IGOR DE SOUZA PAIVA

SUBSTITUIÇÃO DE OBJETOS EM CACHE NA INTERNET USANDO MODELO VETORIAL PARA COMPARAÇÃO SEMÂNTICA DA INFORMAÇÃO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Orientador: Prof. Dr. Alcides Calsavara

CURITIBA

2005

IGOR DE SOUZA PAIVA

SUBSTITUIÇÃO DE OBJETOS EM CACHE NA INTERNET USANDO MODELO VETORIAL PARA COMPARAÇÃO SEMÂNTICA DA INFORMAÇÃO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: *Sistemas Distribuídos*

Orientador: Prof. Dr. Alcides Calsavara

CURITIBA

2005

Paiva, Igor de Souza

Substituição de Objetos em *Cache* na Internet Usando Modelo Vetorial para Comparação Semântica da Informação. Curitiba, 2005.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. *Cache* 2. Semântica 3. Modelo Vetorial 4. Objetos. I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada

TERMO DE APROVAÇÃO

Dedico este trabalho às memórias

de minha mãe Sandra Paiva e meu avô Rolando de Souza.

AGRADECIMENTOS

Agradeço a Deus que sempre iluminou meu caminho.

Agradeço especialmente a minha esposa Caroline, que esteve sempre ao meu lado durante este trabalho, muitas vezes privando-se de momentos de lazer para me apoiar.

Ao meu pai Ivan Paiva, que sempre colocou em primeiro lugar a educação e formação profissional de seus filhos, não medindo esforços para isto.

À minha mãe Sandra Paiva e meu avô Rolando de Souza, cujos ensinamentos e exemplos foram de fundamental importância para minha formação como pessoa. Também aos meus irmãos Alex Paiva e Janaína Paiva e à minha avó Filomena de Castro.

Ao meu orientador Prof. Dr. Alcides Calsavara, pelo direcionamento, paciência e compreensão nos momentos decisivos deste trabalho.

Aos meus colegas de mestrado que trabalharam ao meu lado compartilhando experiências e sabedoria, em especial aos amigos Vera Marchiori, Ana Carolina Pilatti, Rafael Parpinelli, Simone Ferreira e Marcelo Schuck

À Goldtower Informática Ltda. e às Faculdades SPEI que permitiram meu afastamento parcial do trabalho para que pudesse freqüentar as aulas para obtensão dos créditos necessários ao mestrado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

LISTA DE ABREVIATURAS

ANS	*Advanced Network and Services*
ARP	*Address Resolution Protocol*
CD	*Compact Disks*
CGI	*Common Gateway Interface*
CIX	*Commercial Internet eXchange association*
CPU	*Central Processing Unit*
CRLF	*Carriage Return Line Feed*
DAT	*Digital Audio Tape*
DNS	*Domain Name Service*
DRAM	*Dynamic Random Access Memory*
FIX	*Federal Internet eXchange points*
FTP	*File Transfer Protocol*
GB	*Gigabytes*
GD	*GreedyDual*
GD-Size	*GreedyDual-Size*
GNU	*General Public License*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Secure-HTTP*
ICMP	*Internet Control Message Protocol*
IRC	*Internet Relay Chat*
KB	Quilo *bytes*
Kbps	Quilo *bits* por segundo
LCN-R	*Least Normalized Cost Replacement*
LDAP	*Lightweight Directory Access Protocol*
LFU	*Least Frequently Used*
LRU	*Least Recently Used*
LRV	*Lowest Relative Value*

LSR	*Least Semantically Related*
LSR-VM	*Least Semantically Related - Vector Model*
MB	*Megabytes*
Mbps	Mega *bits* por segundo
ms	Milisegundos
MAC	*Media Access Control*
NFS	*Network File System*
ns	Nanosegundos
NSF	*National Science Foundation*
PUCPR	Pontifícia Universidade Católica do Paraná
RAM	*Random Access Memory*
RARP	*Reverse Address Resolution Protocol*
RDF	*Resource Description Framework*
RFC	*Request For Comment*
SLRU	*Size-Ajusted* LRU
SMTP	*Simple Message Transfer Protocol*
SRAM	*Static Random Access Memory*
SSL	*Secure Socket Layer*
TCP/IP	*Transmission Control Protocol / Internet Protocol*
TFTP	*Trivial File Transfer Protocol*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
UTC	*Coordinated Universal Time*
WSU	*Wayne State University*
WWW	*World Wide Web*

RESUMO

Esta dissertação de mestrado apresenta uma nova abordagem para a comparação de objetos armazenados em mecanismos de *cache* através da análise da semântica da informação contida nestes objetos. A comparação semântica é utilizada como critério para a substituição de objetos armazenados em *cache*, diferentemente das abordagens clássicas que utilizam propriedades operacionais de cada objeto como parâmetro para as tarefas de seleção e substituição. A utilização de uma estratégia baseada em semântica tende a manter no mecanismo de *cache* os objetos semanticamente próximos, ou seja, os objetos que apresentam afinidade entre si e que são de interesse para o cliente. O LSR (*Least Semantically Related*) foi a primeira estratégia de substituição baseada em semântica. Esta estratégia organiza os objetos mantidos pelo mecanismo de *cache* em uma estrutura hierárquica, em forma de árvore semântica, através da qual os objetos são classificados de acordo com assuntos. Porém o LSR não prevê a obtenção da semântica a partir da análise do conteúdo dos próprios objetos, sendo necessário que estes sejam previamente classificados de acordo com uma taxonomia representada por uma árvore semântica. Além disso, o LSR pressupõe que esta semântica é fornecida juntamente com cada objeto, o que não acontece na prática. Este trabalho introduz uma nova estratégia denominada *Least Semantically Related – Vector Model* (LSR-VM), que propõe a substituição da árvore semântica por um mecanismo que utiliza o modelo vetorial, universalmente utilizado em problemas de recuperação de informações, como estratégia para a organização, classificação, comparação e cálculo da distância semântica entre os objetos mantidos no mecanismo de *cache* de forma dinâmica. Ao final desta dissertação são apresentados os resultados obtidos a partir de simulações realizadas com a estratégia LSR-VM. Os resultados obtidos são comparados com as estratégias de substituição tradicionais SIZE, LFU (*Least Frequently Used*) e LRU (*Least Recently Used*).

Palavras-Chave: *Cache*, Algoritmos de Substituição, Semântica, LSR, Modelo Vetorial.

ABSTRACT

This research work presents a new approach to compare objects stored in cache engines through semantic analysis of object contents. Semantic comparison between objects is used as a replacement strategy, differently from classical approaches that use object's physical characteristics as parameters to their selection and replacement strategies. A replacement strategy based on semantic comparison tends to keep in cache, objects that share semantics that are relevant to clients and consequently are related to each other. LSR (Least Semantically Related) was the first replacement strategy based on semantics. That strategy organizes objects in a tree structure of subjects. Each object is associated to one or more subjects from this semantic tree. However, LSR doesn't obtain semantics from objects by analyzing their contents. LSR assumes that each object is previously classified according to a taxonomy that is represented by a semantic tree. This research work presents a new strategy called LSR-VM (Least semantically Related – Vector Model) that eliminates the need for a semantic tree of subjects. To establish the semantic comparison between objects inside the cache, LSR-VM uses a vector model strategy that is typically used to solve information retrieval problems. Vector model is a technique to organize, classify, compare and calculate the semantic distance between two objects. This research work presents the results obtained from LSR-VM strategy simulations, as well as comparisons to other traditional replacement strategies such as SIZE, LFU (Least Frequently Used) and LRU (Least Recently Used).

Keywords *Cache, Replacement Algorithms, Semantics, LSR, Vector Model.*

1. INTRODUÇÃO

1.1 CONTEXTO

Atualmente os mecanismos de *cache* exercem um papel fundamental na Internet, minimizando o problema de congestionamento e sobrecarga de servidores. Estudos realizados pelo governo dos Estados Unidos em 2002 mostraram que 67% da população norte americana, correspondente a 174 milhões de pessoas, são usuárias da Internet. O mesmo estudo demonstrou que a adesão de novos usuários cresce numa taxa de dois milhões ao mês [USDC02]. Uma grande parte das informações que trafegam na Internet é composta de objetos que são requisitados várias vezes por diversos clientes. Pode-se observar que clientes freqüentemente requisitam os mesmos conjuntos de objetos. Toda vez que um cliente executa seu programa para navegação na Internet, uma cópia da página inicial, previamente configurada, é recuperada. Este tipo de comportamento se apresenta de forma freqüente, trazendo impacto principalmente para os servidores *web* e para a infra-estrutura da Internet devido ao aumento da utilização de banda. Neste contexto fica evidente a importância dos mecanismos de *cache*, que têm como principais objetivos a redução do tempo médio de acesso a objetos e a redução do número de requisições em servidores *web*.

A idéia por trás de um mecanismo de *cache* é bastante simples. Pode-se reduzir o tráfego e aumentar a performance na Internet através do armazenamento de objetos comumente requisitados em locais próximos aos clientes. Estes lugares podem ser servidores de *cache* ou até mesmo o próprio programa navegador do cliente. Desta forma, várias cópias de um mesmo objeto que originalmente reside em um servidor *web*, podem estar presentes em um ou mais servidores de *cache* que podem estar distribuídos em uma arquitetura hierárquica colaborativa. Com isto, os objetos freqüentemente acessados serão recuperados a partir de um servidor de *cache* e não

mais a partir de seus locais originais.

Existem aspectos positivos e negativos a serem considerados quanto ao funcionamento de um mecanismo de *cache*. Além de diminuir o consumo de banda e o tempo de transferência dos objetos, a utilização de *caching* reduz a sobrecarga imposta aos servidores *web*, onde os objetos estão originalmente armazenados. Além disso, se por algum motivo um servidor *web* ficar indisponível, ainda assim seus objetos poderão ser recuperados a partir dos servidores *cache*. Os servidores *cache* geralmente possuem recursos para o registro de informações estatísticas sobre os acessos realizados. Através da utilização destes recursos poder-se-ia, entre outras coisas, estabelecer o perfil de interesses do grupo de usuários que utiliza o servidor de *cache*. Por outro lado, existem alguns problemas que podem surgir com a utilização de *caching*. Um dos principais problemas está relacionado à atualização das cópias de objetos que se encontram distribuídas em servidores de *cache*. Um cliente pode recuperar um objeto desatualizado sem tomar conhecimento disto. A utilização de um único servidor de *cache* é totalmente desaconselhável, pois este pode se tornar um único ponto de congestionamento ou falha para toda a rede. Devido a estes aspectos a arquitetura de *caching* em uma rede deve ser muito bem planejada para evitar problemas e enfatizar os benefícios.

Um mecanismo de *cache* deve apresentar algumas características importantes para que possa ser realmente eficaz na obtenção dos objetivos a que se propõe. Os mecanismos de *cache* devem possuir robustez, rapidez, eficiência, estabilidade, transparência, escalabilidade, balanceamento de carga e capacidade de lidar com heterogeneidade [WAN99].

1.2 MOTIVAÇÃO

Existem vários problemas que os mecanismos de *cache* se propõem a resolver. Uma das principais questões gira em torno de quais objetos devem ser

armazenados e quais objetos devem ser removidos quando o limite de armazenamento do mecanismo de *cache* for atingido. Para resolver esta questão os mecanismos de *cache* utilizam estratégias que gerenciam o problema de substituição. Várias estratégias de substituição foram propostas desde a criação dos mecanismos de *cache*, porém a maioria dos estudos realizados nesta área baseia-se em características operacionais dos objetos como critério para substituição, tais como tamanho, custo de acesso, freqüência de acesso, tempo de validade, etc [CAL01].

O *Least Semantically Related* (LSR) [SAN01] é uma estratégia de substituição que se diferencia significativamente das demais, pois se baseia na semântica dos objetos como critério para seleção e substituição. A estratégia LSR explora o princípio de que os clientes tendem a requisitar novos objetos que estejam relacionados com objetos atualmente armazenados no mecanismo de *cache*. Desta forma, quando um novo objeto tiver que ser inserido em um *cache* cujo limite tenha sido atingido, o LSR descartará o objeto que tem menos afinidade semântica com o novo objeto.

O LSR se apresenta como uma idéia inovadora, porém existem alguns problemas que inviabilizam a sua utilização prática. Em sua concepção original a estratégia LSR organiza os objetos mantidos pelo mecanismo de *cache* em uma taxonomia, em forma de árvore semântica, através da qual os objetos são classificados de acordo com assuntos. Porém o LSR não prevê a extração da informação semântica dos objetos de forma dinâmica, sendo necessário que estes sejam previamente classificados de acordo com assuntos contidos na árvore. O principal problema para a utilização do LSR de forma prática reside no fato de que o LSR pressupõe que a classificação semântica é fornecida juntamente com cada objeto.

Uma tentativa para solucionar o problema de fornecimento da semântica juntamente com os objetos foi proposta em [SCH02]. Este trabalho propõe a criação de um servidor de semântica que armazena a classificação dos objetos. Desta forma, quando o mecanismo de *cache* necessitar da semântica de um novo objeto, poderá

obtê-la através de uma requisição ao servidor de semântica. Na realidade esta estratégia somente transfere a responsabilidade do gerenciamento da semântica, do mecanismo de *cache* para o servidor de semântica. Mesmo utilizando um servidor de semântica os objetos ainda precisarão ser previamente classificados em conceitos de forma manual.

O problema da estratégia LSR reside essencialmente na utilização de uma estrutura hierárquica em forma de árvore para organizar e calcular a distância semântica entre os objetos. Esta estrutura pode ser substituída por uma implementação da teoria do modelo vetorial, que além de possibilitar o cálculo da distância semântica entre os objetos, também possibilita a comparação destes, de forma dinâmica.

1.3 OBJETIVOS

O objetivo geral deste trabalho é apresentar os resultados da aplicação da teoria do modelo vetorial, tradicionalmente presente em problemas de recuperação da informação, utilizado como alternativa à árvore originalmente proposta para organizar, classificar e calcular a distância semântica entre os objetos em um mecanismo de *cache*. Para isto foi criada uma nova estratégia de substituição denominada *Least Semantically Related – Vector Model* (LSR-VM).

A estratégia de substituição LSR-VM propõe uma solução para os problemas presentes na estratégia de substituição LSR original, permitindo a comparação da semântica da informação existente nos objetos, de forma dinâmica. O LSR-VM não necessita de nenhum tipo de solução tecnológica para obter a semântica dos objetos como, por exemplo, um servidor de semântica [SCH02]. O cálculo da distância semântica entre os objetos é obtido através da teoria do modelo vetorial. Desta forma, quando um novo objeto tiver de ser inserido em um mecanismo de *cache* cujo limite de armazenamento tenha sido alcançado, os objetos que estiverem semanticamente mais distantes do novo objeto serão progressivamente descartados até que seja

liberado espaço suficiente para acomodá-lo.

Utilizando-se o modelo vetorial, cada objeto é considerado como um documento que pode ser associado a um vetor com *n* termos, onde *n* é o número total de termos distintos encontrados em todos os documentos de uma coleção [SAL83]. A partir do momento em que os documentos estão logicamente representados através de vetores, é possível identificar os documentos que mais se aproximam semanticamente em um espaço vetorial. O processo de geração dos vetores que representam os documentos não é reversível, sendo assim, não é possível reconstruir o documento original a partir do vetor que o representa.

O capítulo 2 desta dissertação apresenta um histórico da Internet, destacando a sua evolução ao longo do tempo e apresentado uma visão geral do conjunto de protocolos TCP/IP. Este capítulo apresenta também as principais características e o funcionamento básico do protocolo HTTP (*Hypertext Transfer Protocol*) com o objetivo de estabelecer a base de entendimento para que seja possível analisar o problema de substituição em *cache* na Internet.

O capítulo 3 desta dissertação apresenta os conceitos de *cache* aplicados tanto a problemas tradicionais de arquitetura de computadores e sistemas operacionais, quanto aos problemas de congestionamento de tráfego e sobrecarga em servidores *web*. Este capítulo apresenta também detalhes sobre a infra-estrutura e arquiteturas de *cache* na Internet. Ao final deste capítulo é apresentado o funcionamento do software *Squid* que é um dos servidores *proxy* mais utilizados atualmente na Internet. O *proxy Squid* é distribuído sob a modalidade de *software* livre através da GNU (*General Public License*).

O capítulo 4 desta dissertação apresenta o problema de substituição em *cache*. São apresentadas as estratégias de substituição tradicionais, que levam em consideração as características físicas e operacionais dos objetos como critério para substituição. Este capítulo apresenta também três estratégias de substituição baseadas em semântica, que levam em consideração o conteúdo dos objetos como critério de

substituição quando houver necessidade de criar espaço no *cache.*

O capítulo 5 apresenta a estratégia de substituição LSR-VM (*Least Semantically Related – Vector Model)*, tema principal desta dissertação. É apresentada uma visão geral dos mecanismos de busca na Internet, com destaque à teoria do modelo vetorial, que é utilizada na estratégia LSR-VM para medir a distância semântica entre objetos armazenados no *cache*. Ao final deste capítulo é apresentado o tratamento dado pela estratégia LSR-VM a objetos binários, dos quais não se pode extrair semântica de forma direta. A estratégia LSR-VM apresenta uma proposta para contemplar também objetos binários durante a comparação semântica para substituição em *cache*.

O capítulo 6 desta dissertação apresenta o *framework* que foi criado para permitir a simulação das estratégias de substituição e obtenção dos resultados. Este *framework* permite a implementação, validação e realização de testes de novas estratégias de substituição em *cache*, usando históricos de acessos reais, extraídos de servidores *proxy*.

O capítulo 7 apresenta os resultados obtidos através das simulações realizadas com a estratégia de substituição LSR-VM, demonstrando a taxa de acerto (*hit rate*) e a quantidade de *bytes* transferidos (*byte hit rate*) em relação às estratégias de substituição SIZE, LRU, LFU, comumente utilizadas em mecanismos de *cache*. Durante os experimentos e trabalhos de pesquisa foram analisados arquivos de *log* gerados por servidores *proxy* de duas instituições de ensino localizadas na cidade de Curitiba, Paraná.

O capítulo 8 apresenta a conclusão deste trabalho de pesquisa com destaque às principais contribuições e aos trabalhos futuros.

2. INFRA-ESTRUTURA DA INTERNET

2.1 HISTÓRICO DA INTERNET

A Internet surgiu no final dos anos 60 como um experimento do ARPA, *Advanced Research Projects Agency* (Agência de Projetos de Pesquisas Avançadas), um órgão do departamento de defesa dos Estados Unidos. O projeto tinha como objetivo a conexão dos computadores dos principais centros de pesquisa da ARPA. Em 1969 iniciou-se a conexão de quatro localidades através de circuitos de 56 Kbps. Em dezembro do mesmo ano a rede experimental denominada ARPANET entrou no ar conectando as Universidade de Los Angeles e Santa Barbara na Califórnia, a Universidade de Utah e o Instituto de pesquisa de Stanford. A tecnologia utilizada levou a criação de outras duas redes similares para fins militares, a MILNET nos Estados Unidos e a MINET na Europa. Durante a década de 70, várias universidades e órgãos do governo norte americano se conectaram à ARPANET, resultando numa intensa atividade de pesquisa que teve como principal resultado a criação de um conjunto de protocolos conhecidos como TCP/IP, que se tornou a base para a Internet até hoje.

A ARPANET tinha uma política interna que proibia a exploração comercial da rede. Em 1985 a ARPANET estava intensamente congestionada, o que levou ao início do desenvolvimento da NSFNET, patrocinada pela *National Science Foundation* ou NSF (Fundação Nacional de Ciência). Esta rede era composta de um *Backbone*[1], que interligava múltiplas redes regionais a seis centros nacionais de supercomputadores. A estas redes regionais estavam conectados vários campi universitários e instituições de pesquisa como a NASA Science Network.

[1] Segmento principal de uma rede de computadores.

A velocidade inicialmente utilizada pelos *links* dos centros nacionais de supercomputadores era de 56 Kbps. Em 1988 A NSFNET passou a ter como parceiros a IBM, MCI (empresa de telecomunicações) e a Merit Networks Inc. (Instituição ligada ao estado de Michigan). Estas novas parcerias patrocinaram alterações nos *links* que resultaram num aumento de velocidade para 1,433Mbps (T1).

Em 1990 a IBM, MCI e Merit fundaram uma nova organização chamada ANS (*Advanced Network and Services*). A partir de então a ANS passou a gerenciar os roteadores da NSFNET através de um centro de operações. A ANS definiu também, um banco de dados com políticas de roteamento para a rede.

Ao final de 1991 o volume de utilização da NSFNET apresentou um intenso crescimento, resultando na melhoria dos links que compunham o seu Backbone para linha tipo T3 com 45Mbps. A figura 1 mostra a estrutura original da NSFNET.

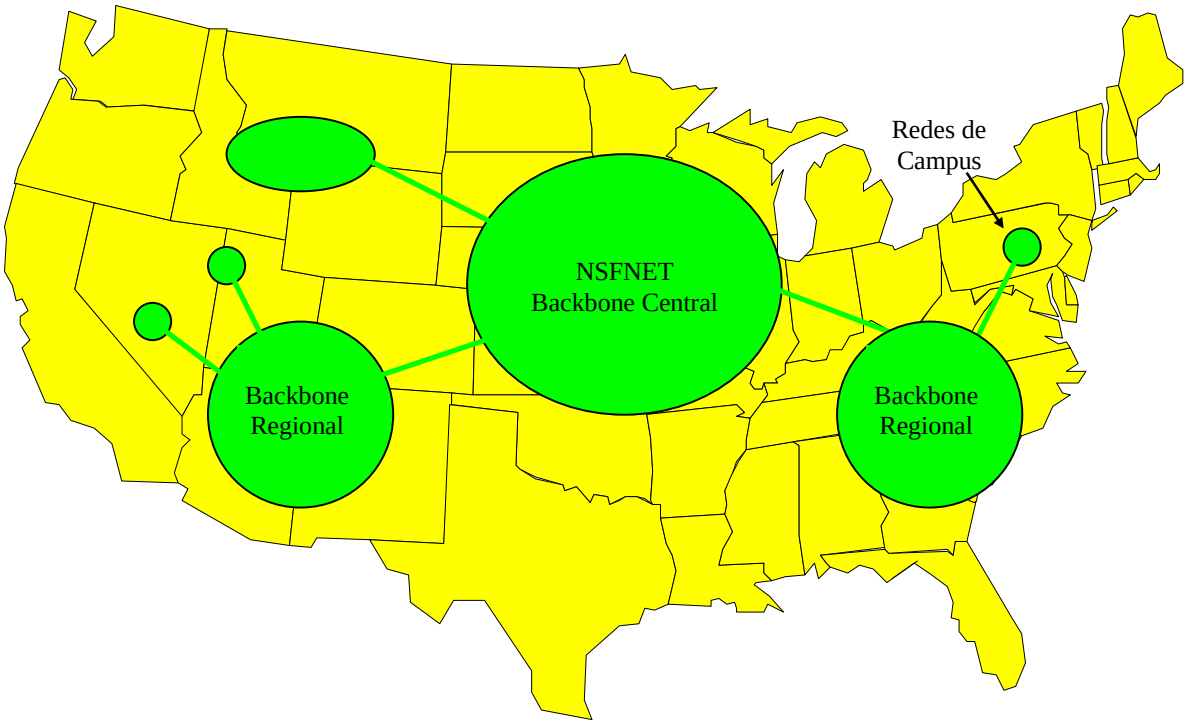


FIGURA 1 - ESTRUTURA ORIGINAL DA NSFNET

No início dos anos 90 a NSFNET ainda impunha restrições quanto a sua utilização para fins comerciais, porém as pressões aumentaram principalmente pelo surgimento de novas redes que tinham necessidade de se interconectar. Desta forma começaram a surgir os provedores de acesso com o objetivo de disponibilizar o acesso para organizações com fins diversos, inclusive comercial. Organizações fora dos Estados Unidos que já vinham desenvolvendo suas redes, também manifestavam interesse de realizar conexões internacionais.

Com a abertura da NSFNET as redes das agências do governo Americano se conectaram a pontos federais de intercâmbio da Internet [HAL97] (FIX - *Federal Internet eXchange points*). As redes de organizações comerciais se reuniram criando a Associação Comercial de Intercâmbio da Internet [HAL97] (CIX - *Commercial Internet eXchange association*). Ao mesmo tempo provedores de acesso por todo mundo já tinham desenvolvido uma substancial infra-estrutura de comunicação e interconectividade. A Sprint, uma empresa privada, foi escolhida pela NSFNET para gerenciar as conexões internacionais, promovendo a interconexão dos *backbones* existentes nos Estados Unidos, com redes na Europa e Ásia.

Em Abril 1995 a NSFNET foi desativada. Este processo teve de ser realizado de forma gradativa para assegurar a continuidade das conexões existentes entre as diversas instituições e agências do governo Norte Americano, com as redes regionais. A estrutura da Internet atualmente existente sofreu mudanças, passando de um núcleo centralizado (NSFNET) a uma arquitetura distribuída, operada por provedores comerciais por todo o mundo.

A desativação da NSFNET foi o acontecimento que efetivamente definiu a estrutura da Internet como a conhecemos hoje. Atualmente existem centenas de milhares de provedores de acesso que disputam ferozmente uma participação mais expressiva no mercado.

O crescimento astronômico da Internet trouxe diversos desafios tecnológicos a serem vencidos.

2.2 CONJUNTO DE PROTOCOLOS TCP/IP

Os protocolos de comunicação foram criados para possibilitar que dois ou mais computadores possam trocar informações através de um processo controlado que segue um padrão predefinido. O principal conjunto de protocolos que viabilizou a expansão da Internet foi o conjunto de protocolos TCP/IP.

No início dos anos 70 surgiu a necessidade de desenvolvimento de um protocolo que viabilizasse o projeto de criação da ARPANET. Desta forma, a partir de um intenso processo de pesquisa, o Departamento de Defesa dos Estados Unidos definiu um modelo para comunicação de dados entre sistemas abertos que era composto de quatro camadas. Cada camada é constituída de um número de protocolos, sendo que ao conjunto dos protocolos que compõem as quatro camadas foi dado o nome de TCP/IP [LEW97]. A especificação técnica de cada protocolo que compõe o conjunto TCP/IP está definida em documentos RFC (*Request For Comment*) que estão publicamente disponíveis na Internet. As camadas do modelo TCP/IP são:

Camada 4 - Aplicação: Uma série de protocolos compõe a camada de aplicação do modelo TCP/IP. Alguns dos principais protocolos que compõem esta camada são:

- TELNET: Disponibiliza emulação de terminal para *login* em um sistema remoto.
- FTP (*File Transfer Protocol*): Este protocolo é utilizado para a transferência interativa de arquivos.
- SMTP (*Simple Message Transfer Protocol*): Este protocolo gerencia o envio e recepção de correio eletrônico entre sistemas.
- DNS (*Domain Name Service*): Realiza a resolução de nomes de máquinas para endereços IP.
- HTTP (*Hypertext Transfer Protocol*): Protocolo utilizado para

requisição de páginas *web* na Internet.

- NFS (*Network File System*): Este protocolo permite o compartilhamento de diretórios entre máquinas de uma rede.

Cada um destes protocolos requer um módulo servidor e outro cliente que geralmente são executados em máquinas diferentes. Os módulos servidores rodam em segundo plano em servidores e são carregados automaticamente durante o processo de iniciação do sistema operacional. Todos estes protocolos estão disponíveis em máquinas que utilizam sistemas operacionais que suportam o conjunto de protocolos TCP/IP. O sistema operacional UNIX foi o primeiro a possuir uma implementação completa do modelo TCP/IP desde suas primeiras versões. Atualmente praticamente todos os sistemas operacionais suportam o conjunto de protocolos TCP/IP.

Camada 3 - Transporte: A camada de transporte tem a função de entregar pacotes entre as camadas de Aplicação e Internet. Esta camada utiliza o conceito de portas. A cada aplicação rodando em uma máquina é atribuído um número de porta que pode ser entendido como o endereço da aplicação naquela máquina. A camada de Transporte utiliza dois tipos de protocolos. O primeiro tipo de protocolo é o TCP (*Transmission Control Protocol*) [RFC793] que é definido como um protocolo orientado a conexão. Isto quer dizer que o protocolo TCP garante a entrega de pacotes através de um mecanismo que estabelece uma conexão criando um circuito virtual entre duas máquinas. Nesta conexão a máquina que deseja iniciar a transmissão envia uma solicitação à máquina destino, que pode aceitar ou recusar a solicitação.

Um protocolo orientado a conexão garante que todos os pacotes enviados serão recebidos em ordem correta, sem perda ou duplicação. Caso isto não seja possível a conexão é interrompida. Este tipo de protocolo garante também a preservação de conexões já existentes através da recusa de novas solicitações de conexão, quando a rede ficar congestionada. Alguns dos principais protocolos da

camada de Aplicação que utilizam o TCP como base são o HTTP, TELNET, FTP e SMTP.

O outro tipo de protocolo utilizado por esta camada é o UDP (*User Datagram Protocol*) [RFC768] que é categorizado como protocolo não orientado a conexão. Neste tipo de protocolo os dados são enviados através da rede e supõe-se que estes chegaram com sucesso ao seu destino. Caso existam múltiplos caminhos entre as máquinas origem e destino, pode ocorrer a recepção de pacotes fora de ordem. Este tipo de protocolo deixa a cargo da aplicação o tratamento desse tipo de problema. Alguns dos principais protocolos da camada de Aplicação que utilizam o UDP como base são o TFTP (*Trivial File Transfer Protocol*), NFS e IRC.

Camada 2 - Internet: O principal objetivo da camada 2 do modelo TCP/IP é de rotear pacotes entre máquinas através de uma ou mais redes diferentes, objetivo este que é atingido através do esquema de endereçamento IP. Esta camada é composta de quatro protocolos, porém o mais importante é o protocolo IP (Internet Protocol) [RFC791], que transporta todo tráfego enviado e recebido. Os outros protocolos são o ICMP (Internet Control Message Protocol) [RFC793], ARP (Address Resolution Protocol) e RARP (Reverse Address Resolution Protocol).

O protocolo IP não é orientado a conexão, portanto não garante a entrega de pacotes através da rede. O cabeçalho de um quadro IP é composto de vários campos, tendo como mais importantes os campos de **endereço de origem**, **endereço de destino** e **tempo de vida**. Os endereços de origem e destino são utilizados para identificar as máquinas que estão trocando informações. O campo **tempo de vida** é utilizado para evitar que pacotes fiquem circulando indefinidamente entre roteadores sem nunca atingir seu destino, permitindo assim, que sejam descartados.

O protocolo ICMP (*Internet Control Message Protocol*) executa quatro funções principais:

- **Controle de fluxo**: Fornece um mecanismo que permite a interrupção temporária do envio de dados quando uma máquina receptora se torna sobrecarregada. Esta interrupção é realizada através de uma solicitação à máquina que está enviando os dados.
- **Alertas de destino não atingível**: Caso alguma máquina da rede detecte que um determinado endereço não é possível de ser atingido devido a inexistência da máquina destino ou à queda de um enlace, uma mensagem de alerta é enviada para a máquina de origem.
- **Redirecionamento de rotas**: Um roteador utiliza este tipo de mensagem para informar a uma máquina de origem que esta deve utilizar outro roteador. Este procedimento é tipicamente utilizado quando um roteador identifica um caminho mais eficiente através de outra rota, para uma máquina que está enviando dados.
- **Verificação de máquina remota**: Este tipo de mensagem é utilizada para verificar se um determinado endereço IP está ativo na rede. Este recurso está disponível na maioria dos sistemas operacionais através do utilitário "*Ping*".

Os protocolos ARP (*Address Resolution Protocol*) e RARP (*Reverse Address Resolution Protocol*) são responsáveis pela resolução de endereços IP em endereços físicos Ethernet ou Token-Ring IEEE 802.2, denominados endereços MAC (*Media Access Control*). Toda máquina que utiliza o TCP/IP como protocolo de comunicação possui uma tabela interna de resolução de endereços (*Address Resolution Table*). O endereço MAC é responsável pela efetiva entrega dos pacotes em uma rede local. Quando uma máquina deseja enviar um pacote para outra máquina da rede, esta procura o endereço MAC da máquina destino em sua tabela de resolução de endereços. Caso a máquina destino não esteja na tabela, uma mensagem em difusão é enviada pela rede. Se a máquina destino estiver ligada e suportar o protocolo ARP, esta

responderá com o seu endereço MAC. O protocolo RARP é utilizado por estações sem disco que, ao serem ligadas, necessitam solicitar a um servido RARP o seu endereço IP.

Camada 1 - Acesso à Rede: A camada de acesso à rede é composta de um conjunto de componentes de hardware e software. Esta camada não “entende” endereços IP, trabalhando apenas com endereços que dizem respeito ao segmento de rede ao qual uma máquina está ligada. Esta camada utiliza o endereço MAC (*Media Access Control*) para entregar pacotes a máquinas que fazem parte de um determinado segmento de rede. Os endereços MAC não são transmitidos além dos roteadores.

2.3 PROTOCOLO HTTP

Os primeiros usuários da Internet utilizavam serviços básicos oferecidos pelo conjunto de protocolos TCP/IP como, por exemplo, transferência de arquivos (FTP), emulação de terminal (TELNET), resolução de nome (DNS), etc. Um dos serviços mais utilizados neste período foi o GOPHER. Este serviço tinha como objetivo organizar e apresentar arquivos texto através de uma estrutura hierárquica. Os usuários podiam navegar por uma estrutura de menus em lista e visualizar arquivos armazenados em um servidor GOPHER.

Um dos principais fatores que culminou na grande expansão da Internet por todo o mundo foi o surgimento do serviço *World Wide Web* (WWW). Este serviço também tinha como objetivo permitir que usuários visualizassem documentos armazenados em servidores. O serviço WWW logo deixou obsoleto o serviço GOPHER, pois ao contrário deste, podia apresentar documentos multimídia contendo gráficos, imagens, etc. A infra-estrutura da *World Wide Web* é composta de um sistema de servidores que disponibilizam acesso a documentos formatados em uma linguagem denominada *Hypertext Markup Language* (HTML), que suporta a

existência de *links* entres documentos possibilitando que o usuário vá de um documento para outro através da seleção de um *link*. A linguagem HTML é responsável apenas pela formatação dos documentos, definindo como estes serão apresentados aos usuários. Para visualizar documentos na WWW os usuários necessitam de programas especiais denominados navegadores *web*.

Para tornar o serviço WWW viável foi desenvolvido um protocolo denominado HTTP (*Hypertext Transfer Protocol*) que é responsável pela comunicação entre servidores de documentos, popularmente conhecidos como servidores *web*, e os navegadores *web* utilizados pelos usuários para a visualização dos documentos, conforme apresenta a figura 2.

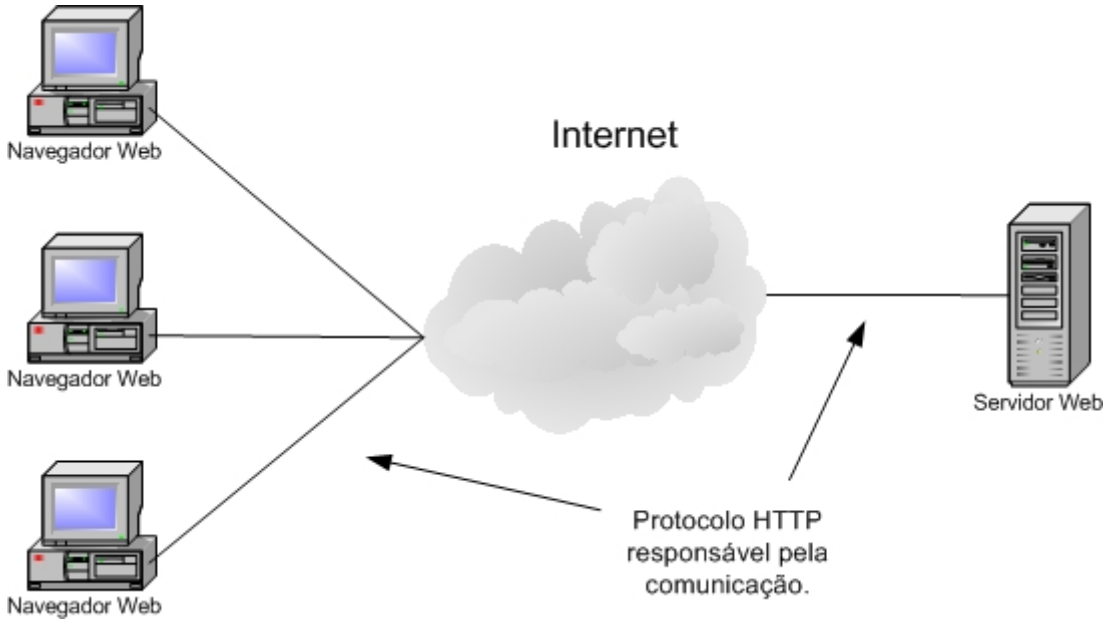


FIGURA 2 - INFRA-ESTRUTURA DO SERVIÇO WWW.

O HTTP é um protocolo da Camada de Aplicação do conjunto de protocolos TCP/IP que começou a ser usado na Internet por volta de 1990 para prover a infra-estrutura necessária para o acesso a documentos no formato de hipertexto distribuídos em servidores por toda a Internet. O protocolo HTTP segue o modelo cliente/servidor. Neste contexto, os navegadores *web*, que estão em execução em vários usuários conectados à Internet, realizam requisições a um ou mais servidores *web* onde estão

armazenados documentos hipertexto juntamente com outros objetos referenciados por estes, como por exemplo, imagens, vídeos, som, etc.

O HTTP é um protocolo de requisição e respostas que não mantém estado (*stateless*). Isto significa que cada operação de requisição/resposta não depende de operações realizadas anteriormente. Esta característica impõe certa complexidade na criação de aplicações baseadas em ambiente *web*. Para suprir esta necessidade surgiram diversas tecnologias atualmente utilizadas como, por exemplo, Java Servlet, Microsoft ASP .NET, CGI (*Common Grateway Interface*), etc.

Quando um usuário requisita uma página usando um navegador *web*, uma seqüência de ações é realizada culminando com a apresentação do resultado final para o usuário. Esta seqüência se dá da seguinte forma:

1. O cliente informa uma URI (*Uniform Resource Identifier*) que corresponde ao endereço de um objeto, neste caso a página desejada, na *web*;
2. A partir da URI fornecida, o programa navegador obtém o endereço IP do servidor remoto que contém a página desejada. Neste momento outros serviços do conjunto de protocolos TCP/IP podem ser utilizados como, por exemplo, o DNS.
3. O programa navegador estabelece uma conexão TCP/IP com o servidor remoto;
4. O cliente envia uma requisição HTTP para o servidor através da conexão estabelecida;
5. O servidor processa a requisição HTTP gerando uma resposta HTTP que é enviada de volta para o cliente. Uma vez enviada a resposta o servidor encerra a conexão;
6. O programa navegador recebe a resposta.

Se a resposta retornada pelo servidor for um arquivo no formato HTML o programa navegador percorrerá o arquivo procurando por referências para outros objetos necessários para compor o resultado final para o usuário como, por exemplo, imagens, vídeos, sons, etc. Neste caso o programa navegador realizará novas requisições até que tenha recuperado todos os objetos necessários.

2.3.1 Mensagem de Requisição do Protocolo HTTP

Uma mensagem de requisição HTTP é composta dos seguintes itens:

- Linha de Requisição (*Request-Line*)
  - Método de Requisição (*Request-Method*)
  - URI (*Uniform Resource Identifier*)
  - Versão do protocolo HTTP
- Cabeçalho Genérico
- Cabeçalho de Requisição
- Cabeçalho de Entidade
- CRLF (*Carriage Return Line Feed*)
- Corpo da Mensagem

Os cabeçalhos são compostos de um ou mais campos que se apresentam no formato “Nome-do-Campo=Valor-do-Campo”. Os campos carregam meta-informação a respeito da mensagem de requisição ou de resposta.

Os campos que compõem o cabeçalho genérico podem aparecer tanto em mensagens de requisição quanto em mensagens de resposta. Os campos do cabeçalho genérico suportados pela versão 1.1 do protocolo HTTP [RFC2616] são:

- *Cache-Control*
- *Connection*
- *Date*

- *Pragma*
- *Trailer*
- *Transfer-Encoding*
- *Upgrade*
- *Via*
- *Warning*

Os campos que compõem o cabeçalho de requisição somente podem aparecer em mensagens de requisição. Os campos do cabeçalho de requisição suportados pela versão 1.1 do protocolo HTTP [RFC2616] são:

- *Accept*
- *Accept-Charset*
- *Accept-Encoding*
- *Accept-Language*
- *Authorization*
- *Expect*
- *From*
- *Host*
- *If-Match*
- *If-Modified-Since*
- *If-None-Match*
- *If-Range*
- *If-Unmodified-Since*
- *Max-Forwards*
- *Proxy-Authorization*
- *Range*
- *Referer*
- *TE*
- *User-Agent*

De acordo com o conceito definido na especificação do protocolo HTTP [RFC2616], **entidades** correspondem a informações que são “carregadas” por mensagens de requisição ou de resposta. Geralmente as entidades representam informações trocadas entre clientes e servidores *web* que implementam o protocolo HTTP.

Os campos que compõem o cabeçalho de entidade podem aparecer tanto em mensagens de requisição quanto em mensagens de resposta. Os campos de entidade carregam meta-informação do domínio da aplicação através de mensagens HTTP. Não existem campos de cabeçalho de entidade predefinidos [RFC2616], desta forma tanto o navegador (cliente) quanto o servidor *web* são livres para definir os campos de entidade necessários durante a comunicação.

TABELA 1 - MENSAGEM MÍNIMA DE REQUISIÇÃO HTTP.

Mensagem de Requisição HTTP
GET http://www.yahoo.com/ HTTP/1.1

A tabela 1 apresenta uma mensagem de requisição HTTP mínima. O termo “GET” representa o método de requisição (*request-method*), a URI é representada pelo texto “*http://www.yahoo.com/*” e a versão do protocolo é representada pelo texto “HTTP/1.1”.

TABELA 2 - MENSAGEM DE REQUISIÇÃO HTTP.

Mensagem de Requisição HTTP
GET / HTTP/1.1 Accept: */* Accept-Language: pt-br Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) Host: www.google.com.br Connection: Keep-Alive

A tabela 2 apresenta uma mensagem de requisição composta de um maior número de elementos. Nesta mensagem é possível observar os campos que compõem os cabeçalhos da requisição como, por exemplo, o campo "*Host*" que é utilizado na formação da URI. Desta forma a URI que está sendo realmente requisitada na mensagem é "*http://www.google.com.br/*". A primeira linha da requisição é chamada de "Linha de Requisição" (*Request Line*). O detalhamento de todo o protocolo HTTP não faz parte do escopo desta dissertação, porém é importante apresentar os métodos de requisição suportados:

- ***OPTIONS***: Este método é usado para recuperar informações a respeito das opções de comunicação disponíveis na cadeia de requisição/resposta identificada pela URI. O método serve também para recuperar informações do servidor *web* sem que haja necessidade de realizar a recuperação de um objeto.
- ***GET***: Este método é utilizado para recuperar o objeto ou as informações identificadas pela URI. Em certos casos o método "*GET*" pode ser considerado como uma recuperação condicional caso os campos "*If-Modified-Since*", "*If-Unmodified-Since*", "*If-Match*", "*If-None-Match*", ou "*If-Range*" estejam presentes na requisição. Estes campos são utilizados principalmente por servidores *proxy* para minimizar a tráfego na Internet. Quando a URI possui parâmetros fornecidos pelo usuário ou pelo navegador, estes parâmetros são enviados como parte integrante da URI na linha de requisição.
- ***HEAD***: Este método é igual ao método "*GET*", porém a resposta do servidor *web* não conterá o corpo da mensagem. Os cabeçalhos da mensagem de resposta devem ser iguais aos de uma requisição realizada através do método "*GET*".

- ***POST***: O método "*POST*" é usado principalmente em operações de submissão de um formulário. Neste caso as informações do formulário são enviadas para o servidor *web* como parte da mensagem de requisição, porém estas informações não aparecem na linha de requisição. Os dados do formulário são enviados como campos de entidade. Uma requisição através do método "*POST*" pode não resultar em um objeto que possa ser identificado por uma URI.
- ***PUT***: Este método é usado para armazenar o objeto presente no conteúdo da mensagem de requisição em um servidor *web*. O objeto é armazenado no local estipulado pela URI. Este método é usado basicamente para a transferência de objetos de um cliente para um servidor *web*.
- ***DELETE***: Este método solicita que o servidor *web* exclua o objeto identificado por uma URI.
- ***TRACE***: O método "*TRACE*" implementa um mecanismo de *loop-back* com o servidor *web*. A resposta a uma requisição deste tipo contém a própria mensagem de requisição enviada ao servidor. Este método é usado principalmente em processos de depuração de erros.
- ***CONNECT***: Este é um método especial que é utilizado por servidores *proxy* que podem se tornar um *Tunnel* dinamicamente. Um *tunnel* pode ser criado durante a requisição de um objeto na *web* através de uma conexão segura utilizando o protocolo SSL (*Secure Socket Layer*).

Um fato importante a ser considerado é que somente os métodos de requisição "*GET*" e "*HEAD*" são passíveis de *caching* em servidores *proxy* na Internet ou em uma rede local de computadores. Requisições realizadas através dos demais métodos devem obrigatoriamente ser submetidas aos servidores *web* de origem para

que estes possam processar as requisições gerando mensagens de resposta que serão devolvidas aos clientes. O próximo capítulo apresenta conceitos de *cache* aplicados ao problema de congestionamento em redes de computadores, bem como maiores detalhes sobre a infra-estrutura de *cache* na *web*.

2.3.2 Mensagem de Resposta do Protocolo HTTP

Após receber e processar uma mensagem de requisição, um servidor *web* gera uma mensagem de resposta que é devolvida para o navegador do cliente.

Uma mensagem de resposta HTTP é composta dos seguintes itens:

- Linha de Status (*Status-Line*)
  - Versão do protocolo HTTP
  - Código de Status (*Status-Code*)
  - Descrição do Motivo (*Reason-Phrase*)
  - URI (*Uniform Resource Identifier*)
- Cabeçalho Genérico
- Cabeçalho de Requisição
- Cabeçalho de Entidade
- CRLF (*Carriage Return Line Feed*)
- Corpo da Mensagem

O código de status de uma mensagem de resposta apresenta o resultado do processamento da mensagem de requisição pelo servidor *web*. Este é um campo numérico que possui três dígitos, sendo que o primeiro dígito define a categoria da resposta. A tabela 3 apresenta os códigos de status do protocolo HTTP, que podem ser classificados em cinco categorias de respostas:

- **1xx - Resposta Informativa**: Define que a mensagem de requisição foi recebida e que o servidor *web* está processando a requisição.

- **2xx - Resposta com Sucesso**: A mensagem de requisição foi recebida, compreendida e aceita pelo servidor *web*.
- **3xx - Resposta de Redirecionamento**: Ações subseqüentes devem ser tomadas para completar a mensagem de requisição.
- **4xx - Resposta Indicando Erro no Cliente**: A mensagem de requisição contém erros de sintaxe ou não pode ser atendida.
- **5xx - Resposta Indicando Erro no Servidor**: O servidor falhou durante a tentativa de atender a uma mensagem de requisição aparentemente válida.

TABELA 3 - CÓDIGOS DE STATUS DE UMA MENSAGEM DE RESPOSTA HTTP.

Código	Descrição	Código	Descrição
"100"	*Continue*	"404"	*Not Found*
"101"	*Switching Protocols*	"405"	*Method Not Allowed*
"200"	*OK*	"406"	*Not Acceptable*
"201"	*Created*	"407"	*Proxy Authentication Required*
"202"	*Accepted*	"408"	*Request Time-out*
"203"	*Non-Authoritative Information*	"409"	*Conflict*
"204"	*No Content*	"410"	*Gone*
"205"	*Reset Content*	"411"	*Length Required*
"206"	*Partial Content*	"412"	*Precondition Failed*
"300"	*Multiple Choices*	"413"	*Request Entity Too Large*
"301"	*Moved Permanently*	"414"	*Request-URI Too Large*
"302"	*Found*	"415"	*Unsupported Media Type*
"303"	*See Other*	"416"	*Requested range not satisfiable*
"304"	*Not Modified*	"417"	*Expectation Failed*
"305"	*Use Proxy*	"500"	*Internal Server Error*
"307"	*Temporary Redirect*	"501"	*Not Implemented*
"400"	*Bad Request*	"502"	*Bad Gateway*
"401"	*Unauthorized*	"503"	*Service Unavailable*
"402"	*Payment Required*	"504"	*Gateway Time-out*
"403"	*Forbidden*	"505"	*HTTP Version not supported*

A tabela 4 apresenta a parte inicial de uma mensagem de resposta HTTP. É possível observar que o código de *status* apresenta o valor “200” informando que a requisição foi recebida, entendida e processada. O corpo da mensagem apresenta parte de um arquivo HTML.

TABELA 4 - MENSAGEM DE RESPOSTA HTTP.

Mensagem de Resposta HTTP
HTTP/1.1 200 OK Date: Wed, 13 Jul 2005 03:35:15 GMT Server: Apache/2.0.48 (Fedora) Last-Modified: Fri, 11 Feb 2005 02:11:46 GMT Accept-Ranges: bytes Content-Length: 6071 Connection: close Content-Type: text/html <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"> <HTML><HEAD> ……

2.3.3 Componentes Envolvidos em uma Comunicação Através do Protocolo HTTP

A comunicação através do protocolo HTTP é iniciada por um cliente usando um navegador. Considerando o cenário mais simples possível este cliente requisita um objeto, identificado por uma URI, a um servidor *web* através de uma conexão TCP/IP direta. Neste caso os únicos componentes envolvidos na comunicação foram o cliente e o servidor *web*, porém existem situações mais complexas onde outros componentes podem estar envolvidos. Os outros componentes que podem estar envolvidos na comunicação são:

- ***Proxy***: Um *proxy* é um componente que é responsável por encaminhar mensagens de requisição HTTP a servidores *web*. O *proxy* recebe uma mensagem de requisição HTTP de um cliente, reescreve total ou parcialmente esta mensagem e a encaminha ao servidor *web* destino identificado na URI, conforme apresenta a figura 3. Durante esta operação o *proxy* pode armazenar uma cópia do objeto recuperado.

Desta forma, se este ou outros clientes associados ao *proxy* requisitarem o objeto novamente, o *proxy* poderá retornar a cópia armazenada, evitando que o objeto seja novamente recuperado a partir do servidor *web* original. Isto diminui o tráfego na *web* e a latência (tempo de resposta para o cliente).

- ***Gateway***: Um *gateway* é um componente intermediário entre um cliente e um servidor *web*, porém, diferentemente de um *proxy*, um *gateway* recebe uma mensagem de requisição como se fosse o servidor *web* destino, conforme apresenta a figura 4. Esta requisição é repassada ao servidor *web* destino de forma transparente. Desta forma nem o cliente nem o servidor *web* têm ciência de que a comunicação está sendo realizada através de um *gateway*. Um *gateway* não faz nenhum tipo de *cache* ou armazenamento de objetos.
- ***Tunnel***: Um *tunnel* é um componente que atua como um agente de repasse de informação entre duas extremidades de uma conexão TCP/IP. A criação de um *tunnel* pode ser iniciada a partir de uma mensagem de requisição HTTP, porém, uma vez que o *tunnel* estiver estabelecido, o protocolo HTTP passa a não ter nenhum controle sobre o *tunnel*. Este mecanismo é utilizado principalmente quando há necessidade de comunicação através de uma conexão segura usando o protocolo SSL (*Secure Socket Layer*) para criptografar as informações que são transmitidas entre um cliente e um servidor *web*.

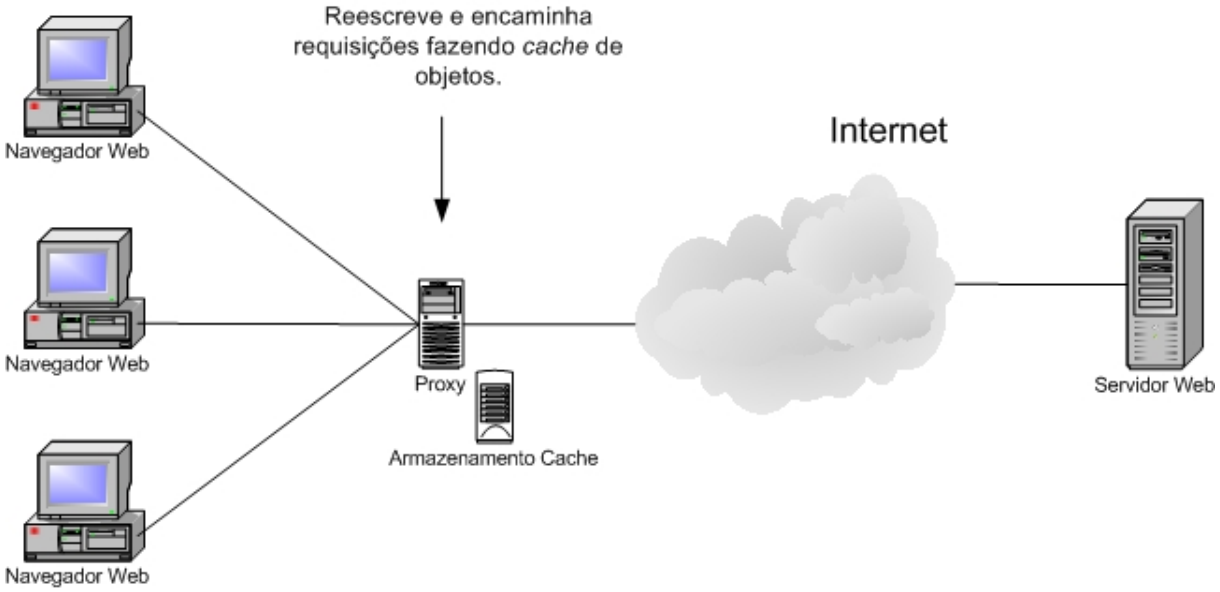


FIGURA 3 - PROXY UTILIZADO EM UMA COMUNICAÇÃO COM O PROTOCOLO HTTP.

O próximo capítulo desta dissertação apresenta mais detalhes relacionados a utilização do conceito de *cache* em redes de computadores e da utilização de *proxies* na *web*. São apresentadas características de um servidor *proxy* denominado *Squid*, que é largamente utilizado na Internet. Este servidor *proxy* foi utilizado no processo de coleta de dados para as simulações apresentadas no capítulo 7 desta dissertação.

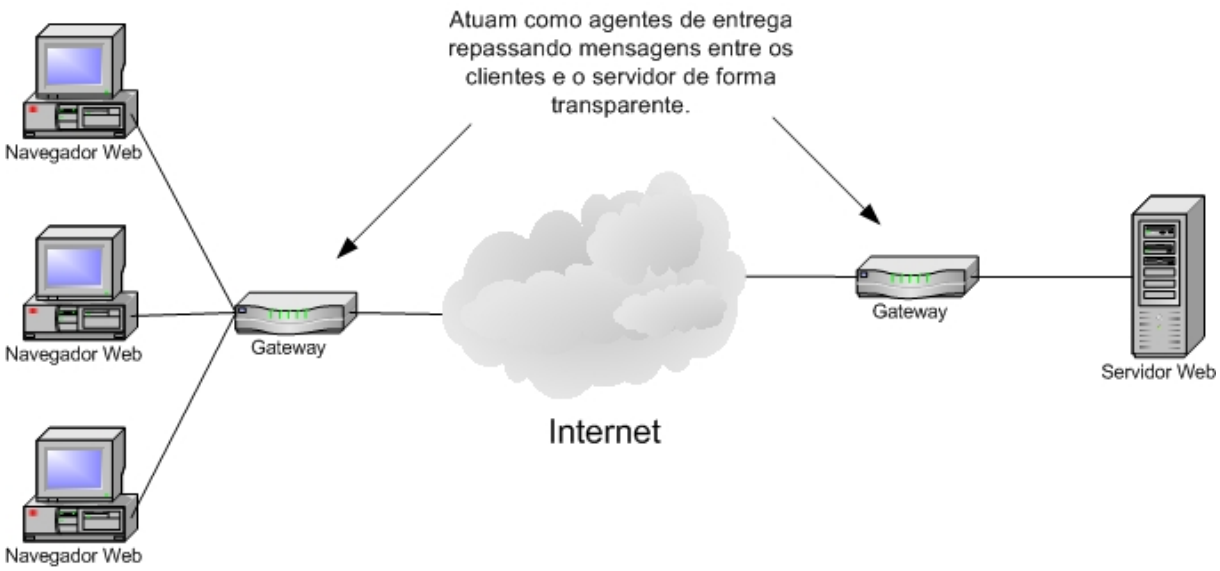


FIGURA 4 - GATEWAYS UTILIZADOS EM UMA COMUNICAÇÃO COM O PROTOCOLO HTTP.

3. *CACHING NA WEB*

3.1 CONCEITO DE *CACHE*

O conceito de *cache* já é utilizado há muito tempo como parte da arquitetura de computadores com o objetivo de melhorar a performance de acesso à memória. Neste contexto o *cache* corresponde a um mecanismo de armazenamento de alta velocidade que é utilizado como um componente de *hardware* intermediário na comunicação entre a CPU (*Central Processing Unit*) e a memória principal do computador.

Um tipo de *cache* que está presente em praticamente todos os computadores modernos corresponde ao *cache* de memória, que é apresentado na figura 5. O *cache* de memória é um componente constituído de um tipo de memória denominada SRAM (*Static Random Access Memory*), cujo tempo de acesso é muito pequeno, porém apresenta um alto custo em relação a outros tipos de memória. O tempo de acesso a uma memória do tipo SRAM é de aproximadamente 10 ns. Existe outro tipo de memória de baixo custo denominada DRAM (*Dynamic Random Access Memory*), que apresenta tempos de acesso em torno de 60 ns. Devido a estas características as arquiteturas de computadores modernas utilizam memórias do tipo DRAM para compor a memória principal, popularmente denominada de RAM (*Random Access Memory*). Quando um usuário adquire um computador com 512 MB (*Megabytes*) de memória, esta quantidade refere-se a memória do tipo DRAM. Se um computador utilizasse apenas memória do tipo DRAM, processadores velozes ficariam ociosos aguardando pela recuperação de dados armazenados nesta memória. Para eliminar este tipo de problema as arquiteturas de computadores modernas utilizam uma pequena quantidade de memória do tipo SRAM que funciona como um repositório intermediário entre o processador e a memória principal constituída de memória do

tipo DRAM. Esta pequena quantidade de memória recebe a denominação de *cache* de memória. Desta forma, quando o processador precisar recuperar um bloco de dados para realizar algum tipo de cálculo ou processamento, inicialmente irá verificar se este bloco de dados está presente no *cache* de memória. Se o bloco desejado estiver no *cache* de memória sua transferência para o processador será realizada de forma rápida. Se o bloco desejado não estiver presente no *cache* de memória, então o bloco será recuperado a partir da memória principal (DRAM). Neste momento uma cópia do bloco recém recuperado será armazenada também no *cache* de memória, permitindo que acessos consecutivos ao mesmo bloco sejam realizados de forma mais rápida a partir do *cache* de memória. Esta estratégia é utilizada, pois estudos mostram que processadores geralmente requisitam um mesmo conjunto de dados repetidamente ao longo do tempo de processamento.

Computadores modernos utilizam dois níveis de *cache* de memória. O primeiro nível corresponde a uma pequena quantidade de memória SRAM que reside no interior dos processadores. O segundo nível de *cache* de memória corresponde a um componente de hardware externo ao processador, que fica instalado na placa-mãe do computador. O segundo nível de *cache* de memória é opcional, porém está presente na maioria dos computadores atualmente em uso.

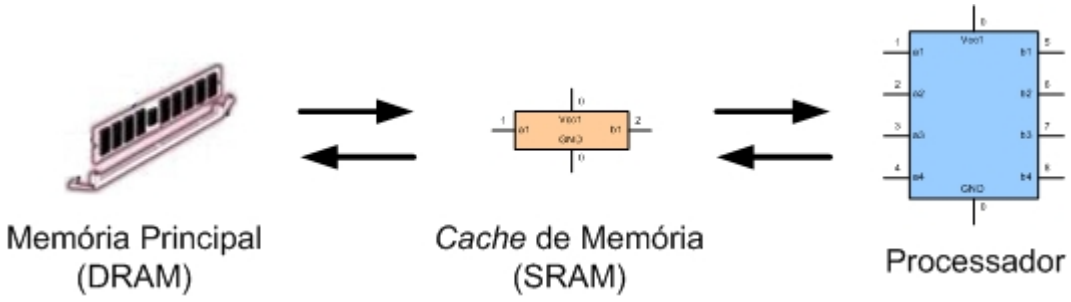


FIGURA 5 - UTILIZAÇÃO DE *CACHE* DE MEMÓRIA NA ARQUITETURA DE COMPUTADORES.

Outro tipo de *cache* também presente em arquiteturas de computadores e sistemas operacionais é denominado *cache* de disco, que pode ser observado na figura

6. O *cache* de disco segue a mesma filosofia do *cache* de memória, porém ao invés de utilizar memória do tipo SRAM o *cache* de disco utiliza memória DRAM como repositório intermediário para a recuperação e atualização de dados que residem em meios de armazenamento persistentes como, por exemplo, disquetes, discos rígidos, CDs (*Compact Disks*), fitas DAT (*Digital Audio Tape*), etc. O tempo de acesso a um conjunto de dados armazenados em memória, mesmo que seja do tipo DRAM, chega a ser de 10^5 a 10^6 vezes menor que o tempo de acesso a um conjunto de dados armazenados em um dos meios de armazenamento persistentes citados anteriormente.

FIGURA 6 - UTILIZAÇÃO DE *CACHE* DE DISCO NA ARQUITETURA DE COMPUTADORES.

Quando um dado requisitado pelo processador é encontrado no *cache*, seja de disco ou de memória, considera-se que houve um **acerto de *cache***, do inglês "*Cache Hit*". Ao contrário, quando é necessário recuperar um dado a partir da sua localização original considera-se que houve uma **falha de *cache***, do inglês "*Cache Miss*". A **taxa de acerto**, do inglês "*hit rate*" é uma das métricas utilizadas para medir a eficácia e eficiência de uma estratégia que envolve a utilização de *cache*.

A partir da análise do funcionamento de mecanismos de *cache*, fica evidente que o repositório do mecanismo de *cache* tende a ficar cheio tão logo entre em funcionamento. Desde o princípio da utilização do conceito de *cache* em arquiteturas de computadores e sistemas operacionais busca-se a estratégia mais eficiente para determinar quais informações devem permanecer no *cache* e quais devem ser descartadas quando o limite de armazenamento é atingido. Estudos realizados nesta área resultaram na criação de diversas estratégias para otimizar a utilização do *cache*.

Uma das primeiras e mais eficientes estratégias criadas para este fim foi o LRU (*Least Recently Used*). Os princípios de funcionamento desta estratégia, aplicadas ao problema de substituição em *cache* na Internet, são apresentados no próximo capítulo.

3.2 INFRA-ESTRUTURA DE *CACHE* NA *WEB*

Com o surgimento do serviço WWW (*World Wide Web*) em 1990 a Internet tornou-se um fenômeno, apresentando uma espantosa taxa de crescimento. Durante o início da década de 1990 a *web* apresentou um taxa de crescimento de 100% ao ano, porém nos anos 1995 e 1996, quando ocorreu a internacionalização da Internet, foram registradas taxas de crescimento próximas de 1000% ao ano [COF99]. Ao final de 1997 a taxa de crescimento voltou aos patamares anteriores com 100% ao ano. A partir de 1998 a taxa de crescimento da *web* apresentou queda até que se estabilizou em 2001, chegando a apresentar queda em 2002 [ONE03]. Em contrapartida, o número de usuários na Internet continua acrescendo em todo o mundo. Um estudo realizado pelo governo dos Estudos Unidos em 2002 mostrou que 67% da população norte americana, correspondendo a 174 milhões de pessoas, são usuárias da Internet. O mesmo estudo demonstrou que a adesão de novos usuários cresce numa taxa de dois milhões ao mês [USDC02]. Este mesmo cenário pode ser observado fora dos Estados Unidos em maiores proporções. Estima-se que atualmente a Internet possui aproximadamente um bilhão de usuários em todo o mundo. A gigantesca expansão da Internet trouxe vários desafios tecnológicos a serem enfrentados, principalmente no que diz respeito ao volume do tráfego de informações em toda a rede. Este ponto pode ser considerado como um dos mais críticos, pois impacta diretamente no tempo de resposta para os usuários da *web*.

A *web* pode ser considerada como um enorme sistema de informação que é composto de milhões de objetos compartilhados em servidores. Estes objetos correspondem a documentos HTML, arquivos de imagem, sons, etc. O acesso a estes

objetos é disputado por milhões de usuários simultaneamente. Sem a utilização de uma estratégia para aliviar o número de requisições aos servidores *web* mais populares, logo estes ficariam sobrecarregados e não conseguiriam atender a requisições de forma eficiente. Desde o surgimento do serviço WWW o conceito de *cache* passou a ser largamente utilizado na Internet para minimizar o tráfego em toda rede e para reduzir a carga em servidores *web*. Fazendo uma analogia ao conceito de *cache* aplicado ao problema de gerenciamento de memória, os blocos de memória correspondem aos objetos armazenados em servidores *web*, o processador corresponde aos usuários que requisitam objetos utilizando navegadores *web*, e a memória principal corresponde aos repositórios de armazenamento de documentos nos servidores *web*.

A estratégia de *cache* aplicada ao problema de tráfego na *web* tem como objetivo armazenar os objetos mais requisitados em repositórios próximos aos clientes. Servidores *proxy* exercem o papel de repositório temporário dentro da arquitetura de *cache* na *web*. Desta forma, os servidores *proxy* atuam como componentes intermediários na "conversação" entre clientes usando navegadores e servidores *web* onde objetos desejados estão armazenados. A figura 7 apresenta o comportamento de um servidor *proxy* quando este recebe uma requisição a partir de um cliente para o "Objeto #X", que não está em seu repositório. Neste caso a requisição é encaminhada para o servidor *web* original.

A figura 8 apresenta o comportamento de um servidor *proxy* quando este recebe uma requisição a partir de um cliente para o "Objeto #C". Neste caso o servidor *proxy* retorna a cópia do objeto solicitado que está presente em seu repositório.

Em uma arquitetura de *cache* na *web*, todas as requisições de objetos realizadas por um cliente serão inicialmente encaminhadas a um servidor *proxy*. O servidor *proxy* verifica se possui uma cópia válida do objeto requisitado pelo cliente em seu repositório de armazenamento. Se o servidor *proxy* tiver sucesso, a cópia do objeto requisitado é retornada ao cliente. Neste caso não é necessário encaminhar a requisição através da Internet até o servidor *web* que armazena o objeto original. Se o

servidor *proxy* não possuir uma cópia válida do objeto requisitado, este então realiza uma requisição ao servidor remoto. Neste caso o servidor *proxy* armazena uma cópia do objeto recém recuperado do servidor remoto antes de retorná-la ao cliente que realizou a requisição.

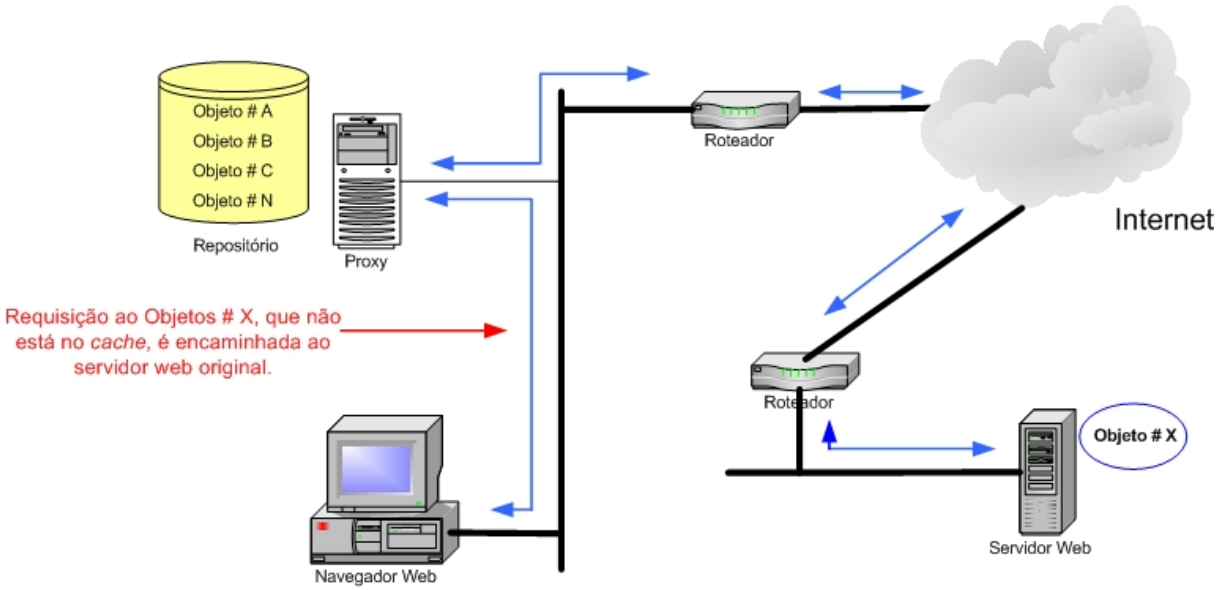


FIGURA 7 - REQUISIÇÃO ENCAMINHADA AO SERVIDOR *WEB* ATRAVÉS DO SERVIDOR *PROXY*.

A utilização de *cache* na *web* apresenta várias vantagens além de reduzir o tráfego na Internet. Uma delas corresponde a diminuição do tempo decorrido entre o envio da requisição de um cliente e a recepção do objeto recuperado. Este tempo é comumente denominado na bibliografia como **latência**. A latência é reduzida quando se utiliza *cache*, pois os objetos mais freqüentemente requisitados por uma comunidade de clientes apresentam maior probabilidade de serem encontrados no servidor *proxy* compartilhado. Geralmente os servidores *proxy* são dispostos em pontos estratégicos das redes de computadores, sendo que muitas vezes estão disponíveis nas mesmas redes locais dos clientes, tornando muito veloz o acesso aos objetos requisitados.

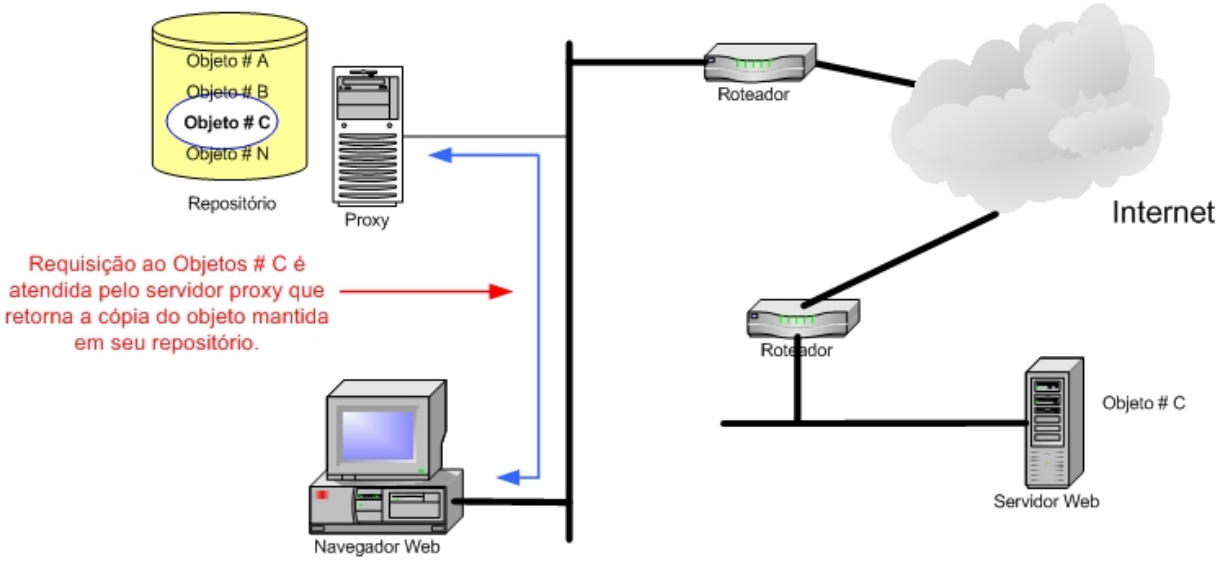


FIGURA 8 - REQUISIÇÃO ATENDIDA PELO SERVIDOR *PROXY*.

Se por algum motivo um servidor *web* ficar temporariamente indisponível a utilização de *cache* permitirá que os clientes continuem tendo acesso aos objetos deste servidor através de cópias armazenadas em servidores *proxy*.

3.2.1 Arquiteturas de *Cache* na *Web*

Um dos principais aspectos a ser considerado em uma arquitetura de *cache* na *web* diz respeito a quantidade e a distribuição dos servidores *proxy* para atender a uma comunidade de clientes. Não é recomendada a utilização de um único servidor *proxy* em uma rede de computadores, pois toda a comunidade de clientes ficaria impossibilitada de utilizar o serviço WWW caso este servidor *proxy* fique indisponível.

Uma arquitetura de *cache* deve permitir que dois ou mais servidores *proxy* possam trabalhar de forma cooperativa para atender com eficiência a uma comunidade de clientes. A partir deste princípio, duas estratégias diferentes são propostas: **Arquitetura de *Cache* Hierárquica** e **Arquitetura de *Cache* Distribuída**.

A arquitetura Hierárquica propõe a existência de quatro níveis hierárquicos de *cache* dispostos na rede. Cada nível de *cache* colabora com o seu nível

imediatamente superior para tentar atender a requisições de clientes da maneira mais eficiente possível, conforme apresenta a figura 9. Segundo [WAN99], os quatro níveis propostos são:

- ***Bottom***: O primeiro nível hierárquico é formado pelo recurso de *cache* nativo dos navegadores utilizados por clientes. Este é o nível mais simples onde não estão presentes servidores *proxy* dedicados à tarefa de *cache*. Praticamente todos os navegadores atualmente existentes possuem algum tipo de recurso de *cache*. Estes recursos geralmente são rudimentares e se restringem ao armazenamento e pré-recuperação de objetos que estão sendo referenciados por documentos HTML que estiverem sendo apresentado a clientes num dado momento.
- ***Institutional***: Este nível de *cache* é utilizado quando uma requisição de um cliente não puder ser atendida pelo primeiro nível de *cache*. Este nível é formado por um ou mais servidores *proxy* que atendem a requisições de uma comunidade de clientes, geralmente vinculados a uma organização ou provedor de acesso. Quando uma requisição não puder ser atendida neste nível é encaminhada ao próximo nível da hierarquia.
- ***Regional***: Este nível também é formado por servidores *proxy*, porém clientes não realizam requisições diretamente para este nível hierárquico. Os servidores *proxy* deste nível recebem requisições a partir de servidores *proxy* institucionais. Desta forma, este nível hierárquico atende a comunidades distintas de clientes que estão vinculadas aos seus respectivos servidores *proxy* institucionais.
- ***National***: Este é o nível mais alto na hierarquia de *cache*. Quando uma requisição não puder ser atendida por nenhum dos níveis anteriores, esta será encaminhada a um servidor *proxy* nacional. Se o

servidor *proxy* nacional não possuir o objeto requisitado em seu repositório de armazenamento, então o servidor *proxy* nacional realizará a requisição do objeto diretamente ao servidor *web* remoto que contém o objeto. Neste caso, cópias do objeto recuperado serão replicadas em todos os níveis inferiores até que o cliente seja atendido.

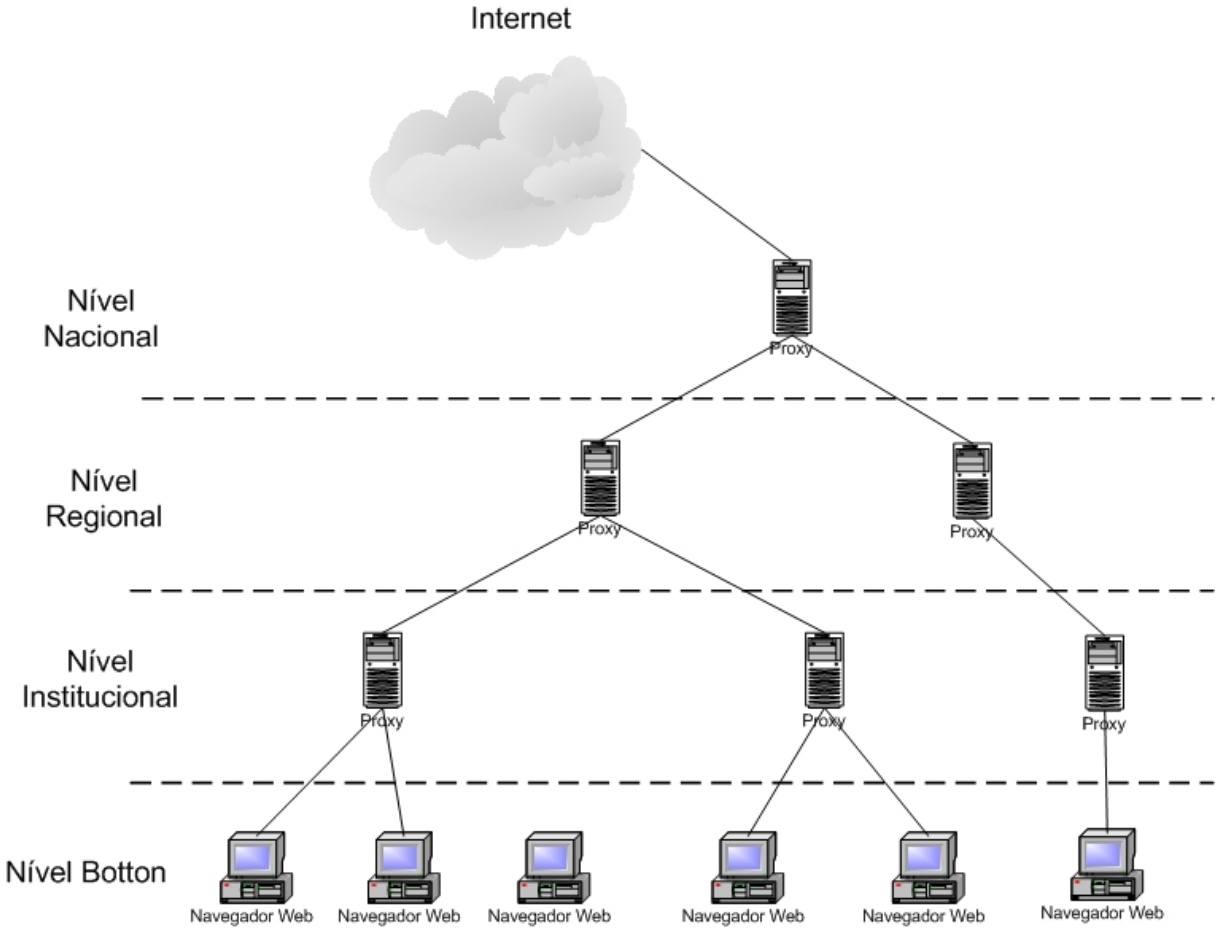


FIGURA 9 - ARQUITETURA DE *CACHE* HIERÁRQUICA NA *WEB*.

A arquitetura hierárquica apresenta algumas vulnerabilidades, principalmente em relação a sobrecarga dos servidores *proxy* nacionais, pois a responsabilidade de requisitar objetos aos servidores *web* de origem fica centralizada neste nível. Outro ponto negativo está relacionado ao fato de que várias cópias de um

mesmo objeto ficam armazenadas em todos os níveis da hierarquia, gerando uma redundância muitas vezes desnecessária.

A Arquitetura de *Cache* Distribuída utiliza uma abordagem diferente, pois considera apenas um nível de *cache* correspondente ao *cache* institucional da arquitetura hierárquica. A diferença crucial reside no fato de que os servidores *proxy* são dispostos de forma totalmente distribuída, colaborando entre si para atender a requisições de clientes. A arquitetura distribuída propõe a utilização de uma camada superior aos servidores *proxy* institucionais para troca de meta-informação referente aos objetos armazenados, conforme pode ser observado na figura 10. Estes componentes funcionam como diretórios que facilitam a colaboração entre os diversos servidores *proxy* distribuídos na rede. Utilizando esta abordagem, a maior parte do tráfego referente à requisição de objetos fica restrita às redes locais onde se encontram as comunidades de clientes com seus respectivos servidores *proxy*.

3.2.2 Características de um Mecanismo de *Cache*

Um mecanismo de *cache* deve apresentar algumas características importantes para que possa ser realmente eficaz na obtenção dos objetivos a que se propõe. Os mecanismos de *cache* devem possuir robustez, rapidez, eficiência, estabilidade, transparência, escalabilidade, balanceamento de carga e capacidade de lidar com heterogeneidade [WAN99].

Um mecanismo de *cache* eficiente deve apresentar características e recursos que sejam capazes de solucionar os problemas que envolvem *cache* na *web*. Se o correto tratamento não for dado a cada um destes problemas, o mecanismo de *cache* pode se tornar um “gargalo” na rede. Isto pode fazer com que a recuperação de um objeto a partir de sua localização original seja mais rápida do que através da utilização do mecanismo de *cache*, ou na pior situação possível, os clientes podem receber objetos desatualizados.

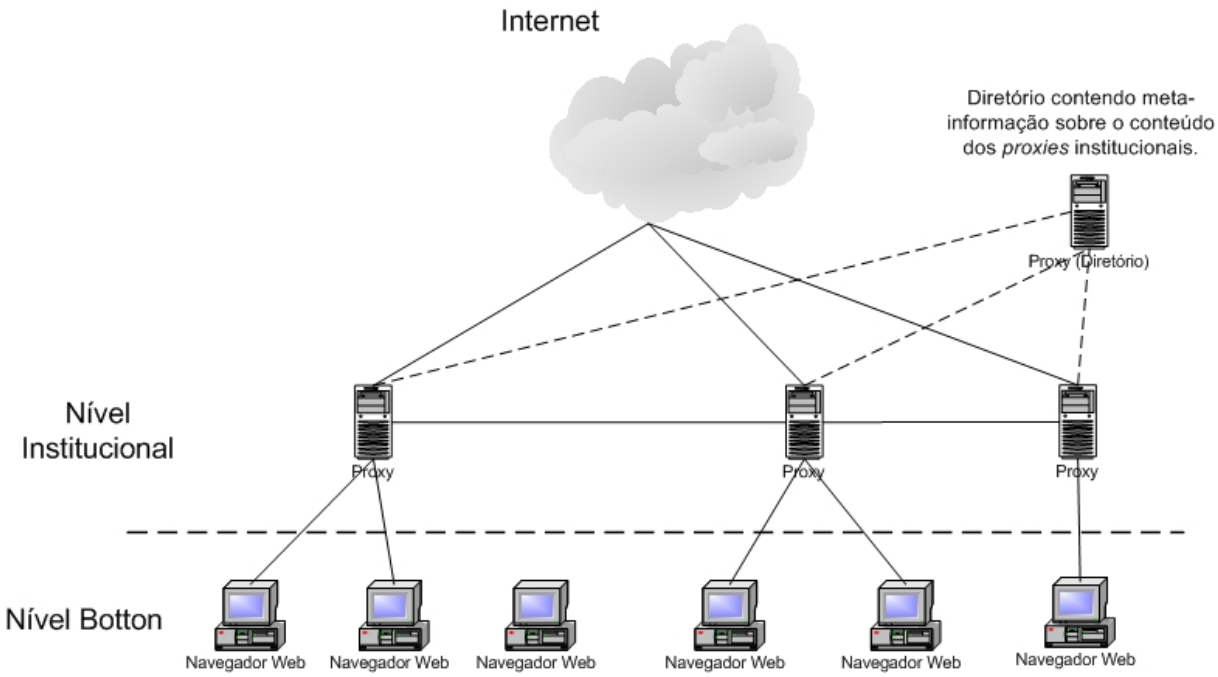


FIGURA 10 - ARQUITETURA DE *CACHE* DISTRIBUÍDA NA *WEB*.

Os principais problemas a serem solucionados por um mecanismo de *cache* são:

- **Resolução de *Cache***: Este problema está presente principalmente em uma arquitetura de *cache* distribuída onde é necessário estabelecer uma estratégia que permita descobrir em qual servidor *proxy* um objeto requisitado por um cliente pode ser encontrado. Em arquiteturas distribuídas os servidores *proxy* trabalham de forma cooperativa para atender a requisições de clientes. Desta forma, quando um servidor *proxy* recebe uma requisição de um objeto que não está em seu repositório, antes de recuperar o objeto a partir de seu servidor *web* original, o servidor *proxy* pode interagir com outros servidores *proxy* para verificar se estes possuem uma cópia do objeto solicitado. Um mecanismo de *cache* eficiente deve definir quantas interações entre os servidores *proxy* poderão ser realizadas antes que o servidor *proxy* que atende o cliente decida recuperar o objeto a partir do servidor *web* original. Se este critério não for eficiente a latência

para recuperar um objeto usando o mecanismo de *cache* poderá ser maior do que a latência para requisitar o objeto diretamente do servidor *web* de origem.

- **Pré-Recuperação de Objetos**: A pré-recuperação é uma estratégia que permite a obtenção de maiores taxas de acerto (*cache hits*) em mecanismos de *cache*. Esta estratégia consiste em prever e recuperar de forma antecipada os próximos objetos que serão requisitados por um cliente. Isto é possível através da análise dos objetos que estão sendo requisitados pelo cliente num dado momento. Quando um cliente requisita um documento HTML, existe uma grande probabilidade de que os próximos documentos HTML, ou objetos a serem requisitados, estejam referenciados pelo primeiro documento através de *links*. Neste caso o mecanismo de *cache* pode antecipar a recuperação destes objetos antes da efetiva requisição por parte do cliente. Quando este o fizer, os objetos já estarão no repositório do mecanismo de *cache* registrando um **acerto de *cache* (***cache hit***)**. Estudos mostram que esta técnica colabora para o aumento da taxa de acerto, porém aumentando consideravelmente o tráfego na rede, pois corre-se o risco de recuperar objetos que não necessariamente serão efetivamente requisitados pelo cliente. A pré-recuperação de objetos pode ocorrer entre um navegador e um servidor *proxy*, ou entre um servidor *proxy* e um servidor *web*.
- **Substituição em *Cache***: O problema de substituição em *cache* refere-se ao fato de que todo servidor *proxy* possui uma capacidade máxima de armazenamento em seu repositório. Quando o repositório atinge seu limite o mecanismo de *cache* deve decidir quais objetos deverão ser descartados para dar lugar a novos objetos recuperados a partir de servidores *web* para atender a requisições recentes de clientes.

Existem vários estudos sobre este tema que levaram à proposição de diversas estratégias de substituição. A apresentação da estratégia de substituição LSR-VM (*Least Semantically Related – Vector Model*), que adota uma nova abordagem baseada em semântica, é o tema principal desta dissertação. O próximo capítulo apresenta o estado da arte em relação às estratégias de substituição em *cache*.

- **Coerência do *Cache***: O problema de coerência do *cache* diz respeito ao sincronismo entre as cópias de objetos armazenados nos repositórios dos servidores *proxy* e os objetos originais que residem nos servidores *web*. Um mecanismo de *cache* eficaz deve dar o tratamento adequado a este problema estabelecendo um critério para atualização das cópias de objetos armazenadas em seu repositório. Os cabeçalhos de mensagens do protocolo HTTP possuem um conjunto de campos que serve como recurso para que os servidores *proxy* possam implementar estratégias de atualização das cópias de objetos que residem em seus repositórios. Estes campos transformam o método "*GET*" em uma recuperação condicional, cujo comportamento é definido pelo conjunto de valores dos campos presentes na mensagem de requisição HTTP. Os principais campos que influenciam o método "*GET*" são:
    - "*If-Modified-Since*": O valor deste campo corresponde a uma data que é enviada como parte de uma mensagem de requisição para um servidor *web*. Se o objeto requisitado não tiver sofrido alterações desde a data informada como valor deste campo, o objeto não será retornado pelo servidor *web*. Ao invés disso o servidor *web* retornará uma mensagem de resposta com o código de *status* 304 (Não Modificado), sem o objeto requisitado.

- "*If-Unmodified-Since*": O valor deste campo também representa uma data que é enviada como parte de uma mensagem de requisição para um servidor *web*. Se o objeto requisitado não tiver sofrido alterações desde a data informada como valor deste campo, o servidor *web* retorna o objeto requisitado ignorando o campo "*If-Unmodified-Since*". Caso contrário, se o objeto tiver sido modificado o servidor *web* retornará uma mensagem de resposta com o código de *status* 402 (Pré-condição Falhou), sem o objeto requisitado.

Outro campo que não torna o método "*GET*" condicional, mas que também é utilizado, principalmente por navegadores *web* para implementar a estratégia de atualização, é o campo "*Pragma=no-cache*". Este campo faz com que o servidor *proxy* recupere o objeto a partir do servidor *web* original mesmo que exista uma cópia presente em seu repositório.

- ***Caching* de Objetos Dinâmicos**: Um dos principais problemas que apresenta impacto sobre a utilização do conceito de *cache* na *web* é o fato de que o tráfego referente a objetos gerados dinamicamente para atender a requisições de clientes, tornou-se dominante em relação ao tráfego de objetos estáticos. Estudos realizados por [ZHA03] apresentaram resultados que comprovam esta característica. Como parte deste estudo, foi realizado um experimento onde o tráfego de um dia gerado na rede da *Wayne State University* (WSU) foi coletado utilizando-se uma ferramenta para análise de conteúdo do tráfego de uma rede. Os resultados obtidos a partir da análise dos dados coletados mostraram que 96,7% do tráfego correspondia a conteúdo não passível de *cache* [ZHA03]. Apesar de existirem algumas estratégias propostas para *cache* de conteúdo dinâmico, nenhuma se

mostrou eficaz até o momento. O capítulo 7 desta dissertação apresenta os resultados obtidos durante os experimentos realizados com a estratégia de substituição LSR-VM em relação a objetos que não são passíveis de *cache*.

3.3 *PROXY SQUID*

O *Squid* é um dos servidores *proxy* mais utilizados atualmente na Internet e por organizações em geral. O *proxy Squid* é distribuído sob a modalidade de software livre através da GNU (*General Public License*). Este foi um dos fatos que contribuiu para a grande disseminação do *Squid*. Atualmente existem versões do *Squid* em várias plataformas de sistemas operacionais como, por exemplo, Linux, FreeBSD e *Windows*. O *Squid* foi desenvolvido por Duane Wessels no *National Laboratory for Applied Network Research* na Universidade da Califórnia, Estados Unidos da América. A versão mais atual do *Squid* implementa um mecanismo de *cache* completo, com suporte a vários recursos avançados como, por exemplo:

- **Controle de Acesso**: Permite o gerenciamento do acesso aos serviços do mecanismo de *cache* através de diretivas. Este recurso é bastante flexível possibilitando a controle por faixa de endereçamento IP ou por perfil de usuário. O *Squid* realiza a autenticação de usuários através de várias maneiras que podem ser configuradas, inclusive suportando integração com o serviço LDAP (*Lightweight Directory Access Protocol)*. O serviço LDAP permite o acesso a diretórios de informações, que entre outras coisas, podem ser usados como mecanismos unificados de autenticação. O *Squid* permite ainda criação de regras de acesso aplicadas com base em períodos estabelecidos. Por exemplo: Usuários da sub-rede 192.168.1.0/24 somente podem acessar a Internet através do *Squid* no período das

12:00 até as 13:30 horas.

- **Arquiteturas de *Cache***: O *Squid* suporta a construção de arquiteturas hierárquicas de *cache* ou arquiteturas distribuídas de *cache* através da colaboração entre dois ou mais servidores.
- ***Transparent Caching***: Este recurso é responsável por realizar *cache* de requisições de usuários sem que estes necessitem configurar seus navegadores para explicitamente utilizar um servidor *proxy*. Isto elimina a necessidade de configurações nas estações de trabalho da rede. O *Squid* captura os pacotes IP referentes a requisições HTTP, passando a realizar *cache*.
- **Estratégia de Substituição em *Cache***: Através do arquivo de configurações do *Squid* é possível definir a estratégia de substituição a ser usada. As principais estratégias são suportadas como, por exemplo: LRU (*Least Recently Used*), LFU (*Least Frequently Used*) e SIZE.

Quando está em operação o *Squid* gera um conjunto de arquivos de *log* que pode ser utilizado para diversos fins, como por exemplo, na avaliação da performance do mecanismo de *cache*, no ajuste da configuração mais adequada do *Squid* dentro das características da rede onde está instalado e principalmente para analisar e avaliar o perfil da comunidade de usuários que utiliza o *Squid*. Os principais arquivos de *log* gerados pelo *Squid* são:

- "**cache.log**": Este é o *log* onde o *Squid* registra mensagens em geral, como mensagens de *debug*, mensagens de erro ou mensagens informativas.
- "**store.log**": Este *log* contém informações sobre os objetos que atualmente estão armazenados no repositório do mecanismo de *cache*.

- "**access.log**": Este *log* contém informações a respeito das requisições a objetos submetidas ao mecanismo de *cache* por usuários.

Existe uma quantidade grande de programas de terceiros que são utilizados para análise dos *logs* do *Squid*, apresentando resultados diversos a respeito da performance do mecanismo de *cache* e do perfil da comunidade de usuários. Estes programas realizam suas análises essencialmente utilizando o arquivo "access.log".

Este arquivo foi utilizado durante os experimentos para validação da estratégia de substituição LSR-VM (*Least Semantically Related – Vector Model*) com o objetivo de simular uma seqüência de acessos realizados por uma comunidade de usuários de um mecanismo de *cache*. O arquivo "access.log" possui apenas um tipo de registro que representa um acesso realizado ao mecanismo de *cache* por um cliente. Este registro é composto de dez campos conforme apresenta a figura 11:

- ***time***: Este campo contém a data e hora do acesso, apresentadas no formato UTC UNIX *timestamp* em milisegundos. UTC (*Coordinated Universal Time*) é uma escala de tempo que representa os fusos-horários a partir do horário no meridiano de Greenwich. Unix *time stamp* é uma forma de contar o tempo decorrido em segundos a partir da 00:00 hora do dia 01 de janeiro de 1970. Esta métrica é usada em sistemas operacionais como o Unix e o Linux. O conteúdo do campo "*time"* é apresentado em milisegundos.
- ***duration***: Representa a quantidade de tempo em milisegundos que o mecanismo de *cache* ficou ocupado processando a requisição de acesso.
- ***client address***: Este campo contém o endereço IP do cliente que realizou a requisição.
- ***result codes***: Este campo apresenta o resultado da transação realizada pelo mecanismo de *cache*. Este campo é composto de duas

informações que são separadas por uma barra (/). A primeira informação é denominada "*Squid result code*" e refere-se ao resultado da requisição realizada ao *cache*. Esta informação pode apresentar o tipo da requisição, como esta foi satisfeita, ou como esta falhou. A segunda informação que compõe este campo corresponde ao código de *status* da mensagem de resposta HTTP.

- ***bytes***: Representa o tamanho em *bytes* do objeto retornado ao cliente que realizou a requisição.
- ***request method***: Corresponde ao método de requisição do protocolo HTTP solicitado na requisição realizada pelo cliente.
- ***URL (Uniform Resource Locator)***: A URL corresponde ao endereço global do objeto na *web*. Parte deste endereço identifica o protocolo que será utilizado para recuperar o objeto.
- ***rfc931***: Campo geralmente desativado em versões atuais do *Squid*. O conteúdo deste campo geralmente é definido como "-".
- ***hierarchy code***: Quando o servidor *proxy Squid* está configurado em uma arquitetura hierárquica ou distribuída, este campo passa a apresentar informações pertinentes a esta arquitetura.
- ***type***: Este campo corresponde ao campo *content-type* que aparece no cabeçalho da mensagem de resposta do protocolo HTTP.

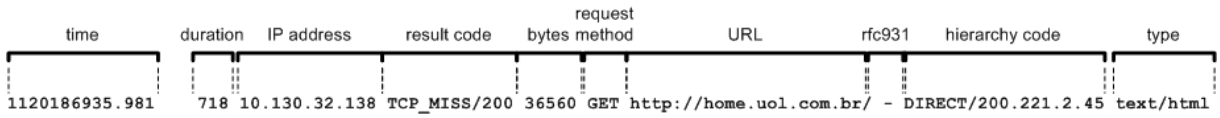


FIGURA 11 - REGISTRO DE *LOG* DO *PROXY SQUID*.

4. ESTRATÉGIAS DE SUBSTITUIÇÃO

4.1 ESTRATÉGIAS DE SUBSTITUIÇÃO TRADICIONAIS

As estratégias de substituição tradicionais levam em consideração as características físicas e operacionais dos objetos como critério para seleção de quais objetos devem ser descartados para liberar espaço necessário para acomodar um novo objeto recém chegado ao *cache*. Destacam-se como características mais utilizadas por estas estratégias o tamanho dos objetos, freqüência de acesso, custo para recuperação na *web* e o tempo de permanência no *cache*.

As estratégias de substituição SIZE, *Least Recently Used* (LRU) e *Least Frequently Used* (LFU) foram algumas das primeiras estratégias de substituição a serem utilizadas em mecanismos de *cache*. Desde então surgiram outras estratégias que exploram variações ou apresentam características híbridas em relação às estratégias clássicas. Existe outra categoria de estratégias que se baseia no custo dos objetos como critério para substituição. Este custo geralmente é calculado através de uma função de valor que pode contemplar diferentes fatores tais como: tempo desde o último acesso, momento de entrada do objeto no mecanismo de *cache*, custo do tempo de transferência do objeto, tempo de expiração do objeto, etc. [WAN99]. As principais estratégias de substituição que se baseiam em custo são: *GreedyDual-Size* (GD-Size), *Hybrid*, *Lowest Relative Value*, *Least Normalized Cost Replacement* (LCN-R), *Size-Ajusted* LRU (SLRU), *Server-assisted scheme e Hierarquical GreedyDual*. Todas estas estratégias utilizam as propriedades operacionais dos objetos como critério para substituição. A seguir são apresentadas as características das principais estratégias de substituição em *cache*:

- **SIZE**: A estratégia SIZE baseia-se no tamanho dos objetos como critério para a substituição. Quando um novo objeto deve ser inserido, e o mecanismo de *cache* está cheio, esta estratégia procura o maior objeto existente no *cache* e o remove para criar espaço para o novo objeto. Se mesmo assim não houver espaço para o novo objeto a estratégia continua removendo os maiores objetos até que seja possível acomodar o novo objeto [AGG99].
- ***Least Frequently Used* (LFU)**: A estratégia LFU baseia-se na freqüência de utilização dos objetos como critério para a substituição. Se houver necessidade de criação de espaço para um novo objeto o LFU descartará os objetos com menor freqüência de uso. Um dos principais problemas deve-se ao fato de que o LFU não faz distinção entre objetos com alta freqüência de acesso no passado, de objetos com alta freqüência de acesso no presente. Desta forma, objetos que foram muito acessados no passado, mas que não serão mais acessados no futuro, tendem a ficar no *cache* [AGG99].
- ***Least Recently Used* (LRU)**: A estratégia LRU baseia-se na recenticidade de acesso aos objetos como critério para a substituição. O LFU preserva no *cache* os objetos mais recentemente acessados. O LRU pressupõe que os objetos mais recentemente utilizados possuem maior probabilidade de serem requisitados novamente. Este princípio é chamado de localidade temporal [AGG99].
- ***GreedyDual-Size* (GD-*Size*)**: A estratégia GD-*Size* baseia-se na recenticidade, tamanho e custo de acesso aos objetos como critério para a substituição. Por utilizar todas estas características esta pode ser considerada uma **estratégia híbrida**. A estratégia GD-*Size* associa um custo aos objetos armazenados no *cache* e descarta o objeto com menor custo e tamanho.

- ***Hybrid***: A estratégia *Hybrid* associa uma função de valor a cada objeto no *cache*. Os parâmetros utilizados para atribuir o valor a cada objeto são: tempo de conexão ao servidor onde o objeto originalmente reside, largura de banda da conexão, freqüência de uso e tamanho do objeto A estratégia descarta o objeto com menor valor quando há necessidade de liberação de espaço.
- ***Lowest Relative Value* (LRV)**: Esta estratégia também associa uma função de valor a cada objeto no *cache*, porém levando em conta a distribuição dos tempos entre acessos a um mesmo objeto. O LRV impõe um alto custo computacional, o que inviabiliza seu uso na prática [RIZ00].
- ***Least Normalized Cost Replacement* (LNC-R)**: Esta estratégia associa uma função racional a cada objeto do *cache*, baseada na freqüência de acesso, no custo do tempo de transferência e no tamanho do objeto.
- ***Size-Ajusted LRU* (SLRU)**: Esta estratégia ordena os objetos do *cache* pela razão entre custo e tamanho, e descarta objetos com a maior razão custo e tamanho.
- ***Server-Assisted Scheme***: Esta estratégia calcula o valor dos objetos no *cache* através de uma função que considera o custo de recuperação do objeto em seu local original na Internet, tamanho, o tempo da próxima requisição e o custo de armazenamento no *cache* durante o período entre requisições de um mesmo objeto. O objeto com o menor valor é descartado.
- ***Hierarquical GreedyDual* (*Hierarquical* GD)**: Esta estratégia trabalha como o *GreedyDual-Size*, porém de forma cooperativa através de uma hierarquia de mecanismos de *cache*.

4.2 ESTRATÉGIAS DE SUBSTITUIÇÃO BASEADAS EM SEMÂNTICA

As estratégias de substituição baseadas em semântica levam em consideração o conteúdo dos objetos, e por conseguinte a informação semântica que estes trazem consigo, como critério durante a seleção dos objetos que serão descartados quando houver necessidade de criar espaço no mecanismo de *cache* para acomodar um novo objeto.

Durante as pesquisas que resultaram nesta dissertação foram identificados dois grupos de estudos voltados à utilização de semântica aplicada ao problema de substituição em mecanismos de *cache*. Um dos grupos está localizado na Pontifícia Universidade Católica do Paraná (PUCPR), Brasil, sendo responsável pela criação da estratégia *Least Semantically Related* (LSR) [CAL01]. Em paralelo aos estudos que resultaram nesta dissertação, a estratégia LSR deu origem a outro trabalho que propõe uma variação denominada *Least Semantically Related with Access History* (LSR/H) [CAL02]. A estratégia LSR/H, além de considerar a semântica dos objetos em um mecanismo de *cache* também utiliza o histórico de assuntos de interesse ao longo do tempo em um mecanismo de *cache*.

Outro grupo de estudos está localizado no *Institut National des Sciences Appliquées de Lyon* (INSA), Lyon, França. Este grupo propôs uma arquitetura de *proxy* colaborativa, que se baseia em tópicos ou assuntos que refletem o interesse da comunidade de usuários que utilizam o mecanismo de *cache*. Os assuntos de maior interesse em um dado momento são classificados como "**quentes**", do inglês ***hot***. Desta forma, a "**temperatura**" de um objeto representa sua relevância em relação aos assuntos de interesse da comunidade [BRU02].

4.2.1 *Least Semantically Related* (LSR)

A estratégia LSR baseia-se no princípio de que usuários apresentam um comportamento característico quando estão acessando a *web*. Este comportamento

prevê que, durante intervalos de tempo variáveis, um determinado usuário realiza uma seqüência de acessos a documentos HTML que estão relacionados a um assunto de interesse em particular. A partir desta premissa é possível supor que os documentos HTML acessados nestes intervalos compartilham algum tipo de semântica. Desta forma, quando há necessidade de liberação de espaço em um mecanismo de *cache* para acomodar um objeto recém chegado, a estratégia LSR descarta objetos que possuem menor afinidade semântica com o novo objeto. É possível observar que a estratégia LSR considera apenas a semântica do novo objeto durante a seleção de objetos a serem descartados. Isto limita o critério de descarte a um único assunto de interesse.

Para viabilizar seu funcionamento a estratégia LSR considera que cada objeto a ser armazenado no *cache* já vem pré-classificado dentro de uma taxonomia semântica. Esta taxonomia é representada através de uma árvore semântica que define uma hierarquia de assuntos. Desta forma, cada nó da árvore corresponde a um conjunto de assuntos, que por sua vez possui subconjuntos de assuntos (nós filho). Cada Nó da árvore representa um conjunto de assuntos que contém $m >= 0$ subconjuntos (sub-árvores), disjuntos, $S_1, S_2, ..., S_m$ e assim sucessivamente, e $n >= 0$ objetos de informação [CAL01].

A figura 12 mostra um exemplo da árvore semântica utilizada pelo LSR. Pode-se observar que o assunto *Root* possui os sub-assuntos **esportes** e **informática**, e que o assunto **esporte**, por sua vez, possui os sub-assuntos **futebol, automobilismo** e **etc**. Já o assunto **informática** possui os sub-assuntos **desenvolvimento de software** e **etc**. Os objetos aparecem nas terminações (folhas) da árvore ou em nós intermediários e recebem um número de identificação. O objeto #15, por exemplo, possui semântica *root.esportes.automobilismo* e tem um tamanho de 1.250 *bytes*. Quando há necessidade de descartar objetos para criar espaço para um novo objeto, o LSR elimina os objetos que pertencem aos conjuntos mais distantes em relação ao conjunto onde o novo objeto será acomodado, até que o espaço necessário esteja disponível. Esta distância pode ser obtida percorrendo-se a árvore em largura e profundidade.

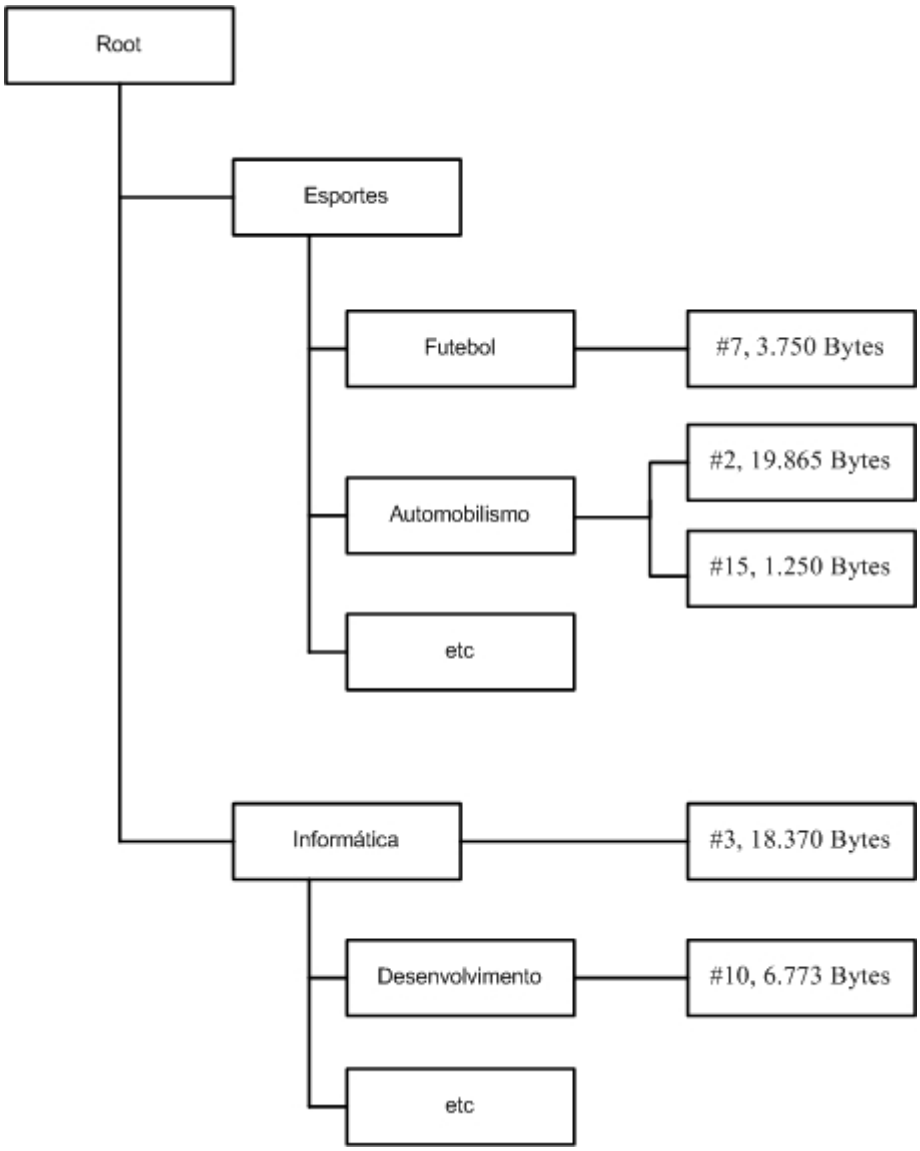


FIGURA 12 - ÁRVORE SEMÂNTICA UTILIZADA PELO LSR.

A estratégia LSR pressupõe que cada objeto, além de sua própria informação, carrega uma seqüência de assuntos e sub-assuntos predefinidos, que corresponde à semântica da árvore. Isto caracteriza um problema, pois esta situação não se verifica na prática. Outro problema está na presença de nós **etc** que aparecem repetidas vezes. Estes nós são utilizados para acomodar objetos cujos assuntos não tenham sido previamente criados na árvore semântica. A partir da árvore apresentada na figura 12, se houvesse a necessidade de se armazenar uma página sobre um jogador

de basquete, por exemplo, este objeto teria de ser vinculado ao nó **etc**, que faz parte do assunto **esportes**. Neste caso a semântica do novo objeto seria *root.esportes.etc*. O LSR coloca os seguintes problemas para serem resolvidos por trabalhos futuros:

- **Disponibilização da semântica juntamente com os objetos**: Toda a estratégia do LSR baseia-se no fato de que a semântica será fornecida juntamente com cada objeto. Pode-se explorar padrões como o XML (*Extensible Markup Language*) e RDF (*Resource Description Framework*), inclusive para organização da árvore semântica, tornando factível o uso do LSR [SCH02].
- **Gerenciamento dinâmico dos assuntos na árvore semântica**: A proposta original do LSR não contempla o gerenciamento da árvore semântica de forma dinâmica, desta forma, objetos pertencentes a assuntos que ainda não estão presentes na árvore semântica acabam tendo de ser acomodados em nós **etc,** vinculados a assuntos mais genéricos na árvore semântica.
- **Adoção de estratégias híbridas**: O autor do LSR propõe a utilização de estratégias híbridas contemplando outros critérios, além da semântica, para a substituição de objetos, como por exemplo, pesos, custo de transferência, freqüência de utilização etc.
- **Impacto da utilização da árvore semântica no mecanismo de *cache***: Não foi realizada nenhuma medição do impacto de se utilizar uma estrutura hierárquica como a árvore semântica, no mecanismo de *cache*. Durante os experimentos realizados com o LSR utilizou-se como árvore semântica um subconjunto de assuntos obtidos através do mecanismo de busca *Yahoo*. Medições mais realistas somente podem ser obtidas utilizando-se toda a estrutura hierárquica de assuntos do *Yahoo*.

- **Armazenamento de histórico de semânticas**: É possível utilizar um histórico das últimas semânticas acessadas pelos clientes para evitar que objetos pertencentes a assuntos diferentes, porém de interesse para um cliente em um determinado momento, sejam descartados.

4.2.2 *Least Semantically Related with Access History* (LSR/H)

A estratégia LSR/H não se limita a avaliar um único assunto de interesse durante a seleção de objetos a serem descartados do mecanismo de *cache*. Esta abordagem faz com que a estratégia LSR/H seja capaz de tratar situações reais que podem ser observadas em uma comunidade de usuários que compartilham um mecanismo de *cache*. Certamente, num dado momento, cada usuário desta comunidade estará interessado em um determinado assunto. Além disso, estes usuários estarão alternando seus assuntos de interesse ao longo do tempo [CAL02].

A estratégia LSR/H identifica os assuntos de maior interesse da comunidade num dado momento como assuntos "**quentes**". O algoritmo implementado pela estratégia LSR/H consiste em descartar objetos que possuam menor afinidade semântica com o grupo de assuntos "quentes" em um dado momento. Para determinar se um assunto deve ser considerado como um assunto "quente" a estratégia LSR/H leva em consideração o histórico de acessos realizados a objetos que estão associados a um determinado assunto.

Da mesma forma que o LSR, a estratégia LSR/H também utiliza uma árvore semântica que define uma hierarquia de assuntos [CAL01]. Um peso é atribuído a cada nó da árvore semântica representando o número de acessos realizados a objetos que estão associados a este nó. Além disso, os acessos mais recentes recebem maior peso. O peso de um nó também é aplicado recursivamente aos nós hierarquicamente ascendentes até atingir a raiz da árvore.

A tabela 5 apresenta o histórico de acessos diretos e acumulados na hierarquia semântica para cada um dos assuntos da árvore apresentada na figura 13. É possível observar que o assunto **informática** apresenta um histórico de cinco acessos a objetos diretamente associados, e mais cinco acessos que foram acumulados recursivamente a partir de seus sub-assuntos. Observa-se também que o assunto **esportes**, apesar de não possuir nenhum histórico de acessos diretos, ainda assim apresenta o valor 3 como peso. Este valor representa o histórico acumulado a partir dos sub-assuntos **ciclismo** e **automobilismo**.

TABELA 5 - HISTÓRICO DE ACESSOS A ASSUNTOS EM UMA ÁRVORE SEMÂNTICA.

Nó (Assunto)	**Número de acessos diretos ao Nó**	**Número de acessos acumulados na hierarquia**
Root	-	-
root.esportes	0	3
root.esportes.futebol	0	0
root.esportes.ciclismo	1	1
root.esportes.automobilismo	2	2
root.informática	5	10
root.informática.software	0	0
root.informática.hardware	5	5

Tomando a árvore apresentada na figura 13 como exemplo, a estratégia LSR/H inicialmente selecionará para descarte objetos associados a sub-assuntos relacionados ao assunto **esportes**, mais especificamente objetos associados ao assunto **futebol**. Somente quando não houver mais nenhum objeto associado à sub-árvore do assunto **esportes** a estratégia LSR/H descartará objetos associados à sub-árvore do assunto **informática**.

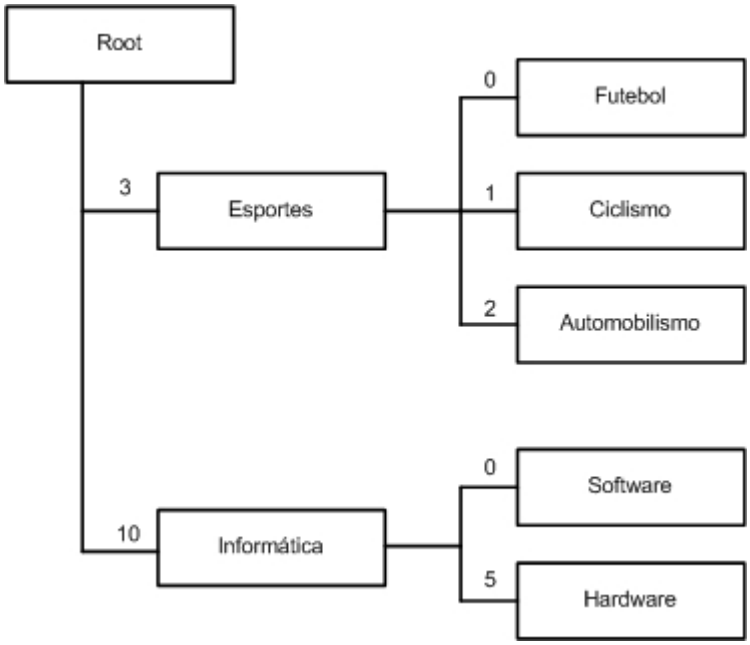


FIGURA 13 - ÁRVORE SEMÂNTICA UTILIZADA PELO LSR/H.

4.2.3 Temperatura

A noção de **temperatura** está relacionada a uma arquitetura colaborativa de *proxies* na *web*, baseada em semântica [BRU02], que foi proposta por um grupo de estudo localizado no *Institut National des Sciences Appliquées de Lyon* (INSA), Lyon, França. Esta arquitetura é composta de três componentes, que podem ser observados na figura 14:

- ***User Proxy***: Os *user proxies* definem a primeira camada da arquitetura que está localizada próxima ao usuário final. Esta camada é composta de *proxies* que estão localizados nos próprios computadores dos usuários finais. Este *proxy* tem como objetivo substituir os recursos de *cache* integrados aos navegadores *web*. Os principais navegadores *web* possuem recursos rudimentares de *cache* que se restringem ao armazenamento de poucos objetos e à pré-recuperação de objetos que possuem *links* em um documento recém exibido pelo navegador. O principal objetivo dos *user proxies* é de capturar o perfil e o comportamento dos usuários finais, bem como

informações de contexto. A arquitetura contempla a interação entre os *proxies* de cada usuário para a troca de semântica, informações de contexto e objetos armazenados em *cache*.

- ***Aggregate Proxy***: Estes componentes fazem parte da segunda camada da arquitetura proposta. Esta camada é responsável por armazenar informações referentes aos documentos, semântica e informações de contexto, armazenadas em um conjunto de *user proxies*. Desta forma um *aggregate proxy* funciona como um diretório dentro da arquitetura. Os *aggregate proxies* podem colaborar entre si trocando semântica, informações de contexto e eventualmente documentos. A arquitetura propõe que um *aggregate proxy*, de forma opcional, também armazene documentos estabelecendo assim o conceito tradicional de hierarquia de *proxies*.
- ***Meta Proxy***: Esta é a camada mais alta da arquitetura e tem como objeto funcionar apenas como um diretório de semântica e informações de contexto que podem ser compartilhadas entre um conjunto de *aggregate proxies*. Esta camada é opcional na arquitetura.

Esta arquitetura apresenta várias semelhanças com relação às estratégias de substituição LSR e LSR/H, principalmente quanto à representação da semântica dos objetos através de temas ou tópicos que são organizados em uma estrutura de árvore.

A arquitetura apresentada na figura 14 trata a semântica como um conjunto de assuntos de interesse para uma comunidade de usuários. Os interesses da comunidade podem mudar ao longo do tempo. Desta forma os assuntos de maior interesse são denominados "**temas quentes**", do inglês ***hot themes***. Os assuntos são organizados hierarquicamente em uma estrutura de árvore, sendo que, assuntos mais genéricos ficam localizados no topo da hierarquia e assuntos mais específicos ficam localizados na base da hierarquia. Os objetos, também denominados **documentos**, são

armazenados nas extremidades da árvore (folhas), ligados aos respectivos temas aos quais estão relacionados. A partir da análise da figura 15, que apresenta a estrutura semântica utilizada pela arquitetura, é possível observar que cada associação entre assuntos da árvore recebe um peso que representa a probabilidade de um documento ligado a um assunto ser requisitado, logo após um documento ligado ao assunto hierarquicamente superior ter sido requisitado. As associações entre assuntos e objetos, ou documentos *web*, também recebem pesos. Estes pesos representam o número de vezes que um determinado documento foi requisitado.

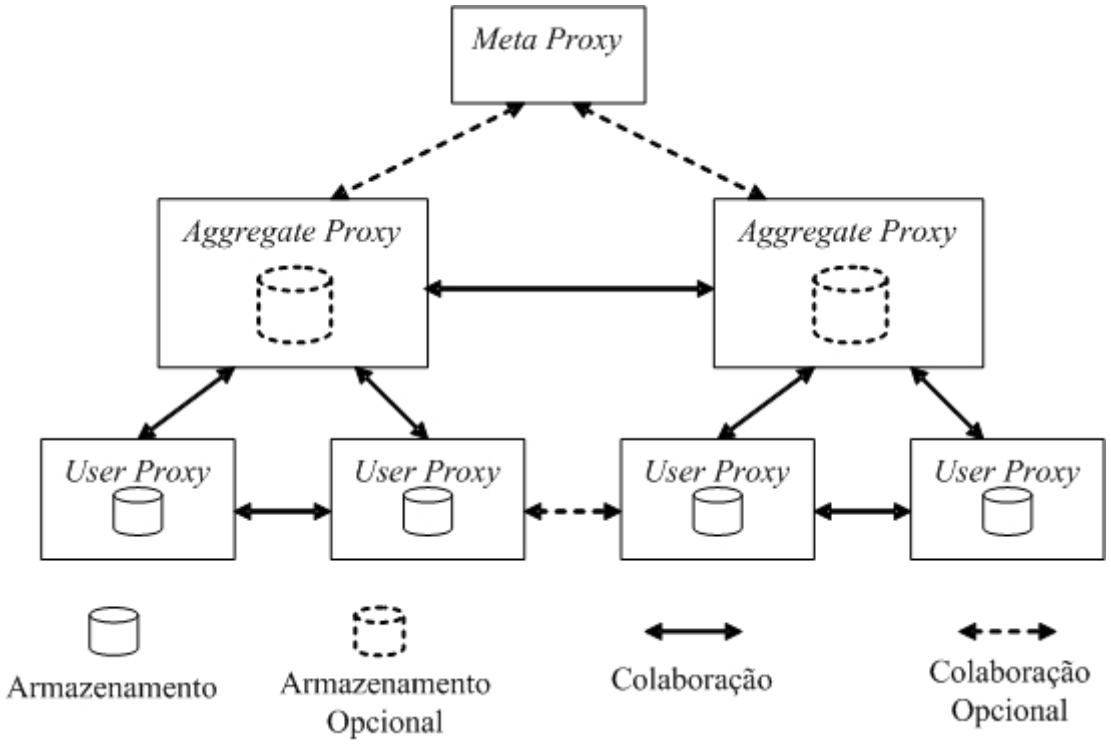


FIGURA 14 - ARQUITETURA COLABORATIVA DE *PROXIES* NA *WEB* BASEADA EM SEMÂNTICA.

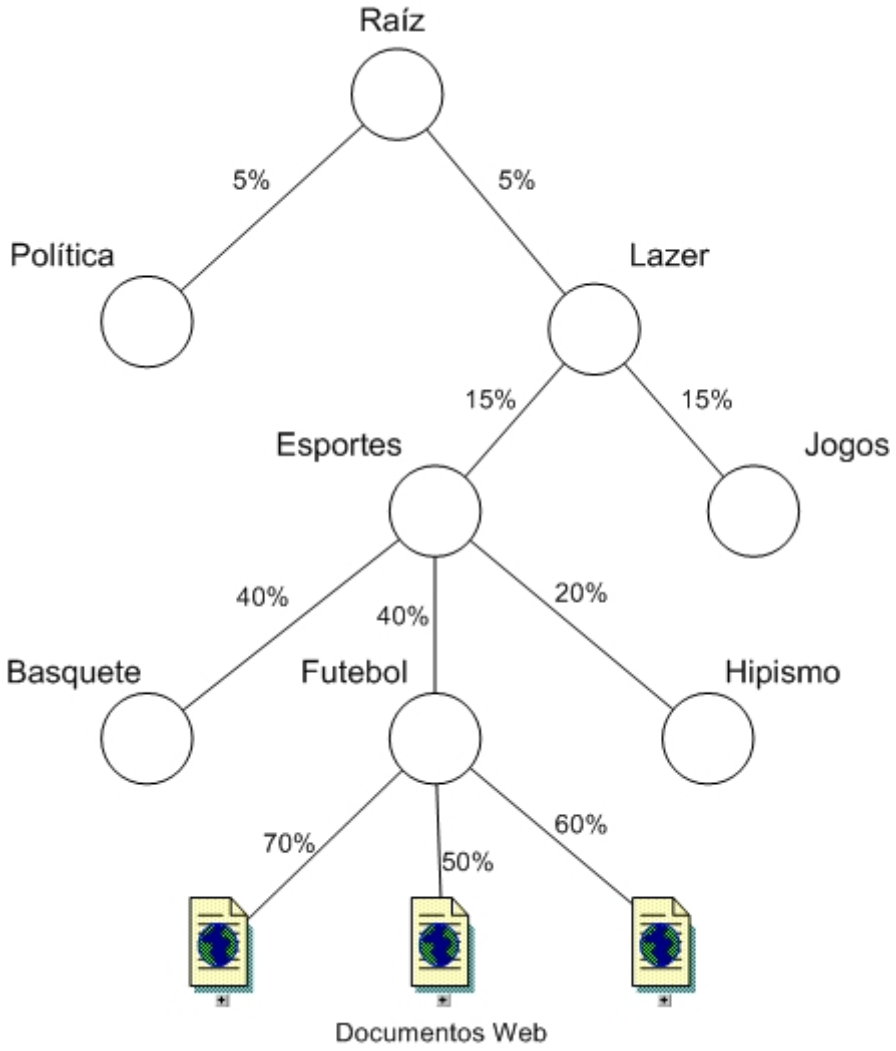


FIGURA 15 - ÁRVORE SEMÂNTICA.

A estratégia de substituição em *cache* definida pela arquitetura utiliza o conceito de **temperatura** como critério para selecionar objetos que serão descartados para acomodar um novo objeto quando o repositório de armazenamento tiver atingido seu limite. Cada documento presente no *cache* recebe um valor que representa sua "temperatura" em um dado momento. A "temperatura" de um objeto é calculada com base na sua localização na árvore que representa a estrutura semântica, e com base no número de vezes que o objeto foi requisitado. Conceitualmente a temperatura é um valor numérico atribuído a um objeto, que representa a probabilidade deste objeto vir a ser requisitado num futuro próximo [BRU02]. Em iterações definidas por um intervalo constante de requisições realizadas ao *cache*, a estratégia de substituição recalcula a

“temperatura” de cada objeto. Durante uma iteração os objetos que foram acessados pelo menos uma vez serão candidatos a se tornarem mais “**quentes**” e os objetos que não foram acessados são candidatos a se tornarem mais “**frios**”. Para cada objeto a estratégia de substituição calcula a variação de temperatura em relação à iteração anterior e a propaga verticalmente através da árvore semântica até a raiz. Em seguida a estratégia de substituição propaga a variação de temperatura no sentido dos assuntos para os objetos [BRU02]. Desta forma, a “temperatura” de objetos que não tenham sido acessados nesta iteração, mas que pertencem aos assuntos de alto interesse para a comunidade, será mantida alta, evitando que estes sejam descartados.

4.2.4 Considerações a Respeito de Estratégias Baseadas em Semântica

O principal aspecto que dificulta a utilização das estratégias de substituição LSR e LSR/H em situações reais está no fato de que estas estratégias pressupõem que os objetos requisitados por um cliente ao mecanismo de *cache* já estão pré-classificados em uma hierarquia semântica. A estratégia baseada em temperatura sugere que a semântica dos objetos seja obtida a partir de um conjunto de informações presentes no objeto. Este conjunto de informações consiste em *meta-tags* de documentos HTML, títulos de documento HTML, e eventuais *links* que um documento HTML tenha para outros documentos HTML. A última abordagem é mais viável, porém é importante observar que um mecanismo de *cache* não gerencia apenas documentos HTML, que são altamente estruturados através do uso de *tags*. Para grande categoria de objetos gerenciados por um mecanismo de *cache* não é possível extrair semântica diretamente. Os estudos realizados mostraram que grande parte dos objetos gerenciados em um mecanismo de *cache* são imagens, arquivos binários de diversos tipos ou até mesmo arquivos de formatação como, por exemplo, arquivos CSS (*Cascading Style Sheet*) que não carregam nenhuma semântica.

Com o objetivo de verificar a quantidade de requisições a objetos binários,

foi realizada a análise de um arquivo de *log* do servidor *proxy squid* da rede acadêmica de computados da PUCPR, que possui aproximadamente 1.580 usuários.

O arquivo de *log* utilizado nas análises contém 19.020 requisições. A tabela 6 apresenta a distribuição das requisições classificadas por *content-type* (campo que faz parte do cabeçalho de resposta do protocolo HTTP). O campo *content-type* tem como objeto informar ao navegador *web* o tipo de objeto que está sendo retornado como resposta a uma requisição feita a um servidor *web* [RFC2616]. É possível observar que os grupos “image/gif” e “image/jpeg” juntos, correspondem a 44,44% das requisições. Considerando-se a soma dos grupos que representam objetos binários, incluindo imagens, tem-se um total de 53,63 % das requisições. É possível observar também que a informação referente ao *content-type* não foi retornada em 11,10% das requisições. Dentro do universo analisado, foram identificados três grupos de objetos dos quais pode-se extrair semântica diretamente. São eles: “text/html”, “text/plain” e “text/xml”. Desta forma é possível observar que somente 23,03% dos objetos acessados estão em um formato que permite a comparação semântica de forma direta.

A partir dos resultados obtidos fica evidente que existe um fator muito importante que não foi considerado em nenhuma das estratégias de substituição em *cache* baseadas em semântica, apresentadas neste capítulo. Este fator diz respeito ao número de objetos de onde realmente é possível extrair semântica de forma direta. Esta distribuição realmente se apresenta de forma prática, pois quando um usuário requisita uma página *web*, inicialmente o navegador realiza uma requisição para recuperar o documento HTML solicitado. Neste documento certamente existirão referências para diversos outros tipos de objetos como, por exemplo, imagens, arquivos de vídeo, arquivos de som, *applets*, entre outros, que também serão recuperados pelo navegador para que seja possível montar a página que será finalmente apresentada ao usuário. A estratégia LSR-VM, apresentada no próximo capítulo, propõe uma nova abordagem baseada em semântica para substituição em *cache*, considerando inclusive o tratamento de objetos binários.

Este capítulo apresentou o estado da arte referente a estratégias de substituição em *cache*. Foram apresentadas estratégias de substituição tradicionais que levam em consideração as propriedades operacionais dos objetos como critério de descarte, bem como estratégias baseadas em semântica que apresentam novas abordagens, propondo soluções inovadoras para tratar o problema de substituição em mecanismos de *cache*.

TABELA 6 - DISTRIBUIÇÃO DE ACESSOS POR *CONTENT-TYPE* NO *LOG* DO *PROXY* DA PUCPR

content-type	**Requisições**	**%**	***KBytes***	**%**	***Hit-%***
image/gif	6.123	32,19	9.459	7,51	57,19
text/html	3.853	20,26	40.214	31,94	1,97
image/jpeg	2.329	12,25	16.947	13,46	47,70
<desconhecido>	2.111	11,10	653	0,52	6,54
<erro>	1.319	6,93	2.020	1,60	9,10
Application/x-javascript	909	4,78	3.715	2,95	51,38
text/css	518	2,72	2.840	2,26	75,68
text/plain	470	2,47	803	0,64	68,72
application/x-shockwave-flash	458	2,41	8.596	6,83	36,90
<seguro>	437	2,30	3.586	2,85	0,00
application/octet-stream	84	0,44	718	0,57	47,62
application/x-mms-framed	81	0,43	31.795	25,25	0,00
application/vnd.ms.wms-hdr.asfv1	76	0,40	271	0,22	0,00
text/xml	63	0,33	355	0,28	28,57
Image/png	50	0,26	210	0,17	68,00
application/pkix-crl	42	0,22	37	0,03	0,00
app/x-hotbar-xip20	37	0,19	14	0,01	0,00
text/javascript	26	0,14	168	0,13	15,38
image/x-icon	7	0,04	25	0,02	71,43
application/msword	3	0,02	376	0,30	0,00
Outros (13 grupos)	24	0,13	3114	2,47	29,17
Total	**19.020**	**100,00**	**125.915**	**100,00**	**33,68**

5. ESTRATÉGIA PROPOSTA

Este capítulo apresenta uma nova estratégia baseada em semântica para solucionar o problema de substituição em um mecanismo de *cache*. Esta estratégia é denominada *Least Semantically Related – Vector Model* (LSR-VM). Parte da denominação atribuída a esta nova estratégia é comum à estratégia LSR (*Least Semantically Related*) apresentada no capítulo anterior. Esta característica deve-se ao fato de que ambas estratégias pressupõem que todos os documentos que fazem parte de um conjunto de acessos seqüenciais realizados por um usuário em um determinado intervalo de tempo, estão de alguma forma semanticamente relacionados.

É possível analisar este comportamento a partir de um exemplo hipotético onde um cliente, utilizando um navegador, tenha iniciado uma sessão na *web* a partir de um mecanismo de busca como, por exemplo, Google ou Yahoo. Este cliente estaria inicialmente interessado em temas relacionados com esportes, mais especificamente em basquetebol. O cliente iniciaria então uma busca a partir do mecanismo de busca usando o termo "*basketball*". Como resposta o cliente receberia uma relação de *links* para páginas relacionadas ao assunto. É razoável supor que o cliente escolha o *link* para a página da NBA (*National Basketball Association*). A partir desta página o cliente poderia se interessar pelos 50 maiores jogadores de todos os tempos, desta forma o cliente selecionaria o *link* para a relação de jogadores. A partir desta relação o cliente poderia se interessar pelo jogador Michael Jordan. Assim o cliente poderia visitar uma seqüência de páginas dentro do *site* da NBA contendo informações sobre o referido jogador. O cliente poderia então iniciar outra busca a partir do mecanismo de busca Google utilizando agora o nome "Michael Jordan". Como resultado desta segunda busca o cliente receberia uma relação de *links* para páginas contendo informações sobre o jogador de basquete Michael Jordan. O cliente poderia então iniciar uma seqüência de acessos pelo *site* oficial do jogador, depois por diversos *sites* de fans. Depois de obter as informações que desejava o cliente finalizaria a sessão

fechando seu navegador.

É razoável afirmar que a maior parte dos documentos HTML, ou páginas *web*, acessados pelo cliente no exemplo anterior compartilham semântica, e por conseguinte estão relacionados aos mesmos assuntos ou conceitos. Supondo que todos os acessos citados no exemplo anterior tenham sido realizado por um cliente através de um servidor *proxy*, é desejável que a estratégia de substituição utilizada no mecanismo de *cache* não descarte objetos relacionados aos conceitos **esportes**, **basquetebol**, **jogadores**, **atletas**, etc., pois existe uma grande probabilidade de que os documentos HTML que serão requisitados em seguida, ou num futuro próximo, também estejam relacionados a estes conceitos.

Tanto a estratégia LSR quanto a nova estratégia apresentada neste capítulo têm como objetivo manter no mecanismo de *cache* os objetos que estejam semanticamente relacionados com os objetos que estão sendo acessados por uma comunidade de clientes em um dado momento. A diferença crucial entre as duas estratégias reside no fato de que o LSR pressupõe que tanto os objetos armazenados em *cache* quanto os novos objetos que estão sendo acessados pela comunidade de clientes, fazem parte de uma taxionomia semântica pré-existente. Em outras palavras, todos os objetos na *web* estariam pré-classificados em uma estrutura em forma de árvore onde cada nó representa um assunto relacionado com a semântica do objeto. Este aspecto torna difícil a utilização prática da estratégia LSR, pois na Internet não existe tal estrutura a partir da qual seria possível obter a semântica de todos os objetos. Alguns projetos como *Open Directory Project*, também denominado DMOZ[2], têm como objetivo criar um diretório mantido por humanos, a partir do qual seria possível obter a classificação semântica de objetos na *web*. Atualmente a base de dados do DMOZ possui aproximadamente 4,6 milhões de *sites* classificados em aproximadamente 790 mil assuntos. O projeto conta ainda com aproximadamente 70

[2] http://www.dmoz.org

mil editores responsáveis por manter a consistência e integridade da base de dados. A princípio poder-se-ia pensar em utilizar a estrutura mantida pelo projeto DMOZ para obter-se a classificação semântica dos objetos, necessária à aplicação prática da estratégia LSR. Porém existe um aspecto muito importante que inviabiliza esta abordagem. Este aspecto refere-se ao fato de que objetos referenciados em documentos HTML, que compõem o resultado visual final quando um cliente informa uma URL em seu navegador, não são diretamente classificados no projeto DMOZ. Desta forma, grande parte dos objetos gerenciados em um mecanismo de *cache*, como por exemplo, imagens, arquivos binários, arquivos de som, arquivos de formatação como (CSS - *Cascading Style Sheet*), não são individualmente classificados no projeto DMOZ. Conforme resultados apresentados no capítulo 7, estes tipos de objetos representam mais da metade dos objetos armazenados em *cache*.

5.1 CONCEPÇÃO DA ESTRATÉGIA LSR-VM

O principal desafio durante a concepção da estratégia LSR-VM (*Least Semantically Related – Vector Model*) foi encontrar uma maneira de eliminar a árvore semântica utilizada em todas as demais estratégias baseadas em semântica que foram apresentadas no capítulo anterior. Desta forma, a principal questão a ser solucionada por este trabalho é: "**Como manter a comparação semântica entre os objetos em *cache*, como critério de substituição, sem utilizar uma taxionomia pré-existente (árvore semântica)?**" A resposta a esta questão surgiu com a análise do funcionamento de mecanismos de busca na Internet. A estratégia LSR-VM adotou o **Modelo Vetorial**, teoria comumente utilizada em problemas de recuperação de informações e em mecanismos de busca na Internet, como solução para estabelecer a comparação semântica entre os objetos do *cache*. Mecanismos de busca na Internet como, por exemplo, Google e Yahoo implementam estratégias de recuperação de informações que permitem recuperar documentos HTML a partir de termos fornecidos

por usuários em uma **consulta** que é submetida ao mecanismo de busca. O objetivo do mecanismo de busca é estabelecer uma comparação semântica entre a consulta e os documentos previamente indexados em sua base de dados, retornando os documentos que apresentarem maior similaridade com a consulta. Atualmente o Modelo Vetorial é uma das estratégias de recuperação de informações mais eficazes. Este capítulo apresenta as características dos mecanismos de busca na Internet, a teoria do Modelo Vetorial e finalmente o funcionamento da estratégia LSR-VM.

5.1.1 Mecanismos de Busca na Internet

Os mecanismos de busca na Internet nada mais são do que sistemas de recuperação de informações cujos documentos recuperados são páginas *web*. As paginas *web* são documentos altamente estruturados, que são geralmente construídos utilizando-se a linguagem HTML ou XML. Um documento HTML é formado por *tags* que definem como as informações carregadas pelo documento serão exibidas. Esta característica especial é explorada por alguns mecanismos de busca na Internet, pois serve como fonte de informação a respeito dos termos que compõem o documento. Uma *tag* HTML “<H1>”, que é usada para destacar um ou mais termos, pode servir como indicativo de que estes termos também são importantes na representação lógica do documento. Um mesmo termo que aparece em dois documentos HTML pode receber pesos diferenciados caso apareça em um deles como título, cabeçalho, negrito ou em fontes diferenciadas. Estas características são exploradas por alguns dos principais mecanismos de busca na Internet como o Google e Excite [MEN02]. Alguns mecanismos de busca na Internet também exploram o fato de que as páginas *web* são documentos altamente encadeados (ligados). Este encadeamento também é representado através de *tags* HTML, e permite a navegação entre documentos na Internet. O número de *links* para um documento pode ser utilizado para calcular a importância global deste documento. Parte-se do princípio que um documento que é

apontado por um grande número de outros documentos apresenta uma maior importância global. Quando existe uma ligação entre dois documentos pode-se supor que estes apresentem uma certa correlação, e que eventualmente **compartilhem semântica**. Alguns mecanismos de busca utilizam os *links* durante a representação lógica de um documento. Seguindo este conceito, se um documento X possuir um *link* para um documento Y, então o vetor de termos que representa o documento Y poderá ser associado a este *link*, passando a contribuir para a representação lógica do documento X. Estudos realizados por [CUT97] apresentam resultados indicando que a eficácia na recuperação de documentos aumenta quando as características dos documentos HTML são exploradas.

É possível categorizar os mecanismos de busca na Internet em dois grandes grupos. O primeiro grupo compreende os mecanismos de busca genéricos. Estes mecanismos têm como objetivo permitir a busca de informação em todas as páginas existentes na Internet. O segundo grupo compreende os mecanismos de busca para fins específicos. Estes mecanismos, ao contrário dos mecanismos genéricos, permitem a recuperação de informações a partir de um domínio ou ambiente específico. Como exemplo pode-se citar um mecanismo de busca que atue na recuperação de documentos dentro da Intranet de uma empresa. Outro exemplo seria um mecanismo de busca que recupere documentos especificamente relacionados à área médica.

Alguns mecanismos de busca na Internet se baseiam na recuperação de dados, porém os principais mecanismos levam em consideração a recuperação de informações. A recuperação de dados se preocupa em determinar quais documentos dentro de uma coleção possuem as palavras chaves de uma **consulta** fornecida por um usuário. Porém, geralmente os usuários estão mais preocupados em recuperar informações relacionadas a um determinado assunto do que simplesmente em recuperar documentos que possuam as palavras chave de uma consulta submetida a um mecanismo de busca. O problema reside essencialmente na distinção entre o conceito de dado e de informação. Um dado pode se apresentar de diferentes formas, como por

exemplo, números, letras, palavras ou figuras fora de um determinado contexto. A informação também se apresenta de várias formas, porém o que transforma um dado em informação é a contextualização, ou seja, um dado deve ser apresentado dentro de um contexto para ser considerado como informação.

Uma linguagem voltada à recuperação de dados tem como objetivo recuperar todos os documentos que atendam a condições rígidas, tais como uma expressão regular. Assim sendo, para um sistema de recuperação de dados, se qualquer um dos critérios da consulta falhar o documento não será recuperado. Já um sistema de recuperação de informações deve ser flexível bastante de forma a permitir que documentos relevantes sejam recuperados mesmo que estes apresentem pequenas divergências em relação à consulta.

Os sistemas de recuperação de dados podem funcionar bem, como soluções para recuperação em bancos de dados relacionais baseados, por exemplo, na linguagem SQL, porém não são eficazes quando o problema é recuperar informações sobre um determinado assunto ou domínio. Para isto, é necessário que o conteúdo dos documentos que fazem parte do conjunto onde será realizada a busca seja interpretado e classificado de acordo com o grau de relevância em relação a uma consulta fornecida pelo usuário. Este tipo de classificação muitas vezes envolve a extração de semântica a partir de uma coleção de documentos. Os sistemas de recuperação de informações têm como desafio solucionar estes problemas, assim sendo a noção de **relevância** passa a ser o aspecto mais importante a ser observado. Um sistema de recuperação de informações tem como objetivo primordial à recuperação de todos os documentos que têm relevância em relação à consulta de um usuário e ao mesmo tempo recuperar o menor número possível de documentos não relevantes.

Os sistemas de recuperação de informações podem utilizar basicamente três modelos formais. O primeiro modelo que foi concebido, sendo largamente aceito, foi o modelo denominado **booleano** [SAL83]. Este modelo se baseia em consultas que são representadas por expressões booleanas com semântica precisa. A estratégia de

recuperação do modelo booleano é caracterizada por utilizar critérios de decisão binários além disso, não existe uma forma de classificar os documentos recuperados por ordem de relevância em relação à consulta. Na verdade o modelo booleano se aproxima muito mais de um sistema de recuperação de dados do que um sistema de recuperação de informações. O segundo modelo é um dos mais utilizados hoje em dia e recebe a denominação de **modelo vetorial** [YAT99]**.** Este modelo caracteriza tanto a consulta do usuário quanto os documentos de uma coleção, como vetores de termos. A similaridade entre a **consulta** e um determinado documento é determinada pela proximidade entre os vetores que os representam em um espaço vetorial. Este modelo é bem mais flexível que o booleano, pois permite a recuperação de documentos que são parcialmente relacionados à consulta. Além disso, os documentos recuperados podem ser classificados por ordem de similaridade em relação à consulta. O terceiro modelo é o **probabilístico** [ROB77] que caracteriza o problema de recuperação de informações através de um modelo probabilístico. O modelo tenta estimar, a partir de uma consulta e de um documento que faz parte de uma coleção, a probabilidade do usuário se interessar pelo respectivo documento. A recuperação é realizada em vários passos e o usuário é solicitado a analisar os documentos recuperados em cada passo. Além disso, apesar de ser possível apresentar documentos recuperados usando uma classificação por probabilidade de relevância, o modelo probabilístico não leva em consideração o número de vezes que um determinado termo aparece em um documento.

O modelo vetorial foi a técnica escolhida para integrar a estratégia de substituição LSR-VM por ser um modelo que permite estabelecer a similaridade através da comparação parcial dos conteúdos dos objetos. Além disso, conforme mencionado anteriormente, o modelo vetorial é uma das técnicas mais utilizadas para solucionar problemas de recuperação de informações.

5.1.2 Modelo Vetorial

Os sistemas de recuperação de informação geralmente utilizam termos índice para indexar e recuperar documentos. Os termos índice nada mais são do que as palavras que compõem um documento. A tarefa de recuperação da informação se baseia na premissa de que a semântica de um documento reside no conjunto de termos índice que o compõe. Na verdade esta é uma abordagem simplista do problema, pois uma parte considerável da semântica de um documento é perdida quando seu texto original é substituído por um conjunto de termos índice durante o processo de indexação. Uma vez que um documento é indexado não é possível reconstruir o documento original a partir do vetor de termos criado.

Em um documento também é possível encontrar um grande número de palavras que não contém semântica alguma. Estas palavras são geralmente preposições, pronomes, advérbios, entre outras. Estas palavras recebem o nome de palavras de parada, do inglês *stop words*, e geralmente aparecem em grande quantidade dentro da maioria dos documentos de uma coleção. Estas palavras devem ser descartadas, pois não são úteis para representar um documento. As palavras restantes são utilizadas para representar o conteúdo do documento, e por conseguinte sua semântica. Algumas palavras variantes também podem ser agrupadas e posteriormente substituídas por sua raiz gramatical como, por exemplo, as palavras “alegre” e “alegremente”, podem ser representadas pelo termo “alegria”. Este processo é denominado ***stemming***. O processo de retirada das palavras de parada, substituição das palavras variantes (*stemming*) e decomposição dos documentos em termos índice é denominado **transformação de texto**. Utilizando-se o modelo vetorial, cada documento passa a ser representado por um vetor com *n* termos, onde *n* é o número total de termos distintos encontrados em todos os documentos de uma coleção [SAL83] conforme pode ser observado na figura 16.

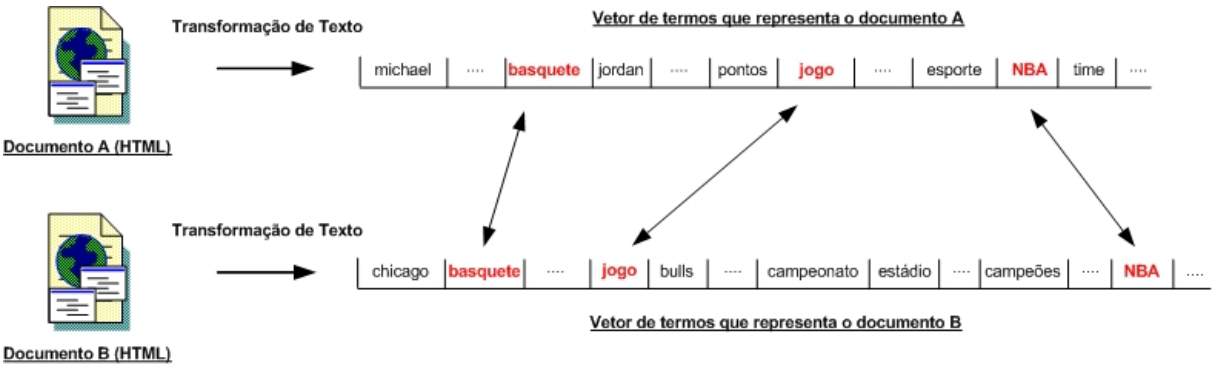


FIGURA 16 - TRANSFORMAÇÃO DE TEXTO CRIANDO UM VETOR QUE REPRESENTA O DOCUMENTO ORIGINAL.

Após a transformação de texto, um documento passa a ser logicamente representado por um vetor [*d1*,....,*di*,....,*dn*] onde *di* é o grau de importância do *i-ésimo* termo que representa o conteúdo do documento. Cada entrada do vetor é composta por um valor numérico denominado **peso**. Geralmente o vetor que representa um documento é um vetor esparso, pois apresenta muitas entradas com valor zero devido a ausência de termos. A maneira de calcular os pesos dos termos pode variar em função do modelo utilizado pelo sistema de recuperação de informações, porém a forma mais aceita se baseia em dois fatores. O primeiro fator está relacionado à freqüência do termo, ou seja, ao número de vezes que este termo aparece no documento. Este fator é denominado *tf*, do inglês *term frequency*. O segundo valor está relacionado a freqüência do documento, que corresponde ao número de documentos que possuem o termo. Este fator é denominado *df*, do inglês *document frequency*. Geralmente, quanto maior o número de documentos onde um determinado termo aparece, menor será a importância deste termo na representação lógica dos documentos de uma coleção. Assim sendo, o fator *df* contribui inversamente para o cálculo do peso, podendo ser caracterizado como o inverso da freqüência do documento, recebendo assim a denominação de *idf*, do inglês *inverse document frequency*. Finalmente o peso de um termo dentro do vetor, que representa logicamente um documento, pode ser calculado pelo produto entre a freqüência do termo e o inverso da freqüência do documento (*tf* * *idf*). A figura 17 mostra um vetor que representa um documento no espaço vetorial

após o cálculo dos pesos.

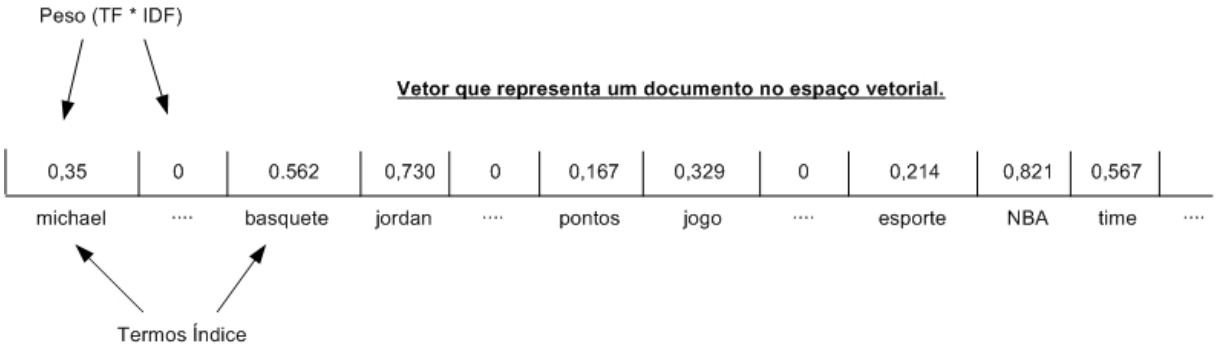


FIGURA 17 - VETOR QUE REPRESENTA UM DOCUMENTO NO ESPAÇO VETORIAL.

A **consulta** de um usuário é caracterizada da mesma forma que um documento, passando pelo processo de transformação de texto e cálculo de pesos para ser finalmente representada por um vetor. A partir do momento em que, tanto os documentos quanto a consulta, estão logicamente representados através de vetores, é possível recuperar os documentos que mais se aproximam da consulta em um espaço vetorial. Para isto pode-se utilizar o co-seno do ângulo entre o vetor da consulta e o vetor de cada documento da coleção. O resultado do cálculo do co-seno é dado no intervalo entre 0 e 1. Desta forma, quanto mais próximo do valor 1 o resultado estiver, maior é a similaridade entre um determinado documento e a consulta. O cálculo da similaridade entre a consulta e um determinado documento é realizado da seguinte forma [YAT99]:

Para o modelo vetorial, o peso $w_{i,j}$ associado a um par (k_i, d_j) é positivo e não binário, onde *k* representa um determinado termo e *d* representa um determinado documento. Os pesos dos termos índice da consulta também devem ser calculados. Seja $w_{i,j}$ o peso associado ao par $[k_i, q]$, onde $w_{i,q} \geq 0$. Então o vetor da consulta $\vec{q}$ é definido como $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$ onde *t* representa o número total de termos do sistema. Da mesma forma o vetor do documento d_j é representado por $\vec{d} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$. Desta forma, a similaridade entre um documento e a consulta é calculada através da fórmula apresentada na figura 18.

$$sim(d_j, q) = \frac{\vec{d}_j \bullet \vec{q}}{\left|\vec{d}_j\right| \times \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

FIGURA 18 - FÓRMULA PARA CÁLCULO DA SIMILARIDADE ENTRE UM DOCUMENTO E UMA CONSULTA USANDO O MODELO VETORIAL.

A eficácia da recuperação é medida através de dois fatores: **cobertura** (ou **recobrimento)** e **precisão**. A cobertura é calculada pelo quociente entre o número de documentos relevantes recuperados e o número total de documentos relevantes. Se a cobertura for igual a 1 significa que todos os documentos relevantes da coleção foram recuperados. Já a precisão é calculada através do quociente entre o número de documentos relevantes recuperados e o número de documentos recuperados. Se a precisão for igual a 1 significa que somente documentos relevantes foram recuperados e nada mais. Um sistema de recuperação de informações ideal procura recuperar todos os documentos relevantes existentes e nada mais, assim sendo, tanto a precisão quanto a cobertura seriam iguais a 1. Na prática isto é impossível, pois a recuperação está diretamente ligada a formulação da consulta. Muitas vezes a consulta pode estar mal formulada, ou até mesmo o conceito de relevância para um usuário pode variar ao longo do tempo [YAT99]. O modelo vetorial utiliza o cálculo de similaridade para tentar aproximar o grau de relevância entre a consulta e os documentos de uma coleção.

5.1.2.1 Cálculo do Peso dos Termos Índice

A maneira mais eficiente de calcular o peso dos termos índice definida por [SAL83] baseia-se nos princípios que envolvem técnicas de *clustering*. Para entender estes princípios é necessário analisar a seguinte situação:

Dada uma coleção *C* de objetos e uma "vaga" descrição de um conjunto *A* formado por objetos pertencentes a *C*, um algoritmo simples de *clustering* tem como objetivo separar a coleção *C* em dois conjuntos de objetos, sendo o primeiro referente a objetos relacionados com *A* e o segundo referente aos demais objetos de *C*. É importante considerar que não existe uma definição exata de quais características definem um objeto como pertencente ao conjunto *A*. O objetivo principal do modelo vetorial é definir quais objetos são relevantes em relação a uma consulta e quais objetos não são.

Para que o modelo vetorial seja abordado como um problema de *clustering* [YAT99] é necessário estabelecer uma analogia na qual a coleção de todos os documentos deve ser considerada como *C* e a consulta fornecida por um usuário deve ser considerada como *A*. A partir deste cenário o problema se reduz a determinar quais documentos fazem parte de *A* e quais não fazem. Em um problema clássico de *clustering* dois aspectos importantes devem ser abordados. O primeiro refere-se a identificar quais características melhor descrevem os objetos pertencentes a *A*. O segundo aspecto refere-se a identificar quais características melhor diferenciam objetos em *A* dos demais objetos em *C*. O primeiro aspecto é utilizado como métrica para estabelecer a similaridade *intra-clustering*. O segundo aspecto é utilizado para determinar a dissimilaridade *inter-clustering*. Um algoritmo de *clustering* eficiente busca atingir o equilíbrio entre estes dois aspectos.

No modelo vetorial a similaridade *intra-clustering* é medida pela freqüência de ocorrência de um termo k_i em um documento d_j. Esta freqüência define o fator *df*, do inglês *document frequency*, anteriormente citado. O fator *df* define quão bem um termo descreve o conteúdo do documento. A dissimilaridade *inter-clustering* é medida pelo inverso da freqüência do termo k_i em todos os documentos da coleção. Esta freqüência define o fator *idf*, do inglês, *inverse document frequency*. O fator *idf* é utilizado para minimizar o fato de que termos que aparecem em muitos documentos não são bons para determinar a relevância de um documento em relação aos outros. A

definição a seguir apresenta a forma mais eficaz de calcular o peso dos termos índice em um vetor que representa um documento em uma coleção [SAL88]:

Seja N o número total de documentos em uma coleção e n_i o número de documentos em que o termo índice k_i aparece. Seja $freq_{i,j}$ a freqüência de ocorrências do termo índice k_i no documento d_j. Então a freqüência normalizada $f_{i,j}$ do termo índice k_i no documento d_j é dada pela figura 19:

$$f_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}}$$

FIGURA 19 - FÓRMULA PARA CÁLCULO DA FREQÜÊNCIA NORMALIZADA DE OCORRÊNCIA DE UM TERMO EM UM DOCUMENTO.

O denominador da fração apresentada na figura 19 ($\max_l freq_{l,j}$) corresponde à freqüência do termo índice de maior freqüência dentre todos os termos índice que aparecem no documento d_j. Se o termo índice k_i não aparece no documento d_j então $f_{i,j} = 0$. O inverso da freqüência do documento, idf_i para o termo índice k_i é dado pela figura 20:

$$idf_i = \log \frac{N}{n_i}$$

FIGURA 20 - FÓRMULA PARA CÁLCULO DO INVERSO DA FREQÜÊNCIA DO DOCUMENTO PARA UM TERMO ÍNDICE.

Desta forma, a melhor fórmula conhecida para cálculo do peso $w_{i,j}$ referente ao termo índice k_i no documento d_j é dada pela figura 21:

$$w_{i,j} = f_{i,j} \times \log \frac{N}{n_i}$$

FIGURA 21 - FÓRMULA PARA CÁLCULO DO PESO DE UM TERMO ÍNDICE NO VETOR QUE REPRESENTA UM DOCUMENTO NO MODELO VETORIAL.

O cálculo dos pesos referentes aos termos índice da consulta é realizado de forma diferenciada. De acordo com estudos realizados por [SAL88] a maneira mais eficiente para calcular o peso $w_{i,q}$ do termo índice k_i de uma consulta *q*, de acordo com o modelo vetorial, é dada pela figura 22, onde $freq_{i,q}$ corresponde à freqüência de ocorrências do termo k_i na consulta *q*, e $\max_l freq_{l,q}$ corresponde a freqüência do termo índice de maior freqüência dentre todos os termos índice que aparecem na consulta *q*.

$$w_{i,q} = \left(0.5 + \frac{0.5\, freq_{i,q}}{\max_l freq_{l,q}} \right) \times \log \frac{N}{n_i}$$

FIGURA 22 - FÓRMULA PARA CÁLCULO DO PESO DE UM TERMO ÍNDICE NO VETOR QUE REPRESENTA UMA CONSULTA NO MODELO VETORIAL.

A grande vantagem do modelo vetorial está no fato de ser uma estratégia que permite a recuperação de informações com base em combinações parciais (*partial match*) dos documentos em relação a uma consulta. Isto permite que documentos que estejam parcialmente relacionados com uma consulta também sejam considerados. Outro aspecto importante reside no fato de que o modelo vetorial é uma estratégia que permite classificar os documentos relevantes em relação a uma consulta, estabelecendo um *ranking* de similaridade. Devido a estas características o modelo vetorial foi escolhido para a estratégia LSR-VM como alternativa para estabelecer a comparação semântica entre objetos em um mecanismo de *cache.*

5.2 FUNCIONAMENTO DA ESTRATÉGIA LSR-VM

A estratégia de substituição LSR-VM propõe uma solução para os problemas presentes na estratégia de substituição LSR original, permitindo a comparação semântica da informação existente nos objetos, de forma dinâmica. O LSR-VM não necessita de nenhum tipo de solução tecnológica para obter a semântica dos objetos como, por exemplo, um servidor de semântica [SCH02]. O cálculo da distância semântica entre os objetos é obtido através da utilização do modelo vetorial. Desta forma, quando um novo objeto tiver de ser inserido em um mecanismo de *cache*, cujo limite de armazenamento tenha sido alcançado, os objetos que estiverem semanticamente mais distantes do novo objeto são progressivamente descartados até que seja liberado espaço suficiente para acomodá-lo.

Cada novo objeto a ser inserido no mecanismo de *cache* passa por um processo de transformação de texto resultando na criação de um vetor de *n* termos índice, que é utilizado como uma representação lógica do objeto original no espaço vetorial. Uma vez concluído o processo de transformação de texto, o vetor resultante não pode ser utilizado para recriar o documento original. Se houver espaço no mecanismo de *cache* para acomodar o novo objeto, este é inserido após a conclusão da transformação de texto. Caso contrário a estratégia LSR-VM determina quais objetos deverão ser descartados para liberar espaço para acomodar o novo objeto.

Quanto maior for o número de termos índice em comum entre dois objetos, maior é a probabilidade destes objetos estarem semanticamente relacionados. No modelo vetorial a distância semântica entre dois objetos é calculada através do co-seno do ângulo entre os vetores de termos índice que representam os objetos. Assim sendo, se um objeto *o1* possuir vários termos índice em comum com o objeto *o2*, e os vetores *v1* e *v2* representarem respectivamente os vetores de termos índice dos objetos *o1* e *o2*, então o co-seno do ângulo entre estes vetores tende a se aproximar do valor 1. Se estes objetos possuírem poucos termos índice em comum então o co-seno do ângulo entre os

vetores tende a se aproximar de zero. A figura 23 mostra os vetores *v1* e *v2* que representam os objetos *o1* e *o2* respectivamente. Quanto mais próximo do valor 1 o co-seno do ângulo entre estes vetores estiver, maior é a similaridade entre eles e, por conseguinte, maior é o compartilhamento de semântica.

Quando há necessidade de descartar objetos do mecanismo de *cache* para criar espaço para acomodar um novo objeto recém chegado, a estratégia LSR-VM descarta os objetos que estiverem semanticamente mais distantes do novo objeto. Para isto, a estratégia LSR-VM considera o novo objeto como sendo uma "**consulta**" de acordo com o modelo vetorial. A estratégia LSR-VM indexa todos os objetos existentes no *cache* juntamente com o novo objeto. Após a conclusão da reindexação dos objetos a estratégia LSR-VM calcula a similaridade entre o novo objeto e os objetos já existentes no *cache* através do co-seno dos ângulos entre o vetor do novo objeto e os vetores dos objetos já existentes no *cache*. Em seguida os objetos existentes no *cache* são classificados em ordem ascendente de similaridade e progressivamente descartados até que haja espaço suficiente para acomodar o novo objeto.

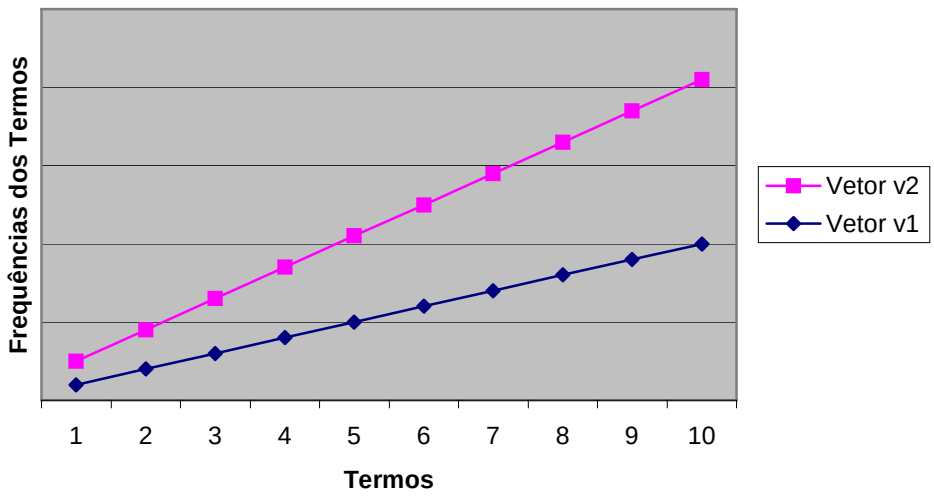


FIGURA 23 - VETORES QUE REPRESENTAM OBJETOS EM UM MECANISMO DE *CACHE* NO ESPAÇO VETORIAL.

Para facilitar o entendimento do algoritmo interno da estratégia LSR-VM foi criado um diagrama de atividades no padrão UML (*Unified Modeling Language*) que pode ser visualizado através da figura 24.

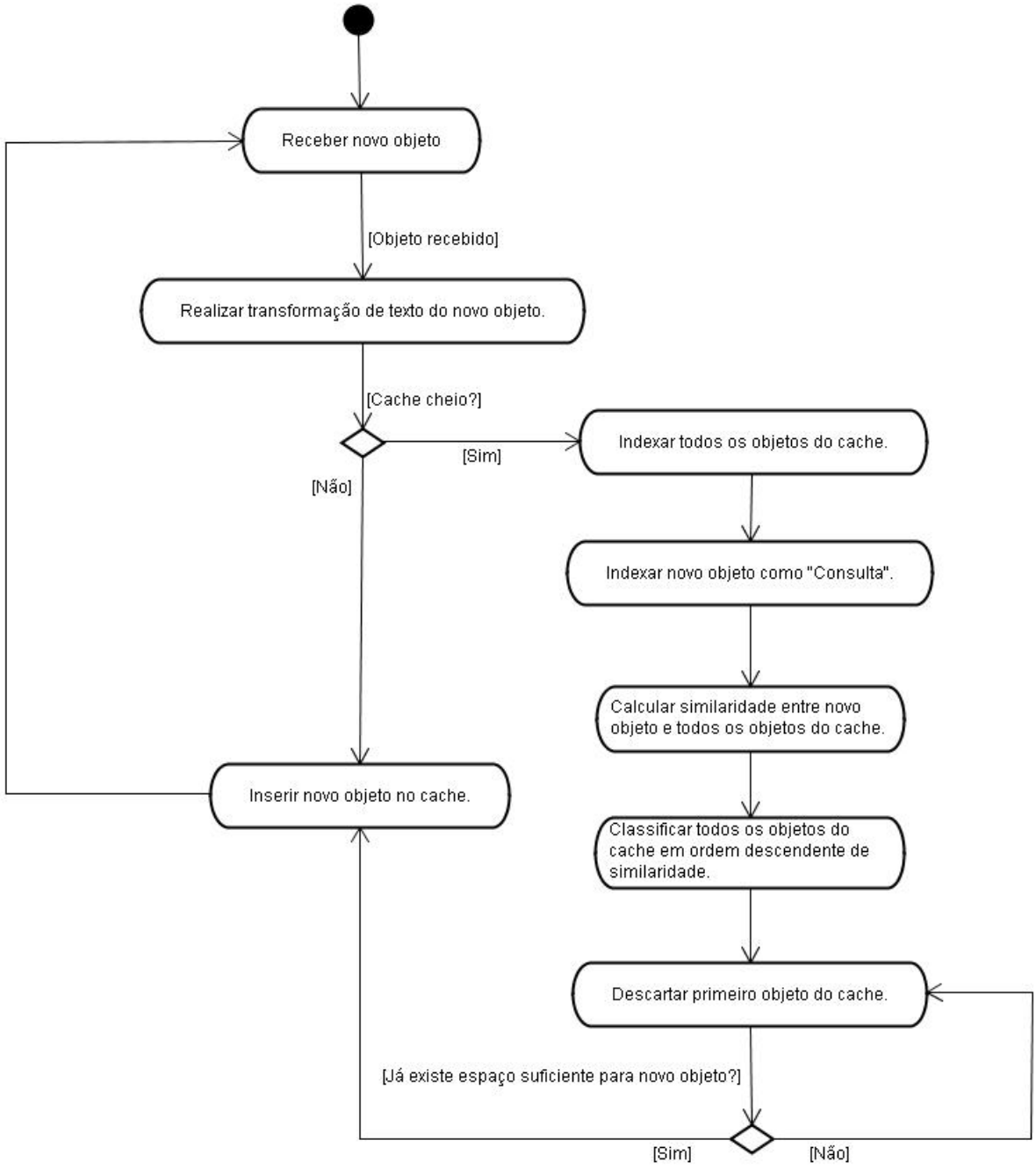


FIGURA 24 - DIAGRAMA DE ATIVIDADES QUE REPRESENTA A LÓGICA DO ALGORITMO INTERNO DA ESTRATÉGIA LSR-VM.

Através da análise da figura 24 é possível observar que toda vez que um novo objeto é inserido no *cache* quando este está cheio, a estratégia LSR-VM realiza a re-indexação de toda a coleção de objetos presentes no *cache* antes de começar a descartar objetos. A re-indexação é necessária para evitar que os valores dos pesos que compõem os vetores que representam objetos no modelo vetorial sejam distorcidos ao longo do tempo. Isto acontece porque o cálculo do peso de cada termo índice em um vetor que representa um objeto no espaço vetorial é baseado na freqüência de ocorrência deste termo no objeto e na freqüência de ocorrência deste termo índice em todos os objetos do *cache*. De acordo com o modelo vetorial o cálculo do peso é dado por *tf * idf* (“*Term Frequency*” e “*Inverse Document Frequency*”).

Este comportamento impõe um alto custo computacional à estratégia LSR-VM. Trabalhos futuros podem ser realizados com o objetivo de tornar o processo de re-indexação iterativo. Desta forma, ao invés de executar uma re-indexação sempre que há necessidade de descartar objetos quando o *cache* está cheio, este processo poderia ser realizado em intervalos predefinidos de acessos. A eficiência da estratégia LSR-VM poderia ser analisada, por exemplo, com re-indexações a cada 1000 acessos, a cada 10.000 acesso, e assim por diante.

5.3 TRATAMENTO DE IMAGENS E ARQUIVOS BINÁRIOS

Um aspecto muito importante que foi considerado durante a elaboração da estratégia LSR-VM refere-se ao tratamento de objetos dos quais não se pode extrair semântica de forma direta. Exemplos destes tipos de objeto são: imagens, arquivos de vídeo, arquivos de som, *applets*, entre outros. Este aspecto não foi considerado por nenhuma das estratégias de substituição em *cache* baseadas em semântica apresentadas no capítulo 4.

O modelo vetorial trata apenas documentos ou objetos no formato texto, não sendo possível estabelecer a similaridade entre objetos binários de forma direta. A

estratégia LSR-VM não despreza objetos binários durante as comparações semânticas, dando o seguinte tratamento para esta situação:

Quando um cliente requisita um documento HTML o navegador realiza a recuperação de diversos objetos para compor o resultado final. A primeira tarefa realizada pelo navegador é recuperar o documento HTML que contém referências para outros objetos como, por exemplo, imagens e arquivos de formatação (*Cascading Style Sheets* e *Java Scripts*). O navegador então inicia o processo de recuperação destes objetos até que seja possível criar o resultado visual que será finalmente apresentado ao cliente.

Analisando este comportamento é possível observar que os documentos HTML sempre são incluídos no *cache* antes dos objetos binários que são referenciados por estes. A estratégia LSR-VM explora esta característica para dar tratamento aos objetos binários.

Durante a tarefa de transformação de texto de um documento HTML, de acordo com o modelo vetorial, a estratégia LSR-VM armazena as referências para todos os objetos binários referenciados no documento HTML. Com isto é possível realizar uma referência cruzada entre os objetos texto e os objetos binários no mecanismo de *cache*.

Como parte do processo de seleção de objetos a serem descartados para liberar espaço para acomodar um novo objeto no mecanismo de *cache*, a estratégia LSR-VM realiza o cálculo da similaridade entre o novo objeto e os demais objetos armazenados no *cache*. A estratégia LSR-VM inicialmente calcula a similaridade apenas entre objetos texto. Em seguida a referência cruzada entre objetos texto e objetos binários é utilizada para atribuir a similaridade aos objetos binários. A estratégia LSR-VM calcula a similaridade de um objeto binário através da média das similaridades dos objetos texto que referenciam este objeto binário. A figura 25 apresenta um diagrama de atividades no padrão UML, que mostra a lógica do algoritmo interno da estratégia LSR-VM contemplando o tratamento de arquivos

binários. Com isto é possível estabelecer a similaridade entre todos os objetos existentes no mecanismo de *cache*. O estudo da melhor função para atribuir similaridade aos objetos binários no mecanismo de *cache* deve ser tema de trabalhos futuros.

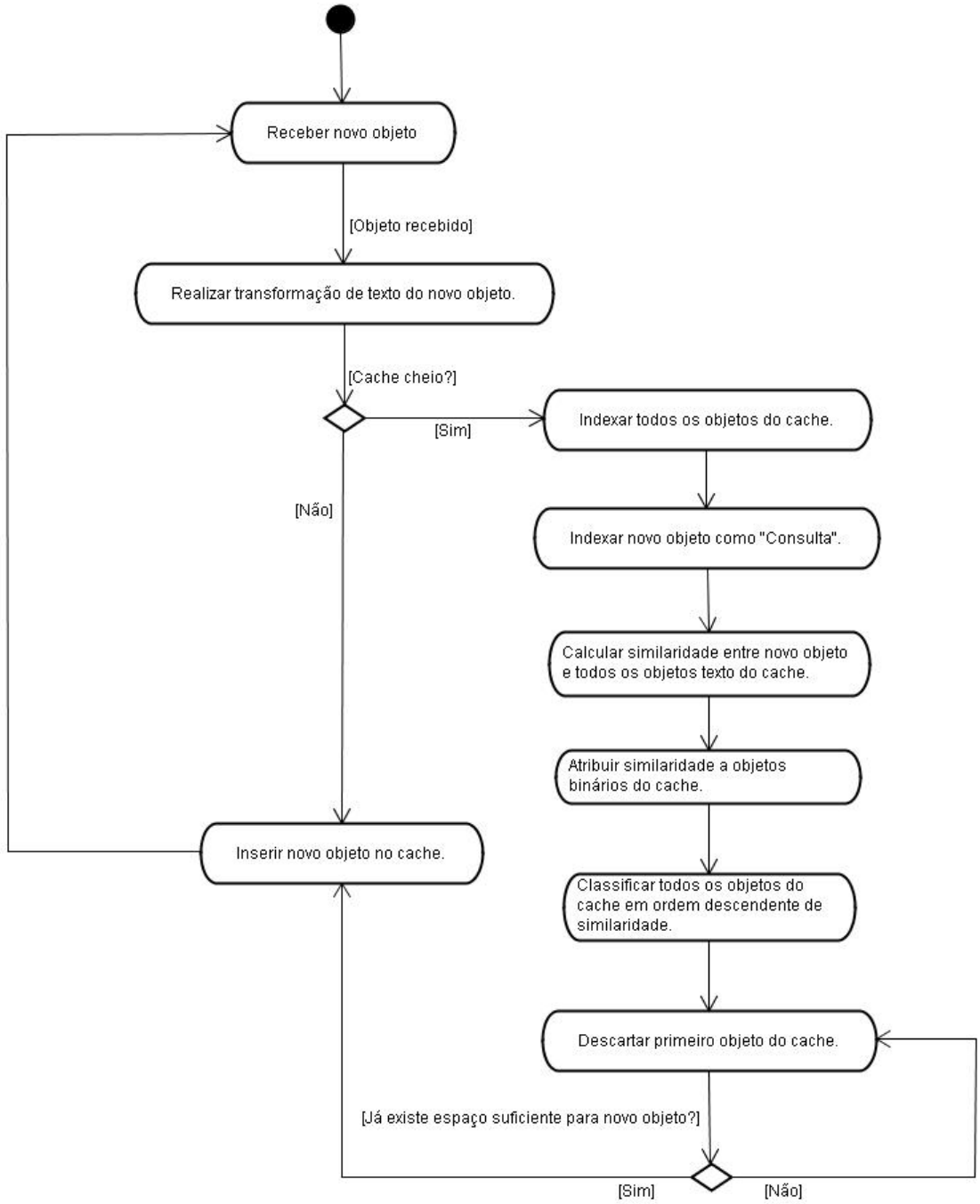


FIGURA 25 - DIAGRAMA DE ATIVIDADES QUE REPRESENTA A LÓGICA DO ALGORITMO INTERNO DA ESTRATÉGIA LSR-VM CONTEMPLANDO OBJETOS BINÁRIOS.

6. SIMULAÇÃO

6.1 ESTRATÉGIA

A avaliação do desempenho das estratégias de substituição em *cache* LRU (*Least Recently Used*), LFU (*Least Frequently Used*) e SIZE em relação a estratégia LSR-VM (*Least Semantically Related – Vector Model*) foi realizada através da utilização de um *framework* de simulação que foi desenvolvido como parte dos trabalhos que resultaram nesta dissertação. Este *framework* foi criado para facilitar a implementação de estratégias de substituição em *cache*.

Arquivos de *log* do servidor *proxy Squid* foram utilizados para simular as seqüências de acessos realizadas a objetos na Internet através de um mecanismo de *cache*, durante um determinado período. Mais especificamente foram utilizados arquivos “access.log” gerados por servidores *proxy Squid* durante seu funcionamento real. Este arquivo de *log* contém informações a respeito das requisições de objetos submetidas ao mecanismo de *cache* por clientes de uma comunidade. Com isto, buscou-se recriar situações mais próximas do funcionamento real de um mecanismo de *cache*. O capítulo 3 apresenta detalhes sobre o servidor *proxy Squid*, juntamente com a estrutura do registro de *log* do arquivo “access.log”.

O *framework* de simulação recebe como entrada um arquivo contendo registros no formato do arquivo “access.log”. Cada registro lido do arquivo de entrada pelo *framework* de simulação é tratado com se fosse uma requisição realizada por um usuário a um objeto na Internet através de um mecanismo de *cache*. Quando o repositório de armazenamento do mecanismo de *cache* atinge seu limite, a estratégia de substituição em vigor é acionada para liberar o espaço necessário para acomodar novos objetos. O *framework* de simulação gera um arquivo de saída no mesmo formato do arquivo de entrada, porém o conteúdo do campo *result-code* dos registros é

modificado para refletir os resultados obtidos a partir da simulação. Este comportamento foi implementado no *framework* de simulação para permitir a utilização de programas externos para analisar as *logs* do *proxy Squid*, apresentando resultados diversos a respeito da performance do mecanismo de *cache* e do perfil da comunidade de clientes. Estes programas realizam suas análises essencialmente utilizando o arquivo "access.log". Para a obtenção dos resultados apresentados no capítulo 7 formam utilizados os programas *Calamaris*[3] e *Squid-Graph*[4]. A figura 26 apresenta as entradas e saídas do processo realizado pelo *framework* de simulação.

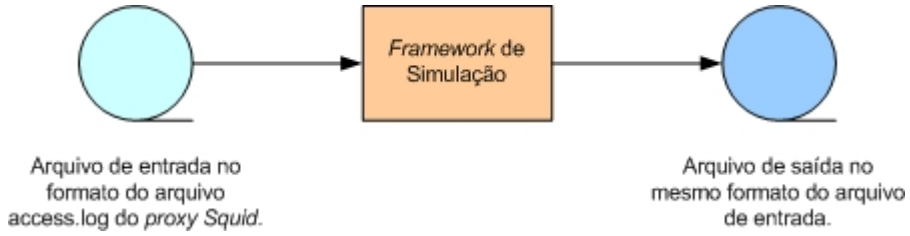


FIGURA 26 - ENTRADAS E SAÍDAS DO PROCESSO REALIZADO PELO *FRAMEWORK* DE SIMULAÇÃO.

6.2 *FRAMEWORK* DE SIMULAÇÃO

Para avaliar o desempenho da estratégia de substituição em *cache* LSR-VM (*Least Semantically Related – Vector Model*) foi desenvolvido um *framework* de simulação com o objetivo de estabelecer um comparativo com as estratégias tradicionais de substituição em *cache* LRU (*Least Recently Used*), LFU (*Least Frequently Used*) e SIZE. Para que este objetivo pudesse ser alcançado o *framework* de simulação foi desenvolvido de forma a permitir a implementação e o teste de desempenho de várias estratégias de substituição explorando os princípios de reuso, modularidade, hierarquia e encapsulamento da orientação a objetos. Para isto foi

[3] http://cord.de/tools/squid/calamaris/Welcome.html.en

[4] http://squid-graph.securlogic.com/

utilizada a linguagem de programação Java na construção do *framework* de simulação, facilitando inclusive a portabilidade para vários sistemas operacionais. Foram utilizados conceitos de engenharia de software e *design patterns* na modelagem do *framework* de simulação para facilitar futuras manutenções. A figura 27 apresenta um diagrama de classes no padrão UML (*Unified Modeling Language*) contendo os principais pacotes do *framework* de simulação. O *framework* de simulação utiliza um gerenciador de banco de dados relacional para armazenar informações durante a simulação. Foi utilizado o banco de dados MySQL[5], que é distribuído sob licença *Open Source*, para dar suporte a simulação. Todo o acesso ao banco de dados é realizado através de uma camada de persistência implementada pelo *framework Hibernate*[6]. O *framework Hibernate* implementa uma camada que encapsula todo o acesso aos dados persistentes da aplicação, abstraindo o acesso ao banco de dados. Este *framework* disponibiliza recursos para recuperação, armazenamento e exclusão de instâncias de classes da aplicação, dando o suporte adequado a heranças e associações entre classes.

As seções subseqüentes apresentam maiores detalhes sobre os principais pacotes do *framework* de simulação.

[5] http://www.mysql.com

[6] http://www.hibernate.org

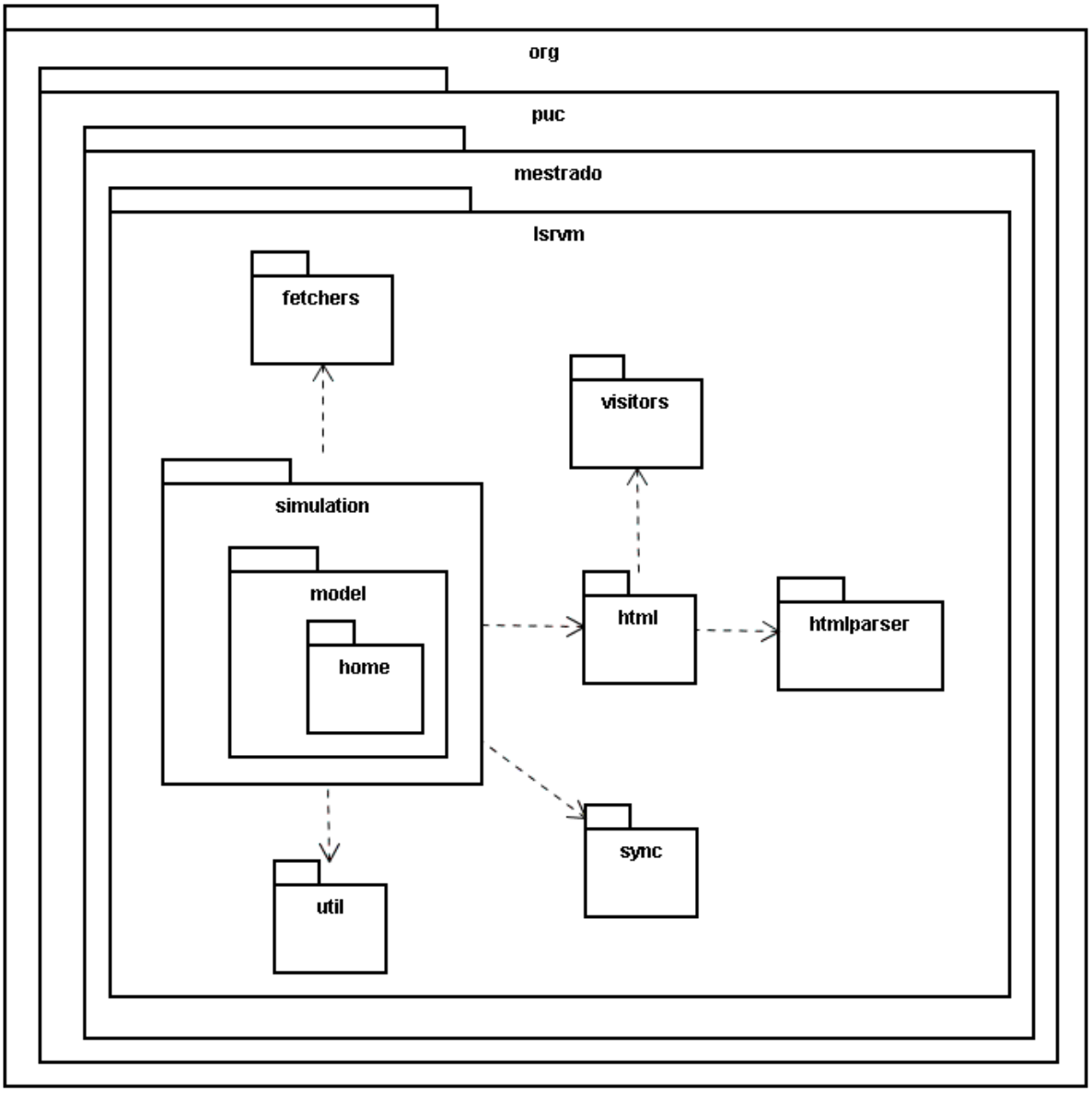


FIGURA 27 - DIAGRAMA COM OS PRINCIPAIS PACOTES DO *FRAMEWORK* DE SIMULAÇÃO.

6.2.1 Pacote "*model*"

O pacote *model* contém todas as classes que representam as entidades do *framework*. As associações entre as classes de entidade podem ser vistas através da análise do diagrama de classes no padrão UML (*Unified Modeling Language*) apresentado na figura 28.

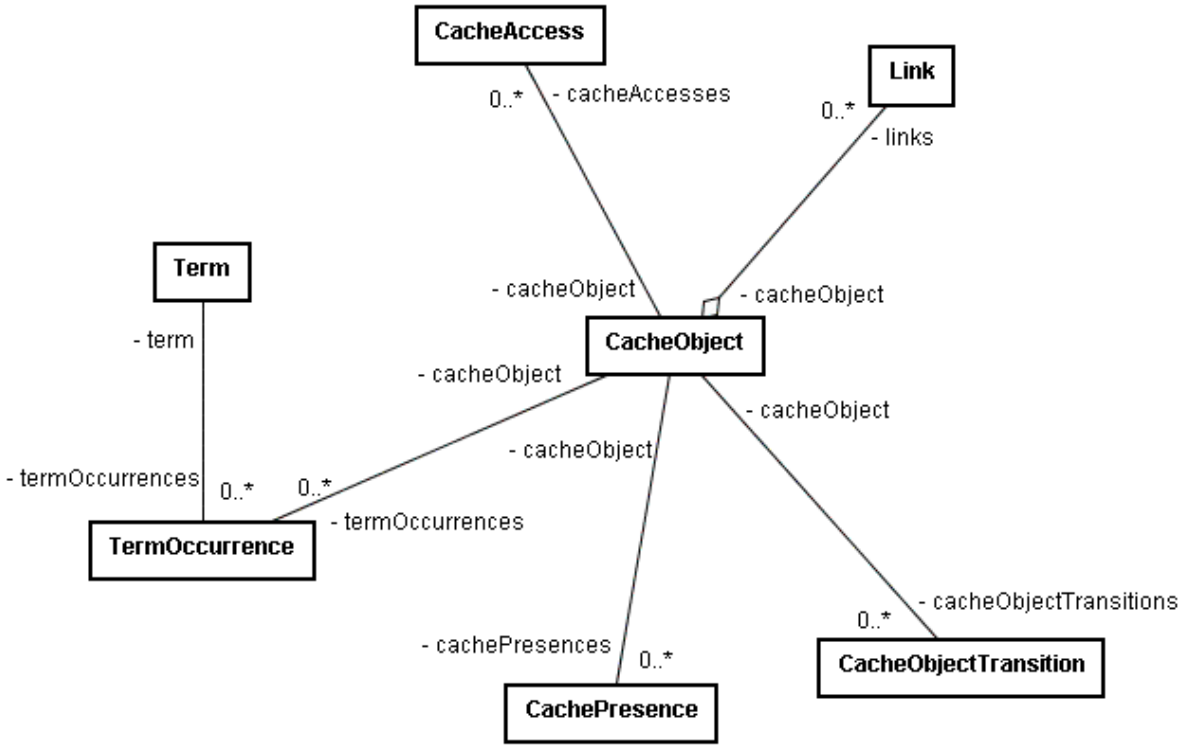


FIGURA 28 - DIAGRAMA DE CLASSES DO PACOTE *MODEL*.

A seguir são apresentadas as descrições das classes do pacote *model*:

- ***CacheAccess***: Cada instância desta classe representa um acesso ao mecanismo de *cache* realizado por um cliente para requisição de um objeto na Internet. Os atributos desta classe, juntamente com os atributos da classe *CacheObject*, correspondem aos campos que fazem parte de um registro do arquivo “access.log”, gerado como *log* pelo servidor *proxy Squid*. As descrições dos campos do registro do arquivo “access.log” podem ser encontradas no capítulo 3.
- ***CacheObject***: Cada instância desta classe representa um objeto que é gerenciado pelo *cache*. Os objetos são requisitados por clientes através de um servidor *proxy*. Os atributos desta classe, juntamente com os atributos da classe *CacheAccess*, correspondem aos campos que fazem parte de um registro do arquivo “access.log”, gerado como *log* pelo servidor *proxy Squid*. As descrições dos campos do registro

do arquivo “access.log” podem ser encontradas no capítulo 3. Os atributos desta classe correspondem às características específicas de objetos requisitados através do servidor *proxy*. Um mesmo objeto pode ser requisitado mais de uma vez, inclusive por clientes diferentes. Desta forma, uma instância de *CacheObject* pode estar associada a várias instâncias de *CacheAccess*.

- ***CacheObjetTransition***: Uma instância desta classe representa uma transição de um objeto para dentro ou para fora do repositório de armazenamento do *cache*. Cada vez que um objeto entra no *cache* o *framework* de simulação gera uma instância desta classe para registrar a transição. O mesmo acontece quando um objeto é retirado do *cache* para dar lugar a novos objetos. É importante observar que um mesmo objeto pode ser inserido no *cache,* descartado por alguma estratégia de substituição e posteriormente inserido novamente no *cache*. Um dos atributos de *CacheObjectTransition* indica a estratégia de substituição relacionada à transição. Isto se deve ao fato de que o *framework* possibilita a simulação de mais de uma estratégia de substituição em uma mesma sessão de simulação. Toda vez que ocorre a transição de um objeto para dentro ou para fora do *cache* o *framework* de simulação altera a situação do objeto que é definida na classe *CachePresence*.
- ***CachePresence***: Uma instância desta classe representa a presença de um objeto no *cache* em um determinado momento para uma estratégia de substituição. Esta classe é utilizada durante a simulação para indicar se um determinado objeto está no *cache*. O *framework* de simulação permite que sejam realizadas simulações simultâneas de diferentes estratégias de substituição. Desta forma, podem acontecer situações onde, em um determinado momento, o mesmo objeto esteja

no *cache* para uma estratégia de substituição, mas não esteja no *cache* para outra. Uma instância de *CacheObject* pode estar associada a muitas instâncias de *CachePresence*. O número de instâncias de *CachePresence* associadas a uma instância de *CacheObject* corresponde ao número de estratégias de substituição que estiverem fazendo parte de uma sessão de simulação.

- ***Link***: Uma instância desta classe representa um *link* encontrado em um objeto. Este *link* corresponde a referências encontradas em um objeto para outro objeto. Isto acontece principalmente em documentos HTML que possuem referências para outros objetos como, por exemplo, imagens, arquivos de formatação (CSS – *Cascading Style Sheet*), arquivos de som, etc. Instâncias desta classe somente são criadas quando a estratégia de substituição LSR-VM está sendo avaliada em uma sessão de simulação. Quando um novo objeto texto é requisitado ao mecanismo de *cache* a estratégia LSR-VM percorre o objeto, após recuperá-lo, identificando todas as referências para outros objetos. Estas referências são representadas através de instâncias da classe *Link*.
- ***Term***: Esta classe representa os termos índice do modelo vetorial. Os termos índices são palavras que são extraídas do conteúdo dos objetos texto, que são armazenados no mecanismo de *cache*. Os termos índice são utilizados pelo modelo vetorial para cálculo da similaridade entre um novo objeto que está chegando ao *cache* e os demais objetos que já estão no *cache*. Instâncias desta classe somente são criadas quando a estratégia LSR-VM está sendo avaliada em uma sessão de simulação.
- ***TermOccurrence***: Esta classe representa a ocorrência de um termo índice em um objeto. Instâncias desta classe somente são criadas

quando a estratégia LSR-VM está sendo avaliada em uma sessão de simulação.

6.2.2 Pacote “*home*”

As classes que pertencem ao pacote *home* são responsáveis por gerenciar a persistência das instâncias de classes de entidade do *framework* que estão presentes no pacote *model*. Estas classes implementam um padrão de projeto, do inglês, *Design Pattern* denominado *Factory*. Para cada classe de entidade do pacote *model* existe uma classe correspondente no pacote *home*. As classes do pacote *home* são responsáveis pela criação, recuperação ou exclusão de instâncias das respectivas classes de entidade no pacote *model*. A figura 29 apresenta um diagrama de classes no padrão UML para representar o relacionamento entre a classe de entidade *CacheObject* e sua respectiva classe “*home*” *CacheObjectHome*. Através análise da figura 29 é possível observar que existe um relacionamento de dependência entre a classe *CacheObject* (*model*) e a classe *CacheObjectHome* (*home*).

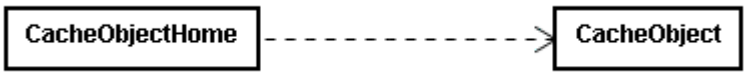


FIGURA 29 - DIAGRAMA DE CLASSES REPRESENTADO O RELACIONAMENTO ENTRE UMA CLASSE DE ENTIDADE E SUA RESPECTIVA CLASSE *HOME*.

As classes “*home*” encapsulam todo o acesso ao sistema gerenciador de banco de dados através da utilização do *framework* de persistência *Hibernate*. Este *framework* permite o mapeamento entre o modelo de objetos e o modelo de dados através de um arquivo XML. Neste arquivo são armazenadas todas as configurações que definem como o *framework Hibernate* deve persistir as instâncias das classes de entidade em tabelas do banco de dados. A figura 30 apresenta um diagrama de classes no padrão UML (*Unified Modeling Language*) contendo todas as classes do pacote

home.

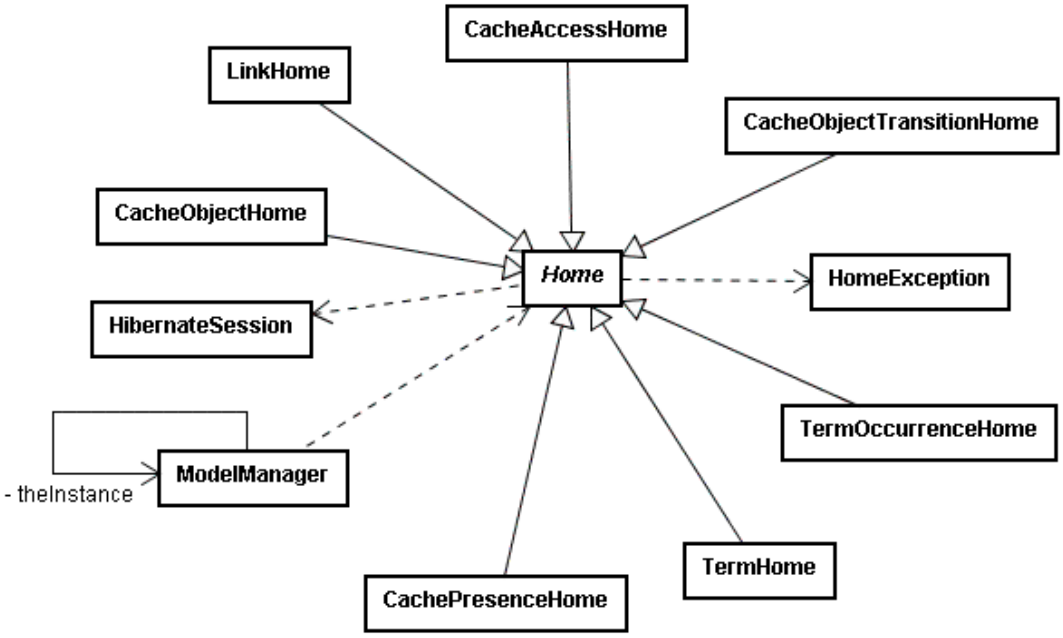


FIGURA 30 - DIAGRAMA DE CLASSES DO PACOTE *HOME*.

A seguir são apresentadas as descrições das classes do pacote *model*:

- ***CacheAccessHome***: Classe responsável pela persistência de instâncias da classe *CacheAccess*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***CacheObjectHome***: Classe responsável pela persistência de instâncias da classe *CacheObject*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***CacheObjectTransitionHome***: Classe responsável pela persistência de instâncias da classe *CacheObjectTransition*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***CachePresenceHome***: Classe responsável pela persistência de instâncias da classe *CachePresence*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***LinkHome***: Classe responsável pela persistência de instâncias da

classe *Link*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.

- ***TermHome***: Classe responsável pela persistência de instâncias da classe *Term*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***TermOccurrenceHome***: Classe responsável pela persistência de instâncias da classe *TermOccurrence*. Esta classe implementa o padrão de projeto (*Design Pattern*) *Factory*.
- ***HibernateSession***: Esta classe é responsável por gerenciar uma sessão do *framework Hibernate*. Todo acesso a instâncias persistentes deve ser realizado através de um objeto do próprio *framework Hibernate* denominado *Session*.
- ***Home***: Esta classe possui as operações básicas para persistência de objetos através do *framework Hibernate*. Todas as classes *home* fazem herança desta classe estendendo suas funcionalidades.
- ***HomeException***: Esta classe é utilizada no tratamento de exceções eventualmente geradas por classes do pacote *home*, indicando problemas ocorridos durante a persistência de instâncias do pacote *model*.
- ***ModelManager***: Esta classe implementa os padrões de projeto (*Design Pattern*) *Factory* e *Singleton*. Esta classe possui apenas uma instância em toda a aplicação, que é responsável por retornar instâncias de classes *home*. Para se obter uma instância da classe *CacheAccess* previamente persistida através do *framework Hibernate*, é necessário primeiro solicitar uma instância de *CacheAccessHome* à instância única (*Singleton*) da classe *ModelManager*. A partir da instância de *CacheAccessHome* obtida, é possível recuperar uma instância de *CacheAccess*.

6.2.3 Pacote “s*imulation*”

O pacote *simulation* contém as principais classes que compõem a estrutura do *framework* de simulação. A classe *CacheSimulator* é responsável por gerenciar sessões de simulação. Uma sessão de simulação corresponde à leitura de um arquivo de *log* gerado por um servidor *proxy* para recriar as requisições de objetos realizadas por uma comunidade de clientes ao longo do tempo. Esta tarefa é realizada por uma instância da classe *CacheLogReader*, que está associada a uma instância de *CacheSimulator*. Cada registro lido a partir do arquivo de *log* de entrada é tratado como uma requisição a um objeto através do mecanismo de *cache*. Ao receber uma requisição, a instância de *CacheSimulator* verifica se o objeto requisitado já está no *cache*. O gerenciamento do repositório de armazenamento do *cache* é realizado por uma instância da classe *CacheStorage*. Se o repositório possuir espaço o novo objeto é simbolicamente armazenado no *cache* sensibilizando o limite de armazenamento gerenciado pela instância de *CacheStorage*. Se não houver espaço para o novo objeto a instância de *CacheSimulator* invoca a estratégia de substituição para que esta libere o espaço suficiente para armazenar o novo objeto. As estratégias de substituição são implementadas pelas instâncias das classes *Size*, *Lru*, *Lfu* e *Lsrvm*, que realizam a interface *ReplacementAlgorithm*. Para liberar espaço no repositório do *cache*, instâncias das classes que implementam as estratégias de substituição interagem com a instância de *CacheStorage*. As requisições processadas são gravadas em um arquivo de saída no mesmo formato do arquivo de *log* de entrada para que possam ser analisadas por ferramentas externas. A gravação do arquivo de saída é realizada por uma instância da classe *CacheLogWriter*, que está associada a instância de *CacheSimulator*. A figura 31 apresenta um diagrama de classes no padrão UML (*Unified Modeling Language*) contendo as principais classes do pacote *simulation*.

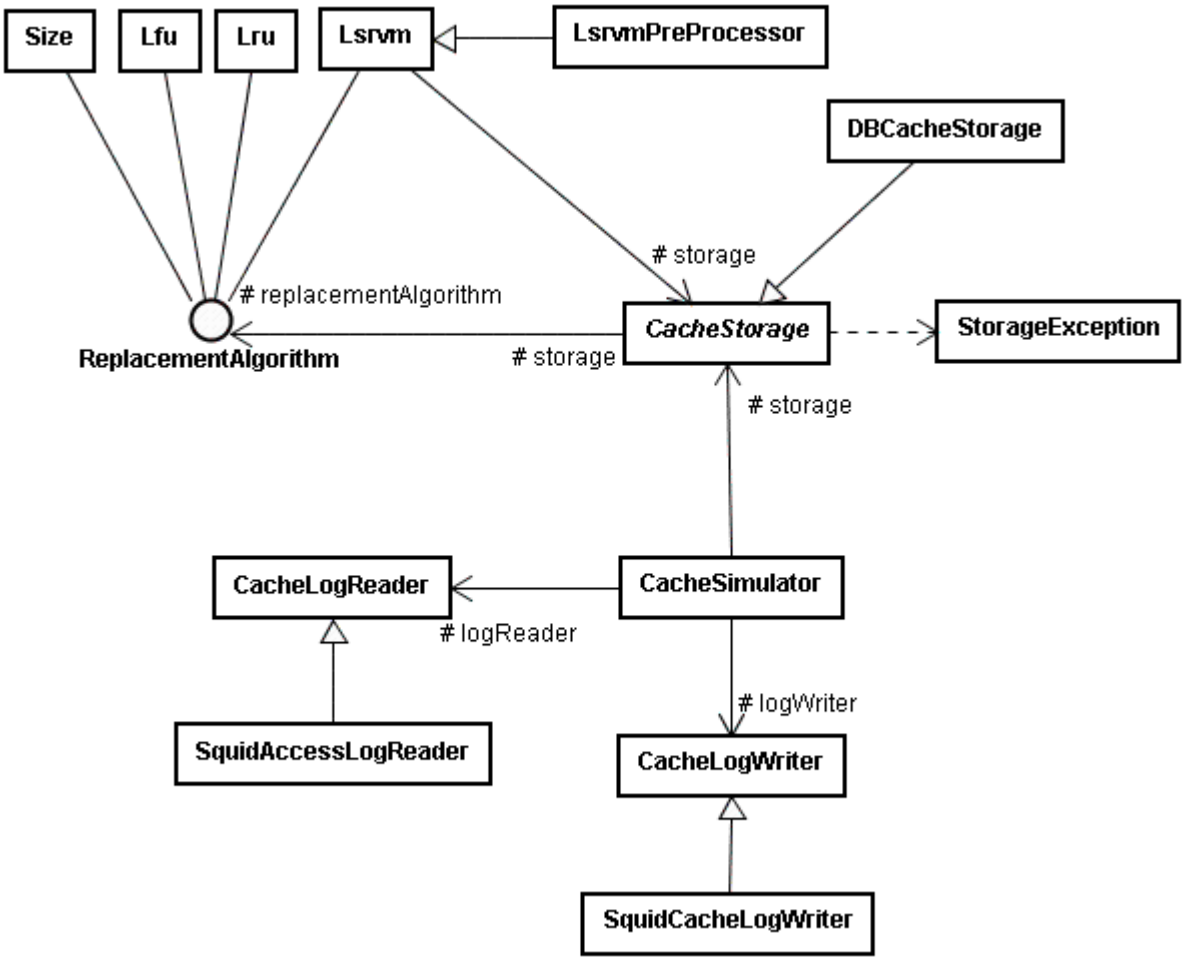


FIGURA 31 - DIAGRAMA DE CLASSES DO PACOTE *SIMULATION*.

A seguir são apresentadas as descrições das classes do pacote *simulation*:

- ***CacheLogReader***: Esta classe é responsável por ler um arquivo de *log* gerado por um mecanismo de *cache*. Este arquivo de *log* serve como entrada para o *framework* de simulação recriando as requisições de acesso realizadas a um servidor *proxy* por um grupo de clientes durante um determinado intervalo de tempo. A classe *SquidAccessLogReader* estende esta classe para ler o arquivo "access.log" gerado pelo servidor *proxy Squid*. Esta classe implementa a interface *Iterator* do J2SE (*Java 2 Standard Edition*), desta forma é possível percorrer o arquivo de *log* da mesma forma utilizada para percorrer os elementos de uma coleção na linguagem de

programação Java.

- ***CacheLogWriter***: Esta classe é responsável por gerar o arquivo de *log* resultante de uma sessão de simulação. Este arquivo possui o mesmo formato do arquivo de entrada lido pela classe *CacheLogReader*. A classe *SquidCacheLogWriter* estende esta classe para gerar um arquivo no mesmo formato do arquivo "access.log" gerado pelo servidor *proxy Squid*.
- ***CacheSimulator***: Esta classe é responsável pelo gerenciamento de uma sessão de simulação. A lógica principal do *framework* de simulação está encapsulada nesta classe. Esta classe possui associações com instâncias da classe *CacheStorage* e com instâncias das classes que realizam a interface *ReplacementAlgorithm*, responsáveis pelo repositório de armazenamento do *cache* e pela estratégia de substituição respectivamente.
- ***CacheStorage***: Esta classe representa o repositório de armazenamento do *cache* durante uma sessão de simulação. Esta é uma classe abstrata e por isto não pode ser instanciada diretamente. Na verdade a classe *DBCacheStorage* estende a classe *CacheStorage* implementando as operações necessárias para complementar as funcionalidades do repositório de objetos de um mecanismo de *cache*. Esta classe possui associação com instâncias de classes que realizam a interface *ReplacementAlgorithm*, responsável pela estratégia de substituição. Estas instâncias interagem quando há necessidade de excluir objetos do *cache*.
- ***DBCacheStorage:*** Esta classe estende a classe *CacheStorage* para implementar um repositório de armazenamento do *cache* que utiliza um sistema de gerenciamento de banco de dados para armazenar informações durante as sessões de simulação.

- ***Lfu:*** Esta classe realiza a interface *ReplacementAlgorithm* para implementar a estratégia de substituição LFU (*Least Frequently Used*). Esta estratégia é utilizada em sessões de simulação quando é necessário eliminar objetos do *cache*.
- ***Lru:*** Esta classe realiza a interface *ReplacementAlgorithm* para implementar a estratégia de substituição LRU (*Least Recently Used*). Esta estratégia é utilizada em sessões de simulação quando é necessário eliminar objetos do *cache*.
- ***Lsrvm:*** Esta classe realiza a interface *ReplacementAlgorithm* para implementar a estratégia de substituição LSR-VM (*Least Semantically Related – Vector Model*). Esta estratégia é utilizada em sessões de simulação quando é necessário eliminar objetos do *cache*.
- ***LsrvmPreProcessor:*** Esta classe estende a classe *Lsrvm*, sendo responsável pelo pré-processamento realizado antes de sessões de simulação que utilizam a o LSR-VM como estratégia de substituição em *cache*. Durante o pré-processamento o arquivo de *log* de entrada é previamente percorrido e os objetos texto existentes são recuperados a partir de suas localizações originais na Internet. Uma vez recuperados os objetos são decompostos em termos índice que serão utilizados durante a sessão de simulação para cálculo da similaridade entre objetos no *cache*.
- ***ReplacementAlgorithm:*** Esta interface representa uma estratégia de substituição no *cache*. Esta interface foi definida para permitir o uso do conceito de polimorfismo. Desta forma, se o *framework* de simulação precisar suportar uma nova estratégia de substituição basta construir uma nova classe que realize esta interface. Atualmente a interface *ReplacementAlgorithm* é realizada pelas classes *Size*, *Lru*, *Lfu* e *Lsrvm* no *framework* de simulação.

- ***Size:*** Esta classe realiza a interface *ReplacementAlgorithm* para implementar a estratégia de substituição SIZE. Esta estratégia é utilizada em sessões de simulação quando é necessário eliminar objetos do *cache*.
- ***SquidCacheLogReader***: Esta classe é responsável por ler um arquivo de *log* de acessos gerado pelo servidor *proxy Squid*. Esta classe estende a classe *CacheLogReader*. O arquivo lido serve de entrada para uma sessão de simulação e deve apresentar o mesmo formado do arquivo "access.log" gerado pelo servidor *proxy Squid*.
- ***SquidCacheLogWriter***: Esta classe é responsável por gerar um arquivo de *logs* de acesso no mesmo formado do arquivo "access.log" do servidor *proxy Squid*. Esta classe estende a classe *CacheLogWriter* para gerar um arquivo de *log* contento o resultado de uma sessão de simulação. Este arquivo pode ser utilizado posteriormente por ferramentas externas que geram estatísticas de utilização do *cache*.

6.2.4 Pacote "html"

O pacote "html" possui classes que são responsáveis por recuperar objetos texto a partir de suas localizações originais, realizando a decomposição destes objetos em termos índice que são utilizados em sessões de simulação com a estratégia de substituição LSR-VM para cálculo da similaridade entre objetos no *cache* através do modelo vetorial. O *framework* de simulação utiliza uma biblioteca de classes distribuída sob a licença GPL (*General Public License*) para realizar a recuperação dos objetos na Internet. Esta biblioteca implementa um "*Parser HTML*" que além de recuperar o objeto a partir de sua localização original, permite que o objeto recuperado seja percorrido de forma ordenada, possibilitando a sua decomposição em termos índice. Além disso, é possível identificar referências para outros objetos que são

armazenados e tratados pela estratégia de substituição LSR-VM durante a comparação semântica de objetos binários como, por exemplo, imagens, vídeos, etc. A biblioteca utilizada é denominada *HTMLParser*[7]. A figura 32 apresenta um diagrama de classes no padrão UML (*Unified Modeling Language*) contendo as principais classes do pacote *html*.

A seguir são apresentadas as descrições das classes do pacote *html*:

- ***StringBean:*** Esta classe é responsável por recuperar um objeto texto a partir de uma URL (*Uniform Resource Locator*), percorrendo o objeto recuperado de forma ordenada. Durante o processo de análise do conteúdo do objeto recuperado esta classe cria uma lista das referências para outros objetos encontrados no objeto recuperado. O resultado deste processamento é um objeto *String* contendo o texto puro extraído do objeto recuperado. Todas as *tags* são eliminadas.
- ***StringExtractor:*** Esta classe simplesmente utiliza uma instância da classe *StringBean* para recuperar um objeto a partir da Internet, fornecendo como resultado de seu processamento uma lista dos objetos referenciados pelo objeto recuperado, bem como um objeto *String* contendo o texto puro do objeto recuperado.

[7] http://htmlparser.sourceforge.net/

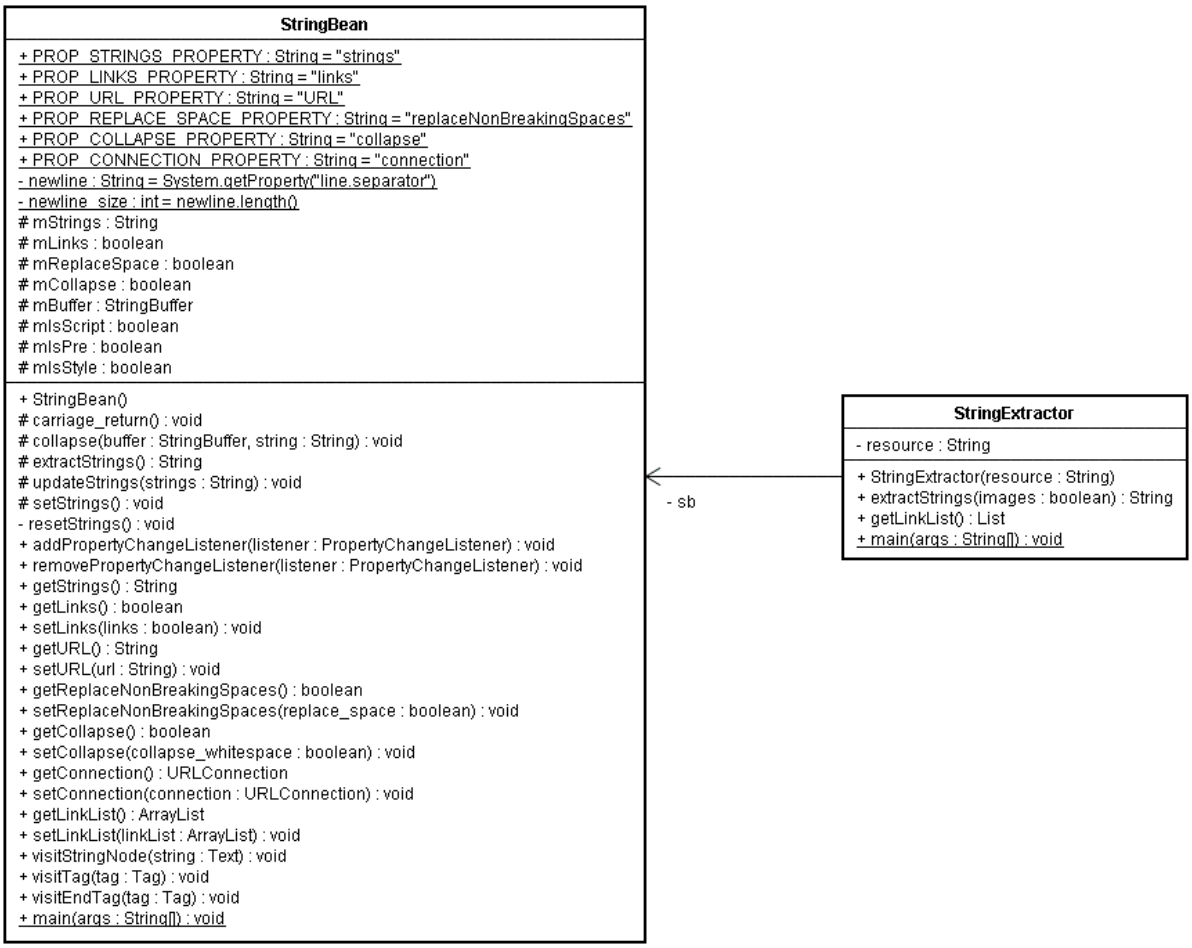


FIGURA 32 - DIAGRAMA DE CLASSES DO PACOTE *HTML*.

6.2.5 Modelo de Dados

A figura 33 apresenta o modelo de dados utilizado pelo *framework* de simulação para armazenar informações durante as simulações. Todas as tabelas presentes no modelo de dados são mapeadas em classes através da utilização do *framework* de persistência *Hibernate*. As classes resultantes do mapeamento são as classes que representam as entidades do *framework* e podem ser encontradas no pacote *model*.

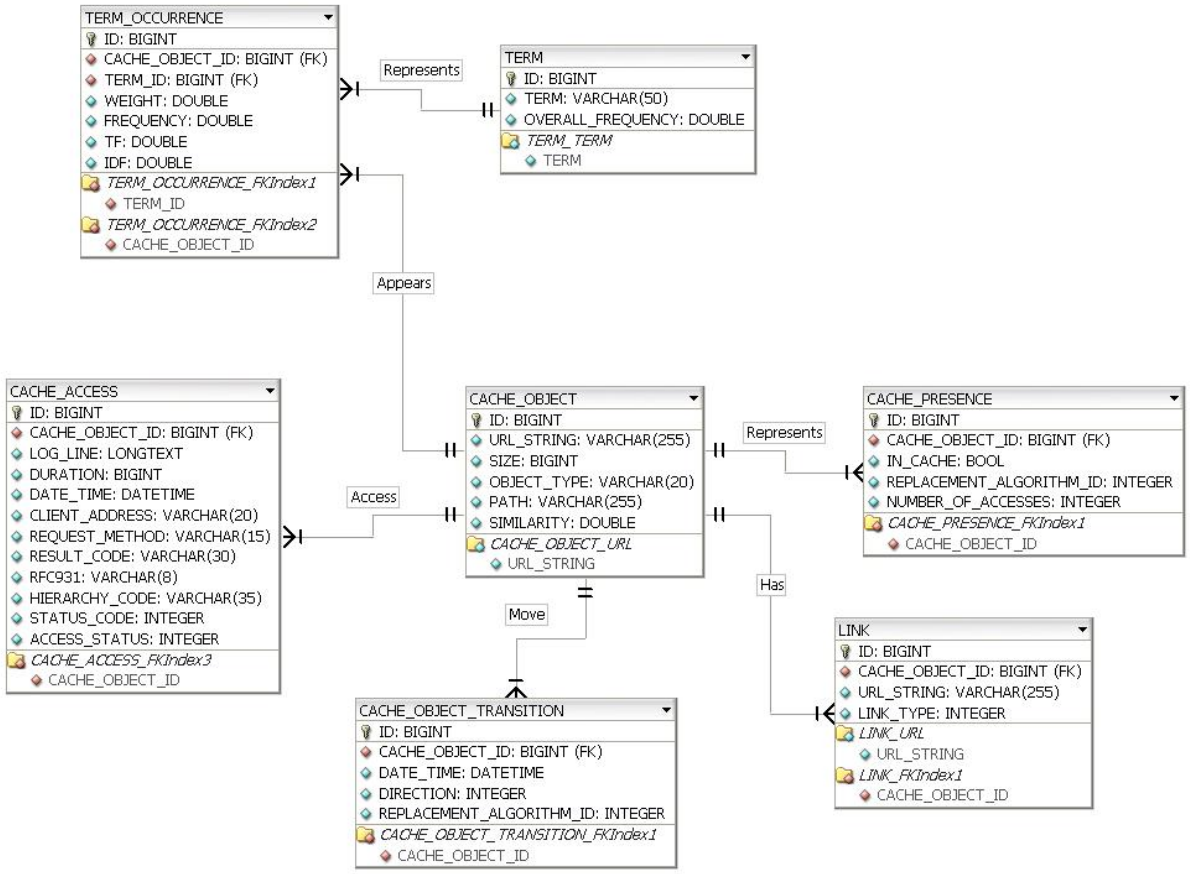


FIGURA 33 - MODELO DE DADOS UTILIZADO PELO *FRAMEWORK* DE SIMULAÇÃO.

7. RESULTADOS

7.1 COLETA DE DADOS

Durante os experimentos e trabalhos de pesquisa que resultaram nesta dissertação, foram analisados arquivos de *log* gerados por servidores *proxy Squid* das instituições de ensino Pontifícia Universidade Católica do Paraná (PUCPR) e Faculdades SPEI, localizadas na cidade de Curitiba, Paraná.

Os arquivos de *log* fornecidos pelas instituições de ensino foram utilizados com dois objetivos específicos. O primeiro objetivo foi estabelecer uma análise detalhada das características das requisições de objetos realizadas pelas comunidades de usuários de cada instituição de ensino, através de seus respectivos servidores *proxy*. O segundo objetivo foi realizar a simulação, estabelecendo um comparativo entre as estratégias de substituição em *cache*: SIZE, LRU, LFU e LSR-VM utilizando para isto, o *framework* de simulação descrito no capítulo anterior.

Ambas instituições de ensino utilizam o servidor *proxy Squid* para realização de *cache* nas respectivas redes. Os arquivos de *log* utilizados durante as pesquisas correspondem aos acessos realizados na PUCPR entre os dias 26 de junho de 2005 e 08 de julho de 2005 totalizando aproximadamente seis milhões e trezentas mil requisições, e na SPEI entre os dias 03 de julho de 2005 e 11 de julho de 2005, totalizando aproximadamente um milhão e setecentas mil requisições. Ao todo foram analisadas aproximadamente oito milhões de requisições.

Para a demonstração dos resultados referentes às características das requisições de objetos realizadas pelas comunidades de usuários de cada instituição de ensino foram separados dois lotes de requisições, que correspondem aos dias 04 de julho de 2005 e 05 de julho de 2005. Estes períodos foram escolhidos por melhor representarem as características da comunidade de usuários de ambas instituições de

ensino. Esta definição foi realizada após análise de todo o conjunto de *logs*. A tabela 7 apresenta a relação dos lotes de requisições referentes aos períodos de *logs* utilizados para demonstração de resultados.

TABELA 7 - LOTES CORRESPONDENTES AOS PERÍODOS DE *LOGS* UTILIZADOS PARA DEMONSTRAÇÃO DE RESULTADOS.

Lote	**Instituição**	**Data Inicial do *Log***	**Data Final do *Log***	**Nº de Requisições**
P-04-A	PUCPR	04/07/2005 – 00:00Hs	05/07/2005 – 00:00Hs	677.445
P-05-A	PUCPR	05/07/2005 – 00:00Hs	06/07/2005 – 00:00Hs	595.656
S-04-A	SPEI	04/07/2005 – 00:00Hs	05/07/2005 – 00:00Hs	169.942
S-05-A	SPEI	05/07/2005 – 00:00Hs	06/07/2005 – 00:00Hs	192.958

Foram extraídos lotes ainda menores, contendo aproximadamente 20.000 requisições, a partir de cada um dos lotes apresentados na tabela 7. Estes lotes foram utilizados para realizar as simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM. Estes lotes foram obtidos a partir dos períodos de picos de acessos em cada um dos lotes apresentados na tabela 7, que correspondem aos horários de 10:00 e 19:00 horas. Para facilitar a identificação dos lotes foi utilizada uma nomenclatura onde a primeira letra identifica a instituição de ensino (“P” para PUCPR e “S” para SPEI), em seguida aparece um dígito que corresponde ao dia em que as requisições foram realizadas e finalmente uma letra que define se o lote foi utilizado para análise das características da comunidade de usuários (letra “A”) ou se foi utilizado para simulação de estratégias de substituição (letra “S”). A tabela 8 apresenta os lotes correspondentes aos períodos de *log* utilizados para simulação das estratégias de substituição em *cache*.

As seções subseqüentes apresentam os resultados obtidos a partir das análises dos lotes apresentados nas tabelas 7 e 8, de acordo com os objetivos definidos no início deste capítulo.

TABELA 8 - LOTES CORRESPONDENTES AOS PERÍODOS DE *LOGS* UTILIZADOS PARA SIMULAÇÃO DAS ESTRATÉGIAS DE SUBSTITUIÇÃO EM *CACHE*.

Lote	**Instituição**	**Data Inicial do *Log***	**Nº de Requisições**
P-04-S	PUCPR	04/07/2005 – 10:00Hs	19.952
P-05-S	PUCPR	05/07/2005 – 19:00Hs	19.948
S-04-S	SPEI	04/07/2005 – 10:00Hs	19.885
S-05-S	SPEI	05/07/2005 – 19:00Hs	19.887

7.1.1 Análise do Lote P-04-A

Este lote corresponde às requisições de objetos realizadas pela comunidade de usuários da PUCPR no período entre 00:00 hora do dia 04 de julho de 2005 e 00:00 hora do dia 05 de julho de 2005. O objetivo da análise deste lote é apresentar as características de requisições da comunidade de usuários da PUCPR.

Ao todo foram computadas 677.445 requisições de objetos neste lote. Estas requisições foram originadas a partir de 1.576 endereços IP distintos, significando que o mesmo número de usuários esteve envolvido nas requisições. A maioria das requisições foi realizada através do método de requisição “GET” do protocolo HTTP. Conforme foi explicado no capítulo 2, apenas os objetos requisitados através dos métodos “GET” e “HEAD” são considerados por mecanismos de *cache*. O fato de um objeto ter sido requisitado através do método “GET” não significa necessariamente que será armazenado no *cache*, pois outros fatores deverão ser levados em conta. A partir da análise da tabela 9 é possível observar que aproximadamente 79,14% (somatória de requisições realizadas através dos métodos “GET” e “HEAD”) das requisições são passíveis de *cache*.

TABELA 9 - REQUISIÇÕES POR *REQUEST-METHO*D NO LOTE P-04-A.

request-method	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
GET	535.150	79	2,21	5.302.668K	10,15	4,49
CONNECT	118.189	17,45	88,74	45.725M	89,59	4,46
POST	22.954	3,39	1,2	137.855K	0,26	5,02
HEAD	966	0,14	0,43	333.085	0	0,78
OPTIONS	123	0,02	10,94	80.326	0	0,06
PROPFIND	58	0,01	0,61	47.333	0	1,32
PROPPATCH	3	0	1,08	2.500	0	0,75
MOVE	1	0	0,7	640	0	0,9
LOCK	1	0	899,55	2.117	0	0
Total	**677.445**	**100**	**17,27**	**51.039M**	**100**	**4,47**

As informações apresentadas na tabela 9 são:

- ***request-method***: Campo da mensagem de requisição do protocolo HTTP que define o método da requisição. Os detalhes do funcionamento do protocolo HTTP podem ser obtidos no capítulo 2.
- **Requisições**: Este campo apresenta o número de requisições realizadas através do método de requisição que aparece na coluna "***request-method"***.
- **%R**: Este campo apresenta o percentual de requisições realizadas através do método de requisição que aparece na coluna "***request-method",*** em relação a quantidade total de requisições do lote.
- **s/Req.**: Este campo apresenta o tempo médio para recuperação (latência) de um objeto requisitado através do método de requisição que aparece na coluna "***request-method"***. Esta informação é apresentada na unidade de tempo "segundos".

- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos requisitados através do método de requisição que aparece na coluna "***request-method".***
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos requisitados através do método de requisição que aparece na coluna "***request-method"***, em relação a quantidade total de *bytes* transferidos no lote.
- **KB/s**: Este campo apresenta a banda utilizada para transferência dos objetos requisitados através do método de requisição que aparece na coluna "***request-method"***. Esta informação é apresentada na unidade de medida *Kbytes* por segundo.

Apesar da maioria das requisições terem sido realizadas através do método "GET", apenas 25,97% resultaram em *cache hit*, conforme pode ser observado na tabela 10, que apresenta as requisições classificadas por *result-code*, e na figura 34 que apresenta um gráfico de requisições no lote P-04-A.

TABELA 10 - REQUISIÇÕES POR *RESULT-CODE* NO LOTE P-04-A.

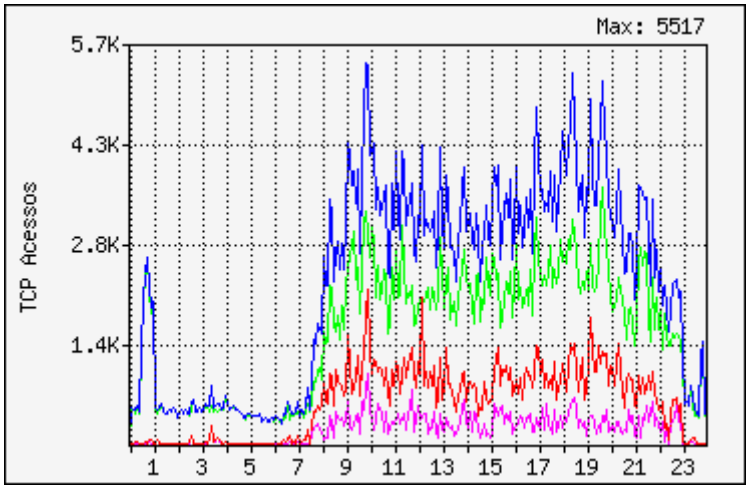
result-code	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
HIT	175.961	25,97	0,24	504.996K	0,97	12,2
MISS	469.462	69,3	24,54	50.479M	98,9	4,49
ERROS	32.022	4,73	4,39	70.320.263	0,13	0,49
Total	**677.445**	**100**	**17,27**	**51.039M**	**100**	**4,47**

As informações apresentadas na tabela 10 são:

- ***result-code***: Este campo é registrado no *log* do *proxy Squid* para cada requisição realizada. A tabela 10 apresenta as quantidades de requisições separadas nas seguintes categorias:

  - HIT: Requisições que resultaram em acerto no *cache*;
  - MISS: Requisições cujos objetos tiveram de ser recuperados a partir de seu local de origem;
  - ERRORS: Requisições que resultaram em erro. Um erro pode ser considerado, por exemplo, como uma requisição que não foi atendida por falta de autorização de acesso ao objeto.

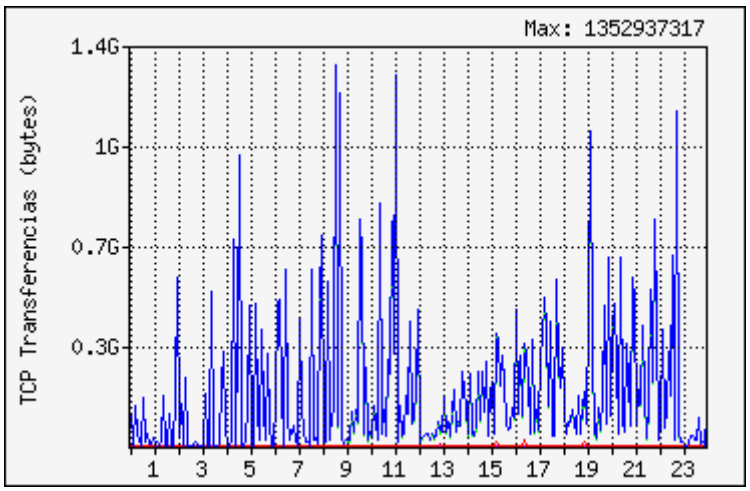
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna "***result-code"***.
- **%R**: Este campo apresenta o percentual de requisições que resultaram na categoria que aparece na coluna "***result-code",*** em relação a quantidade total de requisições do lote.
- **s/Req.**: Este campo apresenta o tempo médio para recuperação (latência) de um objeto cuja requisição resultou na categoria que aparece na coluna "***result-code".*** Esta informação é apresentada na unidade de tempo "segundos".
- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna "***result-code".***
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos cuja requisição resultou na categoria que aparece na coluna "***result-code"***, em relação a quantidade total de *bytes* transferidos no lote.
- **KB/s**: Este campo apresenta a banda utilizada para transferência dos objetos cujas requisições resultaram na categoria que aparece na coluna "***result-code"***. Esta informação é apresentada na unidade de medida *Kbytes* por segundo.



Total de Acessos: 677445
Média de Acessos: 28226,75 por hora
Total de *Cache Hits*: 175961
Média de *Cache Hits*: 7331,7 por hora
% de *Cache Hits*: 25,97 %
Total de *Cache IMS Hits*: 63230
Média de *Cache IMS Hits*: 2634,58 por hora
Total de *Cache Misses*: 474903
Média de *Cache Misses*: 19787,62 por hora
% de *Cache Misses*: 70,1 %

FIGURA 34 - GRÁFICO DE REQUISIÇÕES NO LOTE P-04-A.

O tamanho total dos objetos requisitados no lote P-04-A corresponde a 51.039 MB, sendo que o total de banda salva pelas requisições que resultaram em *cache hit* corresponde a 493 MB. Isto significa que a taxa de acerto no *cache*, em relação ao tamanho dos objetos (*byte hit rate*) corresponde a apenas 0,97% no lote P-04-A. As figuras 34 e 35 apresentam o número de requisições que resultaram em *cache IMS (if-modified- since) hit* para o lote P-04-A.



Total Transferido: 51,0 GB
Média Transferida: 2,2 GB por hora
Total de *Cache Hits*: 493,1 MB
Média de *Cache Hits*: 21,5 MB por hora
% de *Cache Hits*: 0,97 %
Total de *Cache IMS Hits*: 18,1 MB
Média de *Cache IMS Hits*: 754,4 Kb por hora
Total de *Cache Misses*: 50,5 GB
Média de *Cache Misses*: 2,2 GB por hora
% de *Cache Misses*: 97,95 %

FIGURA 35 - GRÁFICO DE *BYTES* TRANSFERIDOS NO LOTE P-04-A.

Uma análise mais detalhada das requisições classificadas por *content-type* pode ser observada na tabela 11. Teoricamente, o conteúdo do campo *content-type*

deveria indicar se o objeto que está sendo retornado é um objeto texto ou um objeto binário. Porém, durante a construção do *framework* de simulação verificou-se que nem sempre os servidores *web* retornam um valor confiável no campo *content-type*. Muitas vezes objetos binários são retornados com valor “text/html” no campo *content-type*.

A linha da tabela 11 que apresenta o valor “<erros>” como *content-type* corresponde a requisições do lote P-04-A que foram realizadas aos serviços de *Messenger*, a requisições de aplicativos para compartilhamento de arquivos ponto-a-ponto como, por exemplo, o *bit-torrent*, ou finalmente a erros de acessos a endereços não autorizados.

Ainda a partir da análise da tabela 11 é possível observar que grande parte das requisições refere-se a objetos binários, mais especificamente a imagens. É possível observar que 24,33% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “gif”. Da mesma forma, é possível observar que 10,76% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “jpeg”.

TABELA 11 - REQUISIÇÕES POR *CONTENT-TYPE* NO LOTE P-04-A.

content-type	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
image/gif	164.834	24,33	60,19	298.489K	0,57
<erros>	130.914	19,32	4,67	67.801.569	0,13
text/html	120.141	17,73	3,17	994.004K	1,9
image/jpeg	72.914	10,76	35,07	773.830K	1,48
<desconhecidas>	46.274	6,83	11,51	680.741K	1,3
application/x-javascript	29.848	4,41	36,97	146.528K	0,28
<seguras> (HTTPS)	26.339	3,89	0	45.723M	89,59
application/octet-stream	18.249	2,69	0,97	633.429K	1,21
aim/http	14.405	2,13	0	5.056.895	0,01
text/plain	13.744	2,03	48,57	140.182K	0,27

text/css	12.478	1,84	64,04	46.531.274	0,09
application/x-shockwave-flash	8.935	1,32	33,07	230.130K	0,44
image/png	7.417	1,09	79,63	18.992.956	0,04
text/xml	4.851	0,72	3,57	4.871.154	0,01
text/javascript	1.173	0,17	11,68	6.616.607	0,01
application/vnd,ms,wms-hdr,asfv1	533	0,08	0	1.332.773	0
application/x-mms-framed	505	0,07	0	329.339K	0,63
application/pdf	421	0,06	12,11	71.323.327	0,13
multipart/byteranges	354	0,05	0	158.997K	0,3
Vídeo/x-ms-asf	289	0,04	0,35	602.724	0
Outras: 81 *content-types*	2.827	0,42	29,82	839.583K	1,61
Total	**677.445**	**100**	**25,97**	**51.039M**	**100**

As informações apresentadas na tabela 11 são:

- ***content-type***: O *content-type* é um campo que faz parte do cabeçalho da mensagem de resposta HTTP. Este campo é gerado pelo servidor *web* e serve para informar aos clientes o tipo do objeto que está sendo retornado como resposta a uma mensagem de requisição. É possível obter a tabela de *content-types* a partir das especificações do protocolo HTTP.
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna "***content-type"***.
- **%R**: Este campo apresenta o percentual de requisições que resultaram na categoria que aparece na coluna "***content-type",*** em relação a quantidade total de requisições do lote.
- **HR *(Hit Rate)***: Este campo apresenta o percentual de requisições que

resultaram em acerto no *cache* (*cache hit*)**,** em relação a quantidade de requisições da categoria que aparece na coluna “***content-type”***.

- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna “***content-type”*.**
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos cuja requisição resultou na categoria que aparece na coluna “***content-type”***, em relação a quantidade total de *bytes* transferidos no lote.

Somando-se os percentuais correspondentes a requisições a objetos texto (*content-type* igual a “text/html”, “text/plain”, “text/xml”), tem-se um total de 20,42%. É importante observar que este percentual não reflete o número real de objetos texto retornados como resposta a requisições, pois não existe um grau aceitável de confiabilidade no valor do campo *content-type* retornado por servidores *web*.

Para que fosse possível obter um valor mais confiável para o percentual de objetos texto requisitados no lote P-04-A, foi preciso realizar uma análise considerando-se a extensão da URI (*Uniform Resource Identifier*) de requisição. A tabela 12 apresenta a quantidade de requisições classificadas pela extensão da URI.

TABELA 12 - REQUISIÇÕES POR EXTENSÃO DE URI NO LOTE P-04-A.

Extensões	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
gif	184.964	27,3	57,85	289.157K	0,55
<dinâmicas>	165.303	24,4	1,36	1.700.087K	3,25
<erros>	130.914	19,32	4,67	67.801.569	0,13
jpg	66.973	9,89	37,77	640.526K	1,23
<seguras> (HTTPS)	26.339	3,89	0	45.723M	89,59
<nenhuma>	21.851	3,23	4,35	938.768K	1,8

js	20.311	3	64,5	111.045K	0,21
css	15.304	2,26	53,58	36.875.587	0,07
png	7.517	1,11	81,22	18.083.041	0,03
swf	5.908	0,87	53,99	187.481K	0,36
htm	4.877	0,72	19,15	40.774.802	0,08
xml	4.547	0,67	3,45	4.094.320	0,01
html	3.800	0,56	29,92	29.591.044	0,06
txt	3.545	0,52	1,13	2.448.412	0
asp	3.021	0,45	0,07	32.440.069	0,06
php	2.763	0,41	0	28.499.080	0,05
JPG	1.938	0,29	12,38	38.481.020	0,07
jar	742	0,11	0	8.084.519	0,02
jsp	565	0,08	0	3.825.284	0,01
pdf	557	0,08	8,26	76.424.844	0,14
Outras: 177 extensões	5.706	0,84	20,75	1.197.744K	2,29
Total	**677.445**	**100**	**25,97**	**51.039M**	**100**

As informações apresentadas na tabela 12 são:

- **Extensões**: Este campo corresponde às extensões dos arquivos onde estão armazenados os objetos identificados através de uma URI (*Uniform Resource Identifier*), que são requisitados pelos clientes através de um servidor *proxy*.
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna "**Extensões"**.
- **%R**: Este campo apresenta o percentual de requisições que resultaram na categoria que aparece na coluna "**Extensões",** em relação a quantidade total de requisições do lote.

- **HR *(Hit Rate):*** Este campo apresenta o percentual de requisições que resultaram em acerto no *cache* (*cache hit*)**,** em relação a quantidade de requisições da categoria que aparece na coluna “**Extensões”**.
- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna “**Extensões”**.
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos cuja requisição resultou na categoria que aparece na coluna “**Extensões”**, em relação a quantidade total de *bytes* transferidos no lote.

A partir da análise da tabela 12 é possível observar que apenas 2,74% correspondem a objetos texto. Este percentual foi obtido através da soma das requisições referentes às extensões de URI: “jsp”, “php”, “asp”, “txt”, “html”, “htm” e “xml”.

É importante observar que os objetos que representam imagens correspondem na verdade a 37,19% das requisições (extensões “gif” somadas a extensões “jpg”).

Estas características exercem grande influência sobre as estratégias de substituição baseadas em semântica.

As análises realizadas no lote P-04-A contemplaram também a distribuição dos objetos requisitados por tamanho. É possível observar através da tabela 13 que 38,7% da taxa de acerto (*hit rate*) correspondem a objetos que possuem tamanhos variando entre 100 e 999 bytes. Estes objetos correspondem a 44% das requisições do lote P-04-A. Isto explica a relação entre *hit rate* (25%) e *byte hit rate* (0,97%) apresentada na tabela 10, e que também pode ser observada nas figuras 36 e 37.

TABELA 13 - DISTRIBUIÇÃO DE OBJETOS POR TAMANHO NO LOTE P-04-A.

Tamanho do Objeto (*bytes*)	**Requisições**	**HR**	**s/Req.**	***Bytes***	**BHR**	**KB/s**
0-0	99.225	1,66	48,28	0	0	0
1-9	62	0	0,37	62	0	0
10-99	725	0	1,66	28.479	0	0,02
100-999	297.062	38,7	0,94	131.253K	33,27	0,47
1.000-9.999	211.939	24,24	5,43	642.667K	23,08	0,56
10.000-99.999	63.097	12,22	5,32	1.713.900K	9,94	5,1
100.000-999.999	4.179	5,62	159,96	1.289.189K	4,3	1,93
1.000.000-9.999.999	756	2,65	966,78	2.138.604K	2,75	2,93
10.000.000-99.999.999	271	0,74	5.485,34	9.676.656K	0,29	6,51
100.000.000-999.999.999	129	0	17.487	35.812M	0	16,26
Total	**677.445**	**25,97**	**17,27**	**51.039M**	**0,97**	**4,47**

As informações apresentadas na tabela 13 são:

- **Tamanho do Objeto (*bytes*)**: Este campo corresponde ao intervalo em *bytes* referente ao tamanho dos objetos requisitados pelos clientes através de um servidor *proxy*.
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna "**Tamanho do Objeto (*bytes*)"**.
- ***HR (Hit Rate)***: Este campo apresenta o percentual de requisições que resultaram em acerto no *cache* (*cache hit*)**,** em relação a quantidade de requisições da categoria que aparece na coluna "**Tamanho do Objeto (*bytes*)"**.
- **s/Rec.**: Este campo apresenta o tempo médio para recuperação (latência) de um objeto cuja requisição resultou na categoria que

aparece na coluna “**Tamanho do Objeto (*bytes*)”.** Esta informação é apresentada na unidade de tempo “segundos”.

- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna “**Tamanho do Objeto (*bytes*)”**.
- **BHR *(Byte Hit Rate)***: Este campo apresenta o percentual de requisições que resultaram em acerto no *cache* (*cache hit*)**,** em relação a quantidade de *bytes* apresentada na coluna “***Bytes***”, para as requisições da categoria que aparece na coluna “**Tamanho do Objeto (*bytes*)”**.
- **KB/s**: Este campo apresenta a banda utilizada para transferência dos objetos cujas requisições resultaram na categoria que aparece na coluna “**Tamanho do Objeto (*bytes*)”**. Esta informação é apresentada na unidade de medida *Kbytes* por segundo.

A tabela 14 apresenta os resultados obtidos referentes a distribuição de requisições em relação a latência no lote P-04-A. A latência corresponde ao tempo de resposta de uma requisição realizada por um cliente. É possível observar que 76% das requisições foram atendidas em menos de um segundo.

TABELA 14 - REQUISIÇÕES POR LATÊNCIA NO LOTE P-04-A.

Latência (ms)	Requisições	%R	HR	s/Req.	*Bytes*	%B	BHR	KB/s
<= 0,1	776	0,11	34,02	0	691.831	0	0,89	8.706,39
<= 0,2	776	0,11	34,02	0	691.831	0	0,89	8.706,39
<= 0,5	776	0,11	34,02	0	691.831	0	0,89	8.706,39
<= 1	12.971	1,91	6,52	0	18.796.217	0,04	2,22	1.495,66
<= 2	19.721	2,91	11,89	0	27.878.838	0,05	5,44	1.056,37
<= 5	25.266	3,73	20,58	0	34.553.751	0,06	12,06	739,71

<= 10	27.508	4,06	22,86	0	38.516.478	0,07	16,59	595,97
<= 20	30.340	4,48	24,63	0	44.477.126	0,08	21,54	408,76
<= 50	71.660	10,58	57,48	0,02	98.883.532	0,18	50,99	58,33
<= 100	136.005	20,08	60,49	0,05	202.557K	0,39	55,83	31,05
<= 200	227.422	33,57	56,67	0,09	416.622K	0,8	47,36	21,02
<= 500	421.664	62,24	37,87	0,2	976.608K	1,87	30,14	11,4
<= 1.000	515.596	**76,11**	33,07	0,29	1.553.606K	2,97	22,51	10,36
<= 2.000	556.926	82,21	31,29	0,37	2.221.313K	4,25	17,16	10,79
<= 5.000	576.979	85,17	30,43	0,47	2.734.427K	5,23	16,13	10,17
<= 1.0000	583.811	86,18	30,12	0,54	3.052.416K	5,84	15,37	9,66
<= 20.000	586.955	86,64	29,97	0,61	3.274.677K	6,27	14,9	9,12
<= 50.000	589.941	87,08	29,83	0,78	3.652.851K	6,99	13,6	7,89
<= 100.000	670.858	99,03	26,23	7,9	3.906.332K	7,47	12,76	0,74
<= 200.000	674.211	99,52	26,1	8,61	4.206.676K	8,05	12	0,72
<= 500.000	675.787	99,76	26,04	9,32	4.964.495K	9,5	10,17	0,79
<= 1.000.000	676.704	99,89	26	10,3	5.841.502K	11,18	8,64	0,84
<= 1E+10	677.445	100	25,97	17,27	51.039M	100	0,97	4,47
Total	**677.445**	**100**	**25,97**	**17,27**	**51.039M**	**100**	**0,97**	**4,47**

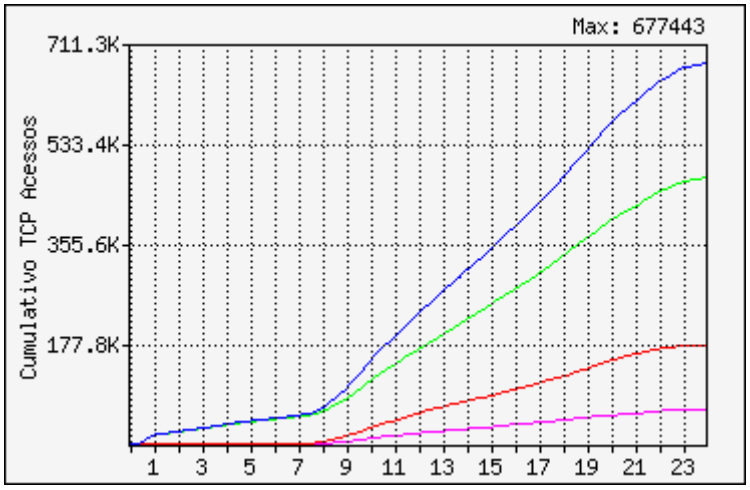
As informações apresentadas na tabela 14 são:

- **Latência(ms)**: Este campo corresponde ao intervalo de tempo entre a requisição realizada por um cliente e o retorno do objeto requisitado. Os valores apresentados neste campo estão expressos em milisegundos.
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna “**Latência”**.
- **%R**: Este campo apresenta o percentual de requisições que resultaram

na categoria que aparece na coluna “**Latência”,** em relação a quantidade total de requisições do lote.

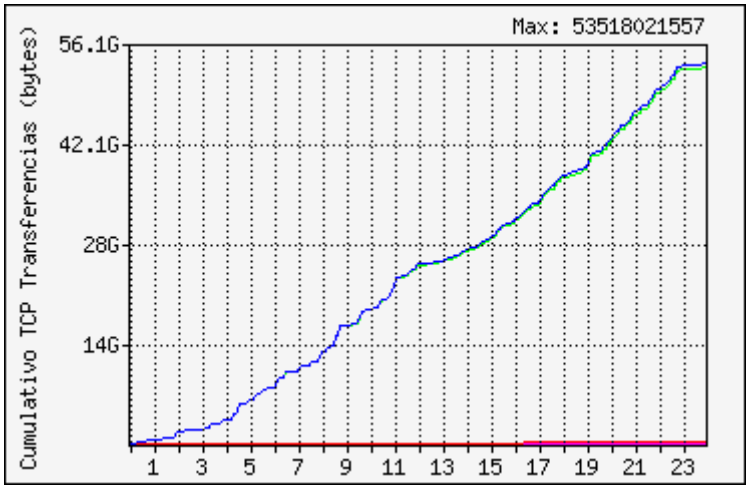
- **HR (*Hit Rate*)**: Este campo apresenta o percentual de requisições que resultaram em acerto no *cache**,*** em relação a quantidade de requisições da categoria que aparece na coluna “**Latência”**.
- **s/Req.**: Este campo apresenta o tempo médio para recuperação (latência) de um objeto cuja requisição resultou na categoria que aparece na coluna “**Latência”.** Esta informação é apresentada na unidade de tempo “segundos”.
- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna “**Latência”**.
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos cuja requisição resultou na categoria que aparece na coluna “**Latência”**, em relação a quantidade total de *bytes* transferidos no lote.
- **BHR (*Byte Hit Rate*)**: Este campo apresenta o percentual de requisições que resultaram em acerto no *cache**,*** em relação a quantidade de *bytes* apresentada na coluna “***Bytes***”, para as requisições da categoria que aparece na coluna “**Latência”**.
- **KB/s**: Este campo apresenta a banda utilizada para transferência dos objetos cujas requisições resultaram na categoria que aparece na coluna “**Latência”**. Esta informação é apresentada na unidade de medida *Kbytes* por segundo.

A figura 38 também apresenta um gráfico da duração média de transferência (latência) no lote P-04-A.



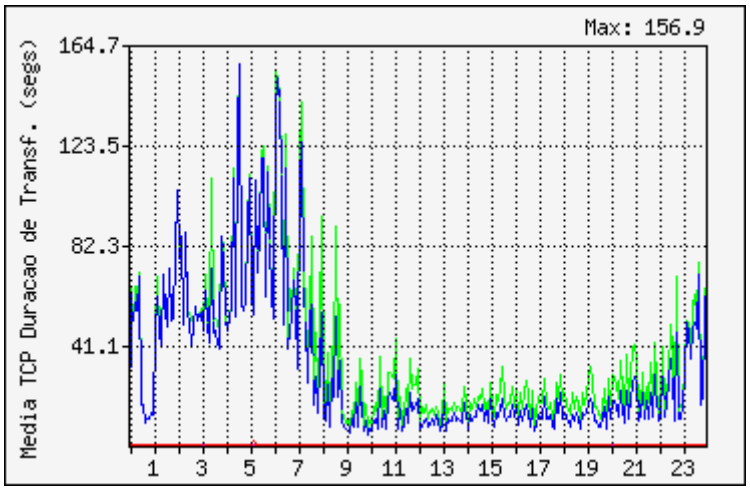
Total de Acessos: 677445
Média de Acessos: 28226,79 por hora
Total de *Cache Hits*: 175961
Média de *Cache Hits*: 7331,7 por hora
% de *Cache Hits*: 25,97 %
Total de *Cache IMS Hits*: 63230
Média de *Cache IMS Hits*: 2634,58 por hora
Total de *Cache Misses*: 474904
Média de *Cache Misses*: 19787,66 por hora
% de *Cache Misses*: 70,1 %

FIGURA 36 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE P-04-A.



Total Transferido: 51,0 GB
Média Transferida: 2,2 GB por hora
Total de *Cache Hits*: 493,1 MB
Média de *Cache Hits*: 21,5 MB por hora
% de *Cache Hits*: 0,97 %
Total de *Cache IMS Hits*: 18,1 MB
Média de *Cache IMS Hits*: 754,4 Kb por hora
Total de *Cache Misses*: 50,5 GB
Média de *Cache Misses*: 2,2 GB por hora
% de *Cache Misses*: 97,95 %

FIGURA 37 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE P-04-A.



Duração Média de Transferência: 31,91 segundos
Duração Média de *Cache Hit*: 0,19 segundos
Duração Média de *Cache Miss*: 37,65 segundos

FIGURA 38 - GRÁFICO DE DURAÇÃO MÉDIA DE TRANSFERÊNCIA (LATÊNCIA) NO LOTE P-04-A.

A tabela 15 apresenta a distribuição de requisições do lote P-04-A, classificadas pelo campo *status-code* do protocolo HTTP. O campo *status-code* é retornado como parte da mensagem de resposta HTTP, indicando o resultado da requisição. A partir da análise da tabela 15 é possível observar que 13% das requisições apresentam *status-code* igual a “503 (*Service Unavaiable*)”. Este tráfego foi analisado e verificou-se que se trata de clientes tentando utilizar ferramentas para *Messenger* ou ferramentas para compartilhamento ponto-a-ponto de arquivos em rede, como por exemplo, o *bit-torrent.*

TABELA 15 - REQUISIÇÕES POR *STATUS-CODE* NO LOTE P-04-A.

status-code	**Requisições**	**%R**	***Bytes***	**%B**
000 (Mais usado com tráfego UDP)	7.531	1,11	62	0
200 (OK)	408.023	60,23	50.006M	97,98
201 (*Created*)	2	0	1230	0
204 (*No Content*)	818	0,12	263.100	0
206 (*Partial Content*)	746	0,11	273.938K	0,52
207 (*Multi Status*)	5	0	13.633	0
301 (*Moved Permanently*)	1.516	0,22	956.941	0
302 (*Moved Temporarily*)	21.879	3,23	1,9E+07	0,04
303 (*See Other*)	2	0	1.909	0
304 (*Not Modified*)	105.919	15,64	2,7E+07	0,05
305 (*Use Proxy*)	16	0	582.543	0
307 (*Temporary Redirect*)	7	0	6.267	0
400 (*Bad Request*)	436	0,06	621.467	0
401 (*Unauthorized*)	86	0,01	9.7530	0
403 (*Forbidden*)	26.752	3,95	4E+07	0,08
404 (*Not Found*)	9.805	1,45	1,6E+07	0,03
410 (*Gone*)	21	0	26.747	0

411 (*Length Required*)	196	0,03	641.296	0
500 (*Internal Server Error*)	814	0,12	4.333.700	0,01
502 (*Bad Gateway*)	284	0,04	590.914	0
503 (*Service Unavailable*)	91.919	13,57	4.282.089	0,01
504 (*Gateway Timeout*)	601	0,09	1.358.440	0
600 (*Squid header parsing error*)	66	0,01	670.826K	1,28
999 (desconhecido)	1	0	2.155	0
Total	**677.445**	**100**	**51.039M**	**100**

As informações apresentadas na tabela 15 são:

- ***status-code***: Campo da mensagem de resposta do protocolo HTTP que define o resultado da requisição. Os detalhes do funcionamento do protocolo HTTP podem ser obtidos no capítulo 2.
- **Requisições**: Este campo apresenta o número de requisições que resultaram na categoria que aparece na coluna "***status-code*"**.
- **%R**: Este campo apresenta o percentual de requisições que resultaram na categoria que aparece na coluna "***status-code*",** em relação a quantidade total de requisições do lote.
- ***Bytes*:** Este campo apresenta a quantidade de *bytes* envolvidos na transferência dos objetos cuja requisição resultou na categoria que aparece na coluna "***status-code*"**.
- **%B**: Este campo apresenta o percentual de *bytes* envolvidos na transmissão dos objetos cuja requisição resultou na categoria que aparece na coluna "***status-code*"**, em relação a quantidade total de *bytes* transferidos no lote.

7.1.2 Análise do Lote P-05-A

Este lote corresponde às requisições de objetos realizadas pela comunidade de usuários da PUCPR no período entre 00:00 hora do dia 05 de julho de 2005 e 00:00 hora do dia 06 de julho de 2005. O objetivo da análise deste lote é apresentar as características de requisições da comunidade de usuários da PUCPR.

Ao todo foram computadas 595.656 requisições de objetos neste lote. Estas requisições foram originadas a partir de 1.573 endereços IP distintos, indicando que o mesmo número de usuários esteve envolvido nas requisições. Da mesma forma que no lote anteriormente analisado, a maioria das requisições foi realizada através do método de requisição “GET” do protocolo HTTP. A partir da análise da tabela 16 é possível observar que aproximadamente 76,1% (somatória de requisições realizadas através dos métodos “GET” e “HEAD”) das requisições são passíveis de *cache*.

TABELA 16 - REQUISIÇÕES POR *REQUEST-METHO*D NO LOTE P-05-A.

request-method	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
GET	448.079	75,22	3,47	4.912.029K	8,68	3,16
CONNECT	122.272	20,53	94,07	50.342M	91,06	4,48
POST	24.296	4,08	3,82	146.055K	0,26	1,57
HEAD	739	0,12	0,88	263.151	0	0,4
OPTIONS	179	0,03	3,32	237.313	0	0,39
PROPFIND	84	0,01	1,06	93.691	0	1,03
LOCK	3	0	0,16	860	0	1,72
PROPPATCH	3	0	3,77	2498	0	0,22
MOVE	1	0	4,18	615	0	0,14
Total	**595.656**	**100**	**22,08**	**55.283M**	**100**	**4,31**

Os campos da tabela 16 correspondem aos campos da tabela 9, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

Apesar da maioria das requisições terem sido realizadas através do método “GET”, apenas 25,19% resultaram em *cache hit*, conforme pode ser observado na tabela 17, que apresenta as requisições classificadas por *result-code*, e na figura 39 que apresenta um gráfico de requisições no lote P-05-A.

TABELA 17 - REQUISIÇÕES POR *RESULT-CODE* NO LOTE P-05-A.

Status	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
HIT	150.046	25,19	0,69	433.821K	0,77	4,2
MISS	416.618	69,94	31	54.778M	99,09	4,34
ERROS	28.992	4,87	4,55	84.499.192	0,15	0,63
Total	**595.656**	**100**	**22,08**	**55.283M**	**100**	**4,31**

Os campos da tabela 17 correspondem aos campos da tabela 10, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

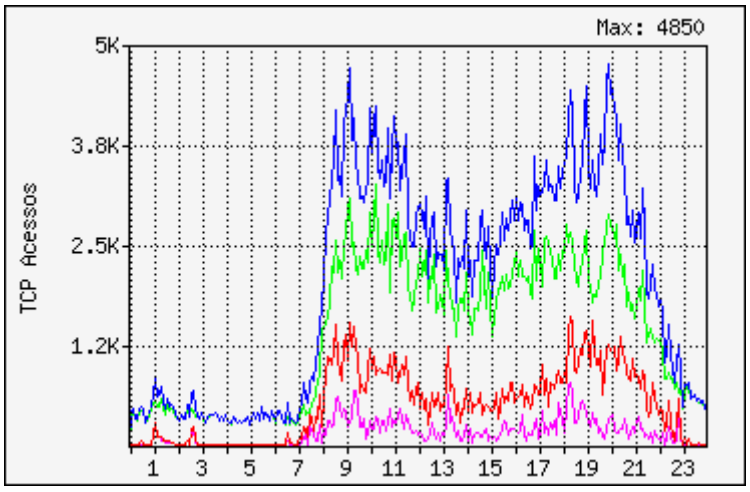
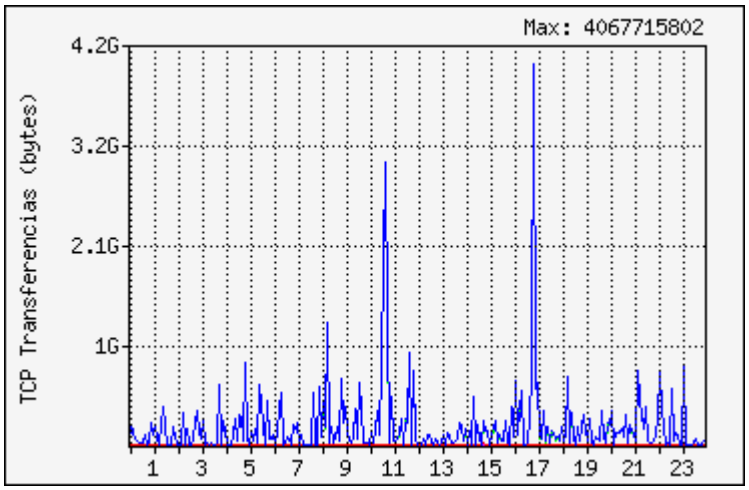


FIGURA 39 - GRÁFICO DE REQUISIÇÕES NO LOTE P-05-A.

O tamanho total dos objetos requisitados no lote P-05-A corresponde a 55.283 MB, sendo que o total de banda salva pelas requisições que resultaram em *cache hit* corresponde a 424 MB. Isto significa que a taxa de acerto no *cache*, em relação ao tamanho dos objetos (*byte hit rate*), corresponde a apenas 0,77% no lote P-05-A.



Total Transferido: 55,2 GB
Média Transferida: 2,4 GB por hora
Total de *Cache Hits*: 424,2 MB
Média de *Cache Hits*: 18,5 MB por hora
% de *Cache Hits*: 0,77 %
Total de *Cache IMS Hits*: 13,4 MB
Média de *Cache IMS Hits*: 559 Kb por hora
Total de *Cache Misses*: 54,8 GB
Média de *Cache Misses*: 2,3 GB por hora
% de *Cache Misses*: 99,13 %

FIGURA 40 - GRÁFICO DE *BYTES* TRANSFERIDOS NO LOTE P-05-A.

Uma análise mais detalhada das requisições classificadas por *content-type* pode ser observada na tabela 18. A linha que apresenta o valor “<erros>” como *content-type* corresponde a requisições do lote P-05-A que foram realizadas aos serviços de *Messenger*, as requisições de aplicativos para compartilhamento de arquivos ponto-a-ponto como, por exemplo, o *bit-torrent*, ou finalmente a erros de acessos a endereços não autorizados.

Ainda a partir da análise da tabela 18 é possível observar que grande parte das requisições refere-se a objetos binários, mais especificamente a imagens. É possível observar que 24,37% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “gif”. Da mesma forma, é possível observar que 9,84% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “jpeg”.

TABELA 18 - REQUISIÇÕES POR *CONTENT-TYPE* NO LOTE P-05-A.

content-type	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
image/gif	145.143	24,37	61,87	251.070K	0,44
<erros>	124.446	20,89	3	54.077.306	0,09
text/html	100.129	16,81	3,94	1.052.904K	1,86
image/jpeg	58.636	9,84	33,46	591.449K	1,04
<desconhecidas>	49.728	8,35	10,72	737.155K	1,3
<seguras> (HTTPS)	29.889	5,02	0	50.341M	91,06
application/x-javascript	23.614	3,96	32,01	130.895K	0,23
aim/http	17.336	2,91	0	6.044.094	0,01
text/plain	11.750	1,97	53,2	338.862K	0,6
text/css	10.629	1,78	60,14	43.748.590	0,08
application/x-shockwave-flash	7.021	1,18	34,95	173.689K	0,31
image/png	4.969	0,83	78,71	11.523.016	0,02
text/xml	4.079	0,68	2,99	5.971.302	0,01
application/octet-stream	3.385	0,57	6	479.005K	0,85
application/vnd,ms,wms-hdr,asfv1	749	0,13	0	1.998.067	0
application/x-mms-framed	642	0,11	0	257.212K	0,45
text/javascript	400	0,07	16,75	2.420.788	0
video/x-ms-asf	305	0,05	3,61	7.894.662	0,01
application/pdf	248	0,04	4,84	118.428K	0,21
audio/wav	177	0,03	98,87	1.309.341	0
Outras: 86 *content-types*	2381	0,4	19,36	798.028K	1,41
Total	**595.656**	**100**	**25,19**	**55.283M**	**100**

Os campos da tabela 18 correspondem aos campos da tabela 11, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

Somando-se os percentuais correspondentes as requisições a objetos texto (*content-type* igual a “text/html”, “text/plain”, “text/xml”), tem-se um total de 19,46%. É importante observar que este percentual não reflete o número real de objetos texto retornados como resposta a requisições, pois não existe um grau aceitável de confiabilidade no valor do campo *content-type* retornado por servidores *web*.

Para que fosse possível obter um valor mais confiável para o percentual de objetos texto requisitados no lote P-05-A, foi preciso realizar uma análise considerando-se a extensão da URI (*Uniform Resource Identifier*) das requisições. A tabela 19 apresenta a quantidade de requisições classificadas pela extensão da URI.

TABELA 19 - REQUISIÇÕES POR EXTENSÃO DE URI NO LOTE P-05-A.

Extensões	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
gif	171.179	28,74	57,57	247.358K	0,44
<erros>	124.449	20,89	3	54.078.788	0,09
<dinâmicas>	124.130	20,84	1,12	1.447.397K	2,56
jpg	55.590	9,33	35,57	463.116K	0,82
<seguras> (HTTPS)	29.889	5,02	0	50.341M	91,06
<nenhuma>	20.822	3,5	4,15	901.455K	1,59
js	16.643	2,79	56,23	99.461K	0,18
css	13.800	2,32	48,86	34.991.583	0,06
png	4.906	0,82	80,53	9.658.118	0,02
txt	4.737	0,8	0,19	3.280.219	0,01
swf	4.393	0,74	60,16	142.876K	0,25
xml	4.063	0,68	4,45	3.532.092	0,01
htm	4.046	0,68	18,09	35.460.297	0,06
html	3.479	0,58	26,88	30.738.943	0,05

asp	2.702	0,45	0	32.870.927	0,06
php	2.532	0,43	0	27.640.063	0,05
JPG	1.187	0,2	9,18	39.597.972	0,07
jsp	513	0,09	0	2.931.783	0,01
jar	505	0,08	0,2	1.395.688	0
GIF	456	0,08	22,59	1.877.278	0
Outras: 188 extensões	5.635	0,95	17,3	1.487.323K	2,63
Total	**595.656**	**100**	**25,19**	**55.283M**	**100**

Os campos da tabela 19 correspondem aos campos da tabela 12, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

A partir da análise da tabela 19 é possível observar que apenas 3,71% correspondem a objetos texto. Este percentual foi obtido através da soma das requisições referentes às extensões de URI: “jsp”, “php”, “asp”, “txt”, “html”, “htm” e “xml”.

É importante observar que os objetos que representam imagens correspondem na verdade a 39,17 % das requisições (somatório das extensões “gif”, “jpg”, “png”, “GIF” e “JPG”).

As análises realizadas no lote P-05-A contemplaram também a distribuição dos objetos requisitados por tamanho. É possível observar através da tabela 20 que 38,6% da taxa de acerto (*hit rate*) correspondem a objetos que possuem tamanhos variando entre 100 e 999 bytes. Estes objetos correspondem a 42,34% das requisições do lote P-05-A. Isto explica a relação entre *hit rate* (25,19%) e *byte hit rate* (0,77%) apresentada na tabela 17, e que também pode ser observada nas figuras 41 e 42.

TABELA 20 - DISTRIBUIÇÃO DE OBJETOS POR TAMANHO NO LOTE P-05-A.

Tamanho do Objeto (*bytes*)	**Requisições**	**HR**	**s/Req.**	***Bytes***	**BHR**	**KB/s**
0-0	105.786	3,17	44,7	0	0	0
1-9	52	0	0,83	52	0	0
10-99	826	0	11,81	32979	0	0
100-999	252.259	38,6	2,37	107.422K	35,83	0,18
1.000-9.999	179.820	23,66	8,86	551.704K	22,97	0,35
10.000-99.999	52.293	12,42	8,98	1.461.011K	9,97	3,11
100.000-999.999	3.487	7,54	184,36	1.080.292K	6,15	1,68
1.000.000-9.999.999	701	3	994,52	1.809.812K	1,81	2,6
10.000.000-99.999.999	268	0,37	6.309,46	9.719M	0,24	5,89
100.000.000-999.999.999	164	0	16579	40.671M	0	15,32
Total	**595.656**	**25,19**	**22,08**	**55.283M**	**0,77**	**4,31**

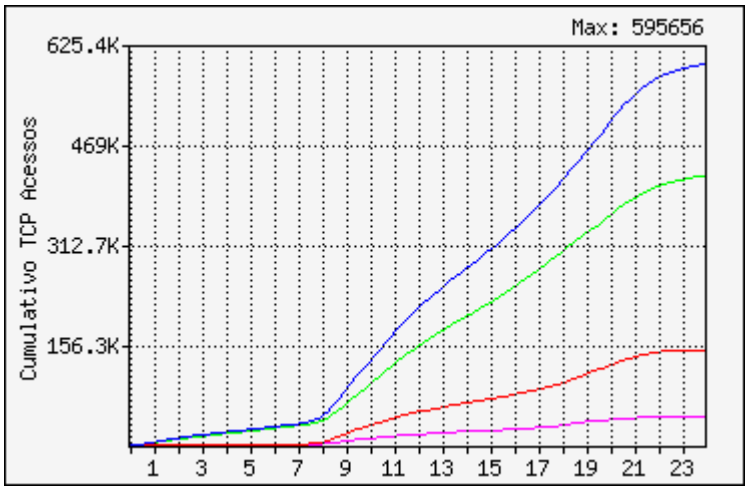
Os campos da tabela 20 correspondem aos campos da tabela 13, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

A tabela 21 apresenta os resultados obtidos referentes a distribuição de requisições em relação a latência no lote P-05-A. A coluna “Latência” apresenta o tempo de resposta a uma requisição em milisegundos. É possível observar que 53,31% das requisições foram atendidas em menos de um segundo. A figura 43 também apresenta um gráfico da duração média de transferência (latência) no lote P-05-A.

Os campos da tabela 21 correspondem aos campos da tabela 14, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

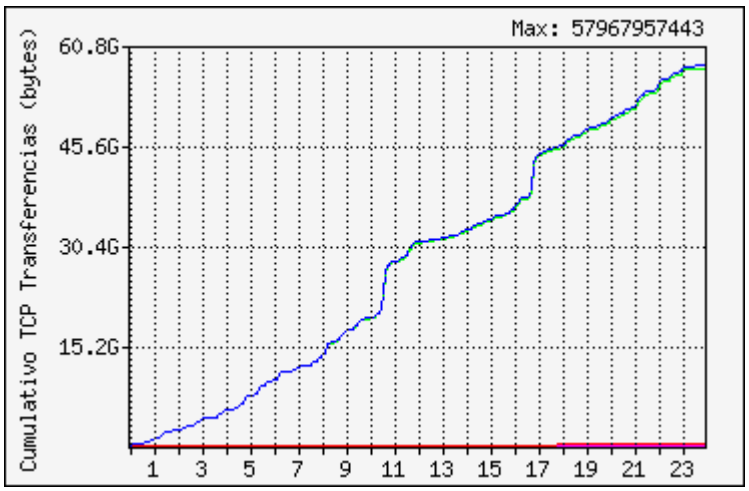
TABELA 21 - REQUISIÇÕES POR LATÊNCIA NO LOTE P-05-A.

Latência (ms)	Requisições	%R	HR	s/Req.	*Bytes*	%B	BHR	KB/s
<= 0,1	770	0,13	57,53	0	278.684	0	2,04	3.534,45
<= 0,2	770	0,13	57,53	0	278.684	0	2,04	3.534,45
<= 0,5	770	0,13	57,53	0	278.684	0	2,04	3.534,45
<= 1	3.087	0,52	18,3	0	3.728.732	0,01	2,77	1.521,03
<= 2	6.997	1,17	15,22	0	9.336.477	0,02	5,27	892,66
<= 5	11.882	1,99	19,61	0	16.135.917	0,03	10,93	562,25
<= 10	14.316	2,4	22,08	0	19.337.516	0,03	16,03	400,96
<= 20	17.199	2,89	23,05	0,01	23.226.760	0,04	20,77	255,9
<= 50	28.138	4,72	40,92	0,02	38.522.821	0,07	36,6	71,61
<= 100	57.556	9,66	54,49	0,05	78.343.680	0,14	49,02	27,97
<= 200	108.733	18,25	57,52	0,1	165.244K	0,29	49,27	15,95
<= 500	222.720	37,39	47,05	0,22	425.909K	0,75	37,91	8,68
<= 1.000	317.551	**53,31**	40,18	0,37	765.757K	1,35	29,86	6,54
<= 2.000	400.362	67,21	35,24	0,59	1.243.271K	2,2	22,73	5,28
<= 5.000	474.032	79,58	31,33	0,97	1.828.446K	3,23	18,21	3,98
<= 10.000	495.472	83,18	30,22	1,22	2.234.521K	3,95	16,16	3,69
<= 20.000	503.147	84,47	29,8	1,41	2.532.812K	4,47	15,11	3,57
<= 50.000	506.998	85,12	29,59	1,64	2.857.923K	5,05	14,1	3,43
<= 100.000	586.785	98,51	25,57	9,59	3.191.533K	5,64	13,56	0,57
<= 200.000	592.056	99,4	25,34	10,83	3.490.081K	6,17	12,43	0,54
<= 500.000	594.099	99,74	25,26	11,84	4.332.056K	7,65	10,01	0,62
<= 1.000.000	594.794	99,86	25,23	12,66	5.061.639K	8,94	8,57	0,67
<= 1E+10	595.656	100	25,19	22,08	55.283M	100	0,77	4,31
Total	**595.656**	**100**	**25,19**	**22,08**	**55.283M**	**100**	**0,77**	**4,31**



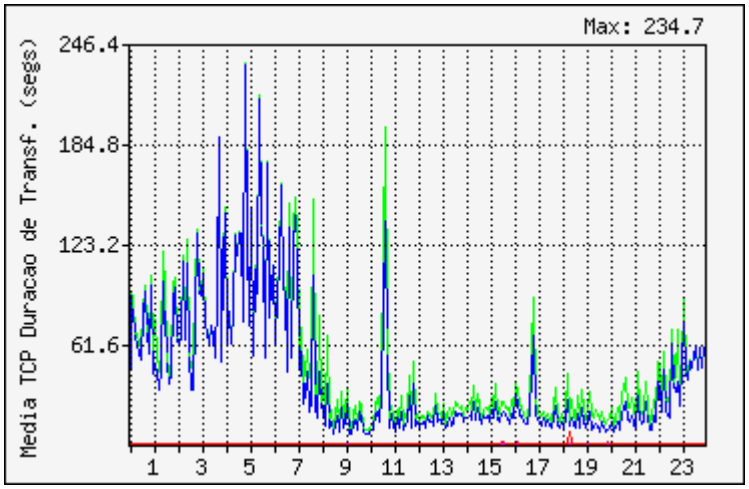
Total de Acessos: 595656
Média de Acessos: 24819 por hora
Total de *Cache Hits*: 150046
Média de *Cache Hits*: 6251,91 por hora
% de *Cache Hits*: 25,19 %
Total de *Cache IMS Hits*: 46705
Média de *Cache IMS Hits*: 1946,04 por hora
Total de *Cache Misses*: 422299
Média de *Cache Misses*: 17595,79 por hora
% de *Cache Misses*: 70,89 %

FIGURA 41 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE P-05-A.



Total Transferido: 55,2 GB
Média Transferida: 2,4 GB por hora
Total de *Cache Hits*: 424,2 MB
Média de *Cache Hits*: 18,5 MB por hora
% de *Cache Hits*: 0,77 %
Total de *Cache IMS Hits*: 13,4 MB
Média de *Cache IMS Hits*: 559 Kb por hora
Total de *Cache Misses*: 54,8 GB
Média de *Cache Misses*: 2,3 GB por hora
% de *Cache Misses*: 99,13 %

FIGURA 42 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE P-05-A.



Duração Média de Transferência: 41,48 segundos
Duração Média de *Cache Hit*: 0,43 segundos
Duração Média de *Cache Miss*: 48,27 segundos

FIGURA 43 - GRÁFICO DE DURAÇÃO MÉDIA DE TRANSFERÊNCIA (LATÊNCIA) NO LOTE P-05-A.

A tabela 22 apresenta a distribuição de requisições do lote P-05-A, classificadas pelo campo *status-code* do protocolo HTTP. O campo *status-code* é retornado como parte da mensagem de resposta HTTP, indicando o resultado da requisição. A partir da análise da tabela 22 é possível observar que 15,34% das requisições apresentam *status-code* igual a “503 (*Service Unavaiable*)”. A partir da análise deste tráfego verificou-se que se trata de clientes tentando utilizar ferramentas para *Messenger* ou ferramentas para compartilhamento ponto-a-ponto de arquivos em rede, como por exemplo o *bit-torrent*.

Os campos da tabela 22 correspondem aos campos da tabela 15, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

TABELA 22 - REQUISIÇÕES POR *STATUS-CODE* NO LOTE P-05-A.

status-code	**Requisições**	**%R**	***Bytes***	**%B**
000 (Mais usado com tráfego UDP)	10.587	1,78	4.184	0
200 (OK)	349.323	58,65	54.387M	98,38
201 (*Created*)	2	0	1.211	0
204 (*No Content*)	693	0,12	238.781	0
206 (*Partial Content*)	592	0,1	100.528K	0,18
207 (*Multi Status*)	42	0,01	76.221	0
301 (*Moved Permanently*)	1.493	0,25	768.985	0
302 (*Moved Temporarily*)	17.607	2,96	14.949.375	0,03
303 (*See Other*)	3	0	2.144	0
304 (*Not Modified*)	90.828	15,25	22.878.732	0,04
307 (*Temporary Redirect*)	6	0	5.046	0
400 (*Bad Request*)	397	0,07	589.429	0
401 (*Unauthorized*)	113	0,02	143.897	0
403 (*Forbidden*)	23.372	3,92	35.011.980	0,06

404 (*Not Found*)	7.686	1,29	11.552.211	0,02
405 (*Method Not Allowed*)	5	0	2.875	0
410 (*Gone*)	8	0	4.380	0
411 (*Length Required*)	285	0,05	931.347	0
500 (*Internal Server Error*)	453	0,08	1.131.964	0
502 (*Bad Gateway*)	306	0,05	605.089	0
503 (*Service Unavailable*)	91.360	15,34	3.116.852	0,01
504 (*Gateway Timeout*)	461	0,08	987.282	0
600 (*Squid header parsing error*)	34	0,01	725.897K	1,28
Total	**595.656**	**100**	**55.283M**	**100**

7.1.3 Análise do Lote S-04-A

As Faculdades SPEI possuem dois servidores *proxy Squid* que atendem a sub-redes distintas. A primeira sub-rede corresponde aos usuários administrativos da SPEI e a segunda sub-rede contempla as estações de trabalho que compõem os laboratórios utilizados por alunos e professores. O lote S-04-A corresponde a requisições realizadas ao servidor *proxy Squid* que atende aos laboratórios da SPEI. Apesar de terem sido realizadas análises tantos nos *logs* da rede administrativa quanto nos *logs* da rede de laboratórios, somente os *logs* da rede de laboratórios foram utilizados para a demonstração de resultados, por apresentarem maior diversidade em relação aos assuntos de interesse da comunidade de usuários. Este aspecto é importante principalmente para a avaliação do desempenho da estratégia de substituição LSR-VM.

Este lote corresponde às requisições de objetos realizadas pela comunidade de usuários da SPEI no período entre 00:00 hora do dia 04 de julho de 2005 e 00:00 hora do dia 05 de julho de 2005. O objetivo da análise deste lote é apresentar as características de requisições da comunidade de usuários da SPEI.

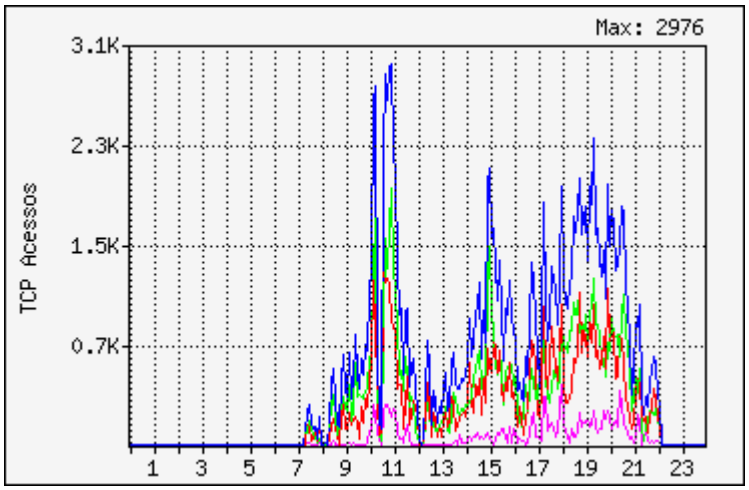
Ao todo foram computadas 169.942 requisições de objetos neste lote. Estas requisições foram originadas a partir de 70 endereços IP distintos. Atualmente a rede de laboratórios da SPEI conta com cinco laboratórios totalizando 100 estações de trabalho. A maioria das requisições foi realizada através do método de requisição "GET" do protocolo HTTP. Conforme foi explicado no capítulo 2, apenas os objetos requisitados através dos métodos "GET" e "HEAD" são considerados por mecanismos de *cache*. O fato de um objeto ter sido requisitado através do método "GET" não significa necessariamente que será armazenado no *cache*, pois outros fatores deverão ser levados em conta. A partir da análise da tabela 23 é possível observar que aproximadamente 95,47% (somatória de requisições realizadas através dos métodos "GET" e "HEAD") das requisições são passíveis de *cache*.

TABELA 23 - REQUISIÇÕES POR *REQUEST-METHO*D NO LOTE S-04-A.

request-method	**Requisições**	**%R**	**s/Rec.**	***Bytes***	**%B**	**KB/s**
GET	162.245	95,47	1,12	924.732K	94,09	5,1
POST	3.526	2,07	3,73	3.535.1917	3,51	2,63
CONNECT	3.125	1,84	9,77	23.633.100	2,35	0,76
PROPFIND	1.036	0,61	2,53	520.209	0,05	0,19
OPTIONS	8	0	0,08	3.692	0	5,62
HEAD	2	0	0,83	670	0	0,4
Total	**169.942**	**100**	**1,34**	**982.847K**	**100**	**4,32**

Os campos da tabela 23 correspondem aos campos da tabela 9, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

Este lote apresentou um taxa de acerto no *cache* (*Hit Rate*) bem melhor do que os lotes obtidos na PUCPR. O lote S-04-A apresentou 45,2% de *cache hit*, conforme pode ser observado na tabela 24, que apresenta as requisições classificadas por *result-code*, e na figura 44 que apresenta um gráfico de requisições no lote S-04-A.



Total de Acessos: 169818
Média de Acessos: 7075,75 por hora
Total de *Cache Hits*: 76806
Média de *Cache Hits*: 3200,25 por hora
% de *Cache Hits*: 45,22 %
Total de *Cache IMS Hits*: 15693
Média de *Cache IMS Hits*: 653,87 por hora
Total de *Cache Misses*: 93009
Média de *Cache Misses*: 3875,37 por hora
% de *Cache Misses*: 54,76 %

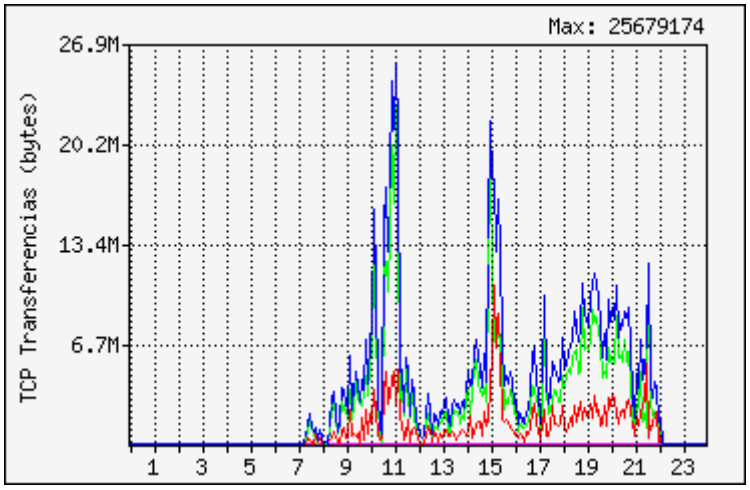
FIGURA 44 - GRÁFICO DE REQUISIÇÕES NO LOTE S-04-A.

TABELA 24 - REQUISIÇÕES POR *RESULT-CODE* NO LOTE S-04-A.

Status	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
HIT	76.806	45,2	0,26	273.877K	27,87	13,67
MISS	92.105	54,2	2,14	708.685K	72,11	3,6
ERROS	1.031	0,61	10,21	291.350	0,03	0,03
Total	**169.942**	**100**	**1,34**	**982.847K**	**100**	**4,32**

Os campos da tabela 24 correspondem aos campos da tabela 10, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

O tamanho total dos objetos requisitados no lote S-04-A corresponde a 982.847 KB, sendo que o total de banda salva pelas requisições que resultaram em *cache hit* corresponde a 267 MB. Isto significa que a taxa de acerto no *cache*, em relação ao tamanho dos objetos (*byte hit rate*) corresponde a 27,87% no lote S-04-A. Este desempenho foi superior aos lotes obtidos na PUCPR. É possível observar na figura 44 que existem picos de requisições, bem característicos, por volta das 10:00, 15:00 e 19:00 horas. Da mesma forma, é possível observar através da figura 45 que existem picos de transferências de *bytes* por volta das 10:00 e 15:00 horas no lote .



Total Transferido: 982,847 MB
Média Transferida: 41,9 MB por hora
Total de *Cache Hits*: 280,4 MB
Média de *Cache Hits*: 11,6 MB por hora
% de *Cache Hits*: 27,87 %
Total de *Cache IMS Hits*: 3,8 MB
Média de *Cache IMS Hits*: 160,5 Kb por hora
Total de *Cache Misses*: 725,8 MB
Média de *Cache Misses*: 30,2 MB por hora
% de *Cache Misses*: 72,12 %

FIGURA 45 - GRÁFICO DE *BYTES* TRANSFERIDOS NO LOTE S-04-A.

Uma análise mais detalhada das requisições classificadas por *content-type* pode ser observada na tabela 25. O *content-type* é um campo que faz parte do cabeçalho da mensagem de resposta HTTP.

É possível observar na tabela 25 que apenas 1,83% das requisições foram consideradas como erro por se tratar de tráfego não autorizado. Porém, em 4,62% das requisições o servidor *web* não informou o conteúdo do campo *content-type*. Neste caso estas requisições foram identificadas como *content-type* desconhecido.

Da mesma forma que nos lotes da PUCPR, grande parte das requisições do lote S-04-A refere-se a objetos binários, mais especificamente a imagens. É possível observar que 31,01% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “gif”. Da mesma forma, é possível observar que 21% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “jpeg”. Outras requisições com *content-type* igual a “image/png” e “image/bmp” também se referem a objetos que representam imagens. Ao todo, 54,23% das requisições correspondem a imagens no lote S-04-A.

TABELA 25 - REQUISIÇÕES POR *CONTENT-TYPE* NO LOTE S-04-A.

content-type	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
image/gif	52.700	31,01	76,98	84.209.349	8,37
text/html	47.163	27,75	13	352.714K	35,89
image/jpeg	35.683	21	47,92	194.596K	19,8
<desconhecidas>	7.859	4,62	6,58	1.414.359	0,14
application/x-javascript	7.515	4,42	36,02	60.092.132	5,97
text/css	3.727	2,19	62,97	18.250.452	1,81
image/png	3.203	1,88	92,79	9.134.851	0,91
<seguras> (HTTPS)	3.122	1,84	0	23.633.100	2,35
<erros>	3.102	1,83	26,08	4.990.219	0,5
text/plain	2.324	1,37	77,19	38.583.771	3,83
application/x-shockwave-flash	2.011	1,18	59,42	525.650.90	5,22
image/bmp	575	0,34	84,87	16.332.641	1,62
text/javascript	264	0,16	4,17	1.746.940	0,17
video/mpeg	136	0,08	35,29	80.164.774	7,97
application/octet-stream	136	0,08	30,15	24.338.424	2,42
text/xml	60	0,04	20	109.820	0,01
application/xml	52	0,03	9,62	258.239	0,03
video/x-ms-wmv	50	0,03	12	9.346.325	0,93
httpd/yahoo-send-as-is	44	0,03	0	23.517	0
application/zip	31	0,02	3,23	8.631.669	0,86
Outras: 28 content-types	185	0,11	28,65	12.164.149	1,21
Total	**169.942**	**100**	**45,2**	**982.847K**	**100**

Os campos da tabela 25 correspondem aos campos da tabela 11, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

Somando-se os percentuais correspondentes as requisições a objetos texto (*content-type* igual a “text/html”, “text/plain”, “text/xml”), tem-se um total de 29,16%. É importante observar que este percentual não reflete o número real de objetos texto retornados como resposta a requisições, pois não existe um grau aceitável de confiabilidade no valor do campo *content-type* retornado por servidores *web*.

Para que fosse possível obter um valor mais confiável para o percentual de objetos texto requisitados no lote S-04-A, foi preciso realizar uma análise considerando-se a extensão da URI (*Uniform Resource Identifier*) das requisições. A tabela 26 apresenta a quantidade de requisições classificadas pela extensão da URI.

TABELA 26 - REQUISIÇÕES POR EXTENSÃO DE ARQUIVO NO LOTE S-04-A.

Extensões	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
gif	55.525	32,67	75,04	82.697.961	8,22
<dinâmicas>	41.528	24,44	3,43	359.447K	36,57
jpg	37.383	22	57,47	163.137K	16,6
<nenhuma>	7.449	4,38	7,29	49.869.009	4,96
js	5.863	3,45	56,81	67.543.166	6,71
css	4.410	2,6	49,64	17.168.071	1,71
png	3.219	1,89	92,45	8.766.786	0,87
<seguras> (HTTPS)	3.122	1,84	0	23.633.100	2,35
<erros>	3.102	1,83	26,08	4.990.219	0,5
asp	2.247	1,32	0,22	22.459.190	2,23
swf	1.081	0,64	69,94	42.363.191	4,21
htm	822	0,48	41,12	6.168.498	0,61
html	617	0,36	27,39	4.089.382	0,41
php	591	0,35	0	5.371.516	0,53

bmp	553	0,33	87,52	863.127	0,09
JPG	419	0,25	14,08	2.716.450	0,27
jsp	327	0,19	0,31	647.492	0,06
GIF	324	0,19	96,91	601.729	0,06
aspx	215	0,13	0	4.374.764	0,43
tpl	133	0,08	0	473.219	0,05
Outras: 75 extensões	1.012	0,6	25,2	123.547K	12,57
Total	**169.942**	**100**	**45,2**	**98.2847K**	**100**

Os campos da tabela 26 correspondem aos campos da tabela 12, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

A partir da análise da tabela 26 é possível observar que apenas 2,83% correspondem a objetos texto. Este percentual foi obtido através da soma das requisições referentes às extensões de URI: “jsp”, “php”, “asp”, “aspx”, “html”, “htm” e “xml”.

É importante observar que os objetos que representam imagens correspondem na verdade a 57,33 % das requisições (somatório das extensões “gif”, “jpg”, “png”, “bmp”, “JPG” e “GIF”).

Estas características exercem grande influência sobre as estratégias de substituição baseadas em semântica.

As análises realizadas no lote S-04-A contemplam também a distribuição dos objetos requisitados por tamanho. É possível observar através da tabela 27 que 53,96% da taxa de acerto (*hit rate*) correspondem a objetos que possuem tamanhos variando entre 100 e 999 bytes. Estes objetos correspondem a 46,61% das requisições do lote S-04-A.

Os campos da tabela 27 correspondem aos campos da tabela 13, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

TABELA 27 - DISTRIBUIÇÃO DE OBJETOS POR TAMANHO NO LOTE S-04-A.

Tamanho do Objeto (*bytes*)	**Requisições**	**HR**	**s/Req.**	***Bytes***	**BHR**	**KB/s**
0-0	1.762	1,48	6,89	0	0	0
10-99	102	0	85,3	4.192	0	0
100-999	79.221	53,96	0,56	37.210.573	49,63	0,83
1.000-9.999	68.540	42,59	1,3	209.925K	39,15	2,36
10.000-99.999	19.809	23,71	3,05	507.115K	20,96	8,38
100.000-999.999	443	27,77	20,17	118.990K	34,07	13,32
1.000.000-9.999.999	65	29,23	67,05	110.475K	24,3	25,35
Total	**169.942**	**45,2**	**1,34**	**98.2847K**	**27,87**	**4,32**

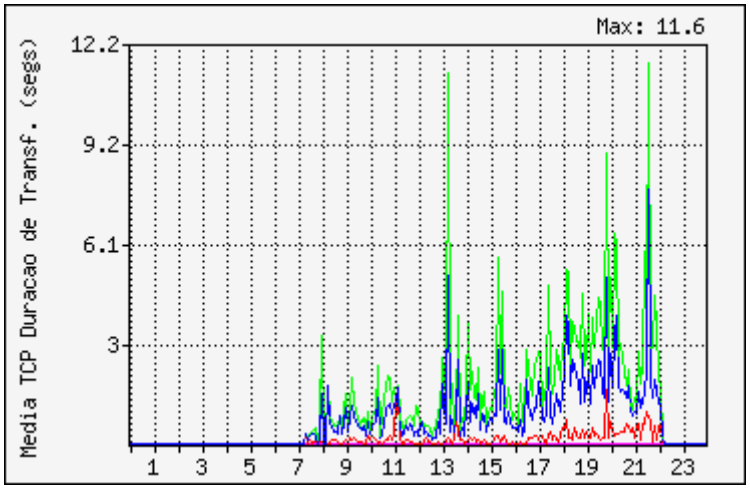
A tabela 28 apresenta os resultados obtidos referentes a distribuição de requisições em relação a latência no lote S-04-A. A latência corresponde ao tempo de resposta de uma requisição realizada por um cliente. A coluna “Latência” apresenta este tempo de resposta em milisegundos. É possível observar que 81,15% das requisições foram atendidas em menos de um segundo. A figura 48 também apresenta um gráfico da duração média de transferência (latência) no lote S-04-A.

Os campos da tabela 28 correspondem aos campos da tabela 14, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

As figuras 47 e 48 apresentam os gráficos de *hit rate* e *byte hit rate* respectivamente, para o lote S-04-A.

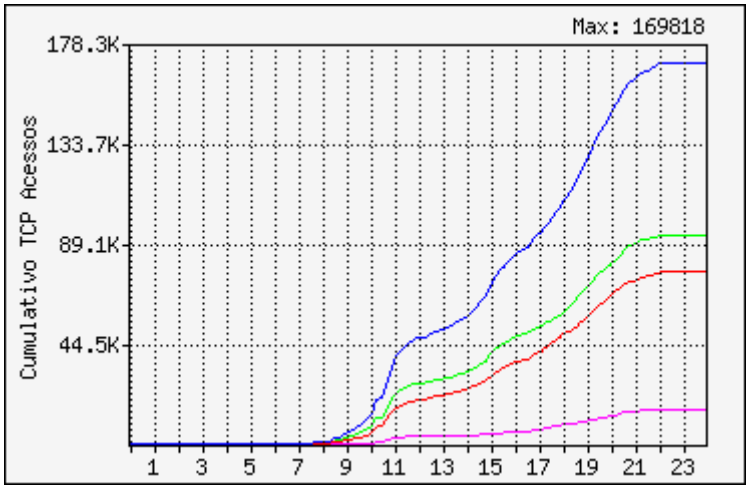
TABELA 28 - REQUISIÇÕES POR LATÊNCIA NO LOTE S-04-A.

Latência (ms)	Requisições	%R	HR	s/Req.	*Bytes*	%B	BHR	KB/s
<= 0,1	4.793	2,82	99,17	0	6.308.770	0,63	99,69	12.854
<= 0,2	4.793	2,82	99,17	0	6.308.770	0,63	99,69	12.854
<= 0,5	4.793	2,82	99,17	0	6.308.770	0,63	99,69	12.854
<= 1	10.970	6,46	98,69	0	14.405.477	1,43	99,52	2.113,46
<= 2	16.359	9,63	97,65	0	21.910.414	2,18	98,85	1.227,29
<= 5	30.221	17,78	95,44	0	45.357.417	4,51	96,99	6.17,43
<= 10	43.733	25,73	94,52	0	70.574.619	7,01	95,7	389,28
<= 20	59.029	34,73	93,52	0,01	101.388K	10,32	94,59	249,92
<= 50	63.415	37,32	91,18	0,01	125.867K	12,81	92,97	228,08
<= 100	70.548	41,51	85,84	0,02	150.201K	15,28	89,23	137,67
<= 200	80.515	47,38	78,72	0,03	194.406K	19,78	78,92	74,49
<= 500	115.611	68,03	60,64	0,13	302.057K	30,73	64,29	20,72
<= 1.000	137.913	**81,15**	53,15	0,22	426.902K	43,44	50,35	14
<= 2.000	151.914	89,39	49,49	0,33	589.502K	59,98	39,24	11,77
<= 5.000	162.657	95,71	46,93	0,52	714.739K	72,72	33,94	8,37
<= 10.000	166.531	97,99	46,01	0,67	787.419K	80,12	31,63	7,02
<= 20.000	168.734	99,29	45,48	0,84	864.942K	88	29,77	6,11
<= 50.000	169.565	99,78	45,28	0,98	926.164K	94,23	28,75	5,59
<= 100.000	169.773	99,9	45,24	1,06	952.656K	96,93	28,63	5,29
<= 200.000	169.831	99,93	45,22	1,11	962.206K	97,9	28,46	5,11
<= 500.000	169.927	99,99	45,2	1,26	982.306K	99,94	27,88	4,58
<= 1.000.000	169.941	100	45,2	1,33	982.658K	99,98	27,87	4,35
<= 1e10	169.942	100	45,2	1,34	982.847K	100	27,87	4,32
Total	**169.942**	**100**	**45,2**	**1,34**	**982.847K**	**100**	**27,87**	**4,32**



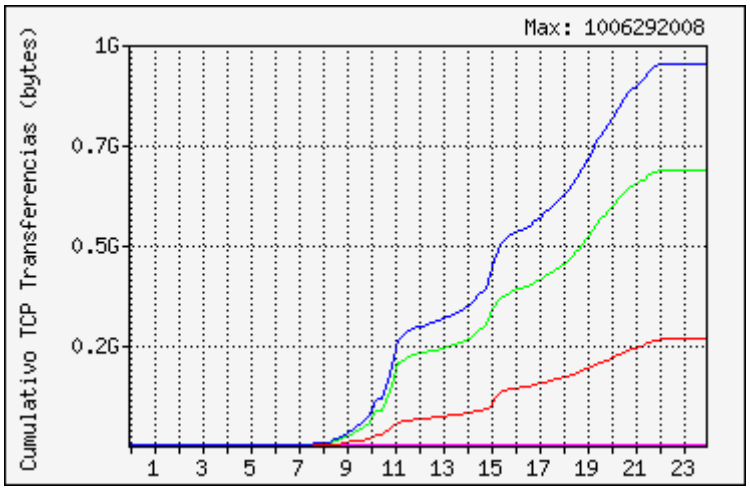
Duração Média de Transferência: 0,73 segundos
Duração Média de *Cache Hit*: 0,14 segundos
Duração Média de *Cache Miss*: 1,24 segundos

FIGURA 46 - GRÁFICO DE DURAÇÃO MÉDIA DE TRANSFERÊNCIA (LATÊNCIA) NO LOTE S-04-A.



Total de Acessos: 169818
Média de Acessos: 7075,75 por hora
Total de *Cache Hits*: 76806
Média de *Cache Hits*: 3200,25 por hora
% de *Cache Hits*: 45,22 %
Total de *Cache IMS Hits*: 15693
Média de *Cache IMS Hits*: 653,87 por hora
Total de *Cache Misses*: 93009
Média de *Cache Misses*: 3875,37 por hora
% de *Cache Misses*: 54,76 %

FIGURA 47 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-04-A.



Total Transferido: 982,847 MB
Média Transferida: 41,9 MB por hora
Total de *Cache Hits*: 280,4 MB
Média de *Cache Hits*: 11,6 MB por hora
% de *Cache Hits*: 27,87 %
Total de *Cache IMS Hits*: 3,8 MB
Média de *Cache IMS Hits*: 160,5 Kb por hora
Total de *Cache Misses*: 725,8 MB
Média de *Cache Misses*: 30,2 MB por hora
% de *Cache Misses*: 72,12 %

FIGURA 48 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-04-A.

A tabela 29 apresenta a distribuição de requisições do lote S-04-A, classificadas pelo campo *status-code* do protocolo HTTP. O campo *status-code* é retornado como parte da mensagem de resposta HTTP, indicando o resultado da requisição. Diferentemente dos lotes da PUCPR, o lote S-04-A apresentou uma quantidade não significativa de requisições com *status-code* correspondente a erro.

TABELA 29 - REQUISIÇÕES POR *STATUS-CODE* NO LOTE S-04-A.

status-code	**Requisições**	**%R**	***Bytes***	**%B**
000 (Mais usado com tráfego UDP)	1.617	0,95	0	0
200 (OK)	135.472	79,72	967.019K	98,39
204 (*No Content*)	105	0,06	37.644	0
206 (*Partial Content*)	23	0,01	100.7842	0,1
301 (*Moved Permanently*)	651	0,38	292.191	0,03
302 (*Moved Temporarily*)	5.069	2,98	4.190.219	0,42
304 (*Not Modified*)	23.899	14,06	5.688.105	0,57
307 (*Temporary Redirect*)	1	0	623	0
400 (*Bad Request*)	263	0,15	338.558	0,03
401 (*Unauthorized*)	699	0,41	285.001	0,03
403 (*Forbidden*)	155	0,09	107.248	0,01
404 (*Not Found*)	1.373	0,81	1.987.400	0,2
405 (*Method Not Allowed*)	8	0	4.322	0
500 (*Internal Server Error*)	354	0,21	1.943.065	0,19
502 (*Bad Gateway*)	72	0,04	119.615	0,01
503 (*Service Unavailable*)	115	0,07	127.792	0,01
504 (*Gateway Timeout*)	63	0,04	77.218	0,01
600 (*Squid header parsing error*)	3	0	1.065	0
Total	**169.942**	**100**	**982.847K**	**100**

Os campos da tabela 29 correspondem aos campos da tabela 15, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

7.1.4 Análise do Lote S-05-A

O lote S-05-A também corresponde a requisições realizadas ao servidor *proxy Squid* que atende aos laboratórios da SPEI. Apesar de terem sido realizadas análises tantos nos *logs* da rede administrativa quanto nos *logs* da rede de laboratórios, somente os *logs* da rede de laboratórios foram utilizados para a demonstração de resultados, por apresentarem maior diversidade em relação aos assuntos de interesse da comunidade de usuários. Este aspecto é importante principalmente para a avaliação do desempenho da estratégia de substituição LSR-VM.

Este lote corresponde às requisições de objetos realizados pela comunidade de usuários da SPEI no período entre 00:00 hora do dia 05 de julho de 2005 e 00:00 hora do dia 06 de julho de 2005. O objetivo da análise deste lote é apresentar as características de requisições da comunidade de usuários da SPEI, na rede de laboratórios.

Ao todo foram computadas 190.958 requisições de objetos neste lote. Estas requisições foram originadas a partir de 58 endereços IP distintos. Atualmente a rede de laboratórios da SPEI conta com cinco laboratórios totalizando 100 estações de trabalho. A maioria das requisições foi realizada através do método de requisição “GET” do protocolo HTTP. A partir da análise da tabela 30 é possível observar que aproximadamente 95,73% (somatória de requisições realizadas através dos métodos “GET” e “HEAD”) das requisições são passíveis de *cache*.

TABELA 30 - REQUISIÇÕES POR *REQUEST-METHO*D NO LOTE S-05-A.

request-method	**Requisições**	**%R**	**s/Rec.**	***Bytes***	**%B**	**KB/s**
GET	182.788	95,72	0,6	1.186.358K	92,82	10,82
POST	3.723	1,95	1,69	46.397.204	3,55	7,22
CONNECT	3.509	1,84	7,11	47.103.571	3,6	1,84
PROPFIND	914	0,48	0,69	459.896	0,04	0,72
OPTIONS	12	0,01	0,09	5.570	0	5,15
HEAD	12	0,01	0,22	3.606	0	1,34
Total	**190.958**	**100**	**0,74**	**1.278.125K**	**100**	**9,03**

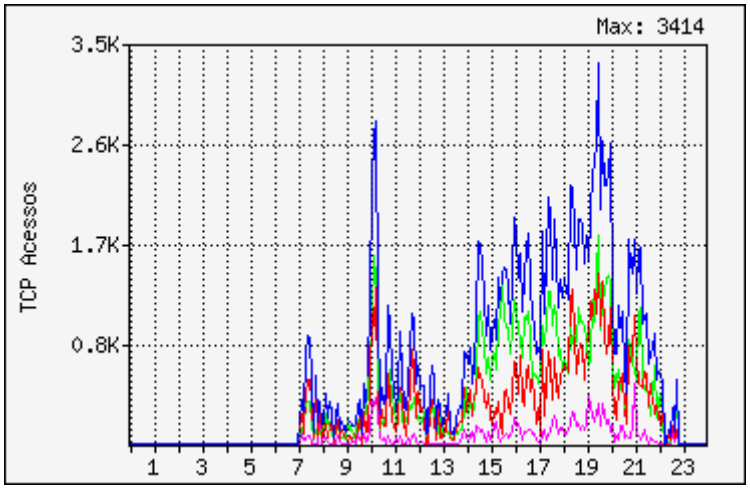
Os campos da tabela 30 correspondem aos campos da tabela 9, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

Este lote apresentou um taxa de acerto no *cache* (*Hit Rate*) bem melhor do que os lotes obtidos na PUCPR. Este desempenho se deve ao fato de que, na rede da SPEI as restrições a URI não autorizadas são estabelecidas através de regras no *firewall* preservando o *cache*. Em contra partida, na rede da PUCPR as restrições de acesso são estabelecidas no servidor *proxy Squid*, gerando registros nos *logs* e influenciando o *hit rate*. O lote S-05-A apresentou 43,39% de *cache hit*, conforme pode ser observado na tabela 31, que apresenta as requisições classificadas por *result-code*, e na figura 49 que apresenta um gráfico de requisições no lote S-05-A.

Os campos da tabela 31 correspondem aos campos da tabela 10, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

TABELA 31 - REQUISIÇÕES POR *RESULT-CODE* NO LOTE S-05-A.

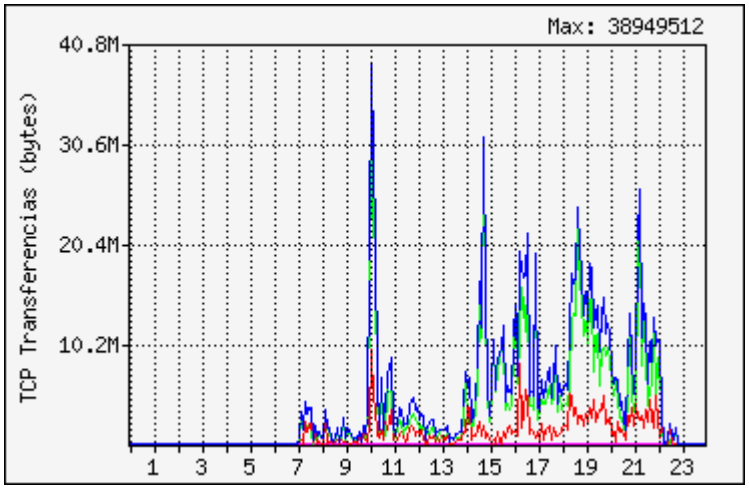
Status	**Requisições**	**%R**	**s/Req.**	***Bytes***	**%B**	**KB/s**
HIT	82.864	43,39	0,1	286.128K	22,39	35,08
MISS	107.358	56,22	1,14	991.706K	77,59	8,1
ERROS	736	0,39	14,84	297.768	0,02	0,03
Total	**190.958**	**100**	**0,74**	**1.278.125K**	**100**	**9,03**



Total de Acessos: 190806
Média de Acessos: 7950,25 por hora
Total de *Cache Hits*: 82864
Média de *Cache Hits*: 3452,66 por hora
% de *Cache Hits*: 43,39 %
Total de *Cache IMS Hits*: 17017
Média de *Cache IMS Hits*: 709,04 por hora
Total de *Cache Misses*: 107936
Média de *Cache Misses*: 4497,33 por hora
% de *Cache Misses*: 56,59 %

FIGURA 49 - GRÁFICO DE REQUISIÇÕES NO LOTE S-05-A.

O tamanho total dos objetos requisitados no lote S-05-A corresponde a 1.278.125 KB, sendo que o total de banda salva pelas requisições que resultaram em *cache hit* corresponde a 279 MB. Isto significa que a taxa de acerto no *cache*, em relação ao tamanho dos objetos (*byte hit rate*) corresponde a 22,39% no lote S-05-A. Este desempenho foi superior aos lotes obtidos na PUCPR. É possível observar na figura 49 que existem picos de requisições, bem característicos, por volta das 10:00 e 19:00 horas. Da mesma forma, é possível observar através da figura 50 que existem picos de transferências de *bytes* por volta das 10:00, 15:00, 17:00, 19:00 e 21:00 horas no lote S-05-A.



Total Transferido: 1,28 GB
Média Transferida: 54,5 MB por hora
Total de *Cache Hits*: 279,9 MB
Média de *Cache Hits*: 12,2 MB por hora
% de *Cache Hits*: 22,39 %
Total de *Cache IMS Hits*: 4,1 MB
Média de *Cache IMS Hits*: 174,1 Kb por hora
Total de *Cache Misses*: 1 GB
Média de *Cache Misses*: 42,3 MB por hora
% de *Cache Misses*: 77,61 %

FIGURA 50 - GRÁFICO DE *BYTES* TRANSFERIDOS NO LOTE S-05-A.

Uma análise mais detalhada das requisições classificadas por *content-type* pode ser observada na tabela 32. O *content-type* é um campo que faz parte do cabeçalho da mensagem de resposta HTTP.

É possível observar na tabela 32 que apenas 1,89% das requisições foram consideradas como erro, por se tratar de tráfego não autorizado. Porém, em 4,54% das requisições o servidor *web* não informou o conteúdo do campo *content-type*. Neste caso estas requisições foram identificadas como *content-type* desconhecido.

Da mesma forma que nos lotes da PUCPR, grande parte das requisições do lote S-05-A refere-se a objetos binários, mais especificamente a imagens. É possível observar que 31,21% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “gif”. Da mesma forma, é possível observar que 18,96% das requisições apresentam *content-type* informando que o objeto retornado trata-se de uma imagem no formato “jpeg”. Outras requisições com *content-type* igual a “image/png”, “image/bmp” e “image/jpeg” também se referem a objetos que representam imagens. Ao todo, 52,12% das requisições correspondem a imagens.

Os campos da tabela 32 correspondem aos campos da tabela 11, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

TABELA 32 - REQUISIÇÕES POR *CONTENT-TYPE* NO LOTE S-05-A.

content-type	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
image/gif	59.604	31,21	73,55	96.506.422	7,37
text/html	56.947	29,82	11,79	384.667K	30,1
image/jpeg	36.215	18,96	47,74	269.038K	21,05
<desconhecidas>	8.667	4,54	6,98	225.1160	0,17
application/x-javascript	8.573	4,49	35,73	52.063.178	3,98
<erros>	3.613	1,89	41,1	5.129.931	0,39
text/css	3.585	1,88	70,79	16.470.920	1,26
<seguras> (HTTPS)	3.509	1,84	0	47.103.571	3,6
image/png	2.912	1,52	94,68	5.138.455	0,39
application/x-shockwave-flash	2.668	1,4	62,67	70.532.328	5,39
text/plain	2.615	1,37	79,04	30.776.979	2,35
image/bmp	721	0,38	85,71	1.287.373	0,1
application/octet-stream	226	0,12	35,4	25.912.620	1,98
text/javascript	221	0,12	1,36	1.235.520	0,09
video/mpeg	175	0,09	14,86	143.216K	11,21
image/pjpeg	102	0,05	0	8.102.649	0,62
text/xml	76	0,04	38,16	391.921	0,03
httpd/yahoo-send-as-is	65	0,03	1,54	34.740	0
application/pkix-crl	63	0,03	0	620.350	0,05
video/x-ms-asf	49	0,03	0	126.641	0,01
Outras: 40 *content-types*	352	0,18	22,16	126.043K	9,86
Total	**190.958**	**100**	**43,39**	**1.278.125K**	**100**

Somando-se os percentuais correspondentes a requisições a objetos texto (*content-type* igual a “text/html”, “text/plain”, “text/xml”), tem-se um total de 31,23%. É importante observar que este percentual não reflete o número real de objetos texto retornados como resposta a requisições, pois não existe um grau aceitável de confiabilidade no valor do campo *content-type* retornado por servidores *web*.

Para que fosse possível obter um valor mais confiável para o percentual de objetos texto requisitados no lote S-05-A, foi preciso realizar uma análise considerando-se a extensão da URI (*Uniform Resource Identifier*) das requisições. A tabela 33 apresenta a quantidade de requisições classificadas pela extensão da URI.

TABELA 33 - REQUISIÇÕES POR EXTENSÃO DE ARQUIVO NO LOTE S-05-A.

Extensões	**Requisições**	**%R**	**HR**	***Bytes***	**%B**
gif	62.194	32,57	72,7	94.108.065	7,19
<dinâmicas>	52.000	27,23	3,52	462.816K	36,21
jpg	38.930	20,39	56,66	197.623K	15,46
<nenhuma>	9.165	4,8	4,68	57.680.196	4,41
js	5.650	2,96	67,49	56.177.810	4,29
css	4.700	2,46	51,47	15.560.554	1,19
<erros>	3.613	1,89	41,1	5.129.931	0,39
<seguras> (HTTPS)	3.509	1,84	0	47.103.571	3,6
png	2.909	1,52	94,88	5.130.860	0,39
asp	1.664	0,87	0,24	17.152.245	1,31
swf	1.557	0,82	74,25	60.762.488	4,64
htm	936	0,49	45,83	9.158.474	0,7
bmp	720	0,38	85,97	1.160.892	0,09
html	704	0,37	29,97	4.926.745	0,38
php	411	0,22	1,22	4.921.468	0,38
jsp	316	0,17	0	632.647	0,05

JPG	266	0,14	9,4	4.138.190	0,32
GIF	252	0,13	76,59	1.210.122	0,09
tpl	176	0,09	0	672.115	0,05
mpg	170	0,09	15,29	132.378K	10,36
Outras: 72 extensões	1.116	0,58	16,58	108.720K	8,51
Total	**190.958**	**100**	**43,39**	**1.278.125K**	**100**

A partir da análise da tabela 33 é possível observar que apenas 2,12% correspondem a objetos texto. Este percentual foi obtido através da soma das requisições referentes às extensões de URI: “jsp”, “php”, “asp”, “html” e “htm”.

Os campos da tabela 33 correspondem aos campos da tabela 12, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

É importante observar que os objetos que representam imagens correspondem na verdade a 55,13 % das requisições (somatório das extensões “gif”, “jpg”, “png”, “bmp”, “JPG” e “GIF”).

As análises realizadas no lote S-05-A contemplaram também a distribuição dos objetos requisitados por tamanho. É possível observar através da tabela 34 que 47,09% da taxa de acerto (*hit rate*) correspondem a objetos que possuem tamanhos variando entre 100 e 999 bytes. Estes objetos correspondem a 50,9% das requisições do lote S-05-A.

Os campos da tabela 34 correspondem aos campos da tabela 13, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

TABELA 34 - DISTRIBUIÇÃO DE OBJETOS POR TAMANHO NO LOTE S-05-A.

Tamanho do Objeto (*bytes*)	**Requisições**	**HR**	**s/Req.**	***Bytes***	**BHR**	**KB/s**
0-0	1.363	3,01	6,89	0	0	0
10-99	60	0	5,29	2475	0	0,01
100-999	97.131	47,09	0,34	46.575.531	41,96	1,4
1.000-9.999	70.358	45,87	0,78	218.877K	42,93	3,96
10.000-99.999	21.307	21,94	1,72	583.307K	18,1	15,91
100.000-999.999	594	18,52	6,27	142.121K	18,28	38,14
1.000.000-9.999.999	145	16,55	25,18	288.335K	14,4	78,96
Total	**190.958**	**43,39**	**0,74**	**1.278.125K**	**22,39**	**9,03**

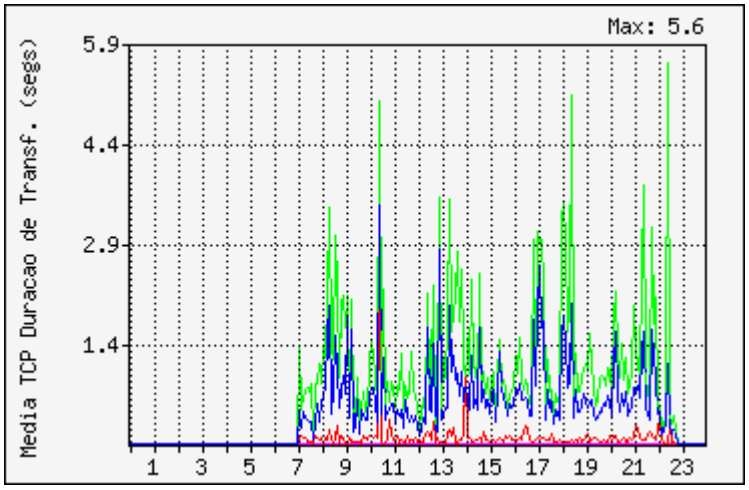
A tabela 35 apresenta os resultados obtidos em relação a distribuição de requisições referentes a latência no lote S-05-A. A latência corresponde ao tempo de resposta de uma requisição realizada por um cliente. A coluna “Latência” apresenta este tempo de resposta em milisegundos. É possível observar que 90,99% das requisições foram atendidas em menos de um segundo. A figura 53 também apresenta um gráfico da duração média de transferência (latência) no lote S-05-A.

Os campos da tabela 35 correspondem aos campos da tabela 14, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

As figuras 52 e 53 apresentam os gráficos de *hit rate* e *byte hit rate* respectivamente, para o lote S-05-A.

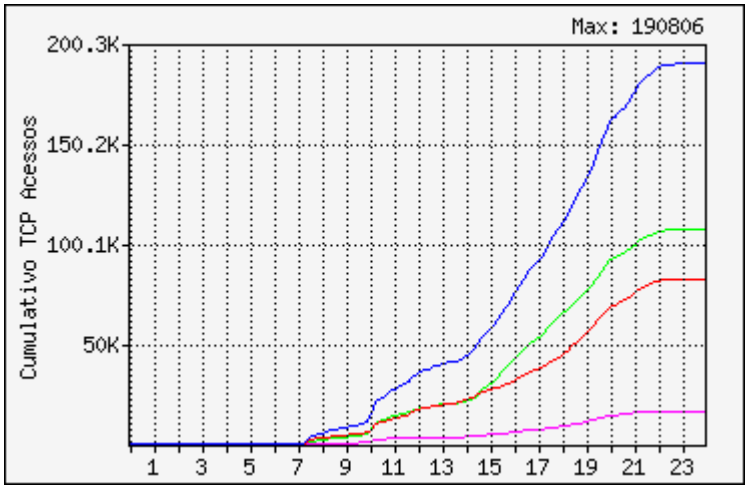
TABELA 35 - REQUISIÇÕES POR LATÊNCIA NO LOTE S-05-A.

Latência (ms)	Requisições	%R	HR	s/Req.	*Bytes*	%B	BHR	KB/s
<= 0,1	5.165	2,7	99,83	0	6.673.345	0,51	99,86	12.617
<= 0,2	5.165	2,7	99,83	0	6.673.345	0,51	99,86	12.617
<= 0,5	5.165	2,7	99,83	0	6.673.345	0,51	99,86	12.617
<= 1	12.767	6,69	99,47	0	16.580.493	1,27	99,68	1.994,44
<= 2	19.597	10,26	98,88	0	25.809.473	1,97	99,28	1.157,31
<= 5	34.753	18,2	97,74	0	53.581.292	4,09	98,22	649,31
<= 10	49.487	25,92	97,13	0	83.259.874	6,36	97,66	415,55
<= 20	65.111	34,1	96,63	0,01	115.504K	9,04	97,11	269,98
<= 50	70.509	36,92	93,35	0,01	141.400K	11,06	95,19	227,41
<= 100	83.926	43,95	82,74	0,02	175.919K	13,76	88	107,84
<= 200	97.462	51,04	74,12	0,04	231.809K	18,14	74,5	62,61
<= 500	154.916	81,13	51,53	0,15	393.263K	30,77	53,44	16,49
<= 1000	173.758	**90,99**	47,11	0,21	554.756K	43,4	42,51	15,15
<= 2000	183.955	96,33	44,9	0,28	792.245K	61,98	32,44	15,65
<= 5000	187.966	98,43	44,06	0,33	933.581K	73,04	28,56	14,95
<= 10000	189.305	99,13	43,76	0,38	1.013.481K	79,29	26,81	14,19
<= 20000	190.198	99,6	43,56	0,44	1.121.188K	87,72	24,82	13,47
<= 50000	190.633	99,83	43,46	0,5	1.215.527K	95,1	23,2	12,63
<= 100000	190.824	99,93	43,42	0,57	1.255.050K	98,19	22,78	11,47
<= 200000	190.886	99,96	43,41	0,62	1.277.402K	99,94	22,38	10,83
<= 500000	190.952	100	43,4	0,71	1.278.096K	100	22,39	9,46
<= 1000000	190.955	100	43,39	0,72	1.278.115K	100	22,39	9,29
<= 1E+10	190.958	100	43,39	0,74	1.278.125K	100	22,39	9,03
Total	**190.958**	**100**	**43,39**	**0,74**	**1.278.125K**	**100**	**22,39**	**9,03**



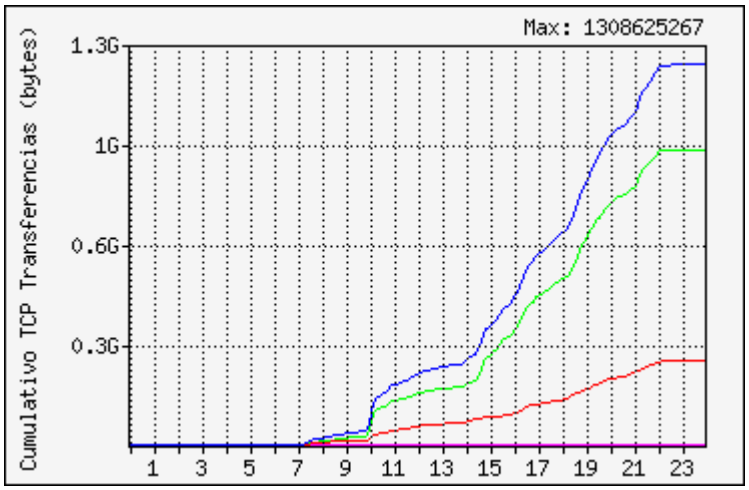
Duração Média de Transferência: 0,49 segundos
Duração Média de *Cache Hit*: 0,06 segundos
Duração Média de *Cache Miss*: 0,86 segundos

FIGURA 51 - GRÁFICO DE DURAÇÃO MÉDIA DE TRANSFERÊNCIA (LATÊNCIA) NO LOTE S-05-A.



Total de Acessos: 190806
Média de Acessos: 7950,25 por hora
Total de *Cache Hits*: 82864
Média de *Cache Hits*: 3452,66 por hora
% de *Cache Hits*: 43,39 %
Total de *Cache IMS Hits*: 17017
Média de *Cache IMS Hits*: 709,04 por hora
Total de *Cache Misses*: 107936
Média de *Cache Misses*: 4497,33 por hora
% de *Cache Misses*: 56,59 %

FIGURA 52 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-05-A.



Total Transferido: 1,28 GB
Média Transferida: 54,5 MB por hora
Total de *Cache Hits*: 279,9 MB
Média de *Cache Hits*: 12,2 MB por hora
% de *Cache Hits*: 22,39 %
Total de *Cache IMS Hits*: 4,1 MB
Média de *Cache IMS Hits*: 174,1 Kb por hora
Total de *Cache Misses*: 1 GB
Média de *Cache Misses*: 42,3 MB por hora
% de *Cache Misses*: 77,61 %

FIGURA 53 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-05-A.

A tabela 36 apresenta a distribuição de requisições do lote S-05-A, classificadas pelo campo *status-code* do protocolo HTTP. O campo *status-code* é retornado como parte da mensagem de resposta HTTP, indicando o resultado da requisição. Diferentemente dos lotes da PUCPR, o lote S-05-A apresentou uma quantidade não significativa de requisições com *status-code* correspondente a erro.

TABELA 36 - REQUISIÇÕES POR *STATUS-CODE* NO LOTE S-05-A.

status-code	**Requisições**	**%R**	***Bytes***	**%B**
000 (Mais usado com tráfego UDP)	1.256	0,66	0	0
200 (OK)	152.542	79,88	1.260.351K	98,61
204 (*No Content*)	121	0,06	31.347	0
206 (*Partial Content*)	23	0,01	2.439.342	0,19
301 (*Moved Permanently*)	454	0,24	202.217	0,02
302 (*Moved Temporarily*)	4.724	2,47	3.697.249	0,28
303 (*See Other*)	1	0	826	0
304 (*Not Modified*)	28.212	14,77	6.674.131	0,51
307 (*Temporary Redirect*)	1	0	711	0
400 (*Bad Request*)	366	0,19	506.252	0,04
401 (*Unauthorized*)	608	0,32	245.683	0,02
403 (*Forbidden*)	600	0,31	298.123	0,02
404 (*Not Found*)	1.588	0,83	3.198.650	0,24
405 (*Method Not Allowed*)	8	0	4.322	0
500 (*Internal Server Error*)	245	0,13	634.610	0,05
502 (*Bad Gateway*)	69	0,04	101.348	0,01
503 (*Service Unavailable*)	91	0,05	100.278	0,01
504 (*Gateway Timeout*)	38	0,02	40.665	0
999 (desconhecido)	11	0,01	24.419	0
Total	**190.958**	**100**	**1.278.125K**	**100**

Os campos da tabela 36 correspondem aos campos da tabela 15, descritos na seção 7.1.1 que apresenta a análise do lote P-04-A.

7.2 RESULTADOS DA SIMULAÇÃO DAS ESTRATÉGIAS DE SUBSTITUIÇÃO

Esta seção apresenta os resultados comparativos do desempenho das estratégias de substituição SIZE, LRU (*Least Recently Used*), LFU (*Least Frequently Used*) e LSR-VM (*Least Semantically Related – Vector Model*).

Os lotes de simulação extraídos dos *logs* dos servidores *proxy Squid* da PUCPR e da SPEI nos dias 04 e 05 de julho de 2005 foram utilizados para simulação das estratégias de substituição. Estes lotes foram apresentados no início deste capítulo, através da tabela 8. Para cada lote de simulação foi criado um gráfico comparativo do desempenho contendo cada uma das estratégias de substituição. Este gráfico apresenta a taxa de acerto (*hit rate*) para cada estratégia de substituição. Da mesma forma, foi criado um gráfico para cada lote de simulação apresentando um comparativo de desempenho baseado na quantidade de *bytes* transferidos (*byte hit rate*).

O *framework* de simulação foi parametrizado para as sessões de simulação, contemplando um *cache* com tamanho de 10 MB. Foram utilizados lotes de simulação com aproximadamente vinte mil requisições. Para permitir a comparação do desempenho das estratégias de substituição considerando deferentes tamanhos de *cache*, as sessões de simulação com o lote S-04-S foram realizadas utilizando *cache* com tamanhos de 5 MB, 10 MB e 20 MB. A tabela 37 apresenta informações referentes ao total de requisições e a quantidade total de *bytes* transferidos para cada lote de simulação.

A tabela 38 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote P-04-S.

TABELA 37 - INFORMAÇÕES SOBRE OS LOTES DE SIMULAÇÃO.

Lote	**Instituição**	**Requisições**	***Bytes***
P-04-S	PUCPR	19.953	1.108M
P-05-S	PUCPR	19.949	546M
S-04-S	SPEI	19.886	104M
S-05-S	SPEI	19.887	108M

TABELA 38 - RESULTADOS DA SIMULAÇÃO DO LOTE P-04-S.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	5707	28,6	5838K	0,51
LRU	2071	10,38	3226K	0,28
LFU	4877	24,44	6882K	0,61
LSR-VM	3113	15,6	4813K	0,42

A figura 54 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote P-04-S, utilizando *hit rate* como métrica. A figura 55 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote P-04-S, utilizando *byte hit rate* como métrica.

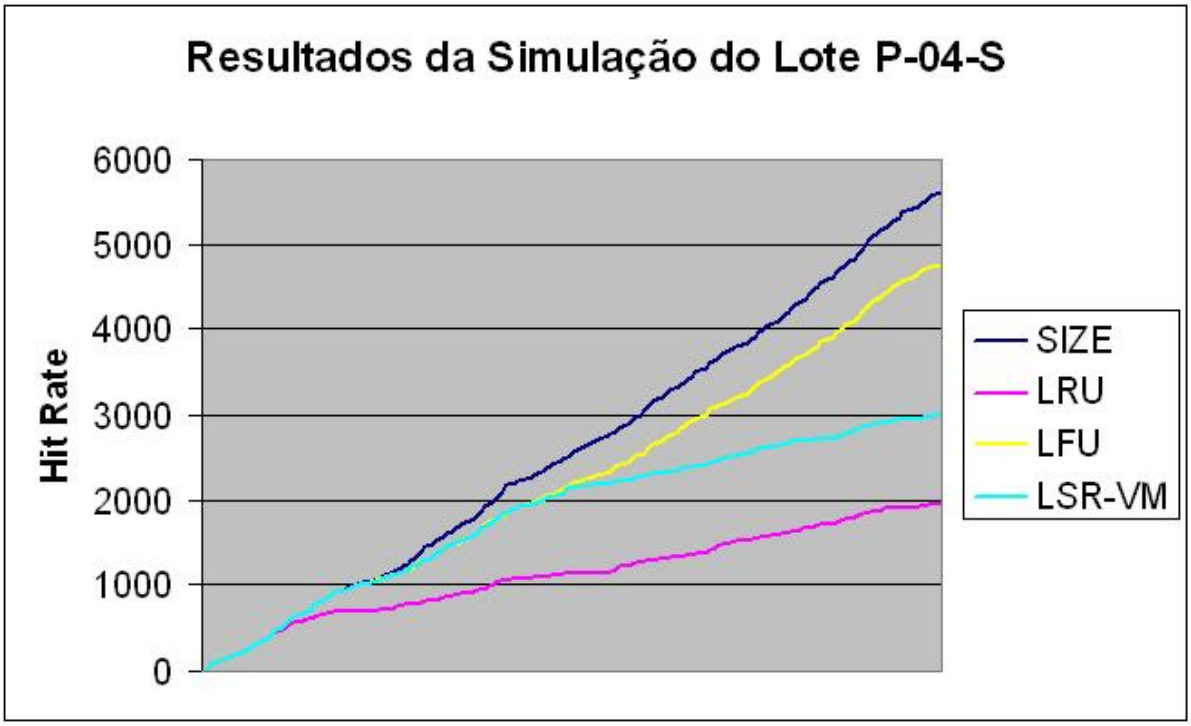


FIGURA 54 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE P-04-S.

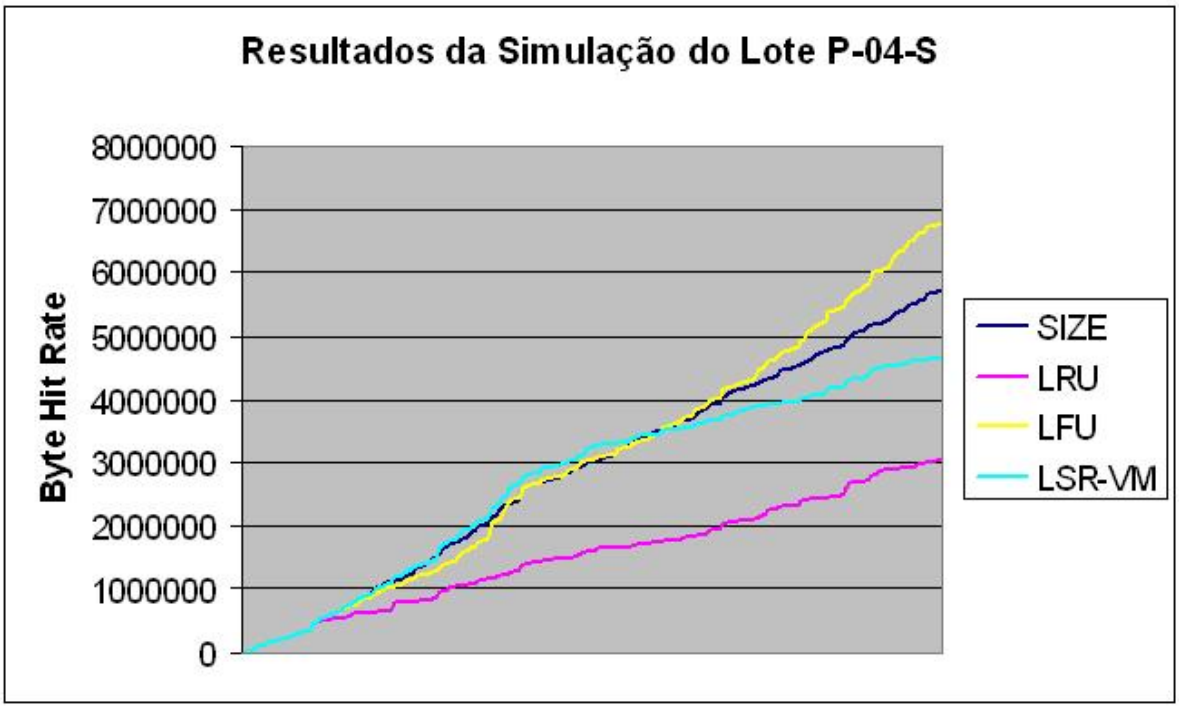


FIGURA 55 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE P-04-S.

A tabela 39 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote P-05-S. A figura 56 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote P-05-S, utilizando *hit rate* como métrica. A figura 57 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote P-05-S, utilizando *byte hit rate* como métrica.

TABELA 39 - RESULTADOS DA SIMULAÇÃO DO LOTE P-05-S.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	4902	24,57	6822K	1,22
LRU	1739	8,72	4581K	0,82
LFU	3649	18,29	6282K	1,12
LSR-VM	2475	12,41	5174K	0,93

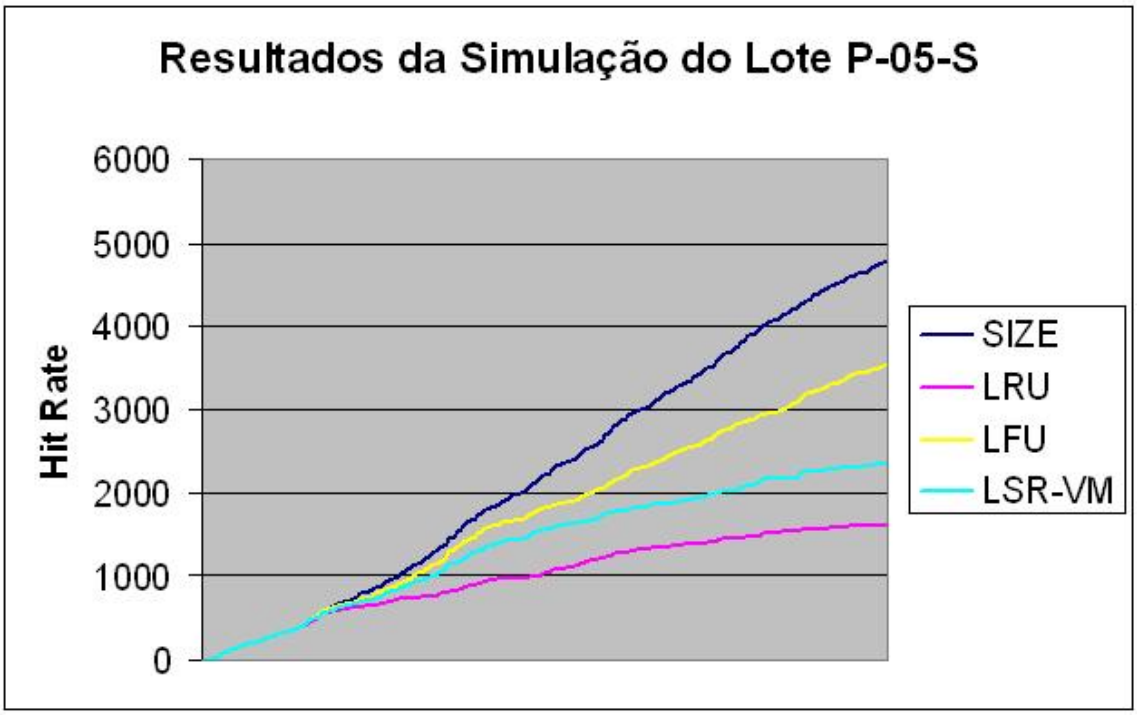


FIGURA 56 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE P-05-S.

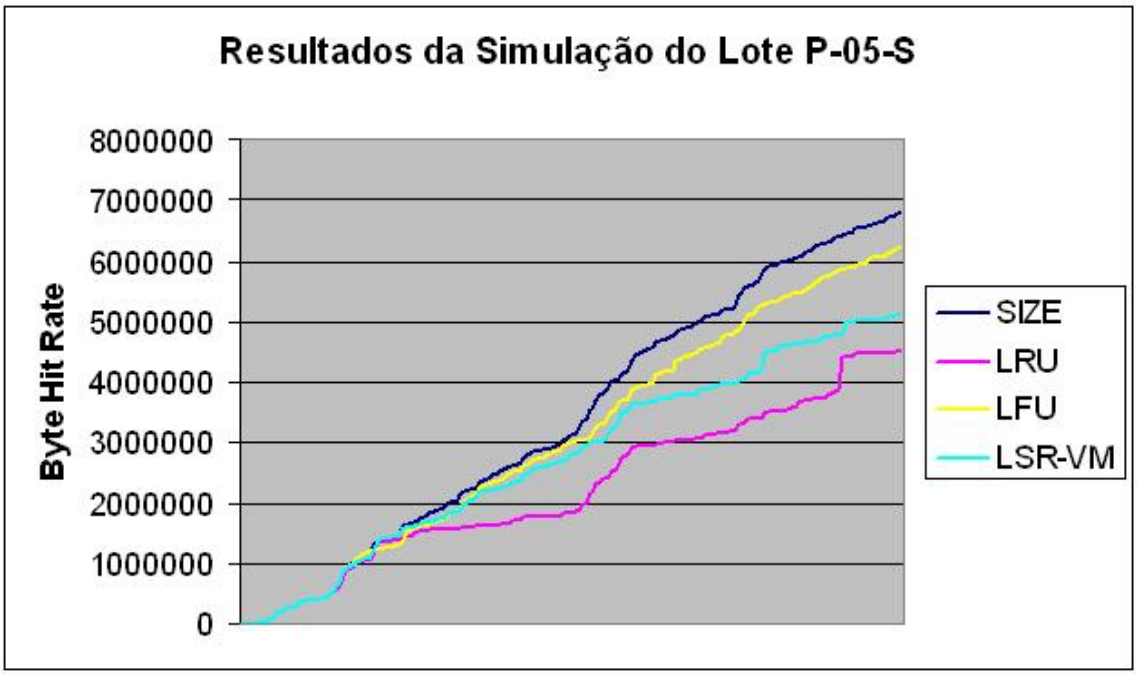


FIGURA 57 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE P-05-S.

A tabela 40 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote S-04-S com *cache* de 5 MB. A

figura 58 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 5 MB, utilizando *hit rate* como métrica. A figura 59 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 5 MB, utilizando *byte hit rate* como métrica.

TABELA 40 - RESULTADOS DA SIMULAÇÃO DO LOTE S-04-S COM *CACHE* DE 5 MB.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	5516	27,74	5283K	4,94
LRU	1898	9,54	4805K	4,49
LFU	4374	22	8746K	8,18
LSR-VM	3159	15,89	6689K	6,26

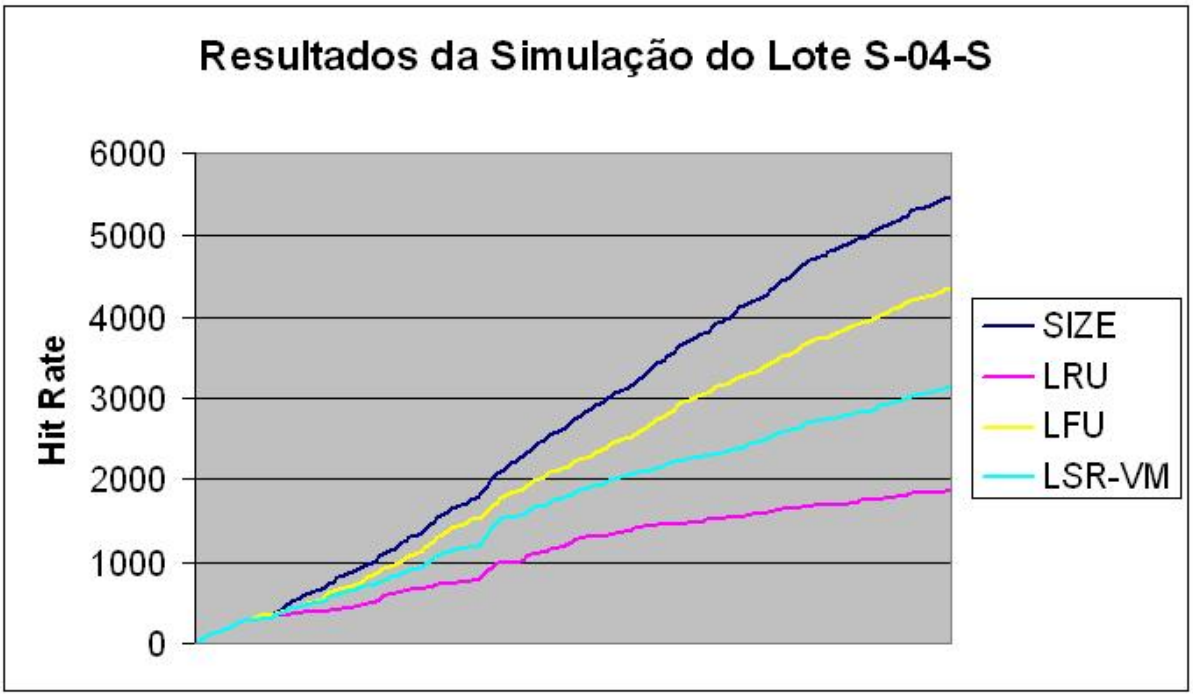


FIGURA 58 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 5 MB.

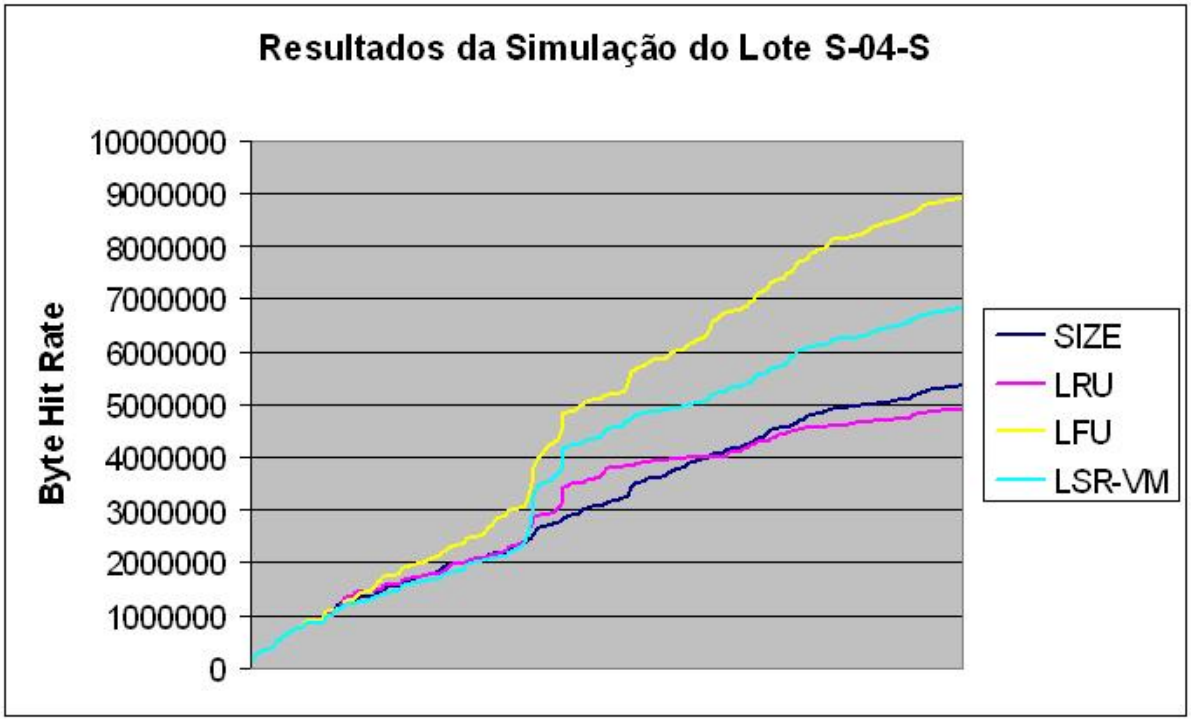


FIGURA 59 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 5 MB.

A tabela 41 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote S-04-S com *cache* de 10 MB. A figura 60 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 10 MB, utilizando *hit rate* como métrica. A figura 61 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 10 MB, utilizando *byte hit rate* como métrica.

TABELA 41 - RESULTADOS DA SIMULAÇÃO DO LOTE S-04-S COM *CACHE* DE 10 MB.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	5774	29,04	6925K	6,48
LRU	2774	13,95	6227K	5,82
LFU	4898	24,63	9970K	9,33
LSR-VM	3635	18,28	7696K	7,2

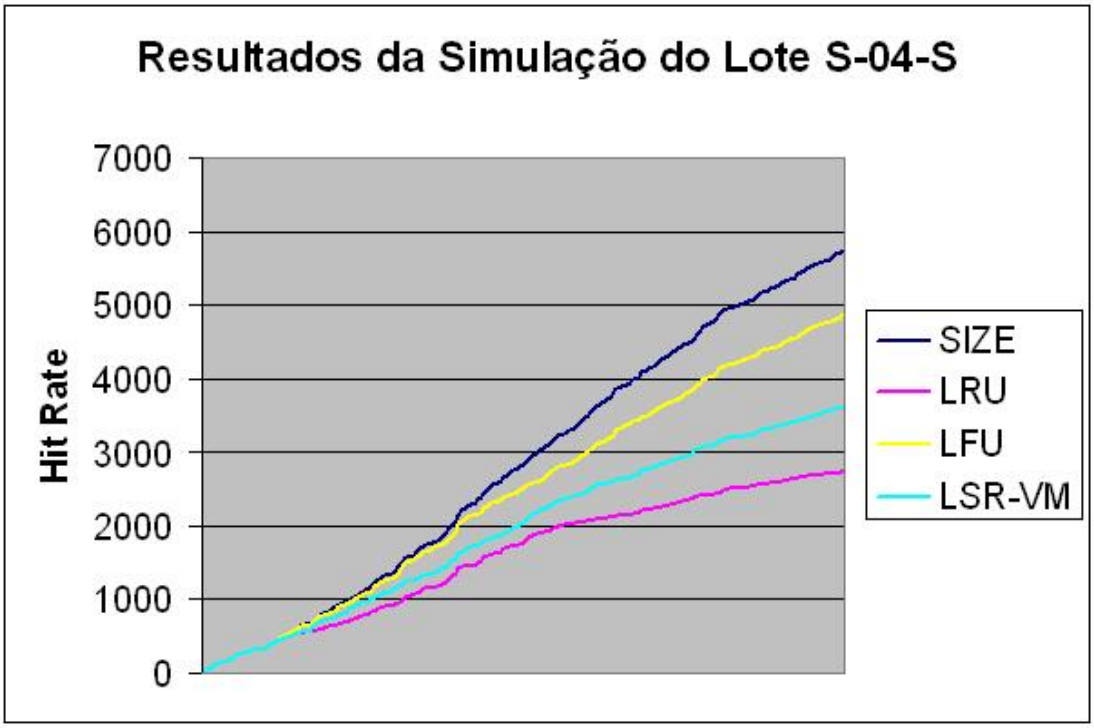


FIGURA 60 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 10 MB.

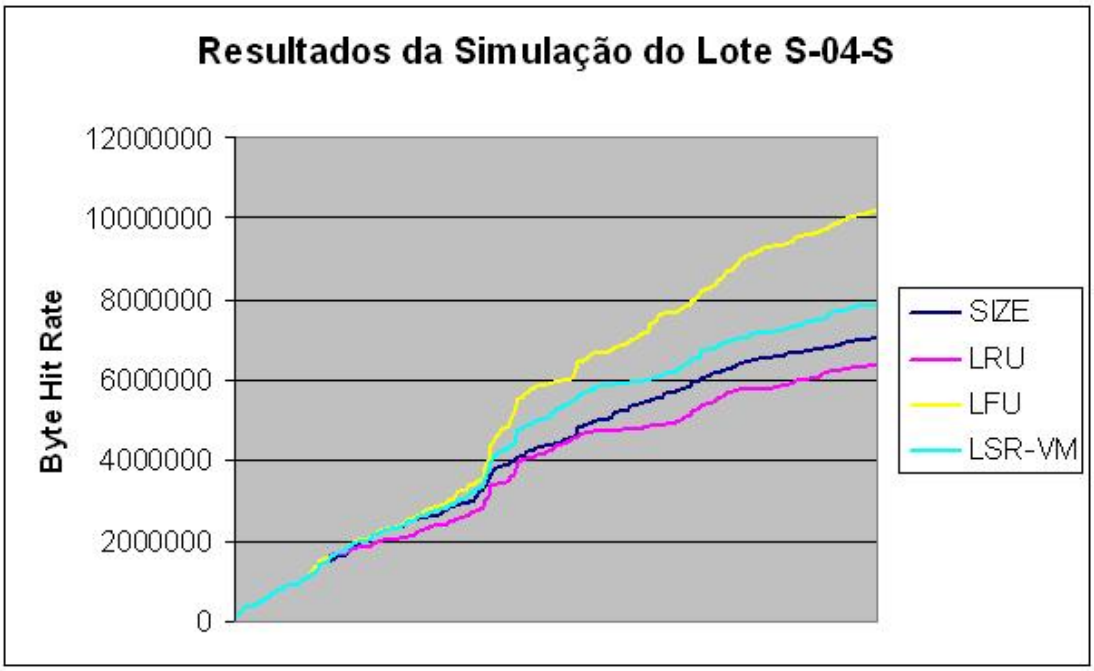


FIGURA 61 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 10 MB.

A tabela 42 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote S-04-S com *cache* de 20 MB. A figura 62 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 20 MB, utilizando *hit rate* como métrica. A figura 63 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-04-S com *cache* de 20 MB, utilizando *byte hit rate* como métrica.

TABELA 42 - RESULTADOS DA SIMULAÇÃO DO LOTE S-04-S COM *CACHE* DE 20 MB.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	5959	29,97	9596K	8,98
LRU	4058	20,41	9327K	8,73
LFU	5350	26,9	12126K	11,34
LSR-VM	5136	25,83	10816K	10,12

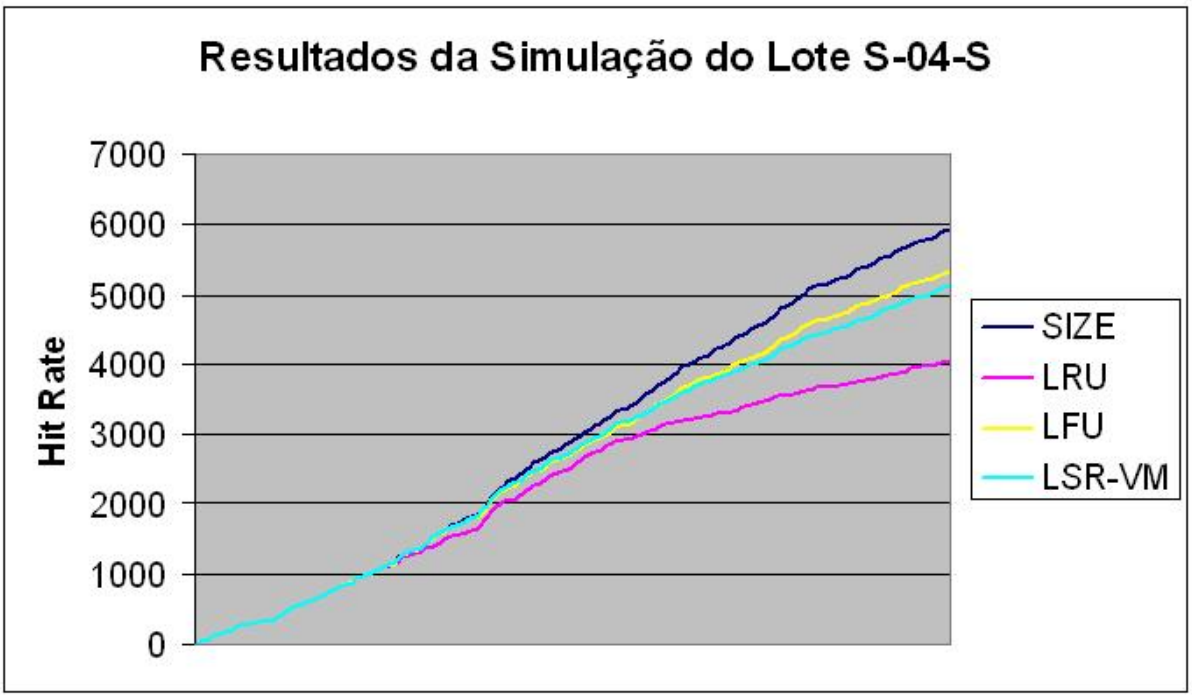


FIGURA 62 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 20 MB.

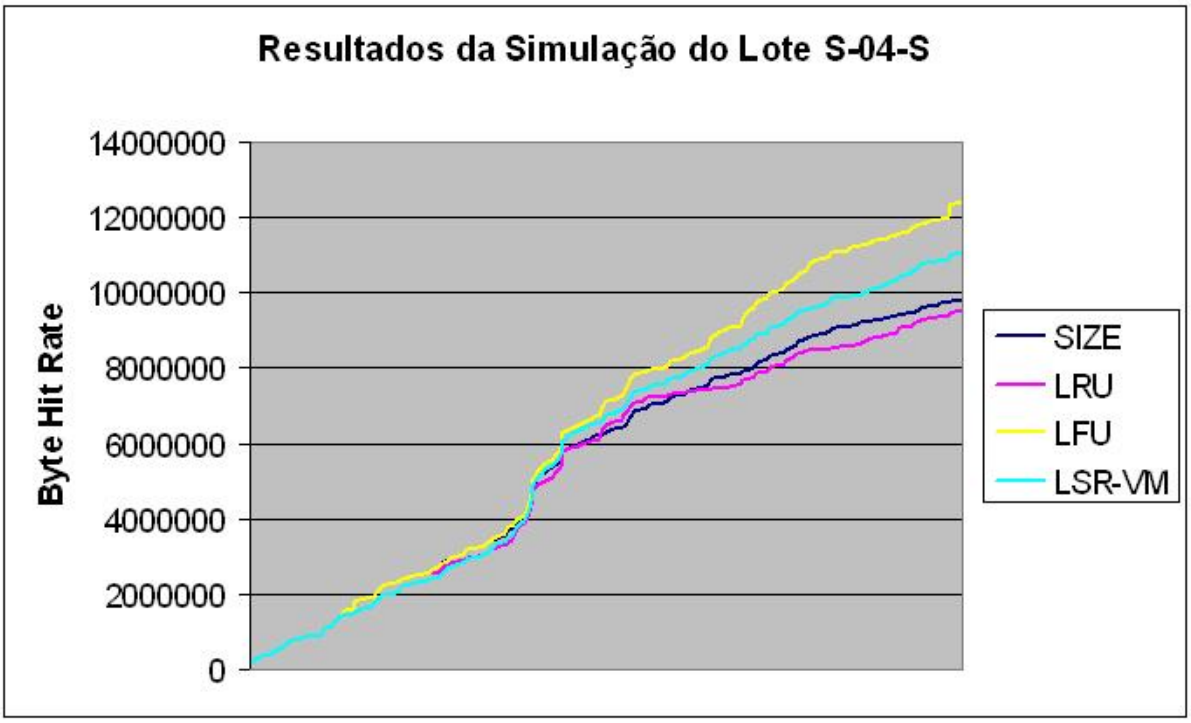


FIGURA 63 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-04-S COM *CACHE* DE 20 MB.

A tabela 43 apresenta os resultados das simulações das estratégias de substituição SIZE, LRU, LFU e LSR-VM para o lote S-05-S A figura 64 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-05-S, utilizando *hit rate* como métrica. A figura 65 apresenta o gráfico comparativo de desempenho das estratégias de substituição, resultante da simulação do lote S-05-S, utilizando *byte hit rate* como métrica.

TABELA 43 - RESULTADOS DA SIMULAÇÃO DO LOTE S-05-S.

Estratégia	***Hits***	***Hit Rate***	***Bytes***	***Byte Hit Rate***
SIZE	8382	42,15	13292K	12,04
LRU	2927	14,72	10500K	9,51
LFU	7214	36,27	19542K	17,7
LSR-VM	4437	22,31	13901K	12,59

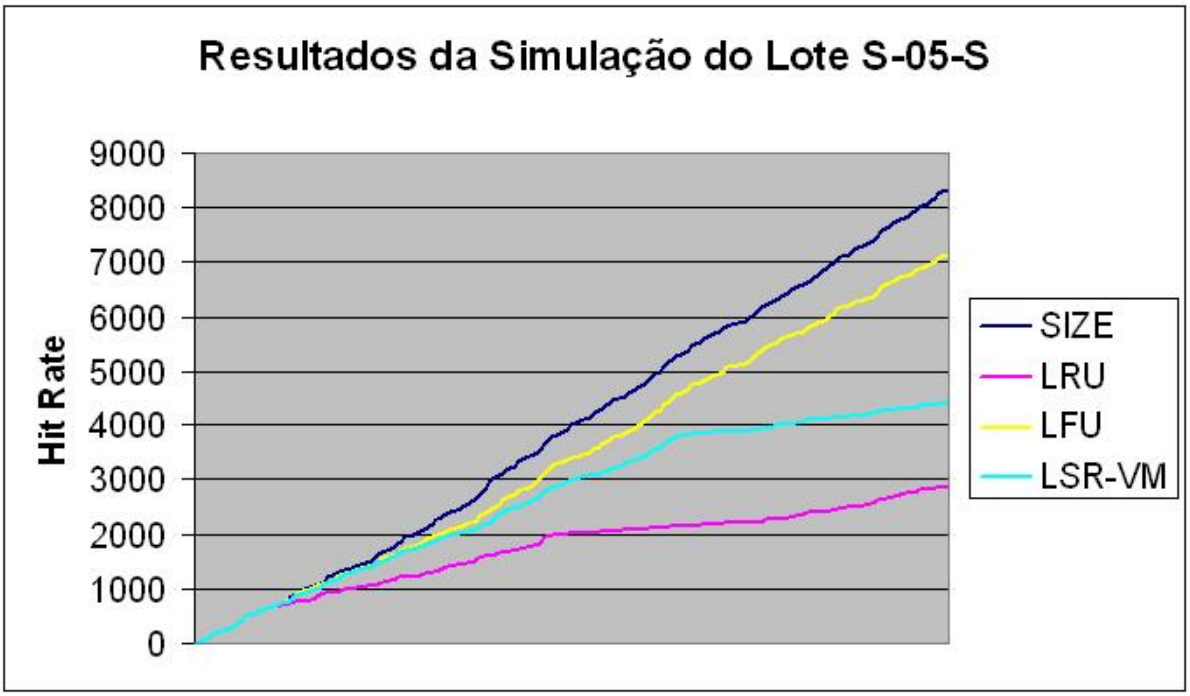


FIGURA 64 - GRÁFICO CUMULATIVO DE REQUISIÇÕES (*HIT RATE*) NO LOTE S-05-S.

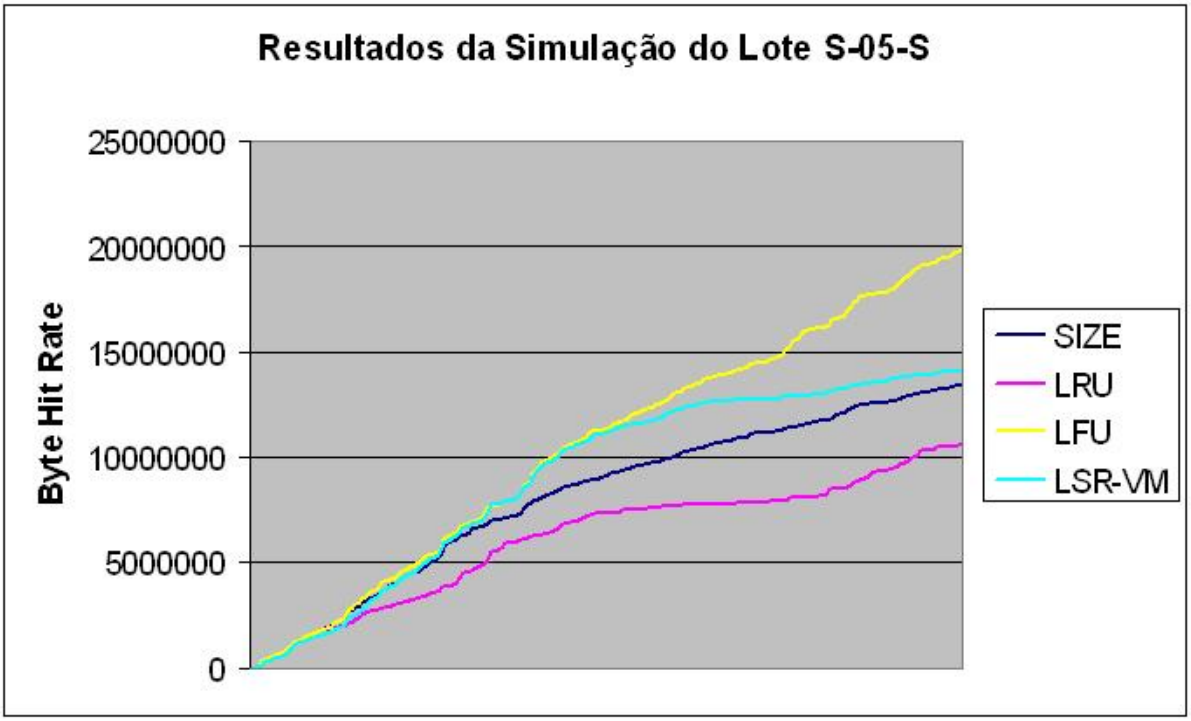


FIGURA 65 - GRÁFICO CUMULATIVO DE *BYTES* TRANSFERIDOS (*BYTE HIT RATE*) NO LOTE S-05-S.

É possível observar que em todos os lotes simulados considerando um *cache* com tamanho de 10MB, a estratégia LSR-VM apresentou desempenho superior à estratégia LRU em relação a taxa de acerto (*hit rate)*, porém apresentou desempenho inferior às estratégias LFU e SIZE.

Em relação a quantidade de bytes transferidos (*byte hit rate)*, nos lotes da SPEI a estratégia LSR-VM apresentou desempenho superior às estratégias SIZE e LRU, e desempenho inferior à estratégia LFU. Já nos lotes da PUCPR a estratégia LSR-VM apresentou desempenho superior à estratégia LRU, porém inferior às estratégias LFU e SIZE, considerando a quantidade de bytes transferidos (*byte hit rate*).

7.3 CONSIDERAÇÕES A RESPEITO DAS SIMULAÇÕES

Não foi possível realizar experimentos comparando a estratégia LSR original com a estratégia LSR-VM devido às características específicas de cada estratégia. Durante os experimentos da estratégia LSR original foram utilizadas somente 1000 URI (*Uniform Resource Identifier*) que referenciam apenas objetos do tipo texto, obtidos a partir da hierarquia existente no *site* Yahoo. Em seguida foi utilizado o *log* de um servidor *proxy Squid* como referência para se obter a distribuição de acessos. Finalmente o método de Monte Carlo foi utilizado para simular as requisições a objetos realizadas por uma comunidade de clientes. A estratégia LSR original não contempla objetos binários nem outros tipos de objeto que não estejam pré-classificados em uma hierarquia semântica. A estratégia LSR-VM tem como objetivo permitir a simulação de situações semelhantes ao comportamento real de uma comunidade de usuários, contemplando em seus experimentos objetos binários e objetos gerados dinamicamente. Durante as simulações realizadas com a estratégia LSR-VM foram utilizados *log* de um servidor *proxy Squid* para recriar seqüências

reais de requisições de duas comunidades de usuários. Não seria possível obter a pré-classificação semântica para os objetos referenciados no *log* real de um servidor *proxy Squid*, de forma a possibilitar a comparação dos resultados da simulação das estratégias de substituição LSR original e LSR-VM, utilizando a mesma base de referência.

As simulações realizadas com a estratégia LSR-VM apresentaram alto custo computacional. Devido a este fato os lotes de simulação foram limitados a aproximadamente 20.000 requisições com tamanho de *cache* de 10 MB. Utilizando estes parâmetros as simulações com a estratégia LSR-VM levaram aproximadamente 10 horas de processamento, sendo 2 a 3 horas dedicadas ao pré-processamento onde os objetos referenciados pelos *logs* são recuperados a partir de seus locais de origem e indexados através do processo de transformação de texto do modelo vetorial. Verificou-se que o tamanho do *cache* influencia de forma exponencial o tempo de simulação. Isto pôde ser observado através da análise dos resultados de simulação do lote S-04-S onde foram realizadas simulações utilizando *cache* com 5 MB, 10 MB e 20 MB. As simulações do LSR-VM com *cache* de 20 MB levaram aproximadamente 24 horas de processamento.

Considerando-se a taxa de acerto (*hit rate*) como critério de avaliação foi possível observar que a estratégia SIZE apresentou melhor desempenho em todas as sessões de simulação. Este comportamento se deve ao fato de que a maior parte dos objetos referenciados nos *logs* usados para simulação corresponde a objetos pequenos, com no máximo 1000 *bytes*. A estratégia SIZE favorece a permanência de objetos pequenos no *cache*. Em relação a quantidade de *bytes* transferidos (*byte hit rate)* e *cache* com tamanho de 10 MB, nos lotes S-04-S e S-05-S a estratégia LSR-VM apresentou desempenho superior às estratégias SIZE e LRU, e desempenho inferior à estratégia LFU.

Foi possível observar melhores desempenhos da estratégia LSR-VM com o aumento de tamanho do *cache*, considerando tanto a taxa de acerto (*hit rate*) quanto a

quantidade de *bytes* transferidos (*byte hit rate*).

A taxa de acerto (*hit rate*) e a quantidade de *bytes* transferidos (*byte hit rate*) são métricas comumente utilizadas para avaliar o desempenho de mecanismos de *cache*. A taxa de acerto (*hit rate*) influencia a avaliação de um mecanismo de *cache* em relação à latência. Já a quantidade de *bytes* transferidos (*byte hit rate*) tem influencia em relação ao tráfego gerado na Internet e a carga imposta aos servidores *web*.

As análises dos *logs* coletados na PUCPR e na SPEI apresentaram resultados diferentes. Isto se deve principalmente a decisões de arquitetura e configuração de cada uma das redes acadêmicas. Na rede da SPEI existem servidores *firewall* que filtram o tráfego não autorizado evitando que estas requisições cheguem ao servidor *proxy Squid*. Na rede acadêmica da PUCPR os servidores *proxy Squid* são utilizados como o primeiro nível para a restrição de acesso a conteúdo não autorizados. Desta forma, os *logs* dos servidores *proxy Squid* da PUCPR são "contaminados" por requisições que não podem ser atendidas e por requisições realizadas por ferramentas para compartilhamento ponto-a-ponto de arquivos em rede como, por exemplo o eMule[8].

[8] http://www.emule-project.net

8. CONCLUSÃO

Esta dissertação apresentou uma nova estratégia de substituição de objetos em *cache* na Internet denominada LSR-VM na qual o critério para remoção de objetos é a distância semântica com relação ao objeto que precisa ser inserido no *cache*. A forma de se calcular tal distância semântica baseia-se na teoria denominada *Vector Model*, normalmente empregada em mecanismos de busca na Internet, tais como Google e Yahoo. Foi apresentada uma implementação dessa estratégia e foram realizados experimentos através de simulações para validar a estratégia e compará-la com outras tradicionalmente empregadas para o mesmo fim.

Os experimentos realizados permitiram identificar diversas características do tráfego de objetos na Internet, tais como:

- Grande parte dos objetos armazenados em *cache* são objetos binários, dos quais não é possível obter semântica de forma direta. As análises realizadas nos lotes P-04-A e P-05-A apresentaram respectivamente 39,57% e 39,99% de requisições a objetos binários. As análises realizadas nos lotes S-04-A e S-05-A apresentaram respectivamente 57,97% e 56,04% de requisições a objetos binários.
- Grande parte das requisições através do *cache* corresponde a tráfego dinâmico ou a tráfego realizado através do protocolo HTTPS (*secure* HTTP), que realiza a criptografia dos dados transmitidos entre o servidor e o cliente. Este tipo de tráfego não é passível de *cache*, sendo necessário recuperar os objetos a partir dos seus locais de origem na Internet. As análises realizadas nos lotes P-04-A e P-05-A, apresentaram respectivamente 28,29% e 41,73% de requisições dinâmicas ou “seguras”. As análises realizadas nos lotes S-04-A e S-05-A apresentaram respectivamente 26,28% e 29,07% de requisições dinâmicas ou “seguras”.

- Grande parte das requisições através do *cache* corresponde a objetos pequenos, com tamanho entre 100 e 999 *bytes*. As análises realizadas nos lotes P-04-A e P-05-A apresentaram respectivamente 42% e 44% de requisições a objetos nesta faixa. As análises realizadas nos lotes S-04-A e S-05-A apresentaram respectivamente 47% e 51% de requisições a objetos nesta faixa.

A identificação dessas características foi de fundamental importância para a realização dos experimentos e para a análise dos resultados. Ficou evidente que as estratégias de substituição em *cache* baseadas em semântica devem, de alguma forma, dar tratamento a objetos binários, pois estes representam a maioria no *cache*. Dentre as estratégias apresentadas nesta dissertação, somente a estratégia LSR-VM considera os objetos binários durante as comparações semânticas.

Durante a realização dos experimentos foi descoberto que não existe um grau aceitável de confiabilidade no valor do campo *content-type* retornado por servidores *web*. O campo *content-type* tem como objeto informar ao navegador *web* o tipo de objeto que está sendo retornado como resposta a uma requisição feita a um servidor *web* [RFC2616]. Devido a este fato foi necessário realizar análises adicionais considerando as extensões das URI (*Uniform Resource Identifier*) dos objetos requisitados. No lote P-04-A, por exemplo, 17,73% dos objetos requisitados apresentaram o valor “text/html” no campo “*content-type*”, porém após uma avaliação das extensões das URI, somente 2,74% dos objetos requisitados realmente correspondem a objetos texto.

Durante os experimentos foram utilizados *logs* de servidores *proxy Squid* das redes acadêmicas de duas instituições de ensino. Os tráfegos gerados por estas comunidades de usuários podem apresentar características específicas que impedem a generalização dos resultados obtidos. Trabalhos futuros poderão estabelecer comparativos utilizando *logs* de servidores *proxy* de comunidades de usuários com

diferentes características, como por exemplo corporações comerciais e provedores de acesso à Internet. Outros *logs* podem ser obtidos gratuitamente no *site* IRCACHE[9].

Os resultados de desempenho da estratégia LSR-VM mostraram que a mesma tende a ser melhor que a estratégia LRU e pior que as estratégias LFU e SIZE considerando-se a taxa de acerto (*hit rate*) como critério de avaliação e *cache* com tamanho de 10 MB. Em relação a quantidade de *bytes* transferidos (*byte hit rate)* e *cache* com tamanho de 10 MB, nos lotes S-04-S e S-05-S a estratégia LSR-VM apresentou desempenho superior às estratégias SIZE e LRU, e desempenho inferior à estratégia LFU.

O trabalho realizado até então define e implementa a versão mais básica da estratégia LSR-VM, pois esta ainda pode ser melhorada em diversos sentidos a fim de aumentar o seu desempenho. Algumas dessas melhorias incluem:

- **Histórico de Acessos**: A versão atual da estratégia LSR-VM considera a semântica de um único objeto que está sendo inserido no *cache* durante o processo de comparação e seleção dos objetos que serão removidos. A estratégia LSR-VM pode ser modificada para permitir que a semântica de um conjunto histórico de objetos seja utilizada no processo de comparação. Para isto, o conteúdo de um novo objeto a ser inserido no *cache* deve ser mesclado com o conteúdo dos *n* últimos objetos que foram inseridos no *cache*, resultando em um único vetor “consulta”, de acordo com o modelo vetorial.
- **Particionamento do *Cache***: O conceito de particionamento de *cache* pode ser utilizado para evitar que seja necessário realizar a re-indexação de toda a coleção de objetos presentes no *cache*, toda vez que um novo objeto é inserido. A re-indexação é necessária para

[9] http://www.ircache.net

evitar que os valores dos pesos que compõem os vetores que representam objetos no modelo vetorial sejam distorcidos ao longo do tempo. Com a utilização do conceito de particionamento, o *cache* poderia ser separado em duas partições (partição principal e partição intermediária). Quando o mecanismo de *cache* for colocado em funcionamento, novos objetos serão inseridos até que ambas partições estejam cheias. Quando um novo objeto tiver de ser inserido no *cache*, a estratégia de substituição realizará a indexação dos objetos de ambas partições e iniciará o processo de remoção. Neste momento deve ser removida a quantidade de objetos necessária para esvaziar toda a partição intermediária. Desta forma, novos objetos passarão a ser inseridos na partição intermediária até que esta atinja seu limite novamente, repetindo o processo de remoção. Com isto, a indexação será realizada em intervalos de tempo maiores, aumentando o desempenho do mecanismo de *cache*.

- **Redução do Tamanho dos Vetores que Representam Objetos**: O processo de transformação de texto, de acordo com o modelo vetorial, resulta em vetores que representam os objetos em um espaço vetorial, permitindo a comparação semântica no *cache*. O tamanho destes vetores pode ser reduzido descartando-se os termos de mais alta freqüência e os termos de mais baixa freqüências. Esta redução terá impacto sobre o processo de indexação do *cache*, melhorando o desempenho geral da estratégia de substituição.

A simulação foi realizada com base em amostras de acessos referentes a períodos entre 30 a 50 minutos, totalizando aproximadamente 20.000 acessos em cada amostra. Tal limitação foi imposta a fim de viabilizar a própria simulação, realizando a simulação em um tempo razoável (cada simulação considerando *cache* com tamanho

de 10 MB demorou aproximadamente 10 horas de processamento). Com isso, os gráficos obtidos ainda não mostram estabilidade. Idealmente, seriam necessários maiores períodos de simulação para se chegar à estabilidade. Dessa forma, será preciso fazer melhorias na estratégia de simulação para se conseguir os resultados em tempo razoável. Algumas dessas melhorias incluem:

- **Utilização de *tags* "<META>" na estratégia LSR-VM**: Para minimizar o custo computacional existente na estratégia LSR-VM, poderá ser utilizada a *tag* HTML "<META>" para se obter a semântica dos objetos texto na Internet. Desta forma não será necessário contemplar todo o conteúdo do documento HTML durante o processo de transformação de texto do modelo vetorial. Isto certamente trará um ganho de desempenho para o mecanismo de *cache*.
- **Reindexações iterativas na estratégia LSR-VM**: A estratégia LSR-VM implementada como parte dos estudos que resultaram nesta dissertação realiza a re-indexação de objetos no *cache* toda vez que há necessidade de liberar espaço. Este comportamento impõe um alto custo computacional à estratégia LSR-VM. Trabalhos futuros podem ser realizados com o objetivo de tornar o processo de re-indexação iterativo. Desta forma, ao invés de executar uma re-indexação sempre que há necessidade de descartar objetos quando o *cache* está cheio, este processo poderia ser realizado em intervalos de requisições. A eficiência da estratégia LSR-VM poderia ser analisada com re-indexações a cada 1000 requisições, a cada 10.000 requisições, e assim por diante.

Uma simplificação feita nesta primeira implementação do LSR-VM foi a não consideração de que os *caches* são organizados hierarquicamente na Internet. Um

experimento mais elaborado poderá simular o uso da estratégia nos diferentes níveis de *cache* e, assim, fazer uma comparação mais efetiva do seu desempenho, inclusive experimentando diferentes estratégias de substituição trabalhando de forma híbrida, como mostrado em [BEN05].

A estratégia LSR-VM, assim como qualquer outra estratégia que se baseia na semântica dos objetos, tem o potencial de uso para fins além da substituição de objetos em *cache*. O trabalho de [BRU02] discute a possibilidade de se empregar essas estratégias para detectar assuntos (ou objetos) que estão sendo mais freqüentemente acessados por uma comunidade virtual. Este tipo de informação pode ser utilizado, por exemplo, para fazer divulgação de produtos relacionados ao interesse da comunidade. A estratégia LSR-VM pode ser utilizada em arquiteturas colaborativas de mecanismos de *cache* baseados em semântica para a pré-recuperação (*prefetch*) de objetos ou para a troca de objetos entre servidores *proxy*. Quando um determinado assunto estiver em alta em uma comunidade de usuários, o servidor *proxy* pode interagir com servidores *proxy* de outras comunidades para realizar a pré-recuperação de objetos que estejam semanticamente relacionados com os assuntos em evidência. Essas parecem ser interessantes frentes para continuação deste trabalho de pesquisa.

REFERÊNCIAS BIBLIOGRÁFICAS

[AGG99] Aggarwal, C.; Wolf, J.; Yu, P.S. *Caching on the World Wide Web*. IEEE Transactions, 11(1), 1999, p.94-107.

[ARL00] Martin Arlitt; Ludmila Cherkasova; John Dilley; Rich Friedrich; Tai Jin. *Evaluating content management techniques for Web proxy caches*. ACM SIGMETRICS Performance Evaluation Review, volume 27, número 4, 2000, ACM Press, p.3-11.

[BRU02] Brunie, L.; Pierson, J.; Coquil, D. *Semantic collaborative web caching.* Proceedings of the 3rd International Conference on Web Information Systems Engineering, 2002, IEEE.

[BEN05] Benevenuto, F.; Duarte, F.;Almeida, V. *Entendendo os Efeitos da Localidade de Referência em Hieraquias de Cache na Web*. Departamento de Ciência da Computação; Universidade Federal de Minas Gerais; XXIII Simpósio Brasileiro de Redes de Computadores, 2005, p. 423-436.

[CAL01] Calsavara, A.; Santos, R. G. *Um algoritmo de substituição de objetos em cache na Internet baseado em semântica*. Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná - PUCPR, 2001.

[CAL02] Calsavara, A.; Schuck, M. R. *Internet object caching based on semantics and access history*. Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná - PUCPR, 2002.

[CHR01] Pablo Rodriguez; Christian Spanner; Ernst W. Biersack. *Analysis of web caching architectures: hierarchical and distributed caching.* IEEE/ACM Transactions on Networking (TON), volume 9, número 4, 2001, ACM Press, p. 404-418.

[COF99] Coffman K. G.; Odlyzko A. M. *The Size and Growth Rate of the Internet.* AT&T Labs-Research, 1999.

[COH99] Edith Cohen; Haim Kaplan. *Exploiting regularities in Web traffic patterns for cache replacement.* Proceedings of the thirty-first annual ACM symposium on Theory of computing, Maio 1999, ACM Press, p.109-118.

[COM95] Comer, Douglas E.; *Internetworking With TCP/IP, Volume I - Principles, Protocols and Architecture*, Third Edition, Prentice-Hall, 1995.

[COO] Cooper, W. *Getting Beyond Boole.* School of Library and Information Studies, University of California, Berkley, CA, p.265-267.

[CUT97] Cutler, M.; Shih, Y.;Meng, W. *Using the Structures of HTML Documents to Improve Retrieval.* In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, CA, Dezembro 1997, p.241-251.

[DAV02] Brian D. Davison. *World Wide Web: Predicting web actions from HTML content*. Proceedings of the thirteenth conference on Hypertext and hypermedia, Junho 2002, ACM Press, p.159-168.

[EAS78] Caroline M. Eastman; Stephen F. Weiss. *A tree algorithm for nearest neighbor searching in document retrieval systems.* Proceedings of the 1st annual international ACM SIGIR conference on Information storage and retrieval, 1978, p.131-149.

[EGG90] L. Egghe. *A new method for information retrieval, based on the theory of relative concentration.* Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval, Brussels, Belgium, 1990, ACM Press, p.469-493.

[FED02] Tomás Feder; Rajeev Motwani; Rina Panigrahy; An Zhu. *Web caching with request reordering.* Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Janeiro 2002, ACM Press, p. 104-105.

[FOX88] E. A. Fox; G. L. Nunn; W. C. Lee. *Coefficients of combining concept classes in a collection*. Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval, Grenoble, France, 1988, ACM Press, p. 291-307.

[GAN03] Prasanna Ganesan; Hector Garcia-Molina; Jennifer Widom. *Exploiting hierarchical domain structure to compute similarity.* ACM Transactions on Information Systems (TOIS), volume 21, número 1, 2003, ACM Press, p.64-93.

[GOP02] Parikshit Gopalan; Howard Karloff; Aranyak Mehta; Milena Mihail; Nisheeth Vishnoi. *Caching with expiration times.* Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, 2002, ACM Press, p. 540-547.

[HAI01] Edith Cohen; Haim Kaplan. *Aging through cascaded caches: performance issues in the distribution of web content.* Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States, 2001, ACM Press, p. 41-53.

[HAL97] Halabi, Bassan. *Internet Routing Arquitectures*. CISCO Press, 1997.

[IRA97] Sandy Irani. *Page replacement with multi-size pages and applications to Web caching.* Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, El Paso, Texas, United States, 1997, ACM Press, p. 701-710.

[IYE02] Iyer, S.; Rowstron, A.; Druschel P. *Squirre: a decentralized peer-to-peer web cache*. Annual ACM Symposium on Principles of Distributed Computing, Session 7, Monterey, California, ACM 2002, ACM Press, p.213-222.

[JEO99] Jaeheon Jeong; Michel Dubois. *Optimal replacements in caches with two miss costs.* Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures, Junho 1999, ACM Press, p. 155-164.

[JIN99] Hongyan Jing; Evelyne Tzoukermann. *Information retrieval based on context distance and morphology.* Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, Berkeley, California, United States, 1999, ACM Press, p. 90-96.

[KAM01] Adam Meyerson; Kamesh Munagala; Serge Plotkin. *Web caching using access statistics.* Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Washington, D.C., United States, 2001, ACM Press, p. 354-363.

[KHA98] Saied Hosseini-Khayat. *Replacement algorithms for object caching*. Proceedings of the 1998 ACM symposium on Applied Computing, Atlanta, Georgia, United States, 1998, ACM Press, p. 90-97.

[KRI00] Balachander Krishnamurthy; Jia Wang. *On network-aware clustering of Web clients*. Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, 2000, ACM Press, p.97-110.

[LAW99] Lawrence, S.; Guiles, C. *Accessibility of Information on the Web*. Nature 400, 1999, p.107-109.

[LEW97] Lewis, Chris. *Cisco TCP/IP Routing Professional Reference*. McGraw-Hill Series on Computer Communications, 1997.

[LIN91] Xia Lin; Dagobert Soergel; Gary Marchionini. *A self-organizing semantic map for information retrieval.* Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval, Chicago, Illinois, United States, 1991, ACM Press, p.262-269.

[LOU02] Wenwu Lou; Hongjun Lu. *Efficient prediction of web accesses on a proxy server.* Proceedings of the eleventh international conference on Information and knowledge management, McLean, Virginia, USA, 2002, ACM Press, p. 169-176.

[MEN02] Meng, W.; Yu, C.; Liu, K. *Building Efficient and Effective Metasearch Engines.* ACM Computing Surveys, Vol. 34, No 1, Março 2002, p.48-89.

[MEY00] Friedhelm Meyer auf der Heide; Berthold Vöcking; Matthias Westermann. *Caching in networks (extended abstract)*, Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States, 2000, ACM Press, p. 430-439.

[ONE03] O'Neill, Edward T.; Lavoie, Brian F.; Bennet, Rick. *Trends in the Evolution of the Public Web*, OCLC Office of Research, United States, Abril 2003, D-Lib Magazine, Volume 9, Number 4, ISSN 1082-9873.

[PSO02] Konstantinos Psounis; Balaji Prabhakar. *Efficient randomized web-cache replacement schemes using samples from past eviction times.* IEEE/ACM Transactions on Networking (TON), volume 10, número 4, 2002, ACM Press, p. 441-455.

[RAU00] Mohammad S. Raunak; Prashant Shenoy; Pawan Goyal; Krithi Ramamritham. *Implications of proxy caching for provisioning networks and servers.* Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Santa Clara, California, United States, 2000, ACM Press, p. 66-77.

[RFC2616] Request for Comments. *Hypertext Transfer Protocol – HTTP 1.1*.; ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt; W3C/MIT, 1999.

[RFC768] Request for Comments. *User Datagram Protocol.*; ftp://ftp.rfc-editor.org/in-notes/rfc768.txt; DARPA Internet Program, 1981.

[RFC791] Request for Comments. *Internet Protocol.*; ftp://ftp.rfc-editor.org/in-notes/rfc791.txt; DARPA Internet Program, 1981.

[RFC792] Request for Comments. *Internet Control Message Protocol.*; ftp://ftp.rfc-editor.org/in-notes/rfc792.txt; DARPA Internet Program, 1981.

[RFC793] Request for Comments. *Transmission Control Protocol.*; ftp://ftp.rfc-editor.org/in-notes/rfc793.txt; DARPA Internet Program, 1981.

[RIB96] Berthier A. N. Ribeiro; Richard Muntz. *A belief network model for IR*. Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, Zurich, Switzerland, 1996, ACM Press, p.253-260.

[RIZ00] Luigi Rizzo; Lorenzo Vicisano. *Replacement policies for a proxy cache* . IEEE/ACM Transactions on Networking (TON), Abril 2000, Volume 8, Issue 2, ACM Press, p.158-170.

[ROB77] Robertson, S. *The Probability Ranking Principle in IR.* School of Library, Archive, and Information Studies, University College, London, 1977, p.280-286.

[SAL75] Salton, G.; Wong, A.;Yang, C. *A Vector Space Model for Automatic Indexing.* ACM and NSF, 1975, p. 273-280.

[SAL83] Salton, G.; McGill, M. *Introdution to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.

[SAL88] Salton, G.; Buckley, C. *Term-weighting approaches in automatic retrieval*. Information Processing & Management, p. 513-523, 1988.

[SAN01] Santos, R. G. *Substituição de Objetos em Cache na WWW Baseado na Semântica da Informação*. Dissertação de Mestrado, Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná - PUCPR, 2001.

[SAR00] Prasenjit Sarkar; John H. Hartman. *Hint-based cooperative caching.* ACM Transactions on Computer Systems (TOCS), volume 18, número 4, 2000, ACM Press, p. 387-419.

[SAR01] Paricia Correia Saraiva; Edleno Silva de Moura; Novio Ziviani; Wagner Meira; Rodrigo Fonseca; Berthier Riberio-Neto. *Rank-preserving two-level caching for scalable search engines.* Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, United States, 2001, ACM Press, p. 51-58.

[SCH02] Schmidt, G. *Servidor de Semântica dos Recursos Web: Uma Abordagem Baseada na Arquitetura RDF*. Dissertação de Mestrado, Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná - PUCPR, 2002.

[SEB02] Sebastiani, F. *Machine Learning in Automated Text Categorization*. ACM Computing Surveys, Vol. 34, No 1, Março 2002, p.1-47.

[SI02] Luo Si and Rong Jin; Jamie Callan; Paul Ogilvie. *A language modeling framework for resource selection and results merging*. Proceedings of the eleventh international conference on Information and knowledge management, McLean, Virginia, USA, 2002, ACM Press, p. 391-397.

[SON99] B. Sonah; M. R. Ito. *Performance evaluation of new adaptive object replacement techniques for VOD systems.* Proceedings of the 1999 ACM symposium on Applied computing, San Antonio, Texas, United States, 1999, ACM Press, p.437-442.

[STE00] Bin Lan; Stephane Bressan; Beng Chin Ooi; Kian-Lee Tan. *Rule-assisted prefetching in Web-server caching.* Proceedings of the ninth international conference on Information and knowledge management, McLean, Virginia, United States, 2000, ACM Press, p. 504-511.

[TAN97] Tanenbaum, Andrew S.; *Redes de Computadores*. Rio de Janeiro: Campus, 1997.

[USDC02] U.S. Department of Commerce. *A Nation Online: How Americans Are Expanding Their Use of the Internet*. http://www.ntia.doc.gov/ntiahome/dn/; Economics and Statistics Administration, 2002.

[WAN99] Wang, J. *A survey of web caching schemes for the Internet*. ACM SIGCOMM Computer Communication Review, Vol. 29, Issue 5, ACM 1999, ACM Press, p.36-46.

[WEI01] Edgar Weippl. *Visualizing content based relations in texts*. Proceedings of the 2nd Australasian conference on User interface, Queensland, Australia, 2001, IEEE Computer Society Press, p.34-41.

[WON87] S. K.M. Wong; W. Ziarko; V. V. Raghavan; P. C.N. Wong. *On modeling of information retrieval concepts in vector spaces.* ACM Transactions on Database Systems (TODS), volume 12, número 2, 1987, ACM Press, p. 299-321.

[YAN01] Qiang Yang; Haining Henry Zhang; Tianyi Li. *Mining web logs for prediction models in WWW caching and prefetching*. Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, California, 2001, ACM Press, p. 473-478.

[YAT96] Ricardo Baeza-Yates; Gonzalo Navarro. *Integrating contents and structure in text retrieval*. ACM SIGMOD Record, volume 25, número 1, 1996, ACM Press, p. 67-79.

[YAT99] Yates, R.; Neto, B. *Modern Information Retrieval*. Addison Wesley – ACM Press, 1999, p.19-34.

[YIN97] Ying Shi; Edward Watson; Ye-sho Chen *Model-driven simulation of World-Wide-Web cache policies.* Proceedings of the 29th conference on Winter simulation, Dezembro 1997.

[YUE02] Cheng-Yue Chang; Ming-Syan Chen. *A new cache replacement algorithm for the integration of web caching and prefectching.* Proceedings of the eleventh international conference on Information and knowledge management, McLean, Virginia, USA, 2002, ACM Press, p. 632-634.

[ZAÏ02] Osmar R. Zaïane; Maria-Luiza Antonie. *Classifying text documents by associating terms with text categories*. Proceedings of the thirteenth Australasian conference on Database technologies, Melbourne, Victoria, Australia, 2002, Australian Computer Society, Inc., p.215-222.

[ZHA03] Zhaoming Zhu; Yonggen Mao; Weisong Shi. *CWorkload Characterization of Uncacheble HTTP Traffic*. Technical Report, Department of Computer Science, Wayne State University, Australia, 2002, Inc., MIST-TR-03-003.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo