACHE

EDSON ROBERTO SOUZA PAES

FLORIANÓPOLIS (SC)
2003

EDSON ROBERTO SOUZA PAES

SIMULADOR PARA ANÁLISE DE DESEMPENHO DE POLÍTICAS DE GERENCIAMENTO DE SERVIDORES WEB CACHE

Dissertação (Tese) submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre (Doutor) em Ciências da Computação.

Orientador: Luiz Fernando Jacinto Maia, Dr.

Florianópolis

2003

FOLHA DE AVALIAÇÃO

SIMULADOR PARA ANÁLISE DE DESEMPENHO DE POLÍTICAS DE GERENCIAMENTO DE SERVIDORES WEB CACHE

Por

Edson Roberto Souza Paes

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação - PGCC.

Prof. Fernando A. Ostuni Gauthier, Dr. - Coordenador

Banca Examinadora

Prof. Luiz Fernando Jacintho Maia, Dr. - Orientador

Prof. João Bosco da Mota Alves, Dr.

Prof. Fernando A. Osthuni Gauthier, Dr

Florianópolis

2003

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

LRU	Least Recently Used
LFU	Least Frequently Used
HR	Hit Ratio
BHR	Byte Hit Ratio
HTTP	Hiper Text Transfer Protocol
TCP	Transmission Control Protocol
WWW	Word Wide Web
SQL	Structured Query Language
CGI	Commom Gateway Interface

RESUMO

A cada dia ocorre um aumento significativo da quantidade de bytes que trafegam pela Internet e os clientes estão cada vez mais exigentes, requerendo um serviço de alta qualidade. Portanto, a otimização dos serviços da Web é uma necessidade. Os servidores Cache Web são utilizados como alternativa nas empresas em que grande parte dos usuários requisitam as mesmas informações em um determinado período de tempo. Normalmente, mantêm-se uma cópia local dos objetos requisitados neste servidor, para que nas próximas requisições deste mesmo objeto, não haja a necessidade de requisitá-lo novamente. Com o aumento das requisições, aumenta o número de novos objetos a serem armazenados. A medida em que o espaço em disco for diminuindo, há a necessidade de implementar neste servidor, políticas de substituição de arquivos que mantenham a organização e manutenção do cache. De acordo com a política escolhida haverá uma maior ou menor quantidade de arquivos encontrados em requisições futuras. Neste trabalho é desenvolvido um simulador Web Cache, que a partir de log's gerados por uma empresa específica, determina-se a melhor política de substituição de arquivos a ser implantada neste local. Neste protótipo foi implementado três políticas: **SIZE, LRU** e **LFU,** que são as políticas mais utilizadas nestes servidores.

Palavras-chave: Servidor Web Cache; *HR* e *BHR*; Políticas de Substituição.

ABSTRACT

Every day, there is a significant increase of the amount of bytes that traffic through Internet and the customers are more demanding, requesting a high-quality service. Hence, the optimization of Web services is a need. The Cache Web services are used in companies, where great part of the users request the same information in one determined period of time. Normally, stored a local copy of objects requested in this server, so that in the next request of the same object , doesn't have necessity to request it again. With the increase of the requests, larger amount of new objects to be stored. In that the space destined to the storage is full, there is need to implement in the Web Cache server replacement policies of files, that maintain the organization and Cache maintenance. According to the chosen policies there will be a larger or smaller amount of files found in Cache in future requests. In this work a Web Cache Simulator is developed, that generated by a specific company starting from log's determines the best replacement policies of files to be implemented at this place. In this prototype it was implemented three policies: **SIZE, LRU** and **LFU** that are the most used politics in these server.

Word-Key: Web Cache Server, HR and BHR, Replacement Policies.

PAES, Edson Roberto Souza
Simulador para Análise de Desempenho de Políticas de Gerenciamento de Servidores Web Cache - Florianópolis, 2003.
xvi, 56f.
Dissertação (Mestrado) – Sistemas de Computação, Universidade Federal de Santa Catarina

1. Introdução 2. Desempenho na Web 3. Servidores Web Cache. 4.Algoritmos para Gerência de Espaço 5. Projeto Simulador Web Cache 6. Construção do Simulador 7. Análises e Resultados 8. Conclusão.

REFERÊNCIA BIBLIOGRÁFICA E RESUMO

Paes, Edson R S., **SIMULADOR PARA ANÁLISE DE DESEMPENHO DE POLÍTICAS DE GERENCIAMENTO DE SERVIDORES WEB CACHE.** Florianópolis, 2003. 56 páginas. Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-Graduação em Ciência da Computação – Universidade Federal de Santa Catarina.

Orientador: Dr. Luiz Fernando Jacintho Maia

Defesa: 12/03/2003

Resumo: Os servidores Cache Web são utilizados como alternativa nas empresas em que grande parte dos usuários requisitam as mesmas informações em um determinado período de tempo. Normalmente, mantêm-se uma cópia local dos objetos requisitados neste servidor, para que nas próximas requisições deste mesmo objeto, não haja a necessidade de requisitá-lo novamente.

Com o aumento das requisições, aumenta o número de novos objetos a serem armazenados. A medida em que o espaço em disco for diminuindo, há a necessidade de implementar neste servidor, políticas de substituição de arquivos que mantenham a organização e manutenção do cache. De acordo com a política escolhida haverá uma maior ou menor quantidade de arquivos encontrados em requisições futuras.

Neste trabalho é desenvolvido um simulador Web Cache, que a partir de log's gerados por uma empresa específica, determina-se a melhor política de substituição de arquivos a ser implantada neste local. Neste protótipo foi implementado três políticas: **SIZE, LRU** e **LFU,** que são as políticas mais utilizadas nestes servidores.

Palavras-chave: Servidor Web Cache; HR e BHR; Políticas de Subtituição.

1. INTRODUÇÃO

1.1 Apresentação

A velocidade de acesso à Internet, depende do desempenho e performance dos chamados *Proxy/Caches* Web. Eles auxiliam o internauta no acesso mais rápido a informações. A grande maioria dos provedores de acesso, utilizam este recurso para diminuir a taxa de utilização de seus enlaces com a Internet, o *Proxy*, é configurado para trabalhar entre um Servidor e um cliente de rede. O cliente ao invés de acessar diretamente os Servidores da Internet, acessam primeiramente o *Proxy*, o qual requisita a informação ao Servidor da Internet e armazena uma cópia desses objetos em cache. Este cache serve para disponibilizar esses objetos para os próximos clientes que os requisitarem, sem haver a necessidade de buscá-lo novamente no Servidor original.

Como o espaço disponibilizado para armazenamento de cache é finito, depois de um tempo de armazenamento de objetos é necessário liberar espaço para armazenamento de novos objetos. Existe uma certa dificuldade na determinação da política de admissão de um novo objeto, é necessário saber se este novo objeto deve ser admitido em cache e qual objeto deve ser removido para poder realizar esta operação.

A determinação desta política de admissão é responsável pelas taxas de diminuição de tráfego dos enlaces, segundo Monteiro (2001, p.21):

> Para obter um *cache* capaz de proporcionar tais benefícios é preciso analisar e definir entre as políticas de substituição de arquivos conhecidas a mais adequada a cada sistema em particular, uma vez que nenhum é igual a outro e muito pelo contrário, varia de acordo com o perfil do usuário e as cargas a que está exposto.

Em um sistema real, não se pode ficar realizando teste de mudanças de políticas na base do "achômetro", desta forma, um simulador auxilia na escolha da melhor política adotada pelo servidor cache.

Este trabalho está organizado da seguinte forma. Na seqüência deste capítulo é apresentada a definição do problema, que explica a razão pela qual o projeto foi desenvolvido,

a justificativa confirmando a importância da realização do trabalho, os objetivos e resultados que será apresentado e a metodologia adotada para a realização do projeto. No segundo capítulo é mostrado o funcionamento e diferenças das versões do protocolo HTTP (*Hyper Text Transfer Protocol*) e uma parte do funcionamento do TCP (*Transmission Control Protocol*) e a relação dos mesmos no desempenho em atender requisições na Internet. Na terceira parte do trabalho, será mostrado o que é e como funciona um servidor Web Cache, suas vantagens e desvantagens. No capítulo 4, é realizada uma breve apresentação e características de algumas políticas de substituição, inclusive as utilizadas e implementadas no Simulador desenvolvido. Já no capítulo 5, é descrito o funcionamento do Simulador e a parte técnica do projeto. No último capítulo á apresentado uma utilização prática do Simulador na análise de Log's de um provedor de Internet. Seguem a conclusão e as referências bibliográficas.

1.2 Definição do Problema

Entre os sistemas de *proxy* de Web e FTP disponíveis para Linux, o *Squid* é o mais utilizado. Segundo Fonseca (1998, p.89), "o *Squid* foi concebido para ser um servidor *proxy* eficiente, com o menor tempo de resposta possível para o usuário". Ele implementa diversas políticas de substituição de arquivos, sendo que a política padrão é a LRU.

Uma forma de avaliar se outras políticas de substituição de arquivos são adequadas às cargas impostas ao sistema em análise, é necessário definir uma nova política, coletar dados amostrais e, comparar as taxas de diminuição de tráfego com a política anteriormente adotada. Segundo Monteiro (2002, p.23), para realizar os testes desta forma "... exige que todos os arquivos armazenados em *cache* sejam eliminados, esse procedimento é necessário porque os dados foram armazenados de acordo com a política anterior, o que inviabilizaria a pesquisa se os mesmos fossem usados". No caso de um sistema em funcionamento, removendo os arquivos em cache, ocasiona um aumento significativo da taxa de utilização dos enlaces com a Internet nos primeiros dias, até que se forme um novo cache. Portanto, é essencial o desenvolvimento de um simulador que indique a política adequada com base nas cargas reais do sistema em estudo, ou seja, com base nos log's gerados pelo *Squid* ou outro Servidor Web Cache.

1.3 Justificativa

Por se tratar de um servidor onde vários usuários acessam, é necessário agilidade no acesso dos dados e para isso ser possível deve-se configurar corretamente o cache para atender esta demanda. O problema é que quando se deseja escolher a melhor política de substituição de arquivos, é necessário excluir todos os arquivos, como já foi citado anteriormente, para poder coletar dados referentes a diminuição de tráfego desta nova política, para então, poder comparar os dados com outras políticas. Desta forma, demanda um alto tempo para coletar todos os dados, ou seja, formar o novo cache de acordo com a escolha da política e só assim, poder verificar se realmente a escolha foi correta.

Com um simulador, pode-se aproveitar os log´s gerados no servidor e simular as taxas de diminuição de tráfego de acordo com diferentes políticas. Implementando no servidor a política correta.

Segundo estudos realizados por Murta e Almeida (1999, p.45) , "a importância para um bom desempenho em um servidor cache, está relacionado com a quantidade de HR (*Hit Ratio*) e BHR (Byte *Hit Ratio*)...". Diversos trabalhos relacionados estão sendo desenvolvidos, destacando-se o particionamento do Cache, onde em cada partição é aplicada uma política de substituição para melhor organizar e aproveitar o espaço dos objetos desta partição, como descreve Oliveira (1997). Com a necessidade de saber qual melhor política deve ser adotada, Brandão e Anido (1999) construíram um simulador que aplicava as políticas LRU, SIZE e Dinâmica nos arquivos log's dos servidores *cache web*.

1.4 Objetivos

1.4.1 Objetivo Geral

Implementar um simulador de *Proxy*/Cache que utiliza os arquivos log reais de um sistema e determina a melhor política de substituição de arquivos.

1.4.2 Objetivos Específicos

- Conhecer o funcionamento de um servidor *cache web*;

- Estudar e analisar as diversas políticas de substituição que pode ser aplicada ao servidor;

- Identificar, através da análise do arquivo log gerado pelo servidor, os tipos e quantidade de acessos sofridos em um determinado período;

- Analisar dados gerados e simular diferentes políticas para o mesmo log, comparando os diferentes resultados para avaliação;

- Auxiliar na tomada da decisão ao adotar uma política a ser utilizada.

1.5 Metodologia

Para conseguir analisar a maneira do armazenamento dos objetos e a freqüência que eles são acessados, é necessário analisar arquivos log que são gerados pelos servidores *Proxy/Cache*. Nestes arquivos estão contidas informações sobre as requisições. Baseado nestes dados, será construído um aplicativo para ler, identificar e analisar as informações de acordo com diversas políticas de substituição de arquivos que serão implementadas, determinando as taxas de HR e BHR.

Para isso ser possível, é preciso, primeiro de tudo, obter estes arquivos log's. Os arquivos utilizados nesta pesquisa são de provedores de internet. Para entendimento e obtenção dos resultados de forma correta, será necessário basear-se em estudos e pesquisas realizadas sobre o desempenho dos servidores *cache web* bem como suas políticas de substituição para as plataformas Linux e Windows NT. O Sistema foi implementado em ambiente *Delphi*.

1.5.1 Obtenção e Organização dos Dados para Análise

Para construir o simulador proposto, foi necessário um programa que extraia do arquivo de log, as informações necessárias para a análise. Estas informações foram armazenadas em um banco de dados que sempre será renovado e atualizado para cada aplicação. Isto é necessário para não carregar a memória e evitar altos processamentos, devido a grande quantidade de informação dos arquivos de log. Como é o caso de empresas, escolas ou até provedores de Internet. As informações armazenadas para a análise possuem os seguintes conteúdos:

- Tamanho do objeto
- Data do último acesso
- Descrição
- Endereço onde se encontra

Depois de adquirir estas informações, foi aplicado uma simulação para cada política de substituição, LRU, LFU, SIZE para posterior análise dos desempenhos. Ao final será apresentado uma comparação das políticas de substituição, determinando qual será a mais adequada para ser utilizada na empresa de onde foi extraído o arquivo de log.

2. DESEMPENHO NA WEB

Neste capítulo será apresentado como são realizadas as requisições dos objetos na Web, bem como a forma com que os servidores atendem a estas requisições e enviam respostas aos requisitantes. Existem muitos detalhes que fazem com que a velocidade dos acessos feitos pelos clientes na Web, não tenham um desempenho esperado ou desejado. Um destes fatores pode estar na versão do protocolo HTTP utilizado no servidor a ser requisitado. As diferenças e características entre o HTTP 1.0 e o HTTP 1.1 serão abordadas neste capítulo bem como uma visão geral do protocolo da camada de transporte, o TCP, uma vez que o HTTP é um protocolo do nível de aplicação do modelo TCP/IP.

2.1 O Protocolo HTTP

O protocolo HTTP é um protocolo do nível de aplicação do modelo TCP/IP, que possui o objetivo básico de transmitir informações de sistemas de informação distribuídos de hipermídia. O HTTP está sendo usado globalmente pela Internet desde 1990. Segundo Meira JR. *et al.* (2002, p.34), "Neste período, o tráfego devido a ele cresceu até atingir 80% do volume total do tráfego nos *backbones*". Atualmente há duas versões do protocolo em uso o HTTP 1.0 e HTTP 1.1, as quais possuem algumas diferenças no processo de comunicação entre cliente e Servidor.

HTTP permite um conjunto aberto de métodos para ser usado para indicar o propósito de uma requisição. A operação prevista pelo protocolo é um processo de requisição e resposta que envolve o envio pelo cliente de uma requisição para um determinado documento, identificado por uma *Uniform Resource Identifier* (URI), e uma reposta do Servidor, normalmente com o conteúdo do arquivo associada a URI. As mensagens são passadas em um formato similar ao usado pelo Internet Mail é o *Multipurpose Internet Mail Extensions* (MIME). As mensagens seguem o formato da RFC822, atualizada pelas RFC1327 e RFC0987.

HTTP também é usado como um protocolo genérico para comunicação entre agentes usuários e *proxies/gateways* com outros protocolos Internet, tais como SMTP, NNTP, FTP,

Gopher e WAIS, permitindo acesso à hipermídia para recursos disponíveis de aplicações diversas e simplificando a implementação de agentes usuários.

2.1.1 Pedido

Uma mensagem de pedido de um cliente a um servidor inclui o método a ser aplicado ao recurso, o identificador do recurso e a versão do protocolo em uso. O formato da mensagem de pedido enviada do cliente ao servidor é descrito abaixo (Figura 1), de acordo com a notação BNF.

```
Request = Simple-Request | Full-Request
Simple-Request = "Get" SP Request-URI CRLF
Full-Request = Request-Line * ( General-Header | Request-Header | Entity-Header ) CRLF [ Entity-Body ]
```

FIGURA 1 - Mensagem de Pedido HTTP.

2.1.2 Resposta

Após receber e interpretar uma mensagem de pedido, um servidor responde na forma de um mensagem de resposta HTTP, conforme é mostrado na Figura 2:

```
Response = Simple-Response | Full-Response
Simple-Response = [ Entity-Body ]
Full-Response = Status-Line * ( General-Header | Response-Header | Entity-Header ) CRLF [ Entity-Body ]
```

FIGURA 2 - Mensagem de Resposta HTTP.

A primeira linha de uma Full-Response é a Status-Line (linha de estado), consistindo da versão do protocolo, seguida de um código de status e sua frase de texto associada, com cada elemento separado pelo caracter SP. Nenhum CR (Carriage Return) ou LF (Line Feed) é permitido, exceto o CRLF final da seqüência.

Status-Line=HTTP-Version SP Status-code SP Reason-Phrase CRLF

O elemento Status-Code (Código de Status) é um inteiro de três dígitos, resultado da tentativa para entender e satisfazer o pedido. O primeiro dígito define a classe da resposta. Os

últimos dois dígitos não têm nenhuma categorização. Existem cinco valores para o primeiro dígito:

1xx: Informacional - Não usado, mas reservado para uso futuro.

2xx: Sucesso - A ação foi recebida, entendida e aceita.

3xx: Redirecionamento - Ações adicionais devem ser executadas para completar o pedido.

4xx: Erro no cliente - O pedido contém erro de sintaxe ou não pode ser completado.

5xx: Erro no servidor - O servidor falhou ao completar um pedido aparentemente válido.

A relação completa de códigos definidos pela RFC 2616, pode ser consultado no Anexo 1.

2.1.3 Entidade

As Mensagens Pedido-Completo (*Full-Request*) e Resposta-Completa (*Full-Response*) podem transferir uma entidade com alguns pedidos e respostas. Uma entidade consiste de campos:

Entity-Header (Cabeçalho da Entidade) e Entity-Body (Corpo da Endidade):
Entity-Header = Allow | Content-Encoding | Content-Lenght | Content-Type | Expires | Last-Modified extension-header
Extension-header = HTTP-header
Entity-Body = *OCTET

2.1.4 Definições de Métodos

O método a ser aplicado a um objeto define para o servidor o tipo de resposta esperada pelo cliente. O conjunto de métodos comuns é definido abaixo.

- O método GET recupera qualquer informação (na forma de uma entidade) que é identificada pelo pedido Request-Uri. Se o Request-Uri refere-se a um processo produtor de dados, ele retornará o produto do processo e não o texto fonte.

A semântica do GET pode ser mudada para uma forma condicional "*conditional Get*" se a mensagem de pedido inclui um campo de *cabeçalho "If-Modified-Since*" (IMS), esta mudança reduz o tráfego na rede, pois objetos que já estejam no cliente devido a um acesso anterior, não são transferidos novamente.

- O método HEAD é semelhante ao GET, exceto que o servidor não precisa retornar nenhuma entidade do tipo *Entity-Body* na resposta. Ou seja, retorna apenas uma meta-informação, representada na forma de campos no cabeçalho da mensagem de resposta, contendo elementos como o tamanho do objeto, a data de alteração e o formato do conteúdo, entre outros.

- O método POST é utilizado para solicitar que o servidor destino aceite informações contidas no corpo da requisição. O URL determina onde na estrutura do servidor a informação será colocada e pode ser usado pelo servidor para determinar qual tipo de operação aplicar aos dados. Segundo Zotto, "As aplicações não devem armazenar respostas ao POST em cache, pois não há meio de saber se o servidor retornará uma resposta equivalente em pedidos futuros". Existem outros métodos definidos para serem usados em requisições de clientes, entre eles o PUT, DELETE, LINK e UNLINK.

- O método PUT requisita que a entidade seja armazenada sob o *Request-Uri* do pedido. Se o *Request-Uri* refere-se a algum recurso existente, a entidade deve ser considerada como uma nova versão daquela existente no servidor de origem.

- O método DELETE requer que o servidor de origem apague o recurso identificado pelo URI.

- O método LINK estabelece uma ou mais ligações de relacionamento entre o recurso existente e identificado pelo URI outros recursos.

- O método UNLINK remove uma ou mais ligações de relacionamento entre o recurso existente pelo *Request-Uri* e outros recursos.

2.1.5 Diferenças entre o HTTP 1.0 e HTTP 1.1

Segundo Meira Jr. *et al.* (2002), na utilização do protocolo HTTP cada resposta a uma requisição é totalmente definida com base no conteúdo da mesma, sem nenhuma relação com interações anteriores entre as duas partes. Nesse sentido o protocolo é independente de estados, não armazenando qualquer tipo de informação sobre comunicações anteriores. Sendo assim, cada requisição pode ser enviada de forma independente ao servidor já que não há nenhuma exigência de que requisições consecutivas de um cliente a um servidor usem a mesma conexão. Na verdade, interações utilizando a versão 1.0 do protocolo fazem exatamente o contrário: cada nova requisição por um documento exige o estabelecimento de uma nova conexão TCP, que é encerrada ao fim da resposta do servidor. Ao buscar um documento contendo várias imagens, por exemplo, isso implica na requisição de vários arquivos, cada qual com sua URI, em conexões independentes. Isso causa o estabelecimento de várias conexões em um curto espaço de tempo, o que aumenta a carga sobre os servidores, os quais têm que processar várias conexões curtas, e sobre a rede, já que cada estabelecimento e fechamento de conexão requer a troca de várias mensagens TCP.

A versão mais recente do protocolo (HTTP 1.1) foi alterada e estendida a fim de reduzir os tempos de respostas das aplicações WWW e reduzir também o tráfego na Internet. Para isso, o protocolo HTTP 1.1 permite o uso de conexões persistentes para que um cliente mantenha a conexão aberta durante toda a sua interação com um dado servidor, de forma que apenas uma conexão tenha que ser aberta e posteriormente fechada.

2.1.6 Passos de uma Requisição WWW

Os passos principais para uma requisição WWW, será descrito a seguir numerado conforme a Figura 3.

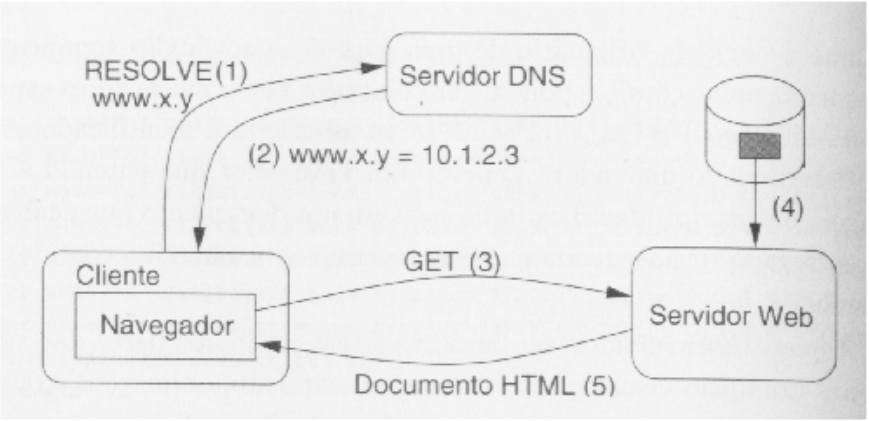


FIGURA 3 – Passos para o acesso a um documento na WWW.
Fonte: Meira JR. *et al*. (2002)

1 – O cliente, de posse de um nome para o servidor WWW desejado (www.x.y no exemplo), faz uma consulta ao serviço de DNS para obter o endereço IP correspondente ao servidor.

2 – O servidor DNS consultado pode ter que repassar a consulta a outros servidores intermediários. A resposta será obtida, direta ou indiretamente, do servidor responsável pelo domínio onde está localizado o servidor www.x.y. O endereço obtido é então devolvido ao cliente.

3 – O navegador do cliente nesse momento, de posse do endereço IP do servidor, pode abrir uma conexão TCP para o mesmo. Caso a URL indique a necessidade de uma conexão segura (o protocolo indicado seja o HTTPS), a troca de mensagens para estabelecimento de um canal de comunicação de criptografia também será feita. Finalmente, o navegador envia uma mensagem HTTP especificando o método GET e fornecendo a URL do documento desejado.

4 – Ao receber a URL, o servidor extrai da mesma o identificador do arquivo dentro da máquina e busca o seu conteúdo no disco.

5 – O conteúdo HTML do arquivo é encapsulado em uma mensagem HTTP e devolvido para o navegador de cliente, que pode finalmente exibi-lo

2.2 Protocolo TCP (Transmission Control Protocol)

O TCP é um protocolo da camada de transporte da arquitetura Internet TCP/IP. O protocolo é orientado a conexão e fornece um serviço confiável de transferência de arquivos fim-a-fim. Ele é responsável por inserir as mensagens das aplicações dentro do datagrama de transporte, reenviar datagramas perdidos e ordenar a chegada de datagramas enviados por outro micro. O TCP foi projetado para funcionar com base em um serviço de rede sem conexão e sem confirmação, fornecido pelo protocolo IP. O protocolo TCP interage de um lado com processos das aplicações e do outro com o protocolo da camada de rede da arquitetura Internet. A interface entre o protocolo e a camada superior consiste em um conjunto de chamadas. Existem chamadas, por exemplo, para abrir e fechar conexões e para enviar e receber dados em conexões previamente estabelecidas. Já a interface entre o TCP e a camada inferior define um mecanismo através do qual as duas camadas trocam informações assincronamente.

Este protocolo é capaz de transferir uma cadeia (stream) contínua de octetos, nas duas direções, entre seus usuários. Segundo Soares (1997, 13),

> Normalmente o próprio protocolo decide o momento de parar de agrupar os octetos e de, conseqüentemente, transmitir o segmento formado por esse agrupamento. Porém, caso seja necessário, o usuário do TCP pode requerer a transmissão imediata dos octetos que estão no buffer de transmissão, através da função push.

Conforme mencionado, o protocolo TCP não exige um serviço de rede confiável para operar, logo, responsabiliza-se pela recuperação de dados corrompidos, perdidos, duplicados ou entregues fora de ordem pelo protocolo de rede. Isto é feito associando-se cada octeto a um número de seqüência. O número de seqüência do primeiro octeto dos dados contidos em um segmento é transmitido junto com o segmento e é denominado número de seqüência do segmento. Os segmentos carregam "de carona" (piggybacking) um reconhecimento.

O reconhecimento constitui-se do número de seqüência do próximo octeto que a entidade TCP transmissora espera receber da entidade TCP receptora, na direção oposta da conexão. Por exemplo, se o número de seqüência X for transmitido no campo Acknowledge (ACK), ele indica que a estação TCP transmissora recebeu corretamente os octetos com

número de seqüência menores que X, e que ele espera receber o octeto X na próxima mensagem (SOARES, 1997).

2.2.1 Mecanismos de Controle de Congestionamento - TCP

Nagle (1984, *apud* Lima e Fonseca, 2002, p.42), "na fase inicial da Internet, foi um dos primeiros a verificar que os roteadores são vulneráveis a um fenômeno que foi denominado como 'Colapso de Congestionamento'". Tal fenômeno, apesar de ser raro, pode ocorrer quando o tráfego é intenso e a rede está congestionada, levando ao descarte de pacotes nos roteadores e consequentemente retransmissões, aumentando assim o RTT *(Round Trip Time*). Quando pacotes são descartados, os recursos que foram por eles utilizados são desperdiçados. Além disto, quando estes pacotes são retransmitidos, novos recursos são alocados.

Um outro problema é que o descarte de pacotes diminui a vazão e aumenta o atraso dos tráfegos na rede. Evitar a ocorrência de congestionamento e controlá-lo são de capital importância para a operação da rede. Em redes *best-effort* o congestionamento pode degradar ainda mais os baixos níveis de qualidade de serviço. Ainda segundo Lima e Fonseca (2002, 34), "...em redes com suporte a QoS, o congestionamento pode inviabilizar o compromisso de oferecimento de QoS".

A primeira solução proposta para tratar do problema de congestionamento foi apresentada para redes TCP/IP, por Jacobson que desenvolveu um mecanismo que faz com que as máquinas envolvidas em uma comunicação TCP diminuam a transmissão de pacotes na ocorrência de congestionamento. Este mecanismo é composto de dois algoritmos: *Slow Start* e *Congestion Avoidanc*e, que na prática são implementados em conjunto. A partir de então, procurou-se melhorar a eficiência dos mecanismos de prevenção e de controle de congestionamento, propondo-se dois outros algoritmos que foram acrescentados ao mecanismo já existente: *Fast Retransmit* e *Fast Recover*y. O mecanismo, composto dos quatro algoritmos (TCP *Reno*, TCP *Vegas*, TCP *Sack*, TCP *New Reno*), fazem parte do controle de congestionamento TCP, para maiores detalhes consulte Lima e Fonseca (2002).

3. SERVIDORES WEB CACHE

Segundo Lima e Fonseca (2002), " O serviço oferecido atualmente pela Internet é o tipo de serviço mais simples que pode ser oferecido por uma rede: o serviço melhor-esforço *(best-effor*t)". Este serviço não oferece nenhuma garantia às aplicações, ou seja, há apenas um compromisso de se tentar entregar ao receptor a informação enviada pelo emissor.

O controle de tráfego, é um conjunto de políticas e mecanismos utilizados para satisfazer os Requisitos de Qualidade de Serviço (QoS) das aplicações, gerenciando e otimizando os recursos da rede de forma eficiente.

Recentemente as pesquisas que buscam otimizar o tráfego da Internet, caminham em diversas direções sendo que as principais estão relacionadas a alterações no HTTP na interação com o protocolo TCP. Outras propostas estão na implementação de *Proxy*/Cache no cliente e nos backbones da rede. Segundo Nielsen *et al.* (1997, p.34), "as alterações realizadas no protocolo HTTP 1.1 que permitem conexões persistentes, reduzem o uso da rede em 10 vezes". Segundo Almeida *et al.* (1996, p.56), "muitos clientes na mesma área podem requerer o mesmo arquivo do mesmo servidor durante um pequeno intervalo de tempo" e com esta evolução do HTTP, o número de sucesso na entrega dos pacotes é muito maior que em sua versão anterior.

Para o usuário da Internet, tempo de resposta é o fator fundamental. Sendo que QoS para a grande maioria destes usuários está vinculado a esta métrica. Este tempo de resposta pode ser interferido ou até prejudicado por diversos motivos, como demora de resposta do servidor à uma requisição, congestionamento na rede, distância física entre cliente e servidor, tamanho da banda utilizada. Para a maioria dos usuários que ainda acessam a Internet através de um modem, é a largura da banda, que prejudica e muito o acesso dos usuários, principalmente se houverem várias requisições ao mesmo tempo para esta banda.

Para amenizar este problema, pode ser utilizado um *proxy*. O *proxy* é responsável em organizar e armazenar os objetos pedidos nas requisições de acordo com sua polícia de substituição. Este servidor de cache, verifica a cada requisição, se ele possui uma cópia do objeto requisitado. Caso tenha, ele repassa ao cliente requisitante este objeto que ele possui

armazenado, sem precisar que o cliente requisite diretamente no servidor onde se encontra o objeto. Caso ele não possua este objeto, ele se encarrega de acessar o servidor onde se encontra o objeto da requisição, e armazena o objeto para futuras requisições do mesmo ou de outros clientes. O fluxo de uma requisição que passa por um *proxy/cache* pode ser visualizada na Figura 4:

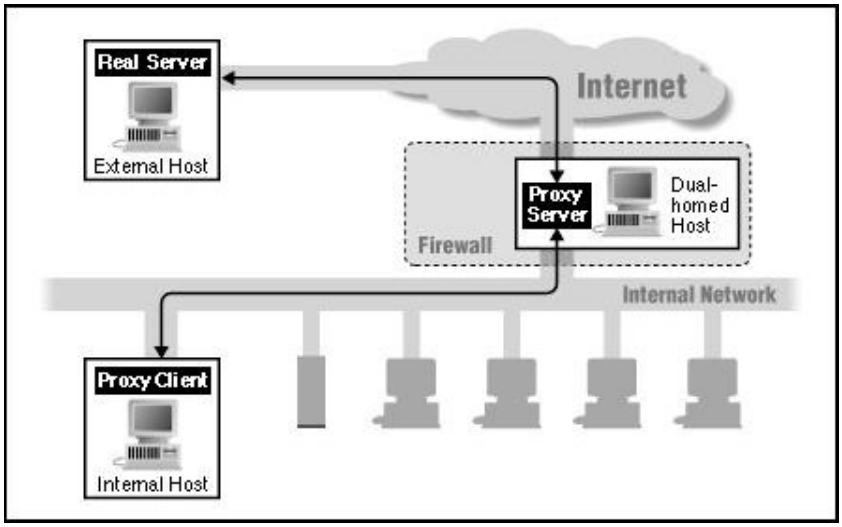


FIGURA 4 - Fluxo de uma requisição que passa por um proxy/cache.

A latência de uma requisição Web é influenciada por três fatores, cada um contribuindo com seus custos: o cliente, o servidor e a rede. O custo do cliente é devido ao tempo necessário para o browser mostrar os objetos requisitados pelo usuário. O custo do servidor é referente ao processamento da requisição e envio da informação pedida. Por último, o custo da rede é o tempo necessário para prover a conexão remota ao servidor Web acrescida do tempo de transmissão dos dados requisitados. Em suma, a latência de uma requisição Web é a soma destes três custos. O uso de servidores *cache web* reduz a latência de rede e a latência do servidor Web, diminuindo o tempo de resposta para o usuário final.

Além dos benefícios de redução da latência das requisições *We*b, o uso de *proxy* permite o acesso à *Internet* em situações de alto congestionamento dos canais ou até mesmo de indisponibilidade dos mesmos, já que os servidores *proxy* continuam a responder a seus usuários com a última versão gravada dos documentos pedidos. O usuário recebe os objetos de forma transparente, sem ficar sabendo que houve um problema com o acesso à Internet. A porcentagem das requisições que consegue ser resolvida pelo servidor *cache* apenas com os

dados armazenados localmente (taxa de acerto ou *Hit Ratio*) varia de acordo com alguns fatores, tais como: a capacidade de armazenamento de objetos e a influência da comunidade atendida pelo *proxy*.

Os algoritmos para a implementação inicial de caches na WWW foram baseados nos estudos sobre sistemas de cache em arquitetura de computadores e sistemas operacionais. Embora haja uma consolidação da pesquisa sobre algoritmos de troca de blocos e páginas nestes ambientes tradicionais, sistemas de cache na Web são mais complexos e, devido às suas peculiaridades, compõem um tema a ser estudado separadamente. Por exemplo, enquanto os caches tradicionais existentes nos sistemas de computação tratam de objetos de tamanho único, os sistemas de cache na Web operam com objetos de tamanhos extremamente variáveis.

Os sistemas de *cache web* têm o mesmo princípio de funcionamento do cache tradicional. Os documentos requisitados são copiados em posições mais próximas do usuário com o intuito de diminuir o tempo do próximo acesso ao documento. Os caches podem ser implementados nos vários componentes do sistema: cache de cliente, implementado no browser, cache de servidor, implementado no próprio servidor, e cache de *proxy*, implementado em pontos estratégicos da rede, entre os demais caches, geralmente em pontos mais próximos dos clientes. O objetivo do cache de *proxy* é atender a um grande conjunto de clientes que utilizam a mesma saída (backbone) para a Internet.

Apesar da semelhança funcional, os sistemas de cache tradicional e de cache na Web apresentam características um pouco diferentes. Nos esquemas tradicionais, os dados são movidos em blocos do mesmo tamanho. Portanto, o número de bytes encontrados no cache em relação ao número de bytes requisitados é exatamente a taxa de *hit*s. O mesmo não ocorre nos caches da Web porque os documentos são transmitidos e armazenados na sua forma integral; não há o conceito de bloco. Como consequência, é preciso trabalhar com duas métricas para medir o desempenho dos sistemas de cache na Web: a fração das requisições atendidas pelo cache, (*Hit Rate*, *HR*) e a fração dos bytes requisitados que é atendida pelo cache, (*Byte Hit Rate*, *BHR*), conforme citado anteriormente.

A otimização de cada métrica, separadamente, atende à primeira vista a interesses diferentes. Usuários Web querem ter respostas rápidas às suas requisições, por isso são atraídos por uma elevada taxa de *hits*. Administradores de servidores Web e da Internet querem economia na utilização da rede, portanto estão interessados em aumentar a taxa de acertos por byte, reduzindo o volume de tráfego na rede. No entanto, a otimização das duas métricas é interessante, pois a redução do número de requisições aos servidores (maior HR) também reduz o volume do tráfego além de diminuir a carga no servidor destino da conexão.

Por outro lado, o descongestionamento da rede (maior BHR) influencia fortemente o tempo de resposta, uma vez que as requisições não presentes nos caches poderão ser respondidas mais rapidamente do que quando há sobrecarga na rede. Portanto, o aumento de ambas as métricas, HR e BHR, ajuda a conter o tráfego na rede, evitando gasto desnecessário de recursos (largura de banda), além de otimizar o tempo de resposta.

No entanto, a maximização simultânea das duas frações, HR e BHR encontra um limite de compromisso entre as duas métricas. Este comportamento é decorrente da característica de grande variabilidade nos tamanhos dos documentos associada à sua popularidade. Para otimizar o HR, por exemplo, basta armazenar os arquivos menores, que são os mais populares. Além disto, o armazenamento exclusivo de arquivos pequenos permite que mais arquivos caibam no cache. Os arquivos pequenos são responsáveis por um número grande de *hits* mas constituem uma pequena fração dos bytes servidos. Por isso, a otimização do HR implica num aumento do número de *miss* por byte pedido. Por outro lado, os arquivos maiores são responsáveis por um pequeno número de *hits* que constituem uma grande fração dos bytes requisitados. O aumento do BHR pode levar a uma diminuição do HR porque implica no armazenamento de documentos maiores, que podem ocupar o lugar de vários documentos pequenos, porém mais populares.

3.1 Importância do uso de Cache Web

A importância do uso de *cache web*, tanto em nível local, como estadual, nacional e até internacional se evidencia pelo fato de que, para o usuário final, a informação solicitada chegará com maior rapidez, sem passar por congestionamentos nos elos nacionais e/ou internacionais. Tudo isso com 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.

Além disso, o uso do cache traz algumas outras 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.

3.2 Problemas no uso de Caches

Os documentos requisitados são obtidos do servidor remoto e armazenado localmente para uso futuro. Se uma versão atualizada do documento é encontrado no cache do *proxy*, nenhuma conexão ao servidor remoto é necessária. 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 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 heurísticas, que fornecem tão somente uma estimativa grosseira do tempo de vida do objeto. Desde que muitos documentos Web são documentos ”vivos”, 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 informada na data de expiração.

3.3 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, sejam garantidas, sempre que possível.

Alguns desses objetos nunca podem ser mantidos em cache, como os programas 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, etc. Para os contadores de acesso, recomenda-se colocá-lo em um pequeno documento e transformando-o em um objeto não disponível para o cache (através do seu cabeçalho).

4. ALGORITMOS PARA GERÊNCIA DE ESPAÇO

A análise do tráfego gerado pela Web revela características importantes para o seu desempenho, que devem ser consideradas no projeto de seus componentes. Uma característica essencial é a extrema variabilidade nos tamanhos dos documentos que circulam na Web. Há documentos que são várias ordens de grandeza maiores do que outros. Como conseqüência, o armazenamento de apenas um documento grande no cache pode significar a retirada de centenas ou milhares de documentos pequenos, alterando completamente o estado do cache.
Políticas de substituição, segundo Oliveira (2002), são algoritmos executados para escolher um documento a ser removido do cache para abrir espaço para o armazenamento de um novo documento, quando o espaço livre é insuficiente para armazenar o novo documento. Para descobrir qual a melhor configuração de parâmetros relativos a um determinado sistema de caches para Web, devem-se testar várias combinações de valores para os parâmetros de forma a determinar qual dessas combinações possibilita o melhor desempenho.

A gerência de espaço de caches trata de duas questões: a organização do espaço disponível e a política de substituição (ou reposição), que define quais arquivos devem ser retirados para que um seja armazenado. Associada à política de substituição podemos ter uma política de controle de admissão para definir se um documento requisitado não presente no cache vai ser armazenado.

Como conseqüência da caracterização apresentada na seção anterior, os algoritmos de reposição de documentos nos caches na Web procuram levar em consideração os parâmetros da carga que afetam o desempenho, a saber: o tamanho dos documentos, o tempo para buscar um documento em caso de *miss*, a freqüência de acesso e a data de acesso aos documentos. A identificação destes parâmetros indica que a gerência de espaço nos caches da Web é um problema mais complexo do que a gerência de espaço em caches de memória.

Os próximos itens descrevem alguns exemplos de política de substituição e suas características:

4.1 Size

Este algoritmo remove primeiramente do cache, quando ao chegar de um novo objeto, o maior objeto em tamanho, de forma que possa criar espaço suficiente para o armazenamento do novo objeto, sendo que, ao remover o maior objeto em tamanho e ainda não tenha sido criado espaço suficiente para entrada de novo objeto, deve-se retirar o próximo objeto maior, e assim sucessivamente até que o espaço seja suficiente (AGGARWAL *et al.,* 1999, *apud* CALSAVARA E SANTOS).

SIZE tem a vantagem de reter os documentos menores, responsáveis pelo maior número de *hits*, mas é mais sensível à variações na relação entre tamanho e número de acesso, além de reter documentos pequenos que não podem ser mais referenciados. Documentos pequenos sem muita requisição de acesso, podem ficar mantidos no cache por um longo período. Assim como a política LRU, essa política possui a vantagem de não necessitar de parametrização (RLITT *et al.,* 2000, a*pud* OLIVEIRA, 2002). Essa política, que tende a reter os objetos menores, responsáveis pelo maior número de acerto no cache, gera a possibilidade de reter objetos pequenos indefinidamente que podem não ser mais referenciados, se nenhum mecanismo de expiração for utilizado.

4.2 Least-Recently-Used (LRU)

Esta política privilegia as referências mais recentes, protegendo-as da substituição. Assim, ela explora o princípio da localidade temporal que diz que documentos recentemente acessados têm maior prioridade de serem requisitados em futuro próximo. A localidade temporal foi observada em seqüência de acessos a códigos e dados de programas. Neste ambiente de cache tradicional, a reposição de blocos fica a cargo do algoritmo LRU por ser baseado no princípio da localidade (AGGARWAL *et al.,* 1999, *apud* CALSAVARA e SANTOS).

Essa política é implementada usando uma lista ordenada pelo último tempo de acesso. Adicionar ou remover elementos nessa lista é feito em tempo constante, 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 cachê. (DILLEY *et al,.*1999, *apud* OLIVEIRA 2002).

Segundo LIU *et al.* "O LRU mostrou-se eficiente para memória cache conforme demonstrado no trabalhos de SMITH (1978, 1991), no entanto, esta política não é boa para *Cache web*, como demonstrado em muitos trabalhos de simulação, devido a variabilidade no tamanho dos arquivos.

4.3 Least-Frequently-Used (LFU)

O objetivo dete método é retirar os objetos utilizados com menor freqüência, ignorando a data dos acessos, com isso podem ser retirados os objetos recém-acessados que serão mais requisitados no futuro. Por outro lado, o algoritmo preserva objetos mais referenciados, que é uma vantagem. Porém, pode preservar eternamente objetos muito referenciados no futuro. (AGGARWAL *et al.,* 1999, *apud* CALSAVARA e SANTOS).

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. (DILLEY *et al.,* 1999, *apud* OLIVEIRA, 2002).

4.4 Least Normalized Cost Replecement (LNC-R)

Maximizar o tempo de resposta é a sua finalidade. Para isso, é estimada a probabilidade de uma futura consulta a um determinado objeto, usando uma movimentação média dos últimos tempos de chegadas das requisições do objeto. Contudo, foi observado que os clientes da Internet tem preferência por acessar objetos pequenos. E por isso, a informação de domínio especifico é baseada em estimativas da probabilidade de uma futura consulta no padrão de referência do objeto como no tamanho do objeto. (SCHEUERMANN *et al.,* 1997, *apud* CALSAVARA e SANTOS).

4.5 Lowest Relative Value (LRV)

Este algoritmo é baseado na data de acesso, tamanho e freqüência do objeto. Os autores encontram funções matemáticas que são aproximações da distribuição dos tempos entre acessos previamente i vezes. Estas funções são baseadas em uma extensa caracterização de duas cargas de cache de rede. O cache acumula, em tempo de execução, estatística de cada objeto requisitado, atualizadas a cada acesso. A principal desvantagem do LRV, além do seu custo proibitivo para implementação é a sua grande parametrização com base em apenas duas cargas, o que deixa dúvidas sobre o bom desempenho deste algoritmo com cargas diferentes. (RIZZO e VICISANO, 1998, *apud* CALSAVARA e SANTOS).

4.6 Lru Min

Baseia-se através da variação do LRU, o qual verifica o tamanho do objeto a ser armazenado e procura no cache um objeto com tamanho maior ou igual. Caso existam documentos que atendem a estes requisitos, o algoritmo LRU é aplicado a este conjunto. Caso contrário, todos os objetos que maiores ou iguais à metade do tamanho do objeto a ser armazenado, são incluídos no subconjunto. A operação é repetida até que o espaço necessário seja liberado. (WILLIAMS *et al.,* 1996, *apud* CALSAVARA e SANTOS).

4.7 Greedydual-Size (GD-SIZE)

É um algoritmo híbrido que considera a data de acesso, tamanho e custo de busca de um objeto. Os autores mostram que considerar a latência não leva a bons resultados devido a grande variabilidade de latência de busca de um mesmo objeto, o que confirma resultados anteriores. O algoritmo ordena os objetos de acordo com um valor H definido por H = custo/tamanho. Objetos com menor valor H são candidatos a sair do cache. A recenticidade é considerada da seguinte forma. A cada acesso, o valor h do objeto é calculado e acumulado ao valor residual anterior. Portanto, quanto mais recente o acesso do objeto, maior o seu H e menor sua probabilidade de sair do cache. (CAO e IRANI, 1997, *apud* CALSAVARA e SANTOS).

4.8 Lowest–Latency-First (LLF)

Este método substitui primeiro o objeto que teve menor latência no acesso via rede. No entanto, o mesmo trabalho mostra que é um bom critério. Em sistemas com grande variabilidade, onde picos numa medida impactam negativamente o desempenho, considerar apenas a média para representar os dados pode levar a conclusões errôneas. Portanto, a utilização destes tempos deve ser feita por meio de valores médios acompanhados para medidas de dispersão dos dados, o que pode dificultar a sua utilização. (WOOSTER e ABRAMS, 1997, *apud* CALSAVARA e SANTOS).

4.9 Hybrid

Seu objetivo é calcular, para cada objeto, seu valor para o cache. Este cálculo é feito por uma função que combina os parâmetros tempo de conexão ao servidor do objeto, *bandwidth* (largura de banda) da conexão, a freqüência e o tamanho do objeto. Objetos com menor valor são descartados. O algoritmo apresentou bom desempenho em HR e tempo de resposta frente a LRU, LFU, e SIZE. Suas desvantagens são a necessidade de armazenar mais informações para cada objeto e a necessidade de ajustes cuidadosos das constantes utilizadas na função. (WOOSTER e ABRAMS, 1997, *apud* CALSAVARA e SANTOS).

4.10 Least Dynamic Frequency Rule (LDRF)

Essa política usa a freqüência 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 freqüência de acesso é escolhido para ser retirado. Esse esquema é chamado de LDRF e é uma generalização direta da política LRU 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 freqüência de cada objeto no cachê. (AGGARWAL e YU, 1997, *apud* OLIVEIRA, 2002).

4.11 Localized Least Dynamic Frequency (LLDR)

Para minimizar o tempo gasto para verificação da freqüência 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 freqüência de acesso aos objetos no cache. Tal freqüência é comparada com a freqüência de referência do objeto a ser admitido.

4.12 Cache Particionado

Este algoritmo divide o espaço do cache em partições. Cada partição é dedicada ao armazenamento de objetos cujos tamanhos pertencem a um intervalo. Cada intervalo define uma classe de objetos. As classes são em número igual ao número de partições e cobrem todos os tamanhos possíveis de objetos, sem sobreposição. Substituição dos objetos são feitas apenas entre objetos de uma mesma classe. Este modelo não permite nem retirada de um objeto pequeno para a inserção de um objeto grande, pois não é possível fazer substituição entre objetos com tamanhos extremos. A proposta de dividir em partições que armazenem classes de objetos com tamanhos similares desdobra a gerência de espaço de caches em dois campos: a organização dos espaços e a política de substituição. A regra de ordenação é dada pela política de reposição. Por exemplo, a política LRU gera uma lista LRU. A política LFU gera uma lista cuja chave de ordenação é o número de acessos a cada objeto. A lista é atualizada a cada requisição. A atualização consiste na reordenação da lista em caso de *hit* ou na inserção com possibilidade de retirada em caso de *miss*. Cada inserção envolve no máximo uma retirada. Portanto o estado do cache muda lentamente, é quase estacionário, e a correlação entre o estado do cache no passado imediato é alta. Através do particionamento foi possível provar que o desempenho em HR e BHR é a função do tamanho médio dos arquivos solicitados e do tamanho médio dos *hits*. (MURTA *et al.,* 1998, *apud* CALSAVARA e SANTOS).

4.13 First-In, First-Out (FIFO)

Faz substituições de objetos que chegam em primeiro no cache. (SILBERSCHETZ e GALVIN, 1994, *apud* CALSAVARA e SANTOS).

4.14 Least Semantically Related (LSR)

O princípio do algoritmo LSR é dar prioridade aos objetos que possuam maior relação semântica com o objeto que esteja entrando no cache, isto é, os objetos com maior relação semântica com o novo objeto tenderão a permanecer no cache. Por isso, o algoritmo analisa os objetos alocados em cache e os organiza de acordo com sua semântica. Assim, quando um novo objeto n estiver sendo inserido no cache com maior tamanho que o espaço livre, outros objetos serão removidos de tal forma que o espaço livre seja suficiente para a inserção do objeto n. Os objetos a serem removidos do cache serão escolhidos, um por vez, dentre os que possuem menor relação semântica com objeto de n, até que o espaço, ocupado pelos objetos que forma removidos, seja maior ou igual ao espaço necessário para acomodar o objeto n.

4.15 Dinâmica

Ele organiza os mesmo documentos só que em três filas diferentes: Primeira com LRU, Segunda com LFU e última com SIZE, assim na hora da substituição ou eliminação, podemos escolher o melhor objeto analisando sua freqüência de acesso, tamanho e tempo do último acesso.

4.16 Least Dynamic Frequency Rule (LDFR)

Quando um novo arquivo será inserido no cache, é selecionado um ou mais objetos, de acordo com o tamanho do novo objeto, com menor freqüência de acesso para ser retirado. Uma técnica interessante que pode ser útil, o problema é que quando este objeto novo for grande demais, dependendo do nosso tamanho de cache, ele pode retirar objetos que tem uma freqüência significante de acesso, mas por algum motivo ou determinado tempo, não foi acessado recentemente e como é necessário retirar, ele retira os que realmente tem menos freqüência de acesso e, devido espaço, também estes com acesso significativos.

4.17 Pyramidal Selection Scheme (PSS)

Nesta política, os objetos são classificados e ordenados em forma piramidal utilizando a política LRU e são organizados em grupos. Quando um objeto novo chega, esta política escolhe os objetos menos recentes usados do grupo levando em conta o tamanho e a freqüência utilizada.

4.18 Least Frquently Used With Dynamic Aging (LFUDA)

A política de substituição chamada LFUDA utiliza um contador de referência máxima que possibilita determinar os objetos mais importantes sem levar em conta seus tamanhos, otimizando assim a taxa de BHR e denegrindo o HR, já que um objeto maior e mais popular vai evitar que qualquer outro menor e levemente menos popular seja mantido. (MONTEIRO *et al.*, 2001).

4.19 Considerações Finais

Conforme foi apresentado neste capítulo, existe uma quantidade relativamente grande de políticas de substituíção, as quais, podem ser implementadas e comparadas, objetivando identificar a mais adequada ao perfil do servidor. A complexibilidade para aplicar estes algoritmos é um fator relevante, pois são tratados parâmetros que envolvem informações dinâmicas, porém, essenciais para o gerenciamento das trocas de objetos.

5. PROJETO SIMULADOR WEB CACHE

Neste capítulo será descrito como utilizar o simulador, explicando os passos e procedimentos necessários para realizar a simulação de um determinado log de acesso.

5.1 Motivação

O Simulador, apresentado na Figura 5, simula as requisições de clientes para o *proxy*, determinando as taxas de HR e BHR para as mesmas requisições, considerando três diferentes políticas de substituição de arquivos (LRU, LFU e SIZE), avaliando qual a melhor política que pode ser adotada no servidor em estudo.

FIGURA 5 – Tela inicial do simulador *cache web*.

5.2 Servidor *Proxy* utilizado como Base no Projeto

Neste projeto, foi utilizado os log´s do *proxy* de um servidor *Squid.* O *Squid* foi desenvolvido pela Universidade do Colorado em Boulder, seu código é distribuído livremente, facilitando o trabalho de instrumentalização e análise. É capaz de suportar protocolos HTTP, *FTP*, *Gopher.* Possui regras para controle de acessos, recursos para geração de *logs* dos *requests* atendidos e é compatível com SSL (*Secure Sockets Layer)* (MONTEIRO e*t al.*,2001).

5.3 Características do Servidor Squid

Segundo Fonseca (1998, *apud* MONTEIRO, 2001), "o *Squid* foi concebido para ser um servidor *proxy* eficiente, com o menor tempo de resposta possível para o usuário". Para isso, mantém em memória RAM os objetos em trânsito, os objetos mais recentemente acessados (com uma quantidade de memória pré-alocada para este fim), uma tabela com dados de todos os objetos armazenados no *cache*, as últimas chamadas de resolução de nomes e os últimos objetos acessados com erro.

O *Squid* depende da infra-estrutura de rede. Quando ele entra em operação, as rotas e interfaces de rede da máquina já devem estar ativadas, sejam elas, sobre uma linha discada, uma conexão através de um roteador dedicado, uma conexão sob demanda (*diald*) ou uma rede local. Ele oferece vantagens na administração, controle de acesso, segurança e ocupação otimizada dos meios de transmissão. Sua base instalada é muito grande, e técnicos que implantam redes não podem deixar de conhecê-lo em profundidade. Tem recursos suficientes para definir com precisão quais tipos de serviços podem ser acessados por quais máquinas e em quais horários.

5.4 Arquivos de Log

Os arquivos de *log* (registro) dos servidores *WWW e Proxy* são arquivos no formato texto que contém informações sobre os acessos às páginas do sistema. Eles possuem um registro completo da atividade do servidor, no entanto, extrair tais informações de dados brutos não é uma tarefa simples.

Existem vários programas que podem analisar tais arquivos, mas normalmente são voltados para o controle de acessos dos clientes e geração de relatórios para o provedor, sem ir de encontro às necessidades do presente trabalho. Neste tipo de solução, é necessária a implementação de um analisador de *logs*, que possa realizar uma varredura nos arquivos e obter informações sobre seu comportamento. Nosso objetivo é simular os acessos deste log fazendo o papel do *proxy* (o software) recebendo requisições dos clientes (linhas do arquivo log)

5.5 O Arquivo ACCESS.LOG

Atualmente os programas de análise de arquivos de *log* são baseados nos registros existentes no arquivo *access.log*, cujo formato depende da configuração da opção *emulate_httpd_log*, no arquivo *squid.conf*. Como padrão, o *Squid* irá criar estes registros em seu formato nativo. Se a opção acima citada estiver habilitada, os registros serão criados no formato padrão HTTPD, como definido pelo *World Wide Web Consortium* (W3C), também chamado *common log format*, que pode ser observado na Figura 6, apresentado por Monteiro *et all.* (2001).

FIGURA 6 – Exemplo de uma linha do arquivo Log gerado no Squid.

1 – Endereço IP do *host* requisitante;
2 – Data e hora da requisição;
3 – O método e a URL exata do objeto requisitado;
4 – O código de status HTTP retornado ao cliente;
5 – O tamanho em *bytes* do documento transferido;
6 – Campo somente encontrado no *log Squid*: código de resultado do *Squid*.

5.6 Importando os Dados

5.6.1 Colocando os Arquivos Log no Padrão para o Simulador

Para analisar os dados do log, inicialmente é necessário que o arquivo seja processado por um programa externo, cujo seu nome é UNIX2DOS (Unix para dos) que executa uma atualização. Esta atualização é feita em cada linha, colocando um caracter de fim de linha (enter). Este processo é necessário devido a forma com que o *software* simulador reconhece as linhas com as dados necessários para a análise.

5.6.2 Selecionando e Importando o Arquivo Log

Para selecionar e importar os dados utiliza-se a interface de importação. Para selecionar o arquivo, deve-se pressionar o botão “Selecionar a rquivo”, que por sua vez abrirá uma interface padrão Windows para localizar o arquivo. Após definir o arquivo que será utilizado para análise, já é possível realizar a importação, pressionando o botão “Importar Arquivo”.

Nesta etapa, o simulador pega linha por linha do arquivo, de cada linha lida é extraída as informações necessárias para alimentar a tabela dos dados importados. As informações extraídas são:

1 – Data do acesso.
2 – Hora do acesso.
3 – Nome do objeto, ou seja, o endereço que foi acessado pelo usuário.
4 – Tamanho do objeto.

Nesta interface é possível observar os possíveis problemas que podem ter acontecido durante a importação, através do “log de importação”. Esta parte foi criada para mostrar as linhas que por algum motivo contém erros, como por exemplo, sem a informação de tamanho do objeto etc, como mostra a Figura 7.

FIGURA 7 –Log das importações das linhas do arquivo log.

Ainda pode-se ver a tabela do arquivo importado, pressionando o botão “Tabela Dados Importados”. Nesta interface são mostradas informações como número de linhas importadas, data inicial do arquivo importado, data final do arquivo importado, além dos registros com

informação de data, hora, nome do objeto e tamanho do objeto. Nesta mesma tela, é possível organizar a tabela por ordem crescente de importação, de data ou de tamanho, como pode-se observar na Figura 8. Esta organização é feita através de comandos SQL (*Standart Query Language*).

FIGURA 8 – Dados importados do arquivo log.

5.7 Definindo os Parâmetros para a Simulação

Através da interface de definições dos primeiros parâmetros (figura 9), define-se o período desejado para a análise, os limites de vazão (*threshold)* que são duas variáveis que existem no *Squid.* A primeira, *lower threshold,* é responsável em indicar até que porcentagem é aceitável o armazenamento de arquivos no cache, a segunda, *upper threshold,* é o percentual limite de ocupação do cache, quando esse valor é atingido a política de substituição utilizada entra em operação e descarta arquivos do cache até que a ocupação do cache fique abaixo do valor definido em *lower threshold.*

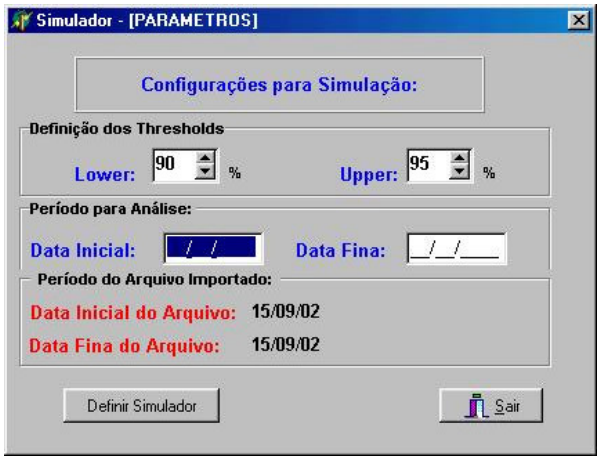


FIGURA 9 – Definição do período a simular e das faixas de thresholds.

No próximo passo, como mostra a Figura 10, será escolhida qual política de substituição que deve ser simulada e a capacidade de armazenamento do cache, cujo tamanho pode ser informado manualmente ou deixar o programa calcular de acordo com a porcentagem da soma do tamanho dos arquivos à simular. Nesta tela ainda, será escolhido o tipo de visualização do simulador, completo (Figura 11) ou compacto (Figura 12). A diferença entre os dois está na maneira com que as informações estão sendo mostradas para o usuário e a velocidade de processamento, como será descrito mais adiante.

FIGURA 10 – Definição da política e tamanho do cache virtual para a simulação.

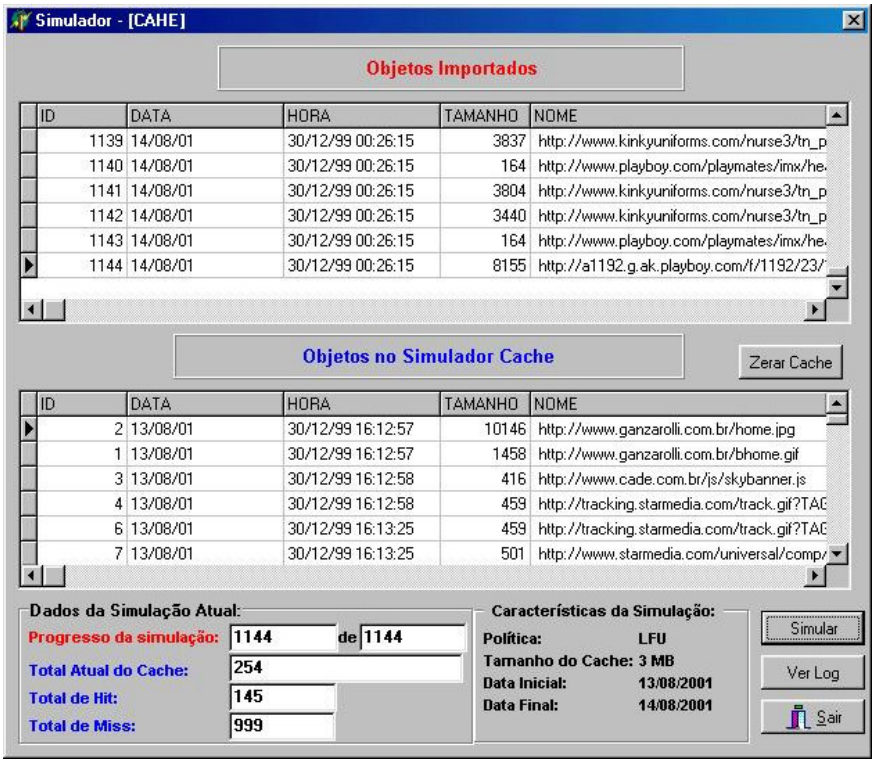


FIGURA 11 – Simulador completo.

FIGURA12 – Simulador compacto.

5.7.1 Simulador Completo

Com o simulador completo, é possível observar todo o funcionamento de armazenamento e retirada dos objetos no cache virtual. Existem duas visualizações nesta interface, a superior mostra os dados importados e a inferior mostra os objetos atuais do cache virtual. Para iniciar a simulação, deve-se pressionar o botão "Simular". Utilizando o simulador completo, pode ser visto todos os estágios da simulação, tais como número de *hits* e *miss* atual, tamanho atual do cache, objeto do log analisado no instante, total de objetos para a

análise, data inicial e final do arquivo log a analisar, qual política foi adotada para a análise atual e até os objetos sendo retirados, criando espaço para o novo objeto a ser inserido no cache.

5.7.2 Simulador Compacto

Com o simulador compacto, não é possível observar todos os detalhes que o simulador completo mostra. Esta parte foi implementada utilizando a mesma parte do algoritmo criado para os dois tipos de simulador, sua diferença é que muitas informações serão vistas somente no log da simulação, que será mostrado mais adiante. Pode-se observar que as quantidades de informações a serem atualizadas na tela são menores e a finalidade desta interface é a obtenção de um melhor desempenho, que foi um dos fatores que dificultou a análise dos log´s a serem simulados. Este problema será abordado mais adiante no capítulo construção do simulador.

5.7.3 Verificando os Resultados

Ao terminar a simulação, é gerado um arquivo log, com o resultado para a análise. Este arquivo está disponível no botão "Ver Log" que irá mostrar a int erface com o resultado (figura 13). Este log traz diversas informações da simulação, como política utilizada, *HR* e *BHR*, tamanho do cache, quantidade de h*it* e m*iss* etc. Cada nova simulação, utilizando outra política de substituição é acrescentada no final do log.

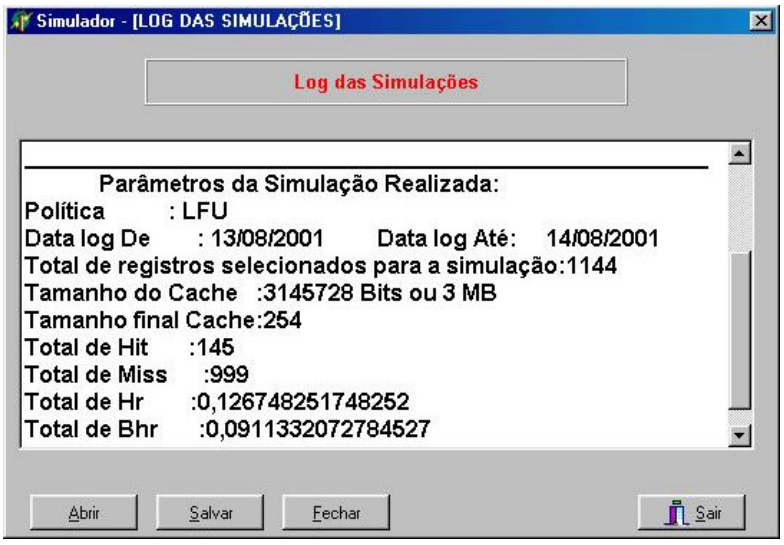


FIGURA 13 – Log das simulações.

6. CONSTRUÇÃO DO SIMULADOR

6.1 Ambiente de Programação

Para elaborar o simulador foi necessário escolher uma linguagem visual com recursos abrangentes a nível de programação e ao mesmo tempo que fosse de meu conhecimento, garantindo agilidade na construção do Simulador e facilitando a evolução do mesmo. Para contemplar estas requisições foi escolhido o Delphi.

6.2 Hardware utilizado nas Simulações

Os resultados obtidos através do simulador *cache web*, foram processado em um Pentium 4 1,6 Ghz com 256 MB de memória, com disco rígido de 10 GB e o sistema operacional Microsoft Windows 98 SE.

6.3 Obtendo Dados para a Simulação

Nos primeiros estudos do simulador, optou-se por somente processar linha por linha dos arquivos de Log, sem se preocupar em armazenar no banco de dados as informações que seriam retiradas do log, ou seja, somente os objetos no cache virtual seriam armazenados. Observou-se nesta etapa dois problemas: dificuldade de selecionar um determinado período e a quantidade de processamento que seria necessário para obter os dados principais dos registros e em seguida simular de acordo com a política no cache virtual. Para uma pequena quantidade de registros, isso não seria problema, mas os log´s gerados no *Squid*, tem um número de linhas (registros) muito grande, nas amostras obtidas, em média tem-se 500.000 registros, o que tornaria muito lento as comparações no cache virtual.

6.4 Banco de Dados

Como a necessidade era importar os registros com os dados mais relevantes para a simulação e o armazenamento dos objetos no cache virtual, foi criada uma base de dados utilizando BDE (*Borland Database Engine*) como forma de acesso e o banco de dados escolhido inicialmente foi o Inter Base 6. Para todos os testes realizados com arquivos de log

pequenos, em torno de 5.000 registros, todo o processo foi feito sem problemas, mas na medida em que os log´s maiores foram testados, o banco de dados começou a apresentar retardo de processamento já na etapa de importação dos registros. Para quantidades superiores a 100.000, seu desempenho degradava, na medida em que o cache virtual aumentava. Foi nesta etapa que foi criada a interface do simulador compacto, que diminuiu um pouco o tempo de processamento, porém ainda o tempo de simulação alcançado não foi aceitável.

Pensou-se então, em construir o Simulador em outra linguagem de programação, porém, optou-se em continuar utilizando *Delphi*, buscando outras soluções de acesso ao banco de dados. Utilizou-se então a biblioteca ADO (*Active-X Data Objects*), que são componentes usados na programação que permitem o acesso a dados sem utilização do BDE. Geralmente o ADO é usado com provedores OLE DB, pois o ADO encapsula a complexidade das chamadas de baixo nível aos provedores OLE DB, fornecendo uma interface simples e clara aos programadores.

Toda a estrutura foi alterada e o ganho de performance do Inter Base aumentou um pouco mais, mas mesmo assim, não era suficiente. Foi optado em experimentar outro banco de dados, escolheu então, o Microsoft Access 97, que apresentou um bom desempenho já na importação, não perdendo processamento em momento algum para importar os dados, independente de ser 500.000 ou 700.000 registros. Na simulação ele também superou a velocidade de processamento em relação aos outros testes. Optou-se então pelo uso do Microsoft Access.

6.5 Implementação das Políticas

Para a construção do simulador *cache web*, foi definido a implementação de três políticas de substituição: LFU, LRU e SIZE. Nas três políticas, os métodos de inclusão, alteração e exclusão no cache virtual são implementadas da mesma maneira. Para otimizar o processo, foi criada *querys* que são responsáveis em realizar estes processos. As *querys* são objetos que o *Delphi* disponibiliza para realizar as consultas e comandos via SQL.

A estrutura de comparação se o objeto já se encontra armazenado, também é feito com *query*. Ela recebe o objeto atual da tabela de importados como parâmetro e os compara na

tabela do cache virtual. Se ele existir, será acrescentado o número de *hits* (sucesso, ou seja, o objeto foi encontrado), se ele não existir, será acrescentado *miss* (falha, ou seja, o objeto não foi encontrado). Para os objetos não encontrados, será necessário acrescentá-los no cache virtual. Caso exista espaço (o percentual do cache estiver abaixo no valor definido em *upper threshold*), é executada uma *query* que se encarrega de incluí-lo. Se não existir espaço, será necessário tirar objetos até que haja espaço necessário para a inclusão do objeto, acionando-se então, as políticas de substituição. Para cada política escolhida, a tabela do cache virtual, fica ordenada de acordo com as suas características, por exemplo, a política SIZE tem o cache virtual organizado por ordem de tamanho. Como a tabela em que será criado o espaço necessário já está ordenado, o que resta fazer é ir excluindo os últimos registros até que o espaço seja criado para a inclusão do novo objeto.

6.5.1 Algoritmo Genérico para as Políticas Implementadas

A tabela de cache está ordenada por tamanho, do menor para o maior.
Vai para primeiro registro tabela importado
Repita
Verifica se objeto tabela importado está no cache
Se falso (não está no cache)
Início
Se tamanho objeto tabela importado > tamanho atual do cache então
Início
Repita
Vai para último registro do cache
Tamanho atual do cache := Tamanho atual do cache + tamanho último registro cache
Deletar último registro do cache
Até Tamanho atual do cache >= tamanho objeto tabela importado
Fim
Grava dados da tabela importado na tabela do cache
Total Miss := Total miss + 1
Fim
Senão (está no cache)
Total Hit := Total Hit + 1;

Leia próximo registro tabela importado

Até final (arquivo de log)

7. ANÁLISES E RESULTADOS

Neste capítulo é apresentado os resultados das simulações realizadas a partir de um arquivo de log de um provedor de acesso a Internet.

O arquivo analisado no projeto teve sua origem do *Squid*, instalado no Servidor do Centro de Ciências Agroveterinária - CAV de Lages, referente aos acessos realizados no dia 15 de setembro de 2002.

7.1 Quantidade de Registros Processados

Cada arquivo de log obtido, referente a aproximadamente 4 dias, possui em torno de 600.000 registros, sendo que para simular com um cache virtual de 2 Gb, o tempo de simulação obtido foi superior a 36 horas, o que torna esta simulação inviável. A idéia inicial do trabalho era conseguir simular os acessos de uma semana e analisar para um cache virtual de 2Gb, que é o tamanho real do cache no *Squid* do CAV, para as três políticas. A segunda proposta estudada foi a tentativa de pegar o log de um ou dois dias para análise, mas os resultados não ficariam corretos para o tamanho do cache de 2Gb, continuando ainda com um tempo de processamento muito alto.

Outro fator que foi analisado foi o tamanho do cache para esta simulação de um dia, deveria ter uma proporção ao período em análise, uma vez que 2Gb de cache virtual para simular somente um dia, acarretou em um grande número de *hit*. Com o objetivo de minimizar o tempo de processamento e aproximar o tamanho do cache virtual proporcionalmente ao período da análise de um dia, adotou-se então, a proposta de NICLAUSSE, LIU e NAIN (1998), que é simular os arquivos log, com tamanho de cache variando entre 5% a 35% do tamanho total da soma dos tamanhos dos arquivos do log analisado, independente do período de análise. Selecionou-se 150.000 registros do dia 15 de agosto de 2002 para efetuar as simulações para 5%, 10%, 15%, 20% e 25% do tamanho proporcional do arquivo importado, para cada política implementada (LRU, LFU e SIZE), como pode-se observar nos resultados dos log´s apresentados no Anexo 2.

7.2 Resultados Obtidos nas Simulações

Os resultados das simulações, de acordo com a proposta descrita anteriormente, são apresentados na Tabela 1.

TABELA 1- Resultados das simulações realizadas para as políticas LFU, LRU e SIZE. Unidade de medida HR: fração de requisições atendidas pelo cache. Unidade de medida BHR: fração de bytes atendidos pelo cache.

	Cache para 5%		**Cache para 10%**		**Cache para 15%**		**Cache para 20%**		**Cache para 25%**	
Políticas	**HR**	**BHR**	**HR**	**BHR**	**HR**	**BHR**	**HR**	**BHR**	**HR**	**BHR**
LFU	0,57208	0,35992	0,61450	0,42071	0,62572	0,42728	0,64864	0,45422	0,66308	0,47707
LRU	0,45795	0,27928	0,56003	0,36404	0,59112	0,40783	0,60903	0,41090	0,62789	0,44045
SIZE	0,64529	0,31566	0,66191	0,36776	0,67033	0,40897	0,67366	0,45040	0,67656	0,48006

Com o aumento do tamanho do cache virtual, observou-se nas simulações, que o tempo de processamento também aumentou, como mostra a Tabela 2, que apresenta o tempo das simulações em relação as porcentagens do tamanho do cache.

TABELA 2- Tempo final do processamento das simulações.

	Cache para 5%	**Cache para 10%**	**Cache para 15%**	**Cache para 20%**	**Cache para 25%**
LFU	03:59:36	06:32:15	09:32:06	09:09:21	12:04:53
LRU	06:25:10	10:50:32	13:39:43	15:07:29	17:32:36
SIZE	09:23:38	11:44:09	12:10:42	14:04:32	14:18:13

Com os dados obtidos, pode-se fazer um comparativo entre as taxas para cada política implementada, onde pode-se observar que aumentando o tamanho do cache, não há um aumento significativo as taxas de HR e BHR , e desta forma aumentando o *overhead* devido a um maior tempo de consulta no cache. Pode-se comprovar o descrito nas Figuras 14, 15 e 16.

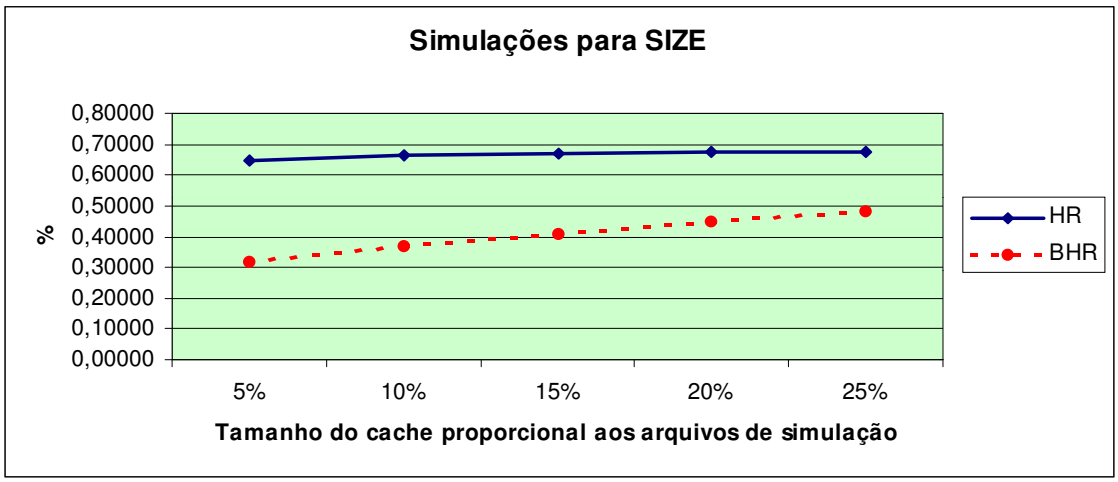


FIGURA 14 – Taxas na simulação da política SIZE.

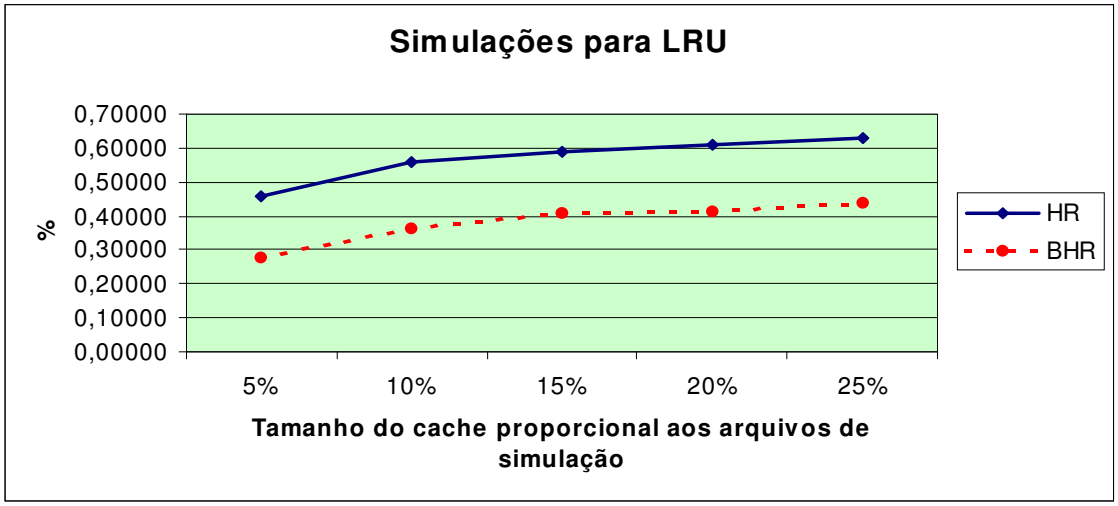


FIGURA 15 – Taxas na simulação da política LRU.

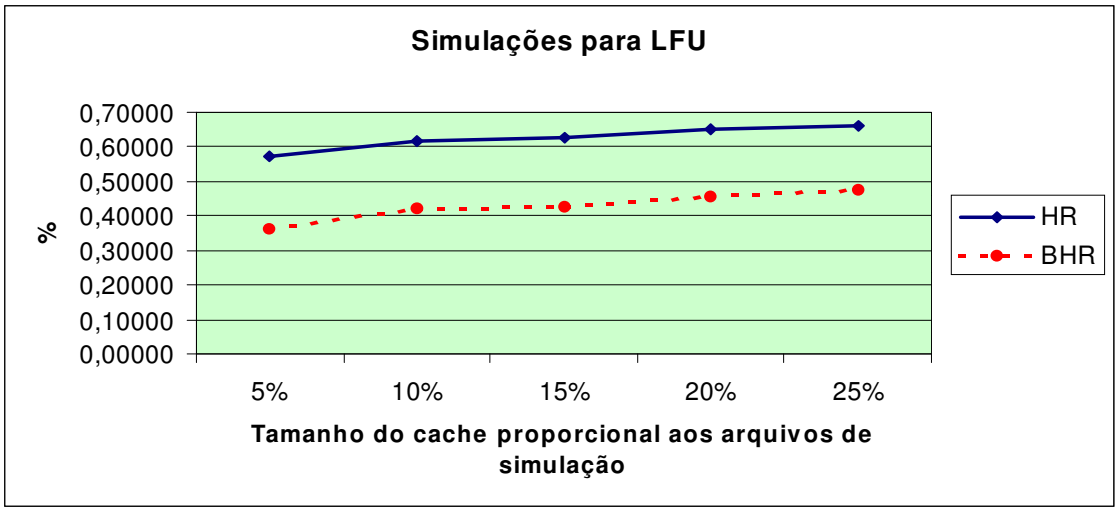


FIGURA 16 – Taxas na simulação da política LFU.

7.3 Análise dos Dados

Para tirar uma conclusão para o log proposto, deve-se fazer uma comparação com as taxas obtidas de *HR* e *BHR* entre as três políticas propostas como é mostrado nas Figuras 17 e 18.

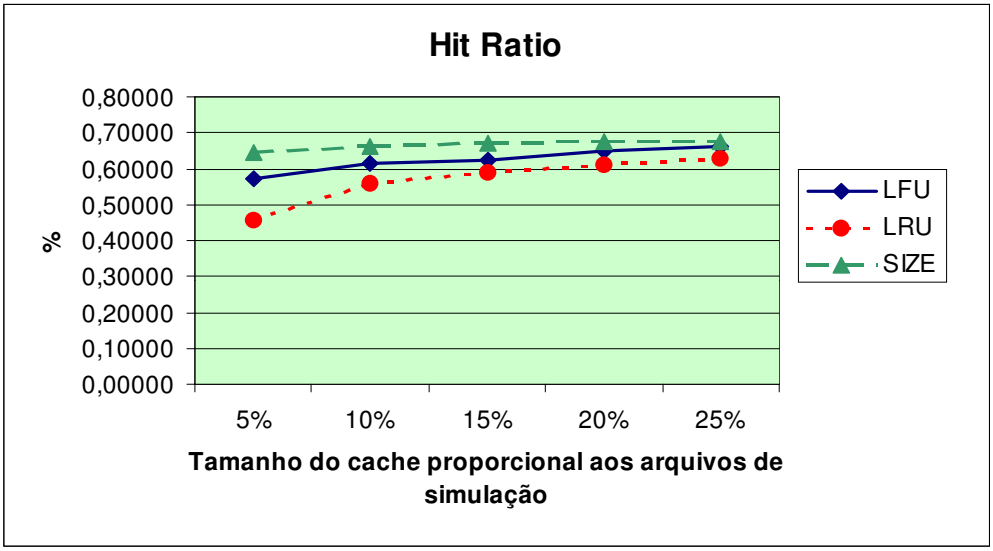


FIGURA 17 – Taxas de *Hit Ratio* para as três políticas.

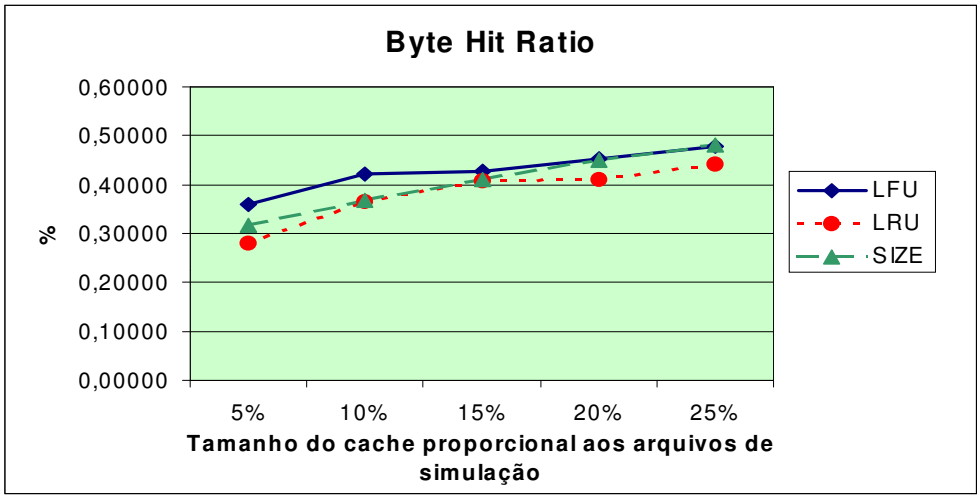


FIGURA 18 – Taxas de *Byte Hit Ratio* para as três políticas.

Para todos os tamanhos de cache, a política que apresentou a melhor taxa de HR foi a SIZE, como era esperado, pois ela retira os maiores arquivos do cache, desta forma armazenando uma quantidade maior de arquivos, consequentemente há

uma probabilidade maior de encontrá-los em cache, porém, pode-se observar que esta política não possui os melhores valores para BHR.

A política LFU apresentou os melhores resultados na análise de BHR.

Portanto, as simulações permitem a escolha de uma política adequada para a empresa de onde foram obtidos os dados. Para a escolha correta, deve-se levar em consideração as particularidades da empresa, como por exemplo, a taxa de utilização do enlace com a Internet, pois, se o enlace estiver sobrecarregado, o melhor é otimizar BHR, diminuindo esta taxa. Porém, caso isto não esteja ocorrendo, deve-se optar por otimizar HR, pois tendo uma maior taxa de acerto, ter-se-á uma maior quantidade de clientes satisfeitos.

Para obter a taxa de utilização do enlace Internet, sugere-se a utilização do MRTG[1] (*Multi Router Traffic Ghapher*) que monitora o enlace constantemente.

Na Figura 19, pode-se observar o gráfico gerado pelo MRTG no CAV, com as estatísticas diárias de uma semana, onde se observa que em picos, estão sendo utilizados aproximadamente 68 % (1062 kbps) da largura de banda total.

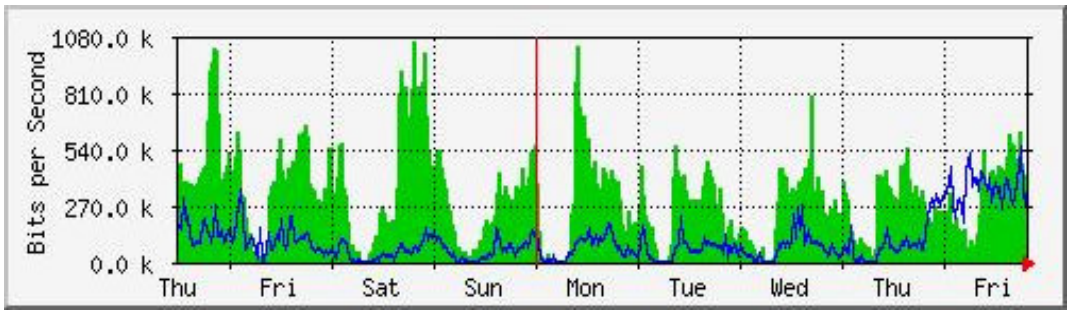


FIGURA 19 – Estatística gerada pelo MRTG do enlace de uma semana do CAV.

Portanto, para os dados que trafegam nesta rede é importante otimizar a taxa de HR, desta forma, a política mais adequada entre as que foram testadas é a política LFU. Porém, a política configurada atualmente no Servidor é LRU (a política padrão do *Squid*).

É importante salientar que o perfil dos usuários e tráfego de uma empresa ou instituição em relação a outras tem grande variabilidade, portanto, é necessário analisar cada caso em particular.

---

http://people.ee.ethz.ch/~oetiker/

8. CONCLUSÃO

Uma das alternativas para analisar as taxas de HR e BHR em um servidor Web Cache para diferentes políticas de substituição de arquivos, é apagar todos os arquivos armazenados em cache, implantar a nova política e depois de um longo período analisar os arquivos de log (MONTEIRO, 2001). Porém, este método de monitoração pode denegrir a qualidade de serviço prestado pela empresa. Uma outra alternativa para resolver este problema é o uso de técnicas de simulação.

Nas análises realizadas obteve-se um elevado tempo de processamento, inviabilizando a sua utilização com grandes arquivos de *log* e tamanho de *cache* real. Para diminuir o tempo de processamento, implementou-se a proposta de Niclausse, Liu e Nain (1998), que é realizar a simulação com o tamanho de *cache* variando entre 5% a 35% do total da soma dos tamanhos dos arquivos do *log* analisado, independente do período de análise, o que diminuiu significativamente o tempo de processamento. De qualquer forma, o tempo observado nos dois casos, ainda é muito inferior ao tempo necessário para a realização das comparações em um sistema real, o que causaria também uma queda de desempenho. Este trabalho também se justificou, devido a necessidade de pesquisadores desta área, possuírem um simulador para o estudo de novas políticas de substituição de arquivos, possibilitando analisar as taxas de HR, BHR etc.

Como proposta para trabalhos futuros, destacam-se duas sugestões. A primeira seria implementar um simulador utilizando a linguagem de programação C, lendo os arquivos de *log* no modo texto e a implementação da política de substituição proposta por Murta (1999) e Oliveira (1997) em um *proxy* real, nestes trabalhos foi constatado que a utilização do cache particionado é uma maneira de tentar otimizar as taxas de HR e BHR, onde os objetos são separados por classes conforme seus tamanhos. Podendo então, para cada partição adotar uma política de substituição diferente, aumentando assim o desempenho das taxas de *HR* e *BHR* para cada parte do cache. A segunda sugestão é elaborar uma análise detalhada, comparando simulações elaboradas a partir das políticas mostradas neste trabalho com resultados obtidos a partir de servidores ativos em picos de utilização.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, V., M.E. Crovella, A. Bestavros & A. de Oliveira, **Proceedings of PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems**, Miami Beach 1996, Florida. Disponível em: <http://www.cs.bu.edu/> Acesso em 25 de outubro de 2002.

BRANDÃO, Roberto Ferreira; ANIDO, Ricardo de Oliveira. **Um simulador paralelo para sistemas de caches para WEB.** Campinas, São Paulo. Instituto de Computação, UNICAMP.

CALSAVARA, Alcides; SANTOS, Rogério Guaraci dos. **Um algoritmo de substituição em cache na Internet baseado em semântica.** Artigo Pontifícia Universidade Católica do Paraná – PUCPR, 16 pág.

MONTEIRO, Valdirene de Souza. **Descrição e análise das cargas de trabalho de um sistema distribuído.** (Monografia, Bacharel em Informática). UNIPLAC, Lages–SC, 2001.

MEIRA, W. J.; MURTA C. D.; CAMPOS S. V.; GUEDES D. O. **Sistemas de comércio eletrônico.** Projeto e desenvolvimento. São Paulo:Campus, 2002.

MURTA, Cristina Duarte. **Desempenho de sistemas de cache da WW: limitações e potencial.** Minas Gerais, 1999. Disponível em: <http://www.dcc.ufmg.br/pos/html/pg99/anais/cristina/cristina.html>. Acesso em: 04 junho 2002.

_____. **Gerência de espaço em caches na World Wide Web** Minas Gerais, 1999.

NIELSEN, H. Frystyk, J. Gettys, A. Baird-Smith, E. Prud' hommeaux, H. Lie & C. Lilley , **Network performance effects of http/1.1 , CSS1, and PNG** in ` Proceedings of the ACM SIGCOMM ' 97' , Cannes 1996, France. Disponível em : <http://www.w3.org/pub/WWW/Protocols/HTTP/Performance/Pipel ine.html> Acesso em 20 de outubro de 2002.

NICLAUSSE, Nicolas; LIU, Zhen;NAIN, Philippe**. A New Efficient Caching Policy for the World Wide Web.** Disponível em: <http://www.cs.wisc.edu/~cao/WISP98/html-versions/Nicolas/final.html> Aceso em 10 de outubro de 2002.

OLIVEIRA, Jacqueline Augusto de. **Uso de caches na web - influência das políticas de substituição de objetos.** São Carlos, São Paulo 2002. Instituto de Ciências Matemáticas e de Computação - ICMC-USP.

OLIVEIRA, Rodrigo Machado. **O uso de caches para o armazenamento de objetos da WEB.** Campinas, São Paulo. 1997. Disponível em: <http://www.dcc.unicamp.br/html>. Acesso em: 20 maio 2002.

SÁ, Paulo Roberto Oliveira de. **Monitoração e otimização de rede mediante análise de desempenho e cargas do sistema.** (Monografia, Bacharel em Informática). UNIPLAC, Lages – SC, 2001.

SMITH**,** A.J., **Operating Systems Reviews** Bibliography on paging and related topics, Cap12, 39-56. 1978.

_____. **Computer Architecture News** Second bibliography for cache memories 19(4). 1991.

SOARES, Luiz Fernando Gomes; GUIDO, Lemos de Souza Filho; COLCHER, Sérgio. **Redes de computadores: das LANs, MANs e WANs às redes ATM**. 2.ed.. Campus, 1997.

ZOTTO, Ozir Francisco Andrade, **Protocolo HTTP (Hypertext Transfer Protocol)** Disponível em: <http://www.pr.gov.br/celepar/celepar/batebyte/edicoes/1996/bb57/> Acesso em 15 de outubro de 2002.

ANEXOS

ANEXO 1

RELAÇÃO DE CÓDIGOS DEFINIDOS PELA RFC

Código	Significado
100	Continuando
101	Trocando protocolos
102	Processando
200	OK
201	Criado
202	Aceito
203	Informação não-autoritativa
204	Sem conteúdo
205	Reiniciando conteúdo
206	Conteúdo Parcial
207	Múltiplos Status
300	Múltiplas escolhas
301	Movido Permanentemente
302	Movido Temporariamente
303	Veja Outro
304	Não Modificado
305	Usando Proxy
307	Temporariamente Redirecionado
400	Requisição Inválida
401	Não Autorizado
402	Pagamento Requerido
403	Proibido
404	Não Encontrado
405	Método Não Permitido
406	Não Aceitável
407	Autenticação no Proxy Requerida
408	Requisição Expirada
409	Conflito
410	Não Existe Mais
411	Tamanho Requisitado
412	Falha em Condição Prévia

413..... Requisição muito grande
414..... URL muito grande
415..... Tipo de mídia não suportado
416..... Requisição Não Satisfatória
417..... Falha na Resposta
424..... Travado
424..... Falha em Dependência
433..... Entidade não Processável
500..... Erro Interno do Servidor
501..... Não Implementado
502..... Erro de Gateway
503..... Serviço não disponível
504..... Gateway Expirado
505..... Versão HTTP Não Suportado
7..... Espaço de armazenamento insuficiente
600..... Erro de *log Squid*

ANEXO 2
FIGURAS COM OS RESULTADOS DAS SIMULAÇÕES

Simulação LFU para 5% de cache proporcional

Simulação LFU para 10% de cache proporcional

Simulação LFU para 15% de cache proporcional

Simulação LFU para 20% de cache proporcional

Simulação LFU para 25% de cache proporcional

Simulação LRU para 5% de cache proporcional

Simulação LRU para 10% de cache proporcional

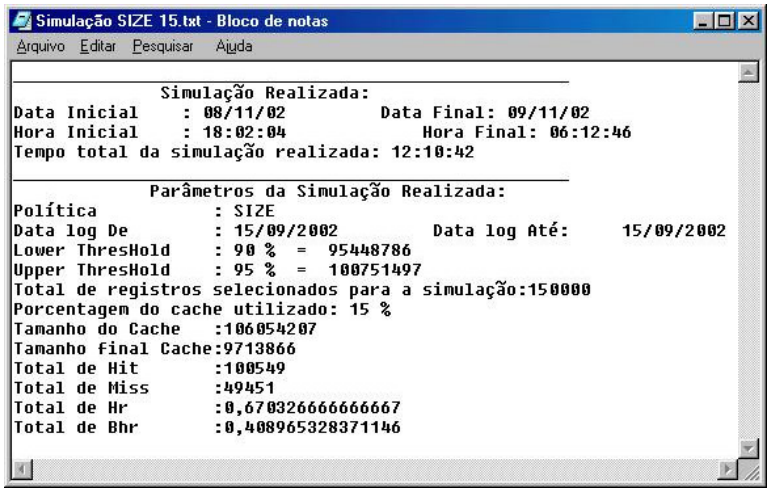
Simulação LRU para 15% de cache proporcional

Simulação LRU para 20% de cache proporcional

Simulação LRU para 25% de cache proporcional

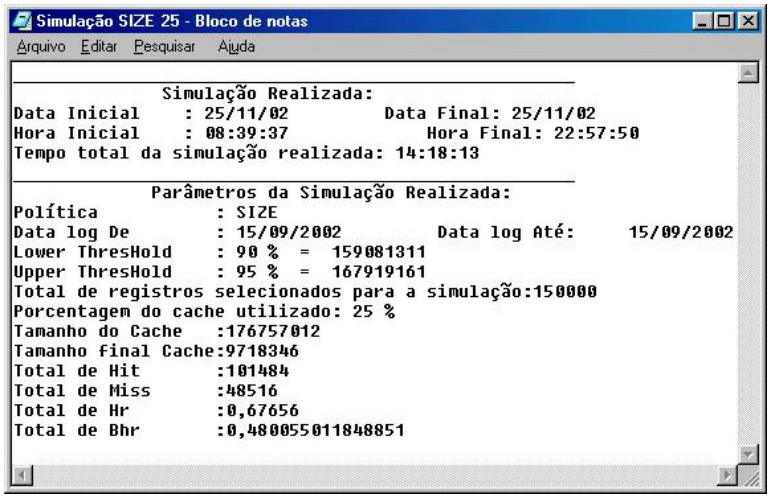
Simulação SIZE para 5% de cache proporcional

Simulação SIZE para 10% de cache proporcional



Simulação SIZE para 15% de cache proporcional

Simulação SIZE para 10% de cache proporcional



Simulação SIZE para 10% de cache proporcional