Um algoritmo de substituição de objetos em cache na Internet baseado em semântica

Alcides Calsavara, Rogério Guaraci dos Santos
Pontifícia Universidade Católica do Paraná – PUCPR
Programa de Pós-Graduação em Informática Aplicada – PPGIA
Rua Imaculada Conceição, 1155, Prado Velho, Curitiba, PR, CEP 80215-901
{alcides,rogerio}@ppgia.pucpr.br

Resumo

Este artigo apresenta o algoritmo de substituição de objetos em cache na Internet denominado *Least Semantically Related* (LSR). Diferentemente dos 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 de 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. Um algoritmo detalhado e uma estrutura de dados para o LSR são completamente projetados e implementados para fins de validação e comparação com outras estratégias de substituição de objetos em cache; são também implementadas as estratégias SIZE, LFU e LRU. Além disso, é projetado um modelo para a realização dos experimentos, incluindo toda a coleta e preparação de dados para análise. Os resultados experimentais mostram que a estratégia proposta oferece, na maioria das situações, uma medida de eficiência – probabilidade de acerto – superior às outras estratégias. As principais contribuições desse artigo são a proposta da nova estratégia de substituição de objetos em cache, a sua implementação e validação, assim como o próprio modelo e correspondentes ferramentas de obtenção de dados para a validação.

Abstract

This article presents the Web cache replacement algorithm named *Least Semantically Related* (LSR). Differently from well-known replacement algorithms, which are based on physical properties of objects, LSR is based on the semantics of the information contained in objects: LSR tends to favor objects to stay in cache when they are close with respect to their semantics, by removing from cache objects that tend to be of less interest to clients. A detailed algorithm and a data structure for LSR are completely designed and implemented for the purpose of validation and comparison with other replacement algorithms, namely SIZE, LFU and LRU, also implemented. Besides, a framework for the experimental work is designed and verified, including the data preparation process. The initial experimental results show that the proposed strategy offers, in most cases, efficiency rate – the hit rate – better than other strategies. The main contribution of this paper are the proposal of a new strategy to replace objects in cache, its implementation and validation, and a framework and corresponding tools to obtain experimental data.

Palavras-chaves: Internet services, cache replacement algorithm, Web caching, Semantic Web

1 Introdução

Com o crescimento da utilização da Internet e o surgimento de novas aplicações, cresce a cada dia a quantidade de dados transmitidos, tornando a infra-estrutura de comunicação existente cada vez mais sujeita à saturação. Grande parte desse tráfego é, muitas vezes, formado pela passagem de diversas cópias dos mesmos objetos de informação. Nesse contexto, a utilização de cache desempenha um papel fundamental por diversas razões. Primeiramente, porque é um meio de habilitar a Internet a disponibilizar seus serviços dentro de níveis aceitáveis de tempo de resposta, pois tende a reduzir a média de latência no acesso aos objetos de informação. Em segundo lugar, permite redução no tráfego da rede em três níveis: (i) entre um cliente (*browser*) e um *proxy*, (ii) entre um cliente e um servidor, (iii) entre *proxies*. Em terceiro lugar, oferece redução no número de pedidos feitos aos servidores, diminuindo a chance de sobrecarregar os mesmos [Williams *et al.*, 1996]. De acordo com [Murta, 1998], a utilização de cache é fundamental para a escalabilidade da Internet, especialmente com o crescimento do tamanho médio dos arquivos transmitidos, principalmente devido à intensificação do uso de recusos de áudio e vídeo.

Devido à natural limitação no tamanho de um cache (tanto em um *proxy* como em um cliente), quando este estiver com sua ocupação completa e for preciso inserir um novo objeto, um objeto (ou um conjunto de objetos) do cache deverá ser eleito para ser removido, ou seja, deverá ser feita a substituição de alguns objetos do cache por novos objetos. De forma abreviada, o problema consiste em responder às seguintes questões: É necessário remover objetos do cache? Se necessário, quais são os objetos a ser selecionados para remoção? Para tanto, existem os chamados algoritmos de substituição.

Observa-se na literatura que os algoritmos de substituição atualmente utilizados na Internet e mesmo os que estão em estudo nos meios acadêmicos baseiam-se invariavelmente em características "físicas" dos objetos, tais como tamanho, tempo de validade *(Time-To-Live -- TTL)*, freqüência de acesso, custo de acesso, etc. Para reduzir a latência de acesso, é desejável que os caches armazenem cópias de objetos relacionados a assuntos mais pessoais dos clientes [Wessels, 1995]. Não se encontram, entretanto, algoritmos que explorem a semântica da informação contida nesses objetos.

Este artigo apresenta um novo algoritmo de substituição de objetos de informação em cache, baseado na *semântica da informação*, denominado *Least Semantically Related*, ou simplesmente LSR, cujo princípio é substituir os objetos que sejam menos relacionados semanticamente com o novo objeto que entrar no cache. O algoritmo assume que os clientes tenham a tendência de procurar por novas informações que estejam relacionadas com a informação atualmente em mãos. Dessa forma, o algoritmo LSR tende a favorecer a permanência no cache de objetos que possuam 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.

As principais contribuições deste artigo são a originalidade do *algoritmo de substituição baseado em semântica*, a sua definição formal e a verificação da sua viabilidade por meio de análise estatística, comparando o seu desempenho com os algoritmos atualmente conhecidos para a substituição de objetos em cache. Verificamos, que o algoritmo proposto é adequado para a Internet – se não de forma genérica, mas pelo menos em algumas situações.

O restante deste artigo descreve resumidamente o experimento realizado em [Santos, 2001] e está organizado como segue. Na Seção 2, apresentam-se trabalhos relacionados, descrevendo-se resumidamente diversos algoritmos para substituição de objetos em cache em uso na Internet. Na Seção 3, o algoritmo LSR é apresentado e exemplificado. Na Seção 4, descreve-se a implementação do algoritmo, usando um esquema baseado em árvore para a classifi-

cação de informações. A Seção 5 discute sobre o experimento e a validação do esquema de simulação do LSR. A Seção 6 apresenta conclusões e propõe trabalhos futuros.

2 Trabalhos relacionados

Os algoritmos de substituição de *cache* têm um papel fundamental no projeto de qualquer componente de armazenamento. Esses algoritmos, por exemplo, vêm sendo intensivamente estudados no contexto da operação de sistemas de controle de memória virtual [O'Neil *et al*., 1993]. Apresentam-se resumidamente, a seguir, os mais relevantes algoritmos de substituição de objetos em *cache*, baseados em características físicas dos objetos, tanto por estarem atualmente em uso na Internet, como por estarem em estágio avançado de pesquisa.

SIZE: Este algoritmo remove primeiramente do *cache*, quando da chegada de um novo objeto, o maior objeto em tamanho, de forma que possa criar espaço suficiente para o armazenamento do novo objeto, sendo que, ao remover o maior objeto em tamanho e ainda não tenha sido criado espaço suficiente para entrada do novo objeto, deve-se retirar o próximo objeto maior, e assim sucessivamente até que o espaço seja suficiente. Se dois ou mais objetos forem escolhidos por terem, coincidentemente, o mesmo tamanho, utiliza-se a política LFU para remover dentre esses o menos freqüentemente usado. [Aggarwal *et al*., 1999].

***LEAST-RECENTLY-USED* (LRU):** Esta política privilegia as referências mais recentes, protegendo-as da substituição. Assim, ela explora o princípio da localidade temporal que diz que documentos recentemente acessados têm maior probabilidade de serem requisitados em futuro próximo. A localidade temporal foi observada em seqüência de acesso a código e dados de programas. Neste ambiente de cache tradicional, a reposição de blocos fica a cargo do algoritmo LRU por ser baseado no princípio da localidade. [Aggarwal *et al*.1999].

LEAST-FREQUENTLY-USED (LFU): Esta política retira os objetos utilizados com menor freqüência, ignora a recenticidade dos acessos e pode, portanto, retirar objetos recém-acessados que serão muito requisitados no futuro. Por outro lado o algoritmo preserva objetos mais referenciados, o que é uma vantagem. Porém, pode preservar eternamente objetos muito referenciados no passado e que não serão mais referenciados no futuro. [Aggarwal *et al*.1999].

***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 Internet 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.[Scheuermann *et al*.,1997].

***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 caches de rede. O cache acumula, em tempo de execução, estatísticas de cada objeto requisitado, atualizadas a cada acesso. A principal desvantagem do LRV, além de seu custo proibitivo para implementação, é a sua grande parametrização com base em apenas duas cargas, o que deixa dúvidas sobre o bom desempenho deste algoritmo com cargas diferentes [Rizzo e Vicisano, 1998].

LRUMIN: Este algoritmo é uma variação do LRU, a 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. [Williams *et al*., 1996].

GREEDYDUAL-SIZE (GD-SIZE): O algoritmo proposto 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. [Cao e Irani, 1997].

LOWEST-LATENCY-FIRST **-** Este algoritmo substitui primeiro o objeto que teve menor latência no acesso via rede. No entanto, o mesmo trabalho mostra que não é um bom critério. Em sistemas com grande variabilidade, onde picos numa medida impactam negativamente o desempenho, considerar apenas a média para representar os dados pode levar a conclusões errôneas. Portanto, a utilização destes tempos deve ser feita por meio de valores médios acompanhados por medidas de dispersão dos dados, o que pode dificultar a sua utilização. [Wooster e Abrams, 1997].

HYBRID - Este algoritmo híbrido 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, *bandwidth* da conexão, a freqüência e o tamanho do objeto. Objetos com menor valor são descartados. O algoritmo apresentou bom desempenho em HR (*hit rate*) e tempo de resposta frente a LRU, LFU e SIZE. Suas desvantagens são a necessidade de armazenar mais informações para cada objeto e a necessidade de ajustes cuidadosos das constantes utilizadas na função. [Wooster e Abrams, 1997]. (Na mesma linha está o MIX [Niclausse *et al.*, 1998] que utiliza os parâmetros latência, número de referências, tamanho e tempo decorrido desde a última referência.)

FIRST-IN, FIRST-OUT(FIFO) **-** Este algoritmo substitui o objeto que entrar primeiro no *cache* [Silberschatz e Galvin, 1994].

CACHE PARTICIONADO **-** Esta abordagem divide o espaço do *cache* em partições. Cada partição é dedicada ao armazenamento dos objetos cujos tamanhos pertencem a um intervalo. Cada intervalo define uma classe de objetos. As classes são em número igual ao número de partições e cobrem todos os tamanhos possíveis de objetos, sem sobreposição. Substituição dos objetos são feitas apenas entre objetos de uma mesma classe. Este modelo não permite nem retirada de um objeto pequeno para a inserção de um objeto grande, nem retirada de um objeto grande para a inserção de um objeto pequeno, pois não é possível fazer substituição entre objetos de tamanhos extremos. A proposta de dividir o espaço em partição que armazenem classes de objetos de tamanho similar desdobra a gerência de espaço de caches em dois campos: a organização do espaço e a política de substituição. A regra para ordenação é dada pela política de reposição. Por exemplo, a política LRU gera uma lista LRU. A política LFU gera uma lista cuja chave de ordenação é o número de acessos a cada objeto. A lista é atualizada a cada requisição. A atualização consiste na reordenação da lista em caso de *hit* ou na

inserção com possibilidade de retirada em caso de *miss*. Cada inserção envolve no máximo uma retirada. Portanto o estado do cache muda lentamente – é quase estacionário – e a correlação entre o estado do cache no passado imediato e no futuro imediato é alta. Através do particionamento foi possível provar que o desempenho em HR e BHR (*byte-hit rate*) é função do tamanho médio dos arquivos solicitados e do tamanho médio dos *hits*. [Murta *et al.*, 1998].

3 O algoritmo LSR

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

3.1 Pseudo-código

A Figura 1 lista um pseudo-código para o algoritmo LSR. O algoritmo inicia-se com a invocação do procedimento *lsr* (linha 1), segundo o qual um objeto *n* deverá ser inserido no cache *C*, de acordo com sua semântica, removendo outros objetos se não houver espaço suficiente para *n*. Na linha 3 verifica-se se o tamanho de *n* é maior que o tamanho de *C*; se o for, aborta-se, pois não é possível inserir o objeto no cache, mesmo que este esteja completamente esvaziado. Na linha 4, a semântica *s* de *n* é extraída. Na linha 5, é definido o local *l* de *C* para a inserção de *n*, de acordo com *s*, isto é, determina-se qual é o assunto mais específico referente à informação contida em *n*. Na linha 6, verifica-se se o tamanho de *n* é maior que o espaço livre em *C*; se o for, na linha 7, cria-se espaço suficiente para que *n* caiba em *C*, removendo objetos que estejam o mais semanticamente distantes do local *l*. Finalmente, na linha 8, insere-se *n* no local *l*.

O procedimento de criação de espaço, invocado na linha 7, é detalhado a partir da linha 10. O conjunto *C* denota o próprio cache, isto é, o conjunto de todos os objetos em cache. O subconjunto *D*, $D \subseteq C$, denota os objetos mais distantes semanticamente do novo objeto *n*, em um certo instante, sendo que, dentre estes, todos têm a mesma distância semântica de *n* e, por isso, todos são igualmente candidatos a remoção. O subconjunto *R*, $R \subseteq D$ contém os objetos que devem ser efetivamente removidos. Esse conjunto é formado a partir de objetos extraídos do subconjunto *D*, utilizando-se para isso um algoritmo *S* que pode variar. Por exemplo, *S* pode ser qualquer um dos algoritmos descritos na Seção 2. Caso a remoção de todos os objetos pertencentes a *R* não criar espaço suficiente para a inserção de *n*, então um novo subconjunto *D* será definido, dando a mesma seqüência ao algoritmo. O pseudo-código, correspondente a esse princípio, situa-se entre as linhas 14 e 23, inclusive. Na linha 12, define-se que o espaço pendente de criação vai diminuindo à medida que objetos vão sendo removidos (linha 22). Na linha 13, invoca-se um procedimento para marcar todos os assuntos mais genéricos que o assunto ao qual o novo objeto deve ser vinculado, representado pelo local *l*. Isto é necessário para se dar prioridade a esse ramo de assuntos na escolha de objetos para remoção. Ao final do procedimento (linha 24), os ancestrais são desmarcados com invocação do procedimento adequado. Todos os procedimentos invocados pelo procedimento *crie_espaço* são dependentes da estrutura utilizada para classificação de assuntos.

```
[01] lsr ( in Cache C, in Objeto n )
       -- Insere um novo objeto n no cache C, de acordo com sua semântica e
       -- remove outros objetos, se não houver espaço suficiente para n.
[02] INÍCIO
[03]     SE (n.tamanho > C.tamanho) aborte;
[04]     s := extrai_semântica(n); -- Extrai informação semântica s de n
         -- Determina o local l do cache C para inserção do objeto n de acordo com s
[05]     l := define_local(C, s);
         -- Tamanho de n é maior que o espaço livre no cache C ?
[06]     SE (n.tamanho > C.espaço_livre)
[07]         ENTÃO cria_espaço (C, n, l); -- Cria espaço suficiente para n em C
[08]     insere(C, n, l); -- Insere n no local l do cache C
[09] FIM
[10] cria_espaço ( in Cache C, in Objeto n, in Local l )
        -- Remove objetos do cache C de acordo com o local l, tal que
        -- o espaço livre do cache seja suficiente para a inserção de n.
        -- O algoritmo de substituição S é um atributo do cache C.
[11] INÍCIO
           -- O espaço pendente de criação inicialmente é o próprio tamanho do objeto n menos
           -- o espaço atualmente disponível no cache.
[12]       p := n.tamanho - C.espaço_livre;
[13]       marque_assuntos_ancestrais( l );
[14]       REPITA
               -- Determina o conjunto D de objetos no cache C de menor relação semântica
               -- (maior distância semântica de) com n :
[15]           D := determine_objetos_mais_distantes (C, l);
[16]           R := { }; -- Inicia o conjunto R de objetos a serem removidos como vazio
[17]           REPITA
[18]               x := C.S(D); -- Aplica o algoritmo S em D, obtendo o objeto x
[19]               transfira(D ,R, x); -- Transfere x de D para R
[20]           ATÉ QUE ( vazio(D) ou R.espaço_ocupado >= p)
               -- D é vazio ou espaço ocupado por R é maior ou igual ao
               -- espaço pendente p de criação
[21]           remova(C, R) -- Remove os objetos de R do cache C
               -- Espaço pendente de criação é diminuído de acordo com os objetos removidos:
[22]           p := p - R.espaço_ocupado;
[23]     ATÉ QUE (p <= 0)--Espaço criado no cache C é suficiente para caber o objeto n ?
[24]     desmarque_assuntos_ancestrais( l );
[25] FIM
```

Figura 1: Pseudo-código para o algoritmo LSR

4 Implementação do LSR baseada em árvore

A classificação da informação pode ser baseada em uma hierarquia de assuntos, isto é, uma árvore cujos nós correspondam aos conjuntos e sub-conjuntos de assuntos [Rodriguez *et al.*, 1999]. Cada conjunto (nó da árvore) contém *m>=0* subconjuntos (subárvores), disjuntos S_1, S_2,..., S_m, e assim sucessivamente, e *n >= 0* objetos de informação. Esta estrutura, em particular, permite o refinamento do LSR com relação aos procedimentos invocados pelo procedimento *crie_espaço* (Figura 1, linha 10).

Uma hierarquia de assuntos é definida a partir da raiz ***Root***. No exemplo da Figura 2, os assuntos **X e Y** foram criadas sob o assunto ***Root***. Sob o assunto **X** foram criados os assuntos **A** e **B,** e sob o assunto **Y** foram criadas os assuntos **C** e **D**. Os objetos inseridos na hierarquia da Figura 2, identificados por #7, #12, #15, #3, #10, estão posicionados nos seus respectivos assuntos (nós da árvore), de acordo com a semântica de cada objeto de informação.

4.1 Refinamento do algoritmo LSR

Os procedimentos invocados pelo procedimento *cria_espaço* (Figura 1, linha 10) são específicos da estrutura utilizada para classificação da informação. Esses procedimentos podem ser implementados da seguinte forma:

- O procedimento *extrai_semântica* (Figura 1, linha 4) supõe que cada objeto de informação carrega, além da própria informação, uma seqüência de assuntos e sub-assuntos, de acordo com a hierarquia; esta seqüência identifica a própria semântica do objeto. Tal suposição não é irrealista, visto que o padrão para a definição de objetos de informação na Internet, o *XML*, e de descrição de recursos – o *RDF* –, componentes do Projeto *Semantic Web* [W3C, 2002], prevêem esse tipo de informação (meta-informação) sobre o objeto. Trata-se de um trabalho ainda em andamento e a sua discussão está fora do escopo deste trabalho.

- O procedimento *define_local* (Figura 1, linha 5) recebe a semântica do objeto, como parâmetro, no formato de seqüência de assuntos, conforme discutido para o procedimento *extrai_semântica*. Com isso, o trabalho do procedimento *define_local*, em uma árvore, fica trivial: basta seguir a seqüência até o seu final, pois para cada assunto há um nó correspondente.

- Os procedimentos *marque_assuntos_ancestrais* (Figura 1, linha 13) e *desmarque_assuntos_ancestrais* (linha 24) para um local da árvore, isto é, para um certo nó correspondente a um assunto, também têm implementação trivial em uma árvore: basta marcar ou desmarcar, respectivamente, o próprio nó e todos os seus ancestrais, recursivamente, até chegar à raiz da árvore.

- O procedimento *determine_objetos_mais_distantes* (Figura 1, linha 15) é mais complexo, e um pseudo-código está descrito na Figura 3. O procedimento supõe que para cada nó haja uma informação denominada *altura real* por indicar quantos níveis de descendentes existem, de forma que o descendente mais longínquo tenha pelo menos um objeto de informação vinculado, sendo que o próprio nó é considerado em sua *altura real*.

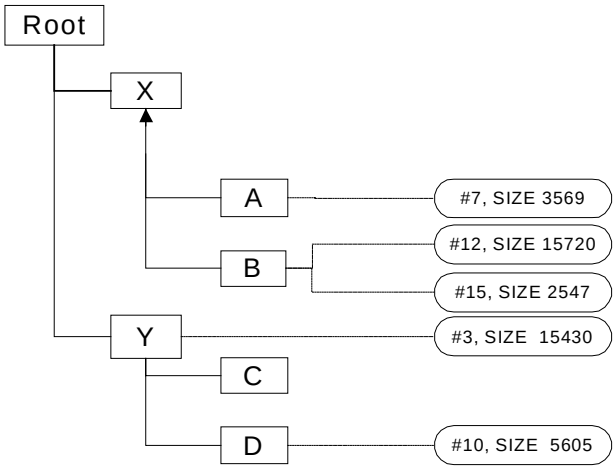


Figura 2: Hierarquia de categorias com objetos de informação e tamanho

```
ConjuntoDeObjetos determina_objetos_mais_distantes (in Cache C, in Local l)
-- Determina o conjunto de objetos no cache C de menor relação semântica
-- (maior distância semântica) com relação ao local l do cache, que indica um assunto.
INÍCIO
        alvo := assunto raiz;
        Candidatos := conjunto de assuntos descendentes do alvo com altura real > 0;
        -- alvo possui algum descendente (direto ou indireto) que tenha associado
        -- pelo menos um objeto de informação ? Ou, equivalentemente, altura real do alvo > 1 ?
        ENQUANTO (Candidatos não vazio)
        FAÇA
        INÍCIO
                eleito: = selecione o elemento de Candidatos com a maior altura real;
                ENQUANTO(eleito estiver marcado como "ancestral"
                        E existe um elemento de Candidatos que não seja o eleito)
                FAÇA
                        --remove o eleito de Candidatos
                        remove (Candidatos, eleito);
                        --elege um novo elemento de Candidatos
                        eleito := selecione o elemento de Candidatos com
                            a maior altura real;
                alvo := eleito;
                Candidatos := conjunto de assuntos descendentes do alvo com altura real > 0;
         FIM
 retorne o conjunto de objetos associado ao alvo;
FIM
```

Figura 3: Pseudo-código para o procedimento *determina_objetos_mais_distantes*

4.2 Exemplo de aplicação do LSR

A Figura 4 mostra uma árvore semântica antes (a) e depois (b) da inserção de um novo objeto de informação. Este novo objeto possui identidade #1, semântica G.H.A e tamanho 17Kb. O tamanho máximo do cache neste exemplo é 30Kb, sendo que, antes da inserção, tem um espaço ocupado de 28Kb. Logo, será necessário remover um ou mais objetos para que o novo ob-

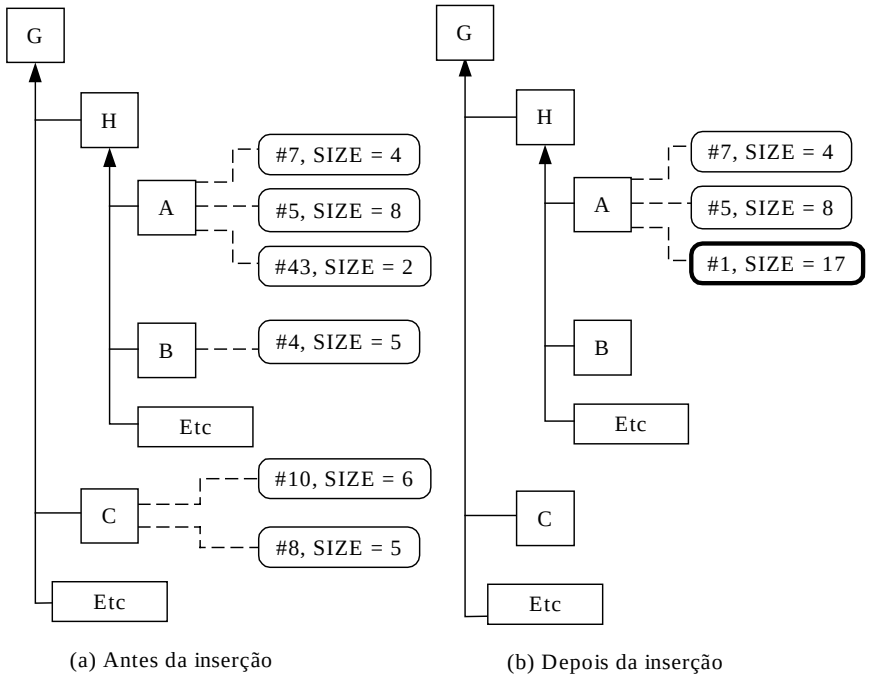


Figura 4: Árvore semântica antes e depois da inserção de um objeto

jeto seja inserido. Mais precisamente, será necessário remover objetos de tal forma que seja criado um espaço de pelo menos 15Kb, calculado por 28+17-30, isto é, *espaço ocupado + tamanho do novo objeto – tamanho do* cache. Aplicando-se o LSR, serão removidos do cache, pela ordem, os objetos #10, #8, #4 e #43, liberando um espaço total de 16Kb. Na verdade, a ordem de remoção entre os objetos #10 e #8 vai depender do algoritmo S empregado, pois este tem a mesma distância semântica do local de inserção (G.H.A).

5 Experimento e validação

O algoritmo LSR foi implementado para fins de validação e comparação com outros algoritmos tradicionalmente utilizados na Internet. Um módulo denominado *Cliente Internet* representa qualquer entidade da Internet que utilize um cache, como um *Web browser*, um *proxy* ou até mesmo um servidor Internet. O módulo *Cliente Internet* acessa uma seqüência de objetos de informação. Cada objeto de informação possui dois atributos: sua semântica e seu tamanho. Para essa seqüência não há qualquer preocupação de estrutura ou ordenação. O módulo *Gerenciador do Cache* contém o cache onde são armazenados os objetos de informação e uma árvore de classificação semântica dos objetos. Todo o controle do cache, isto é, a implementação do algoritmo LSR, é feito por este módulo.

Nesta seção é apresentado o processo de preparação de dados para a simulação, assim como os resultados obtidos e uma análise comparativa entre a estratégia LSR e as principais estratégias atualmente empregadas na Internet, nominalmente, as estratégias *SIZE* (considerada a estratégia mais eficiente na média para emprego em *proxies* compartilhados por muitos usuários), LFU e LRU. O objetivo do processo de preparação de dados para simulação de cache é obter uma seqüência válida de acessos a objetos. Essa seqüência deve, ao mesmo tempo, ter um fator de aleatoriedade, pois representa uma seqüência de acessos realizados por

um usuário qualquer, e ainda obedecer à mesma distribuição de acessos observada na Internet. Para atender ao quesito aleatoriedade foi aplicado o Método de Monte Carlo, o qual requer que os objetos estejam arranjados em categorias de acesso e que cada objeto tenha uma identificação única dentro da sua categoria.

5.1 Seqüência de acessos a objetos

A simulação do LSR baseia-se na implementação de uma hierarquia de categorias semânticas de objetos, de acordo com a sua semântica. Uma seqüência de acessos a objetos é obtida a partir dos objetos contidos nos nós da hierarquia de categorias. Assim, cada item da seqüência deve descrever o objeto acessado completamente, isto é, o caminho de nós no qual o objeto está inserido, a identificação única no universo de objetos (normalmente a sua URL) e o tamanho do objeto (normalmente em *bytes*). A relação a seguir ilustra uma possível seqüência de acessos a objetos da hierarquia apresentada na Figura 2.

Seqüência de acessos a objetos
Root.X.A;#7;3569
Root.X.B;#15;2547
Root.X.D;#10;560
Root.X.B;#12;15720
Root.X.A;#7;3569
Root.X.B;#15;2547
Root.X.D;#10;560
Root.X.B;#12;15720

5.2 Obtenção do universo de objetos para simulação

Foi utilizado um esquema para obtenção de um conjunto de objetos com suas respectivas semânticas que permitisse a simulação. Tal conjunto foi denominado *Universo de Objetos*, enquanto que o conjunto de todos caminhos (os quais definem semântica de objetos) correspondentes a esses objetos foi simplesmente denominado de *Caminhos***.** Cada elemento do *Universo de Objetos* é representado por três itens: (i) o caminho do objeto, (ii) a URL do objeto e (iii) o tamanho do objeto em bytes. Cada elemento de *Caminhos* é representado somente por um caminho de algum objeto contido no *Universo de Objetos*. Os conjuntos *Universo de Objetos* e *Caminhos* utilizados na simulação do LSR foram obtidos manualmente (Filtro *C*) do *site* do *Yahoo*!, pois este já contém uma classificação de objetos de informação. Foi extraído um subconjunto com 983 objetos, definindo, assim, um *Universo de Objetos* e o correspondente conjunto *Caminhos*, com 180 entradas. Neste experimento não será comtemplada toda a hierarquia do Yahoo!, obviamente, inclusive porque a classificação de assuntos neste trabalho é baseada em simples árvore, enquanto que no Yahoo! os assuntos são estruturados segundo um DAG (*directed-acyclic graph*) completo.

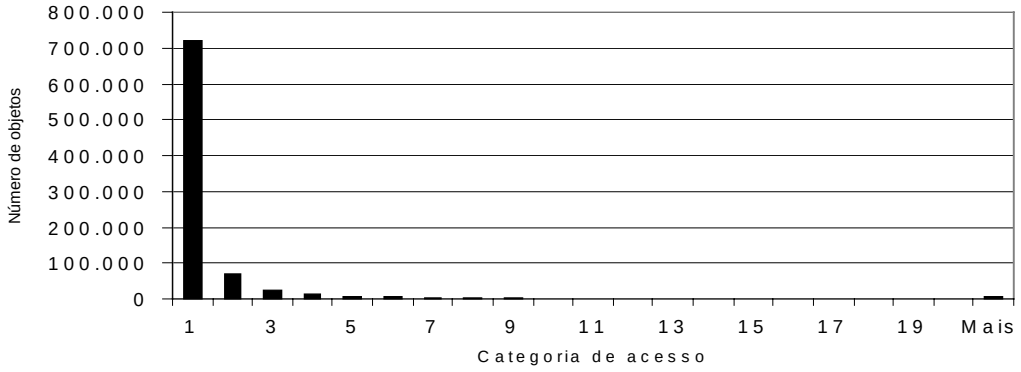
5.3 Obtenção da distribuição de acessos a objetos

Obteve-se a distribuição de acessos a objetos na Internet a partir do arquivo de registros de acessos do *proxy Squid*, por um certo período. A distribuição da quantidade de acessos pela quantidade de objetos, vindos do *proxy* foram essenciais para que se pudesse fazer a obtenção e a distribuição do maior número de objetos acessados pelos usuários. Foram utilizados cerca de 2.000.000 de registros de acessos a 866.915 objetos em um *proxy Squid*. O Gráfico 1 contém um histograma com a distribuição verificada. Nota-se, pela distribuição representada no gráfico, que há uma tendência de muitos objetos serem acessados poucas vezes e de poucos

objetos serem acessados muitas vezes. A partir das URLs com respectivos números de acessos, foi possível produzir uma distribuição da quantidade de acessos pela quantidade de objetos.

A seqüência de objetos acessados pelo *Squid* passou pelo Filtro[1] *A*, transformando-a numa seqüência com o número de acessos por objetos. Esses registros foram separados por URL, que por sua vez foram separadas pela quantidade de acessos, indicando a freqüência com que os usuários acessaram os objetos no período. Esta nova seqüência foi submetida ao Filtro *B* que gerou a distribuição de acessos para os objetos, definindo categorias de acesso e o número de objetos acessados em cada categoria. A distribuição constatada na Internet foi, portanto, reproduzida no Universo de Objetos selecionado. Cada entrada do Universo de Objetos foi acrescida com a categoria de acessos correspondente, pelos Filtro *D* e *E*, obtendo a Tabela Universo. Observou-se a correspondência entre o histograma obtido e o histograma mostrado no Gráfico 1, demonstrando que a distribuição dos objetos do Universo em categorias de acesso segue a mesma distribuição que se verificou na Internet.

Gráfico 1: Distribuição de acessos para os objetos do Squid



5.4 Obtenção de uma seqüência de acessos para simulação

A seqüência de acessos aleatórios ao Universo de Objetos foi obtida através da aplicação do Método de Monte Carlo, a fim de se respeitar a distribuição dos objetos em categorias de acessos. A Tabela 1, obtida a partir da Tabela Universo, contém todos os dados necessários para aplicação do Método de Monte Carlo. Cada linha da tabela corresponde a uma categoria ***c*** de acessos. Para cada categoria constam a quantidade ***q*** de objetos presentes na Tabela Universo, a probabilidade relativa da categoria e o correspondente intervalo entre zero e um. A probabilidade relativa de uma categoria é obtida pela divisão da quantidade de objetos da categoria pelo total de objetos:

$$P_c = \frac{q_c}{\sum_c q_c}$$

O limite inferior correspondente a uma categoria *c* é aberto, exceto se for zero, e dado pela somatória das probabilidades relativas das categorias anteriores:

$$\sum_{i=1}^{c-1} Pi$$

[1] Um Filtro é um programa que faz um arranjo em um conjunto de dados, segundo alguns critérios.

O limite superior correspondente a uma categoria ***c*** é fechado, exceto se for um, e dado pela somatória da probabilidade relativa da categoria ***c*** com as probabilidades das categorias seguintes:

$$\sum_{i=1}^{c} Pi$$

Categoria (Quantidade de acessos por objetos) c	Quantidade de objetos por categoria q	Probabilidade $P_c = \frac{q_c}{\sum_c q_c}$	Intervalo	
			Limite inferior (aberto) $\sum_{i=1}^{c-1} Pi$	Limite superior (fechado) $\sum_{i=1}^{c} Pi$
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
Somatória	43	1		
	$\sum_c q_c$	$\sum_c p_c$		

Tabela 1: Distribuição de probabilidade para aplicação do Método de Monte Carlo

Uma vez construída a tabela de probabilidades das categorias, faz-se a escolha aleatória de objetos que definirão uma seqüência de acessos. A escolha de um objeto deve ser contabilizada e, se o número de acessos a esse objeto atingir o seu limite (o próprio número que identifica a categoria), o objeto deve ser marcado como inválido para escolha dentro da sua categoria.

5.5 Resultados da simulação

A eficiência de um cache pode ser medida pela probabilidade de acerto, que é a ocorrência de um acesso a um objeto já presente no cache. Tal probabilidade pode ser obtida em função da variação do tamanho total do cache ou em função do número de acessos. 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 a objetos. 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.

Observou-se que todas as estratégias tendem a manter o cache o mais ocupado possível, o que demonstra que as implementações dessas estratégias estão corretas neste sentido. (Todas as demais combinações permitiriam essa mesma observação e, por isso, não são aqui representadas.) O Gráfico 2 mostra o efeito do número de objetos acessados em uma seqüência na probabilidade de acerto. Observa-se que, a probabilidade de acerto para todas as estratégias aumenta, tendendo a um patamar próximo de 1 a medida que aumenta o número de acessos a

objetos. Pode-se observar também que, para as quatro estratégias, a probabilidade de acerto aumenta a medida que aumenta o tamanho do cache. Essa constatação pode ser mais facilmente notada nos Gráfico 3, que mostra o efeito do tamanho do cache na probabilidade de acerto para um determinado número de acessos. O Gráfico 3 permite ainda observar que há uma convergência na probabilidade de acerto entre as quatro estratégias a medida que o cache

Gráfico 2: Efeito do número de objetos acessados em uma seqüência na probabilidade de acerto para um cache de tamanho 2.000.000

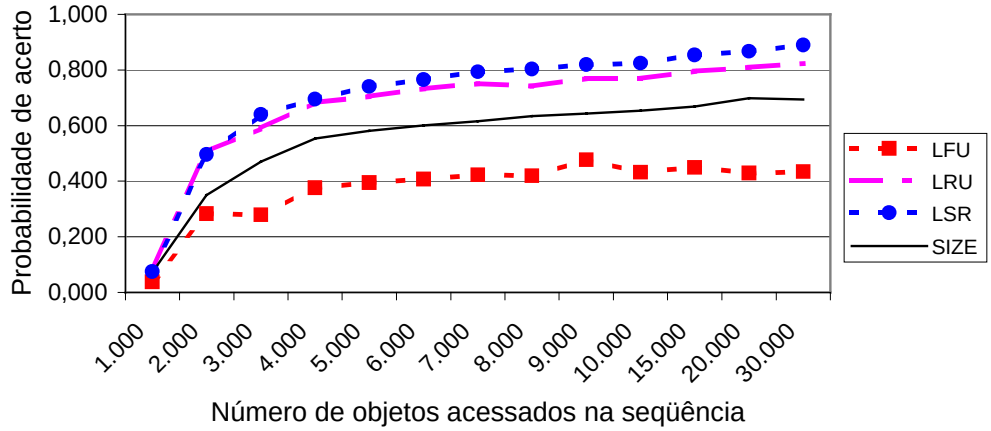
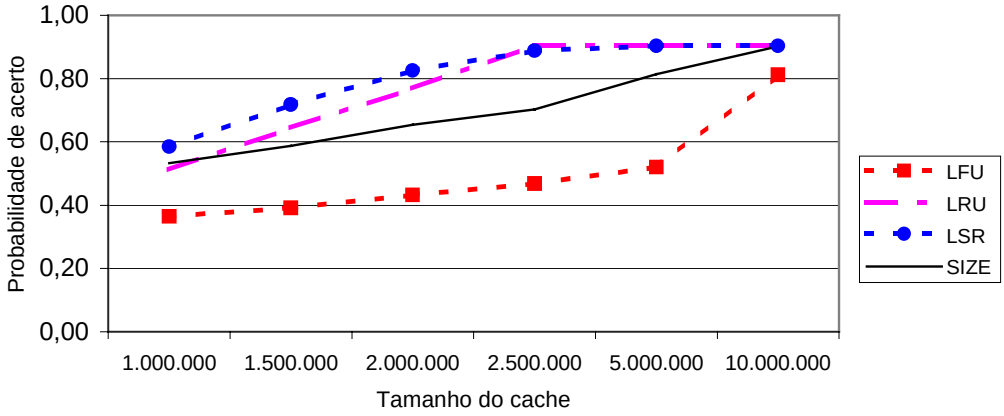


Gráfico 3: Efeito do tamanho do cache na probabilidade de acerto para 10.000 acessos



aumenta, isto é, a estratégia adotada não influencia na probabilidade de acerto. A explicação para isso é que, a medida que o cache aumenta, mais chance tem de conter todo o Universo de Objetos. 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. 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. O que nota-se, de fato, é que existe algum ganho no uso de LSR.

6 Conclusões e propostas de trabalhos futuros

Este artigo propõe uma solução inovadora para o tratamento de caches na Internet com relação à substituição de objetos: a estratégia LSR, que utiliza a classificação dos objetos para dar prioridade de permanência no cache aos objetos mais relacionados semanticamente com o objeto que se deseja inserir no cache. Um algoritmo e uma estrutura de dados para a estratégia LSR foram completamente projetados e implementados para fins de validação e comparação com outras estratégias de substituição de objetos em cache; foram também implementadas as estratégias SIZE, LFU e LRU. Além disso, foi projetado um modelo para a realização dos experimentos, incluindo toda a coleta e preparação de dados para análise. Os resultados experimentais mostraram que a estratégia proposta oferece, na maioria das situações, uma medida de eficiência – probabilidade de acerto – superior às outras estratégias. Esses indicadores favoráveis justificam a continuidade do trabalho apresentado; abaixo seguem alguns trabalhos futuros.

(1) O desenvolvimento da estratégia LSR levou em consideração a disponibilização da semântica da informação contida em cada objeto. Com a adoção de XML e RDF, isso tende a tornar-se factível. Assim, uma direção para pesquisas futuras é explorar esse padrão, inclusive na organização da árvore semântica com a qual pode-se considerar estabelecer uma classificação padrão de objetos de informação. Certamente isso possibilitará a aplicação da estratégia LSR em sistemas reais, tornando possível uma análise sua em um ambiente mais favorável. Uma conseqüência direta da aplicação da estratégia LSR em sistemas reais é a sua implementação em *browsers, proxies* e servidores, o que exigirá mais esforço de desenvolvimento e pesquisas. Certamente, nesse cenário, no qual haveria milhões de entradas na árvore semântica, a eficiência do LSR deveria ser novamente avaliada.

(2) Uma das melhorias que podem ser feitas na estratégia LSR é criar nós de assuntos na árvore de semântica, de forma dinâmica, a fim de refletir novos assuntos de interesse para os clientes e, assim, automaticamente aumentar a probabilidade de acertos do cache. Um sinal de que a árvore precisa ser revista, isto é, que novos assuntos devem ser considerados, é quando os nós *etc* se encontram com muitos objetos de informação associados. Seria possível, ainda, controlar o cache do cliente para saber o perfil do cliente nas consultas e, a partir daí, dinamicamente, configurar a árvore de assuntos para o cliente.

(3) Outra melhoria é evitar remover um objeto eleito como o mais distante semanticamente mas que tenha alto valor (por exemplo, uso freqüente, alto custo para *download*, etc) para outro cliente – o que viabilizaria a aplicação da estratégia LSR para caches multiusuários – ou outra *thread* de consultas do mesmo cliente, supondo que um mesmo cliente possa estar consultando objetos sobre mais de um assunto simultaneamente.

(4) Uma questão importante a ser respondida é como a estrutura da árvore de assuntos afeta o desempenho do cache, pois a estrutura em árvore pode ser custosa ou vantajosa para se inserir e remover objetos, quando se compara com o desempenho obtido por estratégias que utilizam estruturas lineares. Esta questão depende de uma futura melhoria da representação de hierarquia de assuntos para comtemplar a hierarquia do Yahoo!, que se baseia em DAG. Além disso, é preciso considerar a possibilidade de gerar e atualizar a árvore dinamicamente.

(5) O algoritmo projetado e experimentado para a estratégia LSR considera, para efeito de escolher os objetos menos relacionados semanticamente, somente o objeto que se deseja inserir no cache. Uma melhoria imediata é a armazenagem do histórico das semânticas mais recentemente acessadas e utilizar tal histórico para decidir quais os objetos menos relacionados semanticamente não apenas com o último acessado, mas considerando uma janela no pas-

sado de acessos. Certamente, essa modificação deve favorecer ainda mais a estratégia LSR em sua comparação com as demais.

Portanto, as principais contribuições desse artigo são a proposta da nova estratégia de substituição de objetos em cache, a sua implementação e validação, assim como o próprio modelo e correspondentes ferramentas (chamados de filtros) de obtenção de dados para a validação. Os resultados iniciais de medição de eficiência são favoráveis à estratégia LSR e percebe-se que as perspectivas de aplicações e trabalhos futuros de pesquisa e desenvolvimento são muitas. O completo desenvolvimento do LSR – resolvendo-se a pendência de incorporar semântica aos objetos e utilizando-se estruturas de dados adequadas a grandes massas de dados – permitirá a sua utilização real na Internet, possibilitando a comprovação dos resultados obtidos e contribuindo, desta forma, para o avanço da tecnologia de redes de computadores.

Referências bibliográficas

[Aggarwal *et al*., 1999] Charu Aggarwal, Joel L. Wolf e Philip S. Yu. *Caching on the World Wide Web, IEEE Computer, 11(1):94-105.* http://cs-www.bu.edu/faculty/best/res/papers/sdne95.ps.

[Cao e Irani, 1997] Pei Cao e Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)* (Monterey, CA, Dec. 1997). http://www.cs.wisc.edu/~cao/papers/gd-size.ps.Z.

[Krishnamurthy e Wills, 1999] Balachander Krishnamurthy e Craig E. Wills. Proxy cache coherency and replacement-Towards a more complete picture. *In Proceedings of the 19 th IEEE International Conference on Distributed Computing Systems, Austin, TX, (June 1999).* http://www.att.com/~bala/papers/ccrcp.ps.gz

[Murta *et al.*, 1998] Cristiana Duarte Murta, Virgilio Almeida e Wagner Meira Jr. Analyzing performance of partitioned caches for the WWW. In *Proceedings of the 3rd International WWW Caching Workshop* (June 1998). http://wwwcache.ja.net/events/workshop/24/.

[Niclause *et al.*, 1998] Nicolas Niclause, Zhen Liu e Philippe Nain. A New and Efficient Caching Policy for the World Wid Web. In *Proceedings of the 1998 Workshop on Internet Server Performance*, June 1998.

[O'Neil *et al*., 1993] Elizabeth J. O'Neil, Patrick E. O'Neil e Gerhard Weikum. The LRU-K page replacement algorithm for database disk buffering. *In Proceedings of the ACM SIGMOD International Conference on Management of Data.*

[Rizzo e Vicisano, 1998] Luigi Rizzo e Lorenzo Vicisano. Replacement policies for a proxy cache. Tech. Rep. RN/98/13, UCL-CS, 1998. http://www.iet.unipi.it/~luigi/lrv98.ps.gz.

[Rodriguez *et al*., 1999] Pablo Rodriguez, Christian Spanner e Ernest W. Biersack. Web caching architectures: Hierarchical and distributed caching. In *Proceedings of the 4th International Web Caching Workshop* (Apr. 1999). http://www.ircache.net/Cache/Workshop99/Papers/rodriguez-final.ps.gz.

[Santos, 2001] Rogério Guaraci dos Santos. Substituição de Objetos em Cache na WWW baseada na Semântica da Informação. Dissertação de Mestrado. Pontifícia Universidade Católica do Paraná, Programa de Pós-Graduação em Informática Aplicada. Setembro de 2001.

[Scheuermann *et al*., 1997] Peter Scheuermann e Junho Shim e Radek Vingralek. A case for delay-conscious caching of Web documents. In *Proceedings of the 6th International WWW Conference* (Santa Clara, Apr. 1997).

http://www.scope.gmd.de/info/www6/technical/paper020/paper20.html.

[Silberschatz e Galvin, 1994] Silberschatz, A. and Galvin, P. B. (1994). Operating Systems Concepts. Addison-Wesley, Reading, Mass., fourth edition.

[Wessels, 1995] Duane Wessels, D. Intelligent caching for World-Wide Web objects. In *Proceedings of the INET '95 conference* (Honolulu, Hawaï, June 1995). http://www.nlanr.net/~wessels/Papers/wessels-inet95/wessels-inet95.ps.gz.

[Williams *et al.*, 1996] Stephen Williams, Marc Abrams, Standridge, C. R., Abdulla, G., and Fox, E. A. Removal policies in network caches for World-Wide Web documents. In *Procedings of the ACM SIGCOMM '96 Conference* (Stanford University, CA, Aug. 1996). http://ei.cs.vt.edu/~succeed/96sigcomm/.

[Wooster e Abrams, 1997] Roland P. Wooster, e Marc Abrams. Proxy caching that estimate page load delays. In *Proceedings of the 6rd International WWW Conference* (Apr. 1997). http://www.scope.gmd.de/info/www6/technical/paper250/paper250.html.

[W3C, 2002] The World Wide Web Consortium (W3C). http://www.w3.org/