UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
COORD. DE PÓS-GRADUAÇÃO EM INFORMÁTICA- COPIN

PROJETO E IMPLEMENTAÇÃO DE UM SERVIDOR WORLD WIDE WEB PROXY COM SUPORTE A RECURSOS ESPELHADOS

TÉRCIO EULÁLIO DE ALBUQUERQUE FONSÊCA

CAMPINA GRANDE, JULHO DE 1998

Tércio Eulálio de Albuquerque Fonsêca

PROJETO E IMPLEMENTAÇÃO DE UM SERVIDOR WORLD WIDE WEB PROXY COM SUPORTE A RECURSOS ESPELHADOS

Dissertação apresentada ao Curso de MESTRADO EM INFORMÁTICA da Universidade Federal da Paraíba, em cumprimento às exigências parciais para a obtenção do grau de Mestre.

Área de Concentração: **Ciência da Computação**

Sub-Área: **Redes de Computadores e Sistemas Distribuídos**

Francisco Vilar Brasileiro
Orientador

Campina Grande, Julho de 1998

F676p Fonsêca, Tércio Eulálio de Albuquerque.
Projeto e implementação de um servidor world wide web proxy com suporte a recursos espelhados / Tércio Eulálio de Albuquerque Fonsêca. - Campina Grande, 1998.
72 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
"Orientação : Prof. Dr. Francisco Vilar Brasileiro".
Referências.

1. World Wide Web - Servidor. 2. Servidor WWW - Projeto e Implementação. 3. Recursos Web. 4. Dissertação - Informática. I. Brasileiro, Francisco Vilar. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 004.738.5(043)

PROJETO E IMPLEMENTAÇÃO DE UM SERVIDOR WEB PROXY
COM SUPORTE A RECURSOS ESPELHADOS

TÉRCIO EULÁLIO DE ALBUQUERQUE FONSÊCA

DISSERTAÇÃO APROVADA EM 20.07.1998

Francisco Vilar Brasileiro
PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador

PROF. MARCUS COSTA SAMPAIO, Dr.
Examinador

PROF. ROGÉRIO DRUMMOND B.P.DE MELLO FILHO, Ph.D
Examinador

CAMPINA GRANDE - PB

AGRADECIMENTOS

Aos meus familiares, que direta ou indiretamente contribuíram para que eu chegasse até aqui, me dando apoio, ouvindo os meus problemas e tudo mais que uma família pode oferecer. Em especial a meus avós materno, de quem recebo todo incentivo.

Aos funcionários do DSC, principalmente aos funcionários da COPIN, que deram todo o suporte e contribuíram bastante para que este trabalho pudesse ter sido realizado, sempre estando prontos para ajudar.

Aos meus companheiros do LSD, Luíza, Érica, Claudio, Felipe e Tati, que me aturaram bastante e com os quais convivi durante todo este tempo.

À colega Raquel, com quem troquei idéias e que dividiu comigo não só o ambiente de trabalho, mas também os bons e os maus momentos, durante os dois anos de trabalho.

Aos amigos particulares do DSC, Dalton, Sandro e Álvaro, que sempre estavam por perto para me dar um conselho. Agradeço, em especial a Vânia, que compartilha comigo todos os meus desabafos e me dá muitos conselhos.

Aos que fazem a COPEX, o Gênesis e a Cantina do DSC, o meu reconhecimento.

E por último, ao meu orientador Francisco Vilar Brasileiro (Fubica), que sem ele não teria sido possível a realização deste trabalho.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

RESUMO

Atualmente, a indisponibilidade dos recursos é um dos principais problemas enfrentados pelos usuários da *World Wide Web*. Este problema é ocasionado principalmente por falhas na infra-estrutura da Web. Ele pode ser solucionado através da introdução de redundância no sistema, o que significa replicar os recursos Web. A replicação de recursos possibilita uma diminuição na probabilidade de que o recurso não esteja disponível. Infelizmente, o esquema de nomeação e endereçamento de recursos provido pela Web e amplamente utilizado não favorece o suporte adequado para a replicação de recursos, pois este esquema está diretamente associado à localização física do recurso. Em nosso trabalho apresentamos um esquema de nomeação de recursos independente de sua localização, que pode ser utilizado no espelhamento de recursos na Web. A nossa proposta é introduzir um servidor *proxy* na infra-estrutura da Web para que ele forneça o suporte adequado ao acesso e recuperação dos recursos espelhados. A recuperação dos recursos espelhados é aprimorada com o fornecimento de um mecanismo de balanceamento de carga.

ABSTRACT

The unavailability of resources, is one of the main problems faced by the World Wide Web users nowadays. This problem is normally caused by failures in the Web's infra-structure and can be solved by the introduction of redundancy on the system, by means of replicating its resources. Resource replication allows the redution of the probability of the unavailability of resources. Unfortunately, the resource identification scheme provided by the Web does not support resource replcation because it maps the resource identification on a unique physical location. Our proposal uses a new resource identification scheme that can be used to implement the mirroring of Web resources. This is implemented by a proxy that is able to access this mirrored resources. The recovery of mirrored resources is improved by a load balancing mechanism.

1. Introdução

1.1 A Importância da Web e o seu Crescimento

A Internet apresenta-se atualmente como um meio de comunicação de grande importância. Seu desenvolvimento tem sido intenso e sua popularidade tem crescido rapidamente nestes últimos anos, principalmente devido ao surgimento e desenvolvimento da WWW - World Wide Web (ou simplesmente Web) [BCLNS94].

Atualmente, muitas pessoas ainda imaginam que a Internet e a Web são uma só coisa, mas a realidade é outra. A Internet se constitui no aspecto físico - computadores, cabos e redes - e é definida como a grande rede mundial de computadores - uma rede de redes. É isto que permite aos usuários se conectarem aos milhares de computadores distribuídos por todo o mundo. Já a Web é uma abstração, é um conjunto de serviços comuns (protocolos e ferramentas que permitem o compartilhamento de informações) no topo da Internet [WM96].

A Web utiliza a Internet para transmitir documentos hipermídia entre usuários de computadores distribuídos pelo mundo inteiro, e foi desenvolvida com o conceito de "leitor universal", ou seja, qualquer participante do sistema deve ser capaz de ler a informação armazenada em qualquer computador conectado ao sistema.

Dados importantes comprovam o crescimento tanto da Web quanto da Internet. Apesar de ter surgido por volta dos anos 70, a Internet só veio tornar-se popular no ano de 1993, quando deixou de ser uma rede de natureza acadêmica e passou a ser explorada

comercialmente. No gráfico da Figura 1.1 abaixo, vemos o crescimento da Internet, ao longo de 6 anos. Este crescimento é medido através do número de computadores conectados à rede durante o primeiro e segundo semestres de cada ano.

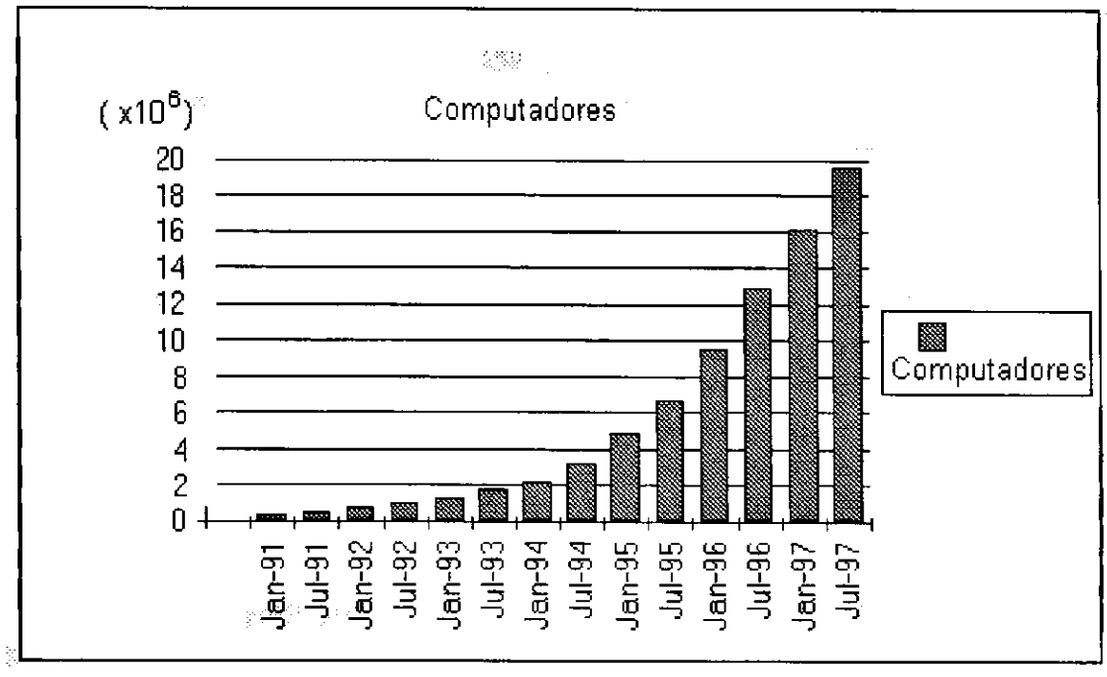

Figura 1.1 - Crescimento da Internet - Fonte: Network Wizards - http://www.nw.com

Além disso, podemos observar no gráfico da Figura 1.1 que o crescimento da Internet foi relativamente pequeno até o segundo semestre de 1993, mas a partir deste ano o número de computadores ligados a ela aumentou de forma acentuada; no segundo semestre de 1994, o número de computadores já tinha dobrado, o que aconteceu também no segundo semestre de 1995 e no segundo semestre de 1996.

Com relação à Web, que surgiu no início dos anos 90, o crescimento foi mais intenso e em um período de tempo menor. Só para se ter uma idéia, em dezembro de 1992, a Web estava em 127º lugar no *ranking* dos serviços de rede mais utilizados, enquanto que em dezembro de 1993, se encontrava em 11º [HUGES93]. Através dos dados da Tabela 1, podemos observar este crescimento, que é medido pelo número de "*Web Sites*", na Internet.

Mês	Nº de Web Sites	% de sites .com
06/93	130	1.5
12/93	623	4.6
06/94	2.738	13.5
12/94	10.022	18.3
06/95	23.500	31.3
01/96	100.000	50.0
06/96	230.000	68.0
01/97	650.000	62.6

Tabela 1 - Número de Web Sites no Mundo desde 1993
Fonte: Matthew Gray - http://www.mit.edu/people/mkgray/net

O que pode ser observado na Tabela 1 é que a taxa de crescimento da Web tem sido e continua sendo exponencial, embora esta taxa de crescimento esteja diminuindo um pouco atualmente. Uma constatação importante é que o número de *Web sites* tem duplicado, ou até aumentado a uma taxa maior do que esta, em um período de apenas 6 meses. O que também podemos observar na Tabela 1 é o crescimento do número de *sites* comerciais (*sites* .com), que contribuiu para o rápido crescimento da Web, que se constitui hoje como um grande ambiente para realização de grandes negócios. Este crescimento dos *sites* comerciais pode ser observado na tabela pela porcentagem que ele representa na Web, em 93 representavam apenas 1,5% de todos os *sites*, em 97 já representava mais 62% da Web.

No Brasil, a Internet chegou em 1988, por iniciativa da FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo -, da UFRJ - Universidade Federal do Rio de Janeiro - e do LNCC - Laboratório Nacional de Computação Científica. Seu maior crescimento foi constatado por volta do ano de 1995, quando passou a ser explorada comercialmente, e também devido ao vultoso desenvolvimento que a Web já apresentava em vários países do mundo.

No Brasil, como em todo o mundo, a World-Wide Web vem crescendo a cada ano. O Comitê Gestor Internet/Brasil (que será apresentado posteriormente) disponibilizou dados em termos de *Web Sites* a partir de junho de 1997, como podemos constatar na Tabela 2:

Serv. Web	30/06/97	30/07/97	30/08/97	30/09/97	30/10/97	30/11/97
com.br	12.452	15.174	16.066	18.426	20.066	22.715
g12.br	93	104	125	139	162	162
gov.br	290	347	375	377	501	501
mil.br	7	9	11	15	17	17
net.br	31	33	33	33	35	35
org.br	491	607	696	766	852	940
br	15.548	18.458	19.490	21.940	23.753	26.554

Tabela 2 - Distribuição de Servidores Web no Brasil.
Fonte: Comitê Gestor Internet/Brasil - http://www.cg.org.br

Podemos perceber, através dos dados da Tabela 2, que a Web comercial teve um crescimento muito maior em relação aos outros setores, tais como a Web militar (mil.br) e governamental (gov.br), por exemplo. Os domínios com.br (organizações comerciais) e org.br (organizações não governamentais) quase dobraram em um período menor do que seis meses. Os domínios .g12.br (para escolas de 1º e 2º graus) e .net.br (para empresas de telecomunicações) também apresentam crescimento, embora em proporções menores.

Existem muitas razões para este rápido crescimento da Web. Dentre estas, podemos citar: a sua fácil utilização, a disponibilidade de editores e de ferramentas que facilitam a publicação de documentos na Web, a natureza independente de máquina dos protocolos e linguagens que ela suporta, a disponibilidade de interfaces gráficas utilizadas para navegação (navegação é o termo utilizado para expressar o ato de visitar as páginas publicadas na Web), etc.

Com isso, podemos concluir que a Web oferece a infra-estrutura necessária para que seus usuários, em qualquer lugar do mundo, possam ter acesso aos serviços disponibilizados na Internet e às informações armazenadas em computadores, também distribuídos por todo o mundo.

Devido ao grande crescimento da Web, surgiu o W3C - World-Wide Web Consortium. O W3C foi fundado em 1994, com o objetivo de desenvolver padrões comuns para o desenvolvimento da Web [W3C96]. O W3C é uma associação de organizações internacionais composta pelo Massachusetts Institute of Technology Laboratory for Computer Science (MIT/LCS), nos EUA, pelo Institut National de Recherche en Informatique et en Automatique

(INRIA), na França, e pelo Keio University Shonan Fujisawa Campus, no Japão. O W3C foi instituído em colaboração com o Colective of European high-energy physics reseachers (CERN - Laboratório Europeu para Física de Partículas - sediado em Genebra), onde a Web foi criada.

O W3C, financiado por membros comerciais, é gerenciado por Jean-François Abramatic, o presidente, e Tim Berners-Lee, o Diretor do W3C e criador da Web. O W3C trabalha com a comunidade global para produzir especificações e recomendar softwares que são desenvolvidos e disponibilizados gratuitamente no mundo inteiro.

O W3C fornece alguns serviços públicos, tais como:

- Um repositório de informações e especificações sobre a World-Wide Web para usuários e desenvolvedores;
- códigos de implementação para personificação e promoção de padrões;
- vários protótipos e exemplos de aplicações para demonstrar o uso de novas tecnologias.

No Brasil, também foi criada, em 1995, uma organização similar, o Comitê Gestor Internet, criado pelo Ministério das Comunicações e pelo Ministério da Ciência e Tecnologia. Este comitê visa tornar efetiva a participação da sociedade brasileira nas decisões envolvendo a implantação, a administração e o uso da Internet. Além dos dois ministérios, o comitê é composto também de entidades operadoras e gestoras de *backbones* (espinhas dorsais), de representantes de provedores de acesso, de representantes de usuários e da comunidade acadêmica.

As principais atribuições do Comitê Gestor da Internet são:

- Fomentar o desenvolvimento de serviços Internet no Brasil;
- recomendar padrões e procedimentos técnicos e operacionais para a Internet no Brasil;
- coordenar a atribuição de endereços Internet, o registro de nomes de domínios e a interconexão de *backbones*;
- coletar, organizar e disseminar infomações sobre os serviços Internet.

1.2 Problemas Relacionados com a Web

Embora a Web seja um sucesso como provedora de informações e serviços, ela não possui uma gerência central, ocasionando com isso um crescimento desordenado. A quantidade de recursos disponíveis, proporcionada pela facilidade e a velocidade com que eles são publicados, é o principal indicador deste crescimento.

O crescimento rápido e desordenado apresentado pela Web tem como conseqüência uma série de problemas que são enfrentados pelos usuários. Um dos mais visíveis é o da quebra de ligação (*broken-link*) em um hipertexto. A quebra de ligação ocorre quando um recurso referenciado em um hipertexto não está disponível no endereço especificado.

Possivelmente, o problema de quebra de ligação irá aumentar à medida que a Web cresça, tornando-a mais difícil de utilizar e de gerenciar. A quebra de ligação afeta tanto os usuários quanto os provedores da informação, ocasionando aos usuários a insatisfação com os serviços Web, abalando a reputação dos provedores de informação e causando perda de oportunidades de negócios.

Existem algumas situações com as quais a Web convive e que são causadoras da quebra de ligação, dentre as quais podemos citar:

- **Falta de Integridade Referencial**

Um sistema suporta integridade referencial se ele garante que seus recursos continuarão a existir enquanto houverem referências para estes recursos [ILCS95, ICL96]. Portanto, a falta de integridade referencial na Web se dá quando algum recurso é removido de um computador ligado à Internet, existindo ainda referências para ele.

É praticamente impossível manter os recursos publicados nos servidores por um tempo indeterminado. Um recurso que não tem vida longa na Web, tal como um anúncio de classificados, torna-se "lixo" e conseqüentemente causa sobrecarga nos servidores. Por isso alguns recursos devem ser armazenados em *backups* ou devem ser removidos definitivamente.

O ideal seria que os provedores de informação mantivessem os recursos que ainda estão sendo referenciados e removessem aqueles que não estão, mas esta é uma tarefa muito difícil de ser realizada devido às dimensões da Web.

Um fator que contribui consideravelmente para agravar o problema da integridade referencial na Web é a falta de transparência na migração de recursos. Os recursos na Web

podem migrar de uma máquina para outra ou até mesmo de uma rede local para outra. O problema ocorre quando as referências a estes recursos não são atualizadas, tornando-se, então, inconsistentes.

A migração de recursos na Web faz-se necessária devido a vários fatores, dentre os quais podemos citar: i) a migração no próprio servidor - os recursos podem migrar devido ao aumento da quantidade de recursos armazenados, havendo então a necessidade de reorganizar a base de dados; ii) a migração entre servidores - os recursos podem migrar devido ao número de acessos aos servidores que o armazenam ser elevado, esta migração acontece porque é preciso que os recursos fiquem armazenados em máquinas mais poderosas, tornando mais fácil a sua recuperação.

Uma solução para este problema seria fornecer diretivas de redirecionamento que forneçam um apontador para a nova localização do recurso. Mas esta seria apenas uma solução parcial, pois os recursos que referenciam outros recursos que migraram não são atualizados automaticamente, ou seja, os pedidos continuarão tendo acesso à antiga localização, primeiro, ao invés de ter acesso direto ao recurso; além disso, não podemos prever quantos níveis de redireção poderão estar associados ao recurso, o que dificulta a recuperação do mesmo.

- **Falhas na Infra-Estrutura da Web**

Mesmo quando os recursos existem na Web, e na localização indicada, é possível que ainda ocorra a quebra de ligação. Neste caso, a quebra de ligação ocorre devido a falhas na infra-estrutura que compõe a Web.

Estas falhas ocorrem quando existe algum problema de comunicação na rede, como por exemplo, falhas nas máquinas servidoras, o congestionamento ou particionamento na rede de comunicação entre um usuário e o recurso desejado, e a sobrecarga nas máquinas servidoras. Quando alguma destas falhas acontece, os recursos Web podem tornar-se indisponíveis para alguns usuários da rede.

Estes problemas são agravados quando as rotas são pouco redundantes, ou seja, o número de caminhos entre os componentes do sistema é pequeno. Para ilustrar, no Brasil existem dois *backbones* principais. O *backbone* provido pela Rede Nacional de Pesquisa, RNP (Figura 1.2), responsável principalmente pelo tráfego de informações de instituições de

pesquisa, e o *backbone* provido pela Empresa Brasileira de Telecomunicações (Embratel - Figura 1.3) , que atende principalmente ao setor comercial.

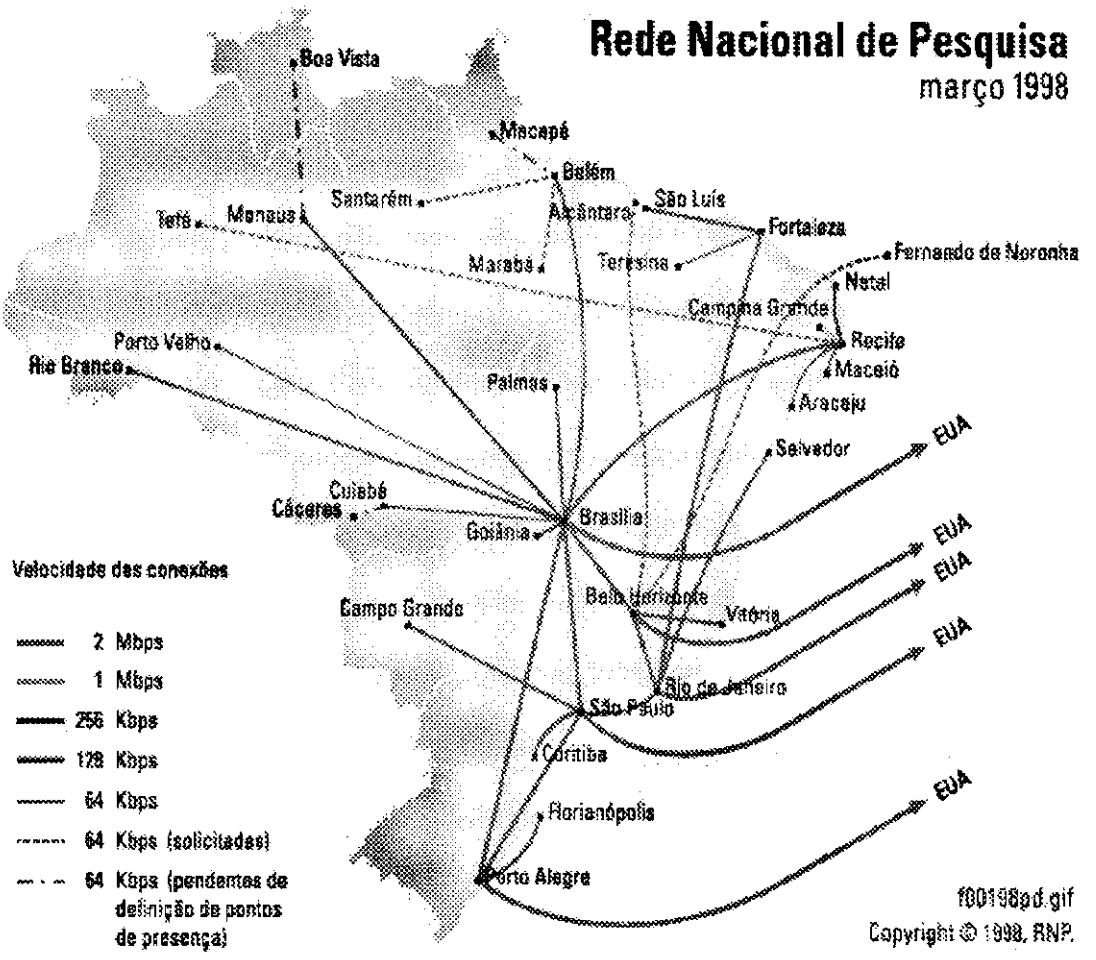

Figura 1.2 -*Backbone* da RNP

O que podemos observar na Figura 1.2 é uma malha de conexões dedicadas interligando quase todas as capitais do Brasil, compondo um *backbone* nacional de 2Mbps em nove capitais e mais cinco linhas de 2Mbps (no Rio de Janeiro, São Paulo, Belo Horizonte, Porto Alegre e Brasília), que conectam a RNP à Internet mundial através dos Estados Unidos. Um problema que pode ser constatado observando-se a Figura 1.2 é a pouca redundância apresentada pelo *backbone*; podemos perceber que praticamente existe apenas uma ligação entre cada dois pontos na Internet brasileira, principalmente naquelas cidades que apresentam uma largura de banda inferior a 2Mbps.

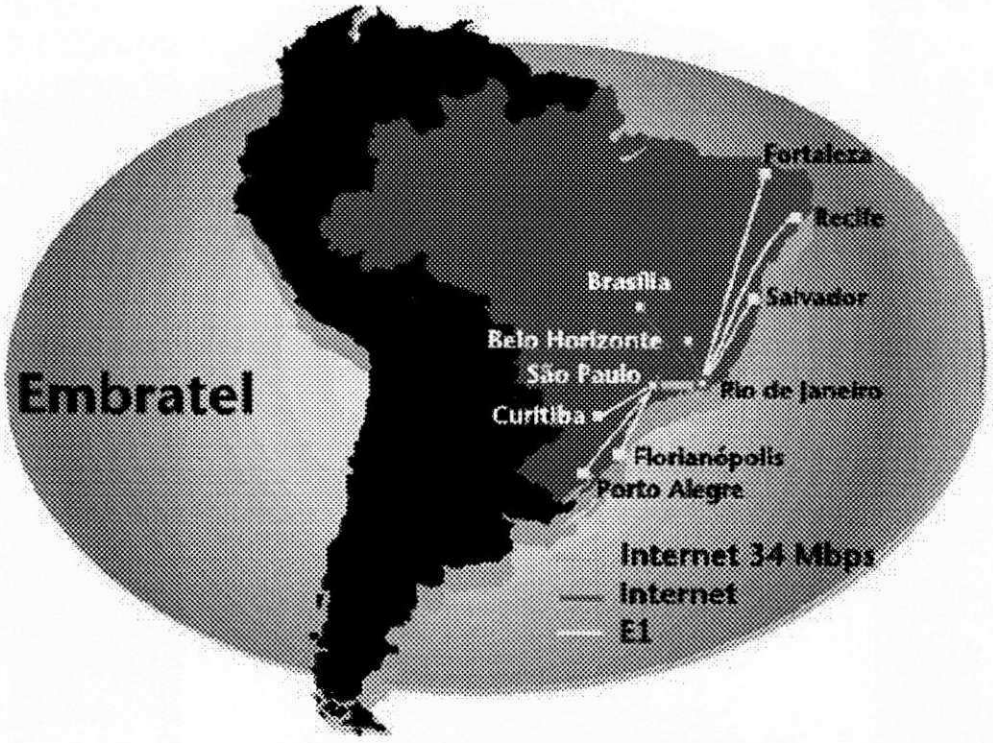

Figura 1.3 - *Backbone* da Embratel

A Figura 1.3 indica a localização dos roteadores[1] e dos principais concentradores E1[2], e apresenta o *backbone* Internet da Embratel. Este é o maior *backbone* da América Latina, tanto em abrangência, como em capacidade de circuitos de transmissão de dados, em níveis nacional e internacional. A figura não apresenta toda a malha atendida pelo *backbone* da Embratel, apenas os pontos principais, mas ele atende mais de 80 localidades em todo o país, das quais 70 já são atendidas com *links* de 2Mbps.

Não estão apresentados na Figura 1.3, mas o *backbone* Embratel disponibiliza 20 *links* de 2Mbps para os Estados Unidos e Canadá e três *links* de 128 Kbps para o MERCOSUL, divididos entre o Rio de Janeiro e São Paulo.

Como se pode ver, devido às dimensões do país, a malha brasileira ainda é deficiente. Os *links* são muito esparsos e um problema de partição em um *link* pode acarretar problemas em diversas áreas, causando o isolamento total da Internet. Também devido a esta deficiência

[1] Roteador é o equipamento responsável pela interligação entre redes locais.

[2] Os concentradores E1 têm a função de concentrar o tráfego local e direcioná-lo até o roteador mais próximo.

na rede, os congestionamentos de informações se tornam muito freqüentes, pois existem muitos gargalos nos *backbones* brasileiros.

1.3 Objetivos e Contribuições do Trabalho

Em um sistema distribuído, não existe tolerância a falhas sem haver redundância [JAL94]. Na Web, redundância significa replicar[3] os seus recursos e as rotas que dão acesso a estes recursos. Através da replicação de recursos, podemos aumentar a disponibilidade e obter um ganho no desempenho do sistema.

Este trabalho é um projeto de pesquisa do Laboratório de Sistemas Distribuídos (LSD), do Departamento de Sistemas e Computação (DSC). Ele faz parte de um projeto maior, chamado *Super Spider*, que visa fornecer serviços Web de alta disponibilidade.

O nosso objetivo, no escopo deste trabalho, é possibilitar a replicação (ou o espelhamento) de informações na Web, de forma a diminuir a indisponibilidade dos seus recursos e gerar caminhos alternativos sobre as rotas já existentes. Nesse trabalho, nós trataremos apenas a indisponibilidade de recursos ocasionada por falhas na infra-estrutura da Web. A indisponibilidade ocasionada pela falta de integridade referencial não será tratada, pois um sistema que se proponha a solucionar este problema deve ter o conhecimento de onde se encontram todas as referências a um determinado recurso publicado na Web, constituindo-se em uma tarefa demorada e bastante complexa.

O acesso aos recursos espelhados será possibilitado via servidor World Wide Web *proxy* com suporte a documentos espelhados. O servidor *proxy* implementará os mecanismos e outros componentes necessários para que os usuários da Web, que utilizarem os nossos serviços, possam ter um acesso tolerante a falhas aos recursos Web. Também propomos um mecanismo de balanceamento de carga (*load balancing*) dos pedidos que são enviados aos servidores que armazenam os recursos espelhados. A informação deve chegar ao usuário de maneira rápida e eficiente, e o balanceamento de carga contribui para que este objetivo seja alcançado.

[3] Replicar (do inglês "*to replicate*") e suas derivações, é um abuso na escrita da língua portuguesa. Este termo é utilizado neste trabalho para expressar a criação de múltiplas cópias de um recurso.

1.4 Organização do Trabalho

O restante do nosso trabalho está organizado em mais cinco capítulos. No segundo capítulo, descrevemos a World-Wide Web, em termos de sua estrutura, componentes principais, componentes auxiliares e o seu funcionamento, como também apresentamos o protocolo utilizado por estes componentes para se comunicar entre si.

No terceiro capítulo, descrevemos os conceitos e as tecnologias que permitem a replicação de recursos na World-Wide Web e apresentamos alguns trabalhos relacionados com o nosso, que também objetivam aumentar a disponibilidade dos recursos Web.

No quarto capítulo, apresentamos o nosso projeto, a arquitetura definida, os recursos utilizados e o seu funcionamento.

No quinto capítulo, tratamos dos detalhes da implementação da nossa proposta, a linguagem de programação utilizada, os recursos computacionais que foram necessários para o seu desenvolvimento e as dificuldades enfrentadas nesta fase do nosso trabalho.

Finalizando, no sexto capítulo apresentamos nossas considerações finais e discutimos sugestões para que a pesquisa possa ter continuidade.

2. A Infra-Estrutura da Web

Neste capítulo, apresentamos e discutimos os principais componentes da Web e alguns aspectos do protocolo que estes componentes utilizam para se comunicar, que são de grande importância para o entendimento do nosso trabalho.

Na primeira seção, apresentamos os componentes básicos que compõem a infra-estrutura básica da Web e o seu funcionamento. Na seção seguinte, estudamos com mais detalhes o protocolo utilizado por estes componentes para se comunicar.

2.1 Componentes Básicos da Web

A Web é constituída basicamente de dois componentes principais, os servidores e os clientes, que se comunicam utilizando o protocolo HTTP (*HyperText Transfer Protocol* - que será estudado na próxima seção) [RFC1945, RFC2068]. Os servidores são comumente conhecidos como servidores Web (ou servidores http) e os clientes são mais conhecidos como *browsers*.

Uma interação usual entre um cliente e um servidor Web ocorre da seguinte maneira: uma mensagem de consulta é enviada do cliente para o servidor; ao receber a mensagem, o servidor analisa a sua base de dados para recuperar as informações relevantes. Uma vez que o

servidor tenha a informação, ele envia os resultados para o cliente através de uma mensagem de resposta. Por fim, o cliente recebe os resultados e os apresenta para o usuário. Um exemplo desta interação pode ser visualizado na Figura 2.1.

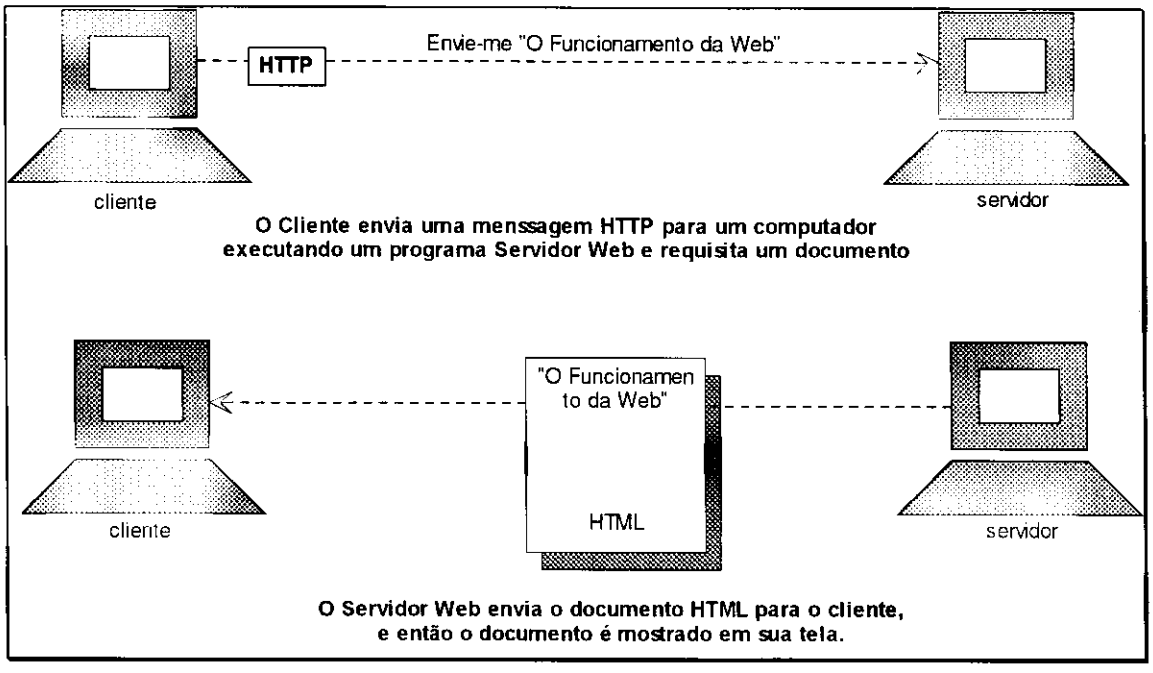

Figura 2.1 Interação entre um servidor e um cliente Web

Os servidores Web são os responsáveis pelo armazenamento e pelo fornecimento dos recursos Web solicitados por outros computadores.

São funções dos servidores Web:

- Receber os pedidos dos clientes;
- processar as consultas;
- retornar os resultados para os clientes;

Atualmente, existem diferentes servidores na Web, executando em diversas plataformas. Alguns dos mais conhecidos e utilizados são o *Netscape Communication Server*, o *Apache Server* e o *Microsoft Server*.

O desempenho dos servidores Web a cada dia tem se tornado um fator de grande importância, pois, como já vimos anteriormente, os servidores são os responsáveis pelo armazenamento e fornecimento dos serviços e recursos Web. São duas as medidas de desempenho de um servidor Web:

- *Throughput*: é a velocidade na qual o servidor pode processar pedidos. É medido em operações HTTP, executadas por segundo (HTTPops/Sec).
- Tempo de resposta: é o tempo que o servidor leva para processar um pedido.

Os clientes Web são os programas utilizados pelos usuários da Web. São de fácil utilização, possuem interfaces gráficas e têm o objetivo de localizar, recuperar e exibir os recursos contidos na Web. Os *browsers* mais conhecidos são o *Netscape Navigator* e o *Microsoft Explorer*.

Os clientes Web são projetados para permitir que os usuários tenham um alto grau de controle sobre sua interação com a Web. Por exemplo, o usuário pode controlar a aparência de seus documentos ou pode chamar aplicações que permitam visualizar recursos que não podem ser visualizados pelo *browser*. Eles também controlam e manejam documentos HTML (*HyperText Markup Language*) [RFC1866], que é a linguagem padrão da World Wide Web, utilizada na criação e publicação de documentos.

Os documentos HTML são conhecidos como documentos hipertextos ou hipermídias [HUGES93, CM93], uma vez que estes documentos incluem ligações (*links*) para outros documentos ou para outras mídias (som, imagem etc.). Estas ligações tornam o acesso a outros documentos, em servidores locais ou em servidores remotos, rápido e simples.

Feitas estas considerações, um cliente Web deve possuir as seguintes características:

- Possuir uma interface gráfica orientada por mouse;
- exibir documentos hipertexto e hipermídia;
- exibir textos em uma grande variedade de fontes;
- oferecer suporte para sons, vídeos, imagens sensitivas, gráficos interativos (GIF, XBM e JPEG) e formulários eletrônicos interativos;
- exibir caracteres acentuados, definidos no conjunto ISO 8859;
- formatar documentos segundo elementos como parágrafos, listas, etc;
- oferecer suporte para *links* hipermídia e para serviços de rede, tais como: FTP, Gopher, Telnet, NNTP e WAIS.

As funções de um cliente Web são:

- Manipular a interface do usuário;
- traduzir o pedido do usuário no protocolo desejado;
- enviar o pedido para o servidor;
- esperar pela resposta do servidor;
- traduzir a resposta do servidor em resultados legíveis para o usuário;
- apresentar os resultados para o usuário.

Além dos componentes principais, servidores e clientes, podem existir agentes intermediários na comunicação entre os servidores e os clientes Web, tais como *proxies* e *gateways*.

Um *gateway* é um agente recebedor de requisições, é um servidor que atua como agente mediador para outros servidores, pode servir como tradutor de protocolos e, em muitos casos, ele atua como se fosse o próprio servidor que deveria receber os pedidos. Um cliente necessariamente não sabe que está se comunicando com um *gateway*. Os *gateways* são geralmente utilizados como tradutores de protocolos para o acesso a recursos que estão armazenados em sistemas que não implementam o protocolo HTTP.

Um *proxy* [AL94] é um programa intermediário que atua em favor dos clientes Web: ele age como um servidor para esses clientes e como cliente para os servidores Web. O servidor *proxy* espera por pedidos dos clientes e repassa estes pedidos para um servidor remoto na Internet, e, quando ele recebe a resposta do servidor, ele a repassa para o cliente que realizou o pedido. Os pedidos podem ser atendidos pelo próprio *proxy* ou serem reescritos (se necessário) e repassados para servidores ou outros *proxies*.

Os *proxies* são freqüentemente utilizados como agentes auxiliares para aplicações que manipulam pedidos via protocolos que o cliente não implementa ou podem ser utilizados como *firewalls* [CZ95], uma medida de segurança que permite limitar o acesso de terceiros a uma determinada rede ligada à Internet.

Nos casos em que clientes não implementem algum protocolo Internet, eles podem ser configurados para direcionarem os pedidos realizados através deste protocolo para servidores *proxies* que o implementem. Assim, os *proxies* enviarão estes pedidos em favor do cliente e retornarão as respostas requisitadas por ele. Observe o exemplo da Figura 2.2.

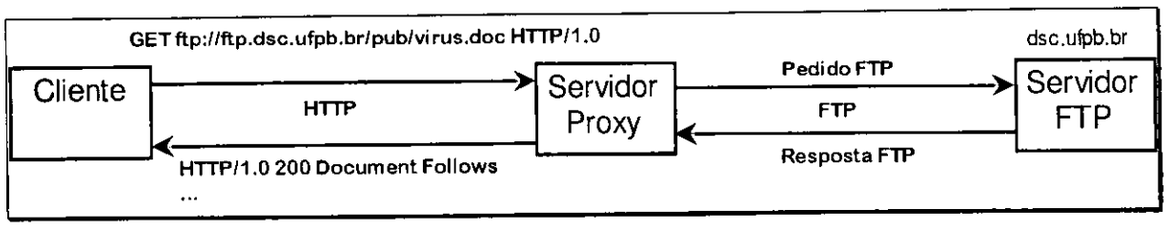

Figura 2.2 - Exemplo de comunicação onde o cliente não implementa o protocolo FTP

No exemplo da figura acima, em um cliente qualquer, que não implementa o protocolo FTP (*File Transfer Protocol* - Protocolo para Transferência de Arquivos), um usuário deseja realizar a transferência de um arquivo que está armazenado no servidor do DSC. Então, este usuário configura o seu cliente para direcionar os seus pedidos para um servidor *proxy* que implemente o protocolo FTP. Assim, o cliente envia um pedido para o *proxy*, utilizando o protocolo HTTP (que é o protocolo que ambos implementam), e requisita que o *proxy* realize esta transferência de arquivo em seu favor, então, o *proxy* envia o pedido FTP para o servidor do DSC, que, por sua vez, retorna o arquivo requisitado para *proxy*, que repassa a resposta para o cliente.

A cadeia de comunicação entre um cliente e um servidor Web com a presença de *gateways* e *proxies* é exemplificada na Figura 2.3 abaixo.

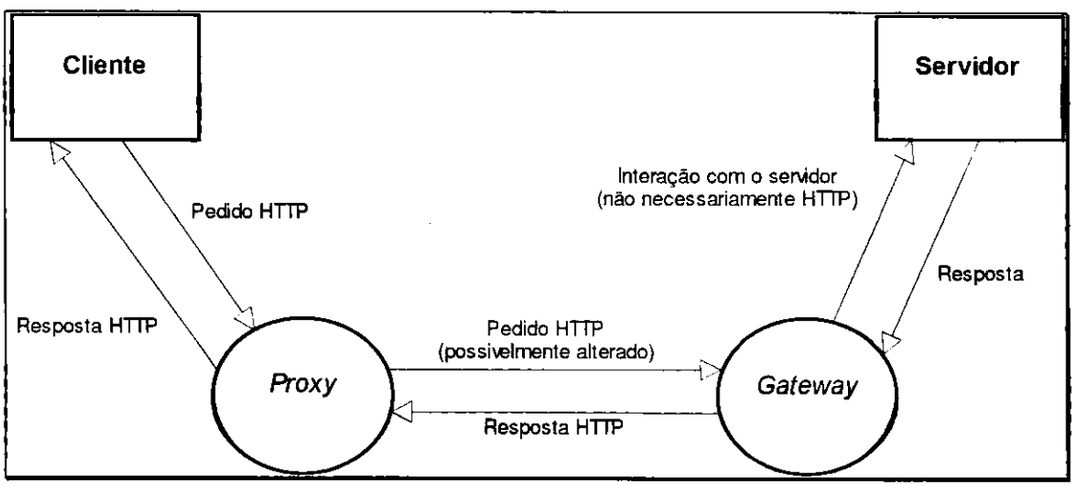

Figura 2.3 - Comunicação com a presença de componentes intermediários

A comunicação entre o cliente e o *proxy* sempre ocorre utilizando-se o protocolo HTTP, pois é o protocolo padrão da Web e que ambos os componentes devem implementar, o que não é verdadeiro para outros protocolos. A iteração entre o *proxy* e outros servidores, ocorre de uma maneira ligeiramente diferente da iteração usual; o *proxy* recebe do cliente um pedido completo, pois o *proxy* deve ter conhecimento de todas as informações necessárias

para que ele possa realizar a conexão com um servidor remoto. Mas, o *proxy* pode modificar este pedido, pois a conexão já foi realizada e nem todas as informações recebidas do cliente precisam ser repassadas para o servidor. O servidor que recebe o pedido do *proxy* só necessita de informações relevantes para a recuperação do recurso, tais como, o caminho na sua hierarquia de diretórios onde ele pode ser encontrado e o nome do recurso a ser recuperado. Já a interação de um *gateway* com os servidores pode ou não ocorrer utilizando-se o protocolo HTTP, pois, como vimos anteriormente, um *gateway* pode ser utilizado como um tradutor de protocolos, e, portanto deve se comunicar seguindo os padrões daqueles protocolos que ele implementa. Assim, para se comunicar com um servidor Web, o *gateway* deve implementar o protocolo HTTP.

É importante observar que o servidor *proxy* deve estar o mais próximo possível dos clientes Web, enquanto que os servidores Web podem estar em qualquer lugar na rede.

Um recurso de grande importância e bastante utilizado por estes componentes, *proxies* e *gateways*, é a memória *cache*. A memória *cache* é utilizada para que o processamento dos pedidos se torne mais rápido, pois ao invés de recuperar o recurso a partir de sua localização original, ele pode ser recuperado da *cache*, diminuindo-se com isso o caminho que este recurso deve percorrer na cadeia de comunicação entre um cliente e um servidor Web.

Considere, por exemplo, a cadeia de comunicação apresentada na Figura 2.2. Se o recurso solicitado em um pedido HTTP feito ao servidor estiver armazenado na memória *cache* do *proxy*, não será necessário que este pedido siga a cadeia de comunicação até o servidor de destino, o *proxy* se encarregará de servir ao cliente. Esta recuperação automática de um documento armazenado na *cache* pode ocorrer em qualquer ponto da cadeia de comunicação.

2.2 HTTP - HyperText Transfer Protocol

O *Hypertext Transfer Protocol* é um protocolo da camada de aplicação da pilha TCP/IP, baseado em pedidos e respostas. O HTTP é um protocolo *stateless*, ou seja, a cada conexão realizada, iniciada quando o cliente envia um pedido, ela é tratada como uma nova conexão, e, quando o servidor envia a resposta, a conexão é finalizada e esquecida.

O HTTP trabalha principalmente com documentos HTML, mas na verdade, está aberto para suportar um ilimitado e extensível conjunto de formatos. Este protocolo também define um conjunto de métodos, que são utilizados para indicar o propósito de um pedido. Esses métodos serão estudados no decorrer desta seção. O protocolo HTTP utiliza o conceito de referência fornecido por uma URI (*Universal Resource Identifier*), que é utilizada para identificar o recurso requisitado em um pedido HTTP. Uma URI nada mais é do que uma cadeia de caracteres que identifica, via nome, localização, ou alguma outra característica, um recurso na rede.

A URI mais utilizada atualmente na Web é a URL (*Uniform Resource Locator*) [RFC1738], que identifica um recurso, baseando-se em sua localização física. Uma URL é composta basicamente de três partes: a primeira, especifica qual o protocolo a ser utilizado para conectar o recurso (como mencionado anteriormente, pode ser um texto, uma imagem, uma música, um arquivo etc.); a segunda, especifica o endereço do servidor onde o recurso está disponível; e a terceira, especifica o caminho (na estrutura hierárquica do servidor) onde se localiza o recurso [SBE96].

Uma URL pode ser iniciada com qualquer um dos protocolos Internet. Além do http, os mais comuns são telnet, ftp e gopher. Observe abaixo um exemplo de URL:

```
http://www.dsc.ufpb.br/~tercio/index.html
```

- http → significa que o recurso será manipulado via protocolo http.
- :// → é o separador requerido em todas URLs, a fim de separar o nome do protocolo e o endereço do recurso.
- www.dsc.ufpb.br → é o endereço do servidor Web do Departamento de Sistemas e Computação (DSC), também chamado de nome do domínio.
- /~tercio/index.html → é o caminho no servidor Web do DSC onde está localizada a página de Tércio (o recurso em questão).

O HTTP define seus próprios formatos de mensagens, que se constituem em pedidos, dos clientes para os servidores, e respostas, dos servidores para os clientes, como definido abaixo[4].

```
mensagem ::=    (pedido | resposta)
                *(<cabeçalho_geral>| <cabeçalho_de_pedido>
                | <cabeçalho_de_resposta> | <cabeçalho_de_
                entidade>)
                LINHA_EM_BRANCO
                [<entidade_da_mensagem>]
```

Um pedido HTTP dever conter no mínimo a identificação do serviço que está sendo solicitado (denominado de método pelo HTTP), a identificação do recurso ao qual será aplicado o método (uma URI) e a versão do protocolo HTTP que está sendo utilizada pelo cliente, atualmente, existem na Web duas versões do protocolo HTTP, o HTTP 1.0 e o 1.1.

```
pedido ::= <método> <URI_requisitada> <versão_do_HTTP>
```

Um exemplo de pedido HTTP é o seguinte:

```
GET http://www.dsc.ufpb.br/~tercio/index.html HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do Departamento de Sistemas e Computação, através do método GET, o recurso identificado pela URL, passada como parâmetro, que identifica a página do aluno Tércio.

O HTTP, em sua versão 1.1, define o seguinte conjunto de métodos [RFC2068]:

> **OPTIONS:** Esse método requisita informações sobre as opções de comunicação disponíveis na cadeia de comunicação entre um cliente e o servidor, identificado pela URI requisitada. O método OPTIONS permite o cliente determinar as opções e/ou os requisitos associados a um recurso, antes de iniciar qualquer ação sobre o recurso.
>
> ```
> OPTIONS http://www.dsc.ufpb.br/copin.html HTTP/1.1
> ```
>
> Este pedido HTTP requisitaria ao servidor Web do DSC que fosse enviado a lista dos métodos implementados por ele e que seriam aplicáveis ao recurso

[4] Nas regras que definem uma mensagem HTTP, o símbolo * significa que a regra pode ser repetida zero ou mais vezes.

identificado pela URL, passada como parâmetro, que identifica a página da COPIN (Coordenação de Pós-graduação em Informática).

GET: Esse método é utilizado para recuperar em um servidor o recurso identificado pela URI requisitada. Se a URI requisitada identifica um processo, a resposta ao método deverá ser a informação gerada pelo processo e não o seu código fonte. Por exemplo:

```
GET   http://www.dsc.ufpb.br/~tercio/index.html
HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do DSC que fosse enviado recurso identificado pela URL, passada como parâmetro, que identifica a página do aluno Tércio.

HEAD: Esse método é semelhante ao método GET, exceto pelo fato de que o servidor não deve enviar o recurso como resposta ao pedido. O método HEAD recupera as meta-informações associadas a um recurso. O método HEAD é útil quando se deseja testar a validade ou a acessibilidade de *links* hipertexto ou verificar a data da última modificação de um recurso sem propriamente recuperar o recurso.

```
HEAD http://www.dsc.ufpb.br/lsd.html HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do DSC que ele enviasse informações associadas ao recurso identificado pela URL, passada como parâmetro, que identifica a página do LSD.

POST: O método POST transfere dados do cliente para o servidor. Ele é utilizado para pedir ao servidor de destino que aceite a entidade anexada ao pedido com uma nova subordinada ao recurso, identificada pela URI passada como parâmetro. O método POST foi projetado para realizar funções, tais como: postar mensagem em *bulletin boards*, *mailing list* etc; processar dados associados a um formulário; adicionar informações em uma base de dados etc. Por exemplo:

```
POST http://www.dsc.ufpb.br/cgi-bin/ferramenta.cgi HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do DSC que processe o recurso anexado ao pedido (a entidade) utilizando o recurso identificado pela URI passada como parâmetro.

PUT: Esse método é utilizado para pedir que um recurso seja armazenado no servidor, e ainda que este recurso seja identificado pela URI passada como parâmetro. Se a URI já se refere a um recurso existente, a entidade recebida deve ser considerada uma versão modificada do recurso existente e deve substituí-lo. A interpretação que é dada à URI passada como parâmetro é o que difere os métodos PUT e POST. No método PUT, a URI passada como parâmetro identifica a própria entidade enviada ao servidor. Já no método POST, a URI identifica o recurso que irá manipular a entidade enviada. Por exemplo:

```
PUT http://www.dsc.ufpb.br/lsd.html HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do DSC, que o recurso anexado ao pedido fosse armazenado e identificado pela URL passada como parâmetro, que identifica a página do LSD.

DELETE: Esse método é utilizado para remover do servidor o recurso identificado pela URI passada como parâmetro. Por exemplo:

```
DELETE http://www.dsc.ufpb.br/lsd.html HTTP/1.1
```

Este pedido HTTP requisitaria ao servidor Web do DSC que removesse o recurso identificado pela URL passada como parâmetro, que identifica a página do LSD.

TRACE: Esse método é utilizado para que o cliente possa visualizar o que está sendo recebido no outro lado da cadeia de comunicação. O método TRACE é bastante útil na realização de testes e diagnósticos do sistema. A mensagem recebida como resposta a um pedido utilizando o método TRACE deve conter a própria mensagem enviada e nenhuma outra entidade.

```
TRACE http://www.dsc.ufpb.br/ HTTP/1.0
```

Este pedido HTTP poderia ser utilizado para verificar a existência de componentes intermediários na cadeia de comunicação entre o cliente e o

servidor Web do DSC, e também para verificar resposta enviada pelo servidor ou algum destes componentes intermediários, quando o cliente desejasse recuperar o recurso identificado pela URL, passada como parâmetro, que identifica a página do DSC.

Já uma resposta HTTP dever conter ao menos a versão do protocolo e um código de *status* numérico, seguido de uma frase explicativa.

```
resposta ::= <versão_do_HTTP> <código_status> <frase>
```

Um exemplo de resposta HTTP é o seguinte:

```
HTTP/1.1 200 Document Follows
```

Esta resposta indica que o recurso requisitado através do pedido realizado com o protocolo HTTP 1.1 segue como a entidade da resposta.

O código numérico retornado em uma resposta HTTP é para ser utilizado pelo cliente, ele representa o resultado do processamento executado pelo servidor, e a frase explicativa serve de referência para os usuários. O cliente não deve examinar a frase e sim o código. O código de *status* retornado se divide em cinco classes, e estas são definidas pelo primeiro dígito do código. Estas classes são as seguintes:

- 1xx : Informacional - pedido recebido, continuando o processo. Geralmente respostas que possuem este código de *status* são enviadas pelo servidor para informar ao cliente que o pedido foi aceito com sucesso e que será processado, exemplo: "100 *Continue*";
- 2xx : Sucesso - a ação foi recebida, entendida e aceita com sucesso. Geralmente respostas que possuem este código de *status* são enviadas pelo servidor quando não existe nenhum problema com relação ao pedido requisitado pelo cliente, o servidor irá processar o pedido e enviar a resposta de acordo com o que o cliente solicitou, exemplo: "202 *Accepted*", significa que o servidor aceitou o pedido de solicitação de um recurso;
- 3xx : Re-direção - mais ações devem ser realizadas para que o pedido possa ser completado. Geralmente respostas que possuem este código de *status* são enviadas pelo servidor quando um cliente requisita um recurso que não

está mais armazenado nele, por exemplo: "301 *Moved Permanently*", significa que o recurso foi removido permanentemente do servidor;

- 4xx : Erro no Cliente - o pedido contém erro de sintaxe. Geralmente respostas que possuem este código de *status* são enviadas pelo servidor quando um cliente envia um pedido que não possa ser executado ou que contenha uma URI com erros de sintaxe, exemplo: "404 *Not Found*", significa que o recurso referenciado pela URI não foi encontrado no servidor;
- 5xx : Erro no Servidor - o servidor falhou. Geralmente respostas que possuem este código de *status* são enviadas pelo servidor quando ele não implementa o método requisitado no pedido enviado pelo cliente, ou quando ocorre algum problema no seu funcionamento, exemplo: "501 *Not Implemented*", significa que o servidor não implementa o método requisitado no pedido.

Opcionalmente, as mensagens HTTP podem conter cabeçalhos, nos quais são passados parâmetros que podem modificar a execução do método solicitado ou modificar o formato da resposta enviada por um servidor. Alguns métodos definidos pelo HTTP podem manipular recursos, que são passados como parâmetros, desta forma, algumas mensagens podem conter o próprio recurso. A parte da mensagem que contém este recurso é denominada de entidade pelo HTTP.

O cabeçalho é uma informação genérica que pode, entre outras coisas, especificar a ação requerida no servidor, o tipo de dado que está sendo transferido ou um código de *status*. O uso dos campos de cabeçalho dá ao protocolo HTTP uma enorme flexibilidade. Estes campos permitem o uso de informações descritivas para serem enviadas na transação, fornecendo possibilidades para autenticação, encriptação e/ou identificação do usuário. O cabeçalho é referenciado como meta-informação, por se tratar da informação da informação.

Os cabeçalhos das mensagens HTTP são compostos de campos (chamados de campos de cabeçalhos). Estes campos podem ser específicos de pedidos, que permitem o cliente passar mais informações sobre ele para o servidor; de respostas, que permitem o servidor passar informações adicionais sobre ele e a resposta, que não podem ser passadas na linha de *status*; de entidades, que permitem a inclusão opcional de meta-informações sobre o recurso

identificado na mensagem; ou serem de propósitos gerais, que contém dados que não estão relacionados com as partes que estão se comunicando.

Os campos de cabeçalho são formados por um nome, seguido de dois pontos e o valor do campo. Alguns dos campos mais comuns definidos pelo HTTP 1.1 [RFC2068] são:

Accept: é um campo de cabeçalho de pedidos que é utilizado para especificar certos tipos de mídias que serão aceitas como resposta pelo cliente. Exemplo: `Accept: image/gif;image/jpg`, significa que os tipos de imagens aceitas como resposta ao pedido terão que ser do tipo "gif" ou "jpg".

Allow: é um campo de entidade que é utilizado para listar o conjunto de métodos suportados pelo recurso identificado pela URI passada como parâmetro. Exemplo: `Allow: GET, HEAD`.

Date: é um campo de cabeçalho de propósito geral que é utilizado para representar a data e o tempo no qual a mensagem foi originada. Exemplo: `Date: Wed, 15 Nov 1995 10:50:20 GMT`.

If-Modified-Since: é um campo de cabeçalho de pedidos que é utilizado como modificador do método GET, para torná-lo condicional. Se o recurso requisitado não tiver sido modificado desde a data especificada no campo, o servidor não deve retornar o recurso. Exemplo: `If-Modified-Since: Sat, 29 Oct 1997 20:45:11 GMT`.

Location: é um campo de cabeçalho de respostas que é utilizado para redirecionar o cliente para uma localização diferente daquela passada como parâmetro ou para identificação de um novo recurso. Exemplo: `Location: http://www.dsc.ufpb.br/novapagina.html`.

Um exemplo completo de uma transação HTTP é descrito abaixo:

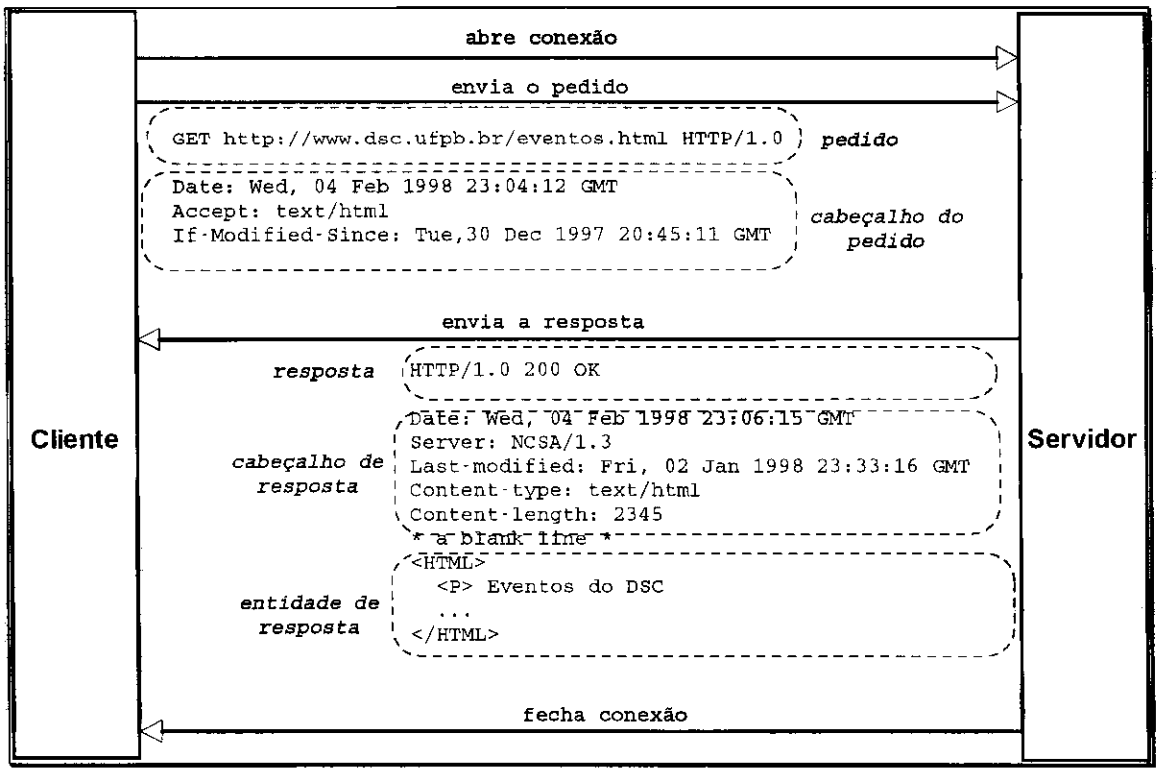

Figura 2.4 - Exemplo de uma transação HTTP

Observando a figura acima, vemos um exemplo de mensagens sendo transferidas entre um cliente e um servidor Web através do protocolo HTTP. Primeiro, o cliente estabelece a conexão com o servidor; e em seguida, ele envia a sua mensagem de pedido. Utilizando o protocolo HTTP versão 1.0, o cliente requisita através do método GET, que o servidor Web do DSC envie a página HTML que contém informações sobre os eventos que irão ocorrer no primeiro semestre do ano no DSC. Como vimos anteriormente, cabeçalhos podem alterar a execução de um método, neste caso, o cliente só deseja recuperar o recurso se ele foi modificado entre o dia 30 de dezembro e a data do envio do pedido e este recurso deve estar no formato HTML, condicionando assim a ação do método GET. Por isso, a mensagem de pedido enviada pelo cliente contém um cabeçalho.

No espaço reservado para o cabeçalho do pedido, são definidos três campos, que serão analisados pelo servidor quando ele processar o pedido, são eles: `date`, que informa a data na qual o pedido foi enviado pelo cliente; `accept`, que indica o tipo dado que ele aceitará como resposta (neste caso, HTML); e `if-modified-since`, que informa ao servidor que se a página de eventos não foi modificada desde 30 de dezembro, o servidor não precisa enviar o recurso.

Recebido o pedido e analisados os dados enviados pelo cliente, o servidor inicia o seu processamento. Como especificado pelo cliente, o servidor (no endereço www.dsc.ufpb.br) deve executar o método GET para recuperar o recurso (eventos.html) armazenado em sua base de dados. Depois de encontrar o recurso, e antes de enviá-lo para o cliente, o servidor deve analisar a data em que ele foi modificado e em que tipo de dado ele está armazenado, como determinado pelo cliente. Constatado que as condições foram satisfeitas, o servidor envia a resposta com o recurso, caso contrário o servidor enviaria uma mensagem de erro para o cliente.

A mensagem de resposta enviada por este servidor está estruturada da seguinte forma: A linha de resposta, que contém o código de *status* (200) que indica que o pedido foi processado com sucesso; os campos de cabeçalho, que são os seguintes: `date`, que informa a data do envio da resposta; `server`, que identifica o servidor que enviou a resposta; `last-modified`, que informa que o recurso foi modificado em 02 de janeiro, por isso que o servidor está o enviando; `content-type`, que identifica o tipo de dado enviado como resposta, uma página HTML, como solicitado pelo cliente; e `content-length`, que informa o tamanho (em bytes) da entidade que está sendo enviada; e a última parte da mensagem, o recurso requisitado pelo cliente, que se constitui na entidade da resposta, ou seja, uma página HTML que descreve os eventos do DSC. Depois que o servidor enviou a mensagem de resposta para o cliente a conexão é fechada. Em um próximo pedido, todo o processo será reiniciado.

3. Replicação de Recursos na Web

Como mencionado anteriormente, a identificação e nomeação de recursos realizadas na Web atualmente são feitas quase que exclusivamente por meio de URLs. Elas realizam um mapeamento um-para-um entre o recurso e seu endereço físico na rede. Entretanto, ter uma única cópia de um recurso reduz a confiabilidade da Web, já que falhas nas máquinas servidoras e particionamentos na rede de comunicação podem tornar os recursos indisponíveis para todos os clientes, ou parte deles.

Uma alternativa para solucionar este problema é fornecer suporte para o espelhamento de recursos na Web, que consiste em replicar os recursos em servidores Web tradicionais, fazendo múltiplas cópias de um recurso disponível para os clientes.

Para que o espelhamento de recursos seja possível, são necessárias a criação e a utilização de um esquema de nomeação que seja independente da localização do recurso e que forneça um serviço que converta URIs em uma ou mais URLs, que identificam as cópias físicas de um mesmo recurso.

Este capítulo está estruturado da seguinte forma: Na seção 3.1, definiremos e discutiremos um esquema de identificação e nomeação de recursos, independente de localização, que vem sendo estudado e desenvolvido por diversas organizações acadêmicas e

de pesquisa. Além disso, apresentaremos na seção 3.2 alguns trabalhos relacionados, que também visam solucionar o problema de indisponibilidade dos recursos na Web.

3.1 Uniform Resource Names

Uniform Resource Names [RFC2141], ou simplesmente URNs, são URIs que têm o propósito de fornecer um identificador único, global, persistente e independente da localização, utilizado no reconhecimento de um recurso, no acesso às características de um recurso ou no acesso ao próprio recurso.

URNs são compostas de URCs (*Uniform Resource Characteristics*), que mantêm meta-informações sobre os recursos, e de uma ou mais URLs, que especificam a localização dos recursos.

As URCs são utilizadas para descrever os recursos e devem conter informações tais como título e autor de um documento HTML na Web, além de informações administrativas, como data da criação do recurso. As URCs devem possuir as seguintes características básicas [IAN96]:

- Serem úteis a vários tipos de recurso;
- possuírem metadados administrativos;
- e terem uma implementação simples, para que se tenha um desenvolvimento rápido.

Atualmente, não existe nenhuma concordância com relação ao conjunto de meta-informações que pode ser utilizado para identificar recursos. Mas, esta estrutura deve conter informações suficientes que permitam sumariar o recurso, e devendo ser sucinta o suficiente para que seja gerenciável. Também é desejável que a estrutura definida sirva para todos os recursos existentes, pois é preciso evitar os problemas de interoperabilidade que podem ser gerados pela existência de vários conjuntos de meta-informações [ISL96].

Mas, na definição de URCs, é importante observar que alguns elementos que descrevem um recurso podem ser omitidos, pois, para alguns recursos, esses atributos podem não ser aplicáveis. Além disso, é importante destacar que os atributos podem ser multivalorados, como autor, por exemplo.

Para especificar URNs, um conjunto mínimo de requisitos funcionais e de codificação deve ser levado em consideração. Estes requisitos funcionais são definidos em [RFC1737], e são os seguintes:

- Escopo Global: uma URN deve ter o mesmo significado em qualquer lugar;
- Unicidade Global: nunca uma mesma URN será atribuída a dois recursos diferentes;
- Persistência: o tempo de vida de uma URN é permanente, ou seja, será único e global para sempre;
- Escalabilidade: URNs podem ser atribuídas a qualquer recurso disponível na rede;
- Suporte ao Legado Existente: o esquema de URNs deve suportar os esquemas já existentes, desde que estes atendam aos outros requisitos aqui definidos;
- Conversão: uma URN não impedirá a conversão para outros esquemas, como por exemplo URLs;
- Extensibilidade: qualquer esquema de URNs deve permitir futuras extensões;
- Independência: a autoridade que caracteriza uma URN é a única responsável por ela.

Com relação à codificação, ou seja, como as URNs são codificadas em uma cadeia de caracteres, os requisitos são:

- Codificação Única: a codificação de URNs deve ser apresentada às pessoas em uma forma textual clara e simples;
- Comparação Simples: um algoritmo de comparação de URNs deve ser simples, local e determinístico. Isto é, o algoritmo para comparar duas URNs não deverá requerer qualquer contato com servidores externos;
- Humanamente Traduzíveis: para que URNs sejam facilmente traduzidas sem erros por seres humanos é necessário que sejam curtas, usem uma quantidade mínima de caracteres especiais e sejam "insensíveis" ao contexto

(*case insensitive*), ou seja, não façam distinção entre letras maiúsculas e letras minúsculas;

- Transportáveis: uma URN deve ser transportada sem modificações em qualquer Protocolo Internet, tais como FTP e Telnet;
- Analisáveis por Máquinas: ou seja, uma URN deverá ser analisada por um computador.

3.1.1 A Sintaxe das URNs

A sintaxe proposta para URNs [RFC2141] fornece meios de se codificar caracteres de dados de forma que possa permitir que estes sejam enviados nos protocolos existentes. Todas as URNs devem possuir a seguinte sintaxe:

```
<URN> ::= urn: <NID> : <NSS>
```

onde: NID é o *Namespace IDentifier* e NSS é o *Namespace Specific String*.

O NID é quem determina a interpretação sintática do NSS, e é quem os distingue com relação aos outros esquemas de nomeação existentes. O NSS por sua vez é definido e governado pelas regras do NID [ISL96]. O NSS também possui três propriedades básicas:

- é significativo somente no contexto do seu NID;
- é único no espaço identificado pelo NID;
- não tem nenhuma estrutura pré-determinada - qualquer estrutura que venha a ter, estará associada ao NID e não terá base em qualquer rede física.

Na sintaxe definida abaixo, construções com o símbolo "*", antecedendo um elemento, significam a repetição deste elemento. A sua forma geral é <n>*<m>, indicando no mínimo <n> e no máximo <m> ocorrências do elemento.

Os NIDs são *case-insensitive* e têm a seguinte sintaxe:

```
<NID> ::= <let-num> [ 1*31<let-num-hyp> ]
<let-num-hyp> ::= <upper> | <lower> | <number>  | -
<let-num> ::= <upper> | <lower> | <number>
<upper> ::= A | B | C | D | E | F | G | H | I | J | K | L
| M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
```

```
<lower> ::= a | b | c | d | e | f | g | h | i | j | k | l
| m | n | o | p | q | r | s | t | u | v | w | x | y | z
<number> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Os NSSs têm a seguinte sintaxe:

```
<NSS> ::= 1*<URN chars>
<URN chars> ::= <trans> | %<hex><hex>
<trans> ::= <upper> | <lower> | <number> | <other> |
<reserved>
<hex> ::= <number> | A | B | C | D | E | F
<other> ::= ( | ) | + | , | - | . | : | = | ? | @ | ; |
$ | _ | ! | * | ' | ~
<reserved> := % | / | ? | #
```

A referência [RFC1630] reserva os caracteres definidos em `<reserved>` para usos específicos. Estes caracteres ainda não possuem uma semântica precisa, para que possam ser utilizados em URNs, eles são reservados para futuros desenvolvimentos. É recomendável não utilizá-los no momento, mas a sua utilização não é proibida. Além disto, para evitar confusões com o identificador "urn:", o NID "urn" é reservado e não deve ser utilizado.

Devido a vários propósitos, tal como *caching*, é freqüente a necessidade de se determinar se duas URNs são as mesmas, sem necessariamente resolvê-las [RFC2141]. Para isto devemos testar a equivalência léxica entre URNs. Duas URNs são lexicamente equivalentes se elas são iguais caracter a caracter, como os quatro exemplos abaixo:

- urn:teste:abc123
- URN:teste:abc123
- urn:TESTE:abc123
- urn:teste:ABC123

3.1.2 Resolução de URNs

Para que seja possível utilizar URNs faz-se necessária a existência de um serviço acessível pela rede e que mapeie um nome no recurso correspondente. Este serviço é chamado de resolvedor e deve estar de acordo com alguns requisitos [SLO96]:

- Deve ser projetado para que URNs tenham vida longa, e devem possuir pouca informação que possa mudar com o tempo;
- deve ser capaz de mudar sem que a mudança seja visível para a URN;
- deve ser escalável (proporções globais);
- deve ser modular o suficiente para que várias partes (algoritmos de resolução, banco de dados, protocolos etc.) possam ser alteradas independentemente das outras;
- deve ser descentralizado; uma única organização não deve ter o controle absoluto sobre todo o *namespace* de uma URN.

Os serviços de resolução se constituem basicamente em N2L(s), N2R(s), N2C(s) e N2N, onde: N2L(s) é o serviço de resolução que mapeia uma URN em uma ou mais URLs; N2R(s) é o serviço de resolução que mapeia uma URN em um ou mais recursos; N2C(s) é o serviço de resolução que mapeia uma URN em uma ou mais URCs e N2N é o serviço de resolução que mapeia uma URN em outra URN.

3.1.3 Nomeação e Registros URN

Um fato importante a considerar é a atribuição das URNs aos recursos quando se deseja utilizar uma URN para identificar um recurso. A organização responsável pelo NID da URN é que deverá ser a responsável por esta atribuição, devendo criar o seu esquema de atribuição de nomes, que determine um único NSS que identifique o recurso.

Uma proposta ainda em desenvolvimento no âmbito do Internet Engenieer Task Force - IETF[5], se constitui em registros URN, que seriam acessíveis pela rede que armazenariam dados sobres os esquemas de URNs e seus respectivos sistemas de resolução. A necessidade

[5] O IETF é uma comunidade internacional composta de projetistas de redes, vendedores, operadores e pesquisadores preocupados com a evolução da arquitetura da Internet e com a sua operação.

de criar estes registros advém do fato de que existem muitos esquemas de URNs sendo desenvolvidos em todo o mundo. O formato dos dados que serão armazenados nos registros ainda está sendo estudado e definido.

A funcionalidade dos registros URN reside na idéia de que de posse de uma URN o cliente deve ter o conhecimento de qual sistema de resolução é capaz de resolver esta URN e também saber qual a organização responsável pela atribuição das URNs.

3.1.4 URNs Versus URLs

O esquema de identificação mais utilizado atualmente na Internet é a URL. Ela apresenta algumas desvantagens com relação às URNs, a saber:

- A natureza transitória das informações codificadas em uma URL - URLs, que são essencialmente uma fórmula de instruções para se ter o acesso à informação apontada pela URL [HAM95]. As URLs especificam mais a localização do recurso, um arquivo ou uma máquina particular, do que a sua identificação.
- Devido à migração e renomeação de recursos, o problema de quebra de ligação é constante com a utilização de URLs. As URLs se apresentam como uma estrutura frágil quanto a construção de uma infra-estrutura de acesso a informações globais, principalmente na Web onde é raro o contato entre as pessoas que citam *links* e aqueles responsáveis pelos recursos apontados por estes *links*.
- As URLs dificultam a replicação, pois, como mencionado anteriormente, baseiam-se na localização do recurso. Assim, todos os pedidos a este recurso são direcionados a um só servidor, ocasionando com isto a sobrecarga nos servidores e atrasos na rede devido ao congestionamento de informações.
- A informação contida em uma URL recupera uma dada instância de um recurso e não fornece nenhum meio de se associar informações extras sobre o recurso. Em muitos casos, o único meio de se obter estas informações é realizar o *downloading* e estudá-lo manualmente.

Já as *Uniform Resource Names* trazem consigo algumas vantagens, são elas:

- A URN une as noções de identificação (URCs) e de localização (URLs).

- Os recursos podem migrar na rede sem ser necessário atualizar páginas que em todo o mundo apontam para eles.
- As URNs suportam a replicação de recursos, fornecendo com isso acessos tolerantes a falhas na Web.
- A disponibilidade de se obter meta-informações é útil para clientes e usuários determinarem se um recurso é relevante ou não antes de o recuperarem.
- E, por último, através da disponibilização de meta-informações, um cliente pode escolher uma instância disponível do recurso no local mais próximo, provendo assim uma utilização mais racional da largura de banda da rede.

Infelizmente, as URNs também possuem algumas desvantagens [DAN96]. Primeiro, ainda existirão problemas de quebra de ligação: embora as URNs forneçam um alto grau de tolerância a falhas na Web, sempre existirão casos onde recursos deverão ser removidos. A idéia da URN ser um identificador único e persistente, ou seja, não poder ser reutilizada para identificar outro recurso, não indica que ela irá existir para sempre. Por exemplo: uma URN que identificasse uma página que mantém informações sobre um concurso público da UFPB, alguns meses depois esta página não teria mais importância para a universidade e seria removida do servidor. O problema é que esta URN não poderia ser utilizada para identificar outro concurso que a UFPB promovesse, e outras páginas Web (principalmente de outras universidades) manteriam *links* para esta pagina do concurso, que não existiria mais, e quando usuários utilizassem estes *links* veriam uma mensagem de erro na tela do *Browser*.

Segundo, alguns projetos de URNs não estão desenvolvendo URNs curtas e amigáveis ao usuário como se pretendia. Observe esta URN: urn:isbn:0-262-12186-7, que por ser imaginária, serviria para identificar livros que foram publicados na Internet através de seu ISBN; é pouco provável que uma pessoa saiba qual o livro que esta URN esteja identificando na Internet, apesar de ser curta, esta URN não seria facilmente entendida por um usuário, pois nem todos os usuários (leitores) de livros conhecem o sistema ISBN.

3.2 Trabalhos Relacionados

A idéia tentar solucionar o problema de quebra de ligação que ocorre atualmente na Web não é exclusiva da nossa proposta. Um fator importante que diferencia cada um dos

projetos existentes é o modo pelo qual o problema é combatido, que pode variar de acordo com os fatores que o ocasionam.

Por exemplo, a idéia de prover a Web com um mecanismo que suporte recursos espelhados, através da utilização de um esquema de nomeação global, visa combater o problema de quebra de ligação ocasionados por falhas na infra-estrutura que suporta a Web. Os fatores principais que diferenciam cada um dos projetos existentes são: o modo pelo qual os resolvedores são encontrados, os próprios resolvedores (a sua implementação) e o esquema de nomeação utilizado (no nosso caso as URNs).

O IETF (*Internet Engineering Task Force*) formou o URN *Working Group*, que visa estudar os problemas de nomeação e identificação de recursos na Web. Estes problemas são discutidos através de listas de discussão e da elaboração de documentos (RFC - Request For Comments, Internet Drafts etc.).

A abordagem mais difundida e proeminente é a que está sendo desenvolvida por Daniel e Mealling,, que está definida em [DM97]. O grupo que desenvolve este esquema de resolução de URNs também faz parte do URN *Working Group*. Este método de resolução de URNs consiste na realização de uma série de consultas ao DNS e em reescritas de uma URI para poder localizar um resolvedor e, com isso, poder mapear a URN em várias URLs, ou localizar o próprio recurso. O processo de reescrita de uma URI transforma uma URI em um determinado nome de domínio e é realizado através da utilização das expressões regulares.

Nesta proposta, o problema de localizar um recurso é dividido em duas fases. A primeira fase é localizar o "resolvedor" em uma base de dados que contém informações sobre URNs, a segunda fase é se comunicar com este resolvedor para obter o recurso ou a sua localização.

Na abordagem de Daniel e Mealling são criados novos registros de recursos DNS, que são chamados de registros NAPTR (*Name Authority Pointer*), e que são distribuídos em uma versão modificada do BIND[6]. As regras para a reescrita de uma URI são armazenadas nestes novos registros e são expressa por meio de expressões regulares.

[6] BIND - Berkeley Internet Name Domain - A implementação mais popular do DNS para o ambiente UNIX.

Localizar um resolvedor através do processo de reescrita pode levar alguns passos, mas o início do processo é sempre o mesmo. O primeiro passo é analisar a URN e extrair o prefixo "unr:", que identifica que o esquema de URNs está sendo utilizado; também é necessário extrair da URN o prefixo que identifica o tipo de URN, o NID. Em seguida, é adicionado a este NID o sufixo "`urn.net`", constituindo-se assim em um nome de domínio, e então realiza-se uma consulta aos registros do DNS do tipo NAPTR, para este determinado nome de domínio. Baseado no resultado da consulta, zero ou mais consultas podem ser necessárias para a localização dos resolvedores de uma determinada URN.

São apontados alguns problemas com relação a esta abordagem. Primeiro, o procedimento de reescrita pode levar muitos passos e como o cliente é o responsável por toda a reescrita, a comunicação entre o cliente e os registros torna-se então intensa, gerando muito tráfego na rede e requerendo uma maior largura de banda nos canais. Segundo, é que em 95% dos casos, estes múltiplos passos na reescrita de uma URN são desnecessárias, pois seria necessária apenas uma reescrita para se resolver uma URN. Terceiro, expressões regulares são difíceis para os pessoas construírem corretamente e, por isso, existem preocupações quanto a usabilidade e a manutenção das regras. E por último, a proposta utiliza-se do sistema DNS, e com isso traz algumas preocupações com relação a segurança e a administração das regras.

Como veremos mais adiante, a nossa proposta também faz uso do sistema DNS. Mas, ao invés de criar novos registros e utilizar as bases de dados existentes, nós propomos a criação de uma base de dados própria (que utiliza registros já definidos no sistema DNS), que será administrada pela organização ou pessoa responsável pelo gerenciamento das URNs (trabalho proposto por [COS98]).

Em suma, o objetivo principal da abordagem de Daniel e Mealling é desenvolver um sistema que seja capaz de resolver URNs, para que estas se tornem o esquema padrão de identificação de recursos na Web, e com isso possamos recuperar uma das várias instâncias do recursos identificado por uma URN.

Já o objetivo principal de nossa proposta é utilizarmos o esquema de URNs e desenvolvermos um esquema de resolução de URNs para que seja possível espelhar recursos na Web e, assim, fornecer uma maior disponibilidade dos recursos e minimizar o problema da quebra de ligação ocasionado por falhas na infra-estrutura da Web.

Outro importante trabalho, que também visa solucionar os problemas de quebra de ligação e indisponibilidade dos recursos na Web, mas não utiliza URNs, é o **W3O** (***W3 Objects***), desenvolvido por um grupo de pesquisadores da Universidade de *Newcastle*.

O objetivo primário do W3O é desenvolver uma infra-estrutura na Web que seja capaz de suportar uma vasta quantidade de recursos e serviços e não somente recursos baseados em arquivos, como existe atualmente na Web [ILCS95]. Para alcançar este objetivo, o W3O faz uso dos conceitos de orientação a objetos.

Na abordagem W3O, os recursos Web são transformados de recursos baseados em arquivos em recursos baseados em objetos, os *W3Objects*. Os *W3Objects* são recursos encapsulados que possuem estado interno e um contexto bem definido. Os próprios objetos são os responsáveis pelo gerenciamento de suas transições de estados e suas propriedades (persistência, controle de concorrência etc.) em resposta às invocações de métodos.

W3Objects podem suportar um número distinto de interfaces, obtidas via herança de interface. Interfaces comuns podem ser compartilhadas de modo a possibilitarem o acesso polimórfico, que significa que mensagens comuns podem ser enviadas aos objetos da classe raiz e a todos os objetos das classes derivadas. Por exemplo, todos os *W3Objects*, de acordo com a interface HTTP que fornecem os métodos `httpGet()` e `httpPost()`.

Assim como a Web hoje, a arquitetura do W3O consiste de três entidades básicas quais sejam: clientes, servidores e objetos publicados, que correspondem respectivamente a Browsers, HTTP *Daemons* e os recursos Web.

A arquitetura W3O fornece o suporte tanto para a comunicação entre clientes e objetos (a tradicional interação cliente/servidor) quanto para a comunicação entres os próprios objetos. A comunicação entre os objetos é realizada via RPC[7] (*Remote Procedure Call* - Chamada a Procedimento Remoto), principalmente para os propósitos de *caching*, replicação, referência e migração. Por exemplo, um objeto pode ter um método que redirecione os pedidos realizados para a obtenção de um recurso que migrou.

O acesso aos *W3Objects* é fornecido através de um *gateway*, implementado como um *plug-in* em algum servidor Web extensível, como por exemplo, o servidor Apache. Os

[7] RPC é um protocolo que um programa pode utilizar para requisitar um serviço de um programa localizado em outro computador em outra rede sem ter que entender os detalhes de comunicação em rede.

servidores Web são configurados para passarem os pedidos aos *W3Objects* para o módulo *gateway* quando, por exemplo, as URLs iniciarem com `/w3o/`. O módulo então liga o objeto nomeado ao pedido com o servidor e invoca o método apropriado, `httpGet()` (que recupera o recurso) por exemplo.

Na abordagem W3O, o problema de integridade referencial é combatido através da manutenção de um grafo de referências distribuído e da utilização de um contador de referências para detectar objetos não referenciados. A transparência na migração é obtida com a utilização de *stubs*. Quando um recurso (objeto) migra, um *stub* representando o objeto é deixado na antiga localização do objeto, ele é o responsável pelo redirecionamento dos pedidos para a nova localização do objeto.

A abordagem W3O é definida e descrita com maiores detalhes em [ILC95, ICL96], na Web pode ser encontrada no seguinte endereço: `http://w3objects.ncl.ac.uk/pubs`.

Com relação ao W3O, o seu objetivo principal é solucionar o problema de quebra de ligação ocasionado pela falta de integridade referencial. Mas esta abordagem possui alguns problemas. Primeiro, será muito difícil (mas não é impossível) tornar a Web um ambiente totalmente orientado a objetos devido as suas dimensões globais e a quantidades de servidores, clientes e recursos existentes atualmente (são milhões espalhados por todo o mundo). Segundo, no tratamento que se dá a transparência na migração podem ser geradas longas cadeias de *stubs* quando os recursos migram várias vezes. Longas cadeias possuem mais pontos de falhas e com isso diminui-se o desempenho do sistema.

4. Suportando o Acesso a Recursos Web Espelhados

Visto que o espelhamento de recursos na Web consiste em replicar os recursos em servidores Web tradicionais, e que existe um esquema de identificação e nomeação que é independente da localização dos recursos, é preciso prover na Web um mecanismo que possibilite o acesso a estes recursos. Além disso, é necessário definir o nosso próprio esquema de identificação e nomeação de recursos.

Uma arquitetura que fornecesse o suporte necessário para que espelhássemos recursos nos permitiria distribuir réplicas de recursos em toda a Web, permitindo que os clientes pudessem automaticamente recuperar a cópia disponível mais "próxima", proporcionando um menor tráfego na rede, e, conseqüentemente, um ganho no desempenho.

Neste capítulo, apresentamos a nossa proposta de um suporte para o acesso a recursos Web espelhados. Essa funcionalidade pode ser implementada em um servidor *proxy*. O servidor *proxy* deve ser capaz de aceitar e processar pedidos HTTP que contenham URNs

como identificadores de recursos e tratar os pedidos HTTP contendo URLs de maneira tradicional.

O capítulo está estruturado da seguinte maneira: inicialmente, na seção 4.1 apresentamos o esquema de nomeação e identificação de recursos definido para o nosso projeto. Na seção 4.2, apresentamos a arquitetura proposta para que os recursos espelhados possam ser recuperados. Em seguida, na seção 4.3, apresentamos o serviço responsável pela conversão de uma URI em várias URLs, este serviço será utilizado na recuperação dos recursos. Finalmente, na seção 4.4, apresentamos a nossa proposta para o esquema de recuperação de recursos, que será realizado através da introdução de um mecanismo de balanceamento de carga.

4.1 O Esquema de Identificação de Recursos Espelhados

De acordo com as sintaxes definidas para o NID e o NSS, no capítulo anterior, o nosso esquema de nomeação baseado em URNs propõe um NID denominado de WMR, que significa *Web Mirrored Resources*. É através deste NID que as nossas URNs serão identificadas pelos usuários da Web e diferenciadas das demais.

O NID WMR determina um NSS formado por três campos: o primeiro campo é um nome de domínio Internet, que define o domínio do gerente responsável por um determinado grupo de cópias de recursos; o segundo campo é o nome de um grupo, que determina o grupo ao qual um determinado recurso pertence; e, por último, o campo recurso, que determina o recurso que se deseja recuperar.

O campo grupo foi criado com o intuito de aumentar a granularidade da informação espelhada, de forma a diminuir o número de pedidos realizados ao serviço de resolução de URNs, já o campo domínio facilita a geração de URNs únicas e globais.

A sintaxe do NSS é a seguinte:

```
NSS ::= <domínio> / <grupo> / <recurso>

<dominio> ::= <sub dominio> | " "

<sub dominio> ::= <label> | <sub dominio> . <label>

<label> ::= <letra> [ [ <string>] <letra-numero> ]
```

```
<string> ::= <letra-digito-hyp> | <letra-digito-hyp>
<string>

<letra-digito-hyp> ::= <letra-numero> | -

<letra-numero> ::= <letra> | <números>

<letra> ::= <maiusculos> | <minusculos>

<recurso> ::= 1*<caracteres>

<grupo> ::= 1*<caracteres>

<caracteres> ::= <maiusculos> | <minusculos> | <numeros>
| <especiais>

<maiusculos> ::= A | B | C | D | E | F | G | H | I | J |
K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y
| Z

<minusculos> ::= a | b | c | d | e | f | g | h | i | j |
k | l | m | n | o | p | q | r | s | t | u | v | w | x | y
| z

<numeros> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

<especiais> := _ | - | * | ( | ) * | ? | /
```

Como exemplo, poderíamos ter a seguinte URN:

```
urn:wmr:dsc.ufpb.br/alunos/tercio.html
```

que poderia ser utilizada para identificar e recuperar a página HTML que contém as informações sobre o aluno Tércio. As cópias desta página estariam armazenadas nos servidores que mantêm cópias dos recursos pertencentes ao grupo alunos, cujo domínio de gerência é o dsc.ufpb.br.

As URCs não são definidas em nosso projeto porque elas não pertencem ao seu escopo. Como descrito anteriormente, as URCs são conjuntos de meta-informações sobre um recurso identificado por uma URN e podem ser mais úteis para fins de manutenção e gerenciamento dos recursos, como tratado em [COS98].

A definição de nossa URN procurou atender o máximo dos requisitos definidos na seção 3.1. Por se tratar de uma tecnologia recente e em desenvolvimento, na qual não se definiram todos os padrões necessários, alguns destes requisitos podem não ser atendidos.

Os requisitos de unicidade global, escopo global e persistência devem ser garantidos pelo responsável pela nomeação e atribuição das URNs aos recursos. Por exemplo, a atribuição única de uma URN a um recurso é alcançada requerindo-se que cada autoridade responsável pela nomeação de recursos garanta esta unicidade. Estes requisitos são independentes da sintaxe atribuída às URNs e estão estreitamente ligados às funções de gerenciamento e manutenção de URNs. O campo `<dominio>`, que representa o nome de domínio do gerente de um determinado grupo de recursos, facilita bastante a geração de URNs únicas e globais, pois nomes de domínios se constituem em padrão bastante desenvolvido e amplamente utilizado, e são nomes únicos e globais na Internet.

O requisito de escalabilidade das URNs é conseguido com a definição do campo `<recurso>`, que nos permite atribuir nomes (URNs) a qualquer recurso existente atualmente na Web, sem qualquer restrição, ou atribuir nomes a qualquer outro recurso que venha a surgir futuramente.

O requisito de conversão de URNs é conseguido através dos serviços de resolução apresentados na seção anterior. A nossa definição de um serviço de resolução será apresentada na seção 3.4, que não apresentará a proposta de uma implementação de todos os serviços de resolução apresentados na seção anterior, pois estes serviços de resolução (N2C, N2R, N2Rs etc.) fogem ao escopo do nosso trabalho e alguns ainda não podem ser implementados. Por exemplo, o serviço N2C, que converte uma URN em uma URC, ainda não pode ser implementado porque não existe um consenso quanto ao conjunto e ao formato das meta-informações que descrevem um recurso.

Para que se obtenha o suporte ao legado existente é preciso que seja definido o esquema de registro de URNs, citado anteriormente, e que ainda não existe nenhum consenso quanto ao seu formato. Com este serviço implementado, quando o nosso esquema de resolução de URNs receber uma URN de outro tipo, ou seja, definida e implementada por outra organização, ele pode direcionar o cliente para o serviço de resolução adequado. A única organização responsável pelas URNs do tipo WMR é o LSD, pois, foi no âmbito do LSD que estas URNs foram idealizadas e projetadas. Com isto, a independência das nossas URNs está garantida.

Para que URNs possam ser transcritas e transportáveis em outros mecanismos, tal qual o *e-mail* (correio eletrônico), deve-se restringir o conjunto de caracteres utilizados em

URNs. Como ainda não existe um consenso quanto a isto, nós utilizamos o conjunto de caracteres definidos na sintaxe de URNs, apresentada em [RFC2141], que se constitui em um conjunto restrito de caracteres. Por se tratarem de uma cadeia de caracteres e terem uma sintaxe bem definida, sendo constituídas de campos que são facilmente compreendidos e analisados, as nossas URNs podem ser analisadas e comparadas por qualquer computador, sem haver a necessidade de nenhum recurso extra, entendida por qualquer ser humano, pois são *case insensitive*, curtas, e utilizam uma quantidade restrita de caracteres.

4.2 A Arquitetura para o Suporte ao Acesso a Recursos Espelhados

Uma vez definido que o esquema de nomeação e localização dos recursos que permitirá o espelhamento de recursos na Web será o esquema de URNs, definido na seção anterior, é preciso definir e prover na Web os componentes para que se consiga mapear os recursos, ou seja, converter uma URN em várias URLs através do serviço de resolução N2Ls. Este serviço possibilitará o acesso e a manutenção dos recursos espelhados.

Os servidores *proxies* são os candidatos mais indicados para cumprirem estas tarefas, pois são agentes intermediários e permitem que os clientes utilizem protocolos que eles não implementam e também pelo fato dos *proxies* serem servidores que estão mais próximos dos clientes. Além disso, tem o fator econômico, para implementarmos estes serviços nos servidores tradicionais teríamos que adequar o sistema de resolução a todas as plataformas (linguagem de programação e sistema operacional onde o servidor foi implementdo) dos servidores existentes. Mas com o servidor *proxy* só adaptamos um único proxy que atenderá pedidos em qualquer plataforma e repassará os pedidos para qualquer outro sevidor. Na Figura 4.1 está detalhada a arquitetura proposta.

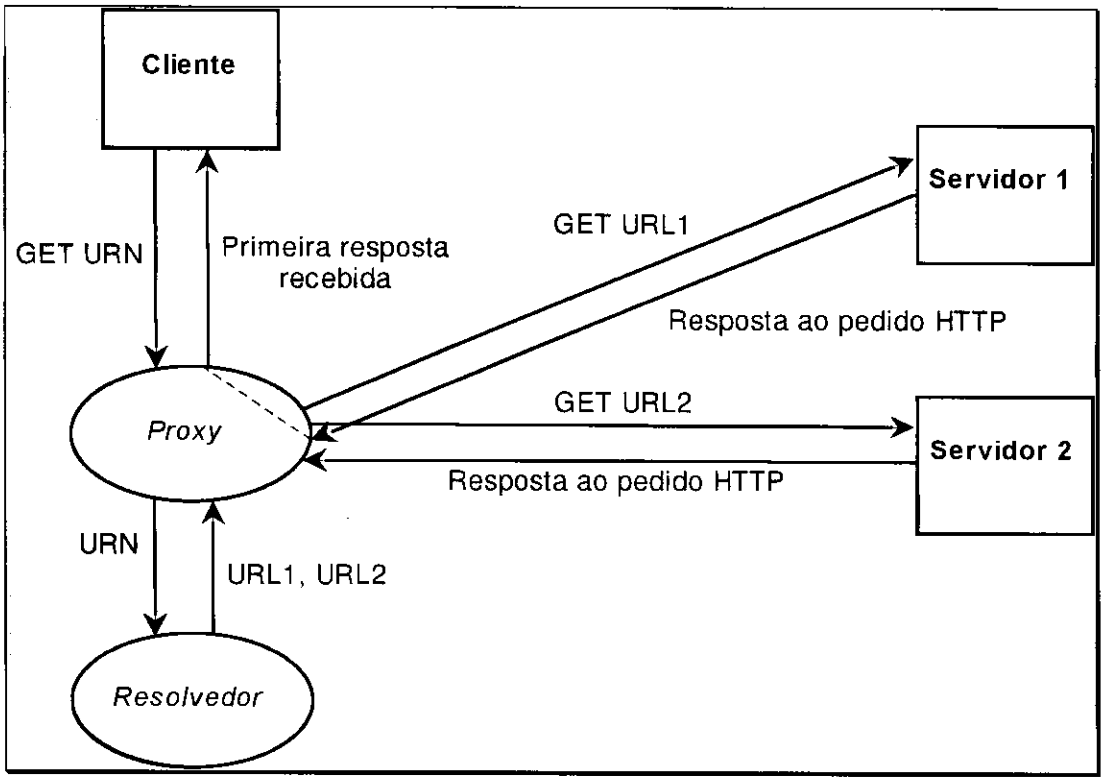

Figura 4.1 - Arquitetura para o Suporte a recursos espelhados

Como vemos na Figura 4.1, os *proxies* interceptam as mensagens dos clientes e consultam um resolvedor para descobrir as localidades onde o recurso identificado pela URN contida no pedido está armazenado. De posse do conjunto de URLs associadas ao recurso, o *proxy* tentará recuperar uma cópia do recurso utilizando o protocolo HTTP e, em seguida, enviá-lo de volta ao cliente que realizou o pedido.

Se o recurso identificado pela URN passada para o *proxy* estiver armazenado na memória *cache*, o *proxy* deve enviar este recurso para o cliente imediatamente, sem realizar qualquer consulta ao resolvedor, e, consequentemente nenhuma consulta aos servidores que mantêm cópias dos recursos.

Para evitar, ou pelo menos minimizar, os problemas de indisponibilidade dos recursos, o servidor *proxy* pode enviar os pedidos para todos os servidores que mantêm cópias do recurso identificado pela URN. As próximas seções apresentarão mais detalhes de como este processo será implementado e a descrição de um mecanismo de balanceamento de carga, que visa recuperar um recurso o mais rápido possível.

Com relação ao protocolo HTTP, utilizado pelo *proxy* no acesso e recuperação das cópias dos recursos, vimos que os pedidos HTTP podem conter qualquer um dos métodos

discutidos na seção 2.2. Os métodos GET, HEAD, PUT e DELETE possuem a propriedade de idempotência, ou seja, desprezando-se possíveis efeitos colaterais relacionados com erros e expiração da validade de recursos armazenados em *cache*, o efeito do processamento de uma requisição não muda, mesmo que essa requisição seja processada mais de uma vez [RFC2068]. Além disto, os métodos HEAD e GET são considerados seguros, pois suas ações não têm nenhum efeito sobre os servidores, a não ser a recuperação de recursos armazenados. O que não podemos garantir para os métodos PUT, POST e DELETE, que são considerados inseguros, pois suas ações causam modificações nos servidores, por exemplo, alterações e remoções de recursos armazenados nestes servidores.

De acordo com o que acabamos de discutir e sabendo-se que o escopo do nosso projeto abrange apenas o acesso e a recuperação de recursos, o servidor *proxy* implementa apenas os métodos considerados seguros, HEAD e GET. A justificativa para não implementarmos os outros métodos é a seguinte: os métodos OPTIONS e TRACE são dependentes da cadeia de comunicação entre o cliente e o servidor, que pode ser alterada dependendo do pedido realizado (mais detalhes serão apresentados na seção seguinte). Com relação aos métodos PUT, POST e DELETE, eles são considerados inseguros, podendo gerar problemas de inconsistências. Por exemplo, se o método DELETE não for aplicado a todas as réplicas de um mesmo recurso, as cópias não removidas ainda estarão disponíveis na Web.

O tratamento dos problemas de inconsistência gerados por estes métodos foge ao escopo do nosso trabalho. Um esquema para a gerência de recursos Web espelhados, que garante a manutenção da consistência destes recursos, independente das falhas que possam ocorrer com a utilização de métodos inseguros é proposto em [COS98].

4.3 O Serviço de Resolução de Identificadores

Como foi apresentado na seção anterior, o serviço de resolução de identificadores pode ser implementado em um servidor *proxy*. O servidor *proxy* deve ser capaz de aceitar e processar pedidos HTTP que contenham URNs como identificadores de recursos e continuar tratando os pedidos HTTP contendo URLs de maneira tradicional.

A implementação do serviço de resolução em um servidor *proxy* tem a vantagem de permitir os usuários de *browsers*, que não suportam o esquema de identificação de recursos baseados em URNs, utilizarem os serviços oferecidos.

O serviço de resolução proposto em nosso projeto é o N2Ls. Este serviço é especificado da seguinte maneira:

- N2Ls (URN para URLs)
  - *Função*: Converter cada URN em lista contendo uma ou mais URLs;
  - *Entrada*: Uma URN
  - *Saída*: Uma lista de zero ou mais URLs;
  - *Condições de Erro*: URN mal formada; URN não existente; URN existente, mas não existe saída para esta operação (não existe uma URL na qual a URN possa ser convertida); URN existiu, mas não se tem mais informações sobre ela; e acesso negado.

Uma característica de fundamental importância é a alta disponibilidade que este serviço deve apresentar para que sempre consiga atender os pedidos que lhe são enviados. A indisponibilidade do serviço de resolução de nomes impedirá que o recurso identificado pela URN seja recuperado, mesmo que a cadeia de comunicação entre o cliente e algum servidor que armazena o recurso não apresente falhas.

Em nosso projeto, o serviço de resolução de nomes implementado pelo resolvedor está dividido em duas etapas, uma local e uma global. A etapa local da resolução é executada pelo próprio servidor *proxy* e consiste em identificar e extrair da URN o domínio do gerente e o nome do grupo, para que seja possível verificar se o mapeamento pode ser realizado no local, utilizando as informações contidas na *cache*.

Se não for possível realizar o mapeamento com as informações contidas na *cache*, o resolvedor deve prosseguir com a segunda etapa, a global. Esta etapa consiste em utilizar os serviços do DNS (*Domain Name Server*) [AL97], que é o servidor de nomes bastante conhecido da Internet.

Para que seja possível utilizar os serviços do DNS, faz-se necessária a criação de um domínio, que será o responsável pelo serviço de resolução das URNs do tipo WMR,

permitindo que o resolvedor consulte a sua base de dados e obtenha os endereços das cópias do recurso identificado pela URN. Este domínio poderia chamar-se, por exemplo, de `wmr.urn.net`; `wmr.urn.br` ou `wmr.br` (para aqueles recursos espelhados em sub-domínios .br). Em nosso trabalho preferimos utilizar o `wmr.br`.

Para garantir a alta disponibilidade requerida para o serviço de resolução e para que os problemas de indisponibilidade sejam solucionados, ou pelo menos minimizados, o servidor de nomes escolhido deve também ser replicado. A arquitetura do DNS, por sua vez, já oferece o suporte adequado para a replicação de servidores de um determinado domínio.

As bases de dados do sistema DNS são constituídas de vários tipos de registros, os chamados RRs (*Resource Records*, Registros de Recursos), onde são armazenadas informações tais como: endereço IP de uma determinada máquina na rede e qual o sistema operacional executando nesta máquina. As chaves para a pesquisa de informações nos servidores de nomes são os nomes de domínio (nomes que identificam um computador na hierarquia da rede). Como por exemplo, `polvo.lsd.dsc.ufpb.br`, que identifica a máquina polvo no LSD. Mais detalhes podem ser obtidos em [AL97, RFC1035].

Em nosso projeto utilizamos os registros TXT da base de dados do servidores de nomes do DNS. Nestes registros são permitidos armazenar qualquer informação textual sobre uma determinada máquina na rede, cuja semântica depende do domínio onde os registros estão armazenados [RFC1035]. Portanto, nós armazenaremos os endereços dos servidores que mantêm cópias dos recursos de um determinado grupo nos registros TXT do servidor de nomes no domínio citado anteriormente, o `wmr.br`. Os endereços dos servidores estarão entre aspas e separados por vírgula, como vemos no exemplo da Figura 4.2, abaixo:

pesquisas.di.ufpe.br	TXT	"www.dsc.ufpb.br,www.di.ufpe.br"
proj-dsc.dsc.ufpb.br	TXT	"www.dsc.ufpb.br,www.unicamp.br"
cursos.dsc.ufpb.br	TXT	"www.dsc.ufpb.br,www.unicamp.br,www.di.ufpe.br"

Figura 4.2 - Extrato da base de dados do DNS

De uma forma geral, o nosso sistema de resolução funcionará da seguinte maneira: primeiro ele tentará resolver a URN localmente, através de informações contidas na cache e utilizando os campos gerente e grupo contidos na URN a ser resolvida. Caso a resolução local não seja possível, o serviço de resolução consultará os registros TXT do domínio

"wmr.br", como definido anteriormente, para poder localizar os servidores que mantêm cópias dos recursos replicados.

Para efetuar esta consulta ao servidor de nomes, o sistema de resolução concatena ao nome do grupo de réplicas o nome do domínio do gerente e, assim, obtém como resposta do servidor de nomes o endereço dos servidores que mantêm as cópias daquele recurso. De posse dos nomes dos servidores, o sistema de resolução pode formar as URLs que identificam as cópias dos recursos replicados da seguinte maneira:

`http://<endereço do servidor obtido na consulta>/<grupo da URN original>/<recurso da URN original>`.

Um exemplo de como funciona o nosso sistema de resolução de URNs pode ser visualizado na Figura 4.3. No exemplo da figura existem três servidores, em três domínios diferentes, nos quais estão armazenados os grupos de réplicas em suas bases de dados, um cliente e um servidor *proxy* (que suporta URNs) em um domínio diferente destes três. Também é mostrado na figura o servidor de nomes `wmr.br` e como as informações sobre a localização das réplicas são armazenadas nele.

Podemos perceber que a consulta inicia-se a partir do *Browser* (o cliente). Ele envia uma URN para o servidor *proxy*, suponha `urn:wmr:dsc.ufpb.br/cursos/Java.html`. Caso a informação não esteja na *cache* do *proxy*, o resolvedor irá consultar o servidor de nomes `wmr.br`, a fim de localizar os endereços dos servidores que mantêm cópias dos recursos requeridos e assim poder ter acesso a eles. A chave utilizada para realizar a consulta neste servidor seria `cursos.dsc.ufpb.br`, e a resposta a esta consulta seria "`www.unicamp.br,www.di.ufpe.br,www.dsc.ufpb.br`".

De posse dos endereços dos servidores que mantêm cópias do recurso desejado, o sistema de resolução formaria as URLs mostradas abaixo e enviaria o pedido aos respectivos servidores Web:

`http://www.dsc.ufpb.br/cursos/Java.html;`

`http://www.di.ufpe.br/cursos/Java.html;`

`http://www.unicamp.br/cursos/Java.html.`

Depois de todo o processo de resolução, o sistema de resolução envia para o cliente apenas uma das cópias do recurso, como resposta ao pedido efetuado anteriormente.

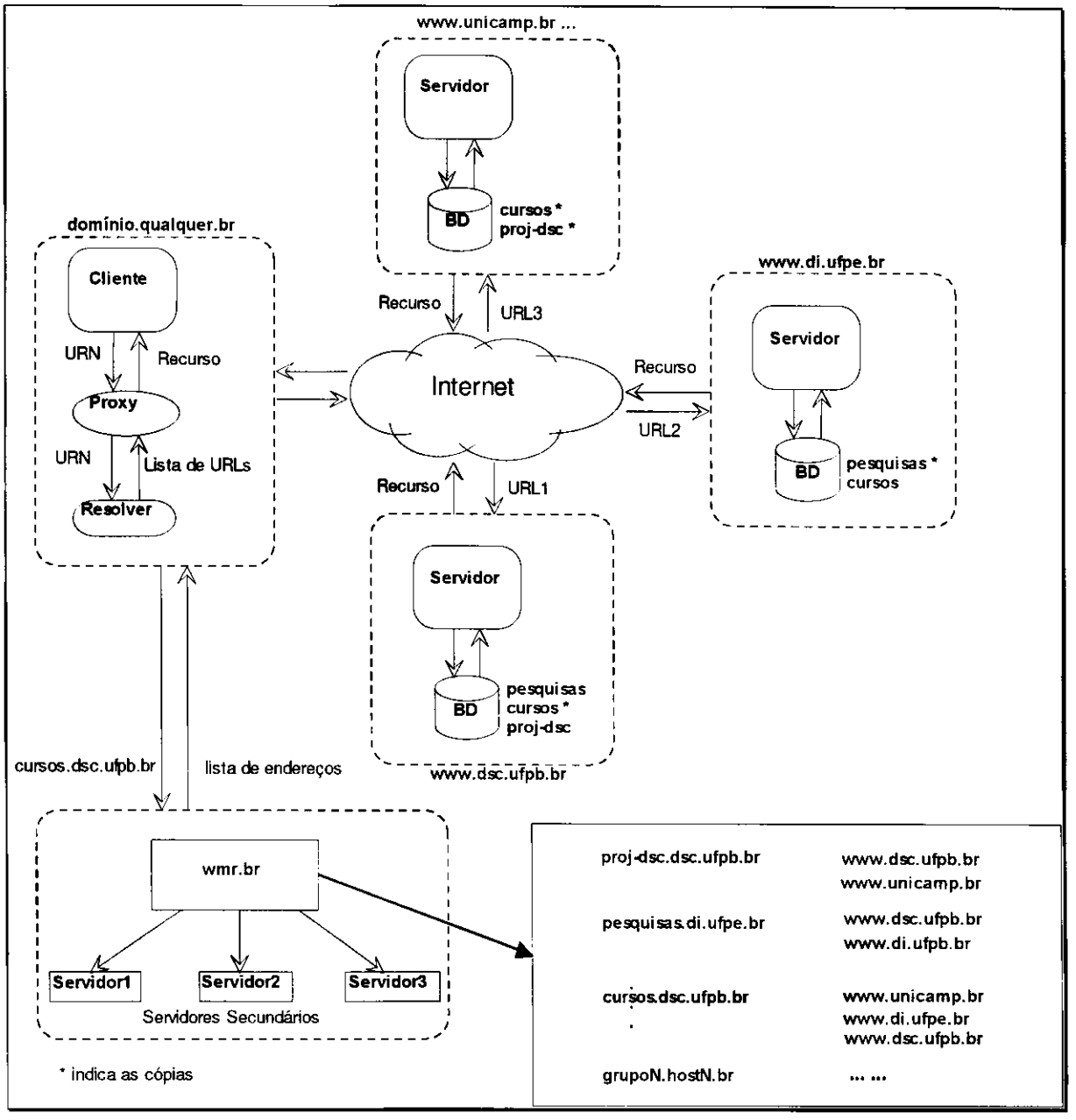

Figura 4.3 - Esquema de resolução de URNs

4.4 O Mecanismo de Balanceamento de Carga

Logo após o servidor *proxy* ter efetuado o mapeamento de uma URN em várias URLs, ele, de forma ideal deveria enviar o pedido HTTP para aquele servidor que tivesse o melhor tempo de resposta e estivesse operacional. Como não existe nenhum mecanismo disponível no servidor *proxy*, que possa prever quais os servidores que estão operacionais em um determinado momento, e qual a carga que está sendo submetida a estes servidores, esta operação não pode ser executada diretamente.

Uma alternativa para solucionar este problema seria o servidor *proxy* enviar os pedidos em paralelo para todos os servidores que armazenam cópias dos recurso, e transmitir para o cliente aquela resposta enviada pelo primeiro servidor que responder. Entretanto, esta abordagem pode acarretar problemas de tráfego, uma vez que os múltiplos pedidos enviados pelo *proxy* podem gerar sobrecarga na rede, e, conseqüentemente, ocasionar queda no desempenho do sistema, ou até mesmo um particionamento virtual do *proxy* com o restante da rede, devido ao congestionamento nos canais de comunicação.

De acordo com o exposto, a nossa proposta é implementar um esquema de balanceamento de carga dinâmico, para que tenhamos um ganho no tempo de resposta dos servidores sem haver quedas consideráveis no desempenho da rede. O esquema de balanceamento de carga funciona da seguinte forma:

- O processo é iniciado logo após o primeiro mapeamento de uma URN, em suas respectivas URLs, enviando-se pedidos para todos os servidores que mantêm cópias dos recursos;
- o *proxy*, então, mede o tempo de resposta de cada servidor que enviou uma resposta e armazena estes tempos. Aqueles servidores que não responderem no tempo máximo determinado terão este tempo computado como o seu tempo de resposta;
- apenas a resposta enviada pelo servidor que respondeu primeiro é repassada para o cliente. Esta resposta é enviada para o cliente assim que ela estiver disponível no servidor *proxy*.
- pedidos futuros para os recursos espelhados no mesmo domínio de gerência e grupo da URN mapeada serão repassados apenas para aquele servidor que respondeu mais rápido.

Mesmo quando são enviados pedidos para um único servidor, o *proxy* continua medindo o seu tempo de resposta e utilizando aquele servidor que teve o menor tempo de resposta na última requisição de pedido realizada. Se este servidor não envia a resposta em um determinado período de tempo, um novo pedido é enviado prontamente para o segundo servidor mais rápido, e assim por diante, até que se tenha tentado todos os servidores possíveis. Por outro lado, é retornada uma mensagem de erro caso nenhum servidor esteja disponível.

Este mecanismo está exemplificado na Figura 4.4.

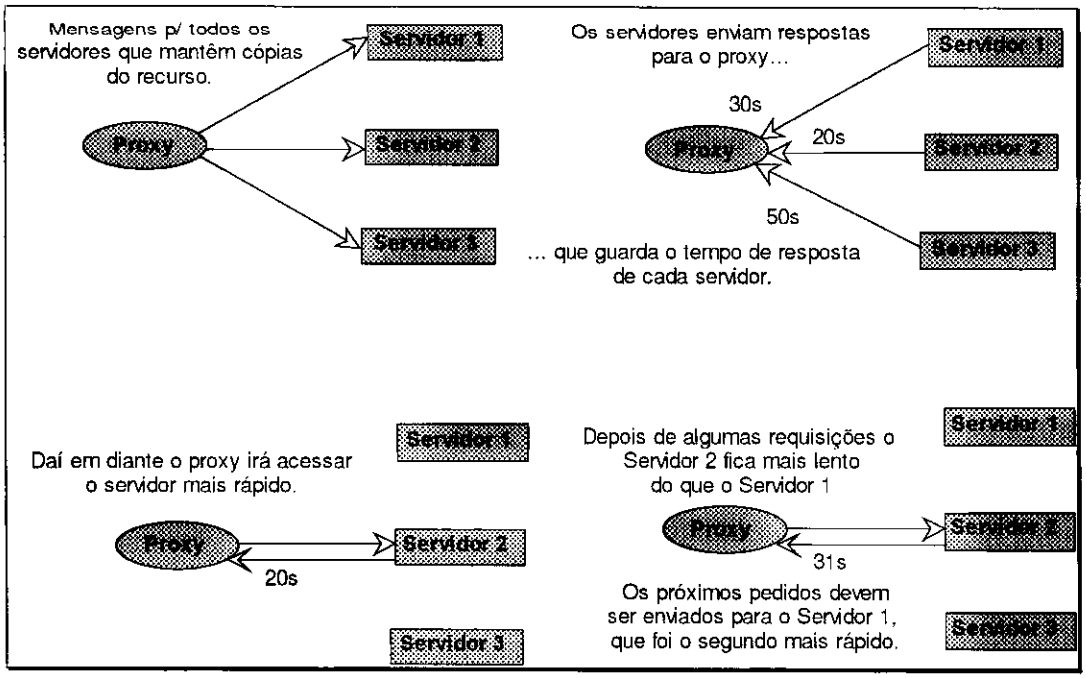

Figura 4.4 - Esquema de balanceamento de carga

Para que as informações sobre o balanceamento de carga estejam sempre atualizadas, é preciso definir um intervalo de tempo no qual o mecanismo de balanceamento de carga será ativado. É preciso definir este intervalo de tempo porque a carga nos servidores Web pode ser alterada ao longo do tempo e também pode ocorrer falhas na cadeia de comunicação entre o cliente e os servidores, o que pode tornar os dados sobre o balanceamento de carga desatualizados.

Definir este intervalo de tempo não é uma tarefa fácil, pois se o intervalo de tempo for muito pequeno, podemos gerar uma sobrecarga na rede devido a intensa troca de mensagens. Mas, por outro lado, não pode ser muito prolongado, pois as informações sobre o balanceamento de carga podem ficar muito tempo desatualizadas e assim prejudicar o desempenho do nosso sistema.

5. Aspectos de Implementação

Neste capítulo, mostraremos os detalhes de implementação da nossa proposta de um servidor *proxy* Web com suporte a recursos espelhados, apresentado no capítulo anterior. As considerações levantadas neste capítulo são direcionadas para o ambiente UNIX.

Antes de tratarmos da implementação propriamente dita, é necessário descrever o servidor que forneceu o suporte necessário para que a implementação pudesse ser realizada, o Servidor Apache, que será apresentado na seção 5.1. Logo em seguida, na seção 5.2, abordaremos os detalhes da implementação, apresentando os módulos e as funções necessárias para que os serviços propostos no capítulo anterior pudessem ser implementados.

5.1 O Servidor Apache

O Servidor Apache [LL97], ou somente Apache, é um Servidor Web que é executado sobre um sistema operacional multitarefa (para o nosso propósito, o UNIX, mas pode ser executado também no Windows95 e no OS/2). O Servidor Apache é *freeware*[8], e o seu código fonte pode ser obtido no seguinte endereço Web:: `http://www.apache.org`.

[8] Freeware significa que o software pode ser distribuído gratuitamente na Internet.

O Apache é desenvolvido por um grupo voluntário de programadores, que não são pagos para isso. Seu código fonte está sempre em modificação, pois são descobertos *bugs* e são adicionadas novas características, freqüentemente.

Devido a disponibilidade que o Apache apresenta e o fato de que um dos seus módulos é um servidor *proxy*, nós o escolhemos para ser a base de nossa implementação. A versão do Apache na qual realizamos as modificações foi a 3.1a.

A porção do código fonte do Apache, na qual nós realizamos modificações, foi justamente no módulo do servidor *proxy*, que é composto pelos seguintes arquivos:

- *mod_proxy.c*: é o arquivo principal, ele contém as funções responsáveis pelo recebimento e tratamento dos pedidos HTTP, direcionados para o servidor *proxy*;
- *proxy_http.c*: é o arquivo que contém as funções responsáveis pelo tratamento dos pedidos que utilizam o protocolo HTTP;
- *proxy_ftp.c*: é o arquivo que contém as funções responsáveis pelo tratamento dos pedidos que utilizam o FTP;
- *proxy_connect.c*: é o arquivo que contém as funções responsáveis pelo tratamento das conexões realizadas pelo *proxy*;
- *proxy_util.c*: é o arquivo que contém as rotinas auxiliares;
- *proxy_cache.c*: é o arquivo que contém as funções responsáveis pelo tratamento da memória *cache* do *proxy*.

5.2 A Implementação do Suporte a Recursos Web Espelhados

5.2.1 O Suporte ao Esquema de URNs

Vimos no capítulo anterior que para fornecer o acesso a recursos espelhados o servidor *proxy* do nosso projeto dever aceitar pedidos contendo URNs. Como o servidor *proxy* do Apache não apresenta tal característica, tivemos que modificá-lo para que ele aceitasse estes tipos de pedido.

Para realizar esta modificação foi necessário alterar o seu arquivo principal, o arquivo *mod_proxy.c*, no qual existe uma função chamada `proxy_handler`, que é a função responsável pela manipulação dos pedidos *proxy* e que realiza chamadas às funções que tratam estes pedidos adequadamente. O trecho de código mostrado na Figura 5.1, pertencente a função `proxy_handler`, que é o responsável pelo tratamento dos diversos protocolos suportados pela Web, no caso do servidor *proxy* do Apache, apenas são suportados os protocolos HTTP e FTP.

Observe que o suporte ao esquema de URNs foi adicionado ao servidor *proxy* através da inserção das linhas de código em destaque na Figura 5.1.

```
...
if (strcmp(scheme,"http"))
      return proxy_http_handler (r, cr, url, NULL, 0);
if (strcmp(scheme,"ftp"))
      return proxy_ftp_handler (r, cr, url);
if (strcmp(scheme,"urn")
      return proxy_urn_handler (r, cr, url, NULL, 0);
else return NOT_IMPLEMENTED;
...
```

Figura 5.1 - Modificação na função proxy_handler.

Antes de realizar a seleção do esquema, a função `proxy_handler` faz uma verificação na memória *cache* para certificar-se de que o recurso requisitado no pedido não se encontra armazenado. Esta verificação é realizada através da função `proxy_cache_check`, já definida no Apache. Primeiro, esta função verifica se o recurso está armazenado na *cache*, se estiver, verifica-se se o tempo de armazenamento já expirou. Caso isto ocorra, a função `proxy_cache_check` deve retornar uma indicação informando que o processamento do pedido deve prosseguir, caso contrário, a função deve retornar o recurso para o cliente que enviou o pedido. Caso a função `proxy_cache_check` mostre que o recurso não está armazenado na *cache*, ela também deve retornar uma indicação informando que o pedido deve continuar a ser processado.

O esquema de URNs do nosso projeto se adequou ao sistema de *cache* fornecido pelo servidor *proxy* do Apache, pois o esquema de *cache*, lida apenas com cadeia de caracteres, não importando se são URLs ou URNs.

A função `proxy_urn_handler` e todas as outras funções de tratamento de URNs são mantidas em um novo arquivo, adicionado ao módulo *proxy* do servidor Apache, chamado de *proxy_urn.c*.

5.2.1.1 A Função *proxy_urn_handler*

A função `proxy_urn_handler` é a função principal de nossa implementação, ela é a responsável pelo tratamento do esquema de URNs, realizando as chamadas às funções que possibilitam a resolução de URNs, o acesso aos recursos espelhados e a realização do balanceamento de carga.

A primeira tarefa que a função `proxy_urn_handler` deve realizar é a analise da URN passada como parâmetro. Esta tarefa é cumprida com a chamada à função `Analisa_URN`, que é a função responsável principalmente pela análise sintática das URNs que são enviadas ao Servidor *Proxy*.

Depois de analisar a URN, a função deve verificar se ela já foi resolvida anteriormente. Esta verificação é realizada através de uma consulta a uma tabela, que armazena as informações sobre as réplicas de uma determinada URN e os tempos de respostas dos servidores que as mantêm, baseando-se no grupo e no gerente dos recursos.

A verificação é feita varrendo-se a tabela e comparando-se o grupo e o gerente da URN corrente com os grupos e os gerentes armazenados nos registros da tabela. A tabela é varrida até o seu final ou até que seja encontrado um registro no qual o grupo e o gerente armazenados sejam igual ao da URN corrente, ou seja, aquela que foi verificada pela função `Analisa_URN`.

Se for constatado que a URN não foi resolvida anteriormente, é preciso que a função `proxy_urn_handler` realize a resolução da URN, através da chamada às funções `Localiza_Replicas`, para que possa identificar os servidores que mantêm armazenadas as réplicas dos recursos identificados pela URN e `Forma_URLs`, para construir as URLs que identificarão os recursos replicados. Em seguida, a função deve realizar a chamada às funções `Faz_Balanceamento`, a fim de computar e armazenar os tempos de respostas destes servidores, e `Guarda_Info`, para armazenar as informações de balanceamento de carga do grupo e gerente do recurso identificado pela URN. Estas informações de balanceamento de carga serão utilizadas em uma posterior resolução de URNs.

Uma vez identificado o registro e constatado que a URN foi resolvida anteriormente, é preciso verificar se o tempo decorrido desde a realização do último balanceamento de carga não expirou. Se este tempo já foi ultrapassado, será necessário realizar um novo balanceamento de carga e efetuar a atualização do registro.

Para que seja realizada uma nova análise da carga nos servidores, a função `proxy_urn_handler` deve chamar a função `Localiza_Replicas` novamente. Esta função deve ser chamada novamente porque pode ter ocorrido alguma modificação na base de dados do servidor de nomes, durante o intervalo de tempo ocorrido desde a última análise, ou a ferramenta que realiza a gerência dos recursos pode ter efetuado alguma modificação nos endereços das réplicas dos recursos. A função `proxy_urn_handler` também deve chamar a função `Faz_Balanceamento`, para que seja possível atualizar o tempo de resposta de cada réplica.

Se o tempo decorrido desde o último balanceamento de carga não foi ultrapassado, o passo seguinte será escolher dentre os servidores armazenados no arquivo aquele que possui o menor tempo de resposta. Identificado este servidor, a função deve montar a URL que identifica o recurso e então enviar o pedido para este servidor. A URL é construída concatenando-se o recurso da URN corrente com o grupo e o gerente obtidos do registro da tabela.

O tempo de resposta deste servidor é computado e atualizado na tabela que mantém as informações sobre o balanceamento de carga. Se houver algum problema com este servidor a função deve escolher o segundo servidor mais rápido, e assim por diante, até não existirem mais réplicas para serem consultadas; neste caso, é retornanda uma mensagem de erro informando que o recurso não está disponível.

5.2.1.2 A Função Analisa_URN

Como mencionado anteriormente, a função `Analisa_URN` é responsável pela análise sintática de uma URN, ela também é responsável por identificar e armazenar em uma estrutura o grupo, o gerente e o recurso que compõem a URN.

A função `Analisa_URN` elimina o prefixo "urn:" da URN passada como parâmetro, em seguida, a função extrai o NID da URN e realiza uma comparação para verificar se o NID da URN é realmente o que identifica o nosso esquema de URNs. Se o NID não for válida, a

função deve retornar uma mensagem de erro informando que este tipo de URN não é suportada pelo servidor *proxy*, caso contrário, a função deve prosseguir o processamento.

De acordo com a sintaxe das URNs, definida na seção 3.2, a função deve procurar a primeira ocorrência de uma barra ("/"), para que seja possível identificar o gerente que mantém as réplicas do recurso. Logo depois, localizar a segunda barra, para que seja possível identificar o grupo ao qual o recurso pertence, e, por último, o restante da URN será o recurso que o usuário deseja recuperar. Essa função deve retornar uma mensagem informando que a análise da URN foi realizada com sucesso ou uma mensagem de erro, se algum destes componentes não existir.

Os dados coletados sobre a URN, tais como gerente, grupo e recurso, são armazenados em uma estrutura chamada `urn_rec`, e que está definida como segue:

```
struct urn_rec {
    char *gerente;
    char *grupo;
    char **replicas; /* endereço das replicas */
    char **urls; /* URLs formadas */
    long *respostas; /*tempo de respostas do servidores */
    int  qtderep; /* quantidade de réplicas */
}
```

Figura 5.2 - Estrutura urn_rec

O restante dos campos desta estrutura serão preenchidos no decorrer da execução da função `proxy_urn_handler`, já descrita anteriormente.

5.2.2 O Serviço de Resolução

Como definido no capítulo anterior, a resolução de URNs é realizada em duas etapas. A implementação da primeira etapa já foi mostrada na seção anterior, onde foram identificados e extraídos os componentes da URN, para que fosse possível verificar se a URN já foi resolvida anteriormente. Caso a URN não tenha sido resolvida, a resolução de URNs prossegue em sua segunda etapa. Esta segunda etapa consiste basicamente em localizar o endereço das cópias do recurso identificado pela URN e formar as URLs que identificam estes recursos. As funções que implementam estas funcionalidades são apresentadas a seguir.

5.2.2.1 A Função Localiza_Replicas

A função `Localiza_Replicas` é a responsável pela consulta à base de dados dos servidores de nomes do domínio "wmr.br", para que seja possível identificar os endereços das réplicas de um recurso.

Antes de efetivamente realizar a consulta, `Localiza_Replicas` deve montar o índice de busca que será utilizado na pesquisa de nomes. O índice de busca é criado concatenando-se o nome do gerente com o nome do grupo da URN, como definido no capítulo anterior.

A consulta ao DNS é realizada através da utilização da função `Consulta_DNS`, que faz uso de uma biblioteca de funções fornecidas pelo resolvedor do DNS, que trabalham com pacotes de mensagens do DNS [AL97]. Os pacotes de mensagens do DNS são constituídos de cinco seções (em muitos casos algumas seções podem ser vazias): cabeçalho, consulta, resposta, autoridade e adicional.

A seção cabeçalho sempre está presente na mensagem e inclui campos que indicam quais as outras seções da mensagem que estão presentes, estes campos especificam se a mensagem é uma consulta ou uma resposta. A seção consulta contém campos que descrevem a consulta ao servidor de nomes, estes campos descrevem o tipo da consulta, a classe e o tipo do registro a ser consultado e ainda o domínio onde ocorrerá a consulta.

As outras seções possuem basicamente o mesmo formato, que consiste de uma lista de registros de recursos. A seção resposta possui registros de recursos que respondem às consultas realizadas. A seção autoridade contém registros de recursos que direcionam as mensagens para aqueles servidores que têm autoridade sobre algum determinado domínio, e, por fim, a seção adicional contém registros que mantêm informações adicionais relacionadas com as consultas.

A função `Consulta_DNS` recebe um nome de domínio como único argumento, que neste caso é o índice de busca, e localiza os servidores de nomes associados ao domínio, determinado em nosso projeto através da função `findNameServers`, e, em seguida, consulta na base de dados de um destes servidores os registros TXT, através de uma chamada à função `queryNameServers`. O resultado da consulta ao servidor de nomes é armazenando em um *buffer*. A função `Localiza_Replicas` analisa este *buffer*, e, a partir

dele, extrai o endereço das replicas, armazenando-o na estrutura `urn_rec`, os dados armazenados são armazenados no formato definido na seção 4.2.1.2.

A função `Localiza_Replicas` retorna uma mensagem de erro quando for identificado que o resultado fornecido pela função `Consulta_DNS` gere um *buffer* vazio, ou quando o nome de domínio passado como parâmetro não existir. Isto significa que não existem réplicas para aquela determinada URN, pois o registro TXT na base de dados ainda não foi gerado ou foi excluído.

5.2.2.2 A Função *findNameServers*

A função `findNameServers` é a função responsável por localizar todos os servidores de nomes associados a um determinado domínio, no nosso caso o "wmr.br", e armazenar os nomes destes servidores em uma lista.

Ela utiliza a função `res_query`, que pertence à biblioteca de funções do resolvedor do DNS, para localizar os registros de recurso do tipo NS (*Name Server*). Estes registros especificam os servidores de nomes para um determinado domínio. A função `res_query` é a responsável por construir o pacote da mensagem de consulta ao DNS, por enviar esta mensagem e também por checar se esta consulta foi respondida pelo servidor. São passados como parâmetro para a função `res_query` o domínio a ser consultado (o "wmr.br"), o tipo de registro a ser pesquisado (TXT), um *buffer* para armazenar a resposta enviada e uma variável numérica para armazenar o tamanho da resposta.

Depois de ter recebido a mensagem de resposta, que contém uma lista de registros de recursos NS, ou seja, os servidores de nomes encontrados; a função `findNameServers` deve ultrapassar a seção de cabeçalho da mensagem que está armazenada no *buffer* para poder analisar a seção de resposta. Feito isto, a seção de resposta é analisada pela função que armazena cada registro de recurso encontrado em uma lista de servidores de nomes. Esta lista será utilizada pela função `queryNameServers`, que será descrita a seguir.

5.2.2.3 A Função *queryNameServers*

A função `queryNameServers` é a função responsável por consultar os registros TXT do DNS, que armazenam os endereços das réplicas dos servidores que mantêm os recursos. Como mencionado anteriormente, estes registros TXT são indexados pelos nomes de domínios formados com o grupo e o gerente de uma URN. Este nome de domínio é recebido

como parâmetro pela função `queryNameServers`. A consulta é realizada utilizando a lista de servidores de nomes preenchida pela função `findNameServers`.

Para cada servidor contido nesta lista, a função `queryNameServers` primeiramente deve pegar o endereço IP de cada servidor, pois são armazenados apenas os nomes de domínio dos servidores. Este processo é realizado pela função `gethostbyname`, fornecida pela biblioteca do resolvedor, que recebe um nome de domínio como argumento e retorna o seu endereço IP como resposta.

De posse dos endereços IP dos servidores de nomes, a consulta é então realizada. Para isto, utilizamos as funções `res_mkquery` e `res_send`, também pertencentes à biblioteca do resolvedor.

A função `res_mkquery` é a responsável por criar as mensagens de consultas, preenchendo os campos do cabeçalho e os campos das outras seções que compõem a mensagem. A função tem como argumentos o nome de domínio do servidor de nomes a ser consultado, o tipo de registro a ser consultado, a operação a ser executada (neste caso uma consulta), o *buffer* onde será armazenada a mensagem de consulta e uma variável numérica que indica o tamanho deste *buffer*.

A função `res_send` é a responsável por enviar as mensagens criadas pela função `res_mkquery`. Ela possui como argumentos o *buffer*, que armazena a consulta gerado pela função `res_mkquery`, e um *buffer* onde será armazenada a resposta enviada pelo servidor de nomes, além de variáveis numéricas que armazenam o tamanho destes *buffers*. O retorno da função é um número inteiro indicando o tamanho da mensagem de resposta ou -1 indicando que algum erro ocorreu. Se a resposta relatar algum erro ocorrido na consulta, a função `queryNameServers` deve recomeçar o processo e realizar a consulta em outro servidor pertencente à lista.

Se nenhum erro for detectado, a função `queryNameServers` deve então partir para a análise da resposta de mensagem. A função `queryNameServers` deve ultrapassar a seção cabeçalho e a seção consulta, e chegar à seção resposta. A seção resposta deve conter apenas uma resposta; se ela possuir mais de uma resposta, deve ser relatado um erro e deve-se recomeçar o processo de consulta no próximo servidor de nomes armazenado na lista. Caso contrário, a função deve analisar a seção resposta e capturar o registro TXT armazenado nela.

5.2.2.4 A Função Forma_URLs

A função `Forma_URLs` é a função responsável por gerar as URLs que identificarão as réplicas do recurso identificado por uma URN. A geração destas URLs é realizada da seguinte forma: para cada endereço das réplicas encontrado pela função `Localiza_Replicas` cria-se uma URL vazia e adiciona-se o prefixo "http://", que identifica o esquema a ser utilizado; logo em seguida, concatena-se sucessivamente a esta URL, o endereço encontrado, uma barra, o grupo, outra barra e finalmente o recurso. As URLs formadas são armazenadas na estrutura `urn_rec`. Estas URLs serão utilizadas pela rotina de balanceamento de carga para enviar os pedidos concorrentemente.

5.2.3 O Mecanismo de Balanceamento de Carga

Uma vez localizado o endereço das réplicas e construídas as URLs que identificam os recursos espelhados, o servidor *proxy* deve enviar o pedido para o endereço do servidor que possui o menor tempo de resposta, como especificado no capítulo 4. Detalhes desta implementação serão apresentados nas seções que virão a seguir.

5.2.3.1 A Função Faz_Balanceamento

A função `Faz_Balanceamento` é a responsável por enviar pedidos para todas as réplicas do recurso identificado por uma URN e computar os respectivos tempos de resposta. Os dados necessários para que esta função possa funcionar corretamente são obtidos através da estrutura `urn_rec`, definida anteriormente, estes dados são as URLs, que identificam as cópias dos recursos.

Para cada réplica, a função `Faz_Balanceamento` deve verificar se o endereço contido em cada URL existe. Depois disto, ela deve transformá-los em endereços IP numéricos e, em seguida, deve alocar um *socket*[9] para cada URL e realizar a conexão com o servidor remoto. Para cada *socket* alocado é aberta uma nova conexão TCP, os *sockets* alocados são armazenados em uma variável do tipo `fd_set`, que define um conjunto de *sockets*.

[9] *Socket* é uma abstração para a comunicação em rede, assim como os arquivos, os sockets são identificados por um número inteiro, que é chamado de descritor de *socket*. Um *socket* pode ser utilizado para receber pedidos de conexões ou iniciar conexões.

Depois de alocado e armazenado todos os *sockets* e abertas todas as conexões necessárias, a função `Faz_Balanceamento` realiza uma chamada a função `Manipula_Pedidos_Concorrentes`, que realiza o tratamento das mensagens HTTP de maneira concorrente.

5.2.3.2 A Função Manipula_Pedidos_Concorrentes

A função `Manipula_Pedidos_Concorrentes`, é a função responsável pelo envio e recebimento dos pedidos HTTP para os servidores, definidos nas URLs, concorrentemente.

A função `Manipula_Pedidos_Concorrentes` é executada até que todos os servidores tenham respondido ao pedido enviado ou até que seja atingindo o prazo para que estes servidores enviem as respostas. No início desta função é inicializado o contador que irá computar o tempo de resposta de cada servidor. Este contador é inicializado quando é realizada uma chamada a função `mstime`, que retorna a quantidade de milisegundos desde primeiro de janeiro de 1970.

Depois de incializado o contador, a função `Manipula_Pedidos_ Concorrentes` verifica se o *socket* já está pronto para a leitura ou para a escrita. Se o *socket* estiver pronto para escrita, a função `Manipula_Pedidos_ Concorrentes` faz uma chamada a função `Writer`, que é a responsável por enviar os pedidos aos servidores. Se o *socket* estiver pronto para leitura, a função faz uma chamada função `Reader`, que é a responsável pelo recebimento das respostas dos servidores.

5.2.3.3 A Função *Writer*

A função Writer é a função responsável pelo envio dos pedidos aos servidores Web que mantêm as cópias dos recursos replicados. Ela é a responsável por construir e enviar os pedidos HTTP.

No início da função `Writer`, antes do envio de um pedido HTTP, é ativado um alarme, que é responsável por verificar o tempo que esta função gasta para realizar o seu processamento. O alarme é ativado através da chamada a função `hard_timeout`, definida pelo servidor Apache. Quando o prazo deste alarme é atingido, o pedido corrente é abortado.

Ao final da função `Writer`, o *socket* alocado para aquela conexão é desabilitado para a escrita e é realizada uma chamada a função `kill_timeout`, também definida pelo servidor Apache. Ela é a responsável por desabilitar um alarme quando este não for mais necessário.

5.2.3.4 A Função *Reader*

A função `Reader` é a responsável pelo recebimento das respostas dos servidores Web que mantêm cópias dos recursos espelhados.

No início da função `Reader`, antes do recebimento da resposta do pedido HTTP, um alarme é ativado através da função `hard_timeout`, já mencionada anteriormente. Quando o prazo deste alarme é atingido o processamento da resposta corrente é abortada.

Quando a função `Reader` recebe a resposta do servidor, ela a armazena em um *buffer* e computa o tempo decorrido desde o envio do pedido até este momento, realizando mais uma vez a chamada a função `mstime`. O tempo é computado em milisegundos e é adquirido através da diferença entre a primeira chamada e a chamada efetuada no momento do recebimento do pedido.

Logo em seguida, a função verifica se esta resposta é a primeira resposta recebida, se for, a função deve retornar esta resposta para o cliente, caso contrário é computado o tempo de resposta do servidor e o processo continua normalmente. Finalmente, a função deve realizar uma verificação na memória *cache* para saber se a URN já está armazenada. Se estiver, ela faz a verificação do tempo que esta URN ficou armazenada na *cache*, se este tempo expirou, a URN deve ser atualizada na *cache*, caso contrário, o processamento deve prosseguir. Caso a URN ainda não esteja armazenada na *cache*, a função `Reader` deve armazená-la.

Ao final da função `Reader`, o *socket* é desabilitado. O alarme também deve ser desabilitado, isto é realizado através de uma chamada a função `kill_timeout`.

5.2.3.5 A Função Guarda_Info

A função `Guarda_Info` é a responsável pelo armazenamento das informações sobre o balanceamento de carga. Esta função armazena um registro no arquivo de carga, o "carga.info", contendo o nome do gerente e do grupo de um recurso, os servidores que mantém as réplicas do recurso e seus respectivos tempos de resposta. Estes dados são obtidos da tabela que armazena as informações sobre o balanceamento de carga.

6. Conclusões e Trabalhos Futuros

O rápido crescimento que a Web vem apresentando nestes últimos anos tem atraído o interesse de milhares de pessoas em todo o mundo, desde usuários domésticos até grandes organizações comerciais.

A Web apresenta-se como um novo e inovador meio de comunicação e de oferta de serviços. A facilidade na publicação de informações em qualquer formato, e em uma diversidade de plataformas de computadores distribuídos por todo o mundo, proporciona a realização de grandes negócios.

Infelizmente, alguns países apresentam deficiências na infra-estrutura que dá suporte à Web. As péssimas condições da infra-estrutura de comunicação e de provimento e acesso à Internet têm gerado grande constrangimento para os usuários, pois os recursos publicados na Web tornam-se cada vez mais indisponíveis.

Neste trabalho, foi apresentada uma alternativa para minimizar este problema de indisponibilidade de recursos que a Web apresenta. Uma maior disponibilidade dos recursos Web foi obtida através do espelhamento de recursos, onde são armazenadas várias cópias dos recursos em localizações distintas. Assim, através desta técnica, é possível ter acesso às cópias dos recursos por rotas alternativas.

Além disso, através da utilização de um servidor *proxy*, que foi introduzido na infra-estrutura da Web, implementamos um esquema alternativo de identificação de recursos, que nos possibilitou tolerar falhas nos servidores e na cadeia de comunicação da Web, de maneira transparente.

Também, o espelhamento de recursos pode ocasionar alguns problemas. Não é uma tarefa fácil gerenciar múltiplas cópias de um recurso, espalhadas por toda a Web. Problemas relacionados com a consistência mútua destas cópias, com a integridade referencial e com a sobrecarga do sistema, podem ser graves. Se algumas das cópias de um recurso forem atualizadas e outras não (uma página que informa o adiamanto de um concurso, por exemplo), os diversos usuários que tem acesso a este recurso terão informações inconsistentes, o que pode trazer prejuízos para ambas as partes, os usuários e a pessoa (ou organização) responsável pela publicação das informações na Web.

Por outro lado, implementamos um esquema de balanceamento de carga que nos forneceu um aumento no desempenho da recuperação dos recursos espelhados, o que nos possibilitou alcançar os objetivos estabelecidos no início deste trabalho. Mas, para que os problemas relacionados com o gerenciamento dos recursos sejam solucionados, uma ferramenta para a gerência de servidores Web com suporte a recursos espelhados é proposta em [COS98].

Por último, é válido salientar que a nossa ferramenta ainda possui algumas limitações. Uma delas é que o servidor *proxy* se constitui em um ponto de falha, pois, se ele parar de funcionar não será mais possível realizar o mapeamento de URNs em URLs, consequentemente não será possível utilizar o esquema de URNs e acessar os recursos espelhados. Uma outra limitação é com relação a escolha do domínio para a implementação do resolvedor global. Da forma como está sendo realizada somente podemos escolher um único servidor de nomes (que pode ser replicado ou não, em seus servidores secundários) que implemente o resolvedor global, e apesar de se constituir em um serviço altamente disponível, este servidor pode se tornar um gargalo. Na próxima seção apresentaremos uma sugestão de como este problema pode ser resolvido.

6.1 Trabalhos Futuros

Como um dos objetivos de nosso trabalho foi obter melhor desempenho na recuperação de recursos Web, uma proposta para trabalhos futuros é realizar uma análise detalhada de quão bom é o nosso esquema de balanceamento de carga.

Nossa sugestão é que seja realizada uma bateria de testes, que terão a finalidade de avaliar a eficiência do mecanismo. Então, seriam apontados os pontos falhos do mecanismo, seriam sugeridas soluções para a correção destas falhas e, por fim, seriam realizadas as modificações necessárias através da implementação.

Ao final do desenvolvimento deste trabalho já detectamos alguns pontos a serem observados e algumas melhorias que podem ser realizadas no mecanismo de balanceamento de carga para que ele funcione de forma mais eficiente. As observações e sugestões para a realização destas melhorias são discutidas a seguir.

Primeiro, o processo de atualização dos tempos de resposta dos servidores que mantêm cópias de recursos espelhados é realizado quando for detectado que o prazo para sua atualização expirou. A expiração destes prazos é detectada quando um usuário envia um pedido HTTP para o servidor e for constatado que o intervalo de tempo desde a última atualização já foi ultrapassado. Esta atualização é realizada utilizando os pedidos HTTP que contém o método GET.

A nossa sugestão é que este processo de atualização seja realizado de forma contínua, e sem esperar por qualquer ação do usuário. O ideal seria ter uma forma de requisição padrão que fosse automaticamente enviada para os servidores em intervalos de tempo bem definidos. Esta requisição seria um pedido HTTP que utilizasse o método HEAD, pois não há necessidade de recuperar recurso algum quando estamos atualizando os tempos de resposta dos servidores.

Segundo, a forma como os tempos de resposta de cada servidor estão sendo armazenados pode ser melhorada. Atualmente, para cada par grupo/gerente (campos que compõem uma URN) são armazenados os tempos de resposta de cada servidor que mantém cópias dos recursos espelhados para este par grupo/gerente. Mas, se o nosso sistema estiver espelhando muitos grupos e estes grupos estiverem espelhados em diversos servidores comuns, estaremos: a) desperdiçando espaço de armazenamento, pois, estarão armazenados os tempos de resposta de um único servidor em vários registros diferentes, um em cada

registro que mantém informações sobre um grupo espelhado; e b) diminuindo o desempenho do sistema, pois, o processo de pesquisa torna-se mais lento devido ao grande número de registros, um para cada grupo existente.

Uma sugestão para minimizar este problema é criar uma lista dinâmica que armazena os tempos de resposta de cada servidor uma única vez e, manter uma outra lista com os nomes dos grupos, esta lista mantém apontadores para aqueles servidores que espelham recursos pertencentes a estes grupos. Como exemplificado na Figura 6.1 abaixo:

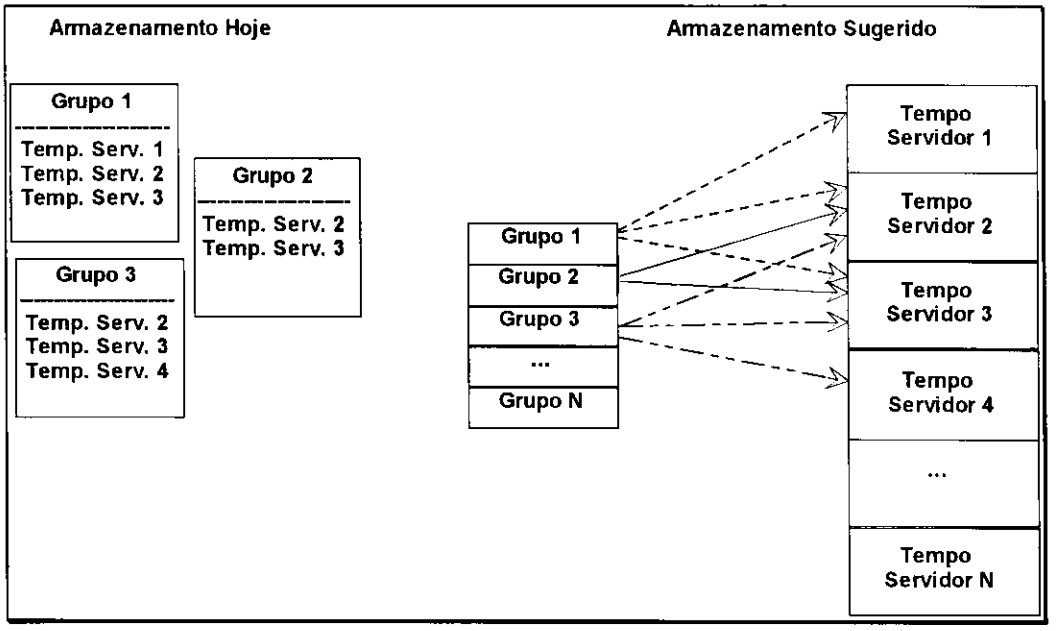

Figura 6.1 - Armazenamento dos tempos de resposta

Observamos na Figura 6.1 que os servidores 2 e 3 espelham os grupos 1, 2 e 3 e, por isso, os seus tempos de respostas são armazenados nos registros que mantém informações sobre os grupos, neste caso, três vezes para a forma de armazenamento atual. Na forma de armazenamento sugerido, o tempo de resposta destes servidores seriam armazenados apenas uma única vez e a atualização dos seus dados serviria para todos os grupos que estão espelhados nestes servidores.

Outra proposta para trabalhos futuros é permitir que o resolvedor global possa ser implementado em mais de um domínio, de forma a permitir uma maior escalabilidade do nosso esquema de espelhamento de recursos. Atualmente em nosso esquema escolhemos um único domínio para implementar o resolvedor global para as URNs do tipo WMR, o que limita o nosso esquema. Por exemplo, se estivermos espelhando recursos apenas na Paraíba, obviamente escolhemos um domínio aqui na Paraíba (poderíamos ter mwr.ufpb.br). Mas se a

Unicamp (Universidade de Campinas) também quisesse espelhar recursos em nosso sistema teríamos três alternativas: a) continuar com o resolvedor global na Paraíba - o que dificulta o acesso para os usuários da Unicamp; b) trasferir o resolvedor para São Paulo - o que dificulta o acesso para os usuários da Paraíba; ou c) definir um reseolvedor global que atenda as duas regiões.

A nossa sugestão é que os recursos sejam espelhados em um conjunto bem definido de redes, que formariam fronteiras e que limitariam o escopo de um domínio WMR. Assim, cada domínio WMR deveria implementar o seu próprio resolvedor global. Por exemplo, poderíamos ter o domínio WMR.PB sendo utilizado para implementar o resolvedor global para o espelhamento dos recursos localizados em computadores na rede da Paraíba. Também poderíamos ter domínio WMR.BR sendo utilizado para implementar o resolvedor global para o espelhamento dos recursos localizados em computadores nas redes do Brasil.

O domínio do resolverdor global seria definido e identificado pelo NID da URN. Por exemplo: `urn:wmr.pb:dsc.ufpb.br/alunos/tercio.html`. Para esta URN seria consultado o resolvedor global no domínio WMR.PB para localizar os servidores Web na Paraíba que espelham o grupo alunos.

6.2 Considerações Finais

Ao longo do desenvolvimento deste trabalho pudemos realmente perceber e constatar os problemas citados e discutidos na introdução. Por ser uma abordagem ainda considerada nova, não existe literatura sobre o esquema de nomeação baseado em URNs e seus mecanismos de resolução. Por isso, a pesquisa foi quase que em sua totalidade realizada através da utilização deste imenso provedor de informações, a World Wide Web.

Com a utilização da Web em nossas pesquisas, nos deparamos constantemente com o conhecido problema de quebra de ligação. Quando achávamos uma referência eletrônica (referência feita a uma publicação na Web), em um artigo, em um livro ou na própria Web, o nosso pensamento era que esta publicação estivesse disponível na Web, o que nem sempre acontecia, às vezes ela estava apenas inacessível, mas às vezes ela tinha sido removida definitivamente.

Um outro problema constatado, foi a péssima infra-estrutura que a Web brasileira apresenta. As baixas larguras de banda apresentadas nas linhas que interligam a Web

brasileira, comparadas com outros países como os Estados Unidos, e os poucos redundantes *backbones* brasileiros se constituem em um bom exemplo.

Não pudemos contabilizar a quantidade de vezes que ficamos isolados do mundo, sem podermos realizar as nossas pesquisas. Tudo isto só por causa de uma falha na comunicação de Campina Grande com Recife, que se constitui no único *link* que conecta a Paraíba à grande rede mundial.

Por todos estes problemas enfrentados pelos usuários da Web, é que temos a convicção de que esta ferramenta será de grande utilidade. A possibilidade de se ter mais de uma cópia de um recurso e o desenvolvimento de um serviço que sempre esteja disponível, e que recupere estas cópias dos recursos de maneira rápida e eficiente, proporciona aos usuários uma maior satisfação ao utilizarem a World Wide Web.

Referências Bibliográficas

[AL94] ALTIS, Kevin. LUOTONEN, Ari. "*World-Wide Web Proxies*". Abril de 1994. URL: http://www.lsu.edu/internet/guides/www-docs/WWW/proxies.

[AL97] ALBITZ, Paul e LIU, Cricket. "***DNS and BIND***". O'Reilly & Associates, Inc - 2ª Edição. USA, 1997.

[BCLNS94] BERNERS-LEE, Tim. CAILIAU, Robert. LUOTONEN, Ari. NIELSEN, Henrik Frystyk e SECRET, Arthur. "*The World-Wide Web*". Communications of the ACM, Agosto de 1994 - Vol.37, Nº 8

[CM93] CTOSian Magazine - The Worldwide Magazine for User and Resellers. "*What is World-Wide Web?*". URL: http://www.eps.net/CTOSian/mar2.htm.

[COS98] COSTA, Raquel Meneses da. "*Projeto e Implementação de uma Ferramenta para a Gerência de Servidores Web com Suporte a Recursos Espelhados*". Dissertação de Mestrado/Departamento de Sistemas e Computação, Universidade Federal da Paraíba - Campus II. Campina Grande, Março de 1998.

[CZ95] CHAPMAN, D. Brant e ZWICKY, Elizabeth D. "***Building Internet Firewalls***". O'Reilly & Associates, Inc. Setembro de 1995.

[DAN96] DANIEL, Ron. "*Uniform Resource Names*". INFINIA - Information Millenia, LANL - Los Alamos National Laboratory. Agosto de 1996. URL: http://www.acl.lanl.gov/URN/urn-home.html.

[DM97] DANIEL, Ron e MEALLING, Michael. "Resolution of Uniform Resource Identifiers using the Domain Name System". Request for Comments 2168, Los Alamos National Labotratory/Network Solutions, Inc - Junho de 1997.

[HAM95] HAMILTON, Martin. "*Uniform Resource Identifiers & The Simple Discovery Protocol*". Loughborough University of Technology, junho de 1995. URL: http://www.roads.lut.ac.uk/Reports/uris.

[HUGES93] HUGES, Kevin. "*Entering the World-Wide Web: A Guide to Cyberspace*". Honolulu Community College, Outubro de 1993. URLs: http://spooky.pdb.sni.de/guide/www.guide.html e http://www.eit.com/goodies/www.guide.

[IAN96] IANNELLA, Renato. "*URCs - Silverlining for URNs*". Research Data Network Cooperative Research Centre - RDN CRC, Bisbane, Australia - Maio de 1996. URL: http://www5conf.inria.fr/fich_html/slides/panels/panel5/iannella/all.html.

[ICL96] INGHAM, David. CAUGHEY, Steve. LITTLE, Mark. "*Fixing the Broken-Link Problem: The W3Objects Approach*". Universidade de Newcastle, Maio de 1996. URL: http://arjuna.ncl.ac.uk:80/W3Objects/papers/www5/Overview.html

[ILCS95] INGHAM, David. LITTLE, Mark. CAUGHEY, Steve. SHRIVASTAVA, Santosh. *W3Objects: "Bringing Object-Oriented technology to the Web"*. Universidade de Newcastle, 1995. URL: http://arjuna.ncl.ac.uk:80/W3Objects/papers/www4/ Overview.html

[ISL96] IANNELLA, Renato. SUE, Hoylen. LEONG, Danny. "*BURNS: Basic URN Service Resolution for the Internet*". Research Data Network Cooperative Research Centre - RDN CRC, University of Queensland - Australia, 1996. URL: http://www.dstc.edu.au/RDU/reports/Apweb96.

[JAL94] JALOTE, Pankaj. "***Fault Tolerance in Distributed Systems***". Prentice Hall, 1994.

[LL97] LAURIE, Ben e LAURIE, Peter. "***Apache - The Definitive Guide***". O'Reilly & Associates, Inc. 1ª Edição, USA - 1997.

[RFC1035] MOCKAPETRIS, P. "*Domain Names - Implementation and Specification*". RFC 1035, ISI, Novembro de 1987.

[RFC1630] BERNERS-LEE, Tim. "*Universal Resource Identifiers in WWW*". RFC1630, Junho de 1994.

[RFC1737] MASINTER, Larry. SOLLINS, Karen. "*Functional Requirements for Uniform Resource Names*". RFC 1737, MIT/LCS, Xerox Corporation, Dezembro de 1994.

[RFC1738] BERNERS-LEE, Tim, MASINTER, Larry e MCCAHILL, Mark. "*Uniform Resource Locators*". RFC 1738, CERN, Xerox Corporation, University of Minnesota, Dezembro de 1994.

[RFC1866] BERNERS-LEE, Tim. E CONNOLLY, D. W. "*Hypertext Markup Language - 2.0*". RFC 1866, Novembro de 1995.

[RFC1945] BERNERS-LEE, Tim. FIELDING, Roy T. NIELSEN, HenrikFrystyk. "*Hypertext Transfer Protocol - Hypertext Transfer Protocol - HTTP 1.0* ". RFC 1945, MIT/LCS, UC Irvine, Maio de 1996.

[RFC2068] FIELDING, Roy T., GETTYS, Jim, MOGUL, Jeffrey C., NIELSEN, Henrik Frystyk, BERNERS-LEE, Tim. "*Hypertext Transfer Protocol - HTTP 1.1*". RFC 2068, DEC - Digital Equipament Corporation, MIT/LCS, janeiro de 1997.

[RFC2141] MOATS, Ryan. "*URN Syntax*". RFC 2141, AT&T, Maio de 1997.

[SBE96] SBEASLEY, Sarah. *What is World-Wide Web ?*. Seatlle Community College District, maio de 1996. URL: http://www.sccd.ctc.edu/~sbeasley/mais/whtiswww.html.

[SLO96] SLOTTOW, Teddy. "*Design of a Global Namespace*". ANA - Advanced Network Architecture.20 de agosto de 1996.

[W3C96] World-Wide Web Consortium - W3C. "*About The World-Wide Web Consortium*". URL: http://www.w3c.org.

[WM96] Webmaster Magazine. "*An Overview of the World-Wide Web*". CIO Communications, Inc. 1996. URL: http://www.web-master.com/WebMaster/sem2_home .html.