Desempenho de Sistemas de Distribuição de Documentos via Web

Roberto Ferreira Brandão

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Desempenho de Sistemas de Distribuição de Documentos via Web

Roberto Ferreira Brandão

Junho de 2003

Banca Examinadora:

- Ricardo de Oliveira Anido (Orientador)
  Instituto de Computação - UNICAMP
- Profa. Dra. Cristina Duarte Murta
  Departamento de Informática - UFPR
- Prof. Dr. Marcos José Santana
  Instituto de Ciências Matemáticas e de Computação - USP
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação - UNICAMP
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP
- Prof. Dr. Maurício Ferreira Magalhães - Suplente
  Faculdade de Engenharia Elétrica e Computação - UNICAMP
- Prof. Dr. Edmundo Roberto Mauro Madeira - Suplente
  Instituto de Computação - UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Brandão, Roberto Ferreira

B733d Desempenho de sistemas de distribuição de documentos via Web / Roberto Ferreira Brandão -- Campinas, [S.P. :s.n.], 2003.

Orientador : Ricardo de Oliveira Anido

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Internet (Redes de computação). 2. World Wide Web – Servidores. 3. Simulação por computador. 4. Programação paralela (Computação). I. Anido, Ricardo de Oliveira. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 13 de junho de 2003, pela Banca Examinadora composta pelos Professores Doutores:

Profa. Dra. Cristina Duarte Murta
UFPR

Prof. Dr. Marcos José Santana
ICMC - USP

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP

Prof. Dr. Célio Cardoso Guimarães
IC - UNICAMP

Prof. Dr. Ricardo de Oliveira Anido
IC - UNICAMP

Desempenho de Sistemas de Distribuição de Documentos via Web

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Roberto Ferreira Brandão e aprovada pela Banca Examinadora.

Campinas, julho de 2003.

Ricardo de Oliveira Anido (Orientador)

Tese apresentada ao Intituto de Computação da UNICAMP como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Resumo

Este trabalho faz contribuições ao avanço do conhecimento de três diferentes áreas dos sistemas de distribuição de documentos via Web: gerenciamento de espaço em caches para Web, redes hierárquicas de caches cooperativos e servidores Web baseados em clusters de processadores.

Primeiramente, é apresentado um estudo sobre a utilização de caches cooperativos. A cooperatividade permite um melhor aproveitamento dos recursos dos meios de transmissão, possibilitando um menor tempo de espera aos usuários dos serviços da Web.

Na área de gerenciamento de espaço em caches para Web foi desenvolvido o GDE (Gerenciador Dinâmico de Espaço), que além de apresentar desempenho compatível com as melhores estratégias de gerenciamento de espaço em caches usadas atualmente, possui a vantagem de poder variar automaticamente o direcionamento do trabalho do cache, podendo fazer com que o cache varie dinamicamente a QoS (*Quality of Service*) oferecida aos usuários do sistema.

Este trabalho apresenta também um simulador paralelo de redes de caches distribuídos, hierárquicos e cooperativos, que é capaz de usar o poder de processamento de computadores paralelos para simular situações reais de uso de caches para Web, permitindo assim avaliar paralelamente um grande número de possibilidades para a construção de redes de caches para Web.

Para aumentar o desempenho dos caches formadores de redes hierárquicas, é proposta a utilização da estratégia de Passagem de Recomendação, que permite conseguir altas taxas de desempenho sem se preocupar com o problema da perda de referência da popularidade inicial dos arquivos.

O estudo de servidores Web baseados em clusters de processadores foi direcionado principalmente a computadores com a arquitetura Beowulf. Neste trabalho são apresentados resultados da avaliação da possibilidade de usar computadores com essa arquitetura como servidores Web de alto desempenho, escaláveis e tolerantes a falhas. Para isso, é proposto um modelo do servidor e apresentados testes realizados com um protótipo implementado.

Abstract

This document describes a research work that makes contributions towards a better understanding of three different areas related to document distribution via Internet: Web cache space management, hierarchical and cooperative caches and clustered Web servers.

First, it is presented a study on Web cache cooperation. Cooperation allows better resource utilization, making possible a lower response time to the users.

The startegy named GDE (Gerenciador Dinâmico de Espaço – Dynamic Space Manager) allows to Web caches both to present good performance and ability of providing different QoS (Quality of Service) to the users.

This document also presents a parallel simulator for distributed, hierarchical and cooperative caches networks. That simulator is able to use the high processing power provided by parallel computers to simulate real situations of caches network work.

To increase the performance of the caches within a Web cache network, this documents propose a strategy named Passagem de Recomendação (Recommendation Passing), that allows getting high performance from the caches regardless the replacement policy they use, avoiding problems caused by the lost of initial popularity reference.

This document also describes a study regarding the possibility of using a Beowulf cluster as a high performance, high availability and fault tolerant Web server. That study was accomplished using a model whose characteristics were used to build a prototype. Some evaluation results of that prototype are also presented.

Agradecimentos

Agradeço ao Instituto de Computação e à Unicamp por proverem a estrutura necessária para a realização do trabalho.

Ao professor Ricardo Anido, meu orientador, pela orientação, aconselhamento e amizade recebidos durante todo o processo de desenvolvimento do trabalho. Aos amigos, colegas e professores do IC, pelas tantas horas de tão frutíferas conversas e tão precioso conhecimento recebido.

À minha família, cujo apoio irrestrito em todos os momentos deram-me forças e tranqüilidade para concluir essa etapa, sem o qual eu nem poderia pensar em ir tão longe. Meus mais sinceros agradecimentos aos meus pais e a meus tantos irmãos: Tuca, Tina, Renata, Teco, Paula, Júlia, Flávio e Lúcia.

À minha querida Tatiana, por todo o companheirismo, paciência, cumplicidade e amizade em todos os momentos.

Aos amigos de república que ajudaram a fazer essa jornada muito mais agradável. Ao Leitão, David, Issao, Marcelo, Ronaldo, Órion, Lumbriga, Alphace, Stressado, Chico Bento, Fred, Snarf, Morando, Vosh, Pacote e Fernando.

Agradeço ainda ao CNPq pelo apoio financeiro durante todo o curso, à FAEP e PRONEX pela ajuda financeira nas viagens para a apresentação de trabalhos e também ao CENAPAD/SP pelos cursos de programação paralela.

À minha família.

Sumário

Lista de Figuras

Lista de Tabelas

Listagens

Capítulo 1

Introdução

1.1. Principais conceitos

A WWW (World Wide Web ou simplesmente Web) é um sistema de distribuição de documentos baseada no modelo Cliente/Servidor. Atualmente, a WWW tem experimentado um crescimento exponencial e esse crescente uso dos serviços oferecidos pela Web resulta numa maior carga sobre a rede devido ao aumento do número de requisições feitas a servidores remotos. Além disso, com o crescimento da popularidade do uso da Web, os servidores considerados mais "populares" experimentam uma grande carga ocasionada pela grande quantidade de requisições. A sobrecarga na rede e nos servidores faz com que o usuário final dos serviços oferecidos pela Web, ao requisitar um documento, experimente um tempo de espera muito maior que o tempo experimentado quando a carga na rede está baixa [Malpani95].

O cache de documentos através da Web é uma forma de diminuir os problemas causados pela sobrecarga nas redes e nos servidores. A instalação de servidores de caches para Web é um resultado de muitos esforços que têm sido feitos no intuito de desenvolver estratégias para economizar recursos de rede e proporcionar ao usuário final o menor tempo de espera possível [Kim98].

Um cache para Web consiste em um sistema de software que tem acesso a espaço para armazenamento de arquivos, instalado de forma a receber requisições feitas por

clientes, repassar essas requisições aos servidores e armazenar cópias dos arquivos para poder satisfazer requisições futuras sem a necessidade de buscar o arquivo novamente no servidor de origem do arquivo.

Os caches podem diminuir drasticamente a carga na rede. Por exemplo, segundo [Willians96], em casos especiais, é possível que um cache estrategicamente colocado consiga diminuir em mais de 80% a carga na rede. Entretanto, caches possuem limitações: em primeiro lugar, apesar de existirem trabalhos que estudem a possibilidade de armazenamento de documentos dinâmicos [Cao99], caches trabalham apenas com documentos estáticos e que, preferencialmente, sejam modificados com pouca freqüência em seus servidores [Abrams95]. Em segundo lugar, garantir a consistência entre os documentos no cache e nos servidores pode causar um aumento na carga das redes devido às mensagens trocadas entre os servidores e os caches [Dingle96].

Quando um cliente requisita um documento a um servidor de cache para Web, esse documento pode ou não estar no cache. Quando um documento requisitado foi encontrado em um cache, diz-se que houve um *Hit*. Ao contrário, quando o documento não está no cache, diz-se que ocorreu um *Miss*.

Para medir a eficiência de um sistema de cache para Web, normalmente são utilizados os métodos de taxa de acerto (*hit ratio*) e taxa de acerto por *byte* (*byte hit ratio*). No método de taxa de acerto é medido o percentual de requisições que são satisfeitas por documentos do cache enquanto que no método do taxa de acerto por byte é medido o percentual de *bytes* que são enviados ao cliente diretamente do cache, sem ser necessário fazer uma requisição ao servidor de origem.

Além do desempenho medido em *hit ratio* e byte *hit ratio*, pode-se usar uma comparação entre esses valores e o valor do desempenho máximo possível que pode ser alcançado pelo cache usando um espaço de armazenamento infinito. Um cache pode ser considerado de tamanho infinito com relação a um *trace* específico quando sua capacidade de armazenamento for igual ou superior à soma dos tamanhos de todos os diferentes documentos que devem, em algum tempo, ser armazenados. Nesse caso, um aumento do tamanho do cache não teria qualquer influência no desempenho do mesmo. Estudos com caches de tamanho infinito são interessantes para mostrar o quão distante do desempenho máximo possível está o desempenho de um cache.

Vários parâmetros estão relacionados ao desempenho de caches para Web [Melve97]. Dentre os principais, pode-se citar o tamanho do espaço de armazenamento, a capacidade de processamento do processador no servidor de cache e a política de substituição utilizada. Políticas de substituição são algoritmos executados para escolher um documento a ser removido do cache para armazenar um novo documento, quando o espaço livre é insuficiente para armazenar o novo documento [Kim98], [Willians96]. Nesse caso, são removidos os documentos considerados mais dispensáveis até que o espaço seja suficiente para armazenar o novo documento. Uma lista das principais políticas de substituição desenvolvidas até o momento é apresentada na seção 2.1.

Caches para Web podem ser utilizados em vários níveis. A divisão em níveis depende da proximidade dos caches com os usuários dos serviços. Essa proximidade com os usuários é normalmente medida através do número de *hops* desde o cliente até o servidor.

No primeiro nível são instalados os caches locais ou caches de *browser*. Esses caches têm um tamanho que varia normalmente de alguns megabytes a centenas de megabytes e são gerenciados diretamente pelos *browsers*. O desempenho desse tipo de cache é bastante aumentado se o computador onde ele está instalado é usado por pessoas que têm hábitos de acesso parecido, ou seja, que têm uma maior probabilidade de acessar as mesmas páginas.

No último nível, o mais elevado em uma hierarquia de caches, encontram-se os caches de servidor. Esses caches, instalados próximos a um servidor Web, têm como objetivo diminuir a carga de acessos aos servidores. Apresentam um bom desempenho principalmente se os arquivos requisitados ao servidor forem passíveis de serem armazenados em caches. Seu tamanho e capacidade de processamento dependem principalmente do número de requisições recebidas.

Os caches instalados entre o primeiro e o último nível são chamados caches de rede ou caches *proxy*. Esses caches podem ter os mais variados tamanhos e configurações e o seu principal objetivo é diminuir o tráfego na rede a partir da sua posição até os servidores. A sua utilização é altamente recomendada, principalmente para economizar tráfego em uma conexão de baixa capacidade.

A instalação de caches para Web sucessivos em um fluxo de transmissão de dados gera uma rede hierárquica de caches. Em sistemas de cache hierárquico, os servidores resolvem requisições não satisfeitas utilizando-se de caches de nível hierárquico superior. Para distribuir a carga em um servidor cache, podem ser instalados servidores cooperativos com o mesmo nível hierárquico.

Quando recebe uma requisição, se o servidor contiver o documento requerido em seu cache ele o envia para o cliente. Caso o documento não esteja em seu cache, o servidor questiona aos demais servidores de nível igual ao seu. Se algum outro servidor contiver o documento, ele é requisitado e depois enviado ao cliente. Caso contrário, a requisição é enviada ao cache de nível hierárquico superior. Esse processo é repetido até que o documento seja encontrado ou seja enviada uma requisição ao servidor de origem do arquivo.

Um exemplo de rede hierárquica é apresentado na figura 1.1.1. Pode-se notar que nesse exemplo existe cooperação entre os caches de nível N. Esse é ainda um exemplo de uma rede hierárquica de caches, pois as requisições não satisfeitas pelo cache de nível 1 são enviadas a caches de níveis superiores.

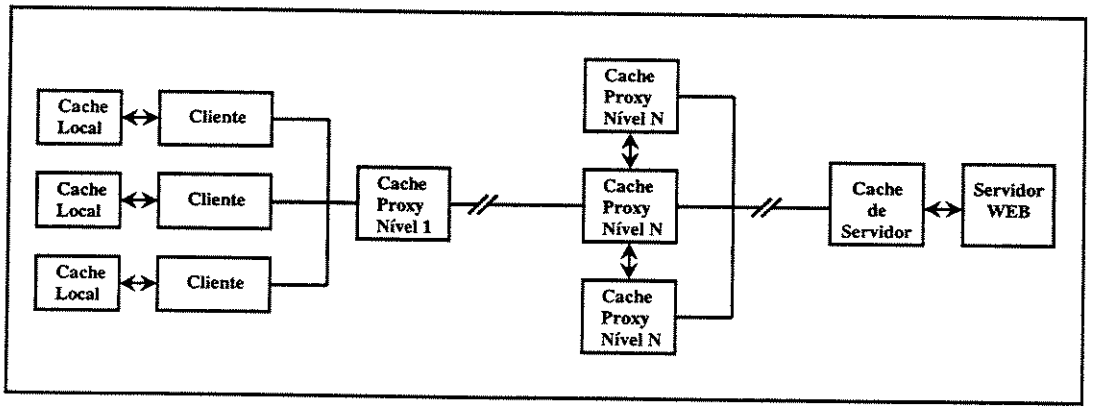


Fig. 1.1.1 – Exemplo de rede hierárquica

A utilização de vários servidores de cache cooperativos torna-se cada vez mais importante devido a sua maior escalabilidade e tolerância a falhas visto que, dessa forma, não existe mais apenas um ponto de falha no sistema de cache. São estudadas então maneiras de aumentar a probabilidade de que um documento requerido já esteja armazenado em um dos caches que compõe a rede de caches cooperativos, diminuindo

assim o tempo de espera experimentado pelo usuário ao requisitar um documento e, além disso, estudos são realizados de forma a determinar quais parâmetros interferem no sistema de cache distribuído.

Em um sistema de caches distribuídos, processadores contendo espaços para cache individuais agem cooperativamente de forma a aumentar o desempenho total do sistema, sendo que esse processo de cooperação é transparente aos usuários do sistema. Mensagens são trocadas entre os processadores que formam o cache proxy distribuído de forma a possibilitar a cooperação entre os mesmos. Em um sistema de caches distribuídos o principal problema está na comunicação, onde deve ser evitada a troca desnecessária de mensagens e mesmo assim manter os dados consistentes [Kurcewicz98]. Uma grande vantagem no uso de caches distribuídos é a tolerância a falhas, visto que sistemas de cache não podem apresentar apenas um ponto de falha, pois caso ocorra uma falha nesse ponto único haverá um comprometimento do funcionamento de todo o sistema.

Malhas de caches devem ser utilizadas para evitar alto tráfego em *links* de alto custo e alto tempo de resposta. É possível definir uma hierarquia dentro da malha inicialmente desordenada de servidores de cache para Web, fazendo assim que as malhas apresentem as mesmas funcionalidades observadas em servidores dispostos hierarquicamente, além da vantagem de oferecer caminhos alternativos aos dados, o que permite uma opção a um link sobrecarregado e uma maior tolerância a falhas.

Uma estratégia interessante relacionada a caches para Web é o *prefetching* de arquivos. Nesse sistema, o cache utiliza o tempo ocioso dos meios de transmissão para buscar documentos que tenham grande chance de serem acessados no futuro. Por exemplo, pode-se usar o tempo que um usuário leva para ler o conteúdo de uma página para buscar páginas com alta probabilidade de serem acessadas num futuro próximo.

1.2. Motivação e objetivos deste trabalho

Com o aumento da popularidade do uso dos serviços oferecidos pela Internet, tanto as redes como os servidores de documentos têm sofrido com a alta taxa de requisições.

O impacto nas redes de transmissão de dados normalmente faz com que o usuário final experimente um grande tempo de espera, principalmente em horas de grande utilização do sistema. O gráfico da figura 1.2.1 apresenta uma aproximação do perfil da utilização da Internet no Brasil, segundo [FAPESP02].

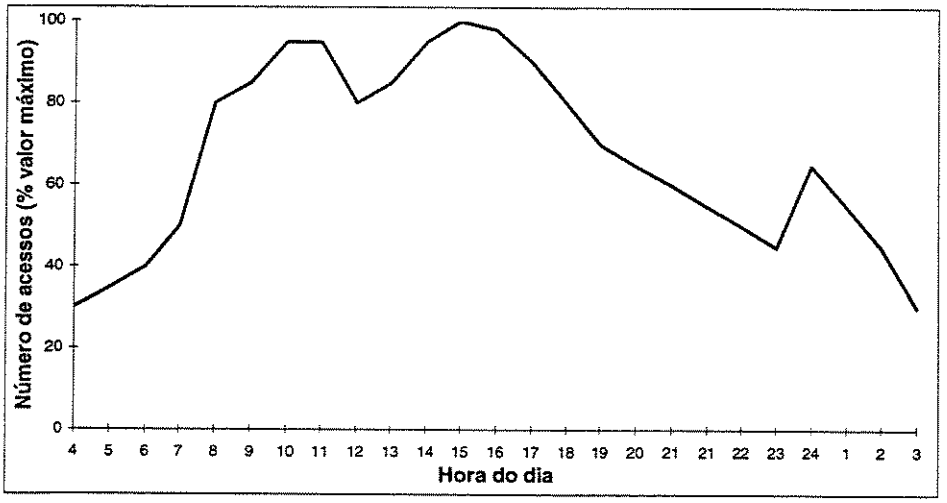


Fig. 1.2.1 – Perfil de acesso à Internet no Brasil

Na figura 1.2.1, o pico de utilização por volta da meia noite ocorre porque muitos usuários ainda utilizam conexões *dial-up* discadas, usando a rede telefônica. Muitas das companhias telefônicas que operam no Brasil cobram o valor de apenas um pulso telefônico para ligações feitas de meia noite às seis da manhã, não importando a duração das ligações. Isso faz com que muitos usuários acessem a Internet depois da meia noite.

Os maiores valores dos números de acessos ocorrem dentro do chamado "horário comercial", quando funcionários de empresas utilizam a Internet no trabalho.

A variação do número de acessos durante o dia é um dos fatores que deve ser considerado quando do projeto de redes de transmissão de dados e servidores de documentos Web, pois caso a rede seja projetada para suportar apenas o tráfego médio, a qualidade do serviço oferecido aos usuários nos horários de pico sofrerá uma grande degradação.

Apesar das capacidades das redes e servidores terem aumentado bastante nos últimos anos, principalmente devido à evolução das tecnologias envolvidas e ao grande investimento feito nesse setor nos últimos anos, a capacidade atual da Internet ainda está

muito aquém das necessidades dos usuários. As redes ainda não são capazes de oferecer o *throughput* necessário para oferecer a todos os usuários dos serviços uma QoS (*Quality of Service*) satisfatória. O problema ainda é pior quando considerados os desejos dos usuários em requisitar mídias tais como arquivos de áudio e vídeo, que normalmente têm tamanhos maiores e, por isso, exigem maior desempenho das redes.

Para diminuir o impacto das capacidades dos sistemas de distribuição de documentos, que são ainda baixas quando comparadas à demanda potencial desse tipo de serviço, faz-se necessário o desenvolvimento de estratégias que possibilitem aumentar ao máximo possível o número de usuários que podem utilizar-se dos serviços ao mesmo tempo, bem como melhorar a capacidade de prover melhor qualidade nos serviços oferecidos.

A busca pelo aprimoramento de estratégias existentes e o desenvolvimento de novas foi a principal motivação para a realização deste trabalho de pesquisa.

1.3. Abordagens utilizadas

Esta seção descreve sucintamente os principais problemas encontrados no decorrer do trabalho de pesquisa e as abordagens utilizadas para encontrar soluções para os mesmos.

1.3.1. Configurações de redes de caches

Para descobrir qual a melhor configuração de parâmetros relativos a um determinado sistema de caches para Web, devem-se testar várias combinações de valores para os parâmetros de forma a determinar qual dessas combinações possibilita o melhor desempenho. No caso de malhas de caches cooperativos, devem-se considerar também a distribuição hierárquica dos servidores e a capacidade de transmissão das redes que os interligam. Devido à quantidade de recursos envolvidos nesse tipo de pesquisa, fica difícil conseguir uma malha real de caches para fazer experiências. A simulação se torna então a opção mais viável para pesquisas com caches para Web. Portanto, foi desenvolvido neste trabalho um simulador totalmente configurável de malhas hierárquicas de caches para Web.

Um dos principais problemas encontrados na simulação de malhas de caches para Web é a manutenção de coerência entre as operações do simulador e as operações de uma malha real de caches. Assim, torna-se necessário utilizar históricos de requisições reais, feitas a caches reais, no simulador. Esses históricos de requisições são chamados *arquivos de log* ou *traces* e podem ser conseguidos em vários *sites* na Internet. O simulador desenvolvido utiliza *traces* obtidos a partir da operação de caches reais, permitindo verificar o impacto da variação de parâmetros no desempenho de toda a malha de caches.

1.3.2. QoS em caches para Web

As características da demanda de QoS dos serviços prestados pela Web não são estáticas. No caso de caches para Web, elas dependem de diversos fatores tais como o número de usuários, a capacidade de transmissão das redes, diferenças entre perfis de usuários, a hora e dia da semana, a época do ano e a variação da popularidade dos arquivos requisitados. Assim, o desempenho de uma política escolhida inicialmente pode diminuir drasticamente devido à incapacidade da política em se adaptar a novas características exigidas para um bom desempenho. Por isso, é necessário que a estratégia de gerenciamento do espaço para cache consiga se adaptar às variações de características da demanda por documentos.

Com esse objetivo, foi apresentado o Gerenciador Dinâmico de Espaço (GDE). Essa estratégia pode alternar dinamicamente os parâmetros utilizados para estimar a utilidade futura dos arquivos armazenados. Assim, ela permite variar dinamicamente o perfil dos arquivos com maior preferência de serem mantidos no cache. Dessa forma, é possível direcionar o trabalho do cache para priorizar a manutenção de arquivos com características específicas desejadas, podendo assim variar a QoS oferecida aos usuários do sistema.

1.3.3. Perda da referência de popularidade

Apesar de muitos trabalhos tentarem aumentar o desempenho individual dos caches em uma rede, muitas vezes uma grande melhora do desempenho de um cache

pode não representar uma melhora significativa no desempenho geral da rede de caches devido à perda de referência da distribuição de popularidade inicial.

A perda de referência de popularidade ocorre quando os caches de níveis mais elevados não conseguem obter um bom desempenho devido a não mais poderem basear-se apenas no fluxo de requisições que chega a eles. Isso ocorre porque as características de popularidade desse fluxo foram alteradas quando da sua passagem por caches de níveis hierárquicos inferiores. Isso faz com que uma melhora de desempenho em *hit ratio* nos caches de nível elevado não signifique que eles estejam dando prioridade a arquivos mais populares entre os clientes e sim aos arquivos mais populares no fluxo de requisições que estão recebendo.

Neste trabalho foi realizado um estudo do impacto da instalação de uma rede hierárquica de caches para Web no fluxo de requisições. Foi então proposto o mecanismo de passagem de recomendação como forma de evitar que caches de níveis hierárquicos altos percam a referência da popularidade inicial, aumentando seu desempenho.

1.3.4. Servidores Web paralelos

Com o aumento da popularidade dos serviços oferecidos pela Internet, muitos estudos têm sido realizados com o objetivo de aumentar o desempenho dos sistemas de distribuição de documentos. Além da elaboração de estratégias de posicionamento e aumento de desempenho de caches para Web, a capacidade do servidor de documentos é também um elemento fundamental no desempenho do sistema de distribuição de documentos.

Apesar de um alto percentual de documentos requisitados a um servidor Web ainda consistir de documentos estáticos, tornam-se cada vez mais comuns as páginas geradas no momento do processamento das requisições, contendo informações muitas vezes tiradas de bancos de dados no momento da geração da página. Essa nova funcionalidade exige dos servidores Web uma capacidade de processamento cada vez maior, fazendo-se necessária a distribuição de processamento de requisições entre servidores cooperativos.

Para o processamento paralelo de requisições feitas a endereços Internet muito populares, podem ser utilizados clusters de processadores. Os nós formadores do cluster dividem a carga de requisições entre si, aumentando a taxa de requisições respondidas pelo sistema. Uma arquitetura de computador baseada em cluster que vem destacando-se entre as demais por seu baixo preço e pela ausência do limite técnico para a quantidade máxima de PC's que podem ser conectados é a arquitetura Beowulf [Beowulf02].

Entretanto, a utilização de um Beowulf como servidor Web apresenta problemas de distribuição e balanceamento da carga de requisições entre os nós, bem como a distribuição das tarefas a serem executadas por cada nó e dos documentos as serem armazenados.

Neste trabalho foram realizados experimentos com o objetivo de analisar a viabilidade da utilização de computadores baseados na arquitetura Beowulf e propor técnicas de solução dos problemas mais comuns encontrados quando da tentativa de usar computadores dessa arquitetura como servidores Web. Para isso, foi proposto um modelo que descreve o servidor, sendo também apresentados resultados de experimentos realizados com um protótipo desenvolvido com base no modelo proposto.

1.4. Organização da Tese

O capítulo 2 apresenta uma relação dos principais trabalhos relativos aos temas abordados por este trabalho de pesquisa. O capítulo 3 apresenta estudos sobre redes de caches distribuídos e cooperativos, bem como sobre a simulação dessas redes.

O Gerenciador Dinâmico de Espaço (GDE) é apresentado no capítulo 4 como forma de permitir a variação da QoS oferecida a usuários de sistemas de caches.

No capítulo 5 é apresentado e discutido o mecanismo de Passagem de Recomendação e um estudo sobre a viabilidade de usar um computador com arquitetura Beowulf como servidor Web é apresentado no capítulo 6.

As conclusões gerais sobre este trabalho de pesquisa são apresentadas no capítulo 7, sendo também feitas algumas sugestões para trabalhos futuros.

O apêndice A faz uma breve descrição das principais ferramentas de software usadas para gerar os resultados apresentados neste trabalho.

Capítulo 2

Trabalhos anteriores

2.1. Gerenciamento de espaço

Estudos mostraram que tanto o *hit ratio* quanto o *byte hit ratio* de um cache para Web crescem logaritmicamente em função do tamanho do cache [Jin00]. Isso implica que um algoritmo que aumenta o *hit ratio* e/ou o *byte hit ratio* em alguns pontos pode ser substituído pelo aumento do espaço em disco. Porém, o crescimento da Web é muito maior que o da tecnologia de fabricação de memória. Isso implica que a melhor maneira de aproveitar um cache para Web é através de técnicas eficientes de gerenciamento de espaço [Murta98].

Os principais artigos que estudam o desempenho de políticas de substituição existentes e propõe novas são [Willians96], [Lorenzetti96], [Kim98], [Abrams95], [Povey97], [Rabinovich98] e [Vakali99].

Uma breve descrição das principais políticas de substituição estudadas é apresentada a seguir:

- FIFO (First In First Out) - Remove-se o documento mais antigo.
- LRU (Least Recently Used) - Remove-se o menos recentemente acessado.
- LRU-MIN [Abrams95] - Similar ao LRU mas com considerações sobre o tamanho do documento, onde são removidos preferencialmente os documentos maiores.

- LRU-Threshold [Abrams95] - Similar à LRU, com a diferença de que documentos maiores que um limite nunca são armazenados no cache.
- LFU - (Least Frequently Used) - Remove-se o documento usado com menor freqüência.
- SIZE - Remove-se o maior documento.
- Log(SIZE) + LRU - Remove o elemento menos recentemente usado dentre todos que possuem os maiores log(tamanho).
- Hyper-G - Refinamento da política LFU, com considerações sobre o último acesso e tamanho.
- Pitkow/Recker - Usa LRU a não ser que todos os documentos tenham menos que 24 horas. Nesse caso, é aplicada a política SIZE.
- Lowest Latency First - Elimina o documento que apresenta a menor latência ao servidor de origem.
- Hybrid - A ordenação de prioridades é feita por um valor calculado como função do tempo para se conectar ao servidor, a capacidade da rede que liga o cache ao servidor, o número de acessos ao documento e o seu tamanho.
- Lowest Relative Value (LRV) - Usa o custo particular de cada documento e o seu tamanho para definir qual documento será removido.
- LFU with Dynamic Aging (LFU-DA) [Cao97] - Utiliza a freqüência de acesso e o custo de cada documento, associado a uma técnica de envelhecimento de documentos para organizar o cache.
- Greedy-Dual – Adaptação de um algoritmo de gerenciamento de memória virtual para caches para Web. Os documentos têm o mesmo tamanho. A utilidade dos documentos para o cache depende do custo para trazê-los e do tempo desde o último acesso.
- GreedyDual-Size - (GDS) [Cao97] – Aprimoramento da política Greedy-Dual, com considerações aos tamanhos dos documentos. A importância dos documentos pode ser proporcional ao número de pacotes IP necessários para transmitir o documento. Nesse caso, a GDS é chamada GDS(packets). Se for considerado que o custo de transmissão é igual para todos os documentos, a GDS é chamada GDS(1) e é equivalente à Greedy-Dual.

- GreedyDual-Size with Frequency (GDSF) [Cao97] - Similar à GreedyDual-Size, porém utiliza também a freqüência de acesso a cada documento para decidir qual será removido.
- GD* [Jin00] – Aprimoramento da GDS, com considerações à popularidade esperada dos documentos. Assim como a GDS, pode-se definir GD*(1) e GD*(packets), dependendo da importância dos tamanhos dos documentos.

A política SIZE e suas variantes proporcionam um alto desempenho em *Hit Ratio* devido à característica apresentada pela maioria dos documentos transmitidos pela Web de ter um tamanho reduzido, em razão do menor tempo necessário para o envio dos mesmos. Porém, o fato de serem escolhidos como mais favoráveis para descarte os documentos maiores faz com que a política SIZE tenha um desempenho em *Byte Hit Ratio* menor que as demais políticas, principalmente para caches pequenos, o que se explica pela poluição do cache por arquivos pequenos que nunca são reacessados e dificilmente são descartados.

Em [Abrams95] são propostas variantes do algoritmo LRU, as políticas LRU-MIN e LRU-THOLD. No algoritmo LRU-MIN são descartados os maiores documentos daqueles menos recentemente acessados no cache. O uso do algoritmo LRU-THOLD faz com que seja estabelecido um tamanho máximo, sendo que são armazenados no cache apenas documentos com tamanhos menores que o máximo. Verificou-se que esse tamanho máximo de documentos é função da quantidade de espaço disponível no cache: quanto mais espaço disponível houver no cache, maior pode ser o tamanho máximo dos documentos.

Em [Murta98] e [Murta99] é proposta a divisão da cache em partições que armazenam classes de documentos, tentando otimizar tanto o *hit ratio* quanto *o byte hit ratio*. O particionamento do cache em seções dedicadas a armazenar documentos cujo tamanho está dentro de uma faixa tem a vantagem de oferecer independência entre os fatores que implicam na variação da performance do sistema, além de permitir implementar tratamento diferenciado para as diversas classes de documentos.

2.2. QoS em caches para Web

Estudos sobre a Qos (*Quality of Service*) dos caches para Web são importantes para determinar os níveis de qualidade de serviço oferecidos pelos caches aos usuários. Uma proposta de implementar a capacidade de oferecer diferentes QoS em um servidor Web é apresentada em [Almeida98]. Em [Afonso98] é apresentada uma maneira de medir a QoS de um cache para Web através do tempo de resposta do mesmo. Em [Kelly99] é proposta uma simples generalização da política LFU, que permite ao servidor Web atribuir valores de importância para a manutenção de arquivos nos caches.

A estratégia introduzida em [Kelly99a] utiliza a predição e estimação de utilidade futura de arquivos para calcular a importância de manter arquivos no cache. O modelo sugerido por [Chan99] adota a perspectiva de que os arquivos mais valiosos para os clientes devem ser mantidos no cache. Nesse modelo, os servidores requisitam espaço no cache para armazenar os arquivos a sua escolha.

2.3. Caches distribuídos e cooperativos

Na arquitetura de cache distribuído proposta em [Kurcewicz98], existem três tipos de servidores: o frontend, o backend e o manager. O frontend é escolhido pelo cliente e a carga entre os frontends é distribuída através de um roteador TCP. O manager recebe as requisições dos frontends e verifica se algum backend possui o documento desejado. Se o documento estiver em algum backend, o manager o instrui para enviá-lo ao frontend. Caso contrário, o manager instrui ao frontend para que ele busque o documento no servidor de origem e armazene uma cópia no backend.

No trabalho descrito em [Malpani95], o objetivo é fazer com que múltiplos servidores cooperem entre si, compartilhando seus caches individuais para criar um grande cache distribuído. O trabalho considera que cada servidor pode responder a requisições de qualquer cliente e que é possível distribuir a carga pelos servidores. É usado um sistema de envio de mensagens através do *Multicasting*, para diminuir o impacto causado na rede. É ainda proposto um protocolo para a busca de arquivos entre os caches cooperativos.

Em [Melve97] é feita uma descrição dos principais conceitos e técnicas relativas à arquitetura de um sistema de *Web Caching* composto de caches cooperativos enquanto que em [Chankhunthod96] é detalhado o impacto da instalação de caches em redes de distribuição de documentos.

A experiência descrita em [Glassman94] mostrou que sistemas de caches provêm melhora no desempenho a baixo custo. Diminui enormemente a latência no acesso a páginas no cache e praticamente não adiciona latência significante nas demais requisições. No caso de *sites* ligados à Internet por redes de baixa capacidade, as vantagens do cache são ainda mais significativas.

Em [Dahlin94], são examinadas quatro estratégias para cooperação entre caches. A primeira é a cooperação direta entre clientes, que permite que um cliente ativo utilize uma parte disponível da memória de outro cliente. A segunda forma trata toda a memória do sistema como um recurso global, disponível para todos os clientes. A terceira forma é semelhante à segunda, apresentando ainda um gerenciamento centralizado dos recursos. A última forma atribui determinada quantidade de recursos a cada cliente, de acordo com sua atividade.

Em [Rabinovich98] é apresentada uma proposta para a diminuição do impacto na rede da instalação de um sistema de caches. O método proposto neste trabalho faz com que ao invés de se pagar o *overhead* da manutenção de informações globais sobre documentos em um cache distribuído potencialmente grande, o cache apenas guarde uma lista de documentos em caches próximos. Esse método também ajuda a minimizar a sobrecarga na rede causada pela propagação de atualizações nos caches.

O WebWave consiste num conjunto de servidores de cache cooperando para servir a requisições de clientes. O objetivo é conseguir um sistema de carga balanceado, minimizando o tempo desocupado dos servidores e maximizando o *throughput* do sistema. Cada servidor tem a habilidade de armazenar e descartar documentos baseados na sua carga de trabalho, na dos seus vizinhos e na popularidade dos documentos. Cada servidor mantém uma estimativa da carga nos seus vizinhos. Periodicamente os nós transmitem por *broadcast* sua carga aos servidores vizinhos. O WebWave é proposto em [Heddaya97].

Em [Grimm98a] é relatado um processo de inclusão de informações de roteamento nos algoritmos de seleção de vizinhos para cooperação, enquanto que em [Inoue98] é proposto o uso do sistema WebHint, incluindo um servidor de "dicas" que permite a atualização automática de informações associadas a vizinhança entre caches de forma a diminuir o número de pacotes trocados entre os servidores e o tempo de espera dos usuários. Para isso, foram acrescentadas novas mensagens ao Internet Cache Protocol (ICP) para permitir o intercâmbio de informações entre os servidores de cache e o servidor de "dicas".

A implementação de um sistema de caches deve levar em consideração não apenas o aumento da eficiência da rede mas também os possíveis processos de manutenção de segurança e confiabilidade dos dados manuseados. Essa é a proposição feita em [Tewksbury98], baseado no fato de que uma malha hierárquica global faz com que a integridade da informação sofra um declínio e riscos de segurança aumentam, pois nada impede que o proprietário do cache altere o conteúdo dos documentos que estão armazenados, repassando documentos alterados aos clientes, reduzindo a qualidade de serviço, acessando dados confidenciais e censurando a permissão de acesso a determinados *sites*.

Através de experiências adquiridas, são sugeridas em [Grimm98] algumas regras práticas para a administração do sistema de caches como, por exemplo: colete dados dos seus caches para análise; determine o comportamento típico do grupo de usuários do serviço; conheça as especificações da topologia de rede e de links; obedeça ao princípio da localidade; evite experiências, planeje.

Em [Cormack96] são descritas características inerentes a utilização de caches para Web O artigo também descreve algumas características dos principais sistemas de cache para Web, mencionando o CERN proxy server, Jigsaw, Apache, Spinner, Purveyor, Catapult, Netscape Proxy Server, Harvest 1 e 2 e Squid. Descreve também algumas características das principais instalações de cache para Web no mundo, mencionando o HENSA (Reino Unido), NZGate (Nova Zelândia), NLANR (Estados Unidos), SingNet (Singapura), Uninett (Noruega) e CNRS (França).

2.4. Simulação de redes de computadores

A maioria dos trabalhos de pesquisa relacionados a caches para Web que têm sido realizados nos últimos anos utiliza-se de simuladores, por motivos de simplicidade e dificuldades com a realização de experiências com uma rede de caches real. Esses simuladores, na grande maioria das vezes, trabalham com apenas um cache, permitindo apenas a comparação entre diferentes configurações, tais como a política de substituição utilizada. O estudo de sistemas de caches cooperativos dispostos em redes hierárquicas também normalmente utiliza simuladores específicos. Os simuladores podem ler requisições armazenadas em *traces* obtidos a partir do acesso a caches reais ou gerados através de ferramentas de geração de carga, como o SURGE, descrito em [Barford98]

2.5. Coerência entre documentos

Todo Web cache precisa contar com um mecanismo que garanta o maior nível possível de coerência entre os documentos no servidor de origem e no cache. Esse mecanismo, porém, é mais simples que o utilizado em *file systems* distribuídos, pois ele não possui capacidade de escritas distribuídas, visto que o cliente não escreve dados no servidor.

Existem duas classes gerais de mecanismos de coerência de cache: o *check de validade* e o *callback*. No mecanismo de check de validade, quando um documento é colocado no cache é associado a ele um *timestamp*. Quando o cliente vai utilizar o documento, ele confere se o documento ainda é válido através de mensagens trocadas com o servidor. O cache pode fazer o check de validade a cada intervalo fixo, de modo a aumentar a performance. No caso do mecanismo de callback, o cliente recebe junto com o documento um certificado de callback, que garante que o cliente será notificado caso o documento seja alterado no servidor.

O cabeçalho do protocolo HTTP possui parâmetros que podem ser usados para exigir uma maior atualidade do documento, sendo utilizados como controle de coerência dos documentos no cache. A coerência dos documentos pode ser checada a cada intervalo ou através de um mecanismo de expiração, que faz com que o documento não seja mais válido após certo tempo.

Apesar do mecanismo de coerência por expiração ser um dos mais utilizados, ele possui as seguintes desvantagens: O usuário precisa esperar que o tempo de expiração do documento termine, não existe a opção de se exigir total coerência através da busca do documento diretamente do servidor de origem, o mecanismo não possui uma forte garantia quanto a atualidade do documento, o usuário não consegue especificar o grau de autenticidade requerido e o cache cancela a transmissão do documento antes do término, quando o usuário cancela antes do final da transmissão.

Foram propostos mecanismos de modo a resolver alguns desses problemas: Para um maior controle da coerência entre os documentos, é proposto que sejam retornados também fluxos de versões dos documentos em resposta a cada requisição e que seja permitido ao usuário especificar o grau de atualidade exigido para cada requisição.

A coerência entre os documentos armazenados nos caches e nos servidores são estudadas em [Dingle96] e [Gwertzman96].

2.6. Protocolos HTTP e ICP

Esta seção faz um resumo das características dos protocolos HTTP (*HyperText Transfer Protocol*) e ICP (*Internet Cache Protocol*). Esses protocolos são apresentados neste trabalho pela sua importância em assuntos relativos à distribuição de documentos via Web: enquanto o protocolo HTTP é o mais utilizado pelos usuários da Web, o ICP é responsável pela cooperação entre caches para Web.

2.6.1. HTTP (*HyperText Transfer Protocol*)

As informações sobre o protocolo HTTP contidas nesta seção foram baseadas principalmente em [Zotto96], [Comer00] e [Stevens94]. As definições e detalhes sobre o protocolo HTTP podem ser encontradas na RFC 822 (*Request for Comments*) [Crocker82] e através de documentos disponibilizados pela W3C (*World Wide Web Consortium*) [W3C03], órgão atualmente responsável pelo protocolo.

O protocolo HTTP está sendo usado globalmente pela World Wide Web desde 1990. É um protocolo do nível de aplicação e foi criado para prover objetividade e

rapidez necessárias para suportar sistemas de informação distribuídos cooperativos de hipermídia.

Sistemas de informação práticos requerem maior funcionalidade do que simples recuperação, incluindo pesquisa e atualização no servidor. O HTTP permite um conjunto aberto de métodos para ser usado para indicar o propósito de uma requisição. Ele constrói na disciplina de referência provida pela *Uniform Resource Identifier* (URI), como uma locação (URL) ou nome (URN) para indicar em qual recurso um método deve ser aplicado. As mensagens são passadas em um formato similar ao usado pelo *Internet Mail* e o *Multipurpose Internet Mail Extensions* (MIME).

O HTTP também é usado como um protocolo genérico para comunicação entre agentes usuários e *proxies* ou *gateways* com outros protocolos Internet, tais como SMTP, NNTP, FTP, Gopher e WAIS, permitindo acesso básico hipermídia para recursos disponíveis de aplicações diversas e simplificando a implementação de agentes usuários.

2.6.2. Terminologia usada pelo HTTP

Essa seção faz uma breve descrição dos principais termos definidos em uma comunicação utilizando o protocolo HTTP:

- **Connection**: Circuito virtual na camada de transporte estabelecido entre dois programas aplicativos para o propósito de comunicação.
- **Message**: A unidade básica de comunicação HTTP, consistindo de uma seqüência estruturada de octetos de sintaxe definida e transmitida via conexão.
- **Request**: Uma mensagem de requisição HTTP.
- **Response**: Uma mensagem de resposta HTTP. Essa mensagem de resposta é relativa a uma mensagem de requisição precendente.
- **URL**: (*Uniform Resource Locator*) Uma *string* que da a localização de uma entidade de informação. Essa *string* começa com o tipo do protocolo (FTP, HTTP, ...), seguido pelo identificador do servidor e o *path* para encontrar a entidade.

- **URN**: (*Uniform Resource Name*) Uma *string* que dá a locação de uma entidade de informação. Ao contrário da URL, é garantido que uma URN exista por um longo período de tempo.
- **URI**: (*Uniform Resource Identifier*) Termo genérico usado para se referir a uma URL ou a uma URN.
- **Resource**: Um objeto de dados da rede ou serviço que pode ser identificado por uma URI.
- **Entity**: Uma representação particular de um recurso de dados, ou resposta de um recurso de serviço, que pode ser incluído numa mensagem de pedido ou de resposta.
- **Client**: Um programa aplicativo que estabelece conexões para o propósito de enviar pedidos e receber respostas.
- **User agent**: O cliente que inicia o pedido. Normalmente *browsers*, editores, *spiders* (robôs que pesquisam através da rede), ou outras ferramentas de usuário final.
- **Server**: Um programa aplicativo que aceita conexões para atender pedidos e enviar respostas.
- **Origin server**: O servidor no qual um dado recurso é armazenado inicialmente.
- **Proxy**: Um programa intermediário que atua duplamente como servidor e como cliente para fazer pedidos em nome de outros clientes. Proxies são geralmente usados como portais para clientes através de *firewalls* e como auxiliares de aplicações para manipular pedidos via protocolos não implementados pelo agente usuário.
- **Gateway**: Um servidor que atua como intermediário na comunicação entre clientes e outros servidores. Gateways são geralmente usados como roteadores de requisições e como tradutores de protocolos em sistemas não-HTTP.

2.6.3. Funcionamento do protocolo HTTP

O protocolo HTTP é baseado no paradigma cliente/servidor, que também pode ser entendido como um paradigma pedido/resposta. Um cliente estabelece uma conexão com

um servidor e envia um pedido ao servidor, o qual o analisa e responde. A conexão deve ser estabelecida antes de cada pedido de cliente e encerrada após a resposta. As mensagens seguem o formato da RFC822 [Crocker82], e suas atualizações.

Requisições

Uma mensagem de pedido de um cliente a um servidor inclui o método a ser aplicado ao recurso, o identificador do recurso e a versão do protocolo em uso.

O formato da mensagem de pedido enviada do cliente ao servidor é descrito abaixo, de acordo com a notação BNF (*Backus-Naur Form*) [Aho86]:

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

Respostas

Após receber e interpretar uma mensagem de pedido, um servidor responde na forma de uma mensagem de resposta HTTP:

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

Linha de Status

A primeira linha de uma *Full-Response* é a *Status-Line*, consistindo da versão do protocolo, seguida de um código de status e sua frase de texto associada, com cada elemento separado pelo caractere SP (*espaço = chr(32)*). Nenhum CR (*carriage return*) ou LF (*line feed*) é permitido, exceto o CRLF final da seqüência.

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

Código de Status

O elemento *Status-Code* é um inteiro de 3 dígitos, resultado da tentativa de entender e satisfazer o pedido. O primeiro dígito define a classe da resposta. Os últimos dois dígitos não têm nenhuma categorização. Os possíveis *Status-Code* são apresentados na tabela 2.6.1.

Status- Code	Significado
1xx	Informacional - Não usado, mas reservado para uso futuro.
2xx	Sucesso - A ação foi recebida, entendida e aceita.
3xx	Redirecionamento - Ações adicionais devem ser executadas para completar o pedido.
4xx	Erro no cliente - O pedido contém erro de sintaxe ou não pode ser completado.
5xx	Erro no servidor - O servidor falhou em completar um pedido aparentemente válido.

Tabela 2.6.1 – Códigos de retorno de requisições HTTP

2.6.4. ICP (*Internet Cache Protocol*)

O protocolo usado para a troca de mensagens entre caches para Web é o ICP (*Internet Cache Protocol*), que é descrito em [Wessels97]. O ICP é um formato de mensagens usado para comunicação entre caches para Web. É principalmente utilizado em malhas de caches para localização de documentos.

O ICP normalmente é implementado usando UDP, mas não existem requisitos que o limitem ao UDP. Foi escolhido utilizar o UDP por proporcionar uma maior velocidade na troca de mensagens entre os servidores de cache para Web. O formato de mensagem consiste em um cabeçalho de tamanho fixo de 20 octetos seguido por um "payload" de tamanho variável. Um dos campos do cabeçalho é o *ICP Opcode*, que contém um valor indicando o tipo da mensagem. Esse valor pode indicar que a mensagem é de consulta, resposta de sucesso, resposta de erro, dentre outras.

Nesse protocolo, são definidas duas relações de hierarquia entre os caches: pais e irmãos. Um cache pai está essencialmente um nível acima na hierarquia enquanto que caches irmãos estão no mesmo nível.

Quando um cache recebe uma requisição através do protocolo HTTP de um cliente, o cache envia consultas através do ICP para outros caches. Esses caches respondem ao cache inicial que, considerando as respostas, decide pra onde enviar a requisição do documento.

Para uma distribuição mais eficiente de mensagens, um servidor de cache pode enviar requisições para endereços *multicast* e os caches vizinhos podem se unir ao grupo *multicast* para receber as mensagens.

Com relação a aspectos de segurança, falsificação, alteração, inserção ou bloqueio de mensagens, o ICP pode fazer com que a requisição HTTP falhe apenas nos casos em que o cache está atrás de um *firewall* e não pode acessar o servidor de origem diretamente ou no caso em que um cache vizinho responda sempre que possui o documento mas se negue a transmiti-lo para o cliente. As principais aplicações do ICP são apresentadas em [Wessels97a].

2.7. Fluxos de requisições

As estratégias para melhora de desempenho em caches para Web são desenvolvidas a partir de características do fluxo de requisições feitas pelos clientes e dos arquivos transmitidos via Web.

Na análise dos resultados, apresentados em [Cunha95], alguns padrões foram observados no uso da WWW. Em particular, foi observado que muitas características do uso podem ser modeladas através de distribuições estatísticas, incluindo a distribuição dos tamanhos dos documentos, a popularidade dos documentos, relativamente ao seu tamanho e o número de referências a documentos como uma função da sua popularidade.

O princípio da localidade tem conseqüências muito importantes no projeto de sistemas de computação. Fluxos que apresentam localidade temporal são beneficiados com sistemas de caches enquanto que fluxos com localidade espacial se beneficiam do

uso de *prefetching*. Em [Almeida96] é usado um método quantitativo baseado na distância de pilha para medir as localidades espacial e temporal de fluxos da Web.

O modelo de distância de pilha, que captura os relacionamentos de localidade temporal presentes em um fluxo é definido da seguinte forma:

Considerando $Rt = r1, r2, ..., rt$, onde Rt é o nome do objeto requisitado no tempo t e a pilha St, ordenada por LRU, num servidor, contendo os objetos Oi, sendo O_1 o mais recente, define-se Dt como a distância do objeto t ao início da pilha. Assim, se $Rt = Oi$ então $Dt = i$.

Através de análises dos *traces*, foi então concluído que os métodos simples de geração de fluxos, baseados em popularidade do documento não captura as localidades temporal nem espacial.

A auto-similaridade *(self-similarity)* é uma propriedade associada a fractais, onde o objeto apresenta a mesma forma, não importando a escala na qual é visto. A ocorrência de auto-similaridade em distribuições de requisições Web é analisada em [Crovella95]. As razões do tráfego em redes apresentar características de auto-similaridade não foram claramente identificadas, sendo que nesse trabalho é mostrado que em alguns casos a auto-similaridade pode ser explicada em termos de sistemas de arquivos e comportamento do usuário.

Uma das mais importantes conclusões obtidas foi que os gráficos dos tempos de transmissão dos documentos pela Web obedecem à uma distribuição de cauda pesada. As transferências apresentam auto-similaridade quando a demanda por documentos é alta.

A análise do tráfego na Internet relatada em [Morris00] mostrou que o tráfego não segue uma distribuição de Poisson. Além disso, existe um ciclo de 24 horas que apresenta tráfegos maiores em determinadas horas do dia, exigindo que as redes sejam projetadas para suportar esse tráfego de pico e não simplesmente a média. Usando experimentos baseados em *traces*, foram apresentadas evidências de que a variância da quantidade de banda necessária é proporcional à média do tráfego. Esse resultado permite um projeto mais realista de redes para suportar o tráfego de dados.

Apesar da modelagem do tráfego na Internet não ser simples, segundo experiências descritas em [Breslau99], um modelo simples para as requisições de documentos da Web pode ser suficiente para entender certas propriedades assintóticas do

desempenho de caches, tais como o desempenho relativo ao tamanho do espaço de armazenamento. Porém, regras tais como a "10/90", que diz que 90% dos acessos se destinam a 10% dos itens não aplicou a muitos fluxos.

2.8. *Clusters* e servidores paralelos

A capacidade de enviar documentos dos servidores Web pode se tornar um problema com o aumento da popularidade dos servidores e o conseqüente aumento do número de requisições aos servidores mais populares. A carga sobre os servidores Web tem se tornado maior principalmente pela utilização de tecnologias que permitem a geração de páginas Web dinâmicas.

A saída é o paralelismo [Carriero92]. Ele foi criado da necessidade de conseguir desempenhos maiores que o conseguido pelos melhores supercomputadores construídos com a tecnologia atual e isso é aplicável a servidores Web. Os principais trabalhos relativos a utilização de clusters e servidores distribuídos, principalmente relativos a escalabilidade e tolerância a falhas estão em [Bung99], [Andresen96], [Narendran97], [EWS02] e [Baker99].

Este projeto de pesquisa direcionou os seus esforços para a arquitetura Beowulf, devido às características de alta escalabilidade e baixo custo. As principais informações referentes a essa arquitetura podem ser encontradas em [Beowulf02].

Para usar o poder do paralelismo no serviço de prover documentos via Web é muitas vezes necessário usar técnicas de balanceamento de carga de trabalho. Essas técnicas têm como objetivo dividir a carga de trabalho aplicada ao servidor entre os nós que o formam. Dentre os principais artigos que apresentam estratégias de balanceamento de carga estão [Mourad97], [Cardellini99] e [Cardellini99a].

2.9. *Prefetching* de arquivos

Apesar do *prefetching* realmente diminuir a média do tempo de espera dos usuários, sua utilização é principalmente recomendada para caches locais, principalmente

em computadores com conexões não compartilhadas, pois a sua utilização em redes compartilhadas pode gerar tráfego na rede que pode vir a prejudicar outros usuários.

Em [Chinen97] é relatado o desenvolvimento de um esquema de *prefetching* chamado "prefetching interativo", onde as referências são colhidas do código HTML que está sendo acessado pelo cliente, buscando os documentos referenciados no código e reduzindo a latência no uso da WWW.

Foi decidido nesse trabalho que o sistema de *prefetching* seria implementado em um servidor proxy, pois se o servidor fizer o *prefetch* os clientes poderão compartilhar dos resultados. Além disso, muitos clientes são produtos comerciais, tais como o Netscape Communicator ou o Intenet Explorer, sendo então difícil substituir os clientes com sistemas com mecanismos de *prefetching*.

Foi constatado que a medida em que é aumentada a atuação do *prefetching*, é aumentada também a taxa de *hit ratio* do cache. Porém, a sobrecarga causada na rede e nos servidores faz com que seja necessário um balanceamento entre a carga na rede e a taxa de acerto.

O trabalho descrito em [Venkata96] investiga um esquema de redução da latência percebida pelos usuários através do *prefetching* de documentos que têm maiores possibilidades de serem acessados brevemente, enquanto o usuário está lendo a página atual.

Nessa proposta, o servidor computa quais as páginas que têm maiores chances de serem acessadas depois da página requisitada, enviando essa informação ao cliente. O cliente então decide se deseja fazer o *prefetching* dessas páginas. Uma vez que o mecanismo de *prefetching* decide buscar uma página, ele envia uma requisição ao servidor.

O algoritmo de predição constrói o grafo de dependência que representa o padrão de acesso a diferentes arquivos armazenados no servidor. O grafo tem um nó para cada arquivo que pode ser acessado. Existe um arco do nó A para o nó B, se e somente se, em algum ponto do tempo B foi acessado w vezes depois de A, sendo w pré-definido. O grafo de dependência é dinamicamente atualizado enquanto o servidor recebe novas requisições.

A sugestão desse trabalho é que o *prefetching* seja particularmente útil em links não compartilhados (*dial up*) e em *links* de banda larga, com alta latência (satélites).

Em [Bestavros95] é proposta a utilização de serviços especulativos. A diferença entre serviço especulativo e prefetching é que no serviço especulativo é o servidor quem coordena o envio de documentos extras.

Serviços especulativos implicam que requisições de clientes a documentos são respondidas enviando, adicionalmente ao documento requisitado, outros documentos os quais o servidor especulou que seriam requisitados pelo cliente num futuro próximo. Essa especulação é baseada numa estatística de acessos mantida sobre cada documento.

Capítulo 3

Simulação de redes de caches

3.1. Simulação computacional

A simulação pode ser definida como a disciplina cujo objetivo é imitar aspectos da realidade da forma mais fiel possível. Um termo que pode ser considerado um sinônimo de simulação é *realidade artificial* [Jeruchin94].

Existem alguns motivos que induzem ao uso de simulação. Um deles é a possibilidade de obter resultados em simuladores antes dos sistemas reais. Isso porque o tempo da simulação pode correr mais rápido que o tempo no sistema real.

Outra importante razão para utilizar a simulação é a impossibilidade de fazer experiências com o sistema real o qual deseja-se estudar. Esse é o caso do estudo de desempenho de redes de computadores, onde é bastante difícil conseguir uma grande rede para fazer experiências. Pelo mesmo motivo, experimentos com redes de caches para Web normalmente utilizam-se de simuladores.

Simulação é essencialmente um trabalho com analogias [Shimizu75]. É uma modalidade experimental de pesquisa que procura analisar situações e tirar conclusões através do exercício com modelos que representam a realidade.

Um modelo é a descrição de algum sistema de forma a predizer o que aconteceria se determinados eventos acontecessem no sistema real modelado [Bratley87]. Normalmente, um modelo simplifica o sistema simulado, representando apenas os

elementos que interferem no comportamento dos valores e objetos os quais deseja-se estudar. Muitas vezes, para manter a simplicidade do modelo, elementos que têm pouco impacto nos resultados no sistema real são desconsiderados quando o modelo é construído.

As simulações normalmente são classificadas em dois tipos: físicas ou simbólicas [Jeruchin94]. Numa simulação física, há apenas um processo de miniaturização ou representação parcial de um sistema real. Um exemplo é o uso de miniaturas e maquetes.

Quando o modelo conserva apenas características lógicas do sistema real, existe a chamada simulação simbólica. Nesse caso, apenas uma descrição lógica dos elementos do sistema e suas relações são utilizadas no modelo. As simulações computacionais são classificadas nessa categoria.

3.1.1. Software de simulação

A utilização de computadores no processo de simulação é feita implementando-se um programa simulador.

Existem algumas estruturas que normalmente aparecem em programas de simulação. As principais são os mecanismos de sincronia de eventos e as estruturas de dados que representam os elementos.

Apesar de ser relativamente simples implementar um simulador usando uma linguagem de alto nível, desde que o sistema simulado seja relativamente simples [Bratley87], pode-se muitas vezes utilizar-se de linguagens criadas para implementar simuladores tais como Simscript [SIMSCRIPT02], GPSS [Karian99] e Simula[ASU02].

Apesar das linguagens especializadas terem sido construídas especialmente para implementar simuladores, proporcionando um código menor do que linguagens de alto nível de uso geral, elas possuem a desvantagem de serem pouco conhecidas. É muito fácil encontrar um programador C++ ou Pascal, mas seria bem mais difícil encontrar um programador de uma linguagem especializada para, por exemplo, fazer uma alteração em um simulador.

Assim, hoje em dia, a maioria dos simuladores é implementada em uma linguagem de alto nível tal como C, Pascal ou Java, ou utiliza-se de ferramentas de

simulação específicas disponíveis, tais como o NS (*Network Simulator*), uma ferramenta de simulação de redes de computadores [NS02].

3.1.2. Corretude da simulação

Um dos problemas mais difíceis em simulação computacional é a prova da corretude da simulação e, por conseqüência, prova da validade dos resultados obtidos na simulação [Bratley87].

A corretude de um processo de simulação representa, muitas vezes com uma certa flexibilidade, a igualdade entre os resultados do sistema real e da simulação desse sistema. Métodos para garantir a certeza da corretude de um processo de simulação computacional devem ser utilizados, visto que a comparação dos resultados da simulação com os do sistema real para todas as possibilidades do sistema real não faz muito sentido, pois nesse caso, poder-se-ia utilizar diretamente os dados do sistema real.

Existem três principais métodos de validação de processos de simulação, ordenados decrescentemente de acordo com seu grau de confiabilidade: prova matemática, construção e semelhança com sistemas reais [Bratley87].

A validação por prova matemática é feita quando os resultados do processo de simulação são obtidos através de um processo cuja corretude pode ser matematicamente provada. Garante uma confiabilidade total nos resultados porém, não são muitos os casos em que pode ser aplicada.

No caso da validação por construção, a validação é feita através da demonstração de que as ações realizadas no simulador correspondem diretamente a ações no sistema real. Esse método possui a vantagem de ajudar no projeto do simulador, que contém o detalhamento do fluxo de ações do simulador. Entretanto, principalmente em simulações de sistemas complexos, é difícil demonstrar com clareza que as atitudes tomadas pelo simulador representam fielmente as ações no sistema real naquelas circunstâncias.

Na validação por semelhança com sistemas reais, simula-se inicialmente condições de sistemas reais dos quais possui-se resultados reais. Comparando-se os resultados reais com os resultados das simulações, pode-se verificar a semelhança entre os mesmos. Apesar de não validar uma simulação com alto grau de confiabilidade, esse

método é muito importante principalmente para detectar disparidades em resultados de simulações.

3.2. Simulação de malhas de caches cooperativos

Para descobrir qual a melhor configuração de parâmetros relativos a um determinado sistema de caches para Web, deve-se testar várias combinações de valores para os parâmetros de forma a determinar qual dessas combinações possibilita o melhor desempenho. No caso de malhas de caches cooperativos, deve-se considerar também a distribuição hierárquica dos servidores e a capacidade de transmissão das redes que os interligam. Devido à quantidade de recursos envolvidos nesse tipo de pesquisa, fica difícil conseguir uma malha real de caches para fazer experiências. A simulação se torna então a opção mais viável para pesquisas.

Um dos principais problemas encontrados na simulação de malhas de caches para Web é a manutenção de coerência entre as operações do simulador e as operações de uma malha real de caches. Assim, torna-se necessário utilizar históricos de requisições reais, feitas a caches reais, no simulador. Esses históricos de requisições são chamados *arquivos de log* ou *traces* e podem ser conseguidos em vários *sites* na Internet.

Neste capítulo é apresentado um simulador totalmente configurável de malhas hierárquicas de caches para Web. Esse simulador utiliza *traces* de caches reais, permitindo verificar o impacto da variação de parâmetros no desempenho de toda a malha de caches.

A cooperação entre servidores de caches para Web permite um melhor desempenho de todo o sistema de caches. Quando um servidor não possui uma cópia de um documento requisitado, ele pode enviar uma mensagem a um outro servidor de cache próximo perguntando se esse outro servidor tem possui uma cópia do documento. Em caso afirmativo, pode-se satisfazer a requisição do cliente com essa cópia. Caso o outro cache também não possua uma cópia do documento, pode-se também optar por requisitar a um servidor de cache de hierarquia superior. Esse outro servidor de cache, geralmente de maior capacidade, recebe requisições não satisfeitas de servidores menores.

Esse procedimento pode prosseguir até o cache de maior nível hierárquico que, se também não encontrar o documento, irá requisitá-lo ao servidor real do documento, guardando uma cópia para si.

Além dos parâmetros que interferem no desempenho de caches não cooperativos, quando é configurada uma malha de caches cooperativos, novos parâmetros interferem no desempenho de toda a malha. Dentre esses parâmetros, pode-se citar a configuração de uma estrutura hierárquica entre os caches e as capacidades da rede de intercomunicação entre os caches. Além disso, a demanda por memória e capacidade de processamento em servidores de uma malha é maior, pois eles devem executar também os protocolos de troca de informações entre caches.

A figura 3.2.1 apresenta um exemplo de malha de servidores de caches distribuídos e cooperativos. Durante a operação dos servidores representados nesse exemplo, usuários fazem requisições aos caches 1, 2 e 3. Caso o cache que recebeu a requisição (1, 2 ou 3) não encontre o documento requisitado, ele envia uma mensagem aos dois caches vizinhos (no caso do cache 1 os vizinhos são os caches 2 e 3), perguntando sobre o documento. Caso nenhum possua o documento, é feita uma requisição ao cache 4. Se o cache 4 possuir o documento requisitado, ele envia uma cópia ao cache de nível mais baixo que requisitou. Esse cache armazena então uma cópia e envia o documento requisitado para o cliente. Caso o cache 4 não possua o documento, ele fará uma requisição ao servidor real do documento, guardando uma cópia para si e enviando outra para o cache que requisitou.

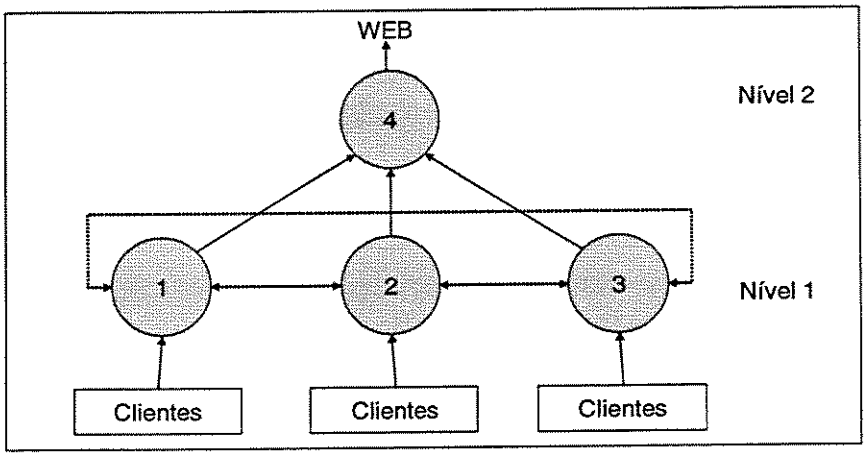


Fig. 3.2.1 - Exemplo de malha de caches cooperativos

É interessante ressaltar que a capacidade de comunicação entre os caches 1, 2 e 3 geralmente é maior que aquele entre esses caches e o cache 4. Isso ocorre porque geralmente o cache 4 está geograficamente distante dos caches 1, 2 e 3. Como o cache 4 recebe um número maior de requisições que os caches 1, 2 e 3, ele deve possuir uma capacidade de armazenamento e processamento maior que os outros caches.

Além do *Hit Ratio* e do *Byte Hit Ratio*, o simulador implementado neste trabalho usa uma outra métrica muito importante: o tempo de resposta ao usuário, que indica qual o impacto no tempo de espera do usuário das variações de parâmetros de configuração.

3.3. O simulador

As principais características do simulador apresentado neste trabalho são a forma de configuração, o sistema de manutenção e troca de mensagens, a manutenção de listas de arquivos e aplicação das políticas de substituição, o cálculo do atraso em transmissões de dados e processamento e a geração de resultados. Cada uma dessas características será discutida a seguir.

3.3.1. Configuração da malha

A configuração da malha a ser simulada é feita através de um arquivo texto que contém, numa linguagem predefinida, a descrição das características da malha. A listagem 3.3.1 apresenta um exemplo de arquivo de configuração que pode ser utilizado para simular a malha representada na figura 3.2.1.

```
structdef
  label Example simulation
  cachedef 1 100
    logfile ..\logs\log1.log
    resolveMiss 4 10
    sub 1000 0 0 lru
    cacheCoop 2 100
    cacheCoop 3 100
  cachedef 2 100
    logfile ..\logs\log2.log
    resolveMiss 4 10
    sub 1000 0 0 size
```

```
    cacheCoop 1 100
    cacheCoop 3 100
  cachedef 3 100
    logfile ..\logs\log3.log
    resolveMiss 4 10
    sub 1000 0 0 lru
    cacheCoop 1 100
    cacheCoop 2 100
  cachedef 4 500
    logfile NONE
    resolveMiss 0 0
    sub 3000 0 0 din
end.
```

Listagem 3.3.1 - Exemplo de arquivo de configuração

Os comandos e parâmetros utilizados no arquivo de configuração da listagem 3.3.1 são descritos a seguir.

- **structdef** - Marca o início da definição de uma estrutura de caches a ser simulada.
- **label** - Permite a atribuição de um título à simulação, possibilitando um melhor acompanhamento durante a execução da mesma. Permite também uma melhor identificação durante a análise dos resultados das simulações.
- **cachedef** ***id cap*** - Cria um cache com o identificador ***id*** sendo executado num processador de capacidade ***cap***. Essa capacidade indica o poder de processamento do computador onde o cache está implantado. Essa medida de capacidade não possui unidade, sendo que seu valor numérico não é importante se considerado sozinho. Terá grande importância para permitir a comparação de atrasos devido a capacidade de processamento entre dois computadores de capacidades diferentes.
- **resolveMiss** ***id*** -Indica que requisições não resolvidas no cache local nem em caches vizinhos deve ser enviada ao cache cujo identificador é *id*. Isso significa que o cache *id* é um cache de nível superior.
- **logfile** ***arqName*** -Define que esse cache vai simular o recebimento das requisições do arquivo de log *arqName*. Caso não exista um arquivo de log, utiliza-se a palavra NONE.
- **sub** ***tam arqMin arqMax política*** - Define uma partição de tamanho ***tam*** do espaço de armazenamento do cache. Nessa partição serão armazenados os arquivos cujos tamanhos estão entre ***arqMin*** e ***arqMax***. No caso de ***arqMax*** ser 0, significa que não

existe tamanho máximo para os arquivos que podem ser armazenados. O parâmetro ***política*** indica qual política de substituição será utilizada nessa partição do cache.

- **cacheCoop *id cap*** - Define que documentos não encontrados localmente devem ser procurados no cache de identificador *id* antes de encaminhar a requisição ao cache de nível superior. O parâmetro *cap* indica a capacidade da rede entre o cache local e o cache *id*.
- **end.** - Indica o final do arquivo de configurações dos caches.

3.3.2. Troca de mensagens

Cada cache da malha deve ser capaz de trocar mensagens com os outros caches para permitir a cooperação. A tabela 3.3.1 apresenta os tipos de mensagens definidas no simulador. Essas mensagens correspondem a tipos de mensagens definidas no *Internet Cache Protocol* [Wessels97].

Tipo	Mensagem do ICP	Significado
1	ICP_OP_QUERY	Requisição de usuário
2	ICP_OP_QUERY	Procurar documento em cache vizinho
3		Procurar documento pedido por cache vizinho
4	ICP_OP_HIT / ICP_OP_MISS	Responder ao vizinho
5		Processar resposta de vizinho
6	ICP_OP_QUERY	Enviar requisição a cache de nível superior
7		Adicionar documento

Tabela 3.3.1 - Mensagens definidas no simulador

Mensagens são trocadas entre os caches de modo a determinar se algum deles possui o documento requisitado. Se o documento é encontrado, ele é enviado ao usuário que o requisitou.

Cada cache mantém uma fila de requisições a ser processada no futuro. As requisições lidas dos arquivos de log são transformadas em mensagens do tipo 1 e enfileiradas na fila de requisições do cache. Ao encontrar uma requisição do tipo 1, é executada uma procura na lista de arquivos armazenados no cache. Se o arquivo não for

encontrado, é enfileirada uma mensagem do tipo 2 na própria fila de requisições, indicando que aquela requisição deve ser enviada aos vizinhos. É enfileirada então, em cada vizinho, uma requisição do tipo 3. Quando é encontrada uma requisição do tipo 3 na fila, um cache entende que deve procurar um arquivo para um vizinho. A procura é feita e o resultado é colocado em uma mensagem do tipo 4. Quando uma mensagem do tipo 4 é encontrada, é enfileirada no cache que requisitou a pesquisa uma mensagem do tipo 5, que significa resposta a uma requisição. Quando algum cache vizinho responde que encontrou o documento, ele é enviado para o usuário. Caso nenhum vizinho contenha o documento, o cache enfileira na própria fila uma mensagem do tipo 6, que significa enviar a requisição ao cache de nível superior. Quando o documento requisitado é encontrado na malha, é enfileirada uma mensagem do tipo 7 em cada cache que enviou uma requisição a cache de nível superior. O processamento dessa mensagem faz com que o arquivo seja armazenado.

Além da fila de requisições, cada cache deve armazenar também quais requisições foram feitas a caches vizinhos e ainda não foram respondidas. Dessa forma, é possível determinar quantas respostas negativas faltam para concluir que nenhum vizinho possui o documento. São também armazenadas as requisições enviadas a caches de nível superior.

As filas de requisições são ordenadas segundo a hora em que a requisição foi feita. Dessa forma, é possível garantir que as requisições sejam processadas no simulador na mesma ordem que seriam processadas numa malha real de caches.

3.3.3. Políticas de Substituição

Nesse simulador podem ser simuladas as políticas LRU, SIZE e Dinâmica. As políticas LRU e SIZE são duas das políticas mais utilizadas em caches para Web. A política dinâmica foi desenvolvida neste projeto, sendo o simulador desenvolvido neste trabalho o primeiro a utilizá-la. A extensão da política dinâmica levou ao desenvolvimento do GDE (Gerenciador Dinâmico de Espaço), apresentado no capítulo 6. As políticas que podem ser utilizadas são descritas a seguir.

- LRU *(Least Recently Used)* - É removido o documento menos recentemente acessado. Dessa forma, mantém-se no cache os documentos que foram acessados mais

recentemente, sendo esses os documentos com maior probabilidade de serem reacessados.

- SIZE - Quando essa política é aplicada a um conjunto de arquivos, é removido o maior documento do conjunto. Dessa forma, tenta-se evitar que muitos arquivos pequenos tenham que ser removidos para a inserção de um único arquivo grande.
- Dinâmica - Essa política utiliza três filas de documentos. Essas filas contem os mesmos documentos, porém ordenados de maneira diferente. A primeira é ordenada utilizando a política LRU, a segunda usando a LFU e a última é ordenada através da política SIZE. Dessa forma, é possível considerar tanto o tamanho dos documentos quanto a freqüência com que foram acessados e o tempo desde seu último acesso, o que permite tomar uma decisão mais correta de qual documento remover do cache. A política dinâmica apresentou nos experimentos os melhores resultados, medidos em *Hit Ratio* quanto em *Byte Hit Ratio.*

Apesar de serem descritas neste trabalho apenas as políticas LRU, SIZE e Dinâmica, a implementação do simulador permite que várias outras políticas, tais como as descritas em [Cao97] sejam utilizadas para fins de análise de desempenho.

3.3.4. Particionamento do cache

O particionamento do cache permite uma melhor adequação do espaço de armazenagem às diferentes classes de tamanhos dos arquivos. O particionamento permite evitar que muitos arquivos pequenos sejam removidos para o armazenamento de um único arquivo grande [Murta98]. Com uma configuração correta das partições é possível aumentar o *Hit Ratio* de um cache mantendo-se praticamente inalterado o valor do *Byte Hit Ratio.*

O particionamento possui a desvantagem de necessitar de um estudo aprofundado das classes de tamanhos dos arquivos de forma a definir uma correta divisão do espaço de armazenamento. Apresenta também o problema de fragmentação do espaço de armazenagem [Murta99]. A fragmentação ocorre quando o espaço livre total do cache é suficiente para armazenar um arquivo porém, o espaço livre está fragmentado entre as partições. Assim, arquivos deverão ser eliminados do cache para a entrada de um novo.

Esse simulador é capaz de definir partições de qualquer tamanho para os caches. Em cada uma dessas partições pode ser utilizada uma política de substituição diferente.

3.3.5. Filas de documentos

Cada partição mantém uma fila dos nomes dos documentos armazenados na partição. A ordenação dessa fila depende da política de substituição utilizada pela partição. No caso de utilizar a política LRU, os documentos são ordenados por tempo de acesso, sendo que quanto mais próximo ao início da fila, mais antiga é a requisição ao documento. Essa ordenação é feita inserindo documentos apenas no final da fila. Para remover o elemento mais antigo da fila basta remover o primeiro elemento.

A fila SIZE é ordenada de acordo com o tamanho dos documentos. Nessa fila, quanto mais próximo da cabeça da fila, maior o documento. Nessa fila, para remover o maior documento, basta remover o primeiro da fila. Porém, ao contrário da fila LRU, deve-se procurar a posição correta com relação ao tamanho para inserir cada novo documento.

No caso da política dinâmica, existe uma fila de documentos com três ordenações, feitas pelas políticas LRU, LFU e SIZE. O modelo utilizado na política de substituição dinâmica está representado esquematicamente na figura 3.3.1.

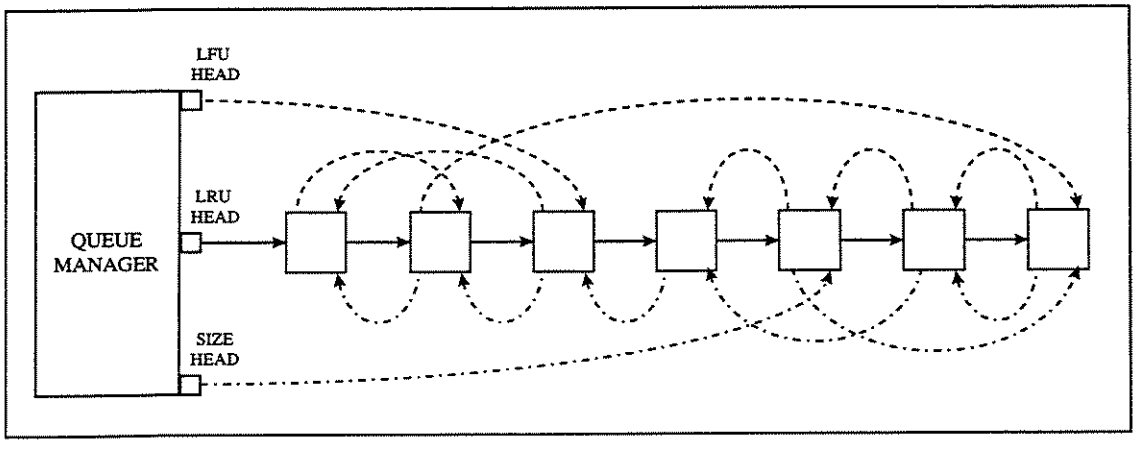


Fig. 3.3.1 - Modelo da política de substituição dinâmica

Cada elemento da fila é ligado ao seu antecessor e sucessor na ordenação LRU, LFU e SIZE. Dessa forma, é possível escolher dinamicamente qual política aplicar para

efetuar cada remoção de documento. Devido à essa característica, essa política possibilita um melhor desempenho do sistema de caches.

3.3.6. Cálculo de atrasos

Para simular os atrasos ocorridos na transmissão de mensagens e documentos é utilizada uma ordenação de eventos. Essa ordenação faz com que os eventos aconteçam no simulador na mesma ordem que aconteceriam num sistema de caches reais. Para isso, é associado à cada evento um *timestamp*, que indica o tempo em que o evento ocorreu. Assim, a cada operação é calculado o tempo que essa operação levaria num cache real, considerando as capacidades de rede e processamento. Esse tempo é somado ao *timestamp* do evento e ele é então colocado na fila do cache que irá processá-lo em ordem crescente de *timestamp*.

A cada iteração do simulador, é analisado qual evento tem o menor *timestamp* dentre todos os eventos nas filas de todos os caches do sistema. Processando sempre o evento de menor *timestamp* é possível garantir uma correta relação de antecedência entre os eventos, simulando corretamente a operação de um sistema de caches reais.

3.3.7. Resultados da simulação

Os resultados da simulação são armazenados em um arquivo texto. Esse arquivo contém informações tais como o número de requisições processadas, o *hit ratio* e o *byte hit ratio* de cada cache e de todo o sistema e também os tempos de atraso em processamento e comunicação.

A listagem 3.3.2 apresenta uma parte de um arquivo de resultados relativo à simulação de uma malha semelhante a da figura 3.2.1.

```
Simulação  1:  Exemplo de malha

Numero de requisições : 1245157
Numero total de Hits: 644507 (51.761%)
Numero total de Miss: 600650 (48.239%)

Total de bytes requisitados: 4426.061MB
Total de bytes encontrados : 1772.698MB (40.051%)

Estatísticas por Caches e Partições
Cache ID: 1
```

```
  Total do cache:
    Numero de Hits: 39026
    Numero de Miss: 643233
    Bytes encontrados      : 220.298 MB
    Bytes não encontrados : 2989.098 MB

Cache ID: 2
    Numero de Hits: 238127
    Numero de Miss: 657249
    Bytes encontrados      : 1002.683 MB
    Bytes não encontrados : 3107.734 MB

Cache ID: 3
    Numero de Hits: 334470
    Numero de Miss: 661110
    Qtde de bytes encontrados      : 269.344 MB
    Bytes não encontrados : 3160.820 MB

Cache ID: 4
    Numero de Hits: 35696
    Numero de Miss: 600650
    Bytes encontrados      : 311.477 MB
    Bytes não encontrados : 2652.714 MB

Mensagens trocadas: 3893112 (3.127 por requisição)
Requisições enviadas a vizinhos : 2656116 (2.133 por requisição)
Requisições enviadas a cache pai: 636346 (0.511 por requisição)

Tempo médio de atraso em transmissões : 0.315 unidades de tempo
Tempo médio de atraso em processamento: 0.384 unidades de tempo
Tempo médio de espera do usuário: 1614 unidades de tempo
```

Listagem 3.3.2 - Exemplo de arquivo de resultado

3.3.8. Execução em computadores paralelos

Devido ao grande volume de processamento necessário para executar simulações, é interessante utilizar computadores paralelos. Porém, para aproveitar a capacidade de processamento dos computadores paralelos foi necessário desenvolver uma versão paralela do simulador. Essa versão paralela foi desenvolvida utilizando o MPI (Message Passing Interface). [MPI02], [Gropp02].

Na versão paralela, a cada processo iniciado é associado um identificador. Esse identificador é um número inteiro e o primeiro processo recebe identificador 0, o segundo processo recebe identificador 1 e assim por diante. Além disso, cada processo sabe quantos processos estão sendo executados ao mesmo tempo.

Ao ler o arquivo de configuração, cada processo só executa as simulações quando a seguinte condição acontece: (NumSim *mod* NumProc = ID), onde NumSim é o número da simulação, NumProc é o número total de processos e ID é o identificador do processo.

Por exemplo, no caso de existirem 3 processos para realizar 8 simulações, o processo 1 vai executar as simulações 1, 4 e 7; o processo 2 executará as simulações 2, 5 e 8 e o processo 3 as simulações 3 e 6. Dessa forma, é possível aproveitar o poder de processamento de um computador paralelo para realizar um número muito maior de simulações.

A execução paralela das simulações possui uma alta granularidade, pois as execuções são independentes. Além disso, como não existe troca de mensagens entre os processadores, o poder de processamento de cada processador é usado apenas para executar simulações.

3.3.9. Esquema de funcionamento do simulador

A figura 3.3.2 apresenta um diagrama esquemático do funcionamento do simulador paralelo.

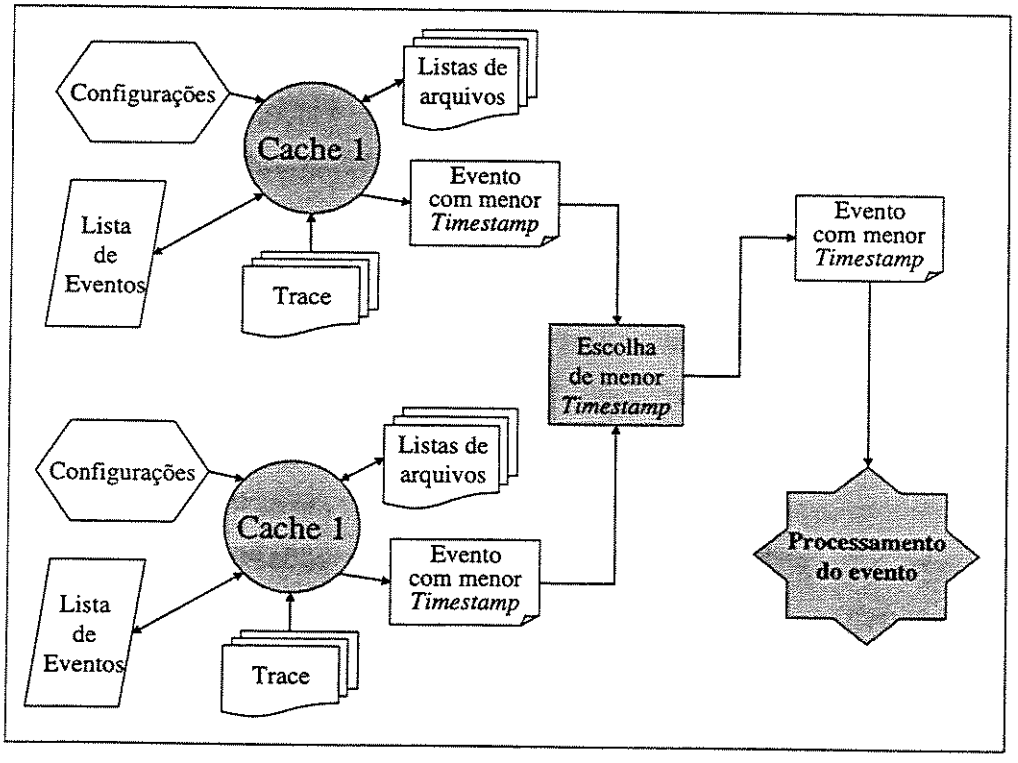


Fig. 3.3.2 – Esquema de funcionamento do simulador

Como pode ser visto na figura 3.3.2, cada cache possui uma lista de eventos que devem ser processados. É escolhido o evento com o menor *timestamp* dentre todos os caches. Esse evento é então processado, escolhendo-se novamente aquele de menor *timestamp* e assim até o processamento de todos os eventos, ou seja, quando todas as filas de eventos estiverem vazias. Dessa forma, a ordem de processamento dos eventos é a mesma que no sistema real simulado.

Ao final, quando todos os eventos tiverem sido processados, é gerado o arquivo de resultados da simulação.

3.4. Simulações de redes de caches

Foram realizadas simulações de forma a comparar o desempenho de uma rede de caches hierárquicos em relação a uma rede sem hierarquia, onde os caches são colocados em linha. Foram utilizados caches idênticos de 100 Mbytes, não particionados, para compor todas as malhas simuladas. Foi utilizada a política LRU em todos os caches, sendo processadas cerca de 10 milhões de requisições, conseguidas em [NLANR00]. Apesar de 100 Mbytes ser um tamanho pequeno em relação à maioria dos caches reais, esse tamanho foi escolhido devido ao número reduzido de requisições utilizadas, sendo que esse número de requisições foi escolhido de forma a diminuir o tempo total da simulação.

Para a simulação dos caches em linha, foram feitas as simulações de um a oito caches cooperativos colocados no nível 1. A estrutura genérica da malha simulada pode ser vista na figura 3.4.1.

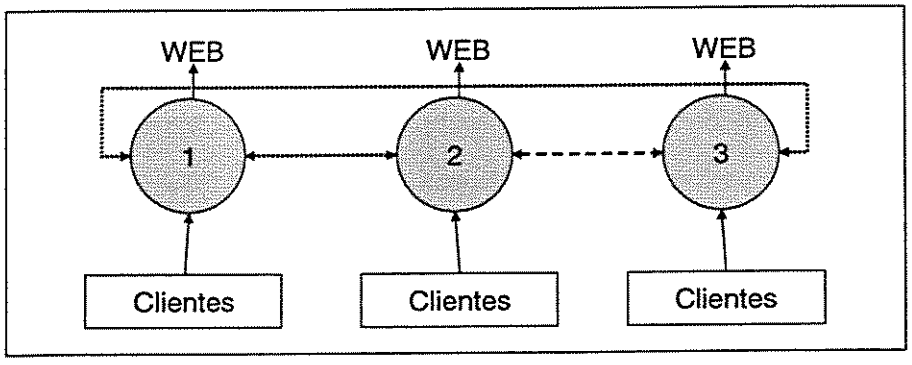


Fig. 3.4.1 - Estrutura genérica das malhas não-hierárquicas simuladas

Os resultados da simulação das malhas não-hierárquicas são apresentados na tabela 3.4.1 a seguir. O número de *hops* indica a quantidade de transmissões da requisição, contadas a partir do cliente. Um acerto com 1 *hop* indica que o cache que recebeu a requisição continha em seu cache o arquivo enquanto que um acerto com 2 *hops* indica que a requisição foi satisfeita por vizinhos do primeiro cache a receber a requisição. O percentual de acertos foi dividido entre os números de *hops* relativos aos acertos, pois um acerto ocorrido com menor número de *hops* indica uma resposta mais rápida ao usuário.

	Percentual de acertos por *hop*	
Número de caches	1 *hop*	2 *hops*
1	22.59	----
2	22.36	2.25
3	20.37	7.36
4	20.14	9.72
5	20.33	13.47
6	20.49	17.60
7	20.77	22.38
8	20.43	24.47

Tabela 3.4.1 - Resultados da simulação de malhas não-hierárquicas

Para a simulação dos caches hierárquicos, foram feitas simulações de malhas formadas por um a oito caches, sendo que um dos caches foi colocado no nível 2 e o restante foi colocado no nível 1. Os caches de nível 1 foram configurados para agir cooperativamente e enviar requisições não satisfeitas ao cache de nível 2. A estrutura genérica da malha hierárquica simulada pode ser vista na figura 3.4.2.

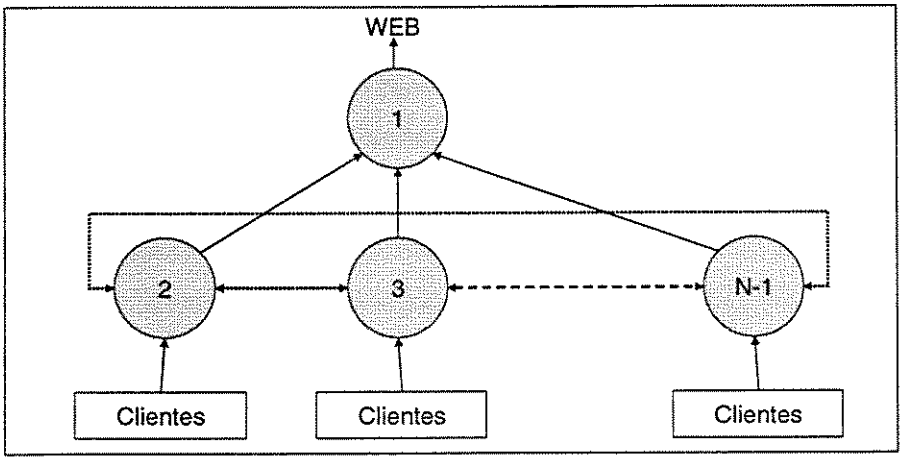


Fig. 3.4.2 - Estrutura genérica das malhas hierárquicas simuladas

Os resultados da simulação das malhas hierárquicas são apresentados na tabela 3.4.2 a seguir. Assim como na simulação anterior, o número de *hops* indica a quantidade de transmissões da requisição, contadas a partir do cliente. Um acerto com 3 *hops* indica que apenas o cache pai do cache que recebeu a requisição continha o arquivo.

	Percentual de acertos por *hop*		
Número de caches	1 *hop*	2 *hops*	3 *hops*
1	22.59	----	----
2	22.59	0.34	----
3	22.36	2.25	0.24
4	20.37	7.36	0.17
5	20.14	9.72	0.16
6	20.33	12.47	0.13
7	20.49	17.60	0.11
8	20.77	22.38	0.11

Tabela 3.4.2 - Resultados da simulação das malhas hierárquicas

A figura 3.4.3 mostra o gráfico dos percentuais de requisições satisfeitas com 2 *hops*, apresentados pelas malhas hierárquicas e não-hierárquicas de caches cooperativos em função do número de caches na malha. É mostrado também, o gráfico dos resultados da simulação de um único cache com o tamanho igual à soma dos tamanhos de todos os caches de uma das malhas simuladas. Esses resultados representam, nas condições da

simulação, o melhor desempenho possível para a capacidade de armazenamento disponível.

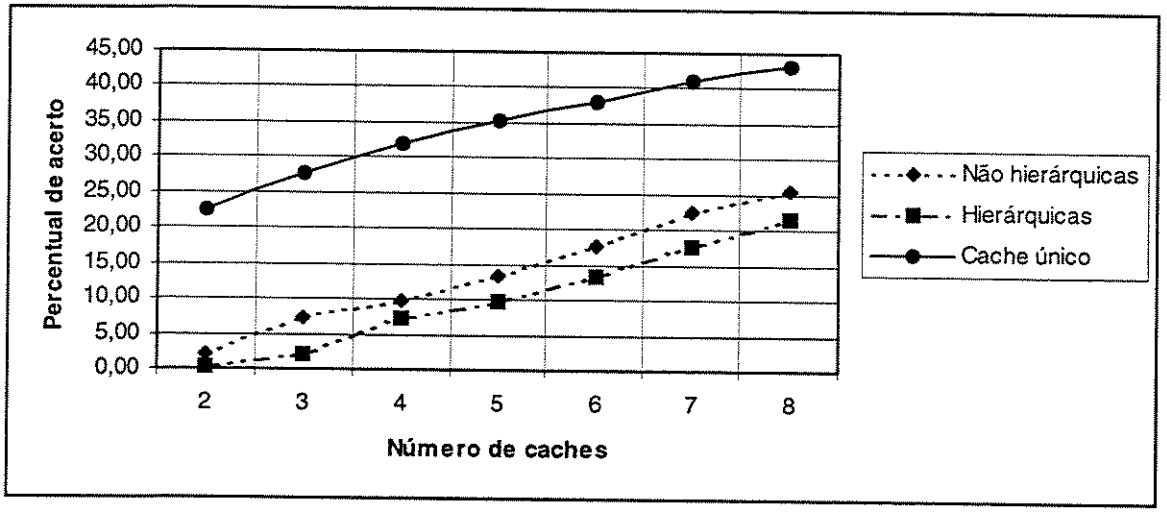


Fig. 3.4.3 - Gráfico comparativo da taxa de acerto

Através do gráfico da figura 3.4.3, pode-se observar que as malhas não-hierárquicas apresentaram um melhor desempenho que as malhas hierárquicas. Porém, nessa simulação foi considerado um tempo de transmissão igual para todos os meios de transmissão que ligam caches vizinhos. Esse fato não acontece na prática em redes de caches cooperativos geograficamente dispersos, onde os tempos de transmissão são diferentes e altos, em comparação com redes locais. Isso força à configuração das malhas de uma maneira hierárquica como forma de diminuir tanto o número de caches consultados em uma única requisição quanto a carga nos meios de transmissão. Porém, apesar da estruturação hierárquica apresentar piores resultados, a semelhança no comportamento das duas curvas no gráfico da figura 3.4.3 mostra que é possível utilizar malhas hierárquicas com resultados compatíveis aos resultados apresentados pela utilização de malhas não hierárquicas. Com relação ao tamanho dos caches, é conhecido atualmente que o desempenho cresce logaritmicamente em função do espaço disponível para cache. Isso significa que as curvas da figura 3.4.3 manteriam seu formato para caches maiores. A limitação para o crescimento logaritmo encontra-se apenas em características do *trace*, pois podem existir *traces* os quais mesmo aumentando o tamanho do cache, não é possível conseguir melhora no desempenho.

3.5. Simulação da RNP brasileira

Para demonstrar a utilização do simulador, foram feitas simulações baseadas da malha da RNP (Rede Nacional de Pesquisa) brasileira [RNP01]. Foi utilizado na simulação o *backbone* RNP2, que começou a ser implantado em julho de 2000. Os Pontos de Presença (PoPs) que concentram maior fluxo de tráfego de dados utilizam conexões com tecnologia ATM. A tecnologia *Frame Relay* foi utilizada como solução de melhor relação custo/benefício nos PoPs que apresentam menor taxa de tráfego de dados.

O RNP2 possui atualmente quatro conexões internacionais de 2 Mbps e 23 Pontos de Presença (PoPs) instalados nas principais cidades do país. A velocidade das Portas de Acesso dos PoPs, de até 155 Mbps, garante o atendimento da soma das diversas conexões virtuais estabelecidas (VP) e permite a elevação da largura de banda dessas conexões na medida em que a demanda justificar esse aumento de velocidade. A malha da RNP brasileira, incluindo os PoPs é apresentada na figura 3.5.1.

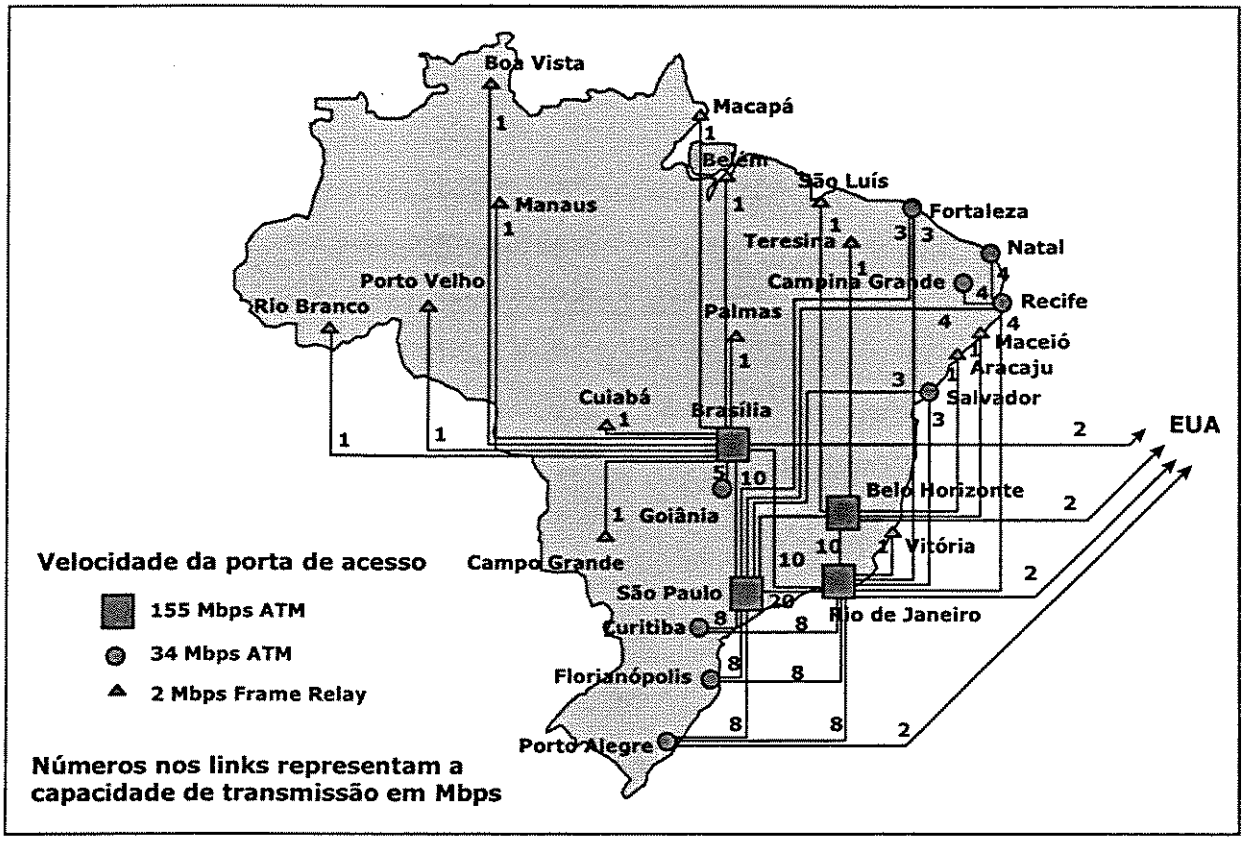


Fig. 3.5.1 - Malha da RNP brasileira

Nas simulações foram utilizadas as características relativas à arquitetura e capacidades da RNP2. A lista das cidades cujos caches pertencem à malha simulada é apresentada na tabela 3.5.1. São apresentados também os nomes das localidades onde estão situados os caches de nível hierárquico superior de cada cache bem como a capacidade de transmissão da rede que interliga-os.

Cache	Cache superior	Link ao cache superior
São Paulo	Rio de Janeiro	20 Mbps
Florianópolis	São Paulo, Rio de Janeiro	16 Mbps
Curitiba	São Paulo, Rio de Janeiro	16 Mbps
Recife	São Paulo, Rio de Janeiro	8 Mbps
Salvador	São Paulo, Rio de Janeiro	6 Mbps
Fortaleza	São Paulo, Rio de Janeiro	6 Mbps
Goiânia	Brasília	5 Mbps
Natal	Recife	4 Mbps
Campina Grande	Recife	4 Mbps
Brasília	USA	2 Mbps
Rio de Janeiro	USA	2 Mbps
Belo Horizonte	USA	2 Mbps
Porto Alegre	USA	2 Mbps
Belém	Brasília	1 Mbps
Manaus	Brasília	1 Mbps
Campo Grande	Brasília	1 Mbps
Cuiabá	Brasília	1 Mbps
Vitória	Rio de Janeiro	1 Mbps
São Luiz	Belo Horizonte	1 Mbps
Teresina	Belo Horizonte	1 Mbps
Maceió	Belo Horizonte	1 Mbps
Aracaju	Belo Horizonte	1 Mbps
Rio Branco	Brasília	1 Mbps
Porto Velho	Brasília	1 Mbps
Boa Vista	Brasília	1 Mbps
Macapá	Brasília	1 Mbps

Cache	Cache superior	Link ao cache superior
Palmas	Brasília	1 Mbps

Tabela 3.5.1 - Pontos de presença simulados.

Um problema muito importante é qual *arquivo de log* utilizar nas simulações. Devido à impossibilidade de obter *arquivos de log* reais de cada servidor a ser simulado, foram utilizados os *arquivos de log* provenientes de servidores de cache do sistema IRCACHE [NLANR00] devido à grande disponibilidade de arquivos. Esses arquivos de log foram utilizados em vários trabalhos de pesquisa relacionados a caches para Web, apresentando por isso uma boa referência na análise dos resultados obtidos.

Apesar dos *arquivos de log* do sistema IRCACHE não representarem com absoluta precisão as requisições feitas a servidores brasileiros, as características dos documentos requisitados, tais como tamanho e freqüência, são semelhantes. Assim, pode-se usar os arquivos do sistema IRCACHE sem alterar as conclusões obtidas nas simulações. *Os arquivos de log* utilizados nas simulações dessa seção são relativos a dez dias de uso dos servidores de cache.

A análise dos *arquivos de log* utilizados mostrou que um cache de tamanho infinito conseguiria um máximo de 67% de *Hit Ratio*. Para os *arquivos de log* utilizados, qualquer cache de capacidade maior ou igual a 42 GigaBytes pode ser considerado de tamanho infinito.

O primeiro experimento analisou o impacto da variação do tamanho do espaço de armazenamento do cache no desempenho. Foi também avaliado o impacto no desempenho da variação da política de substituição utilizada. Os resultados são apresentados na figura 3.5.2. Esses resultados foram medidos em *Hit Ratio* e representam o desempenho de toda a malha da RNP2.

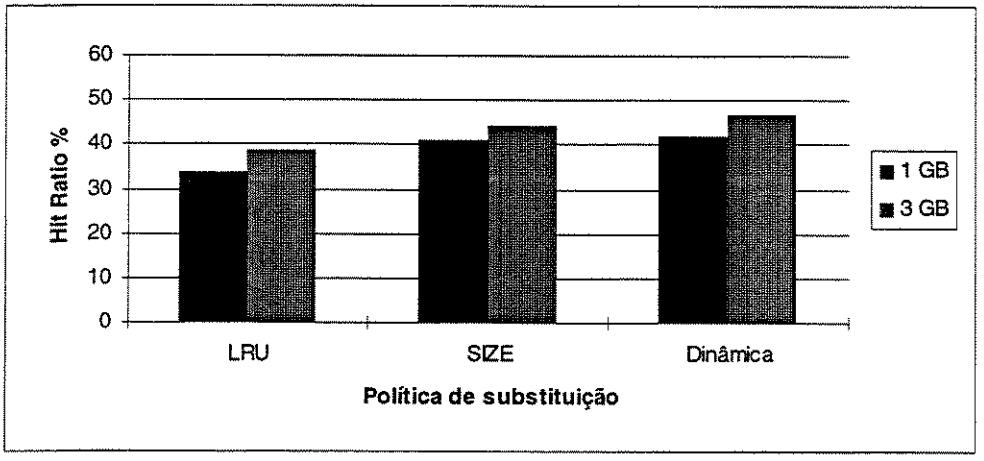


Fig. 3.5.2 - Variação de desempenho relativo a tamanho e política

Através da figura 3.5.2, pode-se notar que apesar do aumento do espaço de armazenamento gerar uma melhora no desempenho do cache, a utilização de uma política de substituição melhor causa uma melhora no desempenho mais significativa. Nessa simulação, verificou-se que o tempo médio de espera por um documento foi de 0.044 unidades de tempo.

O segundo experimento testou a possibilidade em que houvesse a cooperação entre todos os servidores de cache. Os resultados dessa simulação são apresentados na figura 3.5.3.

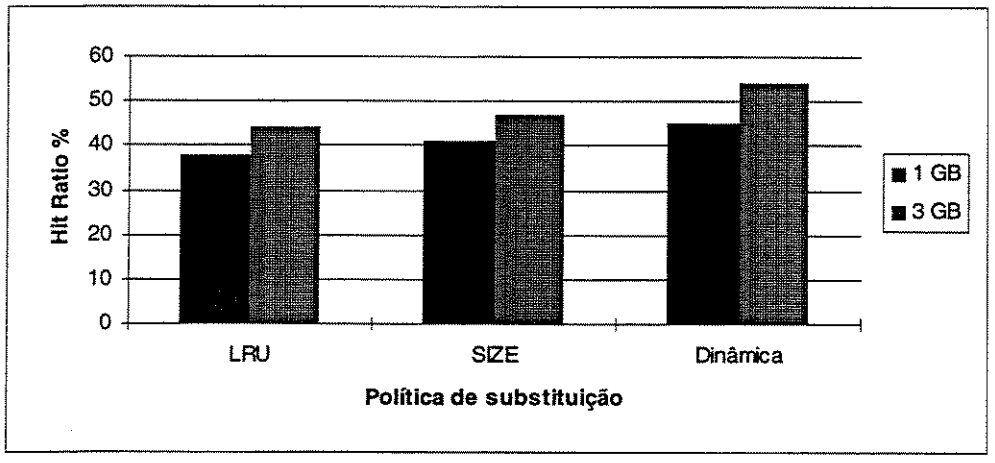


Fig. 3.5.3 - Cooperação entre todos os caches

Através da figura 3.5.3 observa-se que o desempenho do sistema com cooperação entre todos os caches foi melhor que o do sistema sem cooperação nenhuma. Porém,

houve um aumento de 980% no número de mensagens trocadas entre servidores de cache. Esse grande aumento no número de mensagens trocadas faz com que o usuário experimente um alto tempo de espera, pois nesse caso, o usuário terá que esperar a troca de mensagens entre todos os servidores, inclusive entre aqueles ligados por redes de baixa capacidade. Nessa simulação, o tempo de espera médio foi de 1,19 unidades de tempo, que é um tempo impraticável em redes reais. Assim, pode-se concluir que utilizar cooperação entre caches ligados por redes de baixa capacidade de transmissão não é interessante.

O último experimento consistiu em medir o desempenho quando utilizada a cooperação apenas entre alguns dos caches do sistema. Foi então definida a cooperação entre os caches cuja localização é apresentada na tabela 3.5.2. Note que a cooperação foi definida apenas entre caches ligados por redes de alta capacidade.

Cache	Coopera com
São Paulo	Brasília, Belo Horizonte e Rio de Janeiro
Rio de Janeiro	Belo Horizonte e São Paulo
Belo Horizonte	Rio de Janeiro e São Paulo
Brasília	Rio de Janeiro e São Paulo

Tabela 3.5.2 - Alguns caches em cooperação

O desempenho total da malha da RNP2, com cooperação apenas entre os caches ligados por redes de maior capacidade é apresentado na figura 3.5.4.

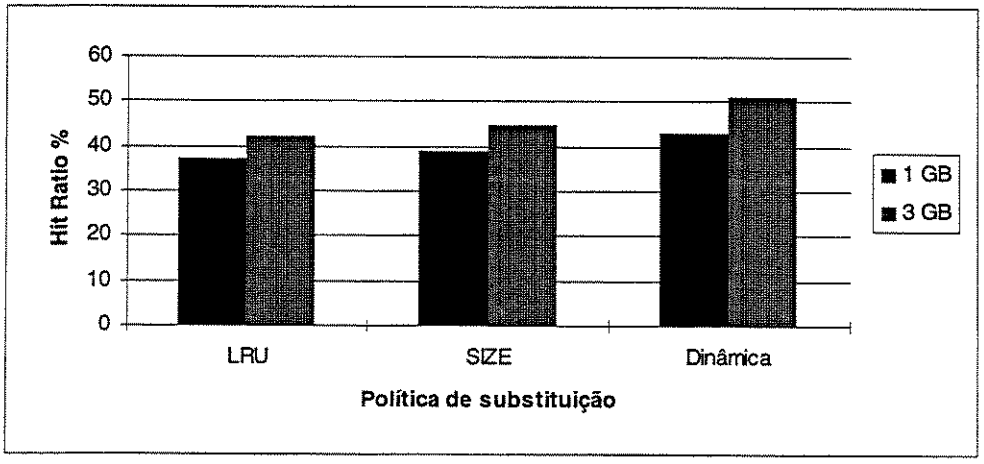


Fig. 3.5.4 - Simulação com caches cooperativos.

A utilização da cooperação apenas entre caches ligados por redes de alto desempenho causou um aumento de 181% no número de mensagens trocadas entre os caches, em relação à primeira simulação. Por outro lado, houve uma diminuição do tempo de espera do usuário de 37%. Isso aconteceu porque o aumento do desempenho em *Hit Ratio* compensou o aumento do número de mensagens. Assim, conclui-se que a cooperação entre os caches melhorou o desempenho do sistema como um todo. Porém, note que não é garantido que os resultados apresentados na figura 3.5.4 são os melhores possíveis. Para determinar qual configuração maximiza o desempenho do cache são necessárias muitas outras simulações, testando muitas outras possibilidades.

Apesar dos caches simulados serem de pequeno tamanho quando comparados com caches reais de alto desempenho, cujo tamanho do espaço de armazenamento chega a centenas de GigaBytes, os resultados obtidos são válidos. Isso ocorre porque tanto o *hit ratio* quanto o *byte hit ratio* de um cache para Web crescem logaritmicamente em função do tamanho do cache [Jin00]. Assim, salvo as devidas proporções as conclusões são as mesmas para caches maiores. Além disso, a análise de resultados para caches pequenos tem sido considerada com grande relevância porque esse tamanho de cache pode ser colocado na memória RAM de servidores. Dessa forma, esses servidores podem responder a requisições sem fazer nenhum acesso a dispositivos de memória secundária [Cao97].

As simulações foram executadas utilizando um computador paralelo *Beowulf 64*, formado por 64 computadores Intel, cada um com 256 Mega Bytes de memória RAM,

rodando o sistema operacional Linux Red Hat. Esse computador pertence ao Laboratório de Alto Desempenho do Instituto de Computação da Unicamp. A utilização de um simulador capaz de aproveitar o poder de processamento de um computador paralelo possibilitou o processamento de quantidades de dados que seriam praticamente impossíveis de serem analisados em um computador monoprocessado.

3.6. Conclusões deste capítulo

A maneira escolhida para analisar os sistemas de redes de caches possivelmente hierárquicos e cooperativos foi a simulação computacional das redes de caches. O projeto e o desenvolvimento de um simulador específico foi necessário por não existirem simuladores disponíveis que satisfizessem todos os requisitos necessários, principalmente o requisito de ser capaz de utilizar a capacidade de processamento de computadores paralelos.

O simulador paralelo apresentado neste capítulo mostrou ser uma ferramenta de grande importância para a análise de malhas de caches distribuídos. Usando esse simulador, é possível testar várias configurações dos parâmetros que influem no desempenho de sistemas de caches para Web. Esse simulador é capaz de utilizar o alto poder de processamento de computadores paralelos, possibilitando a execução de um grande número de simulações. Esse aumento do número de simulações gera uma maior quantidade de resultados, o que permite uma análise mais apurada dos sistemas simulados.

Uma conclusão obtida com a pesquisa descrita neste capítulo foi que, conforme descrito em trabalhos anteriores, a utilização de caches cooperativos possibilita um melhor aproveitamento dos recursos dos meios de transmissão, possibilitando um menor tempo de espera aos usuários dos serviços da Web. Devido às baixas taxas de transmissão apresentadas pelos meios de transmissão que interligam os caches geograficamente dispersos, a configuração de hierarquias dentro das malhas de caches cooperativos permite uma melhor utilização dos meios de transmissão entre caches geograficamente dispersos, apresentando desempenhos compatíveis aos de malhas não hierárquicas.

Além disso, o trabalho de pesquisa descrito nesse capítulo permitiu chegar a outra conclusão interessante: a simulação de variações de parâmetros de configuração em redes de caches deve ser realizada de forma a buscar as melhores opções de utilização de estratégias tais como a instalação de caches e a definição de hierarquias e regras de cooperação, de forma a buscar o aumento do desempenho das redes.

Capítulo 4

Variação da QoS oferecida por caches para Web

4.1. Introdução

Muitos trabalhos publicados propuseram diferentes estratégias de gerenciamento do espaço disponível para cache de arquivos. A maioria desses trabalhos apresenta novas políticas de substituição, com o objetivo de maximizar o desempenho dos caches em uma ou mais métricas específicas. Alguns trabalhos propõem um gerenciamento mais inteligente do espaço disponível. Um exemplo é o modelo PART, proposto em [Murta98], que propõe o particionamento do espaço disponível. Cada divisão do espaço é então destinado a armazenar uma classe diferente de arquivos. Essas classes podem ser definidas, por exemplo, segundo critérios de tamanho e tipo dos arquivos.

Entretanto, muitas vezes deseja-se que um cache privilegie um grupo de arquivos ou servidores. O cache é um bom lugar para implantar estratégias que permitam oferecer diferentes QoS [Kelly99]. Isso porque ele fica estrategicamente instalado no fluxo de dados e pode ser utilizado como mediador entre os diversos servidores que buscam sempre maximizar a qualidade dos serviços oferecida aos seus usuários [Chan99]

Esse trabalho considera que a QoS de um sistema de caches para Web pode ser medida através da medida do tempo economizado aos usuários do sistema ao realizarem requisições. Assim, considera-se que estratégias que, em média, ofereçam tempos de resposta menores apresentam melhores QoS.

Para privilegiar servidores ou conjuntos de arquivos específicos, o cache pode utilizar um valor de utilidade fornecido pelos servidores para estimar a utilidade futura dos arquivos e decidir quais remover do cache [Kelly99]. Do ponto de vista do usuário, é importante que o cache proporcione uma diminuição no tempo de resposta a requisições feitas [Afonso98]. Para que isso aconteça é necessário que a política de substituição implementada consiga prever quais documentos serão acessados posteriormente [Kelly99a]

Porém, as características da demanda de QoS dos serviços prestados pela Web não são estáticas. No caso de caches para Web, elas dependem de diversos fatores tais como o número de clientes, a capacidade de transmissão das redes, diferenças entre perfis de usuários, a hora e dia da semana, a época do ano e a variação da popularidade dos arquivos requisitados. Assim, o desempenho de uma política escolhida inicialmente pode diminuir drasticamente devido à incapacidade da política em se adaptar a novas características exigidas para um bom desempenho. Por isso, é necessário que a estratégia de gerenciamento do espaço para caching consiga se adaptar às variações de características da demanda por documentos.

Com esse objetivo, este trabalho apresenta e propõe a utilização do Gerenciador Dinâmico de Espaço (GDE). Essa estratégia pode alternar dinamicamente os parâmetros utilizados para estimar a utilidade futura dos arquivos armazenados. Assim, ela permite variar dinamicamente o perfil dos arquivos com maior preferência de serem mantidos no cache. Dessa forma, é possível direcionar o trabalho do cache para priorizar a manutenção de arquivos com características específicas desejadas, podendo assim variar a QoS oferecida aos clientes do sistema.

4.2. O Gerenciador Dinâmico de Espaço (GDE)

Os primeiros estudos sobre o fluxo de documentos verificaram que ele obedece à uma distribuição de cauda pesada e que as transferências apresentam auto-similaridade quando a demanda por documentos é alta [Crovella95]. A auto-similaridade *(self-similarity)* é uma propriedade associada a fractais, onde o objeto apresenta a mesma forma, não importando a escala na qual é visto. Estudos posteriores mostraram que o fluxo de documentos obedece à lei de Zipf [Breslau99].

A lei de Zipf diz que a probabilidade relativa de acontecer uma requisição ao *i-ésimo* documento mais popular é proporcional a $^{1}/_{i}$. Assim, a freqüência com que um documento foi acessado no passado tem fortes implicações na probabilidade de ser acessado novamente no futuro. Para retirar do cache documentos com baixa freqüência de acessos pode ser usada a política LFU.

O tempo desde o último acesso ao documento também é um fator muito importante no cálculo da probabilidade de um documento do cache ser acessado novamente. Documentos acessados mais recentemente têm maiores probabilidades de serem acessados novamente. Uma forma de manter no cache documentos acessados recentemente é usar a política LRU, que remove documentos antigos do cache.

O tamanho dos documentos armazenados é também importante no desempenho do cache. A remoção de documentos grandes evita que o cache seja preenchido por poucos documentos grandes ao invés de muitos documentos de tamanhos médios e pequenos, o que diminui o desempenho do cache em *Hit Ratio.* A remoção dos maiores documentos é implementada pela política SIZE.

A idéia do GDE tanto para conseguir um alto desempenho na métrica desejada e prover diferentes QoS é usar vários diferentes critérios para ordenar filas de prioridade de exclusão dos arquivos no cache. A estratégia de usar mais de uma ordenação para possibilitar uma melhor tomada de decisão na remoção de arquivos do cache foi estudada nesse projeto e gerou a Política Dinâmica de Substituição (PDS), apresentada nas seções e 3.3.3 e 3.3.5.

Apesar de apresentar desempenho em *hit ratio* e byte *hit ratio* no mínimo iguais às demais políticas, a PDS apresentava uma maior exigência por memória e capacidade de processamento do que as demais políticas. Considerando que as melhores políticas, incluindo a PDS, apresentavam desempenho semelhante o maior consumo de memória e processamento da PDS faziam que, muitas vezes, não fosse justificada a sua utilização, principalmente em caches que recebiam altas taxas de requisições. Essa conclusão fez com que fosse imaginada outra utilização para sistemas com múltiplas filas de documentos. Foi desenvolvido então o GDE, que usa múltiplas filas para prover diferentes QoS aos usuários da Web.

No primeiro exemplo de utilização apresentado, o GDE utiliza os fatores de freqüência de acesso, tempo desde o último acesso e tamanho dos documentos para decidir quais documentos devem ser removidos do cache. Isso é feito através da escolha dinâmica de uma política a ser aplicada aos documentos no cache, dentre as políticas LRU, SIZE e LFU. Dessa forma, nesse caso, as políticas LRU, SIZE e LFU são chamadas políticas básicas do Gerenciador Dinâmico de Espaço. A cada política básica é associado um peso. O peso da política indica qual a sua importância na escolha do documento a ser removido.

O GDE utiliza uma fila de documentos com várias ordenações. Cada ordenação é feita através de uma política básica. A figura 4.2.1 apresenta um exemplo onde a fila de documentos possui três ordenações.

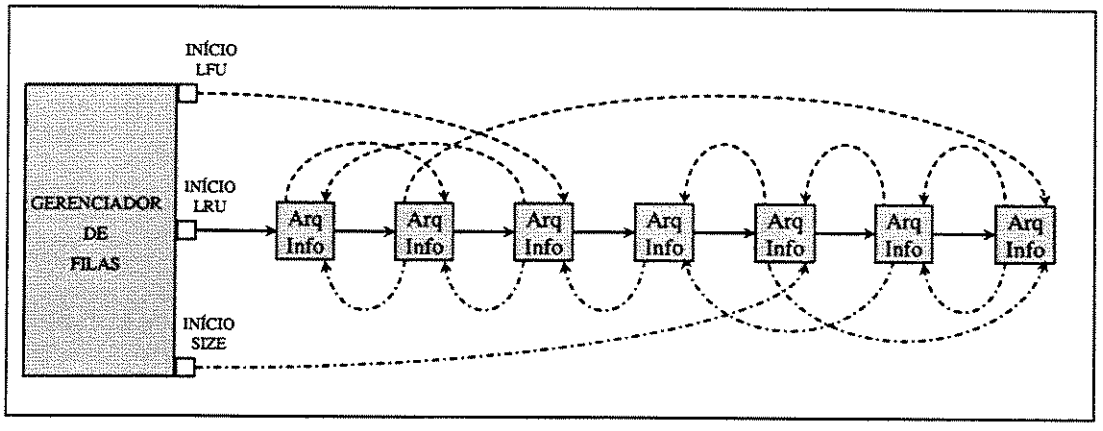


Fig. 4.2.1 - Exemplo de ordenação de documentos

Quando o GDE precisa ser aplicado ao conjunto de documentos armazenados no cache, cada política básica escolhe o documento considerado mais descartável. Serão escolhidos então 3 documentos, não necessariamente distintos. É verificada então a utilidade de cada documento escolhido para cada política básica. A utilidade depende da posição relativa de cada documento em cada ordenação. Usando os pesos de cada política básica, o GDE então escolhe um dos documentos para ser removido. O documento escolhido é aquele cuja soma das utilidades dentre as políticas é a menor.

A tabela 4.2.1 apresenta um exemplo do cálculo da utilidade dos documentos. Considere que o documento LRUChoice é o escolhido pela política LRU para ser

removido. Da mesma forma, a política LFU escolheu o documento LFUChoice e a política SIZE escolheu SIZEChoice. Os valores LRUValue, LFUValue e SIZEValue são os valores de utilidade dos documentos escolhidos para cada política.

	LRUChoice	LFUChoice	SIZEChoice
LRUValue	0.2	0.4	0.2
LFUValue	0.8	0.1	0.4
SIZEValue	0.1	0.3	0.1
Total	1.1	0.8	0.7

Tabela 4.2.1- Exemplo de valores de utilidade

No exemplo da figura 4.2.1, onde os pesos foram colocados apenas como possíveis exemplos, considerando iguais os pesos das políticas básicas, o documento escolhido pela política SIZE, SIZEChoice, seria removido do cache.

A variação dos pesos de cada política básica permite melhorar o desempenho do cache na métrica mais importante para cada cache específico. O cache pode, por exemplo, ser programado para garantir um *Byte Hit Ratio* específico. Durante a operação do cache, se é detectada uma diminuição do *Byte Hit Ratio*, o GDE aumenta o peso da política LRU, que apresenta altas taxas de *Byte Hit Ratio*. Por outro lado, se o cache for programado para apresentar taxas de *Hit Ratio* maiores que um certo limite, ao detectar a queda de *Hit Ratio*, o GDE aumenta o peso da política SIZE, que apresenta melhores resultados em *Hit Ratio*.

4.3. Avaliação de Desempenho

Antes de explorar as características de adaptação dinâmica do GDE, é importante mostrar que essa política permite obter um desempenho nas métricas tradicionais (*hit ratio e byte hit ratio*) compatíveis com as melhores políticas existentes.

A avaliação do desempenho do GDE foi feita utilizando simulação baseada em *traces*. Os *traces* utilizados são provenientes da DEC [DEC00] e NLANR[NLANR00]. Da NLANR foram utilizados os *traces* UC e RTP. Todos os *traces* foram pré-processados para retirar as requisições a documentos não "cacheáveis" tais como páginas

dinâmicas, que representavam menos de 5% das requisições. Na análise foram utilizados *traces* de uma semana da DEC e duas semanas da NLANR.

O trace DEC contém cerca de 700 mil requisições, e o número total de bytes requisitados foi de cerca de 8 GB. O trace RTP contém 3 milhões de requisições, a arquivos cuja soma é aproximadamente 18 GB. O trace UC é formado por 1 milhão de requisições a aproximadamente 6 GB.

Os testes de desempenho foram realizados comparando o GDE com as políticas GDS, GD*, LRU e LRU-DA. As políticas GDS e GD* foram escolhidas devido ao seu alto desempenho. A política LRU foi escolhida por ser largamente conhecida enquanto que a LFU-DA é um exemplo de política que privilegia arquivos com grande freqüência de acesso.

Os resultados do desempenho são apresentados nas figuras 4.3.1a, 4.3.1b e 4.3.1c. Os valores no eixo X representam os tamanhos dos caches analisados, em GigaBytes. Os pesos utilizados nas políticas básicas do GDE foram iguais. Os gráficos apresentam os resultados das políticas LRU, LFU-DA, GDS(1) e GD*(1) obtidos em [Jin00] usando os mesmos *traces* usados neste capítulo. Nesse caso, as políticas GDS e GD consideram fixo o custo de cada documento.

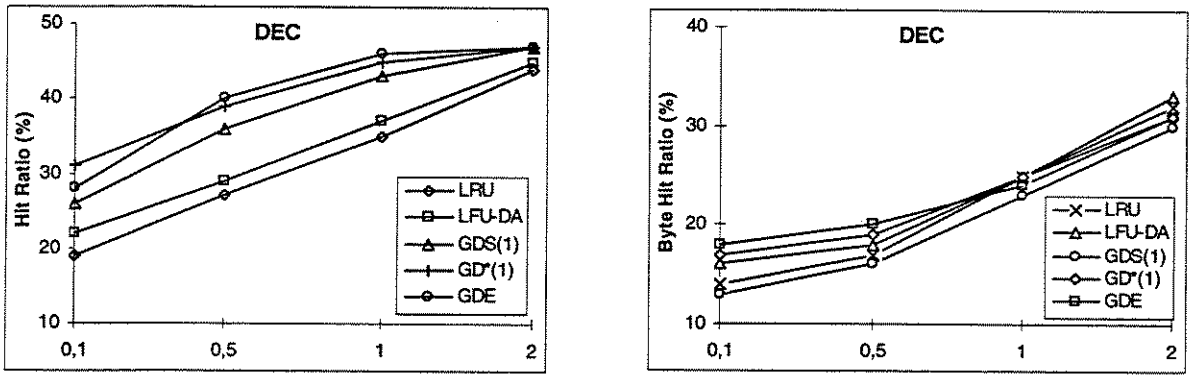
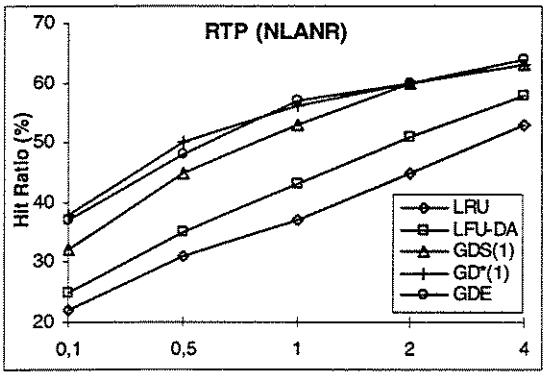


Fig. 4.3.1a - Políticas LRU, LFU-DA, GDS(1), GD*(1) e GDE – Trace DEC



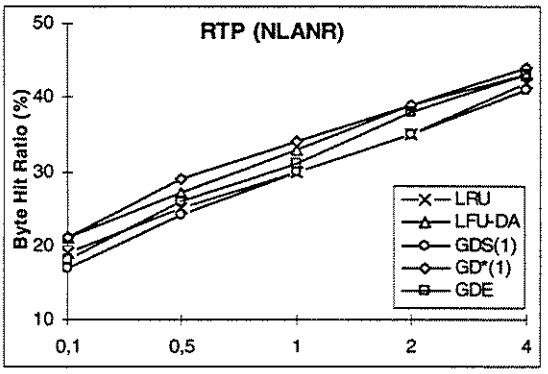
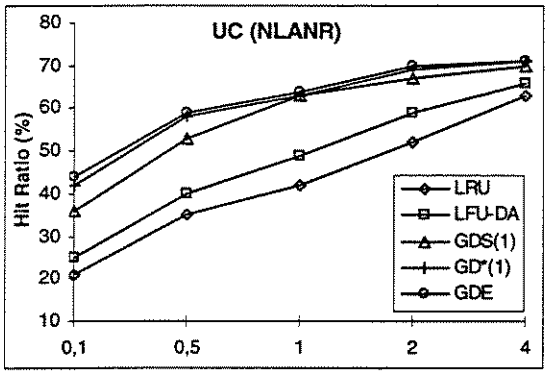


Fig. 4.3.1b - Políticas LRU, LFU-DA, GDS(1), GD*(1) e GDE – Trace RTP (NLANR)



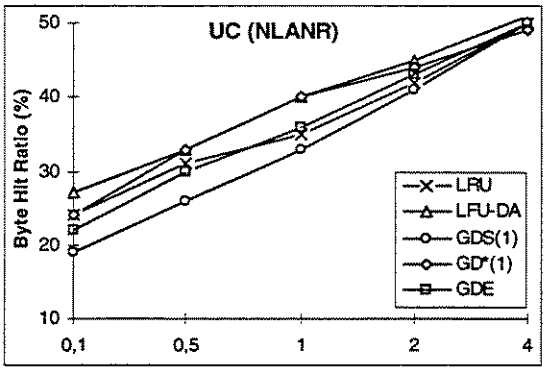
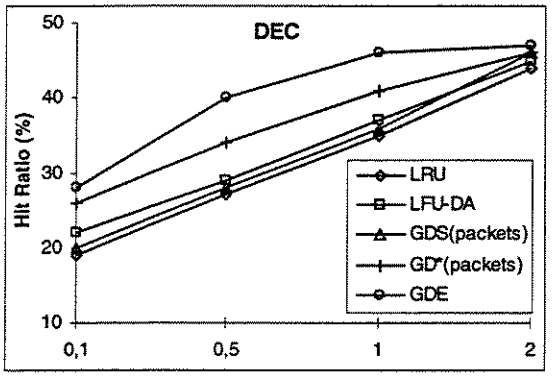


Fig. 4.3.1c - Políticas LRU, LFU-DA, GDS(1), GD*(1) e GDE – Trace UC (NLANR)

Os resultados do desempenho do GDE comparado às políticas LRU, LFU-DA, GDS(packets) e GD*(packets) é apresentado nas figuras 4.3.2a, 4.3.2b e 4.3.2c. Nesse caso, as políticas GDS e GD consideram o custo de cada documento igual ao número de pacotes transferidos no envio do documento.



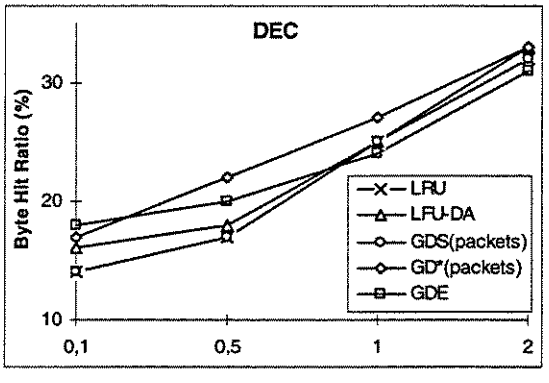
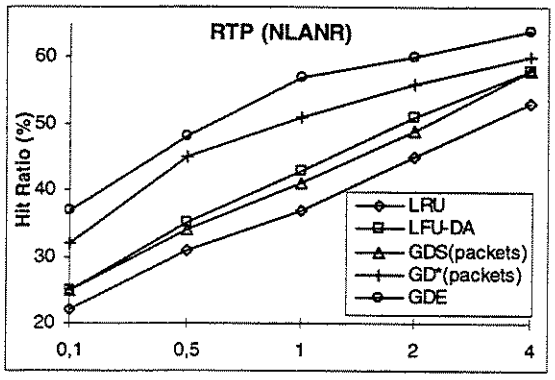


Fig. 4.3.2a - Políticas LRU, LFU-DA, GDS(packets), GD*(packets) e GDE – Trace DEC



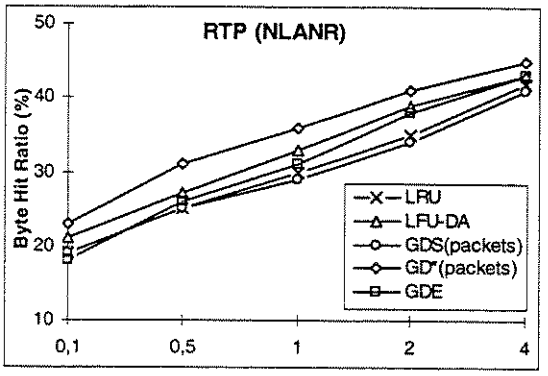
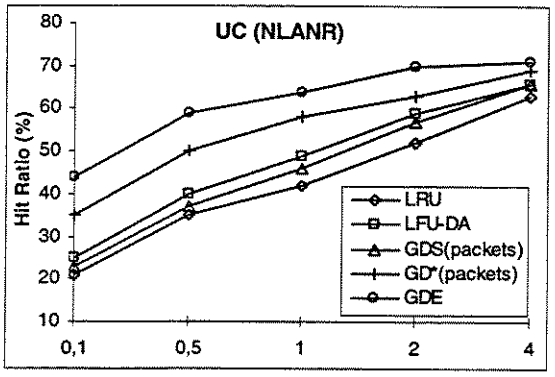


Fig. 4.3.2b - Políticas LRU, LFU-DA, GDS(packets), GD*(packets) e GDE – Trace RTP (NLANR)



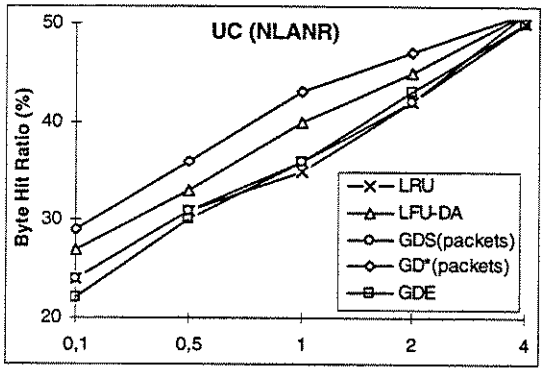
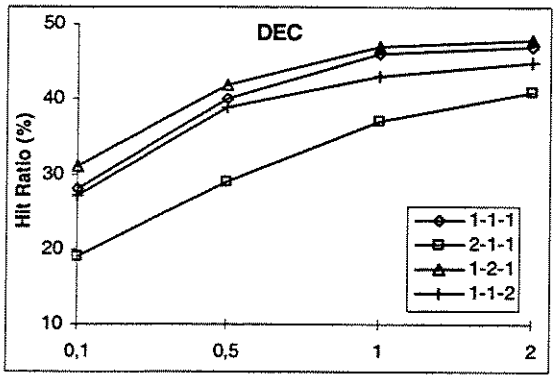


Fig. 4.3.2c - Políticas LRU, LFU-DA, GDS(packets), GD*(packets) e GDE – Trace UC (NLANR)

Foram realizadas simulações para verificar o impacto da variação dos pesos das políticas básicas no desempenho em *Hit Ratio* e *Byte Hit Ratio*. Os resultados das simulações são apresentados na figuras 4.3.3a, 4.3.3b e 4.3.3c. Os nomes nas legendas representam o peso das políticas LRU, SIZE e LFU.



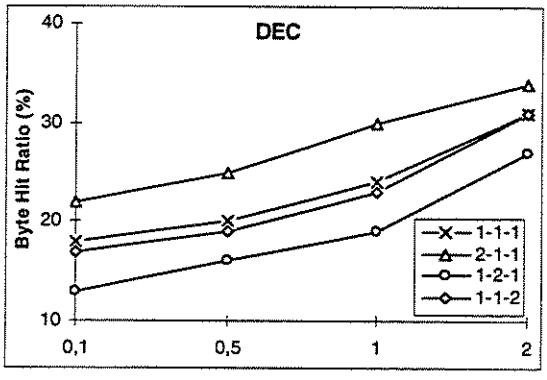
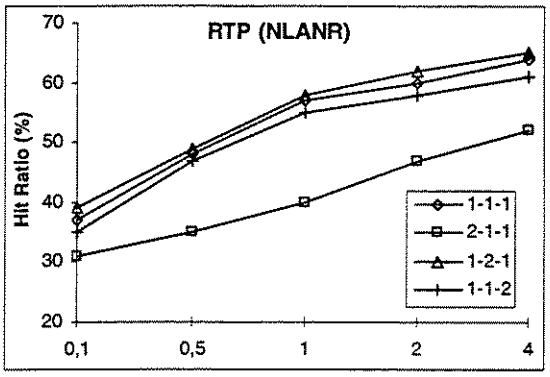


Fig. 4.3.3a - Desempenho do GDE com variação de pesos – Trace DEC



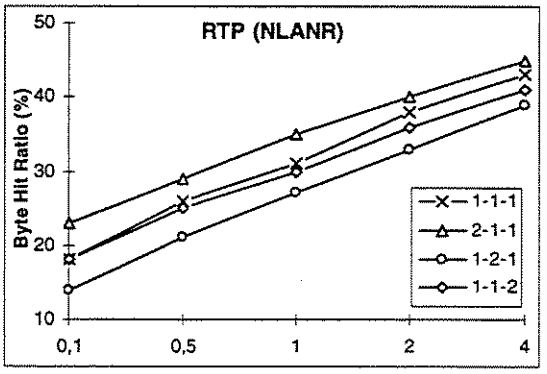
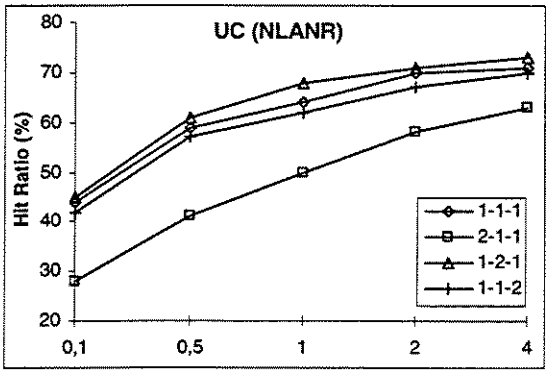


Fig. 4.3.3b - Desempenho do GDE com variação de pesos – Trace RTP (NLANR)



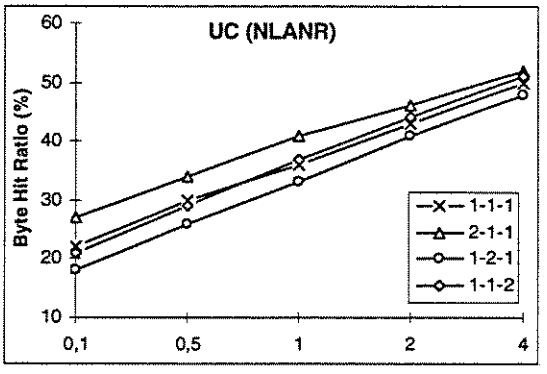


Fig. 4.3.3c - Desempenho do GDE com variação de pesos – Trace UC (NLANR)

Analisando os resultados das figuras 4.3.1a, 4.3.1b e 4.3.1c, pode-se notar que o GDE apresenta desempenho compatível com as melhores políticas analisadas. As políticas LRU e LFU-DA apresentam os piores resultados em *Hit Ratio*. Isso ocorre porque essas políticas não levam em consideração os tamanho dos documentos [Jin00]. A política GDS (1) apresentou os piores resultados em *Byte Hit Ratio*.

As figuras 4.3.2a, 4.3.2b e 4.3.2c mostraram que o GDE obteve os melhores resultados em *Hit Ratio*. Em *Byte Hit Ratio*, obteve desempenho próximo dos melhores resultados obtidos pelas demais políticas.

A variação de desempenho em *Hit Ratio* e *Byte Hit Ratio* pela alteração dos pesos das políticas básicas o GDE mostrou que é possível otimizá-la de acordo com a métrica desejada. Assim, é possível utilizar apenas uma política em diferentes casos de utilização de caches para Web. A variação de desempenho relativa à variação dos pesos das políticas básicas pode ser observada nas figuras 4.3.3a, 4.3.3b e 4.3.3c.

4.4. Aplicações do GDE

Nessa seção são apresentados exemplos de possíveis aplicações para o GDE. Essas aplicações tomam como princípio que são mantidas pelo GDE políticas capazes de favorecer os arquivos cujas QoS oferecidas deseja-se alterar. A alteração das características de favorecimento do cache aos arquivos é feita através da alteração dos pesos das políticas que interferem na probabilidade da manutenção desses arquivos no cache.

4.4.1. Favorecimento de arquivos em horas específicas

A popularidade dos arquivos requisitados varia com o tempo. Em alguns casos, arquivos têm sua popularidade aumentada muito rapidamente. Por exemplo, se um programa de televisão de grande audiência mencionar um Web site em sua programação, existe uma grande chance do número de acessos ao Web site mencionado aumentar muito. Muitas vezes, pode acontecer do aumento de popularidade daquele Web site ser tão grande a ponto de fazer com que o servidor responsável seja incapaz de responder a tantas requisições. Esse seria um caso em que o favorecimento do cache aos arquivos daquele Web site diminuiriam o número de requisições ao servidor.

Uma questão que pode surgir nesse ponto é quanto ao impacto real da utilização do GDE no favorecimento dos arquivos do Web site, pois se esses arquivos são tão populares a ponto de sobrecarregar o servidor, eles terão grande chance de serem mantidos em um cache que não utilize o GDE. Porém, no fluxo de requisições que passa pelo cache podem existir muitos arquivos muito mais populares que os arquivos do Web site que se deseja privilegiar.

A figura 4.4.1 apresenta um gráfico com os números de acessos aos arquivos do *trace* RTP, ordenados de acordo com sua popularidade. Quanto maior o número de acessos, mais popular é o arquivo e mais à esquerda esse arquivo é representado no gráfico. Os valores entre parênteses no eixo X representam a soma total dos tamanhos dos arquivos.

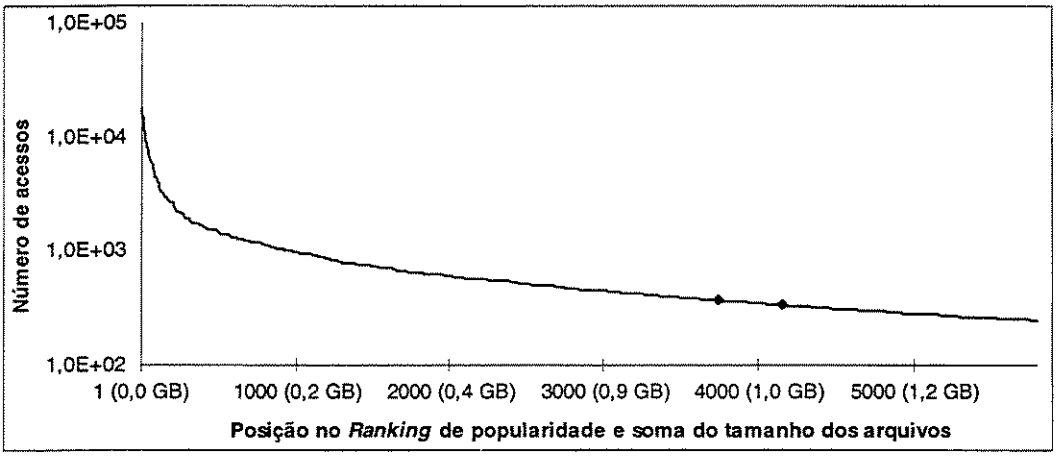


Fig. 4.4.1 –Distribuição de popularidade de arquivos

Um cache que não utilize o GDE terá uma grande probabilidade de favorecer apenas a manutenção dos arquivos mais populares. Por exemplo, na distribuição da figura 4.4.1, um cache de 0,4 GB terá uma grande probabilidade de manter no cache os 2000 arquivos mais populares. No caso de manter no cache arquivos com valores de popularidade entre os dois pontos destacados no gráfico, de forma a melhorar a QoS oferecida ao servidor desses arquivos, é necessário usar uma política que privilegie esses arquivos. Porém, essa política não pode ser usada em todo o tempo de operação do cache, pois prejudicaria outros arquivos em momentos que não é necessário favorecer os arquivos específicos. Nesse caso, a solução usando o GDE seria manter uma política que favorecesse os arquivos mais populares e outra que permitisse favorecer arquivos específicos. Na maioria do tempo, a primeira teria um grande peso na decisão de quais arquivos remover. Para privilegiar arquivos específicos, a segunda política deveria ter seu peso aumentado.

Dessa forma, é possível usar o GDE para garantir uma alta QoS a arquivos específicos, mesmo que eles não estejam dentre os mais populares.

4.4.2. Favorecimento de tipos de documentos

O GDE permite também favorecer tipos de arquivos. Por exemplo, pode-se instalar um cache cujo principal objetivo é diminuir o tempo de espera de usuários na busca por arquivos de música. Porém, o uso do GDE permite que o cache não seja

exclusivo de arquivos de música, podendo também ser usado para outros tipos de arquivos quando não for necessário privilegiar especificamente os arquivos de música. Isso pode ser feito variando-se os pesos de duas políticas básicas: uma que privilegie arquivos de música e outra que apresente um alto desempenho para arquivos em geral.

4.4.3. Favorecimento de usuários

Apesar de exigir uma troca de informações mais complexa, o cache pode ser instruído para favorecer arquivos requisitados por usuários específicos. Para tanto, é necessário que o cache mantenha uma tabela de prioridades de clientes, o que pode fazer com que o processo seja mais demorado. Além disso, é necessário que o cache mantenha dados que indiquem o cliente responsável pela requisição a cada arquivo armazenado, exigindo um maior consumo de memória e processamento.

O favorecimento de usuários permite que o cache concentre seus esforços em usuários especiais, mas sem deixar de permitir que usuários comuns utilizem-se da capacidade não utilizada pelos usuários favorecidos.

4.4.4. Variação dinâmica dos pesos das políticas básicas

A grande vantagem do GDE é permitir que seja possível utilizar tantas políticas de substituição quantas sejam necessárias, variando dinamicamente o peso de decisão de cada política para proporcionar a variação dinâmica da qualidade de serviço oferecida pelo cache a diferentes classes de clientes. Dessa forma, é possível programar o cache para favorecer determinados conjuntos de arquivos apenas em determinadas horas.

A variação automática dos pesos das políticas básicas do GDE pode também ser feita durante o decorrer do dia. A variação de desempenho em *Byte Hit Ratio* e *Hit Ratio* é importante devido às diferentes taxas de utilização da Web durante o dia. A análise do tráfego na Internet mostrou que existe um ciclo de 24 horas que apresenta tráfegos maiores em determinadas horas do dia [Morris00]. Considerando as diferenças de fusos horários pelo mundo, existirão momentos em que uma parte da Web mundial está sobrecarregada enquanto que sobra capacidade de transmissão em outras.

Por exemplo, quando são 4 da tarde na costa oeste dos EUA e os servidores estão sobrecarregados, será 1 da manhã em grande parte da Europa. Nesse momento, pode ser mais importante que os caches europeus economizem requisições aos servidores americanos do que a banda de transmissão dos *links* EUA-Europa. Esse seria então o momento de aumentar a taxa desejada de *Hit Ratio*. O GDE pode ser programada para variar os pesos das políticas básicas conforme a hora e conforme o desempenho do cache.

4.4.5. O GDE e o *push-caching*

Os caches para Web normalmente trabalham em um modelo onde as requisições dos clientes determinam o conteúdo do cache. No modelo alternativo chamado *push-caching*, os servidores replicam seu conteúdo no cache. Isso ajuda a prevenir que o servidor fique sobrecarregado de requisições e, desde que o servidor envie para o cache seus documentos mais populares, diminui o tempo de espera dos clientes e a carga na rede.

O GDE pode ser usado independentemente do *push-caching* pois a busca e armazenamento de arquivos podem ser programados para aceitar "recomendações" de servidores. Nesse caso, é necessário informar ao GDE que um novo arquivo foi incluído, sendo então necessário atribuir valores de utilidade futura para o arquivo em todas as ordenações utilizadas, para que o GDE decida corretamente em que posição o arquivo se encontra em todas as ordenações.

4.5. Aspectos de implementação do GDE

Um dos grandes limites dos sistemas de gerenciamento de espaço em caches para Web é a quantidade de recursos computacionais exigidos para a sua execução. No caso do GDE, ela depende do número de políticas que podem ser usadas pelo cache. Supondo que as informações sobre os arquivos no cache sejam armazenados em uma lista duplamente ligada, para cada política é necessário que seja alocado espaço em memória para mais 2 ponteiros para cada registro de arquivo armazenado. Esses ponteiros são necessários na ligação do registro de cada arquivo armazenado à lista ordenada pela

política. Dessa forma, é possível manter apenas um registro com informações sobre o arquivo e várias ordenações de prioridade de exclusão.

Entretanto, o maior problema é o consumo de ciclos do processador necessários para manter muitas diferentes ordenações para os arquivos no cache. Caso as informações dos arquivos fossem mantidas em uma lista duplamente ligada, a demanda por processamento causada pelas repetidas reordenações poderia tornar o custo do GDE muito alto.

Para diminuir o custo de execução, o GDE usa um sistema com listas de vetores de ponteiros para diminuir o número de ciclos de processador necessários para a manutenção das listas de arquivos. O esquema na figura 4.5.1 representa as estruturas necessárias para procura e remoção de arquivos em uma única política.

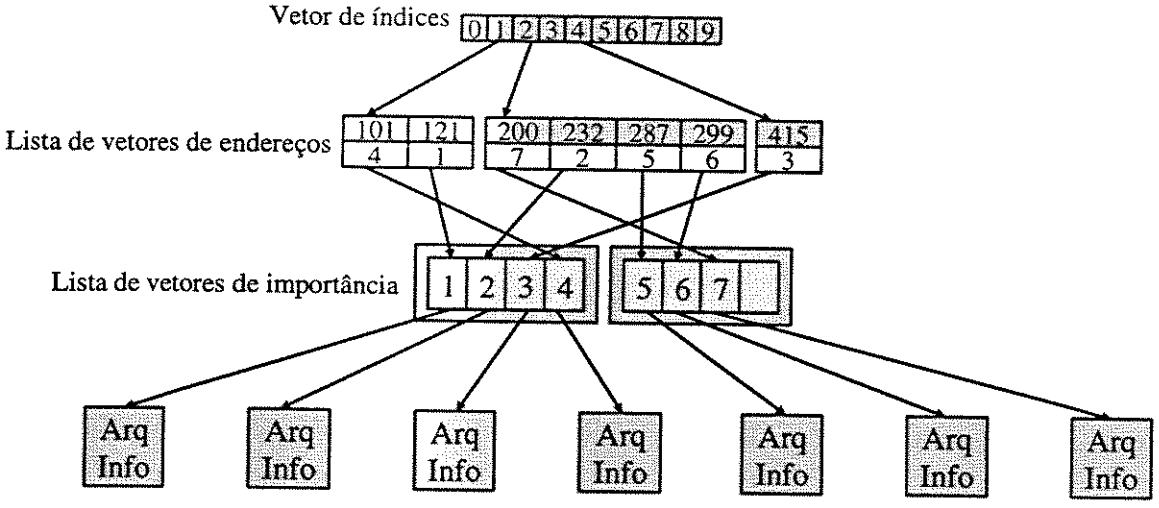


Fig. 4.5.1 – Estruturas necessárias para aumentar o desempenho do GDE

As listas de vetores de endereço e índice são utilizadas apenas para a localização rápida dos arquivos. Os vetores de índice são usados para endereçar rapidamente o início da lista dos arquivos cujos algarismos mais significativos do valor numérico gerado através da aplicação de uma função de *hash* no nome do arquivo são os mesmos. A lista de vetores de endereços é utilizada para armazenar a localização das informações sobre o arquivo nas listas de vetores de importância. O tamanho do vetor de índice depende do número de arquivos armazenados. Quanto maior o vetor, menor o número de posições do

vetor de endereços que devem ser avaliadas. Por outro lado, o aumento exagerado do tamanho do vetor de índice causa um atraso maior na procura dentro deste vetor.

Para cada política básica do GDE, são mantidos o vetor de índices, a lista de vetores de endereços e a lista de vetores de importância. Uma representação de parte das estruturas exigidas pelo GDE quando da utilização de 3 políticas básicas é apresentada na figura 4.5.2.

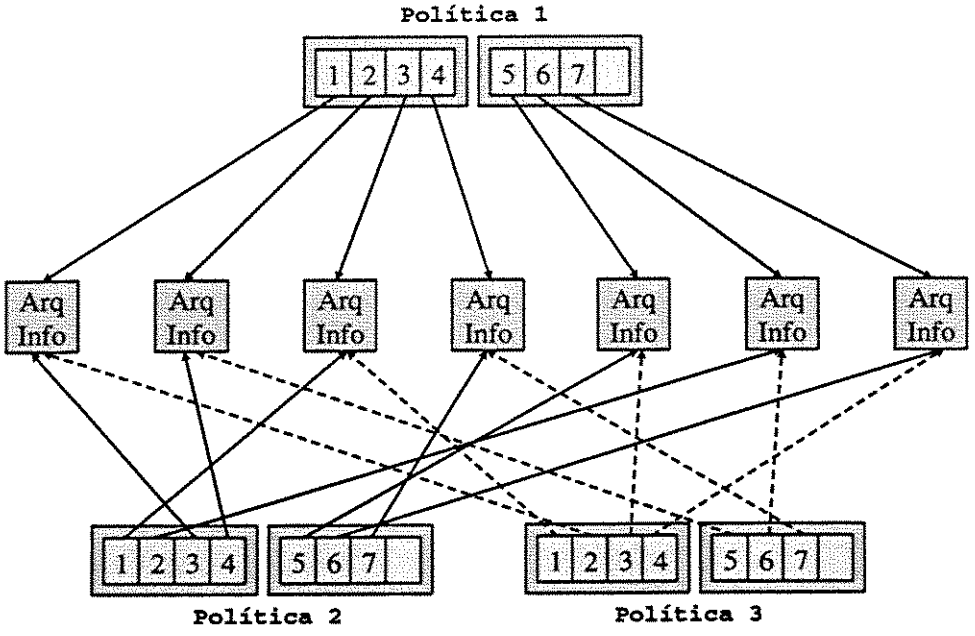


Fig. 4.5.2 – Listas de vetores de ponteiros

No exemplo da figura 4.5.2, é importante mencionar que, usando o sistema de listas de ponteiros, o crescimento da demanda por processamento é linear com o número de políticas básicas desejadas, pois é como se o cache tivesse que manter várias políticas independentes. Já no caso do crescimento do consumo de memória, o consumo é menor que o de várias políticas independentes, pois apenas um registro de informações sobre cada arquivo (Arq Info) precisa ser mantido no cache.

O sistema de listas de ponteiros foi utilizado em todos os experimentos apresentados nesse capítulo. No restante dessa seção é sugerida outra estratégia para diminuir ainda mais a demanda por capacidade de processamento para o controle da prioridade de exclusão dos arquivos. Essa estratégia consiste em manter uma única fila de

documentos, ordenada de acordo com um valor de importância de manutenção do arquivo, calculado utilizado os pesos das políticas básicas. Inicialmente, considera-se o caso em que se os pesos das políticas forem alterados, é necessário reordenar toda a lista de arquivos. A figura 4.5.3 apresenta um exemplo onde um cache usando um GDE altera os pesos das políticas básicas e, por isso, precisa reordenar toda a lista de arquivos.

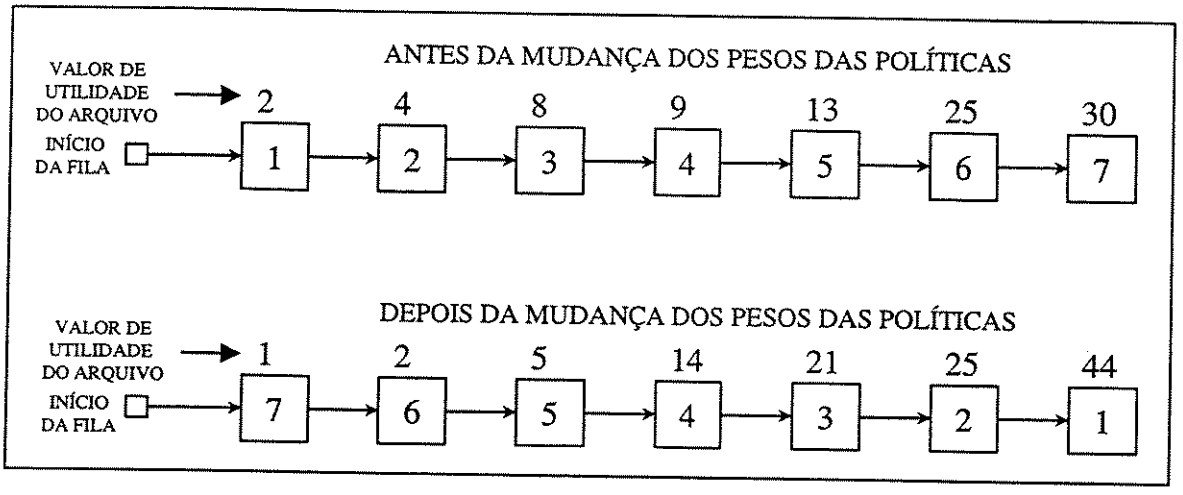


Fig. 4.5.3 – Antes e depois de uma reordenação da lista de arquivos

Entretanto, em muitos casos, a reordenação pode ser computacionalmente muito cara. Por isso, pode-se fazer com que o valor usado para ordenar a lista de arquivos não seja recalculado para todos os arquivos da lista. Nesse caso, apenas os arquivo inseridos a partir da mudança de pesos das políticas básicas terão seu valor de ordenação correto. Dessa forma, a orientação do cache para privilegiar determinados grupos de arquivos será feita gradativamente, a medida em que os arquivos antigos forem removidos.

4.6. Impacto do favorecimento de arquivos

Para demonstrar a utilização do GDE no favorecimento de arquivos específicos, foi realizada uma simulação usando um cache de 1 GB. O GDE usou duas políticas: LRU e LRU favorecendo arquivos específicos.

Os arquivos favorecidos pela LRU com favorecimento foram escolhidos aleatoriamente. Foram escolhidos 100 arquivos do *trace* RTP, usado nessa experiência.

O valor do peso da política LRU foi 1 em todas as simulações. Foram experimentados pesos para a política LRU com favorecimento entre 1 e 10 e foi chamado grau de favorecimento. Os resultados do desempenho em *hit ratio* dos arquivos escolhidos é apresentado na figura 4.6.1.

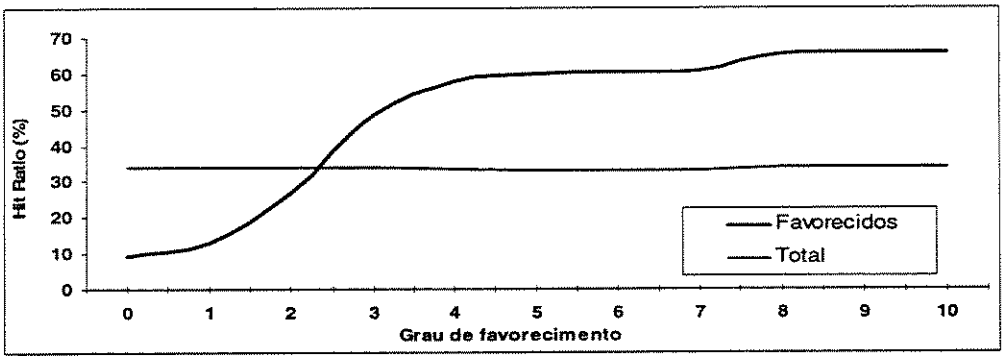


Fig. 4.6.1 – Impacto do favorecimento de arquivos

Através da figura 4.6.1, observa-se que a variação do peso da política LRU com favorecimento permitiu que as requisições aos arquivos escolhidos obtivessem um desempenho em *hit ratio* muito maior, oferecendo assim uma QoS variável de acordo com o grau de favorecimento.

Entretanto, é esperado que o desempenho total do cache diminua quando usado o favorecimento de arquivos. Isso porque o espaço de armazenamento do cache será ocupado por uma quantidade potencialmente muito grande de arquivos que devem ser favorecidos, mas que não contam com grande popularidade dentre todos os usuários do cache. Na figura 4.6.1 isso não pode ser visto, pois nesse caso a soma dos tamanhos dos arquivos favorecidos foi pouco maior que 2% do tamanho do cache.

O efeito de diminuição do desempenho total do cache pode ser visto no gráfico da figura 4.6.2. Nesse caso, também foi usado um cache de 1 GB com duas políticas: LRU e LRU favorecendo arquivos específicos. Porém, foram escolhidos 1.000 arquivos do *trace* RTP para serem favorecidos, o que representou cerca de 23% do tamanho do cache.

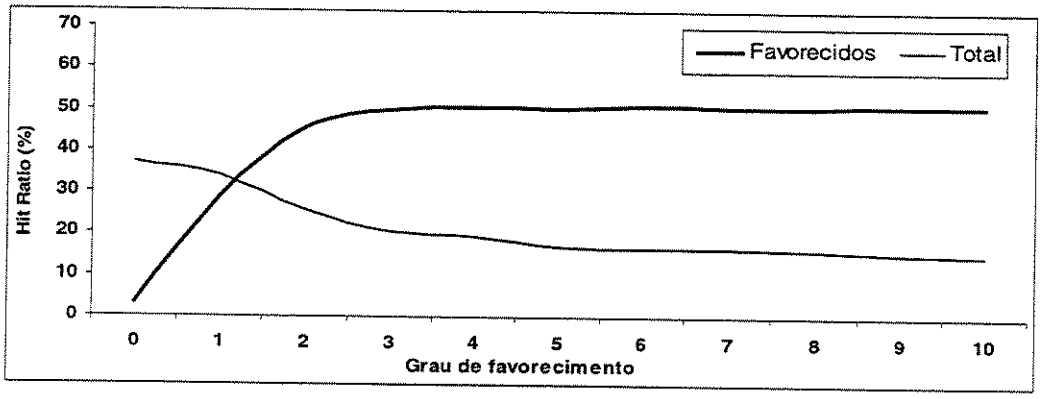


Fig. 4.6.2 – Impacto do favorecimento no desempenho total do cache

Analisando as figuras 4.6.1 e 4.6.2, pode-se verificar que o favorecimento de arquivos é realmente capaz de aumentar o desempenho em *hit ratio* dos arquivos favorecidos o que aumenta a disponibilidade dos mesmos, aumentando assim a QoS oferecida aos interessados nessa maior disponibilidade.

Entretanto, o favorecimento pode causar fortes impactos negativos no desempenho total do cache, principalmente no caso do número de arquivos favorecidos ser muito grande quando comparado com o tamanho do cache.

4.7. Conclusões sobre o sistema GDE

Apesar da idéia de utilizar múltiplas filas de documentos permitir construir políticas de substituição de alto desempenho em *hit ratio* e *byte hit ratio,* como a Política Dinâmica de Substituição, o *overhead* causado pela manutenção das múltiplas filas de documentos é um problema que pode inviabilizar a utilização de tais políticas. Entretanto, a utilização de múltiplas filas em um sistemas utilizado para prover diferentes níveis de QoS a usuários da Web é bastante interessante.

Em simulações de uso, o Gerenciador Dinâmico de Espaço (GDE) apresentou um desempenho comparável ao das melhores políticas de substituição existentes. Porém, além de ser um gerenciador de espaço bastante eficiente, sua capacidade de ser configurado através da variação dos pesos das políticas básicas permite obter do cache o desempenho desejado, na métrica escolhida, quando for necessário. Isso permite que o

GDE seja uma excelente opção para possibilitar que o cache ofereça diferentes QoS tanto a servidores Web quanto a clientes específicos.

No GDE, o problema do *overhead* causado pela manutenção de múltiplas filas é diminuído quando considerada a vantagem de que ele introduz a capacidade de variação de QoS dos serviços oferecidos em um ponto bastante importante na rede de transmissão de documentos: o cache.

Foram ainda apresentadas estratégias que permitem diminuir a capacidade de processamento necessária para executar o GDE, possibilitando que a sua utilização seja feita mesmo em computadores que não possuam capacidades de processamento muito grandes.

Capítulo 5

Passagem de Recomendação

5.1. Introdução

Apesar de muitos trabalhos tentarem aumentar o desempenho individual dos caches em uma rede, muitas vezes uma grande melhora do desempenho de um cache pode não representar uma melhora significativa no desempenho geral da rede de caches devido à perda de referência da distribuição de popularidade inicial.

A perda de referência de popularidade ocorre quando os caches de níveis mais elevados não conseguem obter um bom desempenho devido a não mais poderem basear-se apenas no fluxo de requisições que chega a eles. Isso ocorre porque as características de popularidade desse fluxo foram degradadas quando da sua passagem por caches de níveis hierárquicos inferiores. Isso faz com que uma melhora de desempenho em *hit ratio* nos caches de nível elevado não signifique que eles estejam dando prioridade a arquivos mais populares entre os clientes e sim aos arquivos mais populares no fluxo de requisições que estão recebendo.

Primeiramente, este capítulo apresenta um estudo do impacto da instalação de uma rede hierárquica de caches para Web no fluxo de requisições. A seguir, é proposto o mecanismo de passagem de recomendação como forma de evitar que caches de níveis hierárquicos altos percam a referência da popularidade inicial, aumentando seu desempenho.

5.2. Variação de popularidade dos arquivos

A popularidade tanto de conjuntos de arquivos quanto de arquivos específicos não permanece constante durante longos períodos de tempo. Essa seção apresenta gráficos de análises de *traces* que visaram determinar a variação da popularidade dos arquivos com o decorrer do tempo. Foram analisados *traces* provenientes do sistema IRCACHE (RTP) [NLANR00] e UNICAMP [CCUEC02]. Esses *traces* são relativos a 7 dias de trabalho do servidor de cache para Web.

Para traçar os gráficos das figuras 5.2.1a e 5.2.1b, os *traces* foram divididos em 50 intervalos. Esses intervalos foram então analisados para verificar qual o percentual de arquivos, dentre os 100 e 1000 arquivos mais populares no 25° intervalo, estavam dentre os 100 e 1000 mais populares nos demais intervalos. Pode-se verificar através dos gráficos que a proporção de arquivos populares no 25° intervalo dentre os mais populares cai rapidamente, estabilizando-se em um valor de 20 a 40%. Neste trabalho, essa proporção será chamada de percentual de presença.

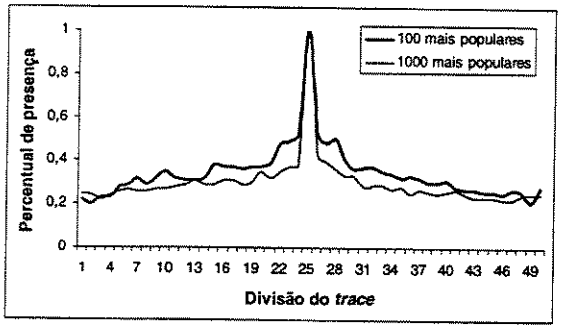


Fig. 5.2.1a – Variação da popularidade - RTP

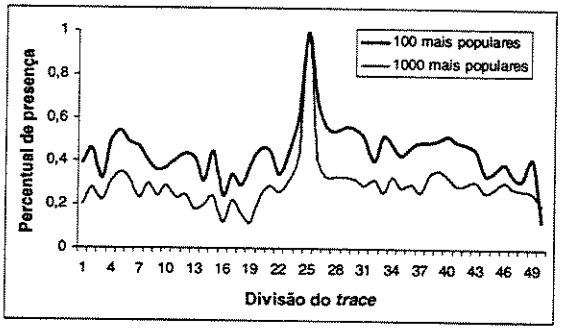


Fig. 5.2.1b – Variação da popularidade - UNICAMP

O percentual de presença dos 100 arquivos mais populares é maior que o dos 1000 arquivos mais populares. Isso mostra que os arquivos mais populares em um certo momento têm maior chance de manter a sua posição de popularidade que arquivos menos populares.

Para verificar o que acontece com a popularidade dos arquivos através do *trace*, os gráficos das figuras 5.2.2a e 5.2.2b apresentam o número de acessos no decorrer do tempo. A posição na fila de popularidade dos arquivos utilizada na legenda dos gráficos foi calculada através da soma do número de acessos aos arquivos em todo o *trace*.

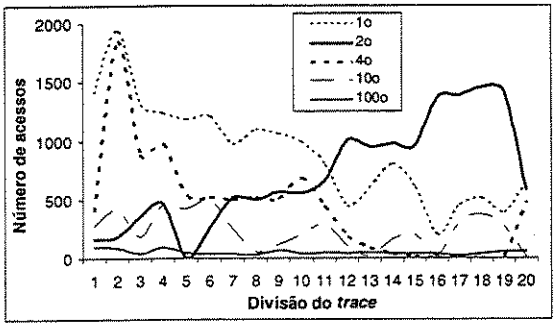


Fig. 5.2.2a – Mais populares - RTP

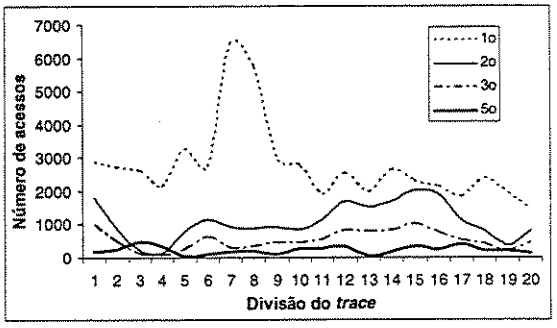


Fig. 5.2.2b – Mais populares - UNICAMP

Note que existem trocas de posições de popularidade entre os arquivos mais populares e que o crescimento e a diminuição de popularidade acontecem rapidamente. Porém, essa troca é muito mais freqüente entre arquivos de popularidade semelhante do que entre arquivos cujos números de acessos difiram por ordens de magnitude.

5.3. A perda da referência de popularidade

A distribuição de popularidade de um fluxo de requisições segue uma distribuição Zipf. Através da lei de Zipf, é possível dizer que a probabilidade relativa de acontecer uma requisição ao *i-ésimo* documento mais popular é proporcional a $\frac{1}{i^{\alpha}}$, com o valor de α variando de acordo com características particulares aos fluxos de requisições. Assim, a freqüência com que um documento foi acessado no passado tem fortes implicações na probabilidade de ser acessado novamente no futuro.

O gráfico na figura 5.3.1 apresenta a distribuição de popularidade de alguns fluxos de requisições reais, como o RTP-NLANR[NLANR00], UNICAMP[CCUEC02] e NASA[NASA00].

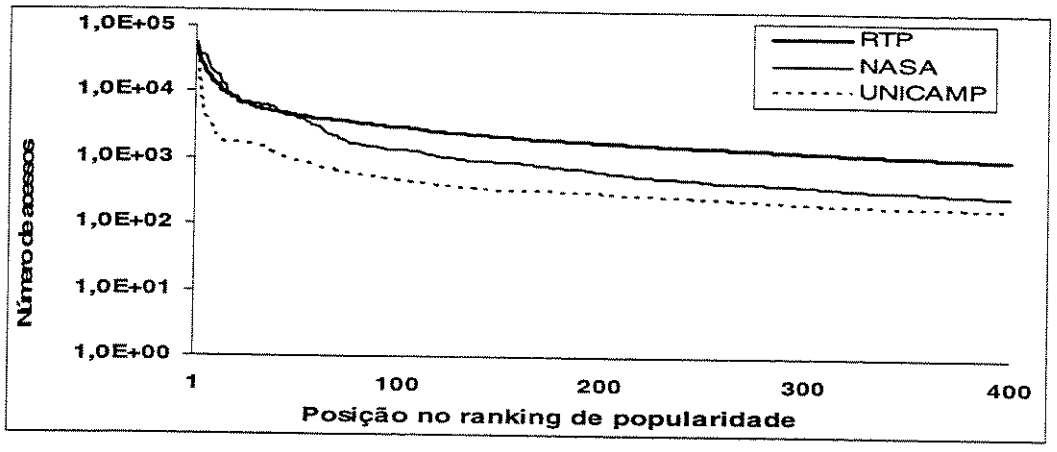


Fig. 5.3.1 – Exemplo de fluxos reais

Através da figura 5.3.1, pode-se observar que o crescimento da relação de popularidade entre os arquivos não é constante em nenhum dos fluxos apresentados. Em outras palavras, isso significa que, por exemplo, a relação de popularidade entre dois arquivos mais populares e entre dois pouco populares é diferente. Pode-se observar através da figura 5.4.1 que essa relação cresce de acordo com o aumento da popularidade.

5.3.1. União de fluxos de requisições

Este trabalho considera que é possível substituir uma rede de caches de mesmo nível por apenas um cache equivalente. Nesse caso, o fluxo de requisições enviadas ao cache equivalente é igual à união dos fluxos de requisições feitas a cada cache, devidamente intercalados de acordo com a hora das requisições. Considera ainda que o fluxo de requisições não satisfeitas pelo cache equivalente é igual à união dos fluxos de requisições não satisfeitas pelos caches.

As figuras 5.3.2a, 5.3.2b e 5.3.2c mostram três exemplos de possibilidades que podem ocorrer ao combinar dois fluxos de requisições. Nesses exemplos, foram utilizados dois fluxos, A e B, sendo que o fluxo A possui α=0,5 enquanto que o fluxo B possui α=0,7. Para fins de clareza do gráfico, ambos os fluxos contem requisições a apenas 10 arquivos diferentes.

Na figura 5.3.2a, um dos fluxos tem um número de requisições muito maior que o outro. Nesse caso, a quantidade de requisições no fluxo B é pequena quando comparada

com o fluxo A. Como a popularidade total dos arquivos requisitados através de B é pequena quando comparada com os arquivos de A, pode-se considerar que a distribuição de um fluxo obtido pela intercalação temporal de A e B, gerando o fluxo A+B é equivalente à distribuição do fluxo A. Portanto, nesse caso, pode-se considerar que a distribuição A+B segue uma distribuição muito similar à distribuição Zipf do fluxo A.

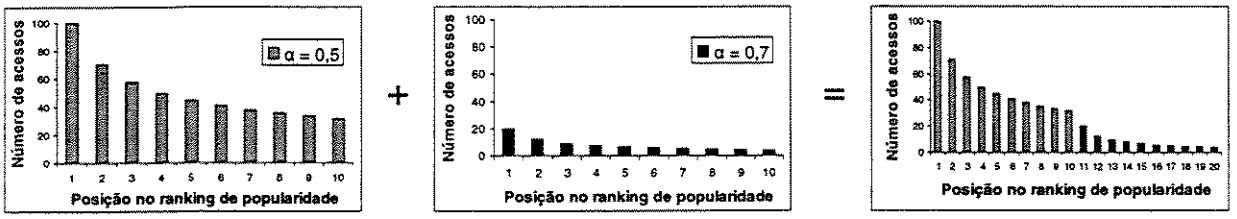


Fig. 5.3.2a – Caso em que um fluxo é mais ativo que outro

No caso representado pela figura 5.3.2b, o número de requisições feitas em ambos os fluxos não é desprezível em relação ao outro. Porém, nesse caso, considera-se que os arquivos requisitados através dos fluxos sejam diferentes.

Como resultado, a união desses fluxos ainda tem uma distribuição parecida com as distribuições A e B. Porém, caso essa nova distribuição seja aproximada por uma distribuição Zipf, o valor de α na distribuição resultante será diferente dos valores de α em A e B.

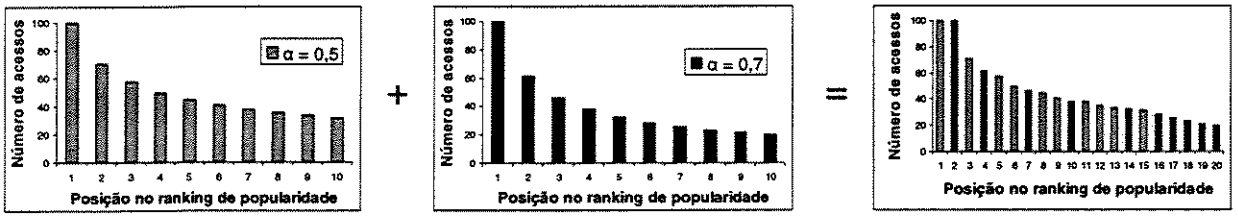


Fig. 5.3.2b – Fluxos de requisições a arquivos diferentes

A figura 5.3.2c apresenta o caso em que tanto A quanto B contém requisições aos mesmos arquivos. Nesse caso, pode-se observar que a distribuição resultante continua semelhante às distribuições A e B. Porém, mais uma vez, caso essa nova distribuição seja aproximada por uma distribuição Zipf, o valor de α na distribuição resultante será diferente dos valores de α em A e B.

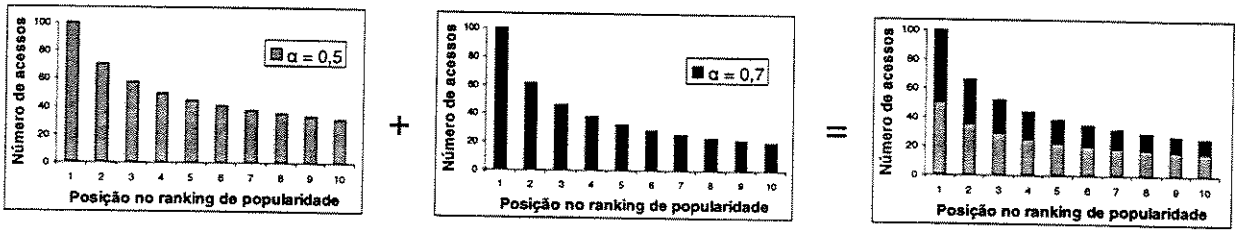


Fig. 5.3.2c – Fluxos de requisições aos mesmos arquivos

Existe ainda um caso misto entre os exemplos das figuras 5.3.2b e 5.3.2c, onde existem requisições aos mesmos arquivos através de A e B e também arquivos requisitados apenas através de um dos fluxos. Ainda nesse caso, a distribuição resultante da união dos fluxos será semelhante às distribuições resultantes em 5.3.2b e 5.3.2c. Uma abordagem matemática do comportamento do fluxo de requisições pode ser encontrada em [Yang02] e [Busari01].

Este trabalho então considera que a união de dois ou mais fluxos de requisições gera uma distribuição cuja forma gráfica é bastante semelhante aos fluxos formadores. Assim, uma simplificação de uma rede de caches hierárquicos pode ser feita como representado na figura 5.3.3, onde cada conjunto de caches de mesmo nível é representado por um único cache instalado em uma fila de caches sucessivos.

Como os *traces* utilizados neste trabalho foram obtidos a partir de caches *proxy* reais, as suas requisições são provenientes da união de fluxos de requisições, após passar por caches de nível hierárquico inferior. Por isso, eles podem ser usados na simulação de malhas de caches.

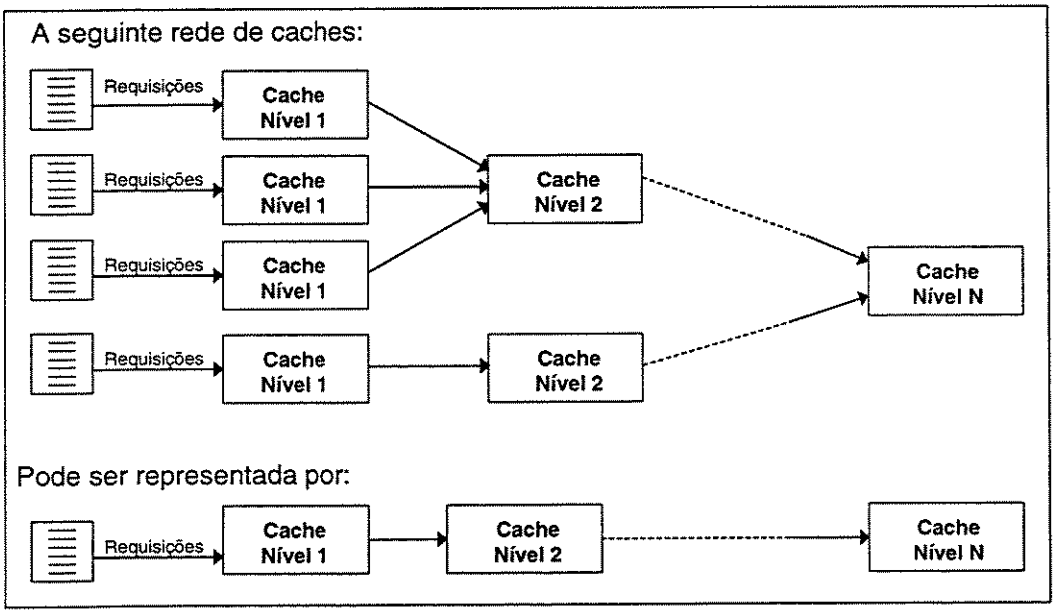


Fig. 5.3.3 – Modelo equivalente à rede hierárquica de caches

5.3.2. Fluxos hipotéticos

Para fins de análise, um gráfico da distribuição de popularidade de um fluxo real pode ser dividido em pequenas partes. Cada uma dessas partes pode ser aproximada por pequenos segmentos de reta provenientes de uma distribuição cujo crescimento é constante. Assim, um gráfico da distribuição de popularidade de um fluxo real pode ser aproximado pela justaposição de pedaços de gráficos de distribuições com crescimento constante. Para fins de comparação, a figura 5.3.4 apresenta gráficos de popularidade de fluxos hipotéticos com crescimento constante.

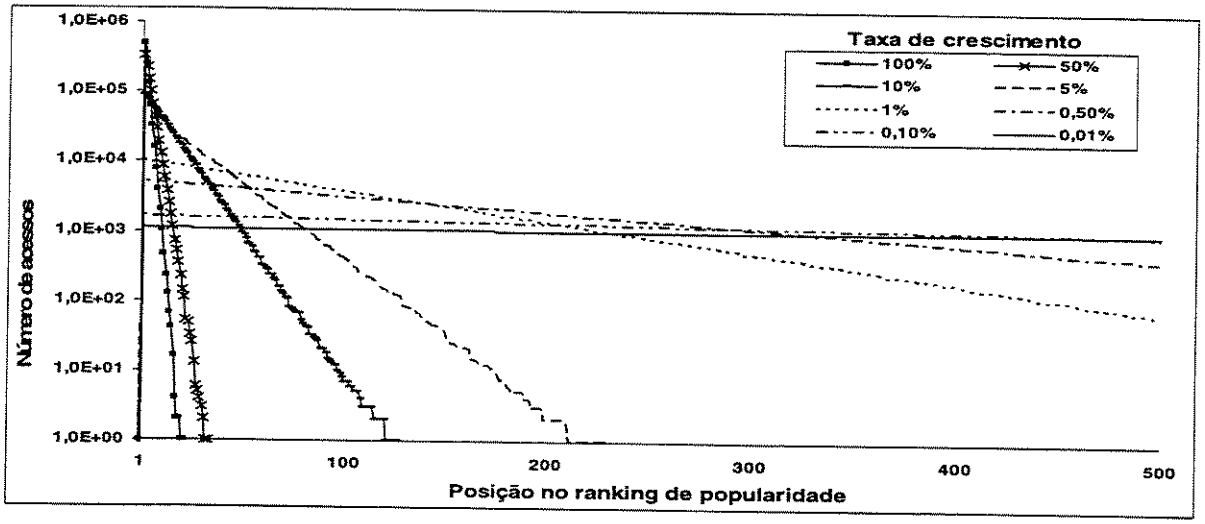


Fig. 5.3.4 – Popularidade de fluxos com crescimento constante

No caso de crescimento constante, a relação entre a popularidade do n-ésimo elemento mais popular e o n+1-ésimo é constante. Por exemplo, no caso de um crescimento de 10%, significa que a popularidade do n-ésimo elemento é 10% maior que a do n+1-ésimo elemento.

A utilização de fluxos hipotéticos com crescimento de popularidade constante permite analisar de forma mais clara o impacto da instalação dos caches, podendo-se verificar o efeito do cache no fluxo de requisições de acordo com cada taxa de crescimento de popularidade dos arquivos requisitados. Alguns exemplos podem ser observados através das figuras 5.3.5a, 5.3.5b e 5.3.5c.

Para obter os gráficos nas figuras 5.3.5a, 5.3.5b e 5.3.5c os *traces* hipotéticos foram aplicados a um cache utilizando política LRU. Foi utilizado um cache de tamanho igual a 1% do valor da soma dos tamanhos dos arquivos no fluxo. Em caso de *miss*, ou seja, quando o cache não possuía o arquivo requisitado, a requisição era armazenada em um arquivo, gerando assim um *trace* formado apenas por requisições não satisfeitas. Esse novo *trace* era então aplicado novamente ao cache, repetindo-se o processo por duas ou três vezes.

Através da figura 5.3.5a, observa-se que os arquivos mais populares foram encontrados no cache, ou seja, obtiveram um *hit*, na primeira vez em que foram requisitados ao cache. Isso aconteceu porque a popularidade desses arquivos era tão maior que a dos arquivos menos populares, que sempre existiam cópias suas no cache,

pois o número de requisições a eles era alto o suficiente para mantê-los com baixa prioridade de descarte pelo cache.

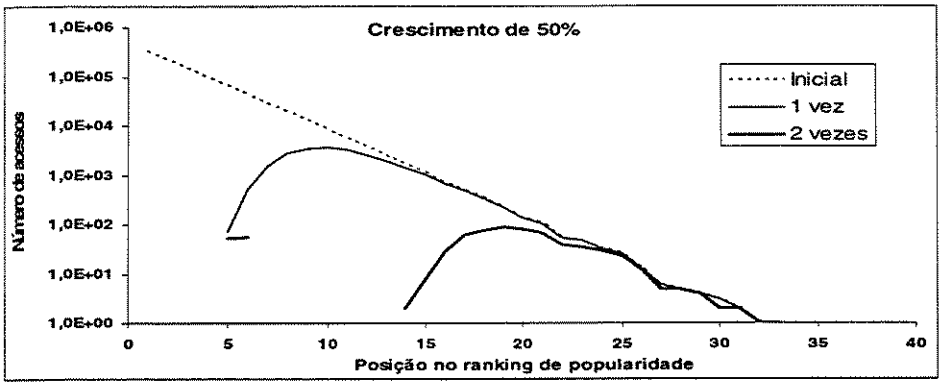


Fig. 5.3.5a – Aplicação sucessiva de trace com taxa de crescimento de popularidade de 50%

A figura 5.3.5b apresenta um exemplo do impacto da perda de referência de popularidade quando da aplicação sucessiva de caches aos fluxos. Nesse caso, quando o fluxo é aplicado pela segunda vez ao cache, os arquivos mais populares no *trace* utilizado não são os mais populares no *trace* inicial. Por isso, o cache prioriza manter armazenados os arquivos que são mais populares no *trace* formado por *misses* da primeira aplicação, não dando preferência aos arquivos mais populares entre os clientes.

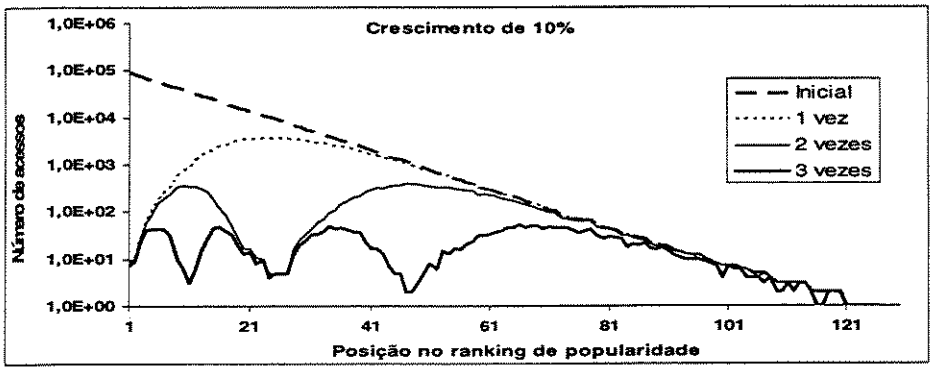


Fig. 5.3.5b – Aplicação sucessiva de trace com taxa de crescimento de popularidade de 10%

A figura 5.3.5c apresenta o caso em que a diferença de popularidade entre os arquivos no fluxo é pequena. Nesse caso, pode-se observar que o impacto da instalação do cache no formato do fluxo é bastante reduzido, principalmente a partir da segunda

aplicação. Isso ocorre devido à dificuldade do cache em decidir quais são os documentos mais populares, devido à pouca diferença entre as popularidades dos documentos.

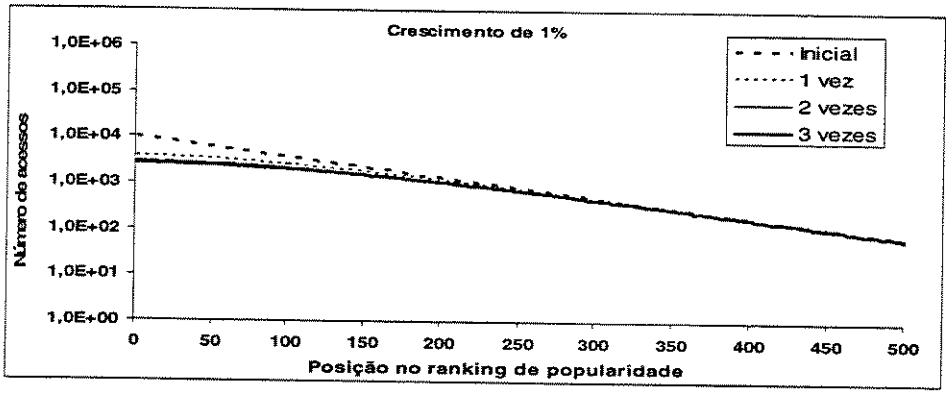


Fig. 5.3.5c – Aplicação sucessiva de trace com taxa de crescimento de popularidade de 1%

5.3.3. Impacto das políticas LRU e LFU

O impacto da passagem do fluxo de requisições por um cache para Web depende da política de substituição usada nesse cache. Apesar de existirem políticas de substituição com melhor desempenho, tal como a Greedy-Dual[Jin00], as políticas LRU e SIZE foram escolhidas por serem simples e largamente conhecidas.

A política LRU, *(Least Recently Used)*, remove do cache o arquivos menos recentemente acessado. Com isso, ela garante um alto de *hit ratio.*

A política LFU, *(Least Frequently Used)*, remove do cache o documento usado com menor freqüência. Com isso, faz com que documentos mais populares sejam privilegiados no cache. Porém, seu desempenho em *hit ratio* é menor que o da LRU.

As figuras 5.3.6a e 5.3.6b apresentam o impacto da instalação de caches com as políticas LRU e LFU no fluxo de requisições. O *trace* utilizado foi o RTP-NLANR [NLANR00], contendo aproximadamente três milhões de requisições. Para a obtenção dos resultados apresentados neste trabalho foi utilizado um simulador de caches que permite simular as políticas LRU e LFU, bem como o mecanismo de passagem de recomendação.

Através da figura 5.3.6a, é possível verificar que a política LRU apresenta um sério problema de perda de referência da popularidade inicial. Isso porque a distribuição

de popularidade do *trace* formado pelos *misses* da primeira vez que o *trace* é aplicado ao cache não mais se assemelha ao *trace* inicial. Isso ocorre porque, apesar de a política LRU privilegiar os arquivos mais populares e, por isso, mais acessados, esses arquivos não são muito mais populares que os arquivos menos populares. Isso faz com que, em determinados momentos, sejam removidos do cache arquivos que são mais populares quando considerado todo o *trace* mas naquele momento não estão muito populares.

O problema é ainda mais agravado quando das reaplicações seguintes ao cache, pois o cache irá agora basear-se em valores de popularidade referentes apenas ao *trace* formado por *misses*, não mais sendo possível considerar a popularidade real dos arquivos, cuja informação ficou perdida quando da primeira passagem do fluxo pelo cache.

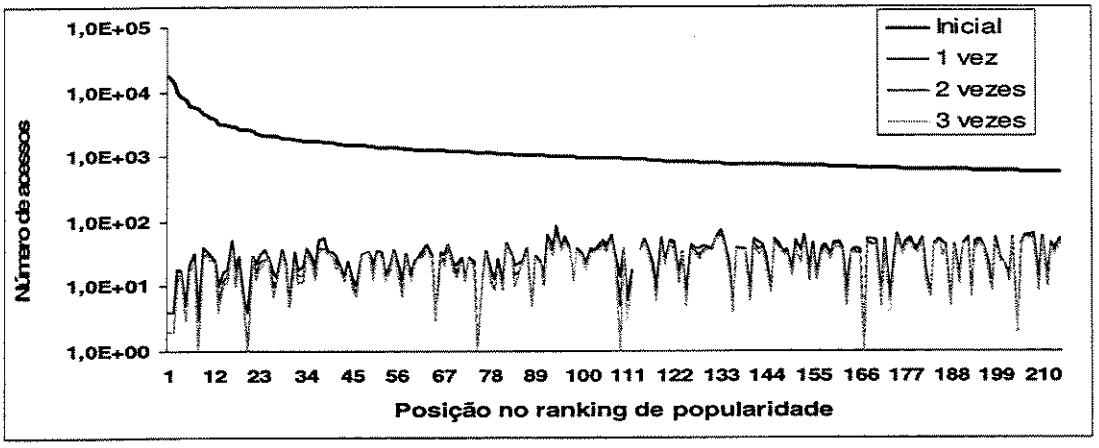


Fig. 5.3.6a – Impacto da política LRU no fluxo de requisições

A seqüência de gráficos nas figuras 5.3.6b a 5.3.6e apresentam o impacto no fluxo de requisições da aplicação da política LFU. Através dele, pode-se observar que a política LFU tem facilidade para manter arquivos populares no cache apenas quando esses arquivos são muito mais populares que outros. Depois da primeira vez em que o *trace* é aplicado a um cache que usa LFU, essa política também apresenta o problema de perda de referência de popularidade. Isso pode ser visto através das figuras 5.3.6d e 5.3.6e, onde os arquivos mais populares que inicialmente não foram escolhidos para permanecer no cache, também não o foram na segunda e terceira aplicações do *trace* ao cache, o que faz com que eles ainda tenham um grande número de acesso no *trace* formado por *misses*

e, conseqüentemente apresentem um grande número de acessos nos gráficos das figuras 5.3.6d e 5.3.6e.

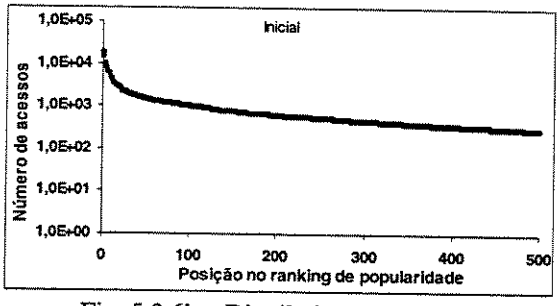


Fig. 5.3.6b – Distribuição inicial

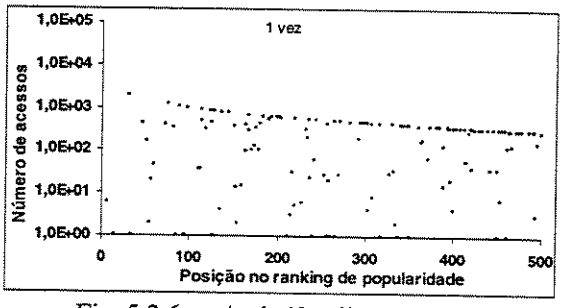


Fig. 5.3.6c – Após 1ª aplicação de LFU

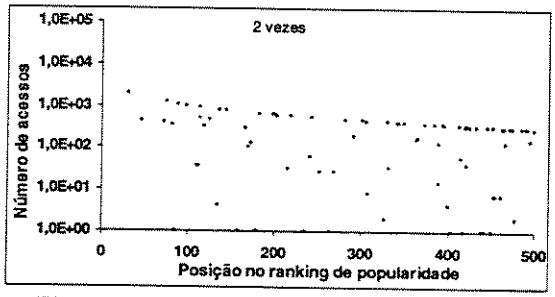


Fig. 5.3.6d – Após 2ª aplicação de LFU

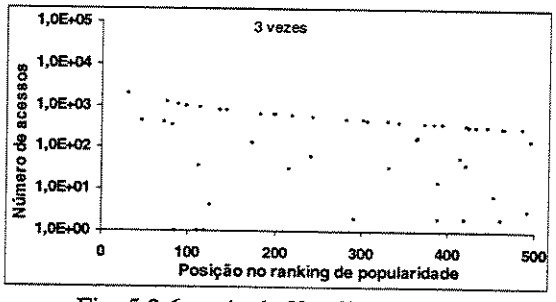


Fig. 5.3.6e – Após 3ª aplicação de LFU

Além de apresentar perda de referência de popularidade, um problema com a política LFU é que ela proporciona um desempenho em *hit ratio* menor que a LRU. O desempenho dessas duas políticas nos testes utilizados para traçar os gráficos das figuras 5.3.6a a 5.3.63e são apresentados na tabela 5.3.1.

Passagem pelo cache	LRU	LFU
Primeira vez	28,90	20,34
Segunda vez	0,36	4,67
Terceira vez	0,01	4,00
Soma	**29,27**	**29,01**

Tabela 5.3.1 – Desempenho em *hit ratio* da Passagem de Recomendação

Como pode ser visto na tabela 5.3.1, a política LRU apresenta um bom resultado quando da primeira aplicação ao cache. Porém, tanto na segunda quando na terceira vez,

o desempenho cai praticamente a zero, devido à perda de referência da popularidade inicial dos arquivos. Já a política LFU apresenta um pior resultado na primeira aplicação ao cache, mas melhores resultados nas aplicações seguintes, por ser menos sensível à perda de referência à distribuição de popularidade inicial. A soma dos desempenhos das duas políticas é semelhante, porém ainda é menor que o máximo possível, usando um cache de tamanho infinito, que foi de 50,14%.

5.4. Passagem de recomendação

Este trabalho propõe a passagem de recomendação como uma estratégia utilizada para evitar que caches de nível hierárquico superior percam a referência de popularidade dos arquivos requisitados pelos clientes.

A estratégia consiste na passagem do número de acessos feitos a documentos descartados em um cache para seu cache de nível superior. Dessa forma, o cache de nível superior terá informações sobre a distribuição de popularidade do fluxo de requisições submetidas aos caches anteriores a ele, mesmo que seja utilizada a política LRU na primeira vez em que o fluxo passa por um cache. Dessa forma, é possível fazer com que a rede de caches hierárquicos apresente um bom desempenho em *hit ratio* em todos os níveis e que os *hits* mesmo nos níveis mais elevados sejam relativos a arquivos mais populares entre os clientes.

A figura 5.4.1 apresenta um diagrama que representa o mecanismo de passagem de recomendação. Nessa estratégia, além das requisições aos arquivos não encontrados, os caches enviam aos caches de nível hierárquico superior o número de acessos dos arquivos removidos. O cache de nível superior usa então esse número para ordenar a prioridade de exclusão dos arquivos que contém.

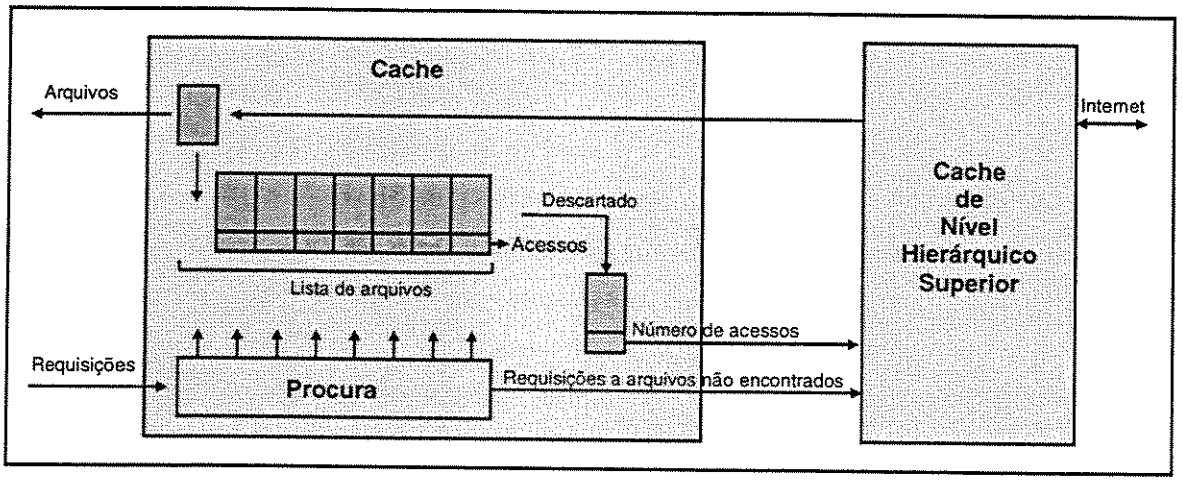


Fig. 5.4.1 – Mecanismo da passagem de recomendação

Durante o funcionamento, os arquivos muito populares não serão removidos dos caches de nível hierárquico baixo. Porém, caso existam arquivos com alta popularidade cuja soma dos tamanhos seja maior que a capacidade dos caches, acontecerá a remoção de arquivos bastante populares. Nesse caso, o número de acessos, que também pode ser chamado de peso da recomendação fará com que esses arquivos tenham grandes garantias de serem mantidos com baixas prioridades de exclusão nos caches de nível superior.

5.4.1. Desempenho e impacto no fluxo de requisições

A passagem de recomendação apresenta a vantagem de permitir que caches de níveis superiores tenham conhecimento da distribuição de popularidade real entre os arquivos. Com isso, é possível utilizar a política LRU no primeiro nível de caches, sem se preocupar com o impacto dessa política no fluxo de requisições.

O gráfico da figura 5.4.2 apresenta o desempenho da estratégia de passagem de recomendação comparado à utilização das políticas LRU e LFU sozinhas. Através desse gráfico, pode-se notar que a política LRU apresenta um bom desempenho apenas na primeira vez que é utilizada pelo cache. A partir de então, devido à perda de referência de popularidade, seu desempenho passa a ser praticamente nulo. A política LFU conseguiu manter um bom desempenho na segunda e terceira vezes que o fluxo foi submetido ao cache. Porém, apresentou um desempenho muito menor que a LFU na primeira aplicação do fluxo ao cache.

Através da passagem de recomendação, podem-se usar políticas de substituição que "destruam" a referência da popularidade inicial para caches seguintes, mas que apresentem altas taxas de desempenho sem se preocupar com a perda de referência de popularidade. Pode ser notado no gráfico da figura 5.4.2 que o desempenho da utilização da passagem de recomendação garantiu o mesmo desempenho da política LRU na primeira aplicação do fluxo ao cache e obteve altos desempenhos nas aplicações seguintes. Considera-se que o menor desempenho da passagem de recomendação quando da terceira aplicação ao cache deveu-se à características do fluxo, que permite um desempenho máximo possível, em um cache de tamanho infinito, de 50,14%. A soma do desempenho da passagem de recomendação nas três aplicações foi de 37,39%, o que representa um desempenho melhor que as políticas LRU e LFU sozinhas, conforme pode ser visto na tabela 5.3.1.

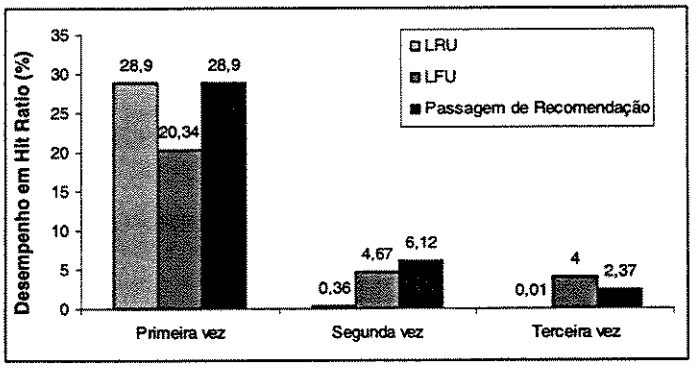


Fig. 5.4.2 – Desempenho da estratégia de passagem de recomendação

Através do gráfico da figura 5.4.3, pode-se notar o impacto da aplicação da passagem de recomendação no fluxo de requisições. Na experiência cujos resultados foram utilizados para traçar esse gráfico, utilizou-se a política LRU no cache durante a primeira aplicação do *trace* RTP-NLANR. A partir da segunda aplicação do fluxo ao cache, este ordenava a prioridade de exclusão baseando-se na recomendação resultante da aplicação anterior. Em caso de empate entre arquivos, era utilizada a política LRU.

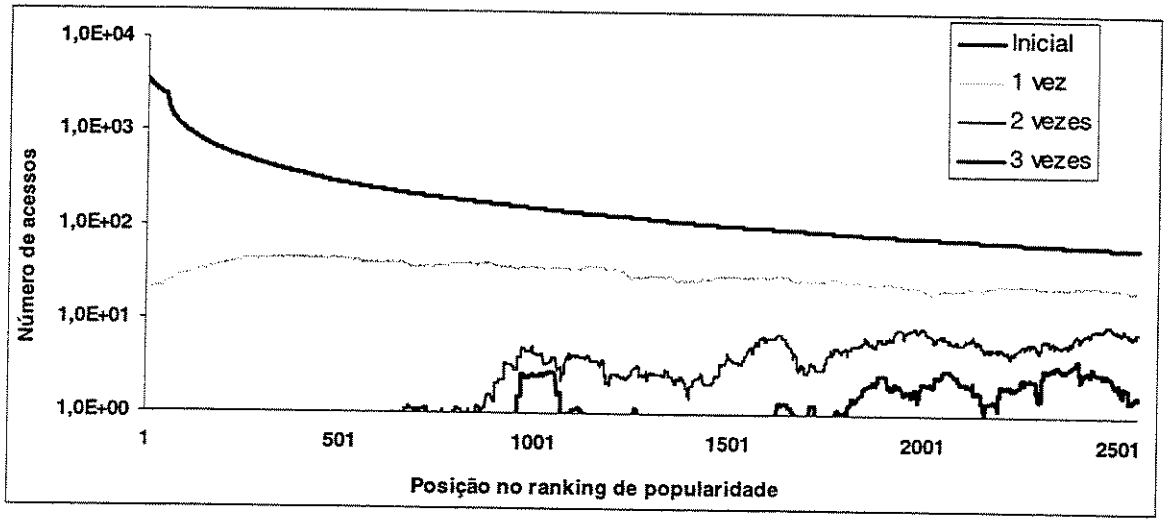


Fig. 5.4.3 – Impacto da passagem de recomendação no fluxo de requisições

No gráfico da figura 5.4.3, a linha cinza representa requisições a arquivos que não foram satisfeitas quando da primeira submissão a um cache que utiliza política LRU. Caso não fosse usada a passagem de recomendação, o cache seguinte teria que se basear apenas em uma distribuição de popularidade igual à representada no gráfico pela linha cinza, o que tornaria praticamente impossível ao cache tomar uma decisão acertada, conforme pode ser observado no gráfico da figura 5.4.2. Porém, a utilização da passagem de parâmetros permite que o cache seguinte elimine do fluxo arquivos mais populares entre os clientes. Isso pode ser visto através do número de requisições a arquivos populares na segunda e terceira aplicações ao cache.

O mecanismo de passagem de recomendação mostrou-se ainda mais eficiente quando utilizado em redes com menor variação de assuntos das páginas requisitadas, como a de uma universidade. A tabela 5.4.1 apresenta o desempenho em *hit ratio* das políticas LRU e LFU bem como da passagem de recomendação quando aplicado um *trace* proveniente do sistema de caches para Web da Universidade Estadual de Campinas, UNICAMP, Brasil [CCUEC02].

Desempenho (hit ratio)	LRU	LFU	Passagem de Recomendação
Primeira vez	61,09 %	49,92 %	61,09 %
Segunda vez	0,58 %	7,32 %	5,24 %
Terceira vez	0,1 %	8,18 %	2,48 %
Soma	61,77 %	65,42 %	68,81 %

Tabela 5.4.1 – Resultados da aplicação da Passagem de Recomendação

Esse *trace* é relativo a uma semana de uso do sistema de caches para Web, em período de funcionamento normal da universidade. Ele consiste de 1.33 milhões de requisições e a soma do tamanho dos arquivos requisitados foi de 5.65 GB. O desempenho máximo possível, em uma cache hipotético de tamanho infinito seria de 76,79%

Apesar deste trabalho apresentar apenas experimentos realizados com as políticas LRU e LFU, a passagem de recomendação pode ser utilizada com qualquer outra política com melhor desempenho, tal como a Greedy-Dual [Jin00].

5.4.2. *Overhead*

O impacto da utilização da passagem de recomendação nas redes de transmissão é muito pequena, visto que a recomendação consiste apenas na URL e no número de acessos, sendo que a URL pode ter sido submetida a uma função de *hash*, que diminua seu tamanho, mas mantenha sua capacidade de ser diferenciada das demais. Além disso, a recomendação pode ser enviada aos caches de nível superior dentro do pacote que carrega a requisição HTTP, o que permite não ser necessária a geração e transmissão de um pacote apenas para a passagem de recomendação.

O principal problema relacionado ao *overhead* causado pela utilização da passagem de recomendação está no processamento necessário à manutenção de filas de prioridade de remoção nos caches de nível superior. Porém, considerando que a mudança na fila de popularidade entre os arquivos ocorre principalmente entre arquivos de popularidade semelhante, pode-se otimizar a implementação da fila de prioridade para exclusão de forma que sua reordenação não tenha um custo computacional muito elevado. Por exemplo, pode-se dividir os arquivos em diversas filas, dependendo do

número de acessos. Conforme visto na seção 5.3, existirão muito mais trocas de posição entre arquivos já existentes nas filas do que a inserção de novos arquivos.

5.5. Conclusões sobre a Passagem de Recomendação

A perda da referência da popularidade inicial é um problema que leva os caches de níveis hierárquicos elevados a favorecerem arquivos que não são os mais populares dentre os usuários em detrimento de arquivos cuja popularidade é mais elevada, mas tiveram suas relações de popularidade alteradas pela passagem por caches anteriores.

Este capítulo apresentou a Passagem de Recomendação, que é uma estratégia desenvolvida para eliminar o problema da perda da referência da popularidade inicial. Além de possibilitar aos servidores de nível hierárquicos elevados obterem melhores taxas de desempenho em *hit ratio* e *byte hit ratio*, a utilização dessa estratégia permite fazer com que os efeitos dos caches na diminuição do tempo de resposta dos usuários seja direcionada aos arquivos mais populares dentre os usuários e não aos arquivos mais populares no fluxo de requisições que chega aos caches de nível hierárquico elevado.

Através dos resultados de simulações apresentados, pode-se concluir que a utilização da Passagem de Recomendação tem grande potencial para aumentar o desempenho de caches para Web em diferentes níveis de uma rede hierárquica, sem causar grande *overhead* nas redes e processadores. O *overhead* causado por essa estratégia nas redes é pequeno devido ao tamanho reduzido das informações de recomendação enviadas, conforme apresentado na seção 5.4.2.

Outra vantagem dessa estratégia é permitir a utilização de qualquer política de alto desempenho nos caches de níveis mais baixos, sem se preocupar com a perda de características de popularidade dos arquivos requisitados pelos clientes.

Capítulo 6

Beowulf como servidor Web

6.1. Introdução

Com o aumento da popularidade dos serviços oferecidos pela Internet, muitos estudos têm sido realizados com o objetivo de aumentar o desempenho dos sistemas de distribuição de documentos. Apesar de muitos trabalhos terem se concentrado principalmente no lado cliente desse sistema, através da elaboração de estratégias de posicionamento de caches, a capacidade do servidor de documentos é um elemento fundamental no desempenho do sistema de distribuição.

Apesar de um alto percentual de documentos requisitados a um servidor Web ainda consistir de documentos estáticos, tornam-se cada vez mais comuns as páginas geradas no momento do processamento das requisições, contendo informações muitas vezes tiradas de bancos de dados no momento da geração da página. Essa nova funcionalidade exige dos servidores Web uma capacidade de processamento cada vez maior, fazendo-se necessária a distribuição de processamento de requisições entre servidores cooperativos.

Para o processamento paralelo de requisições feitas a endereços Internet muito populares, podem ser utilizados clusters de processadores. Os nós formadores do cluster dividem a carga de requisições entre si, aumentando a taxa de requisições respondidas pelo sistema. Uma arquitetura de computador baseada em cluster que vem destacando-se

entre as demais por seu baixo preço e pela ausência do limite técnico para a quantidade máxima de PC's que podem ser conectados é a arquitetura Beowulf [Beowulf02].

Entretanto, a utilização de um Beowulf como servidor Web apresenta problemas de distribuição e balanceamento da carga de requisições entre os nós, bem como a distribuição das tarefas a serem executadas por cada nó e dos documentos as serem armazenados.

Este capítulo analisa a possibilidade de utilização de computadores baseados na arquitetura Beowulf e propõe técnicas de solução dos problemas mais comuns encontrados quando da tentativa de usar computadores dessa arquitetura como servidores Web. Para isso, é proposto um modelo que descreve o servidor, sendo também apresentados resultados de medidas de desempenho de um protótipo baseado no modelo proposto.

6.2. A arquitetura Beowulf

Computadores com a arquitetura Beowulf consistem em cpus ligadas por uma rede de comunicação. Na classificação de computadores paralelos, a arquitetura Beowulf se encaixa entre os Processadores Massivamente Paralelos (MPP) como o nCube e o Cray e os NOW's (Networks of Workstations) [Beowulf02].

Computadores com arquitetura MPP são geralmente maiores e bem mais caros. Porém, apresentam tempos menores de troca de informações entre processadores que os computadores baseados na arquitetura Beowulf. Utilizar uma NOW consiste apenas em aproveitar a capacidade de processamento ociosa de computadores potencialmente diferentes interligados por rede, apresentando o problema de necessitar de algoritmos eficientes em balanceamento de carga. Qualquer programa que rode em uma NOW irá rodar, no mínimo, tão bem em um Beowulf [Beowulf02]

O primeiro computador com arquitetura Beowulf foi montado por Thomas Sterling e Don Becker em 1994. Desde então, com a redução dos preços dos componentes de computadores e o aumento da capacidade de processamento dos PC's, a arquitetura Beowulf tem se tornado uma boa opção para se conseguir alta capacidade de processamento a baixo custo. Apesar da proposta inicial da arquitetura Beowulf ser

direcionada principalmente aos meios acadêmico e científico, a sua utilização em companhias que necessitam de solução envolvendo grande capacidade de processamento tem se tornado cada vez maior.

6.3. Modelo utilizado

A figura 6.3.1 apresenta o modelo proposto neste trabalho, utilizado na análise do desempenho do servidor Web. A direção das setas indica o fluxo de informações enviadas. Note que não estão representadas na figura 6.3.1 as interconexões entre os nós de processamento e os caches.

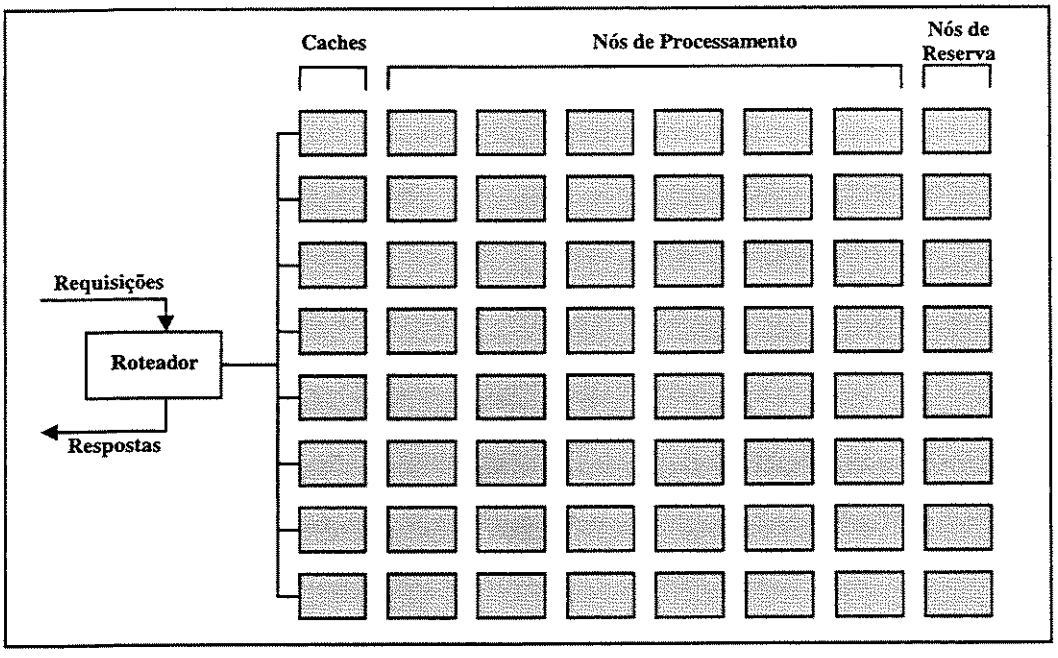


Fig. 6.3.1 - Modelo de servidor utilizado

Os principais elementos nesse modelo são o roteador, os caches e o conjunto de nós de processamento. Cada um desses elementos será discutido a seguir.

6.3.1. O roteador

O roteador tem a função de receber as requisições dos clientes e enviá-las para os elementos do sistema que vão tratá-las: os caches. Existem duas implementações possíveis para o elemento roteador: a utilização de um *gateway* ou um *switch*. Como os

endereços IP dos nós de processamento e dos caches são endereços de rede privada, a única maneira de acessá-los é através do roteador. Assim, o roteador deve ser capaz de responder a requisições HTTP feitas por clientes. A não ser que seja instalado um *switch* capaz de trabalhar como um Proxy, respondendo a requisições HTTP, deve ser usado um computador como roteador para receber essas requisições e repassá-las aos caches.

O *switch* tem a vantagem de trabalhar em altas velocidades podendo, muitas vezes, gerenciar várias conexões de rede. O problema em se usar um *switch* é que ele deve ser capaz de trabalhar como Proxy, fazendo com que esse elemento exija um maior investimento.

Usar um computador dedicado como roteador tem a vantagem de permitir um melhor direcionamento das requisições, visto que podem ser implementadas políticas mais inteligentes de distribuição de requisições. Porém, a replicação de um roteador implementado em um computador dedicado é mais difícil que em um roteador que utiliza um *switch*.

Além dos problemas de tolerância a falhas, o roteador torna-se um gargalo quando o número de requisições se torna muito alto. Isso ocorre porque cada requisição tem que ser processada pelo *gateway*, fazendo com que seja prejudicada a escalabilidade do número de requisições e processadores do cluster.

O problema da escalabilidade é diminuído quando usado um *switch* como roteador, pois o *switch* pode ser projetado de forma a ser otimizado para aumentar a taxa de requisições processadas e a velocidade de processamento de requisições.

No caso de utilizar um *switch* com capacidade de programação limitada, pode-se programá-lo para contatar nós predefinidos, onde serão iniciados processos que farão o roteamento das requisições aos caches.

Apesar das primeiras descrições da arquitetura Beowulf considerarem a utilização de *switches* como roteadores, a tendência é que sejam utilizadas principalmente as configurações que usem computadores dedicados como portas de acesso aos nós do Beowulf. Esses computadores dedicados são normalmente chamados de nós mestre (*master nodes*) e o aumento da sua utilização como roteadores acontece porque normalmente os interessados em construir um Beowulf dispõe de vários PC's e nenhum *switch* capaz de trabalhar como roteadores, que são normalmente caros. Além disso, o

computador dedicado normalmente permite uma melhor capacidade de programação que o *switch*.

6.3.2. Caches

Os caches do sistema são implementados usando os nós do Beowulf. Neste trabalho, considera-se que cada nó de processamento possui capacidades individuais de armazenamento em disco. Essa é uma das principais diferenças entre esse modelo e os modelos de servidores Web distribuídos, onde os caches são instalados fora da malha de servidores.

Os caches são colocados para economizar requisições feitas aos nós de processamento. Apesar de existirem técnicas para o cache de documentos dinâmicos, os caches são utilizados principalmente com documentos estáticos. Nesse modelo, o espaço em disco dos caches é usado para armazenar documentos estáticos requisitados recentemente aos nós de processamento.

No caso do servidor Web enviar muitos documentos estáticos ou raramente atualizados, a utilização de caches é recomendável. Por outro lado, se a maioria do tráfego corresponde a páginas geradas quando do processamento da requisição, a economia de requisições feita aos servidores não compensa o atraso causado pela inserção de caches no fluxo de dados.

6.3.3. Nós de processamento

Os nós de processamento são os processadores do Beowulf responsáveis pelo armazenamento de arquivos e geração de páginas dinâmicas.

Neste trabalho, considera-se que cada nó de processamento é responsável por um subconjunto dos arquivos armazenados no servidor. Uma maneira de decidir quais processadores são responsáveis por quais arquivos é calcular, usando uma função de *hash*, um número natural relativo a cada arquivo. Pode-se então usar esse número para decidir o número do nó de processamento responsável por aquele documento.

Os nós de processamento são também responsáveis pela geração de páginas dinâmicas, que são geradas quando do processamento das requisições.

Para gerar a página, o nó de processamento poderá utilizar-se de estruturas de páginas armazenadas em disco, dados tais como a data e hora atuais, parâmetros passados na pelo cliente através da requisição e dados armazenados em bancos de dados, locais ou não. Caso sejam utilizados bancos de dados distribuídos, estudos com relação à arquitetura e desempenho dos mesmos devem ser realizados. Em [Date86], é possível encontrar um bom texto introdutório sobre o assunto.

Nesse modelo, não existem restrições quanto ao software a ser usado para processar as requisições e enviar os documentos de resposta. Pode ser usado tanto um software de código aberto, tal como o Apache [Apache02] quanto qualquer outro software capaz de processar requisições e, se for o caso, gerar páginas dinâmicas. Essa independência com relação ao software utilizado ocorre porque, do ponto de vista do servidor, as requisições serão recebidas como se ele estivesse se comunicando diretamente com os clientes, sendo transparente a presença do roteador e dos caches.

6.4. Funcionamento do Sistema

O servidor pode trabalhar sob duas condições: funcionamento normal e reconfiguração do Beowulf. Cada uma dessas condições define um estado de funcionamento do Beowulf.

6.4.1. Funcionamento sob condições normais

É considerado que o sistema está trabalhando em condições normais quando está no estado de funcionamento normal. Esse é o estado inicial do Beowulf e o estado no qual espera-se que ele trabalhe na maior parte do tempo.

Nesse estado, requisições de clientes são endereçadas ao roteador. O roteador passa as requisições para um dos caches. Caso o documento seja estático e esteja armazenado no cache, ele é enviado ao roteador que o repassa ao cliente.

Caso a requisição seja por um documento dinâmico, o cache repassa a requisição ao nó de processamento responsável pelo documento requisitado. Para saber qual é o nó de processamento responsável por cada arquivo requisitado, o cache deverá também

utilizar a função de *hash* usada para distribuir os arquivos dentre os nós de processamento.

Caso a requisição seja por um documento estático não encontrado no cache, este fará uma requisição ao nó de processamento responsável pelo documento. O nó de processamento verifica se possui o documento ou não. Caso possua, envia-o para o cache que armazena uma cópia para si e envia uma cópia ao roteador, que a repassa ao cliente.

Quando do armazenamento do documento no cache, este verificará se existe espaço disponível. Em caso negativo, o cache terá que aplicar uma política de substituição para remover os documentos considerados com menor utilidade. Essa política de substituição pode ser escolhida de acordo com o perfil dos documentos armazenados nos nós de processamento. Por exemplo, caso existam documentos estáticos muito acessados, pode-se usar a política LFU, que retira do cache os documentos menos acessados, aumentando a chance de que documentos populares sejam encontrados no cache.

A escolha do cache pode ser feita usando um algoritmo simples, por exemplo, *round-robin* ou aleatório. Isso pode ser feito porque as requisições seguem uma distribuição Zipf [Breslau99], o que indica que os documentos mais acessados são muito mais acessados que os documentos menos acessados. Por isso, depois de um certo tempo de funcionamento, cada cache terá uma cópia dos documentos mais acessados, o que aumentará a disponibilidade de documentos populares, diminuindo o tempo de resposta do servidor.

6.4.2. Reconfiguração do Beowulf

Quando existe uma alteração no número de nós de processamento do sistema, devido ao acréscimo de nós para proporcionar um melhor desempenho ou decréscimo de nós por falha ou outro motivo, o Beowulf passa a operar no estado de reconfiguração. O processo de reconfiguração deve ser iniciado manualmente pelo administrador do sistema ou automaticamente, por exemplo, quando detectada uma falha em um nó de processamento, no caso de não existirem nós de reserva. Os nós de reserva são descritos na seção 6.5.2.

Quando é iniciada a reconfiguração do sistema, são informados aos caches a alteração de estados, o novo número de nós e os endereços IP dos nós envolvidos no processo.

Ao ser iniciada a reconfiguração, é iniciado um processo responsável por mover todos os arquivos para as suas novas posições. Esse processo pode ser configurado para utilizar apenas a capacidade de processamento e a banda passante ociosas nos nós de processamento.

Para informar aos caches da mudança de estado, deve ser implementado um algoritmo de difusão confiável da mensagem de troca de estado. Além disso, o processo responsável por mover os arquivos durante a reconfiguração deve ser implementado considerando as possibilidades de falha nesse processo.

No estado de reconfiguração, os únicos elementos que passam a agir de forma diferente são os caches. Considerando que antes do início da reconfiguração existiam **N** nós de processamento e que o sistema está sendo remodelado para utilizar **M** nós, sendo

hash(A) mod N - o índice do nó de processamento responsável pelo arquivo **A** antes da reconfiguração e

hash(A) mod M - o índice do nó de processamento responsável pelo arquivo depois de encerrada a reconfiguração, temos o algoritmo da listagem 6.4.1.

```
Algoritmo procura_durante_reconfiguração
Início
   Se Copia ( hash(A) mod N, A) então
      Envia A para roteador
   Senão
      Se Copia ( hash(A) mod M, A) então
         Envia A para o roteador
      Senão
         Esse arquivo não existe no servidor
      Fim Se
   Fim Se
```

Listagem 6.4.1 – Algoritmo de procura durante processo de reconfiguração.

A função *Copia* procura um arquivo em um nó de processamento identificado por um número. Obviamente, é necessário que o cache mantenha uma tabela relacionando o

índice do nó de processamento e seu endereço IP. Se o arquivo for encontrado, ele é copiado para o cache.

Durante a reconfiguração, os arquivos deverão ser movidos entre os nós de processamento para a posição correta dentro do Beowulf, considerando o novo número de nós. O algoritmo *Procura_durante_reconfiguração* procura primeiro o arquivo na sua posição antes da reconfiguração. Ele será encontrado caso ainda não tenha sido movido. Em caso contrário, o arquivo, se existir no servidor, já terá sido movido para a nova posição, onde será encontrado.

Depois que todos os arquivos já estiverem em suas respectivas posições finais, são enviadas mensagens aos caches avisando do fim da reconfiguração. Os caches, por sua vez, mudam o estado para o de operação normal.

6.5. Discussão sobre características do sistema

Os critérios de avaliação do sistema considerados neste trabalho serão a escalabilidade, o nível de tolerância a falhas e o desempenho do sistema em operação normal e durante o processo de reconfiguração. Cada um desses aspectos é discutido nas seções a seguir.

6.5.1. Escalabilidade

A escalabilidade de um sistema formado por múltiplos processadores é a medida da capacidade desse sistema em aumentar a sua capacidade de processamento (diminuir o tempo de resposta, no caso de um servidor Web), proporcionalmente ao aumento do número de processadores [Kumar94]. A escalabilidade reflete a habilidade de um sistema em usar efetivamente um eventual ganho de recursos.

Um sistema é dito escalável se é possível manter o desempenho desse sistema, perante o aumento da carga de trabalho com um aumento proporcional do número de processadores.

No caso do sistema descrito neste trabalho, o grande problema em relação a escalabilidade é o roteador. Isso acontece porque um único elemento deve ser capaz fazer toda a intercomunicação entre os clientes e o servidor. Porém, pode-se pensar no roteador

como sendo formado de vários elementos que dividem a carga de requisições entre si. Essa divisão pode ser feita, por exemplo, através da atribuição de múltiplos endereços IP associados a um único nome. Nesse caso, a divisão da carga seria feita por servidores de DNS.

Considerando que os nós do Beowulf são interligados por *switches*, o que favorece a comunicação ponto a ponto, tanto os caches quanto os nós de processamento podem ser considerados escaláveis. Isso porque tanto o número de caches, visando o aumento da disponibilidade dos documentos populares, quanto o número de nós de processamento podem ser aumentados indefinidamente, desde que a estrutura de comunicação entre os nós seja também escalável, o que pode ser conseguido usando-se *switches*.

6.5.2. Tolerância a falhas

Falhas são desvios do comportamento do sistema para situações indesejadas. Em servidores Web, falhas muitas vezes causam grandes prejuízos devido à interrupção ou degradação da qualidade dos serviços oferecidos.

Falhas no sistema são resultado de faltas em componentes de software ou hardware que formam o sistema. As faltas são divididas em três categorias [Burns96]:

1 – Faltas transientes: são as faltas que se iniciam em um tempo específico, permanecem durante algum tempo e desaparecem. Um exemplo é a interferência eletromagnética em um dispositivo de comunicação.

2 – Faltas permanentes: são faltas que aparecem e persistem até serem reparadas. Por exemplo, a queima de um processador é uma falta permanente.

3 – Faltas intermitentes: são faltas transientes que se repetem, em intervalos fixos ou não. Um exemplo é a interrupção do funcionamento de um dispositivo de hardware devido a sobre-aquecimento. O dispositivo pára de funcionar, resfria e volta a funcionar novamente, parando novamente depois de um período, pelo mesmo motivo.

O nível de tolerância a falhas apresentado por um sistema computacional representa o nível de garantia de manutenção da qualidade de serviço oferecida pelo sistema na presença de falhas. Nesta seção, são discutidos resumidamente, aspectos de

tolerância a falhas do servidor Web implementado em um computador com arquitetura Beowulf. Para isso, são discutidos os aspectos de tolerância a falhas individuais dos componentes que formam o servidor: o roteador, os caches e os nós de processamento.

6.5.2.1. O roteador

Esse é o elemento mais sensível a falhas dentre os que formam o servidor. Isso ocorre porque o roteador de requisições funciona como uma interface entre o servidor e os clientes e, caso ele pare de funcionar, os clientes não mais terão acesso aos serviços.

A melhor maneira de introduzir capacidade de tolerância a falhas nesse elemento é a colocação de um ou mais elementos redundantes a ele. Essa estratégia consiste na instalação de equipamentos que não são usados durante as operações normais do roteador. Eles checam a todo instante o funcionamento correto do roteador principal e, caso detectem uma falha deste, assumem a operação, evitando a interrupção do oferecimento dos serviços.

O equipamento redundante ficará responsável pela interface entre os clientes e o servidor até que o roteador principal volte a funcionar. Isso pode levar pouco tempo, no caso de faltas transientes ou até vários dias, no caso de faltas permanentes.

Essa abordagem leva a sistemas mais confiáveis. Porém, note que o sistema não é completamente tolerante a falhas, pois os equipamentos redundantes também podem falhar. A capacidade de tolerância a falhas pode então ser aumentada adicionando-se tantos elementos redundantes quantos se achar necessário para atingir um nível considerado satisfatório de tolerância a falhas. A estratégia de colocação de vários elementos redundantes é chamada Redundância N-Modular.

6.5.2.2. Os caches

Para que os caches apresentem uma alta tolerância a falhas, basta que o elemento roteador seja programado para que, caso um cache pare de responder, elimine-o da lista de caches disponíveis. No caso em que, por motivo de simplicidade e desempenho do roteador não existirem operações de gerenciamento da lista de caches, o roteador deve ser configurado para marcar um tempo de *timeout* após enviar um pedido de arquivo a um cache, depois do qual será escolhido um novo cache para enviar a requisição.

Dessa forma, a ocorrência de falha em um cache fará com que sua carga de trabalho seja redirecionada para os demais caches, o que pode aumentar o tempo médio de resposta dos caches, que terão que processar um número maior de requisições. Essa característica de diminuição gradual da capacidade de serviço é chamada de degradação gradual ou *fail soft.*

6.5.2.3. Os nós de processamento

Se for considerado que a carga de trabalho submetida a um nó de processamento em falha pode ser dividida entre os outros nós de processamento, pode-se considerar que o conjunto dos nós de processamento apresenta características de degradação gradual de desempenho perante falhas. Porém, é importante lembrar que cada nó de processamento é responsável por uma parte dos arquivos no servidor. Nesse caso, a estratégia de usar redundância parece mais uma vez ser a mais apropriada.

À primeira vista, pode-se pensar em colocar um nó de processamento redundante a cada nó de processamento em operação no Beowulf. Porém, uma estratégia melhor seria a de criar uma classe de nós de reserva cujos elementos possam vir a substituir qualquer nó em falha. Para isso, é necessário um sistema detecção de falhas, que configure corretamente o nó de reserva para que ele assuma o lugar do nó em falha e envie mensagens aos outros nós avisando da mudança. Esse sistema é implementado pelo processo Mestre de Execução, que é apresentado na seção 6.7.4.

Uma outra estratégia que pode ser implementada é usar um repositório de arquivos. Esse repositório pode ser implementado em um computador que não faça parte do Beowulf mas que possua uma alta capacidade de armazenamento de arquivos. No caso de uma reconfiguração, os arquivos faltantes a um NP serão obtidos a partir desse repositório.

Para que uma requisição não tenha que esperar que o sistema seja reconfigurado após uma falha antes de ser respondida, é possível fazer com que cópias dos arquivos sejam distribuídas por nós que não sejam os devidos responsáveis por aquele arquivo.

Por exemplo, supondo que um arquivo A seja de responsabilidade do nó N e o número de cópias redundantes dos arquivos seja 2, os nós N+1 e N+2 terão uma cópia do arquivo A. Isso permite que o cache tenha NP's alternativos para os quais ele pode enviar

as requisições ao arquivo A que não foram respondidas por N. Essa estratégia diminui o tempo médio de resposta das requisições no caso de falha de um NP, mas exige um espaço de armazenamento muito maior no NP.

6.5.3. Desempenho

Durante o estado de funcionamento normal, o tempo de resposta, **TR**, a uma requisição é dado por:

$$\boxed{TR = TPR + TTRRC + TPC + TTRCN + TPN + TTANC + TTACR}, \text{ sendo}$$

TR: Tempo de resposta do sistema a uma requisição;
TPR: Tempo de processamento da requisição pelo roteador;
TTRRC: Tempo de transmissão da requisição do roteador para o cache;
TPC: Tempo de processamento da requisição pelo cache;
TTRCN: Tempo de transmissão da requisição do cache para o nó de processamento;
TPN: Tempo de processamento do nó de processamento;
TTANC: Tempo de transmissão do arquivo do nó de processamento para o cache;
TTACR: Tempo de transmissão do arquivo do cache para o roteador.

Essa expressão mostra como pode ser calculado o valor do **TR** médio esperado. Esse valor representa o tempo desde o término do recebimento da requisição pelo roteador até o início da transmissão do arquivo ao cliente.

No caso dos documentos estáticos, o valor de **TR** passa a sofrer um impacto muito menor dos valores de **TTRCN**, **TPN** e **TTANC**. Isso porque existe uma grande probabilidade de uma cópia do arquivo já estar armazenada no cache, economizando o acesso ao nó de processamento.

Por motivo de comparação, é apresentada a seguir a expressão do cálculo de **TR** para um servidor convencional, composto por apenas um computador, multiprocessado ou não:

$$\boxed{TR = TPR + TAA}, \text{ sendo}$$

TPR: Tempo de processamento da requisição;

TAA: Tempo de acesso ao arquivo.

Apesar de **TR** em um servidor convencional ser menor, sob alguns aspectos, usar um Beowulf como servidor apresenta algumas vantagens.

A primeira vantagem é a escalabilidade, visto que a capacidade do Beowulf pode ser expandida indefinidamente, enquanto que o servidor convencional está limitado ao melhor computador possível de ser adquirida com a tecnologia atual. Além disso, o Beowulf pode ter a sua capacidade aumentada com a compra de novos nós enquanto que fica difícil realizar um *upgrade* no servidor convencional sem substituir grande parte dos componentes de hardware.

A capacidade de tolerância a falhas é outra grande vantagem apresentada pelo Beowulf. Enquanto o servidor convencional precisa de um servidor redundante com capacidade compatível e potencialmente caro, a tolerância a falhas no Beowulf pode ser conseguida com a redundância de elementos mais baratos, sendo que os gastos em equipamentos redundantes será muito menor que os gastos em equipamentos efetivamente ativos.

Outra vantagem importante seria vista no caso do tempo de processamento das requisições pelo nó de processamento necessitar de um tempo de computação significativo. Nesse caso, a divisão da carga de trabalho entre os nós de processamento gera uma capacidade potencialmente infinita, o que não é possível de conseguir com um servidor composto de um único computador.

É importante ainda a inclusão de alguns comentários sobre os tempos de processamento e envio de requisições e arquivos, que influem no **TR** de um Beowulf.

Considerando que a rede que interliga os elementos do Beowulf seja de alta capacidade (Fast Ethernet, por exemplo) e que utilize um ou mais *switches*, o que permite o paralelismo ponto a ponto real nas transmissões, o atraso devido ao *overhead* da intercomunicação dos elementos é pequeno quando comparado ao tempo de envio de arquivos do roteador ao cliente, o que normalmente é feito através de um rede de longa distância, que possui um taxa de transmissão menor que a rede local que interliga os elementos do Beowulf.

Durante a reconfiguração, o **TR** pode sofrer um aumento devido ao aumento do tempo para a procura do arquivo. O impacto da reconfiguração no tempo de resposta é diminuído devido à utilização dos caches para enviar cópias de documentos estáticos populares aos clientes, economizando a requisição aos nós de processamento que, por sua vez, contam com mais recursos para executar o processo de reconfiguração. Quanto maior o percentual de documentos estáticos e maior a eficiência dos caches, menos requisições forçarão a realização de duas consultas aos nós de processamento durante a reconfiguração.

6.6. Mensagens trocadas pelos elementos formadores do servidor

Para que seja possível a troca de dados sobre o sistema, foi criado um conjunto de mensagens que são trocadas entre os elementos formadores do servidor Esta seção apresenta essas mensagens. A grande maioria delas tem, como primeiros caracteres, os caracteres "!" e "@", significando que aquela é uma mensagem de controle interno do servidor. A seguir, são colocados o identificador do tipo da mensagem e os parâmetros, quando necessários.

Definições de portas utilizadas nos elementos (!@p)	
0 porta(int)	Definição da porta dos roteador
1 porta(int)	Definição da porta dos cache
2 porta(int)	Definição da porta dos NP's
3 porta(int)	Definição da porta dos NR's
4 porta(int)	Definição da porta do processo mestre

Tabela 6.6.1 – Mensagens de definição de portas

Definição das funções dos elementos (!@f)	
0 idNó(int) IP(string) RankMestre(int)	Função de roteador
1 idNó(int) IP(string) RankMestre(int)	Função de cache
2 idNó(int) IP(string) RankMestre(int)	Função de NP
3 idNó(int) IP(string) RankMestre(int)	Função de NR

Tabela 6.6.2 – Mensagens de definição das funções dos elementos

Mensagens de controle e configuração (!@c)	
0	Encerrar processos filhos e a si próprio
1	Encerrar processos filhos
2	Iniciar processos filhos
3 id(int) ip(string)	Definição de **próprio** ID e informação de IP
4 nCópias(int)	Configuração do número de cópias redundantes
5 id(int) ip(string)	Mestre de execução em id
6 id(int) status(int)	Informação de tudo OK (ping)
8 tempo(int)	Configuração do intervalo do timer
9	Iniciar processo mestre

Tabela 6.6.3 – Mensagens de controle e configuração

Informações sobre arquivos (!@a)	
0 nomeLocal(string) nomeServidor(string) tamanho(int) ip(string) porta(int)	Copiar arquivo

Tabela 6.6.4 – Mensagens informações sobre arquivos

Controle de reconfiguração (!@r)	
0 id(int)	Novo roteador (A partir de NR)
1 id(int)	Novo cache (A partir de NR)
2 id(int)	Novo NP (A partir de NR)
3 id(int)	Novo NR (Não implementado)
4 id(int)	Eliminar elemento
5	Reconfiguração concluída

Tabela 6.6.5 – Mensagens para controle da reconfiguração

Outros comandos	
!@i	Imprimir informações na tela
GET /!	Página com configurações – Interface Web

Tabela 6.6.6 – Outras mensagens

6.6.1. Mensagens definidas em cada elemento

A tabela 6.6.7 apresenta os tipos de mensagens definidas em cada elemento formador do servidor. A lista de mensagens dos NR's está vazia porque, apesar de ser definido como um tipo distinto de processo na definição do sistema, na implementação suas atividades são realizadas pelo processo de controle de execução, não sendo necessário iniciar os processos de reserva.

Elemento	Recebe	Envia
Controle	p, f, c, a, r	c(6)
Mestre de Configuração		p, f, c, a
Mestre de Execução	c(6), r, outros	p, r, a
Roteador	Requisição, r	Requisição
Cache	Requisição, r	Requisição e Resposta
NP	Requisição	Resposta
NR		

Tabela 6.6.7 – Mensagens definidas em cada elemento do servidor

6.7. Inicialização do servidor

Essa seção apresenta a seqüência de passos considerada necessária para a inicialização do servidor a partir de um computador com arquitetura Beowulf.

6.7.1. Considerações iniciais

A primeira condição a ser satisfeita para utilizar um Beowulf como um servidor Web é que exista um elemento do Beowulf que é passível de ser acessado pelos futuros clientes. Esse será o elemento utilizado como roteador.

Devem ser instaladas no Beowulf ferramentas de software que permitam iniciar processos em cada um dos nós. Um exemplo de ferramenta que pode ser utilizada é o *Message Passing Interface* – MPI [MPI02]. Caso o MPI não esteja instalado, foi criada uma ferramenta, chamada *exechost*, capaz de iniciar os processos nos nós necessários. O apêndice A apresenta maiores detalhes sobre o *exechost*.

O modelo do *script* usado para iniciar os processos usando a ferramenta *exechost* é apresentado e explicado na listagem 6.7.1. O arquivo de configuração é formado por quantas linhas tantas forem necessárias com o modelo da listagem 6.7.1.

```
Exechost IP Porta Executável MestreConf Diretório PortaControle
.
.
```

Listagem 6.7.1 – Modelo de *script* usado para iniciar processos

Os parâmetros necessários para a execução do *exechost* são os seguintes:

- **IP:** Endereço IP da máquina onde será iniciado o processo
- **Porta:** Porta na qual o exechost está ouvindo
- **Executável:** Nome e caminho para o arquivo executável do processo de controle
- **MestreConf**: Iniciar mestre de controle ?
- **Diretório:** Diretório de trabalho para o servidor
- **PortaControle:** Porta por onde o processo de controle receberá instruções do mestre

Inicialmente, considera-se que existe um arquivo texto informações sobre os arquivos que serão servidos. Esse arquivo será distribuído aos nós de processamento e reserva de modo a que eles consigam decidir por quais arquivos eles serão responsáveis. Esse arquivo pode conter o endereço IP de um servidor de arquivos que contém uma cópia de reserva dos arquivos.

O modelo do arquivo com informações sobre os documentos que serão servidos é apresentado na listagem 6.7.2.

```
NomeLocal(string) NomeServidor(string) Tamanho(int) IP(string) Porta(int)
.
.
```

Listagem 6.7.2 – Modelo do arquivo de informações sobre documentos servidos

6.7.2. Início da execução dos processos de controle

Considere a representação do beowulf de 16 nós mais um roteador apresentada na figura 6.7.1, onde o elemento roteador apresenta-se destacado.

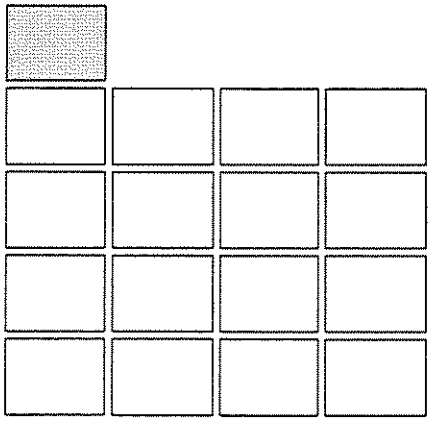


Fig. 6.7.1 – Beowulf de 16 nós usado

É iniciado um processo de controle em cada um dos nós do Beowulf, com as restrições apresentadas na seção 6.7.1. Caso seja possível, um processo de controle é iniciado também no roteador. Caso o roteador seja um *switch*, este deve ser programado de forma a permitir a escolha dos caches.

A figura 6.7.2 apresenta um diagrama representando um Beowulf onde os processos de controle foram iniciados.

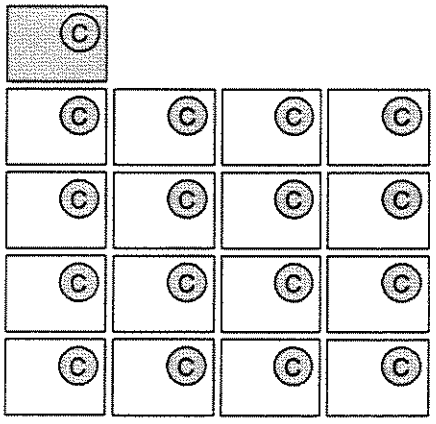


Fig. 6.7.2 – Processos de controle iniciados em cada nó do beowulf

6.7.3. Processos de roteamento e mestre

Ao iniciar os processos de controle, um deles é encarregado de iniciar um processo chamado "Mestre de Configuração". Esse processo é responsável por ler um

arquivo que contem as configurações do sistema. Essas configurações são apresentadas a seguir:

- Portas – Configuração de quais portas serão utilizadas por cada elemento do sistema.
- Elementos – Função, número de identificação e IP de cada elemento do sistema.
- Cópias redundantes – Número de cópias redundantes dos arquivos a serem armazenadas.
- Parâmetros dos caches – Configurações da política de substituição, tamanho máximo dos arquivos que podem ser armazenados, espaço máximo disponível e prazo de expiração da validade dos arquivos no cache (*time to live*).
- Tempo de ping – Freqüência na qual os elementos trocarão informações sobre o funcionamento de forma a detectar a presença de falhas.
- Mestre de execução – Identificação do nó onde o processo "Mestre de execução" será iniciado.
- Prioridade para iniciar o Mestre de Execução – Valor da prioridade de cada processo de controle para iniciar o Mestre de Execução caso detecte que ele está em falha.

A figura 6.7.3 apresenta um diagrama representando um Beowulf onde um dos processos de controle inicia o processo Mestre de Configuração (M).

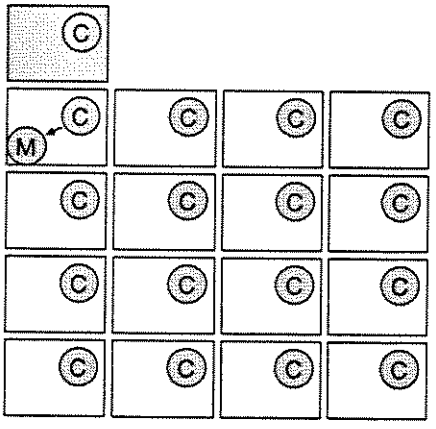


Fig. 6.7.3 – Início dos processos de roteamento e mestre de configuração

6.7.4. Caches, Nós de Processamento de Nós de Reserva

Após analisar o arquivo de configuração, o Mestre de Configuração envia as informações lidas aos processos de controle. Essas informações são enviadas utilizando mensagens de controle predefinidas, apresentadas na seção 6.6.

O modelo do arquivo de configuração é apresentado na listagem 6.7.3.

```
# Definições de portas
prouter Porta(int) #Definição da porta dos roteadores
pcache Porta(int)  #Definição da porta dos caches
pnp Porta(int)     #Definição da porta dos NP's
pmestre Porta(int) #Definição da porta do processo mestre

# Definições das funções
router id(int) ip(string) rankMestre(int) #O nó "id" trabalhará como router
cache id(int) ip(string) rankMestre(int)  #O nó "id" trabalhará como cache
np id(int) ip(string) rankMestre(int)     #O nó "id" trabalhará como NP
nr id(int) ip(string) rankMestre(int)     #O nó "id" trabalhará como NR
mestre id(int) ip(string)                 #local para mestre de execução
.
.
```

Listagem 6.7.3 – Modelo do arquivo de configuração de elementos

As últimas mensagens enviadas instruem aos processos de controle que iniciem seus processos filhos, de acordo com a configuração que eles receberam. Nesse momento, é iniciado o processo Mestre de Execução, que é responsável por monitorar o funcionamento do sistema, através do envio e recepção de mensagens de status a todos os elementos do sistema.

A figura 6.7.4 apresenta um diagrama onde os processos de controle (C) iniciaram os processos roteadores (R), servidores de cache (Ch), nós de processamento (NP) e nós de reserva (NR). Foi também iniciado o processo Mestre de Execução (E). Pode-se notar que foram iniciados caches em 4 processadores, nós de processamento em 10 e nós de reserva em 2.

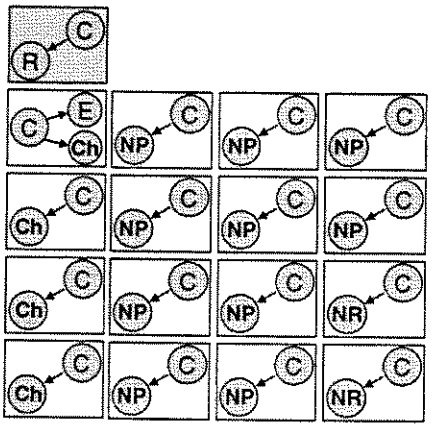


Fig. 6.7.4 – Todos os processos criados

A tabela 6.7.1 apresenta as informações passadas a cada tipo de processo. Essas são as informações necessárias para a execução da função de cada tipo de elemento.

Roteador	- Endereços IP dos nós onde são executados os processos servidores de cache.
Caches	- Tamanho do espaço destinado a *caching* de documentos - Política de substituição a ser utilizada - Endereços IP e identificadores dos nós de processamento, para que os caches possam calcular qual NP é responsável por cada documento.
Nós de Processamento	- Número de NP's - Lista e localização dos arquivos. Cada NP copia os seus arquivos para o disco local. Os arquivos podem estar em um repositório inicial, acessível via NFS. - No caso de usar um servidor como o Apache, o processo de controle copia os arquivos e inicia o Apache.
Nós de Reserva	- Aguarda instruções de assumir o papel de um NP ou Cache, caso ocorra alguma falha.

Tabela 6.7.1- Configurações de cada tipo de processo

6.7.5. Falhas durante a inicialização

A detecção de falhas é feita usando a comunicação entre os processos de controle. Como existe uma troca de mensagens de controle entre o processo Mestre de Execução e os processos de controle deve em cada intervalo fixo, o processo mestre pode detectar que um nó parou de responder, fazendo com que o nó de reserva imediatamente seja acionado para substituir o nó em falha.

Caso o nó em falha seja um cache, o processo Mestre de Execução, depois de notificar o nó reserva de suas novas atribuições, deverá informar ao roteador da substituição do IP do cache defeituoso. A figura 6.7.5 exemplifica o caso em que existem falhas simultâneas em dois elementos do sistema.

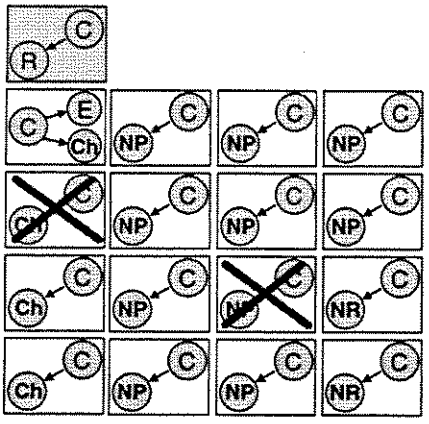


Fig. 6.7.5 – Falhas simultâneas em um cache e em um NP

Caso o nó em falha esteja trabalhando como NP, o processo mestre pode optar por três estratégias diferentes:

1 – Entrar em processo de reconfiguração e continuar o trabalho sem um NP. Essa seria uma estratégia recomendada caso a carga de trabalho seja passível de ser suportada pelos NP's restantes e caso não existam NR disponíveis.

2 – Entrar em processo de reconfiguração enquanto um NR se prepara para trabalhar como um NP. Quando o novo NP estiver pronto, inicia-se um novo processo de reconfiguração, que fará com que o novo NP passe a fazer parte do conjunto de NP's ativos.

3 – Preparar um NR para substituir o NP em falha. Durante o tempo de preparo do novo NP, as requisições que deveriam ser respondidas pelo NP em falha são perdidas. O impacto negativo dessa estratégia pode ser diminuído com a utilização de cópias redundantes.

Um outro problema que pode acontecer é quando ocorre uma falha no nó que executa o processo Mestre de Execução. Nesse caso, antes de um NR entrar no lugar do nó em falha, um processo de comando deve iniciar um novo processo Mestre de Execução. Para decidir qual processo de comando deve iniciar o novo processo Mestre de Execução, os processos de controle utilizam-se de seus valores de prioridade para início do Mestre de Execução. Como os valores de prioridade são diferentes, cada processo precisa esperar um tempo diferente desde a última comunicação com o processo Mestre de Execução para iniciar um novo Mestre de Execução. Assim, o que tiver a maior prioridade inicia o novo Mestre de Execução, enviando mensagens informando aos outros processos de controle da nova posição do Mestre de Execução.

A figura 6.7.6 apresenta o caso onde um processo de controle iniciou o novo processo Mestre de Execução.

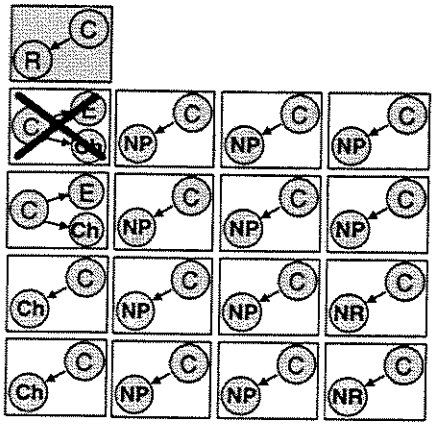


Fig. 6.7.6 – Falha em um processador executando o processo Mestre de Execução

6.8. Controle do servidor e aspectos de segurança do modelo

Esta seção apresenta ferramentas utilizadas para facilitar a controle do servidor pelo administrador do sistema. São feitas ainda breves considerações sobre estratégias de segurança contra ataques que podem ser implementadas pelo sistema.

6.8.1. Controle do servidor

O controle do servidor pode também ser feito através de instruções enviadas de um aplicativo para o processo Mestre de Execução através do roteador. A figura 6.8.1 apresenta um esquema dessa comunicação entre o aplicativo e o Mestre de Execução.

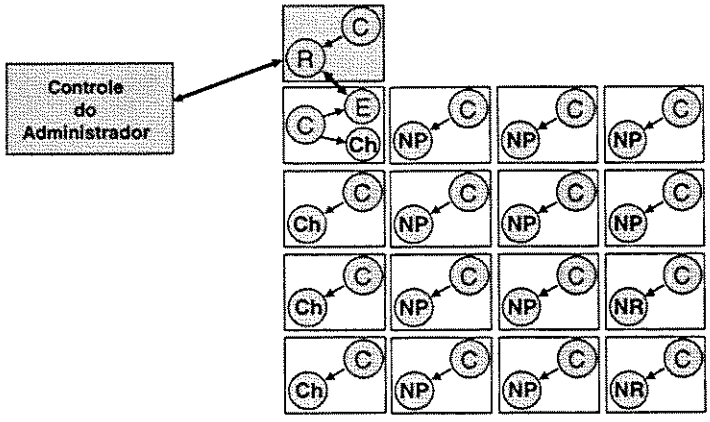


Fig. 6.8.1 – Controle do Administrador interagindo com o processo mestre de execução

O aplicativo de controle do administrador permite ao administrador do sistema configurar o sistema da forma que desejar. Esse aplicativo, desenvolvido em Borland Kylix para facilitar a construção de uma interface X-window troca informações com o processo mestre diretamente, caso o aplicativo esteja sendo executado em um processador que tenha acesso aos nós do beowulf ou através do processo roteador.

Outra forma de acessar informações no processo Mestre de Execução é através de um Web browser. Basta que o browser tente acessar um arquivo especial no servidor que os caches entenderão que a requisição deve ser direcionada ao processo Mestre de Execução e não a um NP. Assim, fica possível configurar e obter informações do sistema utilizando apenas um browser.

Todo processo no servidor sabe o endereço IP do nó onde o processo mestre está sendo executado. Assim, quando o aplicativo de controle do administrador é iniciado, basta que ele pergunte a qualquer processo de controle onde está sendo executado o processo mestre para saber como contatá-lo.

6.8.2. Aspectos de segurança

A capacidade de instalação de mecanismos de segurança simples e eficientes é uma das grandes vantagens da arquitetura Beowulf. Isso porque o único ponto do servidor visível de fora é o roteador. Isso permite que estratégias de segurança sejam implementadas apenas no roteador.

Um cuidado especial com aspectos de segurança deve ser considerado quando da implementação do controle externo do administrador. Porém, os riscos podem ser minimizados com a utilização de senhas e criptografia dos dados trocados.

6.9. Implementação e testes do sistema

Para analisar aspectos de desempenho e escalabilidade do modelo de servidor Web proposto neste trabalho foi implementado em um computador com arquitetura Beowulf pertencente ao Instituto de Computação da Unicamp. Esse Beowulf possui 66 nós com 256 MB de RAM cada, rodando o sistema operacional Linux Red Hat. A linguagem utilizada na implementação foi C, sendo utilizado o compilador GNU GCC.

A tabela 6.9.1 apresenta algumas características relativas aos experimentos realizados.

Parâmetro	Valor
Quantidade de arquivos distribuídos	20000
Tamanho dos arquivos (média)	100 Kbytes
Capacidade da rede de intercomunicação dentro do Beowulf	100 Mbps
Percentual de documentos dinâmicos	50 %
Desempenho dos servidores de cache (Hit Ratio)	60 %
Capacidade máxima de armazenamento dos caches	5 Mbytes

Tabela 6.9.1 – Valores usados na simulação da utilização do servidor

Taxas de 60% de *Hit Ratio* são relativamente fáceis de serem alcançadas pelos caches, considerando a pouca quantidade de arquivos armazenados no Beowulf e a distribuição Zipf das requisições a esses arquivos. Para gerar os documentos dinâmicos, o

NP criava uma página HTTP baseando-se em um arquivo do disco local escolhido aleatoriamente.

6.9.1. Experimentos realizados

As primeiras simulações realizadas destinaram-se a estudar a escalabilidade de um servidor Web implementado em um Beowulf. Foi então analisado o desempenho de Beowulfs formados por 4, 8, 16 e 32 nós, além de nós roteadores. A carga de requisições foi feita através de processos clientes iniciados em alguns nós do próprio Beowulf. Os nós onde processos clientes foram iniciados não eram os mesmos usados para formar o servidor.

A distribuição de funções dos nós é apresentada na tabela 6.9.2. Note que nessa experiência não foram iniciados nós de reserva. Nas configurações testadas, optou-se por ter o mesmo número de caches e NP's devido ao percentual de requisições dinâmicas (50%).

Número total de nós	Caches	NP's
4	2	2
8	4	4
16	8	8
32	16	16

Tabela 6.9.2 – Distribuição de funções das configurações analisadas

O gráfico da figura 6.9.1 apresenta o tempo médio de resposta a cada requisição de acordo com a taxa de requisições feitas ao servidor. Para medir o tempo médio de resposta às requisições, foram feitas 1000 requisições com cada taxa analisada.

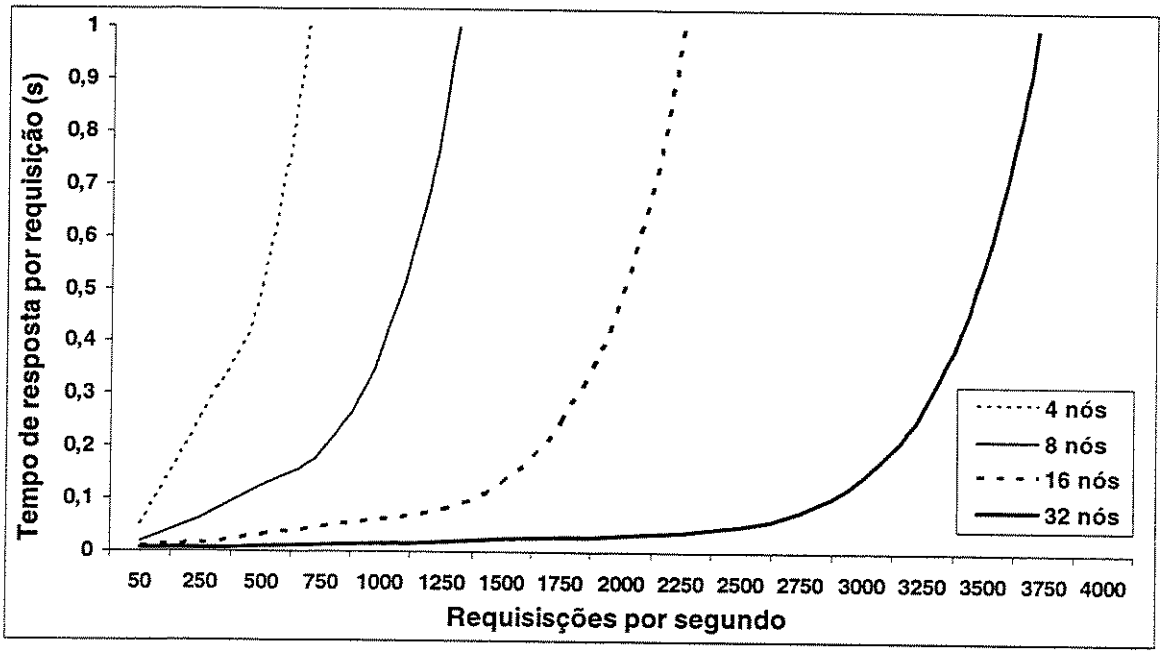


Fig. 6.9.1 – Tempo de resposta de acordo com a taxa de requisições aplicada

Através do gráfico da figura 6.9.1 é possível verificar que para cada tamanho de Beowulf existe uma taxa máxima de requisições, a partir da qual o tempo de resposta a cada requisição apresenta um crescimento muito grande. Pode ser observado que o sistema apresenta uma excelente escalabilidade, visto que o valor da taxa máxima de cada servidor é diretamente proporcional ao número de nós utilizados. Nessas experiências, foram utilizados 4 nós extras como roteador. Esse número de roteadores foi suficiente para impedir que os roteadores representassem um gargalo no fluxo de dados.

Quando aplicadas taxas de crescimento muito grandes, o percentual de requisições cujo tempo de resposta supera o tempo máximo de espera dos clientes, definido nos experimentos como de 20 unidades de tempo apresenta um grande crescimento. Isso pode ser observado no gráfico da figura 6.9.2.

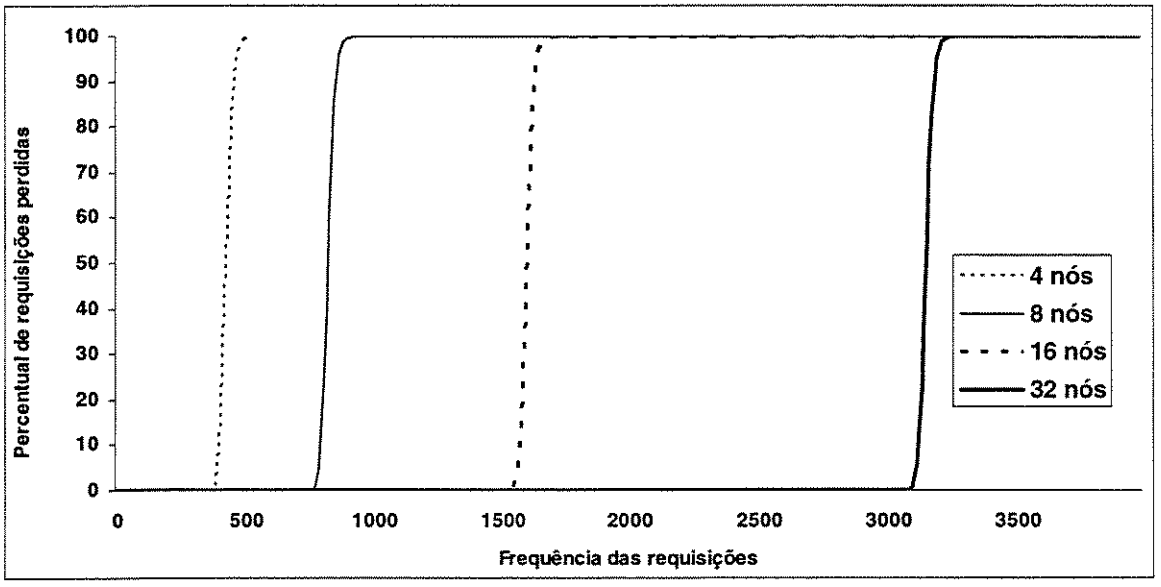


Fig. 6.9.2 – Percentual de requisições cuja resposta ultrapassou o tempo máximo

Pode-se notar que o percentual de requisições cujo tempo de resposta é igual ao tempo máximo de espera dos clientes cresce rapidamente para 100% assim que a taxa de requisições se aproxima de um limite máximo. Isso acontece porque o servidor não consegue calcular a quanto tempo a requisição está esperando na fila. Caso seja implementada uma estratégia que permita ao servidor descartar requisições cujo tempo de resposta ultrapassará um limite predefinido, é possível fazer com que o desempenho do servidor não seja tão degradado com taxas de requisições acima de determinado limite.

O último resultado de teste de desempenho apresenta os tempos de resposta durante a reconfiguração do Beowulf. A figura 6.9.3 apresenta a variação do tempo de resposta causada pela reconfiguração do sistema. No exemplo da figura 6.9.3, um Beowulf de 32 nós, dividido em 16 caches e 16 NP's sofreu um processo de reconfiguração onde um dos NP's foi desativado e os seus arquivos foram distribuídos dentre os NP's restantes. A reconfiguração ocorreu durante a aplicação de uma taxa de requisições constante de 500 requisições por segundo. Nesse caso, o sistema foi configurado para que cada arquivo armazenado possuísse uma única cópia redundante além da cópia armazenada no servidor responsável por ele.

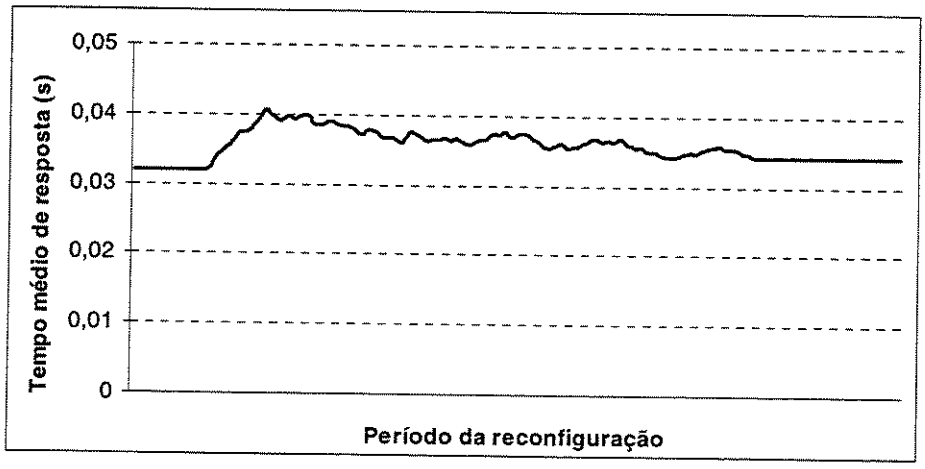


Fig. 6.9.3 – Impacto da reconfiguração no tempo médio de resposta

Note que depois da reconfiguração, o tempo médio de resposta aumentou sutilmente. Isso ocorreu devido ao decréscimo no número de NP's do sistema. Testes com servidores de outros tamanhos mostraram que o impacto da reconfiguração no tempo médio de resposta das requisições depende da proporção entre o número de nós reconfigurados e o número total de nós.

6.10. Conclusões sobre um Beowulf como servidor Web

O estudo apresentado neste capítulo mostrou que a arquitetura Beowulf possui características que podem ser utilizadas na implantação do modelo de servidor Web de alto desempenho apresentado. Porém, os resultados obtidos com pesquisa sobre a viabilidade de usar um Beowulf como servidor Web vão além da afirmação de que é possível construir um servidor de alto desempenho em um computador com essa arquitetura.

Um resultado interessante foi obtido através da divisão dos nós do Beowulf de forma que cada um fosse especializado em uma tarefa. Isso permite concentrar esforços de projeto em cada nó de acordo com a sua utilização futura. A utilização de nós de reserva, capazes de assumir o lugar de qualquer tipo de nó em falha, mostrou-se importante para prover capacidades de tolerância a falhas sem precisar que um grande número de nós fiquem ociosos na espera pelas eventuais falhas.

A utilização de caches de servidor dentro do próprio servidor mostrou-se bastante interessante. Além de prover o serviço de cache de arquivos, o que diminui drasticamente a carga de requisições a arquivos estáticos feitas aos nós de processamento, os caches podem ser utilizados para fazer balanceamento de carga e permitir total transparência aos programas servidores de arquivos nos nós de processamento, que não precisam ser modificados para operar em um nó do Beowulf.

O modelo proposto apresenta boa escalabilidade e tolerância a falhas, sendo uma de suas principais características a capacidade de executar a reconfiguração dos nós do Beowulf sem interromper o fornecimento de arquivos requisitados e sem apresentar muita degradação da capacidade do servidor.

Por último, este estudo mostrou que é possível coordenar os processos que fazem parte do servidor através de um protocolo simples e que não acrescenta grande *overhead* à rede que interliga os nós. Além disso, os próprios processos de controle e gerenciamento do servidor são bastante simples e também não apresentam grande *overhead* aos processadores do Beowulf.

Capítulo 7

Conclusões gerais e sugestões para trabalhos futuros

O estudo de redes hierárquicas de caches mostrou que a cooperação é uma boa estratégia para aumentar o desempenho de redes de caches para Web. Entretanto, o projeto da rede de caches cooperativos deve levar em conta o *overhead* causado pela troca de mensagens entre os caches, pois esse *overhead* pode vir a prejudicar o desempenho da rede.

O simulador paralelo de redes de caches provou ser uma ferramenta de grande utilidade na simulação de redes inteiras de caches distribuídos, hierárquicos e cooperativos. Sua capacidade de usar o poder de processamento de computadores paralelos para simular situações reais de uso de caches para Web é uma de suas grandes vantagens, pois permite realizar muitas simulações em menos tempo, gerando assim mais dados para serem analisados quando da necessidade de melhorar o desempenho de uma rede de caches ou projetar uma nova rede.

O GDE (Gerenciador Dinâmico de Espaço) mostrou ser uma boa opção para permitir a variação automática da QoS oferecida por sistemas de caches para Web, pois além de apresentar um desempenho comparável às melhores políticas de substituição existentes, necessita apenas de uma quantidade de recursos de memória e capacidade de processamento linear no número de opções de qualidade de serviço desejadas.

A estratégia de Passagem de Recomendação mostrou ser bastante eficiente para aumentar o desempenho das redes hierárquicas, sem causar grande *overhead* nessas redes

nem nos processadores dos computadores que as formam. Sua principal vantagem é permitir a utilização de qualquer política de alto desempenho nos caches de níveis mais baixos, sem se preocupar com a perda de características de popularidade dos arquivos requisitados pelos clientes, o que levaria à degradação do desempenho de caches de níveis hierárquicos superiores.

O estudo de servidores Web baseados em clusters de processadores foi direcionado principalmente a computadores com a arquitetura Beowulf. Essa arquitetura foi escolhida devido às suas características de usar clusters de computadores pessoais antigos interligados por rede para construir computadores paralelos de alto desempenho. Por suas características, essa arquitetura permite construir computadores paralelos de alto desempenho a baixo custo, o que tem gerado bastante interesse não só em universidades e centros de pesquisa como também em empresas privadas.

Foi avaliada a possibilidade da utilização de computadores Beowulf, proposto um modelo para o servidor Web, implementado e testado um protótipo. Esse estudo mostrou que a utilização de características da arquitetura Beowulf pode ser feita para atingir objetivos importantes a um servidor Web tais como desempenho, escalabilidade e tolerância a falhas. Uma das principais características do modelo proposto é a capacidade de executar a reconfiguração dos nós do Beowulf sem interromper o oferecimento dos serviços.

Assim, as principais contribuições deste projeto de pesquisa relacionam-se ao avanço do conhecimento de estratégias que podem ser utilizadas para aumentar o desempenho de sistemas de distribuição de documentos via Web. Esse conhecimento foi obtido através da leitura de trabalhos anteriores, seguida da realização de experimentos e análise dos resultados obtidos.

Para permitir o compartilhamento dos conhecimentos adquiridos com outros pesquisadores interessados nos temas estudados neste trabalho, foram escritos artigos científicos e submetidos a congressos nas áreas de Redes de Computadores e Sistemas Distribuídos.

A seguir, são listados os artigos publicados que foram produzidos com conhecimento adquirido nesse trabalho:

R. F. Brandão, R. O. Anido: "Aumentando o desempenho de caches para Web hierárquicos". *Anais do WPerformance 2003*, Campinas, SP, 08/2003.

R. F. Brandão, R. O. Anido: "Variação dinâmica da QoS oferecida por caches para Web". *Anais do 21o Simpósio Brasileiro de Redes de Computadores*, Natal, RN, 05/2003.

R. F. Brandão, R. O. Anido: "Uso de Beowulf's como servidores Web". *Anais do 20o Simpósio Brasileiro de Redes de Computadores*, Búzios, RJ, 05/2002. (Menção honrosa pela qualidade do trabalho).

R. F. Brandão, R. O. Anido: "A Parallel Simulator for Distributed and Cooperative Web Caches". *Proceedings of the 5th International Workshop on Distributed Simulation and Real Time Applications*, Cincinnati, Ohio, USA, pp 113-120, 08/2001.

R. F. Brandão, R. O. Anido: "Um Simulador Paralelo para Sistemas de Caches para Web". *Anais do 19o Simpósio Brasileiro de Redes de Computadores*, em CD-ROM, Florianópolis, SC, 05/2001.

R. F. Brandão, R. O. Anido: "Política de Substituição Dinâmica em Caches para Web". *Resumo nos anais do 18o Simpósio Brasileiro de Redes de Computadores*, Belo Horizonte, MG, 05/2000.

R. F. Brandão, R. O. Anido: "Cooperação entre caches para Web". *Anais do 17o Simpósio Brasileiro de Redes de Computadores*, pp 533-548, Salvador, BA, 05/1999.

Alguns outros trabalhos foram submetidos a congressos, sendo atualmente aguardadas as respostas sobre sua aceitação.

A primeira sugestão para trabalhos futuros é disponibilizar o simulador paralelo de redes de caches hierárquicos e cooperativos para pesquisadores com interesse nessa área.

Outra sugestão para trabalhos futuros é a de realizar testes com o sistema GDE em redes de caches distribuídos e cooperativos reais, onde a capacidade de variação dos pesos das políticas básicas permitirá avaliar diversas possibilidades de configurações de malhas de caches e testar a variação automática da QoS oferecida em redes de caches reais.

A verificação do impacto da utilização do mecanismo de passagem de recomendação em redes reais de caches para Web é também uma boa sugestão para o prosseguimento deste trabalho de pesquisa. O estudo desse impacto permitirá determinar seu desempenho do mecanismo em uma situação real de trabalho dos caches.

São ainda muitas as possibilidades a serem avaliadas com a proposta de usar um computador paralelo com arquitetura Beowulf como servidor Web. Como trabalhos futuros, pode-se avaliar melhor o impacto no tempo de resposta do servidor da variação de parâmetros tais como o percentual de documentos dinâmicos, o tempo de processamento dos documentos dinâmicos, o desempenho dos caches e o tamanho médio dos arquivos. Além disso, podem ser realizados experimentos que exijam que respostas a requisições sejam elaboradas utilizando dados contidos em bancos de dados e que devem ser acessados pelos nós de processamento. O protótipo implementado neste projeto pode ainda ser disponibilizado para pesquisadores que tenham interesse em utilizar ou expandir o sistema.

Referências Bibliográficas

[Abrams95] M. Abrams, C. R. Standridge, G. Abdulla, S. Willians, E. A. Fox: "Caching proxies: Limitations and potentials". *4th International World-wide Web Conference*, 119-133, 12/1995.

[Aho86] A. V. Aho, R. Sethi, J. D. Ullman: "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986.

[Almeida96] V. Almeida, A. Bestavros, M. Crovella, A. Oliveira: "Characterizing Reference Locality in the WWW". *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, USA, 92-103, 12/1996

[Almeida98] J. Almeida, M. Dabu, A. Manikutty, P. Cão: "Providing Differentiated Levels of Service in Web Hosting Services". *Proceedings of the Workshop on Internet Server Performance*, 06/1998.

[Andresen96] D. Andresen, T. Yang, V. Holmedahl, O. H. Ibarra: "SWEB: Towards a Scalable World Wide Web Server on Multicomputers". *Proceedings of IPPS*, 1996, Page(s): 850-856

[Apache02] The APACHE HTTP Server project.
URL: http://httpd.apache.org/, 2002.

[Arlitt96] M. F. Arlitt, C. L. Williamson: "Web Server Workload Characterization: The Search for Invariants", *ACM Proceedings of the ACM SIGMETRICS conference on Measurement & Modeling of Computer Systems*, May 23-26, 1996

[ASU02] ASU: The Association of SIMULA Users.
Home Page: http://www.isima.fr/asu/ , 02/2002.

[Baker99] S. M. Baker, B. Moon: "Scalable Web Server Design for Distributed Data Management". *Proceedings of the 15th International Conference on Data Engineering*, 1999.

[Barford98] P. Barford, M. Crovella: "Generating representative Web workloads for network and server performance evaluation". *Proceedings of the ACM SIGMETRICS Conference*, Madison, WI, 07/1998.

[Beowulf02] The Beowulf Project Web page.
URL: http://www.beowulf.org, 2002.

[Bestavros95] A. Bestavros: "Using Speculation to Reduce Server Load and Service Time on the WWW". *Proceedings of the 1995 International Conference on Information and Knowledge Management*, Baltimore, Maryland, USA, 403-410, 12/1995

[Bratley87] P. Bratley, B. L. Fox, L. E. Schrage: "A guide to simulation", 2nd Ed., Springer Verlag Inc, 1987.

[Breslau99] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker: "Web Caching and Zipf-like Distributions: Evidence and Implications". *Proceedings of IEEE INFOCOM'99*, New York, 03/1999.

[Bung99] R. B. Bung, D. L. Eager, G. M. Oster, C. L. Williamson: "Achieving Load Balance and Effective Caching in Clustered Web Servers". *The 4th International Web Caching Workshop*, 1999.

[Burns96] A. Burns, A, Wellings: "Real Time Systems and Programming Languages", Chapter 5, 2nd ed., Addison Wesley, 1996.

[Busari01] M. Busari, C. L. Williamson: "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics". *Proceedings of the IEEE INFOCOM 2001*, Anchorage, Alaska, April. 2001, pp. 1225-1234.

[Cao97] P. Cao, S. Irani: "Cost-Aware WWW Proxy Caching Algorithms". *Proceedings of the USENIX Symposium on Internet Technoloy and Systems*, 193-206, 12/1997.

[Cao99] P. Cao, J. Zhang, K. Beach. Active Cache: "Caching dynamic contents on the Web". *Distributed Systems Engineering*, 6(1):43--50, 1999.

[Cardellini99] V. Cardellini, M. Colajanni, P. S. Yu: "Dynamic Load Balancing on Web-Server Systems". *IEEE Internet Computing*, 1999

[Cardellini99a] V. Cardellini, M. Colajanni, P. S. Yu: "Redirection Algorithms for Load Sharing in Distributed Web-server Systems". *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999, Page(s): 528 –535.

[Carriero92] N. Carriero, D.Gelenter: "How to write parallel programs: a first course", 3rd Ed., The MIT Press, 1992

[CCUEC02] Centro de Computação da Universidade Estadual de Campinas – CCUEC/UNICAMP. Campinas, São Paulo, Brasil.
URL: http://www.ccuec.unicamp.br - 2002

[Chan99] Y. Chan, J. Womer, S. Jamin, J. K. MacKie-Mason: "The Case for Market-based Push Caching". *Proceedings of the Second International*

Conference on Telecommunications and Electronic Commerce, Nashville, TN, 11/1999.

[Chankhunthod96] A. Chankhunthod, P. D. Danzig, C. Neerdaels, M. Schwartz, K. J. Worrell: "A Hierarchical Internet Object Cache". *USENIX Annual Technical Conference*, 153-164, 1996

[Chinen97] K. Chinen, S. Yamaguchi: "An interactive prefetching proxy server for improvement of WWW latency". *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, 06/1997.

[Cohen99] E. Cohen, B. Krishnamurthy, J. Rexford: "Efficient Algorithms for Predicting Requests to Web Servers". *IEEE INFOCOM '99*. Volume: 1, 1999, Page(s): 284 --293.

[Comer00] D. E. Comer: "Internetworking with TCP/IP – Principles, protocols and architectures", Chapter 28, Volume 1, 4th edition. Prentice Hall, 2000.

[Cormack96] A. Cormack: "Web caching".
Url: http://www.jisc.ac.uk/acn/caching.html, 1996.

[Crocker82] D. H. Crocker: "RFC 822: standard for ARPA Internet Text Messages", 08/1982. URL: http://www.ietf.org/rfc/rfc0822.txt, 01/2003.

[Crovella95] M. E. Crovella, A. Bestavros: "Explaining World Wide Web Traffic Self-Similarity". *Technical Report TR-95-015*, Boston University, CS Dept, Boston, MA, 10/1995.

[Cunha95] C. R. Cunha, A. Bestavros, M. E. Crovella: "Characteristics of WWW Client-based Traces". *Technical Report TR-95-010*, Boston University Computer Science Department, 06/1995.

[Dahlin94] M. Dahlin, R. Wang, T. E. Anderson, D. A. Patterson: "Cooperative Caching: Using Remote Client Memory to Improve File System Performance". *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Monterey, California, 67-280, 11/1994

[Date86] C. J. Date: "An Introduction to Database Systems", Addison Wesley, 1986

[DEC00] Digital Equipament Coorporation Cache Traces
ftp://ftp.digital/com/pub/DEC/traces/proxy - 2000

[Dingle96] A. Dingle: "Web cache coherence". *Computer Networks and ISDN systems*, vol. 28, No 7-11, 907-920, 1996.

[EWS02] "Extensible Web Server Project," The Systems Research Group, Department of Computer Science and Information Systems, The University of Hong Kong.
URL: http://www.srg.csis.hku.hk/EWS/ - 2002

[FAPESP02] Fundação de amparo à Pesquisa do Estado de São Paulo. URL: http://www.fapesp.br

[Glassman94] S. Glassman: "A Caching Relay for the World Wide Web". *Proceedings of the First International Conference on the WWW*, 1994. Also published in "*Computer Networks and ISDN Systems*", volume 27 (1994), Number 2, 11/1994.

[Grimm98] C. Grimm, J. S. Vöckler, H. Pralle: "Load and Traffic Balancing in Large Scale Cache Meshes". *Computer Networks 30*, 16-18, 1998

[Grimm98a] C. Grimm, J.-S. Vöckler, H. Pralle: "Request Routing in Cache Meshes". *Computer Networks 30*, 2269-2278, 1998

[Gropp02] W. Gropp, E. Lusk, A. Skjellum: "Using MPI". 2nd Edition. MIT Press. Available from Url: ftp://ftp.mcs.anl.gov/pub/mpi/using/UsingMPI.tar.gz - 2002

[Gwertzman96] J. Gwertzman, M. Seltzer: "World Wide Web Cache Consistency". *USENIX Annual Technical Conference*, 141-152, 1996

[Heddaya97] A. Heddaya, S. Mirdad: WebWave: "Globally Load Balanced Fully Distributed Caching of Hot Published Documents". *Proceedings of the 17th International Conference on Distributed Computing Systems*, Baltimore, MD, USA, 27-30, 05/1997

[Inoue98] H. Inoue, T. Sakamoto, S. Tamaguchi: "Webhint: An automatic configuration mechanism for optimizing World Wide Web Cache system utilization". *Proceedings of the Eighth Annual Conference of the Internet Society (INET'98)*, Geneva, Switzerland, 07/1998.

[Jeruchin94] M. C. Jeruchin, P. Balaban, K. S. Shanmugan: "Simulation of communication systems", Plenum Press, 1994.

[Jin00] S. Jin, A. Bestavros: "Greedy-Dual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams". *Web Caching Workshop* (available from Boston University Department of Computer Science, technical report 2000-011), 2000.

[Karian99] Z. A. Karian, E. J. Dudewicz: "Modern Statistical, Systems, and GPSS Simulation", 2nd Ed., CRC Press, 1999 (Fisrt edition printed in 1991).

[Kelly99] T. Kelly, Y. Chan, S. Jamin, J. Mackie-Mason: "Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value". *Proceedings of the 4th International Web Caching Workshop*, San Diego, California , 04/1999.

[Kelly99a] T. Kelly, S. Jamin, J.K. MacKie-Mason: "Variable QoS from shared Web caches: user-centered design and value-sensitive replacement", *Proceedings of the MIT Workshop on Internet Service Quality Economics (ISQE 99)*, Cambridge, MA, December 1999

[Kim98] H. Kim, K. Chon: "Update policies for network caches". *Technical Memo*, Department of Computer Science, Korea Advanced Institute of Science and Technology, 07/1998.

[Kumar94] V. Kumar, A. Grama, A. Gupta, G. Karypis: "Introduction to Parallel Computing – Design and Analisys of Algorithms", Chapter 4, The Benjamin Cummings Publishing Company, Inc. 1994.

[Kurcewicz98] M. Kurcewicz, W. Sylwestrzak, A. Wierzbicki: "A distributed WWW cache". *Proceedings of the Third International WWW Caching Workshop*, Manchester, England, 06/ 1998.

[Lorenzetti96] P. Lorenzetti, L. Rizzo, L. Vicisano: "Replacement policies for a proxy cache". *Technical report*, Universita di Pisa, Italy, 10/1996.

[Malpani95] R. Malpani, J. Lorch, D. Berger: "Making World Wide Web caching servers cooperate". *Proceedings of the 4th International World-wide Web Conference*,107-117, 12/1995.

[Melve97] I. Melve, L. Slettjord, H. Bekker, T. Verschuren: "Building a Web

caching system - architectural considerations". Desire Project, 03/1997. URL: http://www.uninett.no/prosjekt/desire/arneberg

[Morris00] R. Morris and D. Lin: "Variance of Aggregated Web Traffic". *IEEE INFOCOM 2000*, Tel Aviv, 360-366, 03/2000

[Mourad97] A. Mourad and H. Liu: "Scalable Web Server Architectures". *Second IEEE Symposium on Computers and Communications*, 1997, Page(s): 12 –16.

[MPI02] The Message Passing Interface (MPI) standard. Url: http://www-unix.mcs.anl.gov/mpi/ - 2002

[Murta98] C. D. Murta, V. A. F. Almeida, Jr. W. Meira: "Analyzing performance of partitioned caches for the WWW". *In Proceedings of the Third International WWW Caching Workshop*, Manchester, England, 06/1998.

[Murta99] C. D. Murta: "Tese de Doutorado: Modelo de Particionamento de Espaço para Caches da World Wide Web". Departamento de Ciência da Computação, UFMG, Minas Gerais, Brasil, 1999.

[Narendran97] B. Narendran, S. Rangarajan, S. Yajnik: "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access". *Proceedings of the 16th Symposium on IEEE Reliable Distributed Systems*, 1997, Page(s): 97 –106.

[NASA00] HTTP requests to the NASA Kennedy Space Center WWW server in Florida, USA. URL: http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html - 2000

[NLANR00] National Laboratory For Applied Network Research - Examples of

Squid Log Files.
URL: http://www.ircache.net/ - 01/2003.

[NS02] NS – The Network Simulator Home Page.
Url: http://www.isi.edu/nsnam/ns/ - 02/2002.

[Povey97] D. Povey, J. Harrison: "A Distributed Internet Cache". *Proceedings of the 1997 Australasian Computer Science Conference*. Sydney, Australia, 1997.

[Rabinovich98] Michael Rabinovich, Jeffrey S. Chase, Syam Gadde: "Not all Hits are Created Equal: Cooperative Proxy Caching Over a Wide-Area Network". *Computer Networks 30*, 22-23, 1998

[Reddy98] M. Reddy, G. P. Fletcher: "Intelligent Web caching using document life histories: A comparison with existing cache management techniques". *Proceedings of the Third International WWW Caching Workshop*, Manchester, England, 06/1998.

[RNP01] Rede Nacional de Pesquisa (RNP)
URL: http://www.rnp.br/ - 02/2001

[Rodriguez99] P. Rodriguez, C. Spanner, E. W. Biersack: "Web Caching Architectures: Hierarchical and Distributed Caching". *Proceedings of the 4th International Web Caching Workshop*, 1999.

[Shimizu75] T. Shimizu: "Simulação em computador digital", Chapter 1, Edgard Blücher, Brazil, 1975.

[SimScript02] Simscript Simulation Language Home Page.
Url: http://www.caciasl.com/simscript.cfm - 02/2002.

[Stallings93] W. Stallings, "Local and Metropolitan Area Networks", Chapter 9, 4th Ed., Prentice-Hall, 1993.

[Stevens94] W. R. Stevens: "TCP/IP Illustrated", Volume 3: "*TCP for transactions, HTTP, NNTP, and Unix® Domain Protocols*", Chapters 13 and 14, Addison-Wesley, 1994.

[Tewksbury98] R. B. Tewksbury: "Is the Internet heading for a cache crunch?" *In Proceedings of the Eighth Annual Conference of the Internet Society (INET'98)*, Geneva, Switzerland, 07/1998.

[Vakali99] A. Vakali: "A Web-based evolutionary model for Internet Data". *2nd Int. Workshop on Network-based Information Systems*, Florence, 08/1999.

[Venkata96] N. P. Venkata, J. C. Mogul: "Using predictive prefetching to improve World Wide Web latency". *Computer Communication Review*, 26(3),22-36, 07/1996.

[W3C03] "HTTP - Hypertext Transfer Protocol", URL: http://www.w3.org/Protocols/, 01/2003.

[Wessels97] D. Wessels, K. Claffy: "Internet Cache Protocol (ICP), version 2". Networking Working Group, RFC 2186, 09/1997

[Wessels97a] D. Wessels, K. Claffy: "Application of Internet Cache Protocol (ICP), version 2". Networking Working Group, RFC 2187, 09/1997

[Willians96] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox: "Removal policies in network caches for World Wide Web

Documents". *ACM SIGCOMM 96*, 293-305, 08/1996.

[Yang02] X.Yang, G. de Veciana: "On Zipf Law and Effectiveness of Hierarchical Caching", *Proceedings of Communication Networks and Distributed Systems 2002 (CNDS02).*

[Zotto96] O. Zotto: "Protocolo HTTP (Hypertext Transfer Protocol)", *Revista Bate Byte,* Número 57, Celepar - Companhia de Informática do Paraná, 09/1996.

Apêndice A: Ferramentas

Essa seção apresenta uma breve descrição das principais ferramentas desenvolvidas no decorrer deste projeto e utilizadas para converter, analisar e gerar dados a serem usados no processo de pesquisa.

A.1. Gerahash

Um *trace* proveniente de um cache real pode ter diferentes formatos. O mais comum é o formato utilizado pelo Squid [NLANR00]. Nesse tipo de *trace*, cada linha de um arquivo texto contém os 10 campos a seguir, separados por espaços e finalizados pelo caractere CR ou Control+M.

Timestamp: Hora em que o *socket* usado na transmissão dos dados é finalizado. O formato usado é o "Unix time" (segundos desde 1° de janeiro de 1970), com precisão de milisegundos.

Elapsed Time: Tempo para processamento da requisição. Esse é o tempo decorrido desde a inicialização (accept) e a finalização (close) do *socket* usado na transmissão do arquivo requisitado. Para requisições HTTP persistentes, esse é o tempo desde a leitura do primeiro byte da requisição e o envio do último byte da resposta.

Client Address: Devido a políticas de privacidade, o endereço IP do cliente, que deveria estar nesse campo é substituído por um endereço IP aleatório. O mapeamento entre o endereço do cliente e o endereço aleatório é o mesmo em todo o *trace*, mas não é mantido entre *traces* diferentes.

Log Tag and HTTP Code: Descreve como a requisição foi tratada localmente. Por exemplo, descreve se ocorreu um *hit* ou um *miss*. Adiciona também o código de status HTTP, que é usado para indicar o resultado da operação de busca por dados.

Size: O número de bytes enviados ao cliente.

Request Method: O método de requisição HTTP usado (normalmente "GET").

URL: A URL da requisição. Argumentos de consultas, CGI por exemplo, não são armazenados. São considerados argumentos todos os caracteres que seguem um sinal de "?".

User Ident: Por motivos de privacidade, sempre trocados por "-" para a maioria dos *traces*.

Hierarchy Data and Hostname: Descrição de como e onde o objeto requisitado foi encontrado, quando ele for encontrado.

Content Type: Tipo do arquivo enviado ao cliente. Pode ser substituído por "-".

Um exemplo de algumas requisições armazenadas em um *trace* é apresentado a seguir. Considere que cada parágrafo é armazenado no *trace* como uma única linha.

```
1020816229.231 516 61.87.2.67 TCP_MISS/304 333 GET
http://www.creationent.com/pics/home_icons/new_sidebar.jpg - DIRECT/216.122.237.6 -

1020816267.836 193 61.87.2.67 TCP_MISS/302 644 GET http://home-
13.tiscali.nl/%7Eti017329/honeyz01.jpg - DIRECT/195.241.76.80 text/html

1020816304.598 55 226.90.141.125 TCP_REFRESH_HIT/304 203 GET
http://ar.atwola.com/content/B0/0/H7pTL2Luf0_kw3xmlj8W1sns8a9RRNke8_SAqLzKBa609jmULHVa8jg
FKtiL69KXCWvLTQ4eKHG6BVFfpwz9J2_nwVlARAAN-pkCJqF1Tww$/aol - DIRECT/152.163.226.185 -

1020816532.277 75 61.87.2.67 TCP_REFRESH_HIT/304 254 GET
http://dl.www.juno.com/images/online_registration/arrow.gif - DIRECT/64.136.25.24 -

1020816537.022 239539 134.202.51.180 TCP_MISS/504 1130 GET
http://www.hot.ee/avznpwzx/link.html - NONE/- -

1020816668.215 6 61.87.2.67 TCP_IMS_HIT/304 267 GET http://www.honda.com/images/1.gif -
NONE/- image/gif
```

O **Gerahash** tem como objetivo gerar um *trace* simplificado a partir de um *trace* real onde o endereço URL foi substituído por um número inteiro que o equivalha. Esse número é calculado a partir da aplicação da URL à função de *hash* cujo código C é apresentado a seguir:

```
int calcula_hash(char *nomeArq)
{
    unsigned int cont;
    for (cont = 0; cont <4; cont++)
       hash[cont]='\0';
    for (cont = 0 ; cont < strlen(nomeArq); cont++)
       hash[cont % 4] = hash[cont % 4] ^ nomearq[cont];
    return (hash[0] + hash[1]*256 + hash[2]*256*256 + hash[3]*256*256*256);
}
```

Listagem A.1.1 – Função para cálculo do *hash* de uma URL

O resultado da aplicação do **gerahash** no *trace* apresentado como exemplo é apresentado a seguir:

```
1.000 333 1446850322
39.605 644 69037915
76.367 203 1175216445
304.046 254 1411981846
308.791 1130 627276590
439.984 267 1141310331
```

O *trace* simplificado possui apenas 3 campos, que são explicados a seguir:

Timestamp: Hora da requisição. Como não é necessário usar o "Unix time", a primeira requisição recebe valor 1.00 e as outras dependem apenas da distância temporal entre elas e a primeira requisição.

Tamanho: Tamanho do arquivo, em bytes.

Hash: Número inteiro que representa o arquivo.

O *trace* simplificado foi usado principalmente na alimentação de simuladores de caches para Web. A principal vantagem é poder usar um número inteiro no lugar de um conjunto potencialmente muito grande de caracteres para representar a URL da requisição, o que economiza memória no simulador e aumenta sua simplicidade quando da leitura dos *traces*.

Linha de comando:

```
gerahash ARQ-LOG ARQ-HASH
```

Parâmetros:

ARQ-LOG: *Trace* a ser processado.

ARQ-HASH: *Trace* simplificado.

A.2. LogStat

Essa ferramenta tem como objetivo analisar um *trace* e sumarizar informações sobre o mesmo. As informações apresentadas são:

- Sites mais acessados, com as respectivas quantidades de acesso.
- Número total de acessos e bytes requisitados
- Desempenho do cache real que recebeu as requisições e gravou o *trace.*

Um exemplo de resultado gerado após processar um *trace* é apresentado a seguir.

```
1: www.uol.com.br - Acessos: 182867
2: channels.real.com - Acessos: 65584
3: www.unicamp.br - Acessos: 52381
4: www.zip.net - Acessos: 22377
5: www2.uol.com.br - Acessos: 21345
6: chatter.uol.com.br - Acessos: 20402
7: www.geocities.com - Acessos: 14607
8: 200.211.190.61 - Acessos: 13798
9: ad.doubleclick.net - Acessos: 13297
10: www.cosmo.com.br - Acessos: 13114
11: www.zaz.com.br - Acessos: 12313
12: adserver.zaz.com.br - Acessos: 9216

Arquivo: ..\logs\813jan.log
Numero de acessos   : 1352972
Numero de hosts     : 15988
Kbytes requisitados : 3098084
Numero de Hits   : 547323 (40.453%)
Numero de Misses : 805649 (59.547%)
Byte Hit (Kbytes): 162767 (5.254%)
Memoria utilizada para processor trace: 187 Kbytes
```

Linha de comando:

```
logstat [OPCOES] logfile
```

Opções:

```
-rank N: Exibe os N sites mais acessados; N = 0 mostra todos.
-norel : Não imprime o relatório final.
-nord  : Não executa ordenação de popularidade (mais rápido).
```

A.3. ArqSizes

A ferramenta **Arqsizes** é usada para analisar um *trace* e apresentar estatísticas relativas aos tamanhos dos documentos requisitados. Ele apresenta um relatório com as distribuições de números de acesso dos arquivos em faixas de tamanhos. Um exemplo de resultado é apresentado a seguir:

```
     Até       NArq    %NArq      Bytes(MB)   %Bytes
    1000      25794    85.56          5         3.85
    2000       1868     6.20          3         2.00
    3000        545     1.81          1         1.04
    4000        420     1.39          1         1.15
    5000        293     0.97          1         1.02
    6000        201     0.67          1         0.86
    7000        116     0.38          1         0.59
    8000         78     0.26          1         0.46
    9000         64     0.21          1         0.42
   10000         64     0.21          1         0.47
      .
      .
      .
Numero de arquivos: 30149
Total de bytes:     128074215
Tamanho médio :     4248.04
```

As informações apresentadas nas colunas são, respectivamente:

Até: Tamanho máximo dos arquivos que estão nessa faixa.

NArq: Número de arquivos cujo tamanho é menor que o tamanho na coluna "Até" e maior ou igual ao anterior.

%NArq: Percentual do número de arquivos cujo tamanho é menor que o tamanho na coluna "Até" e maior ou igual ao anterior.

Bytes (MB): Número de Mbytes cujo tamanho é menor que o tamanho na coluna "Até" e maior ou igual ao anterior.

%Bytes: Percentual do número de Mbytes cujo tamanho é menor que o tamanho na coluna "Até" e maior ou igual ao anterior.

Linha de comando:

```
arqsizes NOME_LOG
```

Parâmetro:

NOME_LOG: *Trace* a ser analisado.

A.4. StatComp

A ferramenta **StatComp** pode ser utilizada para analisar a popularidade de arquivos em diversos *traces* diferentes. Um exemplo de relatório final é apresentado a seguir.

A1.log	A2.log	A3.log	A4.log
17738	8257	2144	561
14312	6749	1845	543
9877	4060	1150	340
8606	4667	1248	348
7731	5924	1676	487
6217	3155	851	255
5952	3590	927	254
5693	3773	1114	319
4632	3181	951	287
4467	3423	1004	308

A primeira linha apresenta os números de acessos aos arquivos no *trace* A1.log, ordenados de forma decrescente. Cada linha apresenta o número de acessos de cada arquivo em cada um dos outros *traces*.

Linha de comando:

```
statcomp OPÇÃO[-s ou -h] NumArqIn ArqIn1,...,ArqInN ArqOut
```

Parâmetros:

OPÇÃO:

-s: *Trace* simplificado.

-h: *Trace* com informações de Passagem de Recomendação.

NumArqIn: Número de *traces* a serem analisados.

ArqIn1,...,ArqInN: Nomes dos *traces* a serem analisados.

ArqOut: Nome do arquivo onde devem ser gravados os resultados.

A.5. CompCresPop

Essa ferramenta tem como função analisar um *trace* e determinar o percentual de presença de um conjunto específico de arquivos dentro de divisões do cache. Em cada divisão é verificado qual o conjunto de N arquivos mais populares. Verifica-se então o percentual de presença desses arquivos nas demais divisões.

Um exemplo de resultado gerado pelo **CompCresPop** após analisar um *trace* é apresentado a seguir.

```
Log: teste.log
Quantidade de Intervalos : 10
Quantidade de Requisicoes: 72731
Tamanho dos Intervalos   : 7273
Tamanho da amostra de populares: 50

1.00 0.84 0.64 0.62 0.64 0.64 0.68 0.66 0.66 0.52
0.84 1.00 0.68 0.66 0.66 0.64 0.70 0.68 0.66 0.52
0.64 0.68 1.00 0.80 0.68 0.66 0.74 0.66 0.62 0.52
0.62 0.66 0.80 1.00 0.72 0.68 0.70 0.64 0.62 0.52
0.64 0.66 0.68 0.72 1.00 0.82 0.74 0.70 0.68 0.56
0.64 0.64 0.66 0.68 0.82 1.00 0.80 0.72 0.68 0.54
0.68 0.70 0.74 0.70 0.74 0.80 1.00 0.82 0.74 0.60
0.66 0.68 0.66 0.64 0.70 0.72 0.82 1.00 0.80 0.58
0.66 0.66 0.62 0.62 0.68 0.68 0.74 0.80 1.00 0.74
0.52 0.52 0.52 0.52 0.56 0.54 0.60 0.58 0.74 1.00
```

A principal utilidade dessa ferramenta é verificar o comportamento da variação da popularidade dos arquivos através do *trace*. Isso permite constatar com que taxa a popularidade dos arquivos cresce e decai, permitindo um que seja possível prever com

que velocidade os arquivos no cache passam de muito populares a pouco populares e, portanto, descartáveis.

Linha de comando:

```
CompCresPop ARQLOG ARQRES QTDE_INTERV TAM_AMOSTRA %DESL_JAN
```

Parâmetros:

```
ARQLOG: Nome do trace a ser analisado.
ARQRES: Nome do arquivo onde devem ser gravados os resultados.
QTDE_INTERV: Número de divisões do trace.
TAM_AMOSTRA: Tamanho da amostra de populares analisada.
%DESL_JAN: Percentual de deslocamento da janela das divisões.
```

Muitas vezes, é necessário que as "janelas" formadas pelas divisões do *trace* se sobreponham, de forma a permitir que se possa acompanhar melhor a variação da popularidade dos arquivos, possibilitando que picos de popularidade ocorridos em uma ou mais divisões possam interferir na medida de popularidade de divisões adjacentes. O parâmetro usado para definir o percentual de sobreposição das divisões é o `%DESL_JAN`.

A.6. GeraDist

O **GeraDist** é usado para gerar *traces* com uma variação de distribuição de popularidade fixa entre os arquivos.

Linha de comando:

```
geradist INC_POP NUM_REQ ARQRES
```

Parâmetros:

```
INC_POP: Variação de popularidade entre arquivos.
NUM_REQ: Número de requisições a serem geradas.
ARQRES: Nome do arquivo onde devem ser gravados os resultados.
```

O parâmetro `INC_POP` é utilizado para definir qual o incremento da probabilidade do n+1-ésimo elemento em ser acessado com relação ao n-ésimo elemento.

A.7. StatDist

A ferramenta **StatDist** tem como finalidade analisar um *trace* simplificado e gerar uma listagem dos números de acessos aos arquivos, ordenados em ordem decrescente. Com essa listagem, é possível plotar gráficos que demonstrem a variação de popularidade entre os arquivos.

Linha de comando:

```
statdist ARQIN ARQOUT
```

Parâmetros:

```
ARQIN: Trace a ser analisado.
ARQOUT: Nome do arquivo onde devem ser gravados os resultados.
```

A.8. QoSTest

A ferramenta QoSTest foi criada para avaliar o impacto da implementação de diferentes qualidades de serviço oferecidas a clientes de serviços de caches para Web, descritos no capítulo 4. Essa ferramenta analisa um *trace* e avalia o desempenho em hit ratio obtidos por conjuntos de arquivos favorecidos, dependendo do grau de favorecimento a eles aplicado.

A seguir, é apresentado um exemplo gerado pelo **QoSTest** após a análise de um *trace*.

GrauFav	%HitFav	%HitOutros	%HitTot	%Miss	%MissOutros	&MissTot
0	7.3457	14.1283	14.1224	92.6543	85.8717	85.8776
1	2.1085	14.0519	14.0405	97.8915	85.9481	85.9594
2	9.6449	14.0992	14.0956	90.3551	85.9008	85.9044
3	31.0436	13.4650	13.4803	68.9564	86.5350	86.5197
4	30.2384	13.0594	13.0729	69.7616	86.9406	86.9271
5	54.5610	13.7058	13.7531	45.4390	86.2942	86.2469
6	55.1168	13.4402	13.4913	44.8832	86.5598	86.5087
7	42.2429	13.3348	13.3613	57.7571	86.6652	86.6387
8	56.0624	14.4170	14.4636	43.9376	85.5830	85.5364
9	35.0170	14.2778	14.2939	64.9830	85.7222	85.7061
10	52.6763	14.1430	14.1833	47.3237	85.8570	85.8167

A seguir, é brevemente descrito o significado das colunas presentes no resultado da análise do *trace*.

GrauFav: Grau de favorecimento dado aos arquivos cujos clientes devem sofrer variação na QoS oferecida. O conceito de grau de favorecimento é explicado no capítulo 4.

%HitFav: Percentual de *hit ratio* obtida pelos arquivos favorecidos pelo cache.

%HitOutros: Percentual de *hit ratio* obtida pelos arquivos não favorecidos pelo cache.

%HitTot: Percentual de *hit ratio* total obtido pelo cache.

%Miss: Percentual de *misses* obtido pelos arquivos favorecidos pelo cache.

%MissOutros: Percentual de *misses* obtido pelos arquivos não favorecidos pelo cache.

%MissTot: Percentual de *misses* total obtido pelo cache.

A.9. ExtractCol

Essa simples ferramenta foi criada com o objetivo de permitir a geração de um arquivo formado apenas por uma coluna de um *trace*.

Linha de comando:

```
extractCol ArqIN ArqOUT Col
```

Parâmetros:

```
ArqIN: Trace a ser analisado.
ArqOUT: Nome do arquivo a ser criado.
Col: Número da coluna escolhida.
```

A.10. PoeCol

Essa também simples ferramenta foi criada para introduzir colunas em um *trace*. O valor introduzido na coluna é o mesmo em todos os elementos da coluna.

Linha de comando:

```
poecol ARQIN ARQOUT NCOLS POSICAO VALOR
```

Parâmetros:

ARQIN: *Trace* a ser analisado.

ARQOUT: Novo *trace* criado, com as colunas inseridas.

NCOLS: Número de colunas a inserir.

POSICAO: Posição de inserção (posição das novas colunas).

VALOR: Valor dos elementos das colunas.

A.11. Exechost

Essa ferramenta foi criada para possibilitar executar um programa em um computador remoto. Ele possui duas partes: o **ExechostS** e o **ExechostC**.

O **ExechostS** implementa um processo que fica "ouvindo" em uma porta IP por instruções de um programa a executar. Ao receber essas instruções, executa um *fork* seguido por um comando *execlp*.

Linha de comando:

```
exechosts [PORTA]
```

Parâmetros:

PORTA: Porta IP onde o processo esperará por instruções. Se nenhuma porta for informada, é assumido o valor 32100.

A parte **ExechostC** é responsável pelo envio de requisições de execução de programas ao **ExechostS**. Para tanto, ele abre uma conexão TCP para o endereço e porta definidos e envia as informações sobre o programa a ser iniciado. A conexão TCP é encerrada em seguida.

Linha de comando:

```
exechostc HOST PORTA COMANDO ARG1,...,ARGN
```

Parâmetros:

HOST: Nome ou endereço IP do computador que irá executar o programa.

PORTA: Porta onde o ExechostS está esperando receber instruções.

COMANDO: Nome do programa a ser executado, incluindo *path*.

ARG1,...,ARGN: Argumentos a serem passados ao programa que será executado no computador remoto.

A.12. Simulador de Servidor Web em Beowulf

Essa ferramenta foi criada para simular a utilização de um computador com a arquitetura Beowulf utilizando o modelo descrito no capítulo 7. A simulação foi importante para tentar prever principalmente resultados de desempenho antes do protótipo descrito no capítulo 7 ser implementado.

Esse simulador, cuja interface é apresentada na figura A.1, permite simular Bewulf's com qualquer número de nós de processamento e caches. Permite configurar opções tais como os tempos de procura por arquivos e transmissão de requisições bem como a capacidade da rede e qualidade de serviço esperada do Beowulf, através do tempo de *timeout* das requisições.

O simulador apresenta durante a simulação a possibilidade da variação da taxa de requisições, de forma a permitir verificar o impacto dessa variação nos tempos de resposta às requisições. Os tempos de resposta das requisições podem ser verificados na parte inferior do simulador. Além disso, dados relativos ao fluxo de requisições e tempo de resposta das mesmas podem ser gravados em um arquivo texto para posterior elaboração de gráficos e análise estatística dos dados, com uma ferramenta destinada a esse fim.

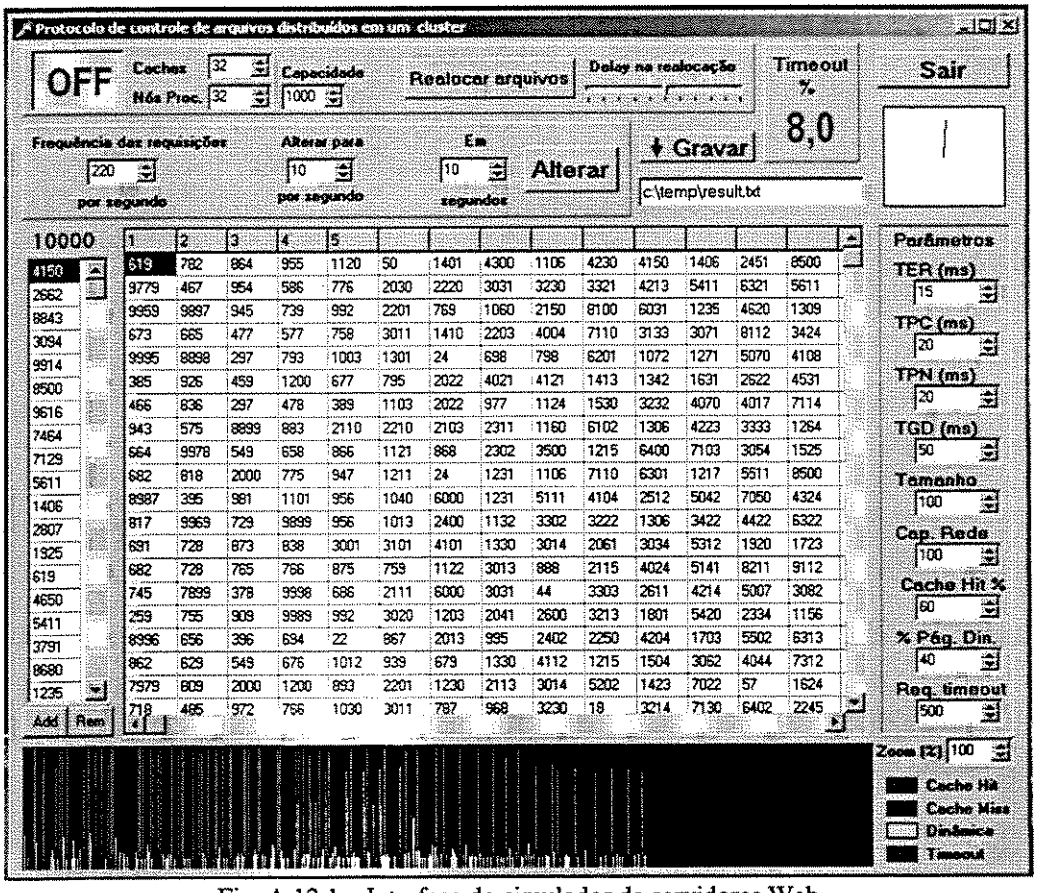


Fig. A.12.1 – Interface do simulador de servidores Web