Pontifícia Universidade Católica do Paraná – PUCPR
Centro de Ciências Exatas e de Tecnologia – CCET

Substituição de Objetos em Cache da Internet Baseada em Semântica e Histórico de Acessos

Marcelo Roberto Schuck

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná, como parte dos requisitos para obtenção do título de Mestre em Informática Aplicada

Julho de 2005

Resumo

O crescimento exponencial da Web e a conseqüente necessidade de diminuir a sobrecarga dos servidores tornam altamente crítica a tarefa de substituição de objetos em *cache*. Neste trabalho, é apresentada uma nova estratégia de substituição de objetos em *cache* chamada *Least Semantically Related/History - LSR/H*. A estratégia tem como objetivo manter em *cache* os objetos pertencentes aos grupos semânticos de maior interesse em um passado recente. Para tanto, supõe-se que os objetos estejam classificados segundo uma hierarquia de grupos semânticos, isto é, cada objeto acessado deve pertencer a um certo grupo que define a sua semântica. É mantido um histórico de acessos, registrando os últimos n grupos acessados, onde n é o comprimento do histórico. Tal histórico permite identificar e dar maior peso aos grupos semânticos de maior interesse num certo período e, conseqüentemente, escolher para remoção do *cache* um objeto que seja de menos interesse com relação a uma seqüência de objetos recentemente acessados. Dessa forma, a estratégia contempla o caso em que existam diversos interesses simultâneos entre os usuários de um certo gerenciador de *cache*. Foi projetada e implementada uma estrutura para a coleta e preparação dos dados a fim de poder validar e comparar com estratégias de substituição de objetos tradicionalmente utilizadas na Internet, tais como LRU, LFU, SIZE, e também com o LSR em sua versão original, isto é, sem histórico. A análise de desempenho da estratégia proposta constata a importância do histórico na remoção dos objetos, pois obtém uma probabilidade de acerto superior às outras estratégias.

Palavras-chaves: Algoritmos de substituição, *Web caching*, *Semantic Web* , Histórico de acessos.

The exponential growth of the Internet and its consequent need for diminishing server load make object cache replacement a highly critical task. In this research work, it is introduced a novel object replacement strategy named *Least Semantically Related/History - LSR/H*. Such strategy aims at keeping in cache objects that belong to semantic groups of greater interest in a recent past. For that reason, the strategy assumes that all objects are classified according to a hierarchy of semantic groups, that is, each requested object must belong to a certain group which defines its semantics. The strategy keeps an access history that records the last n accessed groups, where n is the history length. Such history permits to identify semantic groups of greater interest in a recent past and, consequently, choose an object to evict from cache which should be of less interest with respect to a sequence of recently requested objects. Thus, the strategy provides for the case where there can be several distinct threads of interest simultaneously in a given cache manager. A framework has been designed and implemented in order to collect and prepare data to validade the proposed strategy and compare it to traditional strategies, such as LRU, LFU, SIZE and also to the original version of LSR, that is, the version that does not take history into account. The performance analysis shows the importance of access history for object replacement because the hit rate and the byte hit rate obtained by employing the proposed strategy are higher than rates obtained by employing other strategies.

Aos meus pais Guilherme e Vera

Agradecimentos

À Pontifícia Universidade Católica do Paraná.

Ao Programa de Pós-graduação em Informática Aplicada.

Ao meu orientador, Professor Doutor Alcides Calsavara, pela orientação e, principalmente, pelos valiosos conhecimentos transmitidos durante o mestrado.

Aos Professores e Funcionários da PUC-PR, em especial aos Professores Doutores: Robert Carlise Burnett, Roberto Cesar Betini, Carlos Alberto Maziero e Júlio César Nievola

Aos amigos do Programa de Pós-graduação em Informática Aplicada.

À minha mãe Vera Besten Schuck, pelo apoio, carinho e dedicação.

Às minhas irmãs Margaret Schuck e Márcia Schuck, pelo apoio e entusiasmo.

Aos meus sobrinhos, Paula Andressa Schuck Borcoski, Guilherme Henrique Schuck Borcoski e Eduardo Augusto Schuck Lira, pelos momentos de descontração.

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

Em especial, à Cátia Silveira dos Santos, pela compreensão do esforço do trabalho realizado.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Introdução

A utilização de *cache* na Intenet é fundamental para garantir a sua escalabilidade. Basicamente, *cache* permitem que objetos sejam acessados mais rapidamente pelos clientes, causam redução do tráfego na rede e fazem com que seja reduzida a carga dos servidores. Assim, o uso do *cache* permite uma melhor utilização da Internet como um todo e contribui para a melhoria da qualidade de serviço.

Devido à limitação natural do tamanho do *cache*, as políticas de substituição de objetos decidem quais irão permanecer e quais serão retirados para ceder espaço aos novos objetos. A escolha de uma política tem influência no consumo de banda devido à diminuição de tráfego na rede e a quantidade de vezes que o documento é encontrado no *cache*.

Estas políticas estão sendo amplamente estudadas pela comunidade científica, como apresenta o Capítulo 2. De forma geral, as políticas documentadas na literatura e também utilizadas na Internet têm como heurística alguma propriedade externa dos objetos, tais como o seu tamanho (política SIZE), a taxa de freqüência de acesso (LFU) e o instante do último acesso (LRU).

Este trabalho propõe uma nova política que explora propriedades internas do objeto, mais especificamente, explora a semântica da informação representada pelo objeto, combinado com a observação de que os objetos acessados em um certo período de tempo, numa certo gerenciador de *cache*, tendem a se concentrar em algumas semânticas, de acordo com os interesses dos usuários. Assim, este trabalho supõe que os objetos sejam classificados de acordo com uma taxonomia que reflita a sua semântica. Existem diversos esforços para

o emprego de semântica, tais como Semantic Web [W3C-Consortium, 2001] e o DMOZ [DMOZ, 2005], além do próprio Yahoo! [YAHOO!, 2005]. Além disso, em certas situações, como por exemplo, numa biblioteca digital, os objetos são necessariamente classificados. De fato, o trabalho desenvolvido em [Shmidt, 2002] [Calsavara and Schmidt, 2004] mostra a viabilidade da utilização de *servidores de semântica* em ambientes nos quais existe uma taxonomia bem definida para os objetos. Logo, o emprego de uma política de substituição de objetos baseada em semântica deverá ter o seu nicho de aplicação.

A política proposta neste trabalho denomina-se *Least Semantically Related History - LSR/H* e foi inicialmente apresentada em [Calsavara and Schuck, 2005]. Trata-se de uma extensão da política anteriormente proposta por [dos Santos, 2001] [Calsavara et al., 2003] [Calsavara and dos Santos, 2002] denominada LSR, a qual considera, para fins de eleiçção de um objeto para remoção, a sua proximidade semântica somente do último objeto acessado, atendendo apenas a um único interesse por vez. Com a extensão, que atende a múltiplos interesses simultâneos, constatou-se um desempenho melhor em comparação com sua versão original e também com as políticas tradicionais utilizadas hoje na Internet.

1.2 Motivação

Com o rápido crescimento da Web, aliado à necessidade de diminuir a sobrecarga dos servidores, faz com que a tarefa de avaliação de desempenho de uma nova política de substituição de objetos seja um verdadeiro desafio.

Estas políticas estão sendo amplamente estudadas pela comunidade científica para diminuir o tempo de acesso e eliminar o tráfego redundante, ocorrido pela passagem de diversas cópias dos mesmos documentos na rede. A finalidade é diminuir tal problema fazendo uso de dados previamente consultados, ao invés de buscá-los na origem sempre que forem requisitados.

As empresas de hospedagem da Web pagam pela largura da banda, sendo assim, não só os usuários beneficiam-se com a redução do tempo, mas também as empresas na redução dos custos.

Em resumo, o uso de *cache* permite que os seguintes benefícios fiquem evidentes:

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

1.3 Objetivo

O objetivo principal desta dissertação foi dar continuidade e ampliar o estudo iniciado em [dos Santos, 2001] [Calsavara and dos Santos, 2002] [Calsavara et al., 2003] que apresentou a dissertação: Substituição de Objetos em cache na WWW Baseado na Semântica da Informação.

Para tanto, é proposta uma nova estratégia que atenda a diversos interesses simultâneos denominada *Least Semantically Related/History - LSR/H*. Com esse objetivo é projetada e implementada uma estrutura para a devida coleta e preparação dos dados para a simulação. Esta estrutura permite a análise de desempenho de diversas políticas de substituição de objetos de *cache* existentes na Internet, tais como LRU, LFU, SIZE e também com o LSR em sua versão original.

Para a avaliação do desempenho do *cache* é utilizada as métricas *Hit Rate - HR* e o *Byte Hit Rate - BHR*. *Hit Rate* é o percentual de requisições que foram atendidas pelo *cache* enquanto o *Byte Hit Rate* é o percentual de bytes requisitados que foram enviados pelo *cache*, a fim de poder comparar não só o objeto presente no *cache*, mas também, o custo de latência.

1.4 Organização do Texto

Este Capítulo apresentou a introdução da dissertação, a motivação para o seu desenvolvimento e os objetivos a serem alcançados. No Capítulo 2 serão apresentadas diversas políticas

de substituição de *cache* existentes na Internet. O Capítulo 3 apresenta a estratégia original, chamada: *Least Semantically Related - LSR* e como foi elaborado seus experimentos. O Capítulo 4 apresenta a nova estratégia, denominada *Least Semantically Related/History - LSR/H* e também suas definições, histórico e a implementação do algoritmo. No Capítulo 5 será apresentado o processo de coleta e preparação dos dados para o experimento e validação. Também neste capítulo a análise do desempenho será comentada. Por fim, o Capítulo 6 apresenta as conclusões da estratégia proposta e seus novos caminhos de pesquisa.

Capítulo 2

Trabalhos Relacionados

Os pesquisadores têm estudado vários parâmetros para as políticas de substituições de objetos em diferentes aspectos, sempre visando um melhor desempenho do *cache*. Alguns destes trabalhos serão mostrados neste levantamento. Há estudos bem mais detalhados sobre tais políticas, como por exemplo: *A Survey of Web Replacement Strategies* proposta por [Podlipnig and Böszörményi, 2003], que apresenta as vantagens e desvantagens de inúmeras estratégias de *cache*. Nosso objetivo aqui é somente apresentar resumidamente as políticas mais usadas na Internet para dar suporte à discussão de comparação de desempenho com a política proposta neste trabalho.

2.1 Least Recently Used - LRU

A política de substituição de objetos em *cache* proposta por [Dilley and Arlitt, 1999]. remove os objetos mais velhos, isto é, os menos recentemente requisitados. Esta política trabalha bem quando os objetos acessados recentemente são mais prováveis de serem referenciados no futuro. Sua implementação usa uma lista ordenada pelo último tempo de acesso. Para remover elementos basta acessar a cauda da lista. De uma maneira simples faz com que os objetos mais recentes estejam sempre em memória.

2.2 Least Frequently Used - LFU

[Dilley and Arlitt, 1999] propôs outra política, a que mantêm um contador de referências para cada objeto no *cache*. Sendo assim, o objeto que possuir o menor contador é eleito para

ser removido, isto é, o objeto que for menos vezes acessado. Se mais de um objeto tem o mesmo valor do contador, uma segunda política pode ser usada para resolver a critério de desempate. O problema é que alguns objetos podem acumular contadores com valores altos e nunca serem candidatos a substituição. 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.

2.3 Size

Essa política proposta por [Aggarwal et al., 1999], remove os maiores objetos do *cache* quando for necessário espaço para armazenar novos objetos. Esta política tende a reter os objetos menores responsáveis pelo maior número de acerto no *cache*, mas em contra-partida possibilita reter objetos pequenos indefinidamente que podem não ser mais referenciados, se nenhum mecanismo de expiração for utilizado.

2.4 LRUMIN

Este algoritmo é uma variação do LRU, o qual verifica o tamanho do objeto a ser armazenado e procura no *cache* objetos de tamanho maior ou igual. Caso existam documentos que atendem a este requisito, o algoritmo LRU é aplicado a este subconjunto. Caso contrário, todos os objetos cujos tamanhos sejam 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 [Abrams et al., 1996].

2.5 LRU-threshold e LRU-adaptive

São variações de LRU nas quais há um limite para o tamanho de objetos que podem ser armazenado. Na primeira versão este limite é fixo e na segunda é alterado dinamicamente. Na versão dinâmica, a variação correspondente de desempenho indica alterações posteriores.

A definição de um limite para o tamanho do objeto que pode ser armazenado é uma forma de controle de admissão e pode ser utilizado em qualquer política. O estabelecimento de um limite de tamanho para armazenamento inferior ao tamanho máximo dos objetos requisitados equivale a um corte na cauda da distribuição dos tamanhos cujo efeito é um possível aumento em HR (hit rate) acompanhado de uma diminuição do BHR (byte hit rate) [Markatos, 1996].

2.6 Least Normalized Cost Replacement - LNC-R

Este algoritmo de substituição de objetos em *cache* maximiza o tempo de resposta. O algoritmo de substituição de *cache* LNC-R estima a probabilidade de uma futura consulta a um determinado objeto, usando uma movimentação média dos últimos tempos de chegada de K requisições do objeto. Contudo, foi observado que os clientes da Web têm preferência por acessar objetos pequenos. E por isso, a informação de domínio específico é baseada em estimativas da probabilidade de uma futura consulta no padrão de referência do objeto como no tamanho do objeto.

$$DSR = SUM(d_i * h_i)/SUM_i(di * r_i)$$

Figura 2.1 Fórmula para cálculo do custo de substituição de um objeto na estratégia LNC-R

O custo de substituição de um objeto *i é dado pela fórmula apresentada na Figura 2.1, onde* d_i *é a média da espera para buscar o objeto i para o* cache, r_i *é o número total de referências ao objeto i e* h_i *é o número de referências ao objeto i que foram satisfeitas a partir do* cache. *A economia de atraso determina a fração de comunicação e os atrasos do servidor, os quais serão salvos, satisfazendo as referências do* cache *no lugar das de um servidor. Se o tempo para satisfazer uma consulta ao* cache *for menor que o tempo para consultar o servidor (o que é um caso típico), a razão máxima de economia de atraso garante também o mínimo tempo de resposta à consulta do cliente [Scheuermann et al., 1997].*

2.7 Least Dynamic Frequency Rule - LDRF

Essa política proposta por [Aggarwal and Yu, 1997] usa a freqüência de acesso de um objeto como parâmetro para remover objeto do cache. *A cada nova inserção de objetos no* cache, *o conjunto de objetos adjacentes com menor freqüência de acesso é escolhida para remoção. Essa política é uma generalização direta da política LFU para o caso de objetos Web. Está estratégia 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 estratégia é que exige muito tempo para verificação de freqüência de cada objeto no* cache.

2.8 Localized Least Dynamic Frequency - LLDR

Para diminuir o tempo gasto para a verificação e cada objeto no cache, *é proposta um 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 and 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.*

2.9 Particionamento de Espaço para Caches - PART

Na tese de [Murta et al., 1998] é divido o espaço do cache *em partições. Cada partição armazena objetos cujo tamanho pertence a seu 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.*

O algoritmo substitui os objetos apenas entre os de uma mesma classe. Este modelo não permite a retirada de um objeto pequeno para a inserção de um objeto grande, nem retirada de um objeto grande para a inserção de um objeto pequeno.

O trabalho propõe que o gerenciamento do espaço destes cache *seja feito em dois níveis: a organização do espaço e a política de substituição. Para a organização do espaço é proposto o modelo PART.*

Foram utilizadas políticas diferentes nas partições, com o objetivo de obter melhores valores hit rate - HR *e* byte hit rate - BHR. *O modelo PART apresentou soluções para os problemas gerados de variabilidade. O PART impõe restrições baseadas em tamanho para as substituições no* cache *obtendo com isso maior regularidade e melhor desempenho em comparação com o* cache *tradicional. O PART manteve mais arquivos no* cache, *fez menos inserções e retiradas e reteve os arquivos por um tempo maior. Como resultado, o modelo PART demonstrou que é possível atingir bom desempenho simultaneamente em* HR *e* BHR.

2.10 Dinâmica

Essa política utiliza três filas que contêm os mesmos objetos, porém ordenados de maneira diferente. A primeira fila e 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 à freqüência com que foram acessados e o tempo desde seu último acesso [Brandão and Anido, 2001].

2.11 Hybrid

Este algoritmo proposto por [Wooster and Abrams, 1997] calcula 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, banda de conexão, freqüência e o tamanho do objeto. Objetos com menor valor são descartados. O algoritmo apresentou bom desempenho em HR diante dos algoritmos tradicionais, 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. (fórmula Figura 2.2)*

$$\frac{(c_s + \frac{W_b}{b_s}) \times (n_p)^{W_n}}{z_p}$$

Figura 2.2 Fórmula Hybrid

2.12 Frequency-Based Replacement - FBR

A política de [Arlitt and Williamson, 1996] utiliza uma combinação das políticas LRU e LFU na decisão de quais objetos serão mantidos no cache. *O* cache *é dividido em três seções, (Figura 2.3). Cada seção tem o tamanho definido por dois parâmetros,* F(new) *e* F(old) *que determinam a quantidade de espaço para cada seção.* F(new) *determina a quantidade de espaço disponível para a seção* new *e* F(old) *delimita o espaço para a seção* old. *O restante do espaço é dedicado à seção* middle. *Inicialmente os objetos são adicionados na seção* new, *conforme vão sendo inseridos novos objetos nesta seção, os objetos mais antigos, são movidos para a seção* middle, *através da política LRU. Agora, se o objeto continuar a "envelhecer"sem ser referenciado será transportado para a seção* old. *Apenas 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. Logo, não há a necessidade de incrementar o contador porque é usada a política LRU.*

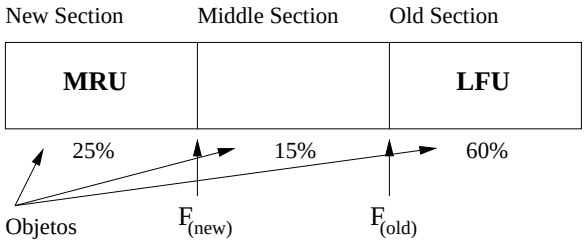

Figura 2.3 Seções usadas na política FBR

2.13 Lowest Relative Value - LRV

Este algoritmo é baseado na recenticidade, tamanho e freqüência do objeto. Os autores encontram funções matemáticas que são aproximações da distribuição dos tempos entre acessos a um mesmo objeto e da probabilidade de mais acessos dado que o objeto foi acessado 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, estatísticas de cada objeto requisitado, atualizadas a cada acesso. Com base nas funções e nos valores das estatísticas, a probabilidade Pr de um objeto ser acessado no futuro é calculada. Pr é diferente para cada objeto e é dependente do tempo. Objetos com menor Pr são descartados. O método é dependente da função de aproximação da distribuição dos tempos entre acessos a um mesmo objeto que, por sua vez, é baseada nas cargas estudadas. A implementação é cara. A função de aproximação utiliza as funções logarítmica e exponencial e constantes empiricamente escolhidas e é calculada freqüentemente para todos os objetos presentes no* cache *[Rizzo and Vicisano, 2000]. Esta política possui a desvantagem de parametrização com funções logarítmicas, que são caras computacionalmente.*

$$D(t) = 0.035log(t+1) + 0.45(1 - e^{\frac{-t}{2\epsilon\theta}})$$

Figura 2.4 Fórmula LRV

2.14 Greedy Dual-Size GD-Size

Esta política proposta por [Cao and Irani, 1997] é também um algoritmo híbrido que considera recenticidade dos acessos, tamanho e custo de busca de um objeto. Os autores mostram que considerar a latência não leva a bons resultados devido à 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 de H do objeto é calculado e acumulado ao valor residual anterior. Portanto, quanto mais recente o acesso ao objeto, maior o seu H e menor sua probabilidade*

de ser retirado do cache. *Essa política não considera o tempo decorrido para recuperar o objeto e a freqüência que o mesmo foi referenciado.*

2.15 Peso

Nesta política proposta por [Pinheiro, 2001] cada objeto deve considerar cinco parâmetros, onde são utilizados na função Peso na escolha dos objetos que serão substituídos:

- *Ti: tempo em que o objeto foi referenciado pela última vez;*
- *Si: tamanho do objeto;*
- *Nrefi: o total de referências;*
- *LATi: tempo decorrido para recuperar o objeto;*
- *Classi: a classe a que o objeto pertence, sendo que, para o estudo foram utilizadas três classes: "HTML/XML/TEXTO", "IMAGENS"e "OUTROS".*

A política utiliza a seguinte função (Figura 2.5) para atribuir um peso para cada objeto presente no cache. *São selecionados para substituição os objetos que tiverem o menor peso. 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 (MAXNref), para evitar o problema apresentado pela política LFU. O tempo gasto para recuperar um objeto foi normalizado apenas pelo seu tamanho (LATi/Si), pois o tempo decorrido para recuperar um objeto apesar de estar diretamente relacionado ao tamanho do objeto, também é enunciado 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 são considerados neste 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 (Ti), sendo que quando maior for Ti, menor será o peso (Pi) do objeto. A utilização do parâmetro (Classi) foi motivada devido a diferentes tipos de objetos apresentarem taxas de atualização diferentes; por exemplo, documentos hipertextos são atualizados com mais

freqüência que arquivos de áudio 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 Classi são: 0.10 para objetos da classe "HTML/XML/TEXTO"; 0.15 aos objetos da classe "IMAGENS"; e 0.20 aos objetos remanescentes.

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

Figura 2.5 Fórmula Peso

2.16 Segmented LRU - SLRU

Essa política foi proposta inicialmente para ser usada em cache *de disco. Ela considera tanto a freqüência quanto quão recentemente um objeto foi requisitado para realizar a substituição. dividindo o* cache *em dois segmentos: um protegido (para armazenar objetos que são acessados com mais freqüência) e outro não-protegido. Quando há uma solicitação de um objeto pela primeira vez, o mesmo é inserido no segmento não-protegido. Quando ocorre uma requisição do objeto e o mesmo se encontra no* cache*, ele é removido para o segmento protegido. Os dois segmentos são gerenciados pela política LRU. 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 recentes no segmento protegido são removidos [Arlitt et al., 2000].*

2.17 Semantic Data Caching and Replacement

Em [Dar et al., 1996] foi proposto um modelo semântico de substituição de objetos em cache *em um sistema de banco de dados cliente-servidor e foi comparado com as estratégias* page caching *e* tuple caching*. Este modelo é baseado em três idéias chave. Primeira, o cliente mantém uma descrição semântica dos dados em seu* cache*, em vez de manter uma lista das páginas ou tuplas. O processamento de query faz uso das descrições semânticas para*

determinar quais dados estão disponíveis localmente no cache *e quais dados será necessário buscar no servidor. Segunda, as informações usadas pela política de substituição de* cache *são mantidas de maneira adaptativa por regiões semânticas, que são associadas com conjuntos de tuplas. O uso de regiões semânticas evita o* overhead *de armazenamento da abordagem* tuple caching *e, diferentemente da abordagem* page caching*, é imune a mau* clustering *das tuplas nas páginas. Terceira, mantendo uma descrição semântica dos dados no* cache *encoraja o uso de funções de valor sofisticadas, em determinar informações de substituição. As funções de valor que incorporam noções semânticas de localidade podem ser criadas para aplicações tradicionais baseadas em* querys *como também para aplicações emergentes, como bancos de dados movéis.*

2.18 Proxy Cache Replacement Algorithms: A History-Based Approach

Neste trabalho é apresentado uma nova abordagem para substituição de objetos em cache*, propondo para vários algoritmos estendê-los de modo que possam utilizar os registros de acessos - histórico - como critério para uma substituição de objetos mais detalhada, os algoritmos estendidos são: LRU, SLRU, MFU e LFU. Segundo [Vakali, 2001], a idéia principal é manter um registro das referências passadas para os objetos, ou seja, um histórico dos instantes de tempo dos últimos acessos e assim para cada substituição de objetos poder avaliar melhor qual objeto deve ser removido do* cache*. A conclusão do artigo mostrou a importância do histórico como parâmetro na substituição de objetos em* cache*, por terem os algoritmos estendidos alcançado um desempenho melhor que os seus correspondentes originais.*

2.19 Semantic Collaborative Web Caching - Temperature

Como é proposto por [Brunie et al., 2002] foi criada uma Arquitetura Colaboradora de Proxies*, baseados em tópicos de relevância e a construção dinâmica de comunidades virtuais. Nesta arquitetura, a* temperatura *de um documento ou assunto reflete seu interesse atual en-*

tre as comunidades. Foi proposta uma estrutura hierárquica para a representação semântica entre os objetos. Essa estrutura conecta nos vértices os objetos e são associados pesos nas arestas e nos objetos da hierarquia. Os pesos são chamados de temperatura, isto é, a temperatura é valor específico para cada documento, ela representa a probabilidade de um documento for requisitado em um futuro próximo. Mais precisamente, é a síntese entre o número de requisições de um documento e o último intervalo de tempo da requisição do vértice semântico onde este documento está na estrutura (Figura 2.6).

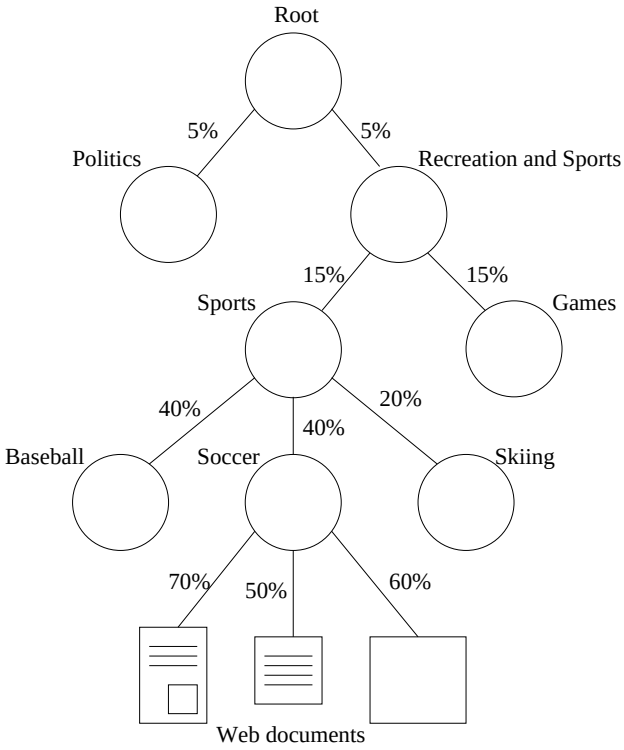

Figura 2.6 Estrutura hierárquica

Durante o processo de simulação, os documentos que forem mais acessados serão aquecidos *e logo, os que tiverem menos acessos serão* esfriados. *Dentre um determinado período de tempo este processo permite refletir para toda a estrutura hierárquica. Para a simulação do experimento foi utilizada a estrutura do Yahoo!. A simulação teve um total de 5000 acessos e foi demonstrado que o algoritmo chamado* temperature *teve um desempenho superior ao algoritmo LRU.*

2.20 Uso de Caches na Web - Influência das políticas de substituição de objetos - MeMoExp

Nesta dissertação de [de Oliveira, 2004], é analisada a influência provocada pelas políticas de substituição de objetos em cache *na Web. Para tal, é feito uma levantamento 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* cache *para a Web. Também, foi desenvolvida uma nova política, chamada* MeMoExP. *Esta política utiliza os conceitos de Média Móvel para otimizar* HR *e* BHR. *As simulações realizadas mostraram que a política* MeMoExP *segue a mesma tendência da FBR, tida como eficiente na literatura.*

2.21 Entendendo os Efeitos da Localidade de Referência em Hierarquias de Caches na Web

O uso de caching *na Web está muitas vezes associado a uma organização hierárquica. Sistemas de* caches *muitas vezes são configurados em hierarquia na tentativa de melhorar a qualidade do serviço percebida pelos usuários e diminuir o tráfego na rede. O* cache *do navegador, localizado na máquina do cliente, é o nível mais baixo da hierarquia. Um nível acima estão os* caches *das intranets, que consistem de* proxies *de universidades e organizações. Quando subimos na hierarquia, temos os* proxies *regionais e assim por diante. Uma requisição que não pode ser satisfeita por um* proxy *pode ser enviada para o* proxy *imediatamente acima da hierarquia até que ela possa ser atendida, tendo como última opção o servidor destino. O trabalho proposto em [Benevenuto et al., 2005] apresenta uma extensiva avaliação dos efeitos de filtragem que ocorrem em servidores* proxy *organizados com uma hierarquia de* caches. *Além disso, apresenta uma avaliação dos efeitos que a localidade de referência de requisições sofre ao passar por uma hierarquia de* caches. *O estudo propõe o uso de entropia média para a comparação da localidade de referência entre seqüências de requisições e também fornece o arcabouço necessário para que a entropia possa ser calculada dinamicamente por um servidor* proxy. *Com o intuito de entender melhor os efeitos da localidade de referência em um sistema de hierarquia de* caches, *foi utilizado o modelo*

ADF: Agregação (A), Desagregação (D) e Filtragem (F) proposto em [Fonseca et al., 2003]. Nas simulações, foi avaliado o comportamento da entropia média quando uma seqüência de requisições passa por uma hierarquia de caches*, variando o tamanho destes* caches*. O tamanho dos* caches *foi incrementado seguindo uma potência de 2, indo de 1 megabyte até 16 gigabytes, sendo que o tamanho máximo foi escolhido com base no tamanho total da carga (ponto no qual o* cache *se torna infinito). Quatro políticas de reposição de* cache *foram consideradas: LRU de [Dilley and Arlitt, 1999], LFU-Aging de [Arlitt et al., 2000], GD-Size de [Cao and Irani, 1997] e LRU-Threshold de [Abrams et al., 1996]. No experimento, foi avaliado o impacto dessas políticas de substituição combinadas em níveis diferentes de hierarquia de* caches*. O resultado mostrou que configurações heterogêneas de políticas de* caches *tiram um melhor proveito da localidade de referência e produzem melhores taxas de acertos.*

2.22 Um servidor para a classificação e filtragem de conteúdo na Internet

Nos últimos anos o grande crescimento da Web, que inclui conteúdos impróprios para algumas classes de usuários, vem sendo acompanhado pelo aparecimento de novos dispositivos móveis de acessos. Nesse contexto, um dos grandes desafios, é a adaptação dinâmica de conteúdo, a fim de que esses dispositivos possam acessar determinado conteúdo independentemente de seu formato original, aliado à oferta de uma variedade de serviços de valor agregado, tais como: escaneamento de vírus, tradução de linguagens e filtragem de conteúdo. Este trabalho de [Forte et al., 2005] propõe e implementa um servidor de classificação e filtragem de conteúdo Web, que é um dos componentes de uma arquitetura de adaptação de conteúdo em desenvolvimento por [Claudino, 2002].

A independência de dispositivos, navegadores e sistemas operacionais é obtida por meio da implementação dessa arquitetura de adaptação de conteúdo em atender as preferências dos usuários e limitações dos dispositivos de acesso é obtida a partir de uma política de adaptação baseada em perfis e regras. Tendo como objetivo uma arquitetura de adaptação de conteúdo aberta o Internet Content Adaptation Protocol - ICAP *de [Elson and Cerpa, 2003] é empregado, o que permite o uso de outros servidores de adaptação disponíveis no mercado. O protocolo ICAP tem como objetivo possibilitar a comunicação entre equipamentos de borda*

e servidores de adaptação, visando adaptar o conteúdo o mais próximo do cliente, o que propicia uma melhor capacidade de personalização aliada a ganhos de velocidade.

Baseado num modelo aberto e expansível, o servidor proposto tem como objetivo estender as capacidades dos servidores de classificação e filtragem de conteúdo, de modo a atender as demandas de tecnologias de clientes Web, cada vez mais ubíquas e buscando a satisfação das preferências do usuário. O servidor apresenta uma solução flexível, uma vez que a cada requisição vários aspectos do processo de classificação e filtragem de conteúdo podem ser definidos. É importante ressaltar que esta personalização é realizada sem ameaçar a privacidade dos usuários, pois os seus perfis são armazenados apenas no proxy*, o que permite também a utilização simultânea de um mesmo servidor por vários provedores e/ou empresas.*

Capítulo 3

Estratégia LSR

Nesse capítulo é apresentado o algoritmo LSR na forma original, ou seja, sem histórico. Também neste capítulo são apresentadas as suposições básicas para o perfeito funcionamento do algoritmo, a obtenção dos objetos do Yahoo! para a simulação, as categorias de acessos, a obtenção das seqüências de acessos e o resultado do desempenho obtido pelo algoritmo.

3.1 Heurística

Em contraste com os algoritmos atualmente conhecidos, que se baseiam em propriedades físicas dos objetos, o LSR baseia-se na semântica da informação contida nos objetos: o LSR tende a favorecer a permanência no cache *os objetos que possuem maior afinidade entre si, com relação à semântica da informação, eliminando do* cache *os objetos que tendem a ser de menos interesse para os clientes. Esse algoritmo analisa os objetos alocados em* cache *e separa de acordo com a sua semântica. O princípio do algoritmo é dar prioridade aos objetos que possuam maior relação semântica com o objeto que será inserido no* cache*, ou seja, os objetos que possuem maior relação semântica com o novo objeto tenderão a permanecer no* cache

3.2 Estrutura hierárquica - Grupos semânticos

Uma abordagem tipicamente utilizada para tal é a definição de uma taxonomia, ou seja, uma estrutura hierárquica de termos correspondentes a assuntos, na qual cada objeto deve se enquadrar. Assim, o conjunto de objetos associados a um termo da hierarquia compõe

um grupo semântico. Exemplos mundialmente conhecidos dessas taxonomias são o Yahoo! e o DMOZ.

Para encontrar informações disponíveis na Internet, usando o Yahoo!, os usuários têm de digitar o assunto que procuram. O Yahoo!, por sua vez, consulta seu banco de dados e apresenta uma breve descrição de todos os sites da Internet, juntamente com os respectivos URLs, que mais se assemelham ao conteúdo solicitado. Por exemplo, ao pesquisar o assunto Photography*, o Yahoo! trará milhares de documentos que mais se assemelham ao conteúdo solicitado, classificado pela probabilidade do* site *conter a informação procurada. O Yahoo! traz, ainda, consigo uma série de grupos de assuntos (grupos semânticos) relacionados com* Photography *(Veja Figura 3.1).*

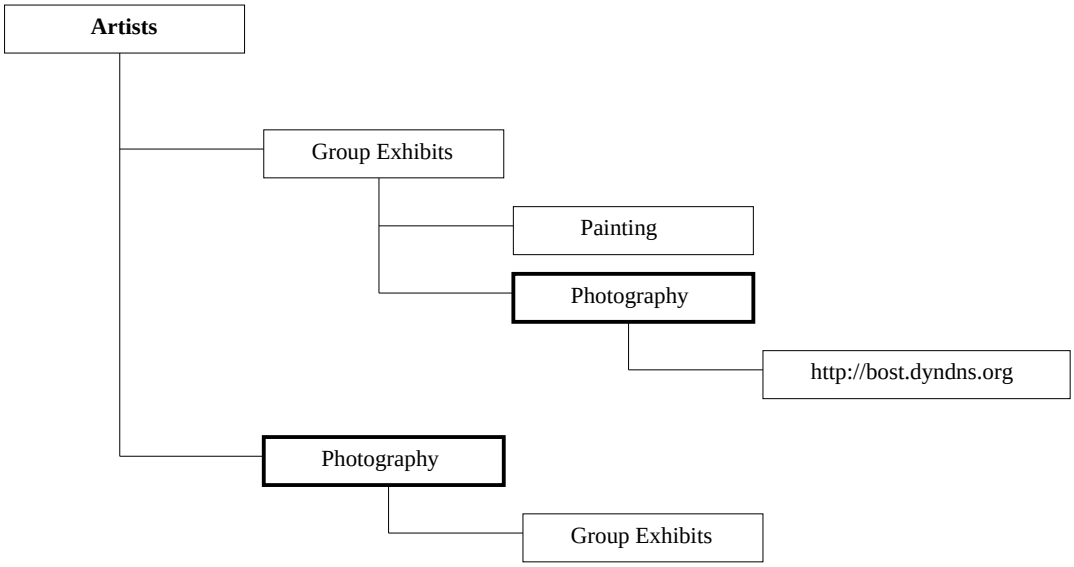

Figura 3.1 Estrutura hierárquica parcial do grupo *Artists* do Yahoo!

3.3 Suposições Básicas

3.3.1 Semântica do objeto

O algoritmo supõe que é possível obter a semântica de cada objeto acessado. Cada objeto pertence a um grupo associado na estrutura hierárquica. Ao percorrer esta estrutura e inserir um objeto nela, o algoritmo identifica quais grupos e subgrupos o objeto pertence.

3.3.2 Estabilidade Semântica

O algoritmo LSR assume que cada usuário tenha por um certo período de tempo um padrão de acesso, isto é, que um usuário permanece um certo tempo *pesquisando um mesmo assunto antes de passar a pesquisar outro. Este padrão de acesso é denominado* estabilidade semântica.

3.3.3 Thread única de interesse

Através da estabilidade semântica representada pelos padrões de acessos dos usuários são identificados interesses comuns entre eles. O algoritmo LSR foi projetado para identificar apenas um interesse por vez, aqui denominado thread única de interesse. *Como exemplo, segue a seqüência de acessos da Tabela 3.1.*

Primeiramente, na Tabela 3.1, foram acessados pelos usuários o grupo semântico Informática, *depois o grupo* Política *e por sua vez o grupo* Esportes. *O algoritmo LSR foi projetado para identificar apenas um interesse por vez. É possível perceber que existe um padrão de acesso muito rígido. Presumimos que no mundo real os usuários não se comportem assim com apenas um assunto de interesse por vez a pesquisar na Internet. Esse comportamento não é muito flexível. Perfeitamente, os usuários podem acessar dois, três ou mais assuntos simultaneamente. Este foi um dos fortes motivos para a extensão do algoritmo proposto neste trabalho.*

Usuário	*Grupos semânticos*		
	Informática	*Política*	*Esportes*
William	x		
James	x		
Steve	x		
Linus	x		
James		x	
Steve		x	
Linus		x	
William		x	
Steve			x
Linus			x
James			x
William			x

Tabela 3.1 Seqüência de acessos do LSR

3.4 Experimento e validação

3.4.1 Obtenção de objetos para a simulação

Foi utilizado o Yahoo! para a obtenção dos objetos. Como amostra foram utilizados 983 objetos, para cada objeto foram extraídos a semântica, a url e o seu tamanho correspondente em bytes. A Tabela 3.2 contém um exemplo da amostragem.

3.4.2 Categorias de acessos

Num certo intervalo de tempo, para um certo gerenciador de cache*, um conjunto de objetos são acessados. Dentre esses, alguns são acessados somente uma vez no período considerado, outros duas vezes, outros três vezes, e assim por diante. Assim, podemos classificar os objetos de acordo com a quantidade de vezes que são acessados, isto é, podemos definir categorias de acesso. Dessa forma, a categoria de acesso* n *corresponde a todos os objetos acessados* n *vezes no período de tempo considerado. Observando-se o (log) de acessos de um*

Semântica	*Objeto (URL)*	*Bytes*
Root.Arts.Art History.Education	www.artstar.com	4115
Root.Arts.Organizations.Arts Commissions	www.wa.gov.art	5275
Root.Arts.Art History.Institutes	www.rkd.nl	478
Root.Arts.Artists.Ceramics	www.ninacam.com	1207
Root.Arts.Artists.Ceramics	www.anngero.com	1702
Root.Arts.Organizations.Councils	www.acbv.org	1659

Tabela 3.2 Exemplos de objetos do Yahoo!

gerenciador de cache *por um certo tempo, como sumariza o gráfico na Figura 3.2, constata-se que, tipicamente, quanto maior o número da categoria, menor é a quantidade relativa de objetos pertencentes à mesma. O gráfico refere-se ao* log *de um servidor* proxy Squid - *um dos gerenciadores de* cache *mais utilizados - do departamento de informática da PUC-PR, por um certo período, totalizando 2.000.000 de acessos a 866.915 objetos*

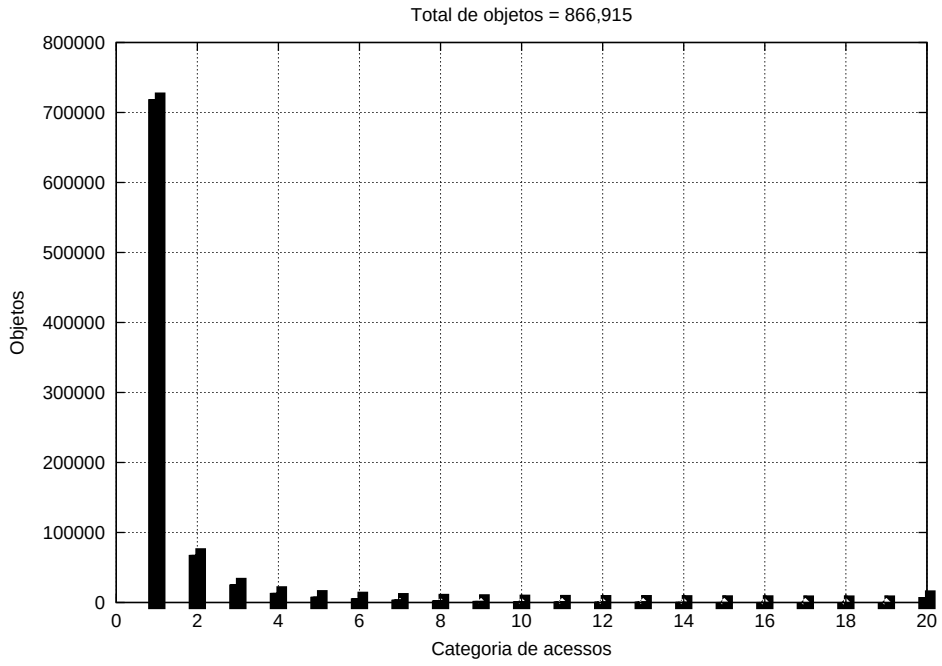

Figura 3.2 Categoria de acessos para simulação LSR

Categoria de Acesso	*Quantidade de Objetos por Categoria*	*Probabilidade*	*Intervalo*	
			Limite Inferior	*Limite Superior*
c	q_c	$P_c = \frac{q_c}{\sum_c q_c}$	$\sum_{i=1}^{c-1} P_i$	$\sum_{i=1}^{c} P_i$
1	20	0.47	0.00	0.47
2	10	0.23	0.47	0.70
3	6	0.14	0.70	0.84
4	4	0.09	0.84	0.93
5	2	0.05	0.93	0.98
6	1	0.02	0.98	1.00
Total	43	1		
	$\sum_c q_c$	$\sum_c P_c$		

Tabela 3.3 Distribuição de probabilidade para aplicação do Método de Monte Carlo

3.4.3 Obtenção de seqüências de acessos a objetos

Nesta fase foi feita uma aplicação que permitisse a criação dos acessos aleatórios aplicando o Método Monte Carlo. Para cada categoria determina-se a Quantidade de objetos*, a* Quantidade de acessos *(multiplicando-se a quantidade de objetos pelo próprio número da categoria), a* Probabilidade *(dividindo-se a quantidade de acessos pelo total de acessos), o* limite inferior *e o* limite superior *(de acordo com a probabilidade encontrada), conforme mostra a Tabela 3.3. Assim, aplicam-se os seguintes passos para gerar cada acesso a objeto:*

(1) *Escolha de um número real aleatório entre zero e um; verifica-se onde este número se encontra de acordo com os intervalos definidos pelas colunas* Limite Inferior *e* Limite Superior *para localizar a categoria sorteada.*

(2) *Escolha de um segundo número inteiro aleatório entre 1 e o valor contido na coluna* Quantidade de Objetos por Categoria*; este segundo número aleatório indicará o objeto escolhido para o acesso.*

Foram geradas seqüências para 1.000, 2.000, 3.000, 4.000, 5.000, 6.000, 7.000, 8.000, 9.000, 10.000, 15.000, 20.000 e 30.000 acessos. Essas 13 seqüências foram submetidas para as quatro estratégias (LSR, SIZE, LRU e LFU) em análise, para os seis seguintes tamanhos de cache*: 1.000.000, 1.5000.000, 2.000.000, 2.5000.000, 5.000.000 e 10.000.000. Portanto, foram realizadas 4 x 13 x 6 = 312 simulações de* cache*. Cada uma das seqüências sofreu um*

rearranjo na ordem dos acessos a fim de contemplar o comportamento suposto dos usuários que acessam objetos na Internet, isto é, que um usuário permanece um certo tempo *pesquisando um mesmo assunto antes de passar a pesquisar outro. Tal rearranjo foi realizado com a ordenação dos acessos de acordo com o primeiro nível de assuntos nos caminhos dos acessos.*

3.5 Análise de desempenho

A observação final e mais importante é que, praticamente em todas as situações, verifica-se que a estratégia LSR tem maior probabilidade de acerto que as demais estratégias experimentadas. Ressaltamos que o experimento foi realizado, para todas as estratégias, com apenas uma thread de interesse *por vez. Não foi feita uma análise dos dados a fim de se determinar quanto maior é a probabilidade de acerto da estratégia LSR em relação às outras estratégias, visto que a amostragem de dados utilizada para a simulação ainda deve ser ampliada para dar maior precisão ao experimento.*

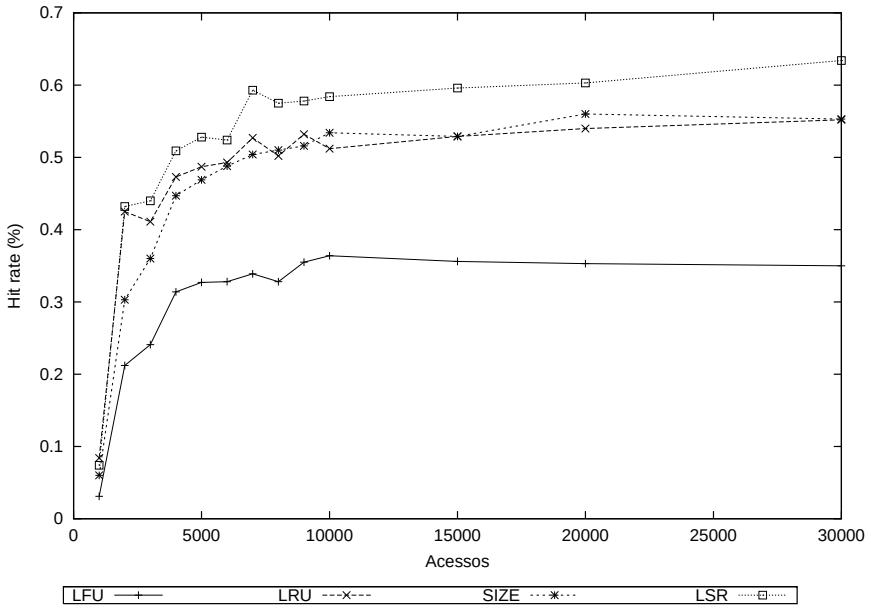

Figura 3.3 Desempenho para um *cache* de tamanho de 1 megabytes

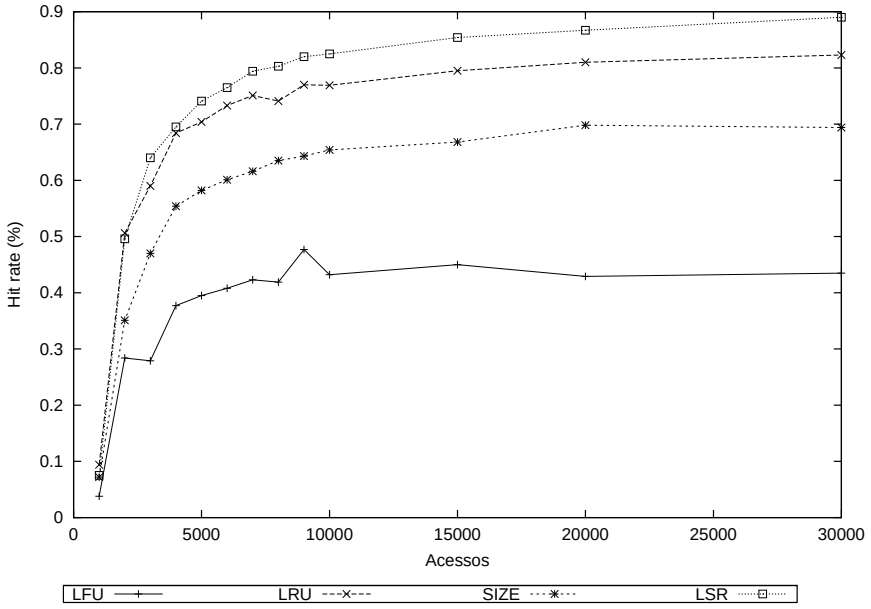

Figura 3.4 Desempenho para um *cache* de tamanho de 2 megabytes

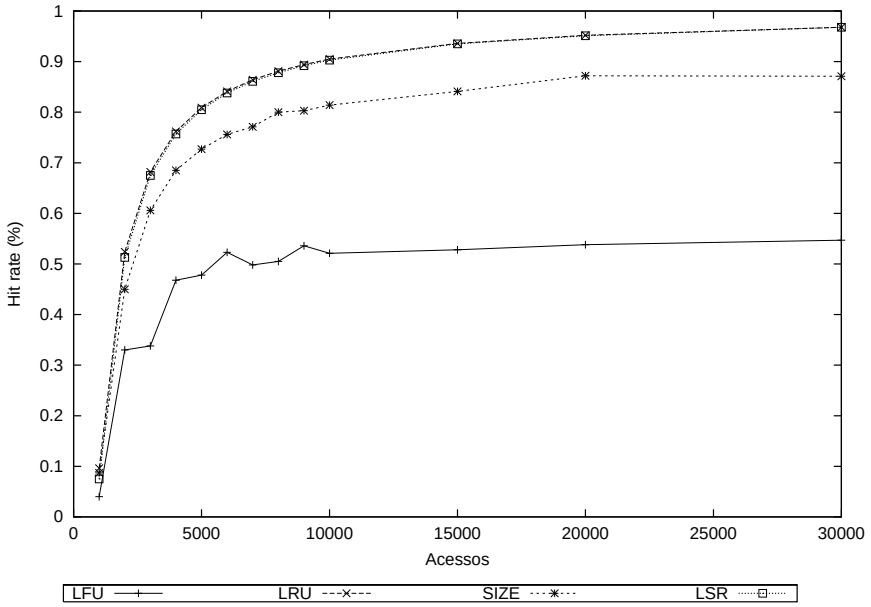

Figura 3.5 Desempenho para um *cache* de tamanho de 5 megabytes

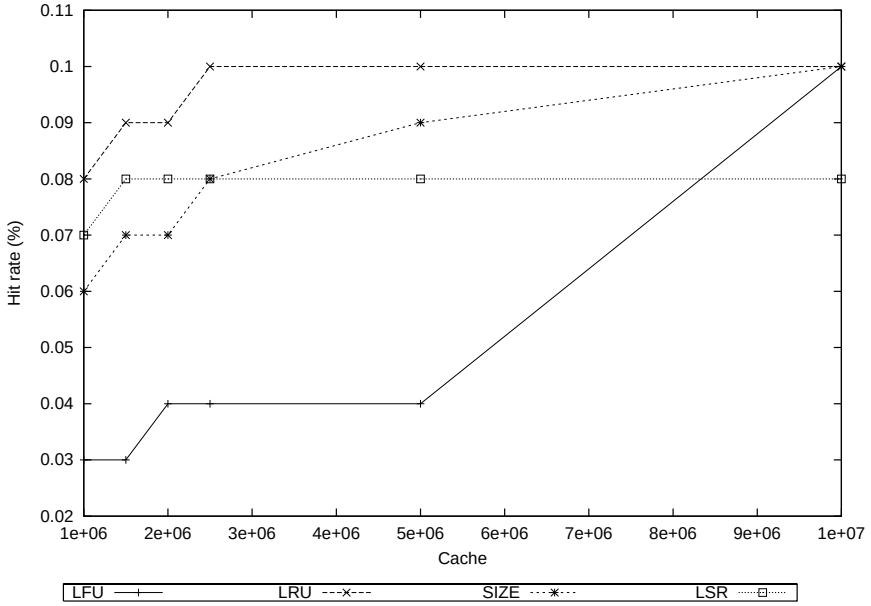

Figura 3.6 Desempenho para 1000 acessos

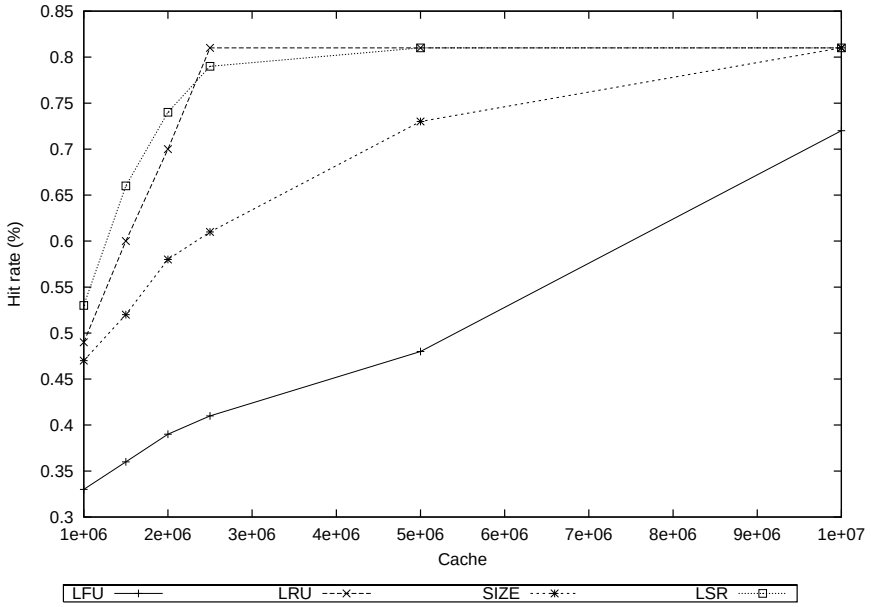

Figura 3.7 Desempenho para 5000 acessos

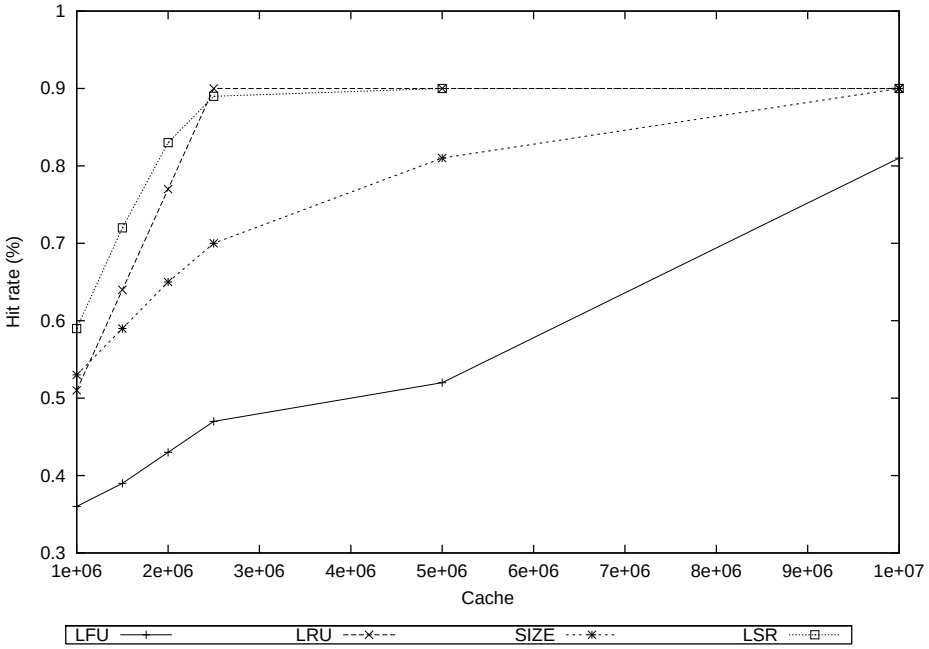

Figura 3.8 Desempenho para 10000 acessos

Capítulo 4

Estratégia LSR/H

Este capítulo apresenta os conceitos básicos necessários para o entendimento do novo algoritmo com histórico chamado LSR/H. Também neste capítulo é apresentado o histórico de acessos, a atribuição de pesos a grupos semânticos, o exemplo e a implementação do algoritmo proposto.

4.1 Heurística

O algoritmo Least Semantically Related/History - LSR/H *tem como objetivo manter em* cache *os objetos pertencentes aos grupos semânticos de maior interesse. Para isso, será mantido um histórico de acessos a objetos para atribuir pesos aos grupos semânticos, dando mais relevância para os últimos acessos registrados, o que dará preferência à remoção dos objetos do grupo semântico de menor interesse, ou seja, os objetos que naquele histórico foram menos acessados. Ao localizar o grupo desejado para remoção, o algoritmo irá utilizar como critério de desempate, se necessário, uma política tradicional, por exemplo, SIZE para a ordem da remoção dentro do grupo semântico.*

4.2 Estabilidade semântica múltipla

O algoritmo LSR/H permite que cada usuário possa variar seu padrão de acesso e não tenha apenas um único interesse por vez. É perfeitamente normal que o usuário tenha mais do que um interesse quando for buscar informações na Internet. A suposição quanto ao padrão de acesso a objetos é mais relaxada e mais realista. Ainda espera-se que cada usuário tenha

um certo tipo de estabilidade semântica, mas aceita-se a possibilidade de haver múltiplos usuários, com interesses distintos, permitindo assim, estabilidade semântica múltipla.

4.3 Múltiplas threads de interesses

Considerando um certo período de tempo no qual um certo grupo de usuários faz acessos a objetos na Internet, tipicamente, há assuntos, que se destacam por serem de maior interesse, isto é, os objetos acessados tendem a se concentrar em alguns grupos semânticos bem definidos. Basicamente, isso é influenciado por questões de tempo e espaço. Por exemplo, em um grupo de usuários cuja atividade principal é fazer pesquisa sobre borboletas, certamente os objetos acessados cairão, em grande parte dos casos, no grupo semântico insetos e seus subgrupos. Por outro lado, quando algum fato extraordinário acontece, como uma guerra, um terremoto ou uma olimpíada, é muito provável que os usuários de um certo grupo de trabalho, independente de sua atividade típica, procurarão informações sobre aquele fato.

O LSR/H, através do seu histórico de acessos, permite identificar quais os assuntos mais quentes *em cada momento, considerando toda a comunidade de usuários e, com base nisso, remover do* cache *os objetos relacionados a assuntos menos relevantes. O algoritmo permite com seu histórico uma maior flexibilidade para que sejam identificados mais de que um interesse por vez, aqui denominada* múltiplas threads de interesses. *O exemplo da Tabela 4.1 mostra uma seqüência de acessos para o LSR/H*

Os usuários acessam mais de um interesse por vez, e podemos identificar três threads *de interesse neste momento. Primeiramente, o grupo semântico* Informática *com 5 acessos, depois o grupo* Política *com 4 acessos e por fim o grupo* Esportes *com apenas 3. Os acessos estão mais distribuídos e também mais realista com o que acontece normalmente na Internet. A responsabilidade de identificar e priorizar os grupos semânticos na hora da remoção dos objetos, é de inteira responsabilidade do algoritmo, em conjunto com o seu registro (histórico) de informações nele contido.*

Usuário	*Grupos semânticos*		
	Informática	*Política*	*Esportes*
William	x		
William		x	
Steve	x		
Linus			x
Linus		x	
Steve	x		
James		x	
William			x
Steve			x
Steve	x		
James		x	
William	x		

Tabela 4.1 Seqüência de acessos do LSR/H

4.4 Threads e Categorias de Acesso

A Tabela 4.2 mostra uma possível distribuição de acessos a objetos em um certo período de tempo, para um certo gerenciador de cache. *Foram acessados objetos de seis grupos semânticos do primeiro nível da hierarquia, identificados na tabela como A a F. Foram identificadas cinco categorias de acesso, numeradas de 1 a 5. Cada célula da tabela mostra a quantidade de objetos acessados por grupo semântico e por categoria de acesso. Por exemplo, 20 objetos do grupo semântico A foram acessados somente uma vez, por isso pertencem à categoria 1 e representam 20 acessos, enquanto que 6 objetos do mesmo grupo semântico A foram acessados duas vezes, por isso pertencem à categoria 2 e representam 12 acessos, e assim sucessivamente pelas categorias 3, 4 e 5, totalizando 36 objetos que representam 70 acessos. Da mesma forma, para o grupo semântico B, há um total de 144 objetos que representam 280 acessos. Assim, considerando todos os grupos semânticos, houve um total geral de 700 acessos sobre 360 objetos. Portanto, o total de acessos a objetos do grupo semântico A corresponde a 10% do total geral de acessos. Para os grupos semânticos B, C, D, E e F, essa taxa é de 40%, 5%, 5%, 35% e 5%, respectivamente. Nota-se claramente, que*

Taxa de Acessos (%)	*Categoria Acessos x Grupo Semântico*	*1*	*2*	*3*	*4*	*5*	*Total*
10	A	20(20)	6(12)	4(12)	4(16)	2(10)	36(70)
40	B	80(80)	24(48)	16(48)	16(64)	8(40)	144(280)
5	C	10(10)	3(6)	2(6)	2(8)	1(5)	18(35)
5	D	10(10)	3(6)	2(6)	2(8)	1(5)	18(35)
35	E	70(70)	21(42)	14(42)	14(56)	7(35)	126(245)
5	F	10(10)	3(6)	2(6)	2(8)	1(5)	18(35)
100		200(200)	60(120)	40(120)	40(160)	20(100)	360(700)

Tabela 4.2 Possível distribuição de acessos a objetos com presença de dois grupos semânticos de maior interesse

os grupos semânticos B e E destacam-se como grupos de maior interesse naquele período de tempo, isto é, são threads de maior interesse *(assuntos quentes). O algoritmo LSR/H tende a identificar e manter no* cache *os objetos pertencentes destas* threads.

4.5 Histórico de Acessos

O histórico de acessos é uma forma de registrar os últimos acessos a grupos semânticos, permitindo identificar quais grupos semânticos são de maior interesse em um certo momento e, assim, quando necessário, remover objetos preferencialmente de grupos semânticos de menor interesse. A expectativa é, seguindo essa heurística, ter como conseqüência um incremento na taxa de acerto.

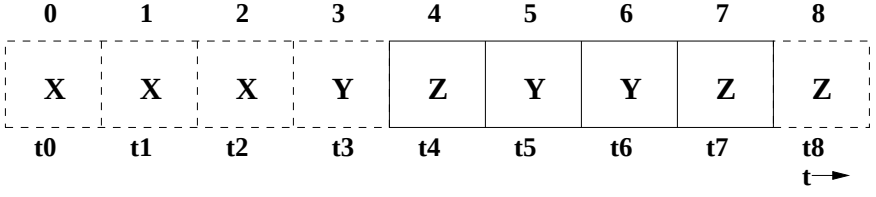

Figura 4.1 Histórico de acessos

Um histórico possui um tamanho de armazenamento finito, normalmente pré-fixado. Quando houver a necessidade, o primeiro registro do histórico, ou seja, o mais antigo será removido. A Figura 4.1 representa uma seqüência de 9 registros (numerados de 0 a 8) de acessos a grupos semânticos. Os valores x, y e z correspondem aos grupos semânticos dos objetos acessados. Os valores t0 a t8 indicam o instante de tempo em que ocorreu cada acesso. No instante t0, o histórico tem tamanho 0, isto é, nenhum acesso está registrado ainda. No instante t1, o histórico tem tamanho 1: registra um acesso ao grupo x, ocorrido no instante t0. No instante t2, o histórico tem tamanho 2: registra o acesso ao grupo x, ocorrido no instante t0, e o acesso ao mesmo grupo x, ocorrido no instante t1. O histórico continua crescendo da mesma forma nos acessos seguintes, até atingir o seu tamanho máximo permitido. Supondo que esse tamanho máximo seja 4, o histórico atingirá o seu limite logo após o acesso no instante t3. Assim, no instante t4, ao acessar o grupo z, para que este acesso fique registrado no histórico, o registro de acesso mais antigo presente no histórico, isto é, o acesso ao grupo x, ocorrido no instante t0, será removido do histórico. Da mesma forma, no instante t5, para que fique registrado no histórico o acesso ao grupo y, deverá ser removido o registro de acesso ao grupo x, no instante t1.

4.6 Atribuição de pesos a grupos semânticos

O histórico de acessos permite atribuir pesos aos grupos semânticos contidos na estrutura hierárquica para fins de remoção de objetos do cache. *Serão removidos, preferencialmente, os objetos de grupos com menor peso, isto é, de menos interesse no momento da remoção. Logo, serão mantidos em* cache *os objetos pertencentes aos grupos semânticos de maior interesse.*

4.6.1 Função linear

Na atribuição dos pesos é utilizada a ordem em que os grupos semânticos se encontram no vetor de histórico na hora da remoção, ou seja, são atribuídos na estrutura hierárquica os índices do vetor onde estão armazenados os grupos semânticos representados pelos últimos acessos. Caso o histórico tenha tamanho 10, os valores a serem repassados para estrutura

Semântica	*Peso*
Top.Computers.Algorithms	10
Top.Computers.Software	9
Top.Business	8
Top.Games	7
Top.Computers.Algorithms	6
Top.Computers.Internet.Protocols.IP	5
Top.Computers.Internet.Protocols.HTTP	4
Top.Games	3
Top.Business	2
Top.Computers.Internet.Protocols	1

Tabela 4.3 Histórico com dez acessos

hierárquica serão de 1 a 10. A Tabela 4.3 mostra os dez últimos acessos, neste exemplo, o último acesso é Top.Computers.Algorithms *com peso 10. A ordem privilegia sempre os últimos acessos com maior peso.*

Na estrutura hierárquica da Figura 4.2 serão repassados os pesos armazenados pelo histórico de acessos representado pela Tabela 4.3. Como exemplo, existem seis acessos do grupo semântico Computers*, a saber:*

- Computers.Algotithms
- Computers.Software
- Computers.Algorithms
- Computers.Internet.Protocols.IP
- Computers.Internet.Protocols.HTTP
- Computers.Internet.Protocols

Os seus pesos são, respectivamente, 10, 9, 6, 5, 4, 1. Estes valores representam a soma dos acessos do grupo Computers *com peso 35. Para contabilizar os pesos não é necessário que o acesso aconteça apenas no grupo* Computers*, a soma também contabiliza os acessos dos grupos semânticos filhos de* Computers.

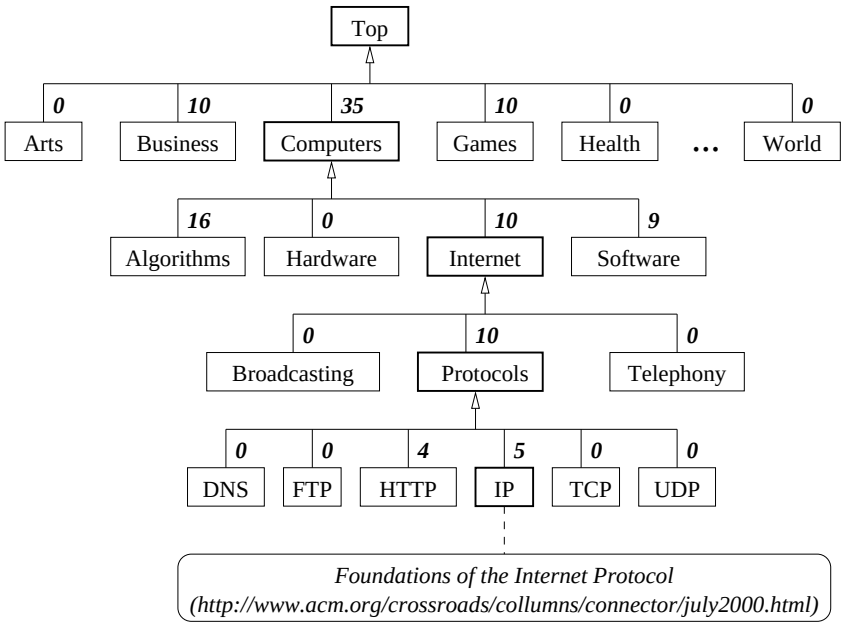

Figura 4.2 Hierarquia semântica com pesos atribuídos pela função linear

4.6.2 Função exponencial

Outra função que poderia ser utilizada para dar pesos aos acessos seria uma função exponencial. Está função mudaria a ordem de remoção dos objetos e traria uma nova taxa de desempenho do cache. *Este experimento não foi realizado neste primeiro momento, fica aqui a oportunidade de ser desenvolvido como um trabalho futuro.*

4.7 Exemplo do LSR/H

A Figura 4.3 mostra um exemplo de um cache *num certo estado, no qual deseja-se inserir um novo objeto, mas que, para tal, é necessário remover um ou mais objetos. Neste exemplo, o* cache *tem capacidade de 150 Kb. Os objetos acessados são classificados segundo uma hierarquia com três grupos semânticos no primeiro nível: A, B e C. Inicialmente Figura 4.3a, o* cache *está totalmente ocupado e o histórico armazena os três últimos acessos que, a partir do mais antigo para o mais recente, são os grupos A (devido a um acesso ao objeto #29), B (devido a um acesso ao objeto #35) e A (devido a um acesso ao #25). Para a inserção do novo objeto #63 com tamanho 50 kb é necessário remover um ou mais objetos*

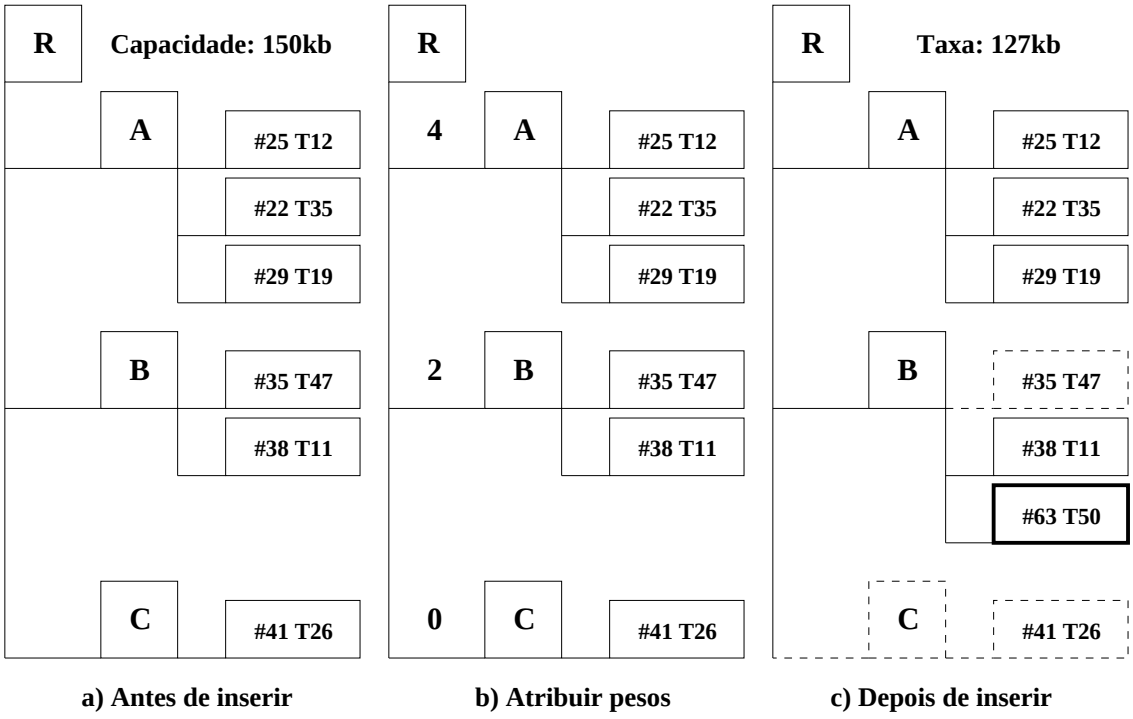

Figura 4.3 Exemplo de uma hierarquia com objetos no *cache*

do cache. *Antes da remoção o algoritmo atribui pesos aos grupos semânticos de acordo com o histórico de acessos. Estes pesos têm o valor das próprias posições em que se encontram no histórico, sempre iniciado por 1 até o último registro de acesso armazenado, que neste caso tem peso 3. Assim, o grupo A possui peso 4, correspondente à soma dos pesos referentes aos registros de acessos 1 e 3, referentes aos objetos #29 e #25, respectivamente. E o grupo B possui peso 2 devido ao segundo registro de acesso, referente ao objeto #35. A Figura 4.3b mostra os grupos semânticos com os respectivos pesos, conforme discutido. Na hora da remoção, o algoritmo localiza o grupo semântico de menos interesse naquele momento, de acordo com o histórico de acessos, que neste caso é o grupo C, que tem peso zero. Por isso, o objeto removido será o #41, cujo tamanho é 26 kb, faltando ainda 24 kb. Como o grupo semântico C não possui mais objetos, a remoção deve continuar em outro grupo: o de segundo menor peso, que neste caso é o grupo B, que tem peso 2. Além disso, como não existem mais objetos no grupo semântico C, o algoritmo remove o próprio grupo. O grupo B contém os objetos #35 (com tamanho 47 kb) e #38 (com tamanho 11 kb). Como critério de desempate, o algoritmo utiliza a política SIZE dentro do grupo, por isso foi removido o objeto #35. Agora não há mais necessidade de remoção, pois a soma dos tamanhos dos objetos*

removidos dá 73 kb, que é maior que o tamanho do objeto que se deseja inserir (50 kb). Portanto, o novo objeto pode ser inserido e ainda haverá um espaço livre de 23 kb no cache. *O algoritmo zera todos os pesos, insere o objeto #63 no histórico e também no* cache, *como mostra a Figura 4.3c. Agora os três últimos grupos semânticos acessados são: B (objeto #35), A (objeto #25) e o B (objeto #63). O registro de acesso ao grupo A (referente ao objeto #29) foi eliminado do histórico porque ele representa agora o quarto último grupo acessado e neste exemplo o histórico armazena apenas os três últimos.*

Agora, vamos inserir o objeto #75, com o grupo semântico D e tamanho 25 kb. Neste momento (Figura 4.4a), o cache *está com uma ocupação de 127 kb, isto é, há um espaço livre é de 23 kb, o que não é suficiente para inserir o objeto #75. Logo será necessário remover um ou mais objetos. O primeiro passo é atribuir os pesos dos três últimos acessos: o grupo B (objeto #63) tem agora o maior peso por ser o último. O objeto #35 foi eliminado na remoção anterior, mas o seu grupo semântica (B) continua valendo na hora da atribuição dos pesos. Então os pesos ficam distribuídos da seguinte forma: o grupo A tem peso 2 pelo antepenúltimo acesso e o grupo B obtém peso 4 pela soma dos registros de acessos referentes aos objetos #35 e #63, como mostra a Figura 4.4b. O algoritmo identifica o grupo A como sendo o grupo de menos interesse neste exato momento e remove o objeto #22 por ser este o de maior tamanho dentro do seu grupo. Com esta remoção a ocupação fica em 92 kb (127 kb espaço livre menos 35 kb do objeto #22 removido), permitindo assim a inserção do objeto #75. Para tal, o algoritmo cria o grupo semântico D e insere o objeto no* cache, *como mostra a Figura 4.4c.*

4.8 Implementação do algoritmo LSR/H

O algoritmo foi implementado para fins de validação, através da simulação de uma seqüência de acessos a objetos. A implementação foi feita na linguagem de programação Java. Existem duas classes principais, a saber:

- Tree*: A instância única desta classe representa o próprio* cache *e toda a hierarquia de grupos semânticos dos objetos nele contidos. É, portanto, responsável por gerenciar a ocupação do* cache *e manter o histórico de acessos. Esta classe disponibiliza o método público* insertInfo *para a inserção de um novo objeto no* cache. *A atualização*

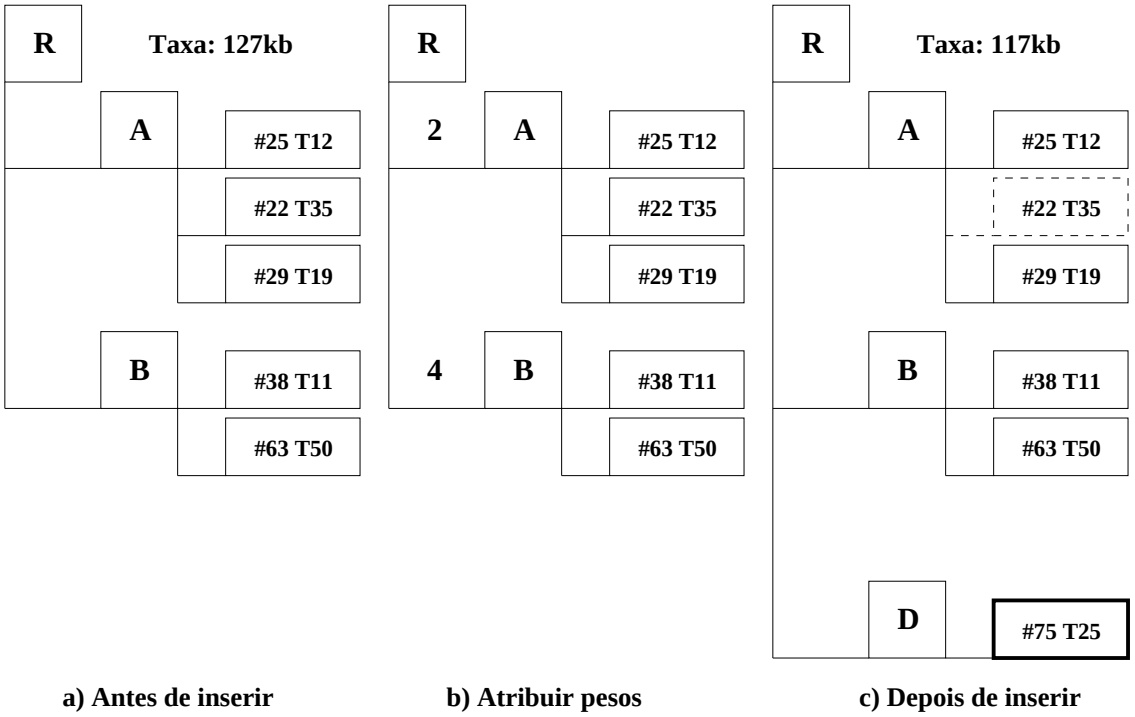

Figura 4.4 Exemplo de uma hierarquia com objetos no *cache*

dos pesos de cada grupo semântico de acordo com o histórico de acessos é feita pelo método includeTreeWeightLinear. *O método* clearTreeWeight *zera tais pesos logo após a remoção dos objetos no* cache.

- TreeNode*: Cada instância desta classe representa um grupo semântico da hierarquia mantida pelo* cache. *Existe uma instância especial desta classe que representa o grupo semântico raiz da hierarquia, o qual é denominado* root *no contexto da classe* Tree. *Cada instância desta classe possui um atributo que representa o peso do grupo semântico no momento de alguma remoção de objetos. Esta classe disponibiliza o método* excludeInfo *para remover objetos do* cache*, considerando os pesos dos grupos semânticos.*

4.8.1 Método insertInfo

Para cada inserção (detalhado na Figura4.5) é passado como parâmetro o objeto info*, no qual constam três informações: a identidade (URL) do objeto de informação, o seu grupo*

semântico e o seu tamanho em bytes. Primeiramente é verificado se o tamanho do objeto é maior do que o tamanho máximo do cache*; caso seja, o objeto será descartado da simulação e será gerada uma exceção. Na linha 09 é chamado o método* searchTreeNode*, que recebe o grupo semântico no qual se deseja inserir e retorna uma referência para o corresponde nó na árvore semântica (o nó é criado, caso não seja encontrado na árvore). A lista de objetos contidos em tal nó, é obtida na linha 10. O laço na linha 11 verifica se o objeto de inserção já está contido no* cache*. Caso esteja, o atributo* numHits *é incrementado. Caso contrário, é necessário inserir o objeto no* cache*. É verificado (linha 18) se existe espaço suficiente para seu tamanho. Caso tenha, o tamanho é adicionado no atributo* cacheCurrent *(linha 26) e o objeto é inserido na lista de seu grupo semântico (linha 27). Senão é necessário retirar um ou mais objetos do* cache*. Primeiramente, na linha 19, são incluídos os pesos na árvore semântica através do método* includeTreeWeightLinear *(detalhado na Figura 4.6). Outro método chamado (linha 22) é o* excludeInfo *(detalhado na Figura 4.8), que faz a remoção dos objetos. Este recebe como parâmetro a quantidade mínima em bytes a remover e retorna a quantidade removida em excesso. Depois de contabilizado o atributo* lenCacheCurrent *(linha 23), na linha 24 é chamado o método* clearTreeWeight *(detalhado na Figura 4.7), que zera os pesos da árvore semântica.*

Na linha 29, é registrado no histórico o último acesso a objeto (parâmetro para inserção no cache*). A lista de histórico tem um tamanho fixo previamente passado por parâmetro no início da simulação. Utiliza-se a técnica FIFO, que consiste em descartar o primeiro elemento (acesso mais antigo), caso seja necessário (linha 31).*

4.8.2 Método includeTreeWeightLinear

O método includeTreeWeightLinear*, mostrado na Figura 4.6, define os pesos de cada nó da hierarquia de grupos semânticos de acordo com o histórico corrente de acessos a objetos. Quanto mais antigo um acesso, menor é o seu peso: o acesso mais antigo tem peso 1, o seguinte tem peso 2, e assim consecutivamente, até o final do histórico. Sempre que um peso p é acrescentado ao peso de um certo nó n, p é também acrescentado ao nó pai de n, recursivamente, até se chegar à raiz da árvore.*

4.8.3 Método clearTreeWeight

O método clearTreeWeight*, mostrado na Figura 4.7, desfaz o que foi feito pelo método* includeTreeWeightLinear*, isto é, zera os pesos de todos os nós da árvore semântica que foram atualizados de acordo com o histórico de acessos. Assim, quando o método* clearTreeWeight *termina, os pesos de todos os nós da árvore semântica serão iguais a zero.*

4.8.4 Método excludeInfo

Este método excluí objetos do cache *sempre que for necessário criar espaço para a inserção de um novo objeto. Como parâmetro, é recebida a quantidade mínima em bytes que se deseja excluir. O método verifica se o nó é folha (linha 08), caso esta afirmação seja falsa, é chamado (linha 28) o método* searchTreeNodeLowerWeight *(detalhado na Figura 4.9) que identifica o nó de menor peso e retorna o nó folha. Agora, se o nó for folha, o método ordena (linha 10) os objetos pela ordem do tamanho em bytes e começa a remover do maior para o menor enquanto (linha 12) a variável* sizeNecessary *seja maior que zero e ainda haja objetos a remover neste nó. Se forem removidos todos os objetos do nó, este será removido (linha 20).*

Se não for satisfeita a condição do método (isto é, sizeNecessary *¡= 0), localizará outro nó de menor peso e começará a remover até a condição seja satisfeita (linha 22); caso contrário, se for removido mais do que era necessário a variável* sizeNecessary *será negativa e este será retornado (linha 24) para o método* insertInfo *para contabilizar a taxa de ocupação do* cache.

4.8.5 Método searchTreeNodeLowerWeight

Este método (detalhado na Figura 4.9) recebe como parâmetro um grupo semântico, o qual define uma sub-árvore da hierarquia de grupos semânticos (quando o parâmetro for o próprio grupo root*, a sub-árvore será toda a hierarquia de grupos semânticos). O objetivo é localizar o grupo semântico folha nessa sub-árvore que pertença ao ramo de menor peso, recursiva e descendentemente na sub-árvore. Então, caso o parâmetro seja uma folha na hierarquia de*

grupos semânticos, ele próprio é a resposta do método (linha 4). Senão, o grupo semântico descendente do parâmetro que tenha o menor peso é localizado (linha 5 a linha 9) e passado como parâmetro na chamada recursiva do método (linha 12), a fim de dar como resposta o resultado dessa chamada (linha 12).

```
[01] public void insertInfo(Info info) throws Exception {
[02]   long excess = 0;
[03]   TreeNode treeNode = null;
[04]   ArrayList lstInfoTemp = null;
[05]   boolean findInfo = false;
[06]   if (info.getSize() > this.lenMaxCache){
[07]     throw new Exception("objeto maior que o tamanho do cache");
[08]  }
[09]   treeNode = this.searchTreeNode(info.getSemantic());
[10]   lstInfoTemp = treeNode.getLstInfo();
[11]   for(int i = 0; i < lstInfoTemp.size(); i++){
[12]     Info infoTemp = (Info) lstInfoTemp.get(i);
[13]     if (infoTemp.getUrl().equals(info.getUrl())) {
[14]       findInfo = true; this.numHits++; break;
[15]     }
[16]   }
[17]   if (findInfo == false){
[18]     if (lenCacheCurrent + info.getSize() > lenMaxCache){
[19]       this.includeTreeWeightLinear();
[20]       long free = (this.lenMaxCache - this.lenCacheCurrent);
[21]       long overflow = (info.getSize() - free);
[22]       excess = root.excludeInfo(overflow);
[23]       this.lenCacheCurrent -= (overflow - excess);
[24]       this.clearTreeWeight();
[25]     }
[26]     this.lenCacheCurrent += info.getSize();
[27]    treeNode.getLstInfo().add(info);
[28]   }
[29]   this.getLstHistory().add(treeNode);
[30]   if (this.getLstHistory().size() == this.getSizeHistory()+1)
[31]   this.getLstHistory().remove(0);
[32] }
```

Figura 4.5 Método insertInfo

```
[01] private void includeTreeWeightLinear() {
[02]    TreeNode treeNode =  null;

[03]    TreeNode fatherTreeNode =  null;
[04]    ArrayList lstHistory = this.getLstHistory();
[05]    for (int i = 0; i < lstHistory.size(); i++) {
[06]       treeNode = (TreeNode) lstHistory.get(i);
[07]       treeNode.setWeight(treeNode.getWeight()+i+1);
[08]       fatherTreeNode = treeNode.getFatherTreeNode();
[09]       while (fatherTreeNode != null) {
[10]          fatherTreeNode.setWeight(fatherTreeNode.getWeight()+i+1);
[11]          fatherTreeNode = fatherTreeNode.getFatherTreeNode();
[12]       }
[13]   }
[14] }
```

Figura 4.6 Método includeTreeWeightLinear

```
[01] private void clearTreeWeight() {
[02]    TreeNode treeNode =  null;
[03]   TreeNode fatherTreeNode =  null;
[04]   ArrayList lstHistory = this.getLstHistory();
[05]    for (int i = 0; i < lstHistory.size(); i++) {
[06]       treeNode = (TreeNode) lstHistory.get(i);
[07]       treeNode.setWeight(0);
[08]       fatherTreeNode = treeNode.getFatherTreeNode();
[09]       while (fatherTreeNode != null) {
[10]          fatherTreeNode.setWeight(0);
[11]          fatherTreeNode = fatherTreeNode.getFatherTreeNode();
[12]       }
[13]    }
[14] }
```

Figura 4.7 Método clearTreeWeight

```
[01] public long excludeInfo(long sizeNecessary) {
[02]    TreeNode treeNodeCurrent = null;
[03]    TreeNode treeNodeFather = null;
[04]    ArrayList lstTreeNode = null;
[05]    TreeNode treeNodeLowerWeight = null;
[06]    ArrayList lstInfo = new ArrayList();
[07]    lstTreeNode = this.getLstTreeNode();
[08]    if (lstTreeNode.size() == 0) {
[09]       lstInfo = this.getLstInfo();
[10]       Collections.sort(lstInfo, new ComparatorSize());
[11]       int i = lstInfo.size()-1;
[12]       while (sizeNecessary > 0 && !lstInfo.isEmpty()) {
[13]          Info info = (Info) lstInfo.get(i);
[14]          sizeNecessary -= info.getSize();
[15]          lstInfo.remove(i);
[16]          i--;
[17]       }
[18]       treeNodeFather = this.getFatherTreeNode();
[19]       if ((lstInfo.isEmpty()) && (this.fatherTreeNode != null))
[20]          treeNodeFather.getLstTreeNode().remove(this);
[21]       if (sizeNecessary > 0)
[22]          return treeNodeFather.excludeInfo(sizeNecessary);
[23]       else
[24]          return sizeNecessary;
[25]    }
[26]    else {
[27]       treeNodeCurrent = this;
[28]       treeNodeLowerWeight = searchTreeNodeLowerWeight(treeNodeCurrent);
[29]       return treeNodeLowerWeight.excludeInfo(sizeNecessary);
[30]    }
[31] }
```

Figura 4.8 Método excludeInfo

```
[01] public TreeNode searchTreeNodeLowerWeight(TreeNode treeNodeCurrent) {
[02]    ArrayList lstTreeNode = treeNodeCurrent.getLstTreeNode();
[03]    if (lstTreeNode.size() == 0) // verifica se é folha
[04]       return treeNodeCurrent;
[05]    TreeNode treeNodeLowerWeight = (TreeNode) lstTreeNode.get(0);
[06]    for (int i = 1; i < lstTreeNode.size(); i++) {
[07]       TreeNode treeNode = (TreeNode) lstTreeNode.get(i);
[08]       if (treeNode.getWeight() < treeNodeLowerWeight.getWeight())
[09]          treeNodeLowerWeight = treeNode;
[10]    }
[12]    return searchTreeNodeLowerWeight(treeNodeLowerWeight);
[13] }
```

Figura 4.9 Método searchTreeNodeLowerWeight

Capítulo 5

Experimento e validação

O algoritmo LSR/H foi comparado com sua versão original - o LSR sem histórico - e com as outras políticas tradicionalmente utilizadas na Internet, os algoritmos: LRU, LFU e SIZE. Para esta comparação utilizou-se a métrica hit rate - HR*, isto é, o percentual de requisições que foram atendidas pelo* cache*. Assim, quando um objeto requisitado foi encontrado no* cache*, diz-se que houve um* hit*, caso contrário diz-se que ocorreu um* miss*. Nesta seção também será apresentado o processo de preparação dos dados para a simulação, bem como a obtenção dos objetos, a distribuição de acessos observada na Internet e a seqüência de acessos. Para a simulação dos acessos tal como tipicamente esses ocorrem no mundo real, foi aplicado o Método Monte Carlo, o qual necessita que os objetos estejam classificados em suas categorias de acessos e que cada objeto possua uma identificação única dentro da sua categoria.*

5.1 Obtenção dos objetos para a simulação

Na estrutura hierárquica DMOZ, estão catalogados - com a contribuição de 64.739 editores voluntários - mais de 4 milhões de sites*, distribuídos em 590.000 grupos semânticos, organizados hierarquicamente. A Figura 5.1 mostra parcialmente a hierarquia de grupos semânticos do DMOZ. A raiz da hierarquia está associada ao termo* Top*. Seus descendentes diretos são os grupos semânticos de primeiro nível. Entre esses, está, por exemplo, o grupo semântico do termo* Computers*, que contém 149.388 objetos, distribuídos pelos seus grupos semânticos descendentes. Agora, entre esses, está, por exemplo, o grupo semântico do termo* Internet*, que contém 44.483 objetos, e assim sucessivamente.*

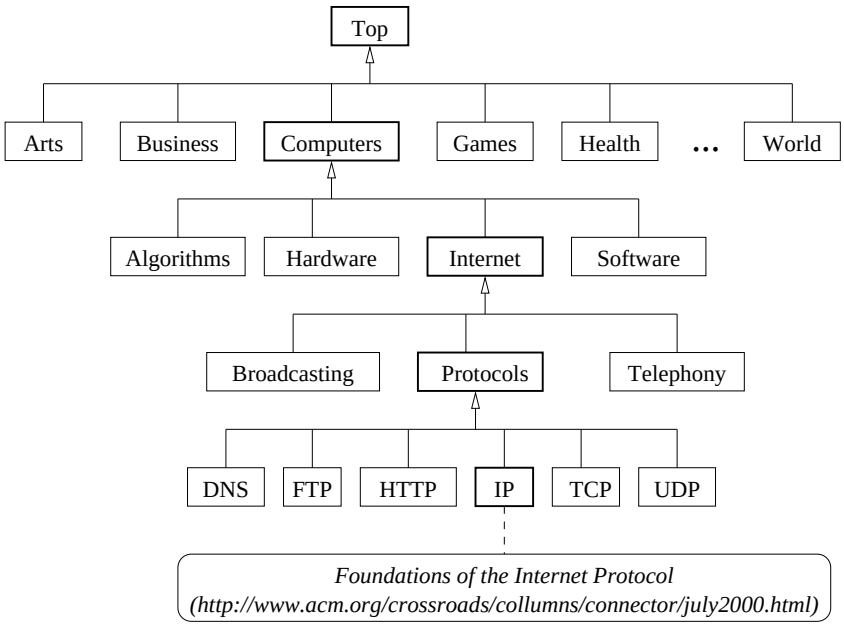

Figura 5.1 Hierarquia parcial do DMOZ

Semântica	*Objeto (URL)*	*Bytes*
Top.World.Brasil.Educação.Superior	www.pucpr.br	13417
Top.World.Brasil.Educação.Superior	www.ufpr.br	47522
Top.World.Brasil.Educação.Superior	www.ufmg.br	25417
Top.World.Brasil.Educação.Superior	www.ufrj.br	30007
Top.World.Brasil.Educação.Superior	www.unicamp.br	39482
Top.World.Brasil.Educação.Superior	www.usp.br	42300

Tabela 5.1 Exemplos de objetos do DMOZ

Esta estrutura foi utilizada como base para a obtenção dos objetos. Como amostra foram utilizados 161.440 objetos desta organização. Para cada objeto foram extraídas a sua semântica (caminho de grupos semânticos, a partir do grupo Top*) e a sua* url*. Para a recuperação do tamanho do objeto foi desenvolvida uma aplicação que percorre todos os objetos e retorna o tamanho em bytes encontrado em suas páginas, como mostra a Tabela 5.1.*

5.2 Categorias de acessos

O Histograma da Figura 5.2 refere-se ao log *de um servidor* proxy Squid - *um dos gerenciadores de* cache *mais utilizados - do departamento de informática de uma instituição de ensino, com aproximadamente 500 usuários, tomando uma amostragem de 9.657.359 acessos, 1.615.363 objetos, pertencentes a 1.591 categorias de acessos, em 3 dias. A categoria 1, ou seja, a categoria de objetos que possui apenas um único acesso, contém 1.066.303 objetos, o que representa apenas 11.04% da amostragem total de acessos. A maioria absoluta (88.96%) dos acessos, refere-se a objetos acessados mais de uma vez, evidenciando a importância de uma política de substituição de objetos em* cache *para a economia de banda e tráfego redundante na rede.*

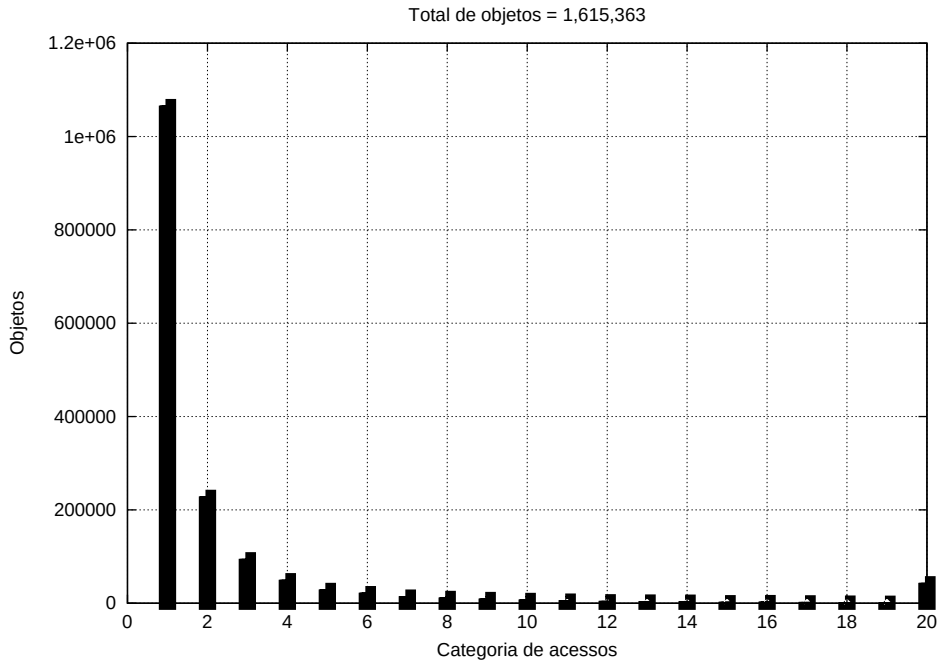

Figura 5.2 Histograma Distribuição de acessos do proxy do PPGIA

5.3 Distribuição dos objetos em categorias de acessos

Com a distribuição em categorias de acessos identificadas a partir da análise do log *do* proxy squid*, conforme descrito na seção anterior, foram geradas 795 categorias de acesso para os objetos da amostra, como mostra a Figura 5.3. (796 categorias de acesso foram desconsideradas porque apresentaram quantidade de objetos inferior a um na projeção)*

A distribuição dos objetos da amostra em categorias de acesso deve satisfazer aos seguintes critérios:

(1) *Obedecer às mesmas proporções das categorias de acesso constatadas na prática. As categorias de acesso identificadas a partir do* log *do* proxy Squid *de um certo grupo de usuários e por um certo tempo serviu como base para definir a proporção de cada categoria de acesso para os objetos da amostra. Assim, foi feita uma projeção na qual foram criadas 795 categorias de acesso para os 161.440 objetos da amostra. A Figura 5.3 mostra a quantidade de objetos em cada categoria de acesso após a projeção.*

(2) *Obedecer ao princípio de grupo semântico de maior interesse. Foram escolhidos dois grupos semânticos (*Business *e* Reference*) para representarem os grupos semânticos de maior interesse, dentre os dez existentes no primeiro nível da hierarquia correspondente à amostra.*

5.4 Seqüência de acessos

Nesta fase foi feita uma aplicação que permitisse a criação dos acessos aleatórios aplicando o Método Monte Carlo, de acordo com as 795 categorias de acesso geradas. Para cada categoria determina-se a quantidade de objetos, a quantidade de acessos (multiplicando-se a quantidade de objetos pelo próprio número da categoria), a probabilidade (dividindo-se a quantidade de acessos pelo total de acessos), o limite inferior e o limite superior (de acordo com a probabilidade encontrada), conforme mostra a Tabela 5.2 . Assim, aplicam-se os seguintes passos para gerar cada acesso a objeto:

(1) *Escolha de um número real aleatório entre zero e um; verifica-se onde este número se encontra de acordo com os intervalos definidos pelas colunas* Limite Inferior *e* Limite Superior *para localizar a categoria sorteada*

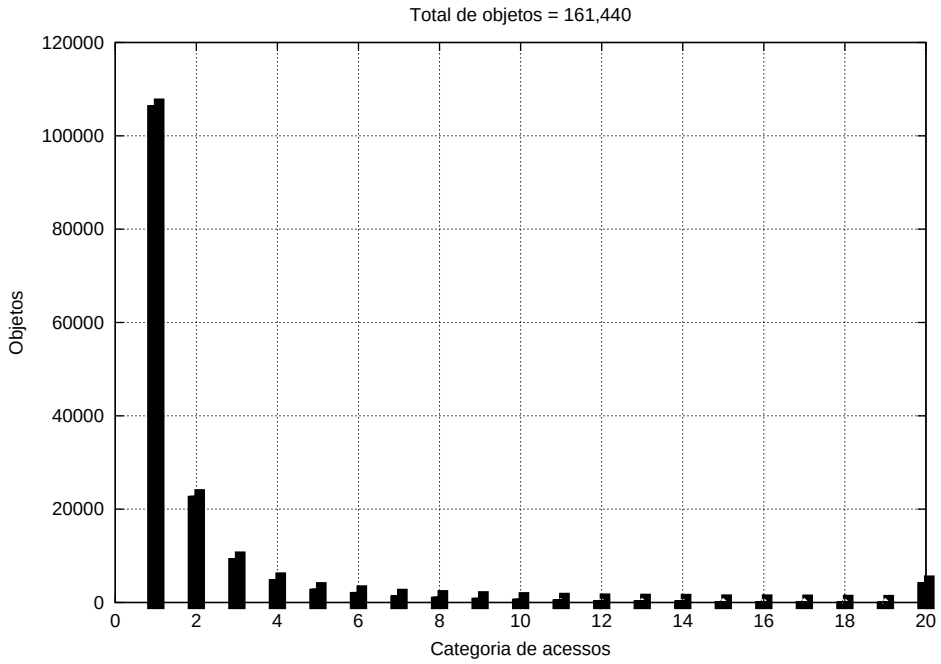

Figura 5.3 Histograma Distribuição de acessos da amostra do DMOZ

(2) *Escolha de um segundo número inteiro aleatório entre 1 e o valor contido na coluna* Quantidade de Objetos por Categoria*; este segundo número aleatório indicará o objeto escolhido para o acesso.*

5.5 Análise de desempenho

Para o experimento foram criadas três seqüências de acessos, de 500.000, 1.000.000 e 1.500.000 de acessos aos objetos da amostra retirada do DMOZ, a fim de poder comparar os algoritmos e observar se a simulação encontra-se estável. Para cada seqüência de acesso, foram coletados dados de desempenho dos algoritmos SIZE, LRU, LFU e LSR/H com históricos de tamanho 1, 5 e 100, totalizando seis estratégias de substituição de objetos. Foram utilizados seis diferentes tamanhos de cache*, a saber de 4, 8, 16, 32, 64 e 128 megabytes de memória. Portanto, foram realizadas 36 simulações para cada uma das três seqüências de acessos, totalizando 108 simulações. Os gráficos de dessa seção mostram as curvas de desempenho em* hit rate - HR *e* byte hit rate - BHR *para essas simulações.*

Categoria de Acesso	*Objetos por Categoria*	*Acessos por Categoria*	*Probabilidade*	*Intervalo*	
				Inferior	*Superior*
c	q_c	$c * q_c$	$P_c = \frac{q_c}{\sum_c q_c}$	$\sum_{i=1}^{c-1} P_i$	$\sum_{i=1}^{c} P_i$
1	106566	106566	13.05	0.0000	0.1305
2	22863	45726	5.60	0.1305	0.1865
3	9510	28530	3.50	0.1865	0.2214
4	5015	20060	2.46	0.2214	0.2460
5	2932	14660	1.80	0.2460	0.2640
Mais	14554	601067	73.59	0.2640	1.0000
Total	161440	816609	100		
	$\sum_c q_c$	$\sum_c c * q_c$	$\sum_c P_c$		

Tabela 5.2 Tabela probabilidade

Observando-se as curvas de desempenho dos algoritmos, chega-se às seguintes conclusões:

- *O desempenho de cada algoritmo é aproximadamente o mesmo para os três comprimentos de seqüência de acessos experimentadas, indicando que há estabilidade nos resultados obtidos com a simulação*
- *Todos os algoritmos apresentam crescimento de desempenho à medida que se aumenta o tamanho do* cache. *Naturalmente, quando o tamanho do* cache *tende ao infinito, todos os objetos acessados serão armazenados no* cache *e, conseqüentemente, não haverá substituição de objetos. Portanto, a partir de um certo ponto, todos os algoritmos terão exatamente o mesmo desempenho e se estabilizarão num certo patamar.*
- *A partir de um certo tamanho de* cache *(aproximadamente 32 mb no experimento realizado), o algoritmo de melhor desempenho é o LSR/H.*
- *Para um tamanho qualquer de* cache, *quanto maior o histórico, melhor é o desempenho do algoritmo LSR/H.*
- *Quanto maior o histórico no algoritmo LSR/H, menor é a taxa de crescimento de desempenho do algoritmo à medida que se aumenta o tamanho do* cache. *Portanto, para um tamanho qualquer de* cache, *o aumento no desempenho não é diretamente proporcional ao aumento no histórico. Isso indica que, para cada tamanho de* cache, *há um comprimento para o histórico a partir do qual o ganho em desempenho é pouco significativo*

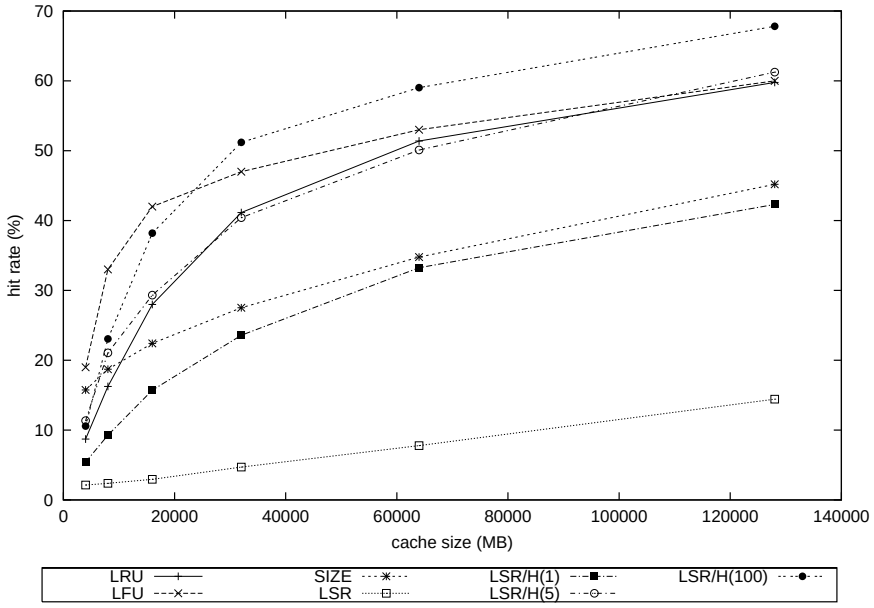

Figura 5.4 Probabilidade de acerto para 0.5 milhão de acessos - HR

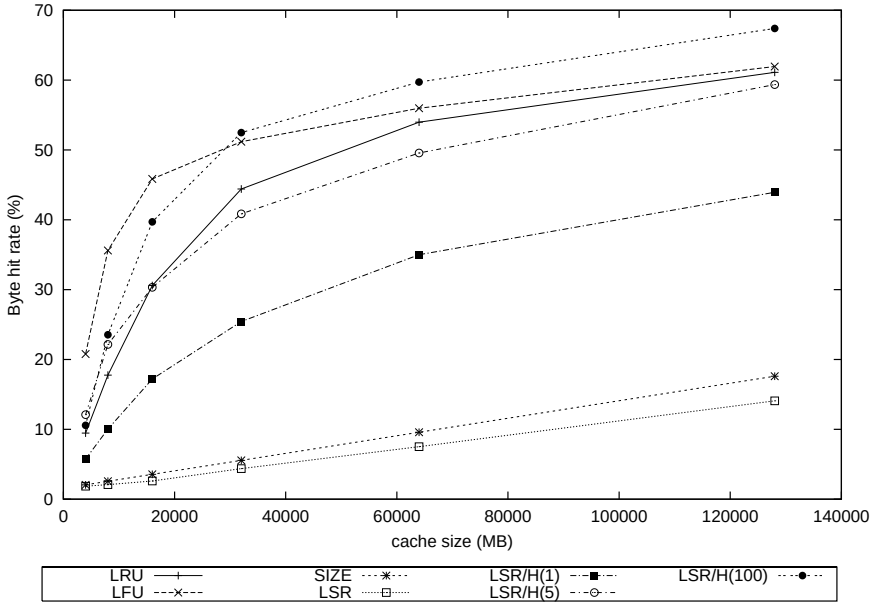

Figura 5.5 Probabilidade de acerto para 0.5 milhão de acessos - BHR

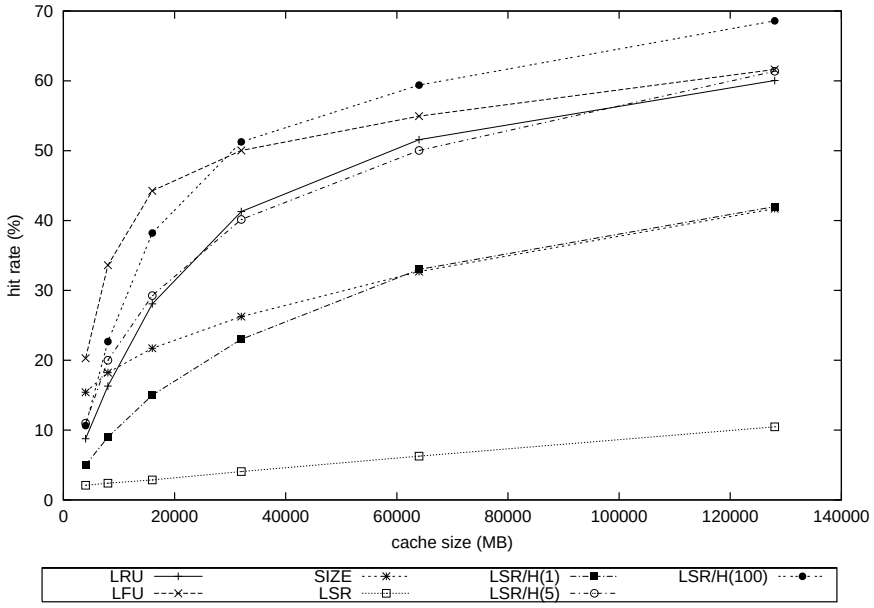

Figura 5.6 Probabilidade de acerto para 1.0 milhão de acessos - HR

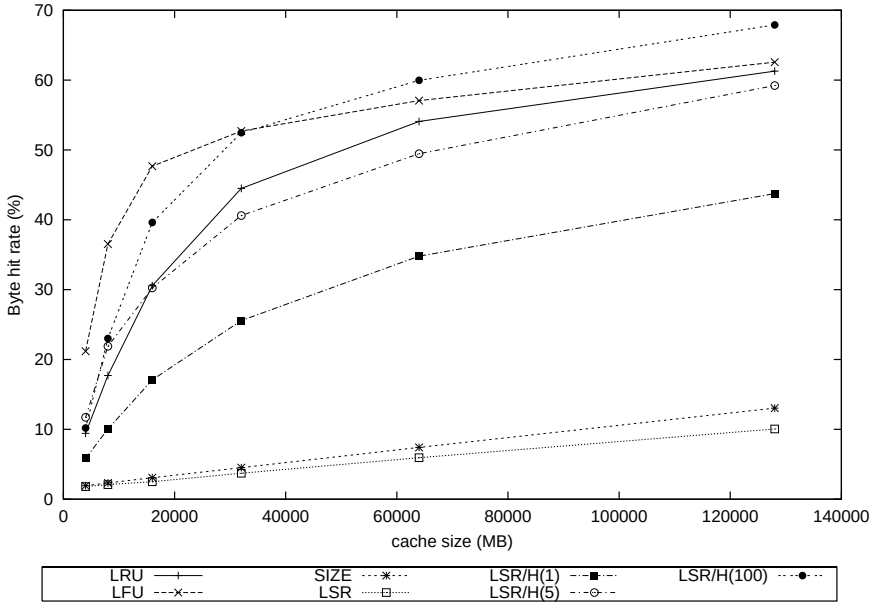

Figura 5.7 Probabilidade de acerto para 1.0 milhão de acessos - BHR

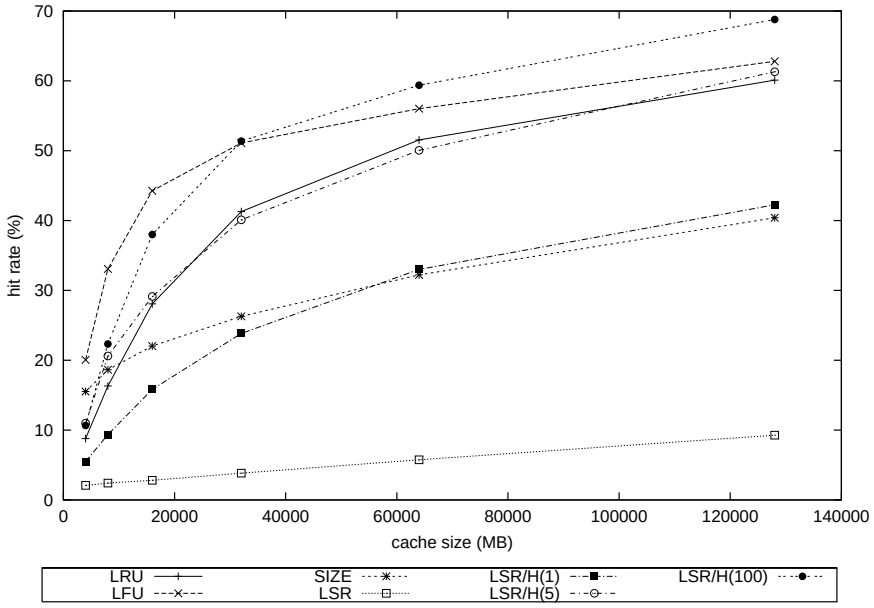

Figura 5.8 Probabilidade de acerto para 1.5 milhão de acessos - HR

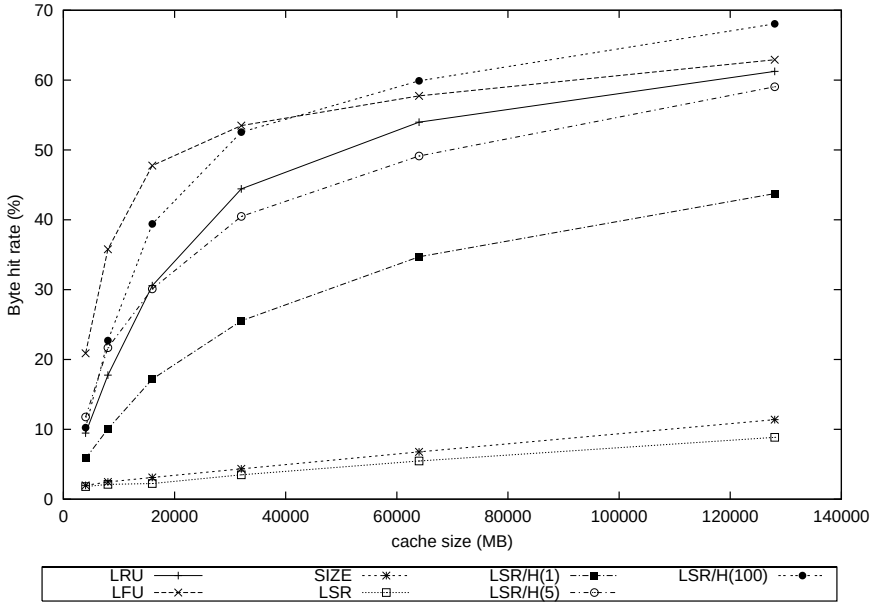

Figura 5.9 Probabilidade de acerto para 1.5 milhão de acessos - BHR

Os resultados vistos nos gráficos de desempenho demonstram a importância do histórico de acessos como heurística para a remoção de objetos do cache. *Através do aumento significativo da taxa de acerto em todos os gráficos de desempenho, também demonstram que foi possível atingir bom resultado simultaneamente em* hit rate - HR e byte hit rate - BHR. *Como foi mostrado em comparação com os outros algoritmo tradicionais utilizados na Internet, o LSR/H manteve mais objetos no* cache*, fez menos inserções e remoções e reteve os objetos em um tempo maior no* cache. *Como conseqüência obteve uma maior taxa de acerto. A desvantagem do algoritmo é a parametrização empírica. O algoritmo LSR/H foi avaliado com tamanhos de históricos a partir dos experimentos. A pergunta natural é qual seria o tamanho do histórico ideal para o LSR/H? Está é uma questão para os trabalhos futuros.*

Capítulo 6

Conclusões e trabalhos futuros

Esta dissertação propôs uma combinação inovadora para substituição de objetos em cache*, baseado na semântica da informação e no histórico de acessos. Além do algoritmo, foi projetada e implementada uma estrutura para a coleta e preparação dos dados a fim de poder validar e comparar com os outros algoritmos de substituição de objetos existentes, e também com a sua versão original, o LSR sem histórico. Esta combinação obteve um desempenho superior às estratégias utilizadas hoje na Internet, motivando assim a continuação do seu estudo e análise da viabilidade do seu uso num ambiente real.*

O estudo do LSR/H pode ser continuado em diversas frentes, entre elas:

- *Utilizar funções não lineares para a atribuição de pesos aos grupos semânticos e comparar o seu desempenho com os resultados obtidos até o momento. Por exemplo, pode-se utilizar uma função exponencial, dando ainda maior peso para os acessos mais recentes.*
- *Calcular a complexidade do algoritmo LSR/H em função do comprimento do histórico, e dimensionar o impacto da sua execução em diferentes plataformas.*
- *Descobrir um método (analítico ou experimental) para o cálculo do tamanho ideal para o histórico de acessos, e para o tamanho do* cache*, de acordo com um o padrão de acessos em uma certa instalação. Basicamente, deverá ser verificado se o ganho de desempenho em hit rate compensa o impacto da execução do algoritmo.*
- *Instalar o LSR/H em um gerenciador de* cache *real (por exemplo, em um proxy Squid) e fazer análise do seu desempenho. Isso vai exigir que, em paralelo, seja utilizado um servidor de semântica - como definido em [Shmidt, 2002, Calsavara and Schmidt, 2004] - a fim de se obter a semântica de cada objeto acessado.*
- *Criar uma classificação dinâmica dos grupos semânticos utilizando, por exemplo,* Platform for Internet Content Selection - PICS *criado pelo* World Wide Web Consortium

- W3C *[W3C-Consortium, 1997]. O sistema de análise e classificação lê os rótulos onde estão inseridas, pelos produtores de conteúdos, as características das páginas solicitadas. Esta plataforma possui uma parte destinada ao produtor de conteúdo Web, que deseja ou necessita que seu conteúdo seja visto por um público específico, e uma parte destinada aos produtores de software, que implementam sistemas de classificação baseados em* PICS.

- *Verificar a possibilidade de particionamento do cache de acordo com as classes de objetos (hipertexto, binário, imagem, etc) e tamanho de objeto. Trabalhos anteriores, como discutido em [Pinheiro, 2001] e [Murta et al., 1998], mostraram que há um ganho significativo em* hit rate *quando o cache é particionado. Entretanto, atualmente, desconhecemos uma maneira de combinar tal particionamento com uma hierarquia de grupos semânticos.*

- *Informação semântica sobre objetos na Internet pode ser útil além de estratégia de substituição de objetos em cache. Como sugerido em [Brunie et al., 2002], pessoas navegando por objetos relacionados a um mesmo assunto definem uma* comunidade virtual*, então podemos investigar o uso do LSR/H das seguintes maneiras a fim de beneficiar tal comunidade:*

  - *Implementação de heurísticas de pre-fetching, isto é, carregar objetos antes de serem realmente requisitados por usuários.*
  - *Otimização de procedimentos de colaboração entre proxies. Quando um assunto torna-se quente em algum proxy, proxies vizinhos a este podem fazer prefetch de objetos relacionados. De forma similar, proxies podem compartilhar informação sobre threads correntes de interesse a fim de otimizar estratégias de gerenciamento de caches.*
  - *Monitoramento da evolução do interesse de uma comunidade virtual.*
  - *Difusão de assuntos quentes atuais, tal que os usuários possam ser notificados sobre o que outros usuários estão acessando.*

Portanto, o trabalho desenvolvido contribuiu para evidenciar e mensurar os benefícios do emprego de informação semântica combinada com histórico de acessos para a remoção de objetos de cache*, atingindo assim os seus objetivos, além de identificar novos trabalhos de pesquisa para o aperfeiçoamento do algoritmo proposto. Os conceitos desenvolvidos neste*

trabalho podem ser explorados em domínios específicos de aplicações, tais como bibliotecas digitais e comércio eletrônico, nos quais normalmente há uma taxonomia bem definida para os objetos de informação. Em última análise, as idéias defendidas neste trabalho podem ser a base para um novo modelo de organização de informação na Internet e para o surgimento de novas aplicações para a mesma.

Referências Bibliográficas

[Abrams et al., 1996] Abrams, M., Standridge, C. R., Abdulla, G., Fox, E. A., and Williams, S. (1996). Removal policies in network caches for world-wide web documents. In SIGCOMM, *pages 293–305.*

[Aggarwal et al., 1999] Aggarwal, C. C., Wolf, J. L., and Yu, P. S. (1999). Caching on the world wide web. IEEE Trans. Knowl. Data Eng., *11(1):95–107.*

[Aggarwal and Yu, 1997] Aggarwal, C. C. and Yu, P. S. (1997). On disk caching of web objects in proxy servers. In Golshani, F. and Makki, K., editors, CIKM, *pages 238–245. ACM.*

[Arlitt et al., 2000] Arlitt, M. F., Friedrich, R., and Jin, T. (2000). Performance evaluation of web proxy cache replacement policies. Perform. Eval., *39(1-4):149–164.*

[Arlitt and Williamson, 1996] Arlitt, M. F. and Williamson, C. L. (1996). Web server workload characterization: The search for invariants. In SIGMETRICS, *pages 126–137.*

[Benevenuto et al., 2005] Benevenuto, F., Duarte, F., and Almeida, V. (2005). Entendendo os efeitos da localidade de referência em hierarquias de caches na web. In Anais do 23. Simpósio Brasileiro de Redes de Computadores, *pages 150–163, Fortaleza-CE.*

[Brandão and Anido, 2001] Brandão, R. F. and Anido, R. d. O. (2001). Um simulador parapelo para sistemas de caches para web. In Proceedings of the 19th Brazilian Simposium on Computer Networks, *pages 480–495.*

[Brunie et al., 2002] Brunie, L., Pierson, J.-M., and Coquil, D. (2002). Semantic collaborative web caching. In Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE'02), *pages 30–42.*

[Calsavara and dos Santos, 2002] Calsavara, A. and dos Santos, R. G. (2002). Um algoritmo de substituição de objetos em cache na internet baseado em semântica. In Anais do 20. Simpósio Brasileiro de Redes de Computadores, *pages 135–150, Búzios-RJ.*

[Calsavara et al., 2003] Calsavara, A., dos Santos, R. G., and Jamhour, E. (2003). The least semantically related cache replacement algorithm. In Proceedings of the IFIP/ACM Latin American Network Conference, *pages 21–34. Nova York: ACM.*

[Calsavara and Schmidt, 2004] Calsavara, A. and Schmidt, G. (2004). Semantic search engines. In Ramos, F. F., Unger, H., and Larios, V., editors, Advanced Distributed Systems: Third International School and Symposium, ISSADS 2004, Guadalajara, Mexico, January 24-30, 2004. Revised Selected Papers, *volume 3061 of* Lecture Notes in Computer Science, *pages 145–157. Springer.*

[Calsavara and Schuck, 2005] Calsavara, A. and Schuck, M. R. (2005). Internet object caching based on semantics and access history. In Anais do 23. Simpósio Brasileiro de Redes de Computadores, *pages 437–449, Fortaleza-CE.*

[Cao and Irani, 1997] Cao, P. and Irani, S. (1997). Cost-aware www proxy caching algorithms. In USENIX Symposium on Internet Technologies and Systems.

[Claudino, 2002] Claudino, R. A. T. (2002). Adaptação dinâmica de páginas web em ambiente de computação ubíqua. In Workshop Cooperação UFSCar - FSA em Ciência da Computação, *São Carlos,Brasil.*

[Dar et al., 1996] Dar, S., Franklin, M. J., Jónsson, B. T., Srivastava, D., and Tan, M. (1996). Semantic data caching and replacement. In Proc. VLDB Conf., *pages 330–341, Bombay, India.*

[de Oliveira, 2004] de Oliveira, J. A. (2004). Uso de caches na web - influência das políticas de substituição de objetos. Master's thesis, Instituto de Ciências Matemáticas e de Computação de São Carlos - USP, São Carlos, Brazil.

[Dilley and Arlitt, 1999] Dilley, J. and Arlitt, M. F. (1999). Improving proxy cache performance: Analysis of three replacement policies. IEEE Internet Computing, *3(6):44–50.*

[DMOZ, 2005] DMOZ (2005). Site dmoz. URL: http://www.dmoz.org, consultado em julho 2005.

[dos Santos, 2001] dos Santos, R. G. (2001). Internet object cache replacement based on semantics. Master's thesis, Pontificia Universidade Catolica do Parana, Curitiba, Brazil.

[Elson and Cerpa, 2003] Elson, J. and Cerpa, A. (2003). The internet content adaptation protocol. In Request for Comments 3507.

[Fonseca et al., 2003] Fonseca, R. C., Almeida, V., Crovella, M., and Abrahao, B. (2003). On the intrinsic locality properties of web reference streams. In INFOCOM.

[Forte et al., 2005] Forte, M., de Souza, W. L., and do Prado, A. F. (2005). Um servidor para a classificação e filtragem de conteúdo na internet. In Anais do 23. Simpósio Brasileiro de Redes de Computadores, *pages 164–183, Fortaleza-CE.*

[Markatos, 1996] Markatos, E. P. (1996). Main memory caching of web documents. Computer Networks, *28(7-11):893–905.*

[Murta et al., 1998] Murta, C. D., Almeida, V. A. F., and Meira Jr., W. (1998). Analyzing performance of partitioned caches for the WWW. In Proceedings of the 3rd International WWW Caching Workshop.

[Pinheiro, 2001] 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, São Carlos, Brazil.

[Podlipnig and Böszörményi, 2003] Podlipnig, S. and Böszörményi, L. (2003). A survey of web cache replacement strategies. ACM Comput. Surv., *35(4):374–398.*

[Rizzo and Vicisano, 2000] Rizzo, L. and Vicisano, L. (2000). Replacement policies for a proxy cache. IEEE/ACM Transactions on Networking, *8(2):158–170.*

[Scheuermann et al., 1997] Scheuermann, P., Shim, J., and Vingralek, R. (1997). A case for delay-conscious caching of web documents. Computer Networks, *29(8-13):997–1005.*

[Shmidt, 2002] Shmidt, G. (2002). Semantics server for internet resources: an approach based on rdf. Master's thesis, Pontificia Universidade Catolica do Parana, Curitiba, Brazil.

[W3C-Consortium, 1997] W3C-Consortium (1997). Platform for internet content selection (pics). URL: http://www.w3.org/PICS/, consultado em julho 2005.

[W3C-Consortium, 2001] W3C-Consortium (2001). Semantic web. URL: http://www.w3.org/2001/sw/, consultado em julho 2005.

[Wooster and Abrams, 1997] Wooster, R. P. and Abrams, M. (1997). Proxy caching that estimates page load delays. Computer Networks, *29(8-13):977–986.*

[YAHOO!, 2005] YAHOO! (2005). Site yahoo! URL: http://www.yahoo.com, consultado em julho 2005.