UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CRISTIANO CACHAPUZ E LIMA

ORPIS: um Modelo de Consistência de Conteúdo Replicado em Servidores *Web* Distribuídos

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, maio de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lima, Cristiano Cachapuz e

ORPIS: um Modelo de Consistência de Conteúdo Replicado em Servidores *Web* Distribuídos / Cristiano Cachapuz e Lima. – Porto Alegre: Programa de Pós-Graduação em Computação, 2003.

76 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Cláudio Fernando Resin Geyer.

1. Consistência. 2. Réplica. 3. Servidor *web* distribuído. I. Geyer, Cláudio Fernando Resin. II. Título.

"Deliver me, out of my sadness
Deliver me, from all of the madness
Deliver me, courage to guide me
Deliver me, strength from inside me
All of my life I was in hiding
(. . .)
Deliver me, the cross that I´m bearing
Deliver me,
Deliver me,
Oh deliver me"
— The Beloved

AGRADECIMENTOS

À minha querida esposa, Súsi, que sempre me apoiou e me compreendeu durante essa longa jornada.

Aos meus pais, que me incentivaram a sempre buscar algo mais em relação ao conhecimento e à formação.

À minha Dadada, Marísia Souto Cachapuz, que tanto torceu por mim.

À Universidade da Região da Campanha, instituição onde comecei a trabalhar com Computação e que sempre me motivou a seguir crescendo profissionalmente.

Aos meus colegas de grupo de pesquisa, especialmente aos amigos Patrícia, Débora, Iara, Clairmont e Adenauer.

Aos professores Jhansy Silveira Collares, Cláudio Marques Ribeiro, Acauan Pereira Fernandes e Luiz Cláudio Dalmolin, pela ajuda valiosa.

Ao meu orientador, Prof. Dr. Cláudio Fernando Resin Geyer, cujo apoio e paciência inesgotáveis foram essenciais para que eu alcançasse meu objetivo. Nunca deixemos "cair a peteca".

À Melitta e à Coca-Cola, companheiras inseparáveis.

Ao amigo Fabrízio de Royes Mello, pelo apoio constante.

A DEUS, por ter me dado inteligência e uma vida feliz e cheia de saúde.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AFS	*Andrew File System*
ASP	*Active Server Pages*
bps	bits por segundo
CGI	*Common Gateway Interface*
DNS	*Domain Name System*
FTP	*File Transfer Protocol*
GPL	*GNU General Public License*
GPPD	Grupo de Processamento Paralelo e Distribuído
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
IIS	*Internet Information Server*
LAN	*Local Area Network*
ORPIS	*One Replication Protocol for Internet Servers*
P2P	*Peer-to-peer*
PDA	*Personal Digital Assistant*
PDF	*Portable Document Format*
PHP	*PHP: Hypertext Preprocessor*
REMMOS	*Replication Model in Mobility Systems*
RSH	*Remote Shell*
SGBD	Sistema de Gerência de Banco de Dados
SSH	*Secure Shell*
TCP	*Transfer Control Protocol*
UCP	*Unidade Central de Processamento*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WAN	*Wide Area Network*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O surgimento de novas aplicações que utilizam o protocolo HTTP nas suas transações e a crescente popularidade da *World Wide Web* (WWW) provocaram pesquisas pelo aumento do desempenho de servidores *Web*. Para tal, uma das alternativas propostas neste trabalho é utilizar um conjunto de servidores *Web* distribuídos que espalham a carga de requisições entre vários computadores, atuando como um só associado a uma estratégia de replicação de conteúdo. Um dos problemas centrais a ser resolvido em servidores *Web* distribuídos é como manter a consistência das réplicas de conteúdo entre os equipamentos envolvidos. Esta dissertação apresenta conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos. São mostrados detalhes sobre arquitetura de servidores *Web* distribuídos, manutenção da consistência em ambientes de servidores *Web* distribuídos, uso de replicação e formas de replicação. Além disso, são citados alguns trabalhos correlatos ao propósito de manter réplicas consistentes em ambientes de servidores *Web* distribuídos. Este trabalho tem por objetivo propor um modelo de manutenção da consistência de conteúdo em servidores *Web* distribuídos com características de transparência e autonomia. O modelo, denominado ***O**ne **R**eplication **P**rotocol for **I**nternet **S**ervers* (ORPIS), adota uma estratégia de propagação otimista porque não existe sincronismo no envio das atualizações para as réplicas. Este trabalho apresenta os principais componentes tecnológicos empregados na *Web*, além dos problemas causados pela escalabilidade e distribuição inerentes a esse ambiente. São descritas as principais técnicas de aumento de desempenho de servidores *Web* que atualmente vêm sendo utilizadas. O modelo ORPIS é descrito, sendo apresentados seus pressupostos, elencados seus componentes e detalhados os seus algoritmos de funcionamento. Este trabalho dá uma visão geral sobre a implementação e os testes realizados em alguns módulos do protótipo do modelo, caracterizando o ambiente de desenvolvimento do protótipo e detalhes da implementação. São enumerados os atributos e métodos das classes do protótipo e definidas as estruturas de dados utilizadas. Além disso, apresentam-se os resultados obtidos da avaliação funcional dos módulos implementados no protótipo. Um ponto a ser salientado é a compatibilidade do modelo ORPIS aos servidores *Web* existentes, sem a necessidade de modificação em suas configurações. O modelo ORPIS é baseado na filosofia de código aberto. Durante o desenvolvimento do protótipo, o uso de software de código aberto proporcionou um rápido acesso às ferramentas necessárias (sistema operacional, linguagens e gerenciador de banco de dados), com possibilidade de alteração nos códigos fonte como uma alternativa de *customização*.

Palavras-chave: Consistência, réplica, servidor *web* distribuído.

ORPIS: A REPLICATED CONTENT CONSISTENCY MODEL ON DISTRIBUTED WEB SERVERS

ABSTRACT

The arise of new applications that use HTTP in their transactions and the growing popularity of the World Wide Web (WWW) caused the search for high performance Web servers. In order to do so, one of the alternatives proposed in this work is the use of a group of distributed Web servers that distribute the load of many computers acting as one, associated to a content replication strategy. One of the main problems to be solved in distributed Web servers is how to keep consistency among the servers. This thesis introduces the fundamental concepts about content replication on distributed Web servers. Details about the distributed Web servers architecture, consistency maintenance on distributed Web servers environments and the use of replication and forms of replication are shown. Besides that, some works related to keeping consistent replicas in distributed Web servers are mentioned. This work aims to propose a content consistency management model on distributed Web servers, in a transparent and autonomous way. Such model, called **O**ne **R**eplication **P**rotocol for **I**nternet **S**ervers (ORPIS), adopts an optimistic propagation strategy because there is no synchronization in the process of updating the replicas. This work presents the main technological components used on the Web as well as the problems caused by the scalability and distribution inherent to this environment. The main techniques to enhance the Web servers performance currently in use are described. The ORPIS model is described, its principles are introduced, its components are listed and its algorithms are detailed. This work brings an overview of the implementation and tests carried out in some modules of the model's prototype, characterizing the prototype's developing environment and implementation details. The prototype's classes attributes and methods are listed and data structures used are defined. Besides this, the results from the functional evaluation of the implemented prototype modules are shown. Something to be emphasized is the compatibility of ORPIS model to existent Web servers, without requiring any modification on their configuration. The ORPIS model is based on open source philosophy. During the prototype development, the use of open source made possible an easy access to the necessary tools (operating systems, programming languages and database management systems), allowing source code editing as a customization option.

Keywords: Consistency, replica, distributed web server.

1 INTRODUÇÃO

1.1 Tema do trabalho

Este trabalho trata do tema gerenciamento da consistência de réplicas de conteúdo em servidores *Web* distribuídos. O foco principal é a elaboração de um modelo de manutenção da consistência de conteúdo em servidores *Web* distribuídos com características de transparência e autonomia.

1.2 Motivação

A Internet sofreu um crescimento exponencial desde sua popularização, no início dos anos 90. A *World Wide Web* (WWW ou simplesmente *Web*) tem crescido a uma taxa maior ainda, com o aparecimento de muitas aplicações anteriormente inexistentes: bibliotecas digitais, comércio eletrônico, áudio e vídeo sob demanda e educação a distância. Essas aplicações ocasionaram um aumento enorme do tráfego na Internet. Alguns sítios *Web* populares recebem milhões de acessos por dia. É fácil perceber que alguns desses possuem um tempo de resposta extremamente alto. Isso frustra muitos usuários e causa preocupação em muitos administradores *Web*. Um dos principais problemas enfrentados por esses administradores é adequar os recursos para atender às exigências dos usuários. Segundo Menascé e Almeida (1998), esse desafio exige que os administradores *Web* estejam aptos a monitorar pontos de contenção, prever a capacidade dos seus sítios *Web* e determinar a melhor maneira de resolver os problemas de desempenho causados pelo aumento da carga de trabalho imposta aos seus servidores.

Quanto mais os usuários interagem via WWW, maior é o problema da escalabilidade da infra-estrutura da *Web*. Para suportar esse crescimento, são necessárias novas formas de acelerar o tráfego ou, do contrário, ocorrerá um grande ponto de contenção. Um desafio atual é oferecer aplicações distribuídas, baseadas em ambiente *Web*, com desempenho aceitável. Um servidor *Web* com inúmeras funções é inútil se demorar para responder a um pedido de informação do cliente (MENASCÉ; ALMEIDA, 1998). À medida que os negócios crescem na Internet, o desempenho dos serviços *Web* torna-se mais crítico (MENASCÉ, 1998). Quanto mais informação e serviços uma empresa disponibiliza em um sítio *Web*, mais visitas ele recebe. Quanto mais visitas um sítio *Web* recebe, mais alta a probabilidade de os usuários terem de esperar muito por uma resposta.

Os problemas de desempenho na Internet e nas *intranets* são aumentados pela natureza imprevisível da busca das informações e das solicitações de serviços na *Web*. Em certos momentos, os sítios estão quase inativos, sem visitantes. De re-

pente, sem qualquer aviso, o tráfego pode aumentar enormemente. Quando há um grande aumento na taxa de solicitações aos servidores *Web*, além da capacidade dos mesmos, os tempos de resposta e erros na conexão aumentam significativamente. A sobrecarga pode acontecer devido à saturação da capacidade da Unidade Central de Processamento (UCP) ou da memória principal do servidor *Web* ou, até mesmo, da redução da capacidade de conexão do servidor à rede. A lista de pedidos *Transfer Control Protocol* (TCP) do servidor pode ficar sobrecarregada, ocasionando uma degradação no desempenho. Dessa forma, soluções são necessárias para atender os pedidos de maneira eficiente e com desempenho admissível.

A grande quantidade de trabalhos publicados sobre o assunto nos últimos anos indica preocupação com as possíveis soluções dos vários aspectos relacionados ao aumento do desempenho global do ambiente da *Web*. Uma das soluções apresentadas em Lima e Geyer (2000) é o uso de servidores *Web* distribuídos. Essa abordagem espalha a carga de requisições *HyperText Transfer Protocol* (HTTP) entre vários computadores conectados atuando como um só com o objetivo de proporcionar um melhor desempenho: um *cluster*. Um servidor *Web* distribuído exporta um nome lógico único e informa-o para o mundo externo. Do ponto de vista do cliente, que envia uma única requisição HTTP, o grupo de servidores é apenas um servidor que irá manipular o pedido HTTP da mesma maneira que qualquer outro servidor *Web* faz (GARLAND et al., 1995). A partir do recebimento do pedido, o *cluster* dirige-o a um dos membros do grupo para o processamento. Cada um dos componentes do *cluster* possui uma réplica do conteúdo a ser oferecido por esse servidor *Web*. Esse *cluster* pode estar instalado fisicamente em um mesmo local ou distribuído geograficamente em diferentes pontos da Internet.

Um problema fundamental em sistemas distribuídos é a manutenção da consistência das réplicas. As cópias do conteúdo *Web* devem permanecer consistentes, para que então todos os servidores replicados possam coordenar suas leituras e escritas (EKENSTAM et al., 2001). Um dos aspectos a serem considerados nessa abordagem é a política de sincronização da atualização das réplicas dos dados no *Web cluster*. Quando da atualização das páginas de um dos servidores, os outros componentes do *cluster* devem refletir exatamente o mesmo conteúdo. A replicação de objetos em sistemas distribuídos, normalmente, é utilizada para torná-los mais confiáveis e seguros, pois o sistema pode sobreviver a falhas de uma ou mais cópias do componente replicado.

Nos ambientes de *Web cluster*, a replicação permite um aumento do desempenho através do balanceamento de carga entre os componentes, proporcionando tolerância a falhas quando um dos servidores deixa de funcionar. Os componentes do *Web cluster* podem estar instalados em um local físico ou espalhados em locais estratégicos na Internet, mais próximos dos clientes, em um ambiente de rede *Wide Area Network* (WAN).

Pretende-se, com este trabalho, propor um modelo que permita a manutenção da consistência do conteúdo de servidores *Web* distribuídos e replicados – *Web cluster*, com características de transparência e autonomia. Esse modelo foi denominado *One Replication Protocol for Internet Servers* (ORPIS).

1.3 Contexto

Este trabalho insere-se no contexto de trabalhos de pesquisa relacionados à busca por soluções de gerenciamento da consistência de conteúdo em ambientes de servidores *Web* distribuídos. O capítulo três apresenta alguns trabalhos relacionados a tal contexto.

O presente trabalho também se insere no contexto local do grupo de pesquisa em Processamento Paralelo e Distribuído (GPPD) do Instituto de Informática da UFRGS, que produziu o trabalho "Um Servidor de Arquivos Multiversão Replicados", de Fernando L. Pedone (PEDONE, 1995), o qual apresentou um sistema de arquivos tolerante a falhas baseado no uso de operações atômicas e replicação de dados. Mais recentemente, o trabalho intitulado "Um Modelo de Replicação em Ambientes que Suportam Mobilidade", de autoria de Débora Nice Ferrari (FERRARI, 2000), apresentou o modelo ReMMoS (*Replication Model in Mobility Systems*), com o objetivo de prover desempenho em ambientes que permitem mobilidade de objetos.

1.4 Objetivos

1.4.1 Objetivo geral

O objetivo geral é elaborar um modelo de manutenção da consistência de réplicas em conteúdos de servidores *Web* distribuídos, com características de transparência e autonomia. A implementação do modelo é baseada em ferramentas de código aberto.

1.4.2 Objetivos específicos

- identificar e estudar alguns modelos de atualização de réplicas existentes;
- projetar um modelo de manutenção da consistência do conteúdo em ambientes de servidores *Web* distribuídos;
- implementar um protótipo de alguns módulos do modelo proposto utilizando ferramentas de software livre;
- validar o modelo através de testes funcionais aplicados nos módulos que foram desenvolvidos.

1.5 Contribuição do autor

Este trabalho tem as seguintes contribuições:

- sistematização de uma discussão sobre as técnicas de aumento de desempenho de servidores *Web* no contexto do Grupo de Processamento Paralelo e Distribuído do Instituto de Informática da Universidade Federal do Rio Grande do Sul;
- definição de um modelo para gerenciamento do conteúdo em um ambiente de servidores *Web* distribuídos, utilizando características de transparência e autonomia:

  - replicação transparente – permite que haja várias instâncias do conteúdo e que as mesmas sejam usadas sem prévio conhecimento das réplicas pelos usuários, proporcionando menor tempo de acesso ao sítio;
  - gerenciamento autônomo das atualizações de conteúdo – o modelo desenvolvido permite o gerenciamento autônomo da consistência das réplicas em servidores *Web* distribuídos geograficamente. A idéia é oferecer uma ferramenta que automatize o processo de gerenciamento da consistência das réplicas;

- implementação de protótipos dos módulos principais do modelo e avaliação de desempenho dos mesmos;

- criação de uma solução extremamente portável e que pode ser livremente usada por administradores de sítios *Web* distribuídos – os protótipos da ferramenta originada a partir do modelo foram implementados utilizando ferramentas de software livre (*http://www.fsf.org*, *http://www.gnu.org*), criando uma solução economicamente viável e acessível a qualquer interessado. Essa disponibilidade permite que se possa utilizar, modificar e, até mesmo, redistribuir novas versões da ferramenta, o que possibilita o surgimento de novas idéias e versões para a mesma, o que é inerente ao conceito de software livre.

1.6 Estrutura do texto

O capítulo dois deste trabalho apresenta a tecnologia que a *Web* emprega, bem como os problemas causados pela escalabilidade e distribuição inerentes a esse ambiente. Esse capítulo encerra apresentando as principais técnicas de aumento de desempenho que atualmente vêm sendo empregadas na *Web*. O capítulo três apresenta conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos, suas características e problemas inerentes à sua implementação e gerenciamento. Esse capítulo também apresenta os trabalhos relacionados à manutenção da consistência de réplicas. A proposição do modelo ORPIS é feita no quarto capítulo, onde são discutidas sua arquitetura e suas principais características. O capítulo cinco descreve a implementação e os testes funcionais feitos no protótipo do modelo ORPIS. O capítulo seis apresenta as conclusões, as limitações e os possíveis trabalhos futuros relacionados ao tema.

2 TÉCNICAS DE AUMENTO DE DESEMPENHO DE SERVIDORES WEB

Este capítulo apresenta a tecnologia empregada na *Web*, bem como os problemas causados pela escalabilidade e distribuição inerentes a esse ambiente. O capítulo encerra apresentando as principais técnicas de aumento de desempenho que atualmente vêm sendo empregadas na *Web* que foram selecionadas a partir de dois critérios: relevância prática e tempo disponível para escrita desta publicação. As referências foram sendo buscadas em eventos específicos que tratam o tema aumento do desempenho da WWW. Escolheram-se publicações, basicamente, do ano de 1997 em diante.

A seção de *caches* não pretende ser exaustiva, visto que o tema é amplamente discutido em muitas publicações. Apresenta-se apenas um panorama geral das principais tendências, e são enumerados alguns conceitos necessários à compreensão do assunto em debate neste trabalho.

2.1 A *World Wide Web*

A *Web* é um sistema em evolução para a publicação de recursos e serviços e para o acesso aos mesmos através da Internet. A Internet é um conjunto de milhares de redes de computadores que servem a milhões de pessoas em todo o mundo (LAQUEY; RYER, 1994). Através dos programas conhecidos como navegadores (*browsers*), tais como Netscape e Internet Explorer, os usuários navegam na *Web* para recuperar e visualizar documentos de vários tipos, para escutar transmissões de áudio e ver vídeos, além de interagir com um conjunto ilimitado de serviços (COULOURIS; DOLLIMORE; KINDBERG, 2001). Os navegadores são clientes *Web* e são programas que permitem a requisição dos recursos disponíveis na WWW (MEIRA; MURTA; RESENDE, 2000).

Uma das características principais mais importantes da *Web* é que ela proporciona uma estrutura de hipertexto entre os documentos que armazena, refletindo a necessidade dos usuários de organizarem seu conhecimento. A *Web* proporciona ligações de um documento localizado em um servidor a outro, localizado em outro servidor, mantendo a localização real e o método de acesso invisível para o usuário. Esse modelo permite a descentralização da *Web* e contribui para sua escalabilidade. Não há hierarquia para clientes e servidores, nem mesmo um controle central. Os clientes podem acessar os servidores através de um navegador, a partir de qualquer máquina conectada à Internet, independente do tipo de documento, do sistema operacional ou do hardware. Um sistema é descrito como escalável se continuar sendo

efetivo quando houver um aumento significativo no número de recursos e no número de usuários (COULOURIS; DOLLIMORE; KINDBERG, 2001).

Nesse sentido, a *Web* é interoperável, pois permite que microcomputadores de arquiteturas diferentes acessem-na. Além disso, é multiplataforma, pois existem navegadores para a maioria dos sistemas operacionais.

A *Web* é também um meio de publicação e compartilhamento de informações instantâneo, de amplo alcance e de custo relativamente baixo.

Inicialmente, a *Web* foi vista como uma grande biblioteca digital, com capacidade para armazenar todo o conhecimento humano, oferecendo acesso gratuito a essas informações. Hoje em dia, é vista também como um meio para a realização de comércio.

A Internet, por sua vez, é composta por um conjunto de redes de computadores que se comunicam e cooperam entre si. As redes que compõem a Internet utilizam diversos tipos de tecnologia, protocolos, meios de transmissão e computadores. A *Web* é um exemplo de sistema distribuído, pois está construída sobre servidores conectados via rede e é acessada por clientes. O conteúdo que os servidores oferecem é compartilhado entre os clientes. Documentos podem ser compostos por arquivos contendo texto, imagens, gráficos, áudio e vídeo, entre outras mídias. As conexões entre documentos de formatos diversos compõem o grande sistema hipermídia global que é a *Web*.

Um hiperdocumento publicado na *Web* pode ser acessado das mais diferentes formas, apenas com pequenas variações em seu aspecto visual. Isso a torna a plataforma ideal para a construção de sistemas interoperáveis, que sejam escritos uma vez e que possam ser executados em qualquer plataforma cliente.

A *Web* é um sistema aberto: ela pode ser estendida e implementada de diferentes maneiras sem prejudicar sua funcionalidade existente. Primeiro, sua operação é baseada em padrões de comunicação e de documentos que são livremente publicados e amplamente implementados. Qualquer navegador que tiver sua implementação em conformidade com os padrões adotados pode obter recursos de qualquer servidor compatível. Os usuários, então, têm acesso, através dos navegadores, a partir da maioria dos dispositivos que usam, desde assistentes pessoais (PDAs) até computadores de mesa. A *Web* é aberta em relação aos tipos de recursos que podem ser publicados e compartilhados. Simplificadamente, um recurso na *Web* é uma página *Web* ou algum outro tipo de conteúdo que possa ser armazenado em um arquivo e apresentado ao usuário, tal como arquivos de programas, arquivos de mídia e documentos em formato PostScript ou *Portable Document Format* (PDF). Se alguém inventar, por exemplo, um novo formato de armazenamento de imagens, as imagens nesse formato podem ser imediatamente publicadas na *Web*. Os usuários necessitam de uma maneira de visualizar as imagens nesse novo formato, mas os navegadores são projetados para acomodar funcionalidade de apresentação de novos conteúdos na forma de aplicações "assistentes" e "plug-ins".

A *Web* evoluiu sem mudar sua arquitetura básica, sendo baseada em três principais componentes tecnológicos (COULOURIS; DOLLIMORE; KINDBERG, 2001):

- a linguagem HTML, usada para especificar os conteúdos e a diagramação das páginas quando elas são mostradas pelos navegadores;
- URLs, que identificam os documentos e outros recursos armazenados como partes da *Web*;

- um sistema de arquitetura cliente-servidor, com regras padrões para interação (o protocolo HTTP), através das quais os navegadores recuperam documentos e outros recursos a partir dos servidores *Web*.

A seguir, são descritos, de forma mais detalhada, cada um desses componentes tecnológicos da *Web*.

2.1.1 A linguagem de marcação HTML

Documentos *Web* são compostos tipicamente por um arquivo no formato da linguagem de marcação de hipertexto *HyperText Markup Language* (HTML) e algumas imagens inclusas. Cada um desses componentes é um objeto independente na *Web*. A HTML permite (RAGGET, 1999):

- publicar documentos com cabeçalhos, textos, tabelas, listas e imagens;
- buscar informação através de ligações de hipertexto;
- desenhar formulários para conduzir transações com serviços remotos, para uso em busca de informações, reservas, pedidos de produtos, etc;
- incluir planilhas, video-clipes, áudio e outros tipos de aplicações diretamente em seus documentos.

Ligações de hipertexto expressam uma ou mais relações (explícitas ou implícitas) entre dois ou mais recursos (LAVOIE; NIELSEN, 1999). Ligações explícitas disparam pedidos de recursos *Web* através da intervenção do usuário. Já as ligações implícitas fazem requisições de outros recursos, sem a ação do usuário.

A figura 2.1 apresenta a anatomia de uma transação HTTP.

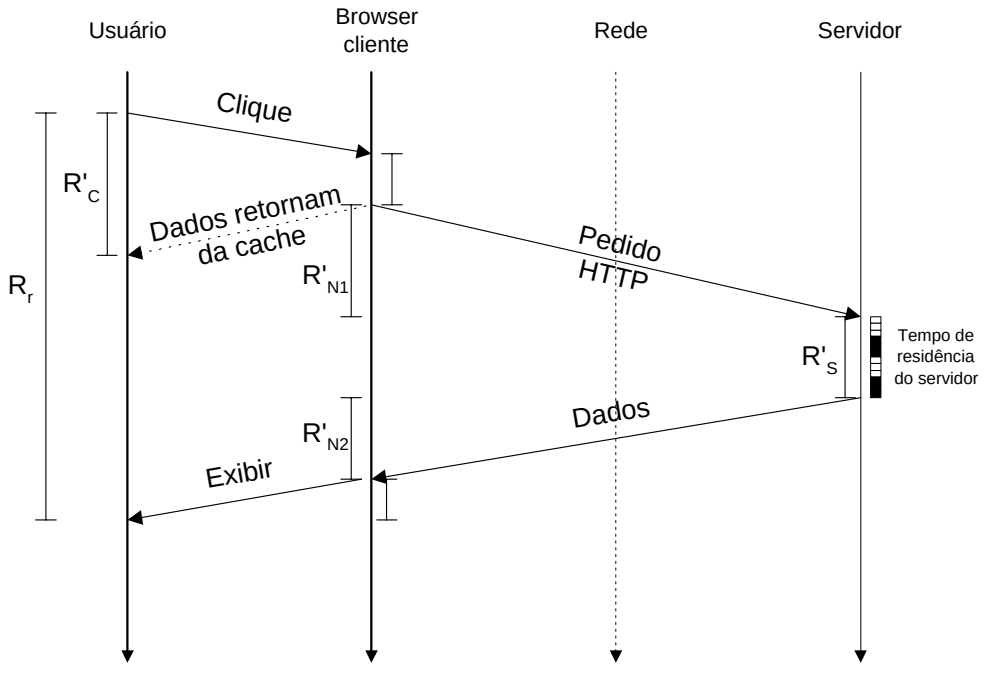

Figura 2.1: Anatomia de uma transação HTTP, traduzido de Menascé e Almeida (1998)

As ligações podem apontar para imagens dentro do documento HTML, o que tem um grande impacto no desempenho do servidor. Quando o navegador faz a leitura dos dados HTML recebidos do servidor, reconhece as ligações associadas com as imagens e automaticamente solicita os arquivos de imagens apontados pelas ligações. Na verdade, um documento HTML é a combinação de vários objetos que gera vários pedidos separados ao servidor. Enquanto o usuário observa apenas um único documento, o servidor atende a uma série de pedidos para esse documento. Em termos de desempenho, é importante observar o conceito de que um simples clique de um usuário gera uma série de pedidos de arquivos ao servidor *Web*.

2.1.2 *Uniform resource locators*

O objetivo de um *Uniform Resource Locator* (URL) é identificar um recurso de maneira que o navegador possa localizar esse recurso. Os navegadores examinam os URLs com o objetivo de requisitar os recursos correspondentes dos servidores *Web*. As páginas *Web* são um conjunto de informações, constituído de um ou mais recursos, o que se pretende representar simultaneamente e é identificado por um único URL (BERNERS-LEE et al., 1998). Um arquivo HTML que contenha uma imagem e uma *applet* Java, identificadas e acessadas através de um único URL e representadas por um cliente *Web*, pode ser um exemplo de página *Web*.

Em geral, URLs têm a seguinte forma (COULOURIS; DOLLIMORE; KINDBERG, 2001):

```
http://nome_do_servidor[:porta][/caminho_no_servidor][?argumentos]
```

2.1.3 O protocolo HTTP

O HTTP é um protocolo de nível de aplicação localizado na camada mais alta do protocolo TCP, utilizado na comunicação entre clientes e servidores na *Web* (BERNERS-LEE; FIELDING; FRYSTYK, 1996). O HTTP define uma interação simples de pedido-resposta chamada de "transação *Web*". Cada interação HTTP consiste em um pedido do cliente para o servidor, seguido de uma resposta enviada de volta ao cliente. Os dados transferidos podem ser de formatos diversos como texto, imagens, hipertexto, áudio ou qualquer outro formato. Formatos proprietários podem ser definidos sem necessidade de padronização. No entanto, é necessária uma negociação entre cliente e servidor sobre o formato da resposta, pois o cliente pode não estar apto a tratar todos os tipos de formato.

Na Internet, a comunicação via HTTP geralmente ocorre sobre uma conexão TCP. A porta padrão é TCP (80), mas outras portas podem ser utilizadas. O protocolo HTTP é independente de estados, isto é, os objetos são retornados com base no URL e de forma independente de qualquer operação prévia realizada pelo cliente. O protocolo não armazena informações sobre requisições anteriores, não definindo, portanto, o conceito de estados. O protocolo HTTP tem sido utilizado como o protocolo da WWW desde 1990 e é responsável por cerca de 80% do tráfego nos *backbones* da Internet. As versões do HTTP utilizadas atualmente são a 1.0 e a 1.1.

2.1.4 Servidores *web*

Um servidor *Web* é um programa apto a responder a uma requisição WWW. A resposta corresponde a uma página estática ou a uma página dinâmica. Páginas estáticas são construídas a partir de arquivos lidos no servidor. Páginas dinâmicas são as geradas em tempo real em função de parâmetros enviados na requisição. Estas páginas são compostas, em sua maioria, por resultados de consultas a bancos de dados, resultados de processamento, ou atualizações feitas no servidor (MEIRA; MURTA; RESENDE, 2000).

Atualmente, as implementações mais populares de servidores *Web* são o Apache e o Microsoft Internet Information Server (IIS). O Apache (*http://www.apache.org*) é considerado o mais popular e mais seguro. Estima-se que sua participação no mercado de servidores *Web* seja próxima de 60%. Seu projeto tem como principal objetivo ser extremamente seguro, em detrimento inclusive do desempenho. A filosofia de desenvolvimento do Apache é de ser software livre e de código aberto. Tem versões para diversos sistemas operacionais: Unix, Linux e Windows 32 bits. Já o IIS, desenvolvido pela Microsoft, é um produto de código fechado e possui versões apenas para servidores Windows NT e Windows 2000.

A figura 2.2 apresenta os componentes típicos de uma interação entre um cliente e um servidor *Web*.

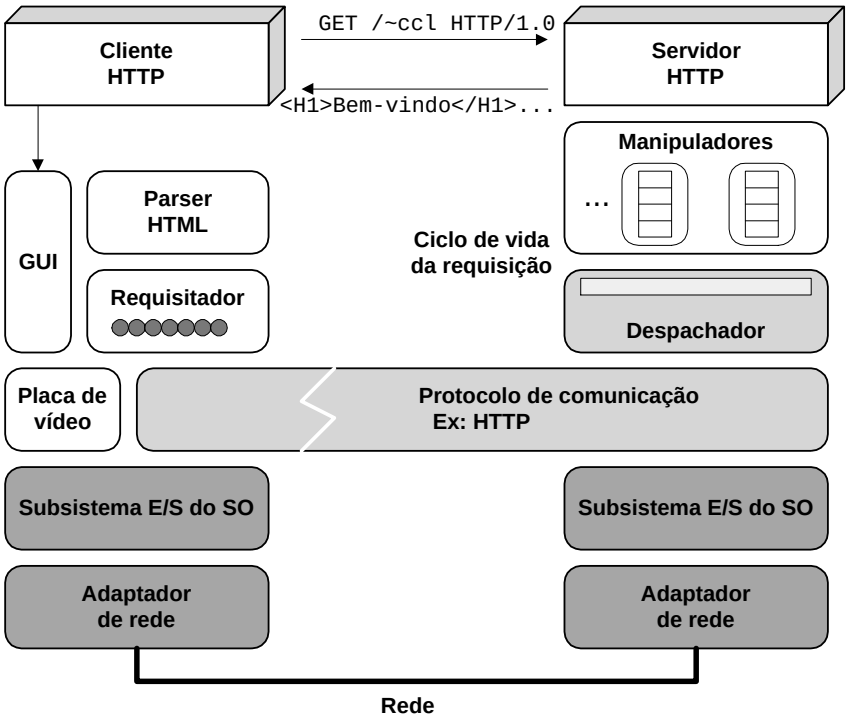

Figura 2.2: Um sistema *Web* típico, traduzido de Hu (1998)

2.1.5 Discussão sobre a *web*

A *Web* torna-se lenta devido a sua natureza de oferecer de maneira transparente recursos distantes geograficamente. O usuário acessa um hiperdocumento dentro de outro, e essa ação transporta-o para outro servidor localizado em uma parte diferente do mundo. Porém, entre esses dois documentos, existe um grande meio de

transporte, composto por enlaces de redes das mais variadas velocidades, roteadores, tráfego de outros usuários e aplicações diversas. O tempo de resposta, geralmente, não é o mais adequado para atender às expectativas dos usuários. Além disso, outros fatores podem torná-la exageradamente lenta, tais como redes sobrecarregadas devido ao grande número de usuários disputando um mesmo meio físico, ou mesmo a utilização de elementos multimídia em páginas *Web* a fim de atrair a atenção do usuário e aproveitar ao máximo os recursos dos processadores atuais.

Pode haver picos no número de acessos a um determinado documento localizado em um servidor *Web*. Esse fenômeno foi batizado de *The Slashdot Effect* (NOBLE et al., 1999; PIERRE; KUZ; STEEN, 2000; CLARKE et al., 2001), descrito originalmente em Adler (1999). Isso ocorre quando um documento é apontado, por exemplo, por um sítio popular de notícias, e acessado por milhões de usuários diariamente. Durante esses momentos de pico, quando a carga aumenta vertiginosamente, não só o tempo de resposta da requisição desse documento é aumentado, mas também o tempo de todos os outros documentos hospedados no mesmo sítio.

Problemas de desempenho podem trazer todo tipo de conseqüências indesejadas, tais como prejuízos financeiros ou de vendas, produtividade reduzida ou, até mesmo, uma má reputação para uma empresa (MENASCÉ; ALMEIDA, 1998). À medida que os negócios crescem na Internet, o desempenho dos serviços *Web* torna-se mais crítico. Quanto mais informação e serviços uma empresa disponibiliza em um sítio *Web*, mais visitas ele recebe. Quanto mais visitas um sítio *Web* recebe, mais alta a probabilidade de os usuários terem de esperar muito por uma resposta. Em muitos casos, os usuários ou clientes compararam como uma empresa comporta-se em relação aos seus competidores. Clientes insatisfeitos trocam de um sítio X para um sítio Y através de um simples clique do mouse.

Os problemas de desempenho na Internet e nas intranets são incrementados pela natureza imprevisível da busca das informações e das solicitações de serviços na WWW. Em certos momentos, os sítios estão quase inativos, sem visitantes. De repente, sem qualquer aviso, o tráfego pode aumentar enormemente. Esse tipo de pico de carga é enfrentado por muitos sítios. Alguns podem receber um grande número de visitas durante a publicação de uma lista de aprovados em um concurso, por exemplo, e permanecerem com poucas visitas durante a maior parte do ano. Outro exemplo desse fenômeno aconteceu no sítio da Enciclopédia Britânica, que recebeu uma enorme quantidade de visitas no momento em que tornou gratuito seu conteúdo. Mais uma oportunidade em que esse fenômeno ocorre: o sítio *Web* da Receita Federal fica quase indisponível poucas horas antes de expirar o prazo de entrega da declaração de Imposto de Renda. O provedor de conteúdo Terra Networks (*http://www.terra.com.br*) teve que usar um plano de contingência para atender à crescente demanda por notícias sobre os atentados acontecidos nos Estados Unidos, em setembro de 2001. Alguns servidores de conteúdo extra tiveram que ser colocados à disposição da imensa quantidade de usuários concorrentes que estavam requisitando as páginas com conteúdo relativo aos atentados.

A área de pesquisa que busca solucionar tais problemas é extensa e rica em propostas que pretendem oferecer soluções. O presente trabalho se insere nessa área, que busca por soluções que minimizem os efeitos produzidos por picos de carga de requisições aos servidores *Web*.

2.2 Técnicas de aumento do desempenho da *web*

Com relação às propostas de aumento do desempenho dos servidores *Web*, notam-se duas grandes tendências. A primeira pretende aumentar a velocidade com que a informação chega ao cliente. Já a segunda tem por objetivo resolver o problema dos picos de carga ocorridos pela grande quantidade de requisições enviadas, simultaneamente, ao servidor *Web*.

2.2.1 Adaptação de conteúdo

Quando há um grande aumento na taxa de solicitações aos servidores *Web*, além de sua capacidade, os tempos de resposta e erros na conexão aumentam significativamente. A sobrecarga pode acontecer devido à saturação da UCP ou da memória principal do servidor *Web* ou, até mesmo, da redução da capacidade de conexão do servidor à rede. A técnica proposta em Abdelzaher e Bhatti (1999) é projetada para aliviar condições de picos de carga e não com sobrecarga permanente, que pode ser solucionada com melhorias na plataforma.

Em Abdelzaher e Bhatti (1999) é proposto uma solução para resolver o problema da sobrecarga em um servidor *Web*, a adaptação de conteúdo para minimizar a sobrecarga no servidor. Essa técnica preocupa-se com o gerenciamento da sobrecarga sobre um único servidor, ao contrário das outras técnicas de distribuição de conteúdo em diversos servidores.

Geralmente, os servidores são superdimensionados ou usam controle de admissão (ABDELZAHER; BHATTI, 1999). Quando são superdimensionados, os administradores alocam duas vezes mais capacidade a um servidor *Web* do que é normal, o que nem sempre resolve o problema de sobrecarga. O controle de admissão atende a um número X de usuários, rejeitando a admissão dos próximos que solicitarem os serviços.

Possivelmente, os clientes queiram receber uma versão simples, sem todos os recursos do conteúdo solicitado. Essa premissa admite a substituição dos esquemas de controle de admissão por mecanismos de adaptação do conteúdo em condições de sobrecarga. Sob condições de cargas pesadas, conteúdo com menos recursos pode ser oferecido.

O conteúdo deve ser adaptado de maneira a preservar a informação essencial ainda que reduza alguns dos recursos. Quando o servidor encontra-se sobrecarregado, a introdução de um estágio a mais de computação, tal como filtragem de dados ou compressão, só irá aumentar ainda mais a carga do servidor. Por isso, a fim de eliminar *overhead* na sobrecarga, o conteúdo deve ser pré-processado *a priori* e armazenado em várias cópias que são diferentes em qualidade e necessidades de processamento. Quando sobrecarregado, o servidor deve possibilitar a troca para uma versão mais leve do conteúdo.

2.2.2 *Caches*

Uma *cache* é um local de armazenamento de dados usados recentemente, situado mais próximo que os dados propriamente ditos (COULOURIS; DOLLIMORE; KINDBERG, 2001). O uso de *caches* na WWW tem sido empregado para aumentar a eficiência e a confiabilidade das informações disponibilizadas na Internet. Uma *cache* próxima ao cliente pode oferecer uma página armazenada bem mais rápido do que ofereceria se ele tivesse que buscá-la através da rede. Pode-se concluir que,

se os pedidos são interceptados por *caches* próximas, poucos deles serão enviados ao servidor *Web*, reduzindo a carga do servidor e o tráfego da rede, proporcionando um aumento no desempenho. A utilização de *caches* diminui a necessidade de largura de banda, reduzindo em 35% ou mais o tráfego entre os clientes e os servidores *Web* (CHANDHOK, 1999).

A localização física das caches é crítica para a sua eficiência e funcionalidade. Há quatro configurações básicas, cada uma favorecendo um grupo diferente de clientes e arquivos (PALPANAS; KRISHNAMURTHY, 1999):

- no cliente – somente um único cliente é beneficiado. No entanto, esses benefícios são bem restritos, já que os recursos alocados para esse tipo de *cache* são limitados, na maioria das vezes;

- no servidor – nesse caso a *cache* possibilita que alguns arquivos muito solicitados sejam servidos mais rapidamente direto da memória, em vez de serem lidos a partir do disco. Além disso, o servidor economiza tempo e recursos usados para acessar o disco;

- próximo aos clientes – existe uma *cache* comum para uma grande comunidade de usuários. Os recursos alocados permitem que a *cache* armazene um número significativo de arquivos originários de muitas fontes e os usuários se beneficiem dos padrões de acesso de seus vizinhos;

- próximo aos servidores – a *cache* armazena arquivos freqüentemente acessados pertencentes a um pequeno conjunto de servidores. Tais *caches* atuam como um mecanismo para disseminar informação popular aos clientes.

Uma *cache* é considerada passiva se reagir somente em conseqüência de um pedido do cliente. Um arquivo é carregado e armazenado na *cache* somente se algum cliente o solicitar. Já as *caches* ativas empregam um mecanismo inteligente para determinar pedidos futuros dos usuários. Por isso, elas buscam antecipadamente (*prefetch*) os documentos que podem ser acessados no futuro próximo e que não estão armazenados na *cache*, o que resulta em um aumento de desempenho.

2.2.3 *Prefetch*

Segundo Palpanas e Krishnamurthy (1999), para se obter um aumento de desempenho, os algoritmos de *caches* devem ser suplementados com outros métodos que aumentem a taxa de acertos (*cache hits*) e que reduzam o tempo de latência. *Prefetch* é um conceito que vai nessa direção e é o nome dado à estratégia de buscar dados antes de os mesmos serem solicitados (WILKINSON, 1996). Essa operação consiste em fazer com que a *cache* responda às solicitações através da busca adiantada dos pedidos dos clientes.

Um método simplista de se implementar *prefetch* é fazer com que a aplicação informe ao sistema operacional suas futuras necessidades. Quando esse método funciona, é muito eficiente, já que a aplicação sabe exatamente quais arquivos serão solicitados e em que momento do futuro isso acontecerá. Porém os problemas dessa técnica são significativos. Em primeiro lugar, as aplicações precisam ser reescritas para se beneficiarem desse método. Portanto, a tarefa de programar uma aplicação desse tipo torna-se extremamente incômoda. O programador deve aprender e usar um conjunto de complexas diretivas para manipular funções de baixo nível com

uma forte dependência dos recursos físicos. Além disso, apenas o conhecimento dos pedidos futuros não é suficiente. A "pré-busca" (*prefetching*) deve acontecer em um tempo significativo, antes de um documento ser efetivamente solicitado, para garantir que esteja disponível quando for pedido. Isso impõe uma dificuldade na programação.

O algoritmo de *prefetch* é baseado em informação sobre os padrões de acesso ocorridos no passado. Dependendo da origem dessa informação, os algoritmos de *prefetch* podem ser classificados em três categorias (PALPANAS; KRISHNAMURTHY, 1999):

- no cliente – o algoritmo usa informações localizadas no cliente para fazer previsões. Essas informações podem pertencer a um único usuário ou a uma comunidade de usuários. O algoritmo é ajustado para cada cliente;
- no servidor – somente informações localizadas no servidor são usadas para controlar o algoritmo. As previsões são baseadas nos padrões de referência de muitos usuários e somente se adaptam aos arquivos armazenados nos servidores participantes;
- no cliente e no servidor – informações dos dois lados são combinadas para o algoritmo de *prefetch*. As previsões podem ser adaptadas a cada usuário particular, enquanto se mantém o conhecimento específico para cada arquivo que o servidor oferece.

2.3 Considerações finais

Este capítulo apresentou os componentes tecnológicos empregados na *Web*, bem como os problemas causados pela escalabilidade e distribuição inerentes a esse ambiente. Também foram apresentadas as principais técnicas de aumento de desempenho que atualmente vêm sendo empregadas na *Web* e que foram selecionadas a partir de dois critérios: relevância prática e tempo disponível para a escrita desta publicação. Foram referenciadas as técnicas citadas em publicações mais recentes. O próximo capítulo apresenta conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos.

3 SERVIDORES WEB DISTRIBUÍDOS

Este capítulo apresenta conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos. São apresentados detalhes sobre arquitetura de servidores *Web* distribuídos, manutenção da consistência em ambientes de servidores *Web* distribuídos, uso de replicação e formas de replicação. Ao final, são apresentados alguns trabalhos relacionados ao propósito de manter réplicas consistentes em ambientes de servidores *Web* distribuídos.

3.1 Arquitetura de servidores *web* distribuídos

Com o crescimento constante do número de acessos diários aos sítios *Web*, o tráfego pode ser muito grande para um único computador manipulá-lo de maneira eficiente. A solução mais óbvia é colocar mais servidores a trabalhar, espalhando a carga de pedidos entre vários computadores conectados atuando como um só. Um mecanismo especializado mantém a organização do acesso, despachando a requisição para o servidor mais adequado, através de uma política de escolha de qual o melhor servidor a ser utilizado no momento. Devido ao fato de que os servidores tornaram-se menores e mais baratos e também com o aumento do desempenho das redes de comunicação, hoje em dia é possível agrupá-los de modo que pareçam aos usuários apenas um único sistema. Esse grupo de servidores é conhecido como um agregado (*cluster*). Segundo Sato (2001), um "*cluster* de estações" é constituído de um conjunto de computadores, denominados nós, interconectados por uma rede de alta velocidade, e classificados como multicomputadores. Um *Web cluster* é um conjunto de servidores e tecnologias *middleware* que permitem que um sítio *Web* ofereça informação para uma *intranet* ou para a Internet (figura 3.1).

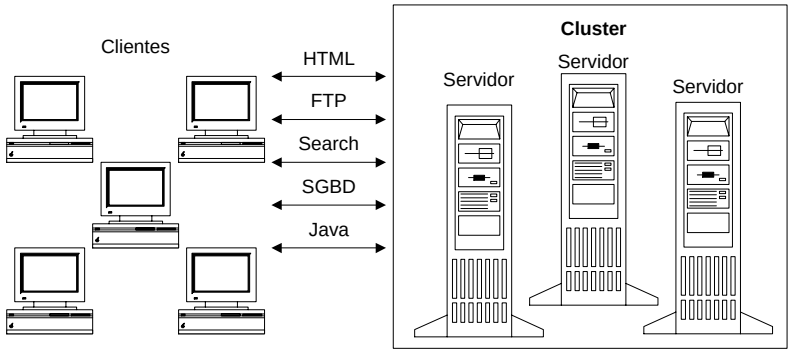

Figura 3.1: Arquitetura de um *Web cluster*, traduzido de Menascé e Almeida (1998)

O agregado funciona para o cliente como um sistema único, aumentando a disponibilidade total do sistema e o desempenho, mas pode ser de difícil gerenciamento. Em um sistema único de alto desempenho do tipo *mainframe*, o administrador precisa gerenciar apenas uma máquina. No agregado de servidores, o administrador deve instalar software em vários servidores, configurar e operar cada um deles, e garantir que os dados, tais como o conteúdo do sítio *Web* seja o mesmo em cada sistema. Isso pode ser um desafio e é a razão primária pela qual as tecnologias de replicação de dados estão se tornando importantes para o mercado (POLYSERVE, 2000).

Existem três formas diferentes de soluções de agregados: de rede, de dados e de processos. Agregados de redes gerenciam a interface da rede ao agregado. Tipicamente, esses elementos incluem sobreposição de IP, *heartbeat*, balanceamento de carga e várias formas de monitoração de serviços e dispositivos. *Heartbeat* é a função de verificar se um determinado componente do agregado ainda se encontra funcionando, através de envio e retorno de pacotes (ROBERTSON, 2000). Agregados de dados gerenciam o acesso a recursos de armazenamento compartilhados do agregado. Agregados de processos permitem que vários nós participem no processamento de uma única aplicação.

Servidores *Web* distribuídos necessitam de mecanismos de decisão do servidor replicado que deverá responder às requisições dos clientes. A situação ideal é descobrir o servidor replicado mais apropriado com o qual o cliente em questão possa se comunicar com o melhor desempenho possível. Essa otimização é uma política normalmente estabelecida por proximidade, mas também pode ser baseada em outros critérios, como, por exemplo, a carga de pedidos. As formas mais comuns existentes são (COOPER; MELVE; TOMLINSON, 2000):

- ligações de navegação: a maneira mais simples de comunicação entre clientes e réplicas. Esse mecanismo usa URLs individuais dentro das páginas que apontam para os servidores replicados. O cliente seleciona manualmente a ligação do servidor replicado que deseja usar;

- redirecionamento de HTTP: os clientes são redirecionados para um servidor replicado ótimo através do uso dos códigos de resposta do protocolo HTTP: "302 *Found*" ou "307 *Temporary Redirect*". O cliente estabelece comunicação com um dos servidores replicados, e este pode escolher aceitar o serviço ou redirecionar o cliente novamente;

- redirecionamento através de DNS: o *Domain Name Service* (DNS) oferece um sofisticado mecanismo de comunicação entre clientes e réplicas. Isso é possível pelos servidores DNS que ordenam os endereços IP resolvidos, baseado em políticas de serviço. Quando um cliente converte o nome de um servidor, o servidor DNS ordena os endereços IP dos servidores replicados, iniciando pela melhor réplica e terminando na réplica menos apropriada.

O redirecionamento através de DNS tem problemas, podendo as resoluções de nomes para números IP nos servidores de nomes causar desbalanceamento no caso de o cliente seguir tentando acessar uma réplica que não está mais disponível. Além disso, a distribuição da carga não será equivalente em todos os componentes do *cluster*, já que o padrão de acesso de cada cliente é diferente.

3.2 Consistência em servidores *web* distribuídos

Um dos aspectos a serem considerados em servidores *Web* distribuídos é a política de manutenção da consistência das réplicas do conteúdo. Quando da atualização das páginas de um dos servidores, os outros componentes do *cluster* devem refletir exatamente o mesmo conteúdo.

Ao se acrescentar servidores *Web* distribuídos, vem à tona a dúvida se se deve replicar todo o conteúdo, isto é, manter várias cópias dos mesmos dados ou simplesmente particionar o conteúdo entre diversos servidores. Ao se escolher a primeira opção, deve-se estar ciente da preocupação adicional em manter esse conteúdo consistente entre os diferentes servidores.

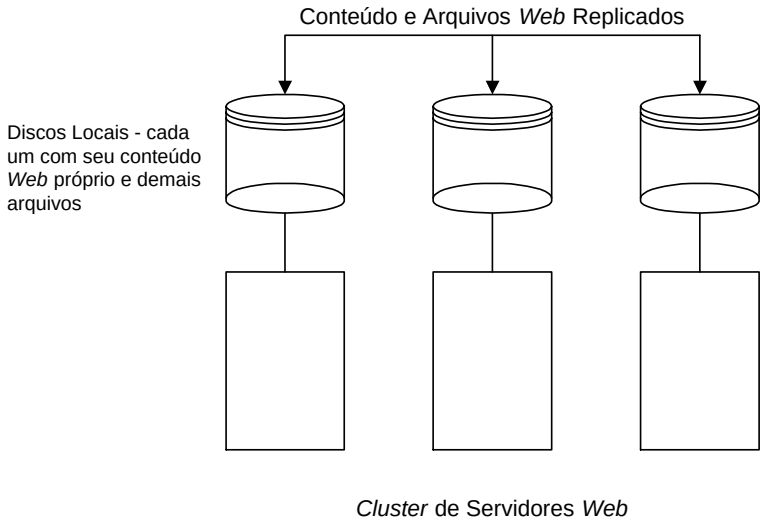

Figura 3.2: *Cluster Web* com armazenamento não compartilhado, adaptado de Polyserve (2000)

Sincronização é o processo de assegurar que os dados estejam consistentes entre todos os servidores no *cluster*. Em alguns sistemas, pode não ser necessário que os dados estejam sincronizados em tempo real. Nesse caso, um processo *batch* que replica e sincroniza os arquivos a cada hora pode ser o suficiente. Mas, nesse caso, durante a hora em que uma mudança é feita e o arquivo ainda tem que ser atualizado em todos os servidores no agregado, os clientes podem ver dados diferentes, dependendo de qual servidor está manipulando as requisições dos clientes.

A figura 3.2 apresenta simplificadamente uma arquitetura de *cluster* de servidores *Web* com o conteúdo necessitando estar consistente nos sistemas de arquivos locais dos seus componentes.

Por outro lado, e especialmente para sistemas de transações em bancos de dados, pode ser necessário que todos os acessos a dados replicados sejam os mesmos durante todo o tempo. Nesse caso, alguns mecanismos de *lock* distribuído deveriam ser empregados para garantir que as informações no banco de dados replicado estejam redundantes e consistentes todo o tempo. O custo de oferecer esse tipo de serviço pode ser alto, e normalmente é garantido somente quando é absolutamente inaceitável oferecer menos que 100% de consistência de dados. Um exemplo dessa necessidade pode ser observado em transações financeiras.

A arquitetura cliente-servidor da *Web* indica que ela não tem uma maneira efi-

ciente de manter os usuários atualizados com as últimas versões das páginas. Os usuários têm que pressionar o botão "atualizar" de seus navegadores para garantir que têm as últimas informações, e os navegadores são forçados a comunicarem-se com os servidores para confirmar se a cópia local de um recurso ainda é válida. Portanto, a arquitetura de sistema da *Web* permite uma semântica de consistência mais "relaxada" em função da disponibilidade e do desempenho.

Uma das técnicas mais usadas para manutenção da consistência e amplamente encontrada na literatura da área é a replicação. Nos ambientes de *Web cluster*, a replicação permite um aumento do desempenho através do balanceamento de carga entre os componentes, proporcionando tolerância a falhas quando um dos servidores deixa de funcionar. Além disso, é desejável que os detalhes da replicação sejam escondidos dos usuários finais, proporcionando transparência de localização, isto é, o fato de o usuário usar uma réplica sem perceber (TANENBAUM, 1995). Transparência é a capacidade de esconder do usuário e do programador da aplicação a separação dos componentes em um sistema distribuído, para que assim o sistema seja percebido como algo inteiro e não como um conjunto de componentes independentes (COULOURIS; DOLLIMORE; KINDBERG, 2001).

3.3 Replicação

A replicação é uma técnica de redundância empregada para melhorar o nível de disponibilidade em sistemas distribuídos. A replicação aumenta a disponibilidade dos dados e processos e o desempenho geral do sistema, já que o tempo para atender a uma requisição será menor se uma réplica mais próxima for usada.

Em sistemas baseados no modelo cliente-servidor, um servidor único pode atender a vários clientes, e um aumento da carga de trabalho no servidor pode acarretar tempos de resposta elevados. Em tais circunstâncias, replicar os dados ou servidores pode aumentar o desempenho. Replicação também pode melhorar a disponibilidade da informação quando os processadores deixam de funcionar ou a rede sofre uma interrupção (AMIR, 1995).

O mínimo necessário para que haja replicação é que as diferentes cópias de um mesmo objeto residam em máquinas independentes umas das outras. Isso permite que se atinjam alguns requisitos necessários à tolerância a falhas: a disponibilidade de uma réplica não pode ser afetada pela disponibilidade das demais réplicas.

A redundância é normalmente introduzida pela replicação de componentes ou de serviços. Apesar de a replicação ser uma idéia intuitiva, rapidamente compreensível, sua implementação é difícil. Replicar um serviço em sistemas distribuídos exige que cada réplica do serviço mantenha um estado consistente. Essa consistência é garantida através de um protocolo de replicação específico (DÉFAGO; SCHIPER; SERGENT, 1998).

Replicação tem sido pesquisada em diversas áreas, especialmente em sistemas distribuídos (especialmente na área de tolerância a falhas) e em bancos de dados por razões de desempenho. Nessas duas áreas, as técnicas e mecanismos usados são similares. Porém alguns mecanismos conceitualmente idênticos, na prática, tornam-se muito diferentes (WIESMANN; PEDONE; SCHIPER, 1999). A replicação de objetos em sistemas distribuídos, normalmente, é utilizada para torná-los mais confiáveis e seguros, pois o sistema pode sobreviver a falhas de uma ou mais cópias do componente replicado.

De acordo com Tanenbaum (1995), há três formas de replicação: replicação explícita de arquivos, replicação "preguiçosa" (*lazy*) e replicação de arquivos usando comunicação de grupo.

O trabalho de Gray et al. (1996) dividiu os protocolos de replicação de bancos de dados usando dois parâmetros: "quando" e "quem". O primeiro parâmetro estabelece quando acontece a propagação, que pode ser "ansiosa" ou "preguiçosa". O segundo parâmetro indica quem pode propagar as atualizações: a cópia primária ou a atualização em todos os lugares. No modo de propagação "ansiosa", as atualizações são propagadas dentro dos limites de uma transação, isto é, o usuário não recebe a notificação de *commit* até que todas as cópias no sistema estejam atualizadas. A abordagem "preguiçosa", por outro lado, atualiza uma cópia local, completa a transação e, somente algum tempo depois, ocorre a propagação. O primeiro método oferece consistência em primeiro lugar, mas existe um custo em termos de *overhead* de mensagens e tempo de resposta. Replicação preguiçosa permite uma grande variedade de otimizações, no entanto, como se permite que as cópias fiquem diferentes, podem ocorrer inconsistências.

Em relação a quem pode propagar as atualizações, o método de cópia primária exige que todas as atualizações sejam feitas primeiro em uma cópia (a cópia primária ou mestra) e então nas outras cópias. Isso simplifica o controle de réplicas ao custo de introduzir um único ponto de falhas e um ponto de contenção em potencial. O método de atualização em todos os lugares permite que qualquer cópia seja atualizada, o que possibilita uma melhora no tempo de acesso, mas ocasiona uma complexidade na coordenação.

Em relação à interação entre os clientes e o servidores, Wiesmann, Pedone e Schiper (1999) estabelece a seguinte classificação de replicações:

- o cliente interage com um servidor específico, chamada de replicação primário-*backup*;
- o cliente interage com qualquer servidor, chamada de replicação multiprimária;
- o cliente interage com todos os servidores, chamada de replicação ativa.

3.3.1 Replicação primário-*backup*

Na abordagem primário-*backup*, um dos servidores, o primário, é o único servidor autorizado a atender as requisições dos clientes. Os outros servidores, chamados de *backups*, atualizam sua cópia da base de dados após o primário informa-los da ação (figura 3.3). Se o primário deixa de funcionar, um dos *backups* assume e torna-se o novo primário (WIESMANN; PEDONE; SCHIPER, 1999).

3.3.2 Replicação multiprimário

Replicação multiprimário é similar à replicação primário-*backup* no sentido em que um cliente envia as operações de transações para somente um servidor de banco de dados, que as executa e repassa a atualização resultante para os outros servidores. No entanto, diferente da replicação passiva, pode haver vários servidores "primários" executando as requisições dos clientes ao mesmo tempo. Dois clientes podem escolher dois servidores diferentes, por exemplo. Já que não existe sincronização durante a execução da transação, deve ser utilizado algum mecanismo para garantir que as execuções concorrentes sejam serializáveis.

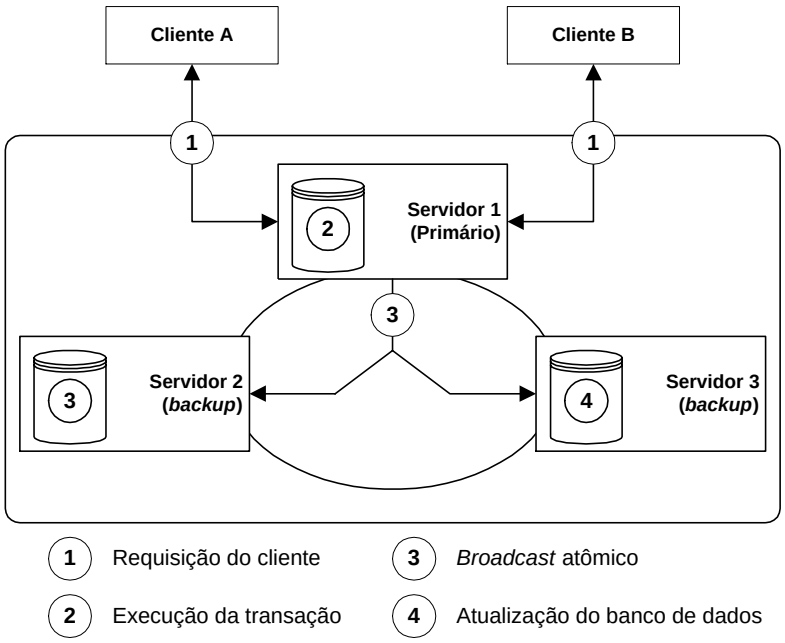

Figura 3.3: Replicação primário-*backup*, traduzido de Wiesmann, Pedone e Schiper (1999)

A figura 3.4 apresenta a comunicação envolvida na replicação multiprimário. O cliente contata qualquer servidor de banco de dados e envia a transação a ser executada. O servidor de banco de dados executa as requisições localmente e, ao final da transação, envia a atualização para os outros membros do grupo, usando uma primitiva de *broadcast* atômico. O mecanismo exigido para garantir a serialização não é discutido aqui.

3.3.3 Replicação ativa

No esquema de replicação ativa, cada cliente interage com os servidores, usando a primitiva de *broadcast* atômico. Cada servidor processa a requisição e responde ao cliente. Uma vez que o cliente receba a primeira resposta, ele sabe que seu pedido foi executado com sucesso. A figura 3.5 demonstra a comunicação entre os clientes e os servidores na abordagem de replicação ativa. O cliente contata diretamente todos os servidores de dados. Cada servidor manipula a requisição (WIESMANN; PEDONE; SCHIPER, 1999).

Replicação ativa, ao contrário da replicação primário-*backup*, é uma técnica simétrica em que cada um dos servidores replicados pode realizar o mesmo conjunto de ações na mesma ordem. Essa técnica necessita que o próximo estado da base de dados seja determinado pelo estado atual e pela próxima ação. Outros fatores, tais como a passagem do tempo, não influem no próximo estado. Algumas arquiteturas de replicação ativa replicam apenas as atualizações, enquanto consultas são replicadas localmente (AMIR, 1995).

Em relação aos algoritmos de escalonamento de propagação de atualizações, (WIESMANN; PEDONE; SCHIPER, 1999) classifica como: replicação pessimista e replicação otimista.

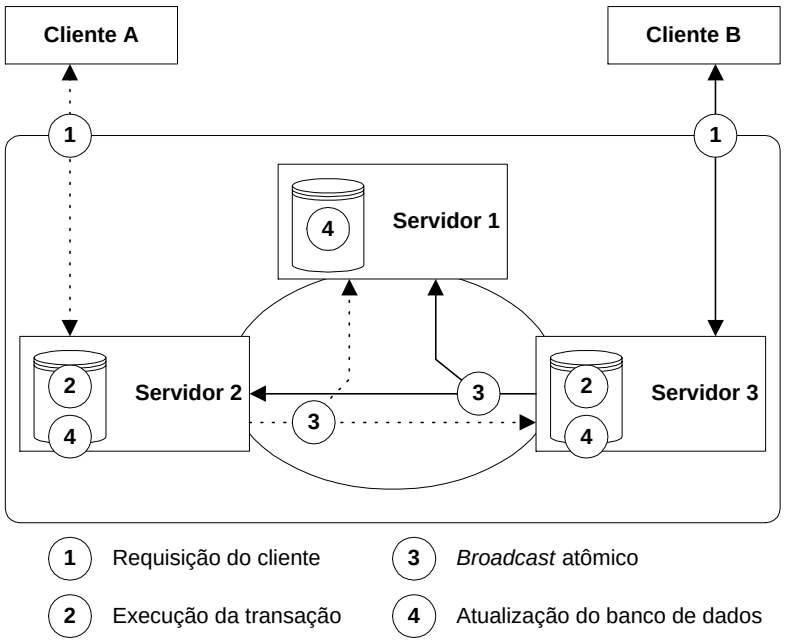

Figura 3.4: Replicação multiprimário, traduzido de Wiesmann, Pedone e Schiper (1999)

3.3.4 Replicação pessimista (conservadora)

Os mecanismos tradicionais de replicação oferecem uma semântica de cópia única, isso é, dão aos usuários a ilusão de ter apenas uma única cópia disponível de um objeto através da manutenção de réplicas idênticas todo o tempo. Esses mecanismos são considerados pessimistas porque proíbem o acesso a uma réplica a menos que o conteúdo dessa réplica esteja atualizado. Esse tipo de semântica é empregado principalmente em aplicações de missão crítica, tais como sistemas bancários, que não podem emitir respostas erradas. Porém um de seus pontos negativos é a exigência de hardware altamente especializado e, por sua vez, caro (GUY et al., 1998).

3.3.5 Replicação otimista

Sistemas otimistas de replicação permitem que qualquer réplica de um objeto seja atualizada a qualquer momento. No entanto, os mecanismos otimistas ganham essa alta disponibilidade sacrificando a consistência. Já que qualquer réplica pode ser atualizada, duas réplicas que não conseguem se comunicar podem ser modificadas independentemente, provocando a existência de conflitos, isto é, conteúdos diferentes nas réplicas. Para manter a consistência, um sistema de replicação otimista deve ter a habilidade de detectar e resolver esses tipos de conflitos. Uma vez detectado o conflito, a resolução de conflitos deve ocorrer antes da atividade normal do conteúdo para que ele seja disponibilizado normalmente.

Replicação otimista adapta-se bem a ambientes distribuídos geograficamente, nos quais a comunicação entre os sítios é lenta e nem sempre confiável.

A seguir, são apresentadas algumas vantagens dos algoritmos otimistas de replicação em relação aos algoritmos pessimistas (SAITO, 2000):

- tolerância a falhas: algoritmos otimistas trabalham sob enlaces lentos de rede

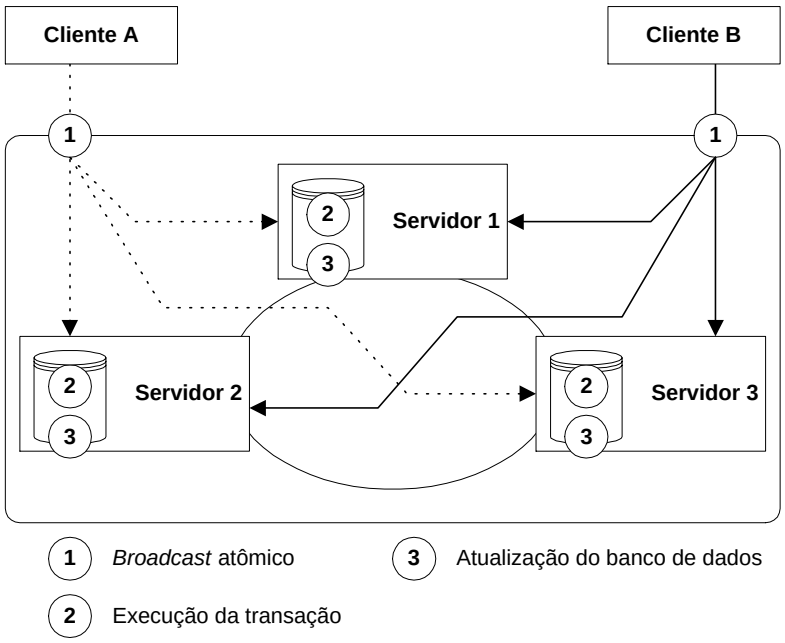

Figura 3.5: Replicação ativa, traduzido de Wiesmann, Pedone e Schiper (1999)

porque eles propagam as atualizações das réplicas em retaguarda. Além disso, eles permitem que os usuários façam operações de escrita ou atualização em um objeto, contanto que a réplica esteja ativa;

- flexibilidade de rede: muitos algoritmos otimistas trabalham sobre enlaces de rede intermitentes ou incompletos ao permitir que as atualizações sejam feitas em qualquer par de nós;

- baixo custo: devido a sua flexibilidade de rede e a sua escalabilidade, serviços replicados através de algoritmos otimistas podem ser implementados utilizando-se hardware não muito caro, comparando-se com o exigido por algoritmos pessimistas;

- autonomia do sítio: algoritmos otimistas aumentam a autonomia do sítio ao necessitar de menor grau de coordenação entre os sítios. Por exemplo, alguns serviços (espelhamento de sítios FTP) permitem que uma réplica seja acrescentada sem qualquer mudança administrativa nas réplicas existentes.

3.3.6 Modelos de comunicação para replicação

Segundo Augustin (2000), existem basicamente dois modelos de comunicação para replicação:

- par-a-par: todas as cópias são iguais, permitindo que qualquer cópia possa comunicar-se com qualquer outra. Cada réplica tem conhecimento de todas as outras, resultando numa grande estrutura de dados replicada (lista com todas as réplicas conhecidas). Portanto, sofre do problema de escalabilidade. Propaga atualizações mais rapidamente usando qualquer conectividade disponível, mas é o modelo mais complexo, tanto na implementação, quanto no estado que alcança;

- cliente-servidor: o cliente armazena um subconjunto dos dados do servidor, as atualizações são enviadas ao servidor, que as propaga para os demais clientes. Este modelo, tradicionalmente, fornece um controle mais simples da replicação. A escala não é tipicamente um problema, pois múltiplos servidores podem ser combinados para atender os clientes. No entanto, aumenta a dependência nos servidores que estão em um nível hierárquico superior. Muitos sistemas somente permitem a comunicação do cliente com o seu servidor *home*. Nesse cenário, a falha do servidor isolará todos os seus clientes.

3.4 Replicação de conteúdo *web*

Pode-se distinguir três formas de replicação de conteúdo em servidores *Web* distribuídos (COOPER; MELVE; TOMLINSON, 2000):

- replicação em lotes: o servidor replicado a ser atualizado é quem inicia a comunicação com um servidor original. A comunicação é estabelecida em intervalos baseados em transações enfileiradas que são agendadas para processamento posterior. As políticas de agendamento variam, mas, normalmente, ocorrem em um determinado intervalo de tempo. Uma vez que a comunicação é estabelecida, conjuntos de dados são copiados para o servidor replicado. Os protocolos mais utilizados são *File Transfer Protocol* (FTP) e Rdist. São muito usados para sincronização de espelhos de sítios *Web* na Internet;

- replicação por demanda: os servidores replicados obtêm o conteúdo quando necessário devido à demanda do cliente. Quando um cliente solicita um recurso que não esteja no conjunto de dados do servidor replicado, é feita uma tentativa de atender ao pedido, buscando o recurso do servidor original e retornando-o para o cliente que o solicitou. Os protocolos mais utilizados são FTP, HTTP e ICP;

- replicação sincronizada: os servidores replicados cooperam usando estratégias sincronizadas e protocolos especializados de réplica para manter os conjuntos de dados replicados coerentes. Estratégias de sincronização variam desde fortemente coerentes (alguns poucos minutos) até fracamente coerentes (algumas horas). As atualizações ocorrem entre as réplicas baseadas nas restrições de tempo do modelo de coerência empregado e são feitas, geralmente, apenas através dos dados que foram modificados. Os dois protocolos abertos de replicação sincronizada mais difundidos são *Andrew File System* (AFS) e Coda. Além desses, há um outro protocolo proprietário, o Novell NRS. Todos os protocolos conhecidos usam métodos de troca com chave criptográfica forte, que são baseados no modelo secreto compartilhado Kerberos ou no modelo de chaves públicas/privadas RSA, desenvolvido por Rivest, Shamir e Adelman.

Uma série de trabalhos publicados recentemente (PIERRE et al., 2001; PIERRE; KUZ; STEEN, 2000; PIERRE; STEEN, 2001; PIERRE; STEEN; TANENBAUM, 2001) defende uma arquitetura de suporte a documentos que se auto-replicam, enfatizando a necessidade de políticas de replicação diferentes para documentos *Web*, não apenas o uso de uma única técnica.

3.5 Trabalhos relacionados

Durante o desenvolvimento deste trabalho, foi possível identificar e estudar alguns trabalhos e ferramentas cujos objetivos encontram-se dentro do escopo desta pesquisa, isto é, sistemas de arquivos distribuídos (InterMezzo e Rumor) e ferramentas de replicação de conteúdo (Rdist, Rsync e Mirrordir). Alguns são mais adaptados a bancos de dados replicados em ambientes de computação móvel nômade. A seguir, são apresentados alguns desses trabalhos.

3.5.1 InterMezzo

InterMezzo é um sistema de arquivos cliente-servidor que mantém réplicas dos dados do sistema de arquivos entre diversas máquinas servidoras. Durante uma falha no servidor ou na rede, InterMezzo grava as atualizações do sistema de arquivos em *journals*, que são usados para fazer com que todos os servidores fiquem consistentes (BRAAM, 1999).

Ao utilizar o sistema InterMezzo, servidores *Web* de alta disponibilidade podem ser mantidos consistentes, até mesmo depois de falhas em um *Web cluster* de alta disponibilidade. A consistência aplica-se tanto aos objetos *Web* estáticos (arquivos HTML e gráficos), quanto a conteúdo dinâmico, tal como carrinhos de compras armazenados no sistema de arquivos.

Diferentemente de outros sistemas de arquivos distribuídos, InterMezzo empacota um sistema de arquivos existente com um módulo do nível do *kernel* chamado Presto. O módulo Presto intercepta as atualizações feitas nos diretórios e arquivos e escreve registros denominados *journals*, que detalham o que foi feito e qual versão do diretório ou do arquivo foi afetada pela atualização.

InterMezzo foi inicialmente desenvolvido na Carnegie Mellon University, como parte do projeto Coda (*http://www.coda.cs.cmu.edu*). O sistema Coda teve grande influência na concepção do InterMezzo, porém não oferece todas as características de Coda, mas é descrito como sendo muito mais modular e fácil de ser integrado com os sistemas de arquivos existentes. Hoje em dia, o sistema InterMezzo está sendo desenvolvido pela empresa Stelias Computing (*http://www.stelias.com*) (CLUSTER FILE SYSTEM, 2002).

São características do InterMezzo:

- encaminhamento e reintegração de atualizações: quando as atualizações nos arquivos e diretórios são feitas, um *journal*, que descreve essas atualizações detalhadamente é automaticamente gerado;

- durante os momentos de conectividade, as atualizações no *journal* são reintegradas a um servidor imediatamente após terem sido feitas. Elas são, então, propagadas aos demais componentes que replicam algumas coleções de diretórios tão logo esses clientes estejam disponíveis na rede;

- suporta operação desconectada durante períodos em que as redes não estão funcionais ou através de desconexões voluntárias de computadores móveis;

- os servidores InterMezzo encarregam-se de repassar as atualizações aos clientes que tenham sido registrados nos servidores como replicadores de coleções de diretórios.

3.5.2 Rumor

Rumor é um sistema otimista de arquivos replicados projetado para uso em computadores móveis. Rumor usa um modelo par-a-par que permite propagação otimista de atualizações entre quaisquer sítios com arquivos replicados (GUY et al., 1998).

Rumor é um serviço portável de arquivos replicados, construído como um pacote sobre um sistema de arquivos existente. A portabilidade do Rumor é obtida através da divisão da funcionalidade do sistema em um sistema independente maior de replicação e um serviço menor dependente do sistema. Uma vez que o Rumor é um sistema projetado para o nível do usuário, ele pode ser instalado e mantido por usuários comuns. Não são exigidos privilégios de superusuário.

Rumor usa algoritmos e estruturas de dados projetados originalmente para o sistema de arquivos replicados Ficus (GUY et al., 1990).

A estratégia de replicação otimista do Rumor permite que atualizações concorrentes possam levar a inconsistências de réplicas de um arquivo único. Rumor detecta esse tipo de inconsistências e aplica mecanismos automatizados para resolvê-las.

O pacote de instalação e o código fonte da ferramenta Rumor podem ser obtidos a partir do sítio de distribuição (*http://ficus-www.cs.ucla.edu/rumor/*). A distribuição é compatível com as plataformas Linux e FreeBSD. A licença de uso permite a modificação e a redistribuição da ferramenta, contanto que seja mantida a referência à Universidade da Califórnia.

3.5.3 Rdist

É um programa cujo objetivo é manter cópias idênticas de arquivos em várias estações. Ele preserva o proprietário, o grupo, o modo e a data/hora dos arquivos. Rdist lê comandos de um arquivo *distfile*, usado como configuração de diretórios a serem replicados. A forma de acessar o servidor remoto é através dos protocolos *remote shell* (*rsh*) ou *rcmd*. A escolha do protocolo é feita no momento de sua compilação.

O uso da ferramenta rdist combinada com o protocolo *rsh* não é considerado seguro devido às falhas apresentadas por esse protocolo. O servidor que vai enviar as cópias deve estar presente no arquivo de configuração *hosts.allow* da máquina destino. Se um invasor de sistemas de rede usar uma ferramenta de *ip spoofing*, ele pode enganar a máquina remetente e fazer-se passar por ela.

Rdist implementa a replicação mestre-escravo, isso é, as atualizações somente podem acontecer no servidor mestre, que as propaga para os demais (escravos).

Rdist é classificada como ferramenta para replicação por lotes. Nesse modelo, o servidor replicado que precisa ser atualizado irá iniciar a comunicação com o servidor mestre. A comunicação é estabelecida em intervalos baseados nas transações enfileiradas que são atrasadas para processamento. A política de escalonamento pode ser baseada em vários fatores, mas elas têm que ocorrer sobre períodos de tempo. Uma vez que a comunicação seja estabelecida, os conjuntos de dados são copiados para o servidor *Web* replicado que iniciou a comunicação.

A versão da ferramenta que foi utilizada como base para essa descrição foi a de número 6.1.5. Um artigo publicado no sítio de notícias Linux Gazette (BAVENDIEK, 1998) indica a ferramenta rdist para sincronização de arquivos entre compu-

tadores *desktop* e *notebooks* com sistema operacional Linux. Conclui-se que em um ambiente controlado, com reduzidos problemas de segurança, seja uma ferramenta adequada.

Rdist foi desenvolvida por Michael A. Cooper, da Universidade da Califórnia. Rdist pode ser obtida no sítio *http://www.magnicomp.com/rdist/*. A sua licença de uso permite a modificação e a redistribuição da ferramenta, contanto que seja mantida a referência à empresa MagniComp (*http://www.magnicomp.com*).

3.5.4 Rsync

Ferramenta desenvolvida por Andrew Tridgell e Paul Mackerras, com o objetivo de ser uma substituta mais rápida e mais flexível ao comando *rcp*. A ferramenta é baseada no algoritmo rsync, usado para identificar os trechos que são diferentes entre dois arquivos que foram atualizados independentemente.

O algoritmo rsync processa quais partes de um arquivo fonte combinam com alguma parte de um arquivo de destino. Essas partes não precisam ser enviadas pela rede. É necessária somente uma referência à parte do arquivo destino. Somente partes do arquivo fonte que não combinam precisam ser enviadas via rede. O receptor pode então construir uma cópia do arquivo fonte, usando as referências às partes existentes do arquivo destino e o material recebido via rede.

A ferramenta rsync usa o protocolo *rsh* como meio de transporte.

Algumas características da ferramenta rsync:

- suporte a cópia de *links*, dispositivos, proprietários, grupos e permissões;
- opções *exclude* e *exclude-from*, similar à ferramenta GNU *tar*;
- um modo de exclusão CVS para ignorar os mesmos arquivos que o sistema CVS ignoraria;
- possibilidade de usar *shell* remoto transparente, incluindo *rsh* ou *secure shell* (*ssh*);
- não exigência de privilégios de superusuário (*root*);
- criação de um *pipelining* para a transferência dos arquivos a fim de minimizar custos de latência;
- suporte a servidores anônimos ou autenticados.

A ferramenta rsync pode ser configurada para ser executada como um processo servidor. Sua licença é a GNU *General Public License* (GPL). Existem versões da ferramenta para várias versões do sistema operacional Unix, e ela pode ser encontrada em *http://rsync.samba.org*. Um relatório técnico mais detalhado pode ser obtido em Tridgell e Mackerras (1996).

3.5.5 Mirrordir

É composta de vários módulos: *Forward*, *Mirrordir* e *Pslogin*. O módulo *Pslogin* faz autenticação segura sobre TCP/IP, usando trocas de chaves *diffie-hellman* e encriptação forte. Pode ser usada como uma alternativa ao protocolo ssh. *Forward* envia *sockets* TCP arbitrários através de um canal seguro, com o objetivo de criar serviços seguros a partir de serviços comuns. *Mirrordir* faz o espelhamento de uma

árvore de diretórios em todos os seus detalhes, incluindo dispositivos, propriedade, permissões, *links* simbólicos e horas de acesso, ajustável para *backups* programados de discos. Opcionalmente, pode criar cópias de arquivos antes da remoção e armazenar várias revisões através de vários níveis. Também implementa seus próprios *sockets* seguros para transferência de arquivos com encriptação forte.

A ferramenta mirrordir foi desenvolvida por Paul Sheer e James R. Van Zandt e é distribuída através da licença GPL.

3.6 Considerações finais

Este capítulo apresentou conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos. Foram apresentados detalhes sobre a arquitetura de servidores *Web* distribuídos, manutenção da consistência em ambientes de servidores *Web* distribuídos, uso de replicação e formas de replicação. Ao final, foram citados alguns trabalhos relacionados ao propósito de manter réplicas consistentes em ambientes de servidores *Web* distribuídos.

4 ORPIS: CONCEPÇÃO E MODELAGEM

Este capítulo apresenta o modelo de sincronização de réplicas de conteúdo em *clusters Web*. O modelo é denominado ORPIS (***O**ne **R**eplication **P**rotocol for **I**nternet **S**ervers*). Sua principal característica é a manutenção da consistência das réplicas do conteúdo em um ambiente de servidor *Web* distribuído geograficamente. São elencados os pressupostos do modelo e descritos os seus componentes. Os algoritmos de funcionamento do modelo são apresentados.

Um fato que deve ser mencionado é a compatibilidade do modelo ORPIS aos servidores *Web* existentes, sem a necessidade de modificação em suas configurações. Essa característica também se aplica aos editores e atualizadores dos diretórios do servidor *Web*. Nesse sentido, um dos objetivos do modelo ORPIS é ser transparente.

Um *Web cluster* é visto como uma solução que minimiza custos, pois proporciona maior desempenho, utilizando-se equipamentos menos exigentes em termos de hardware. Partindo dessa premissa, o modelo ORPIS prevê a utilização de tecnologia de software livre, criando uma solução economicamente viável e acessível. A disponibilização via *Web* do código fonte da implementação do modelo permite o surgimento de novas características ou versões da ferramenta, através das contribuições de terceiros.

4.1 Pressupostos

O modelo ORPIS foi concebido a partir dos seguintes pressupostos:

- existe a necessidade de se manter a consistência do conteúdo replicado em um ambiente de servidor *Web* distribuído geograficamente;
- a utilização de serviços Internet distribuídos (ex. WWW, FTP, E-mail) é uma forma de proporcionar o espalhamento de carga entre diversos componentes. Além desse fato, existe a possibilidade de esse serviço ser tolerante a falhas caso um dos seus servidores interrompa seu funcionamento. Nesse caso, as requisições são desviadas para um dos demais componentes do serviço distribuído;
- o uso de software livre na implementação do modelo permite o rápido acesso às ferramentas necessárias: sistema operacional, linguagens e gerenciador de banco de dados;
- a disponibilização via *Web* do código fonte da implementação do modelo permite o surgimento de novas características ou versões da ferramenta, através das contribuições de terceiros.

4.2 Descrição do modelo

ORPIS é um modelo de replicação de conteúdo para ser utilizado em servidores *Web* geograficamente distribuídos (*clusters Web* distribuídos), os quais têm o objetivo de oferecer mais desempenho no atendimento às requisições, principalmente em situações de sobrecarga. Um módulo anexado ao servidor *Web* encarrega-se de desviar os pedidos para o servidor mais adequado para atender às requisições. O conteúdo (objetos *Web*) é replicado entre os diversos servidores componentes do *Web cluster* distribuído ("comunidade ORPIS"). Um módulo encarrega-se de detectar as atualizações feitas nos objetos *Web*, baseando-se em políticas de intervalos de tempo. Tendo detectado as atualizações feitas, inicia-se o processo de sincronização entre as réplicas.

O modelo ORPIS se baseia na forma de replicação descrita como multiprimária, pois permite que o cliente envie as requisições para qualquer membro do *cluster*. Essa forma de replicação permite que qualquer uma das réplicas existentes possa ser eleita como a primária. As atualizações no conteúdo do sítio *Web* também podem ser feitas em qualquer servidor. A não existência de um servidor central permite essa liberdade. Discussões mais aprofundadas sobre esse tema são tratadas em Wiesmann, Pedone e Schiper (1999).

O modelo funciona com o conceito de **proprietários** de páginas. O conteúdo total do *Web cluster* é replicado em todos os componentes do *cluster*, porém a atualização de um objeto nas páginas somente é autorizada pelo seu proprietário, tendo ele feito a atualização em qualquer uma das réplicas. Não é permitido que o proprietário de uma página atualize páginas pertencentes a outro proprietário. Esse pressuposto impede a existência de conflitos entre as réplicas. Uma característica que o modelo não implementa é o tratamento de duas atualizações concorrentes feitas pelo mesmo proprietário.

O servidor *Web* que tem instalado o componente Gerente de Replicação do modelo ORPIS, ao detectar uma atualização em seu sistema de arquivos, passa, então, a ser o primário, responsável pela atualização dos demais membros, enviando mensagens de atualizações através do mecanismo de transporte de dados. A troca de informações a respeito de atualizações do conteúdo é realizada entre as réplicas através de trocas de mensagens do servidor primário com as réplicas *backups*, até que todos os servidores estejam consistentes. Os servidores primários são chamados de nós propagadores, e os servidores *backups* são denominados nós receptores.

No modelo ORPIS, o conteúdo é totalmente replicado entre os componentes da comunidade, cada réplica funcionando como um "espelho" das demais.

O modelo ORPIS usa uma estratégia de propagação otimista porque não existe sincronismo no envio das atualizações para as réplicas. Os nós propagadores enviam as mensagens de que existem objetos que foram atualizados e os nós receptores somente buscarão tais objetos quando julgarem ser um momento ideal, em função das condições de conectividade ou da carga de requisições que estiverem recebendo. A busca por tais objetos acontece através do uso do protocolo FTP.

4.3 Componentes do modelo

A figura 4.1 apresenta os principais componentes do modelo ORPIS. Como se pode notar, os componentes do modelo ORPIS são executados no nível do usuário,

sem necessidade de reconfiguração do sistema operacional ou do servidor *Web*.

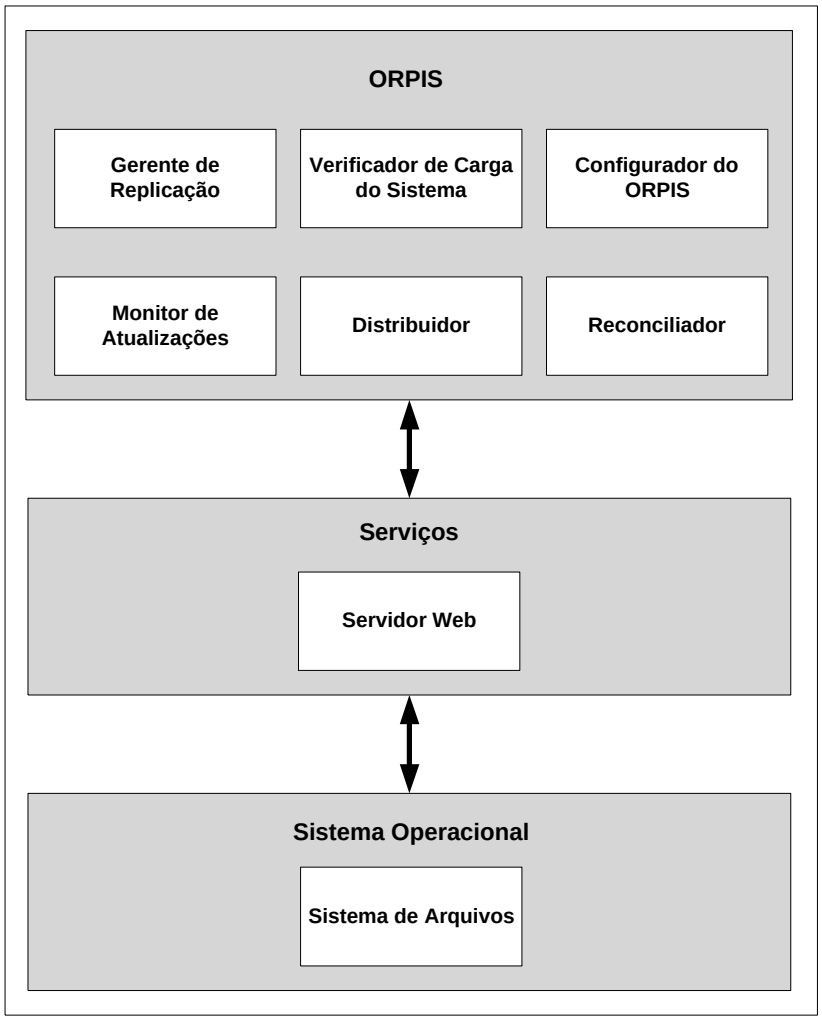

Figura 4.1: Componentes do modelo ORPIS

As seções seguintes descrevem os vários módulos que compõem o modelo ORPIS.

4.3.1 Verificador de carga do sistema

Esse módulo calcula a carga de requisições que atualmente está sendo imposta ao servidor em um determinado momento. Sua concepção foi baseada em Killelea (1998) e Abdelzaher e Bhatti (1999).

Uma forma simples de decidir se o servidor está sobrecarregado é através da monitoração do tempo de resposta das requisições ao servidor. Esse tempo de resposta é obtido através do envio de requisições HTTP (figura 4.2) ao servidor e da medição do tempo de resposta às mesmas. A maneira que permite uma melhor medição é o envio das requisições HTTP a partir de uma sessão no próprio computador que possui o servidor *Web*. Dessa forma, não existe o tempo necessário para que os pacotes circulem pela rede. Um tempo de resposta pequeno pode indicar que o servidor não está sobrecarregado.

De acordo com Abdelzaher e Bhatti (1999), deve-se encontrar um ponto de *threshold*, que é estabelecido através de um acordo de QoS (*Quality of Service*). Na ausência desse tipo de medida, o ponto de *threshold* (T) é obtido através do tamanho

máximo da fila de requisições (Q) e do tempo médio de resposta (S). Se for estabelecido, por exemplo, que 75% de ocupação da fila é um indicativo de sobrecarga, o Verificador de Carga do Sistema deve ser ajustado para considerar o servidor sobrecarregado quando a fila de requisições estiver em 75% de sua capacidade ($T = 0.75QS$). Pode-se obter o parâmetro Tempo Médio de Resposta através de testes de desempenho do servidor.

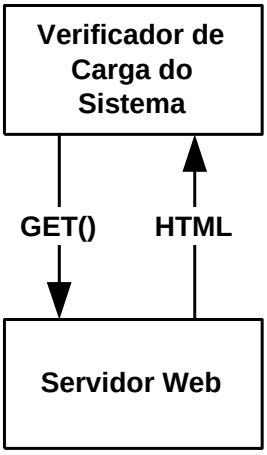

Figura 4.2: Funcionamento do módulo verificador de carga do sistema

Esse módulo somente é acionado pelo Gerente de Replicação quando do início do processo de sincronização, que o utiliza como forma de detectar a carga de trabalho que está sendo imposta no servidor *Web*. A figura 4.3 apresenta o algoritmo de funcionamento do Verificador de Carga do Sistema.

```
início
   iteracoes = 0;
   enquanto iteracoes < k
    submete método HTTP GET ao servidor Web;
    chama método medeTempoResposta();
    iteracoes = iteracoes + 1;
   fim-enquanto
   calcula tempo;
   se tempo > limiar
     status = carregado;
   senão
     status = leve;
   fim-se
   informa estado do servidor Web;
fim
```

Figura 4.3: Algoritmo do módulo verificador de carga do sistema

4.3.2 Monitor de atualizações

Esse módulo faz varredura passiva no sistema de arquivos indicado e tem como finalidade o monitoramento de inclusões, exclusões e modificações feitas em diretórios, arquivos ou sistemas de arquivos completos. Varredura passiva identifica objetos

atualizados através da comparação do estado atual de um dado replicado e um estado conhecido anteriormente (EKENSTAM et al., 2001). A concepção deste módulo baseou-se na ferramenta Tripwire (KIM; SPAFFORD, 1994).

O Monitor de Atualizações possui dois modos de execução, indicados através de parâmetros quando de sua invocação:

- criação da Base de Dados do Sistema de Arquivos – varre o sistema de arquivos e diretórios indicado em um arquivo de configuração e constrói a Base de Dados do Sistema de Arquivos (figura 4.4);
- comparação da Base de Dados do Sistema de Arquivos com o Sistema de Arquivos – neste modo, o Monitor de Atualizações varre o Sistema de Arquivos e compara as informações obtidas com a Base de Dados do Sistema de Arquivos, criada na última vez em que o Monitor de Atualizações foi executado no modo de criação (figura 4.5).

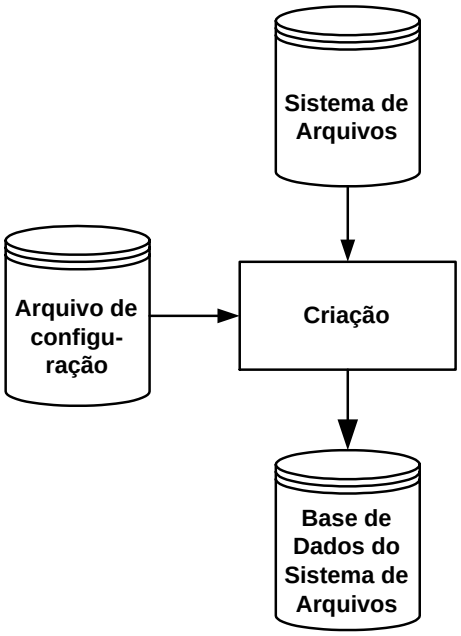

Figura 4.4: Funcionamento do módulo monitor de atualizações no modo de criação

A figura 4.6 apresenta o algoritmo de funcionamento do módulo Monitor de Atualizações.

O Monitor de Atualizações possui como entrada um arquivo de configuração que indica quais diretórios e arquivos que devem ser monitorados, conforme pode ser observado através do exemplo da figura 4.7. Nesse exemplo, foi indicado que o diretório a ser monitorado se chama `Ccl`, que está sob o diretório `/mnt/hda1`.

4.3.3 Distribuidor

É o módulo responsável pelo contato do nó propagador com o módulo Reconciliador no nó remoto. No momento da sincronização, o Distribuidor do nó propagador estabelece a comunicação com o Reconciliador do nó remoto. O Distribuidor envia a lista de atualizações para o Reconciliador do nó receptor. A figura 4.8 apresenta o algoritmo de funcionamento do módulo Distribuidor.

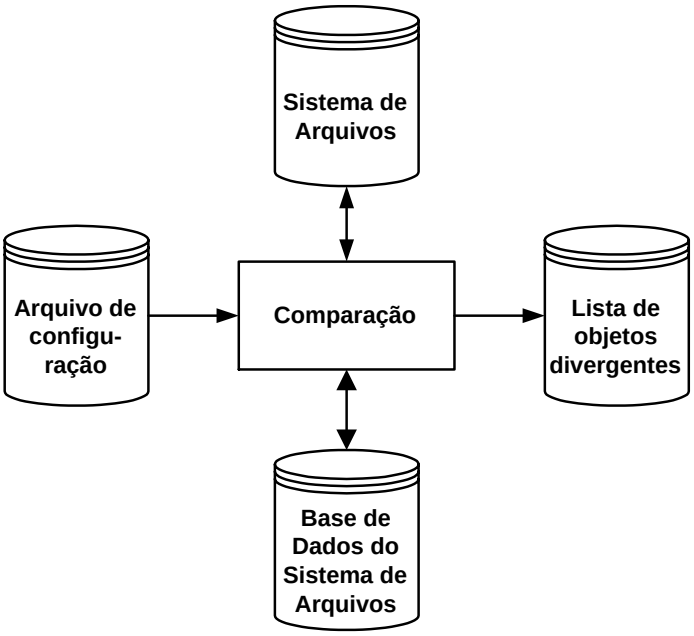

Figura 4.5: Funcionamento do módulo monitor de atualizações no modo de comparação

```
início
   lê arquivo de configuração;
   lê parâmetro da linha de comando;
   se parâmetro = "criação" então
      enquanto existir arquivos a serem varridos;
         lê sistema de arquivos;
         gera base de dados do sistema de arquivos;
      fim-enquanto
   senão
      enquanto existir arquivos a serem varridos;
       lê objeto do sistema de arquivos;
       lê objeto da base do sistema de arquivos;
         se objeto não existir na base de arquivos
            inclui objeto na base de arquivos;
            informa ao Gerente de Replicação;
            volta para leitura de mais objetos;
         fim-se
         compara objetos;
         se objetos iguais
            volta para leitura de mais objetos;
       senão
            altera objeto na base de arquivos;
            informa ao Gerente de Replicação;
            volta para leitura de mais objetos;
         fim-se
      fim-enquanto
   fim-se
fim
```

Figura 4.6: Algoritmo do módulo monitor de atualizações

```
# monitor.cfg
# Diretorios a serem monitorados sao mostrados a seguir. Varias
# entradas podem ser usadas atraves do seguinte formato 'palavra-
# chave=variavel':
# [Diretorio=(caminho/nome)]
# [Diretorio=(caminho/nome)]
# ...
# Se quiser monitoramento recursivo de diretorios, coloque um / no
# final do nome do diretorio, senao o Verificador vai interpretar a
# entrada como um unico arquivo ou unico diretorio a ser monitorado.
#
Diretorio = /mnt/hda1/Ccl/

# Os arquivos da base de dados sao armazenados no diretorio "DataBase"
# que eh definido a seguir.
#
DataBase        = /home/verificador/ccl.dbf

# Usado para determinar tipos de arquivos (Unix e Linux)
TipoDeArquivo = /bin/file

#
# Fim do arquivo monitor.cfg
#
```

Figura 4.7: Exemplo de arquivo de configuração do monitor de atualizações

```
início
   enquanto existirem Componentes do Cluster
   faça
      conectar Reconciliador;
      se conexão estabelecida
         enviar objetos da fila de objetos a propagar;
      senão
         tentar mais tarde;
      fim-se
   fim-enquanto
fim
```

Figura 4.8: Algoritmo do módulo distribuidor

4.3.4 Gerente de replicação

O processo de sincronização é iniciado pelo Gerente de Replicação, em momentos pré-determinados, ou manualmente, através da requisição do usuário. Ele tem a função de acionar o Verificador de Carga. Se o servidor não estiver sobrecarregado, o Gerente de Replicação inicia o Monitor de Atualizações, que faz a varredura do sistema de arquivos. Se há atualizações a serem propagadas, o Gerente de Replicação tem a função de incluí-las na Fila de Objetos a Propagar. A figura 4.9 apresenta o algoritmo de funcionamento do Gerente de Replicação.

```
início
   lê número de tentativas;
   se número de tentativas <> 3
      inicia Verificador de Carga;
      recebe status do servidor Web;
      se status = carregado
         tentativas = tentativas + 1;
         encerra Gerente;
      fim-se
   senão
      tentativas = 0;
      grava número de tentativas;
      inicia Monitor de Atualizações;
      se existem atualizações detectadas pelo Monitor de Atualizações;
         inclui as atualizações na Fila de Objetos a Propagar;
      senão
         encerra Gerente;
      fim-se
   fim-se
fim
```

Figura 4.9: Algoritmo do módulo gerente de replicação

4.3.5 Reconciliador

É o módulo responsável pela comunicação nos nós receptores. Tem por objetivo enviar respostas de conexão e recepção de listas de atualizações, feitas pelo módulo Distribuidor dos nós propagadores. A figura 4.10 apresenta o algoritmo do módulo Reconciliador.

```
início
      conectar Distribuidor;
      se conexão estabelecida
         receber objetos;
         enviar objetos para fila de objetos a propagar;
      senão
         tentar mais tarde;
      fim-se
fim
```

Figura 4.10: Algoritmo do módulo reconciliador

4.3.6 Configurador do ORPIS

Constitui-se de uma aplicação que permite incluir, excluir e alterar endereços de servidores *Web* que fazem parte da "comunidade ORPIS". É uma aplicação cliente de um banco de dados servidor (figura 4.11). Quando os dados do banco são alterados, o próprio mecanismo de detecção do modelo ORPIS encarrega-se de propagar as atualizações aos demais componentes do *cluster*.

Suas funções permitem definir:

- os servidores *Web* envolvidos;

- o tempo de operação do reconciliador (operações de sincronização);
- quais diretórios são replicados e em quais máquinas do *Web cluster* ocorreria a replicação dos mesmos.

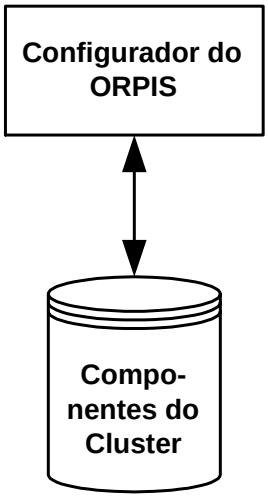

Figura 4.11: Funcionamento do módulo configurador do ORPIS

4.4 Funcionalidade do modelo – nó propagador (algoritmo)

Esta seção apresenta a descrição do funcionamento do modelo ORPIS em um servidor que inicia o processo de manutenção da consistência, o denominado nó propagador (primário).

A figura 4.12 apresenta o diagrama de relacionamentos entre os módulos componentes do modelo ORPIS funcionando em um nó propagador.

O Gerente de Replicação é acionado em intervalos regulares, de acordo com uma programação do sistema *cron* da máquina onde está instalado. O Gerente de Replicação inicia o Monitor de Atualizações, que tem como finalidade o monitoramento de inclusões, exclusões e modificações feitas em diretórios, arquivos ou sistemas de arquivos completos.

O Monitor de Atualizações varre o sistema de arquivos em busca de alterações desde a última varredura. Se encontrar atualizações, gera uma lista com os objetos a serem propagados para as demais réplicas. Se não houve atualizações, envia uma mensagem para o Gerente de Replicação informando-o do fato. Esse processo pode ser observado através da figura 4.13, que apresenta um diagrama de seqüência baseado no modelo *Unified Modeling Language* (UML).

A lista gerada pelo Monitor de Atualizações é manipulada pelo Gerente de Replicação, que gera uma lista maior, concatenando o componente do *cluster* e os detalhes dos objetos a serem propagados. Pode-se exemplificar através de uma situação hipotética em que existem vinte objetos a serem propagados para dez nós receptores. Nesse caso, a lista resultante terá duzentos elementos. Esta lista, resultante da concatenação feita pelo Gerente de Replicação, é incluída na Fila de Objetos a Propagar.

Então, o Gerente de Replicação inicia o Verificador de Carga do Sistema. Para determinar se o servidor *Web* está sobrecarregado, o Verificador de Carga do Sistema

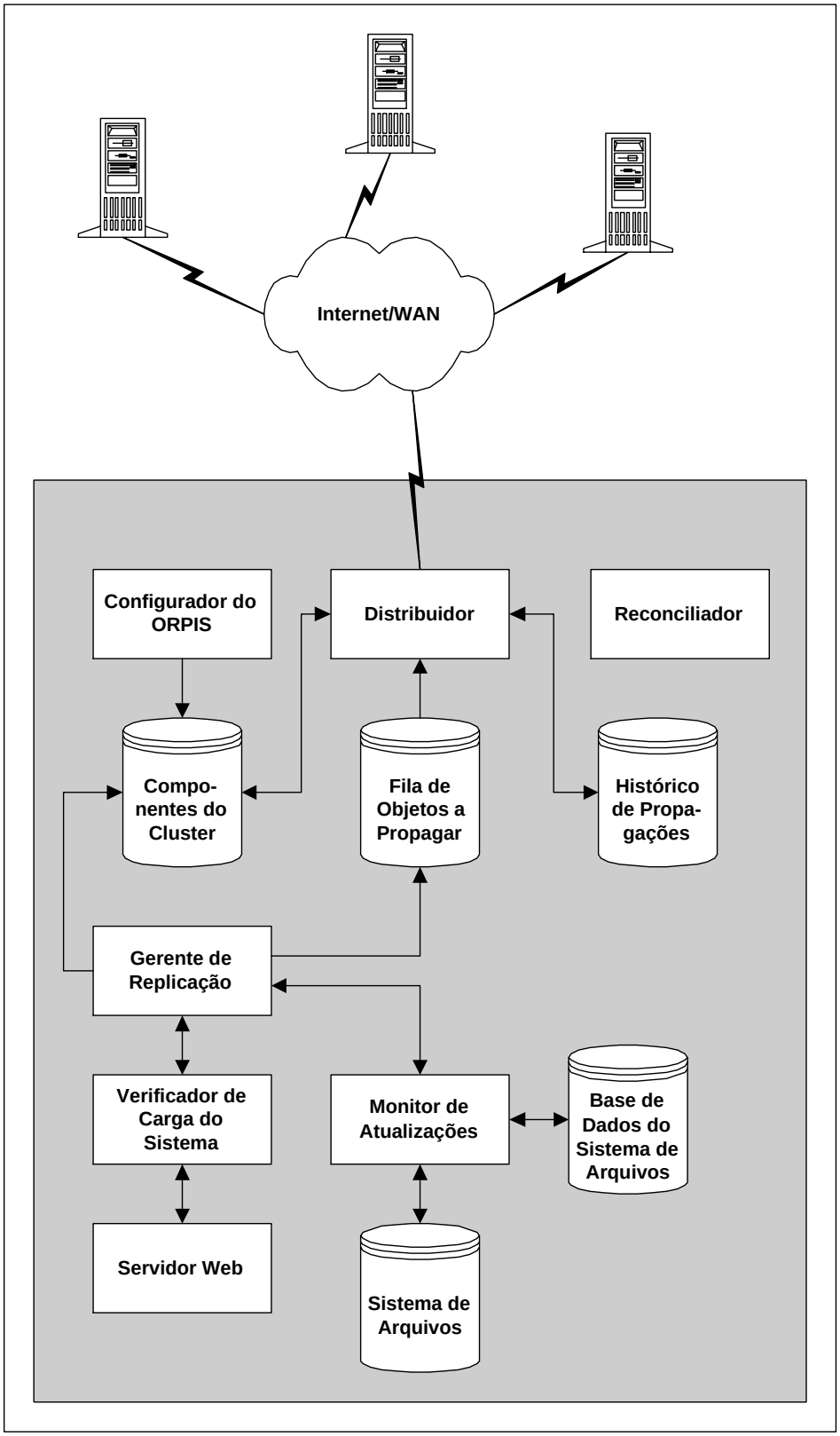

Figura 4.12: Arquitetura do modelo ORPIS em um nó propagador

envia cinqüenta requisições HTTP para o servidor *Web*. Essas cinqüenta requisições são cronometradas, estabelecendo-se uma média dos tempos obtidos. Se o servidor *Web* obtiver uma média elevada, o Verificador de Carga do Sistema declara o servidor

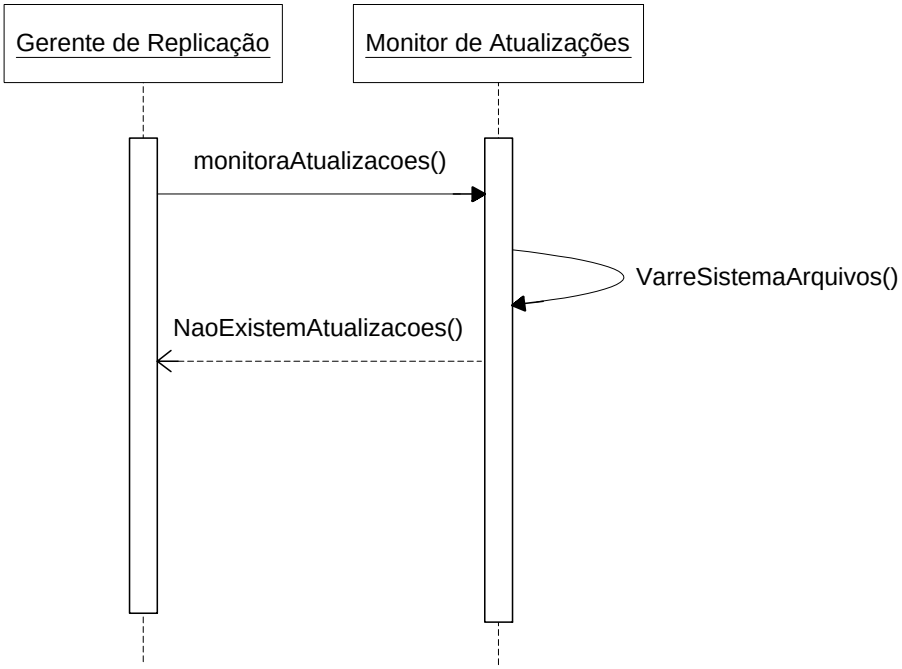

Figura 4.13: Não existem atualizações a serem propagadas

como sobrecarregado e envia um valor verdadeiro (*true*) ao Gerente de Replicação indicando que, no momento, não é possível iniciar o processo de sincronização de servidores. O processo de sincronização não continua, conforme demonstra a figura 4.14. Após a terceira tentativa de se iniciar uma propagação que foi interrompida devido à sobrecarga do servidor *Web*, a propagação é feita sem o teste efetuado pelo Verificador de Carga do Sistema.

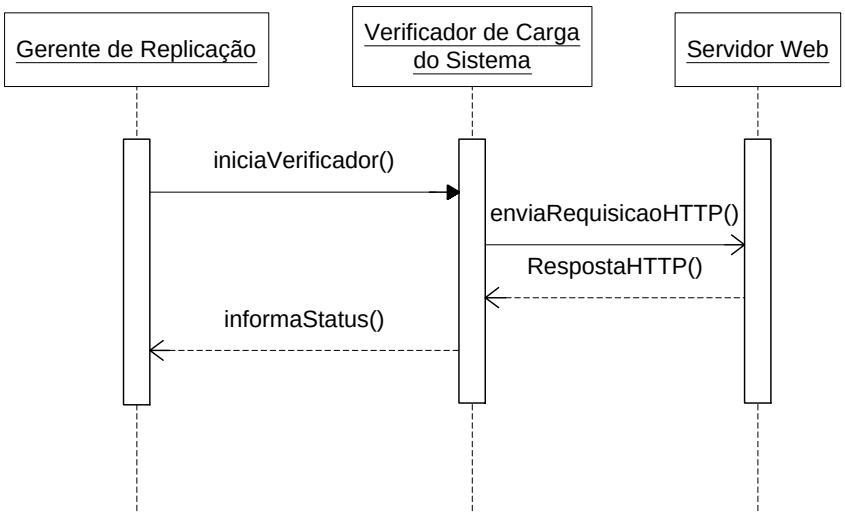

Figura 4.14: Processo de detecção de sobrecarga no servidor

Se o servidor *Web* não estiver sobrecarregado, o Verificador de Carga do Sistema retorna um valor falso (*false*) ao Gerente de Replicação, indicando que o servidor *Web* não está sobrecarregado e que o processo de sincronização pode continuar sua execução.

Então o Gerente de Replicação invoca o módulo Distribuidor. Esse módulo é

responsável pela propagação dos objetos nos nós propagadores e pela recepção das propagações nos nós receptores.

O módulo Distribuidor envia mensagens de início de propagação para todos os componentes do *cluster* e, logo após, aguarda as respostas de ACK (aceitação) por parte dos nós receptores (figura 4.15). Se algum nó não responder em um determinado intervalo de tempo, o Distribuidor atualiza a base de dados Componentes do *Cluster*, informando que aquele componente do *cluster* não foi contatado. O Distribuidor tenta contato com esse nó novamente por mais três vezes. Após essas três tentativas, esse componente é classificado como indisponível, e a base de dados Componentes do *Cluster* é atualizada.

Após ter recebido as respostas de ACK dos nós receptores, o Distribuidor passa a propagar todas as referências aos objetos a serem replicados, a partir da Fila de Objetos a Propagar. A cada resposta de ACK dos nós receptores, o Distribuidor apaga a linha com a referência correspondente ao objeto da Fila de Objetos a Propagar.

O módulo Configurador do ORPIS é composto de uma interface *Web*, que permite incluir, excluir e alterar endereços de servidores *Web* que fazem parte da "comunidade ORPIS". No modelo ORPIS, o diretório que abriga os arquivos que contêm as definições dos componentes do *cluster* também é verificado pelo Monitor de Atualizações. Todas as atualizações feitas em qualquer uma das máquinas da "comunidade ORPIS" serão propagadas para as demais.

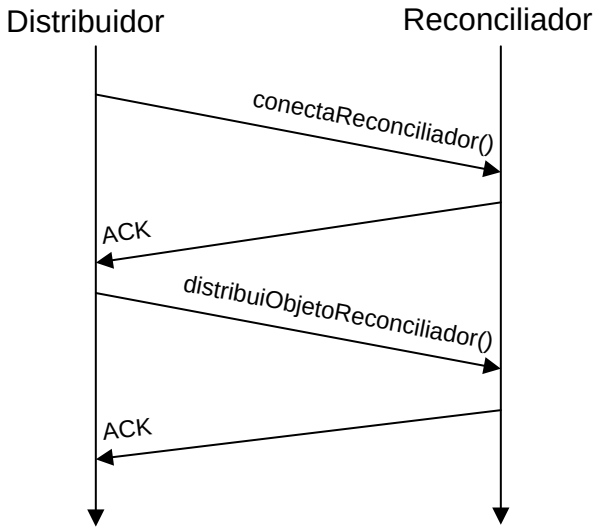

Figura 4.15: Troca de mensagens entre os módulos distribuidor e reconciliador

O Distribuidor mantém uma base de dados Histórico de Propagações, com o objetivo de manter um histórico de todas as operações realizadas, como pode ser observado na figura 4.16. Esse Histórico de Propagações é utilizado com o objetivo de manter a consistência de um servidor que porventura venha a ficar demasiado tempo fora da "comunidade ORPIS", por falta de conectividade da rede ou por falhas de software ou hardware. Quando uma nova máquina é adicionada à "comunidade ORPIS", o Histórico de Propagações permite que a máquina seja suprida com todos os objetos que compõem as páginas *Web* no *cluster*.

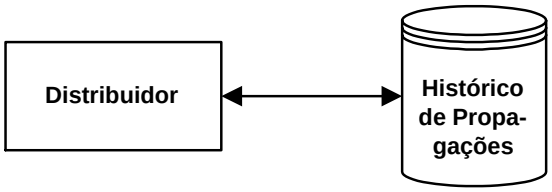

Figura 4.16: Distribuidor mantém histórico das propagações efetuadas

4.5 Funcionalidade do modelo – nós receptores (algoritmo)

Esta seção apresenta a descrição do funcionamento do ORPIS em um servidor receptor (*backup*), que é atualizado a partir do servidor propagador (primário).

A figura 4.17 apresenta a arquitetura do modelo ORPIS, com indicações de controle e fluxo de dados, envolvendo a propagação e a recepção em dois nós.

O módulo Reconciliador de um nó receptor recebe a mensagem de que um nó propagador está iniciando o processo de propagação. Imediatamente, o Reconciliador do nó receptor suspende qualquer atividade de propagação, informando ao Gerente de Replicação do nó receptor que não inicie qualquer processo de atualização até que o processo que foi iniciado pelo nó propagador esteja encerrado. O Reconciliador envia uma resposta ao Distribuidor do nó propagador, que solicitou o início da propagação, informando que está pronto para receber as propagações. Somente após o processo de sincronização estar encerrado, o Reconciliador envia uma mensagem ao Gerente de Replicação para que retome suas atividades de propagação aos demais nós.

O Reconciliador recebe várias mensagens contendo as informações necessárias para que seu sistema de arquivos seja sincronizado com o do nó propagador (referências). Cada uma dessas mensagens contendo todos os detalhes dos objetos a serem recuperados é armazenada na Fila de Objetos a Propagar, que mantém os endereços do remetente e do destinatário dos objetos. O Distribuidor do nó propagador envia uma mensagem final informando que não há mais objetos a serem propagados.

A figura 4.18 apresenta um diagrama de seqüência com o processo completo de verificação de carga, monitoração de atualizações, contato com um reconciliador remoto e gravação do objeto no sistema de arquivos remoto.

4.6 Considerações finais

Este capítulo apresentou o modelo de sincronização de réplicas de conteúdo em *clusters Web*, denominado ORPIS. Sua principal característica é a manutenção da consistência das réplicas do conteúdo em um ambiente de servidor *Web* distribuído geograficamente. Foram elencados os pressupostos do modelo e descritos os seus componentes. Os algoritmos de funcionamento do modelo também foram apresentados.

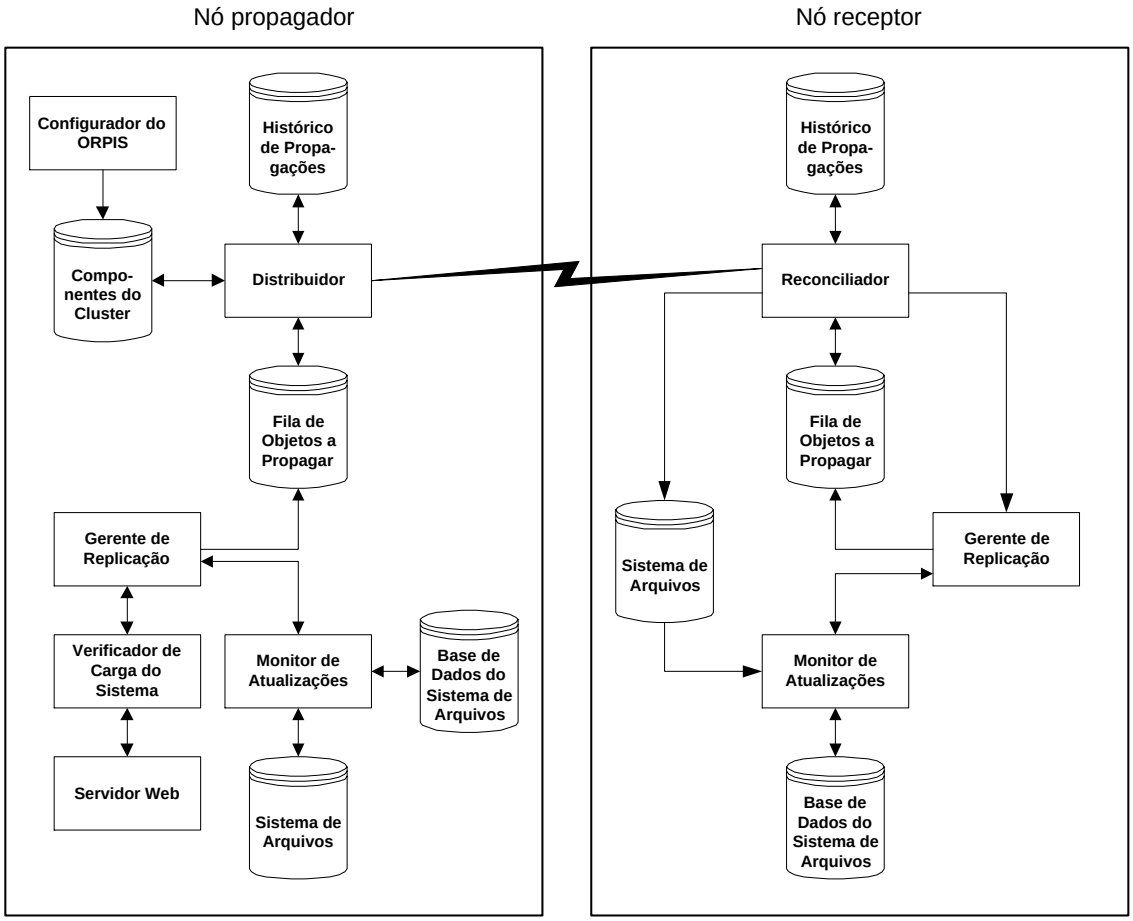

Figura 4.17: Arquitetura do modelo ORPIS em dois nós

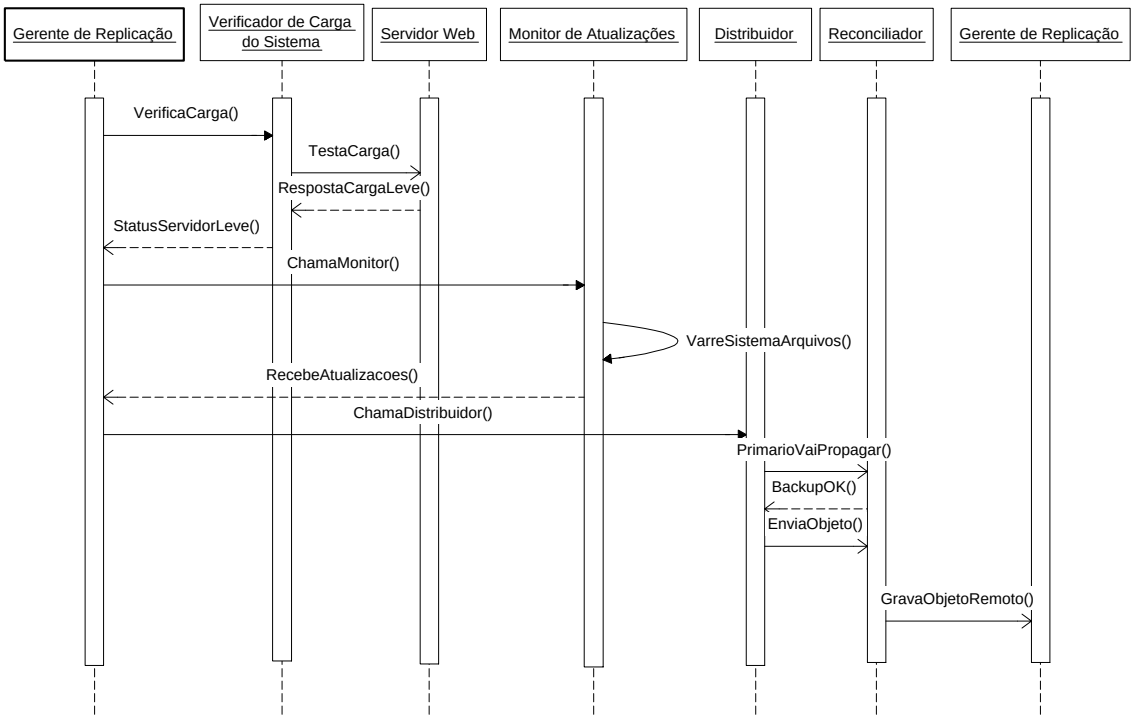

Figura 4.18: Diagrama de seqüência completo

5 IMPLEMENTAÇÃO E TESTES

Este capítulo descreve a implementação e os testes feitos nos módulos do protótipo do modelo ORPIS (*One Replication Protocol for Internet Servers*). O modelo é composto de diversos módulos, conforme descrições apresentadas no capítulo quatro. É feita a descrição do ambiente de desenvolvimento do protótipo, e são apresentados os detalhes da implementação. Apresentam-se os atributos e métodos das classes do protótipo e definem-se as estruturas das tabelas utilizadas. Ao final, são apresentados os resultados obtidos da avaliação de desempenho dos módulos implementados no protótipo.

5.1 Descrição do ambiente de desenvolvimento

O protótipo do modelo ORPIS foi implementado no laboratório de computação da URCAMP, em Bagé. No laboratório de computação da URCAMP, os equipamentos escolhidos foram quatro microcomputadores K6-II, de 500 MHz, com 64 MB de RAM. Como plataforma de sistema operacional, optou-se pelo Linux, com as seguintes características:

- distribuição Conectiva Linux 6, versão servidor, nível de execução três e servidor gráfico desativado (com o objetivo de não degradar o desempenho);
- *kernel* versão 2.2.17-14cl.

Na comunidade científica, Linux é o sistema operacional mais usado basicamente por duas razões: é distribuído como software de código aberto e custa pouco, tanto no aspecto da plataforma de hardware necessário quanto no software (BAKER, 2000). Ele oferece toda a funcionalidade que é esperada de um Unix padrão, e está se desenvolvendo rapidamente já que as funcionalidades que ainda não possui podem ser facilmente implementadas por quem delas precisa. No entanto, essas soluções não são tão exaustivamente testadas quanto às versões comerciais do Unix. Essa falta de testes exige freqüentes atualizações, o que não facilita o serviço dos administradores em manter um sistema estável, como é necessário nos ambientes comerciais. A popularidade do Linux promove uma grande e crescente comunidade que desenvolve várias ferramentas e ambientes para controlar e gerenciar *clusters* que, na maioria das vezes, são gratuitos.

O servidor *Web* utilizado foi o Apache, versão 1.3.14-6cl. A escolha desse software servidor deveu-se à sua ampla utilização na Internet, pois é reconhecidamente considerado o mais utilizado.

A forma de comunicação adotada entre os módulos do modelo ORPIS é através de *sockets*. *Socket* é o ponto final de um canal de comunicação entre dois programas que estão executando em rede. Um *socket* é ligado a um número de porta a fim de que a camada TCP possa identificar o destino de envio dos dados de uma aplicação (SUN, 2001). O protocolo TCP oferece um canal de comunicação ponto-a-ponto que as aplicações cliente-servidor na Internet usam para se comunicar. Cada programa anexa um *socket* ao ponto final da conexão. Para se comunicar, o cliente e o servidor lêem e escrevem no *socket* anexado à conexão.

O recurso de direcionamento de requisições HTTP foi obtido através da recompilação do servidor *Web* Apache, com a inclusão do módulo `mod_backhand` (SCHLOSSNAGLE, 2000). O módulo `mod_backhand` foi iniciado como um projeto de sala de aula no departamento de Ciência da Computação da universidade Johns Hopkins. Após ter sido basicamente comprovado através de uma implementação em C++, o projeto tornou-se um trabalho final de graduação.

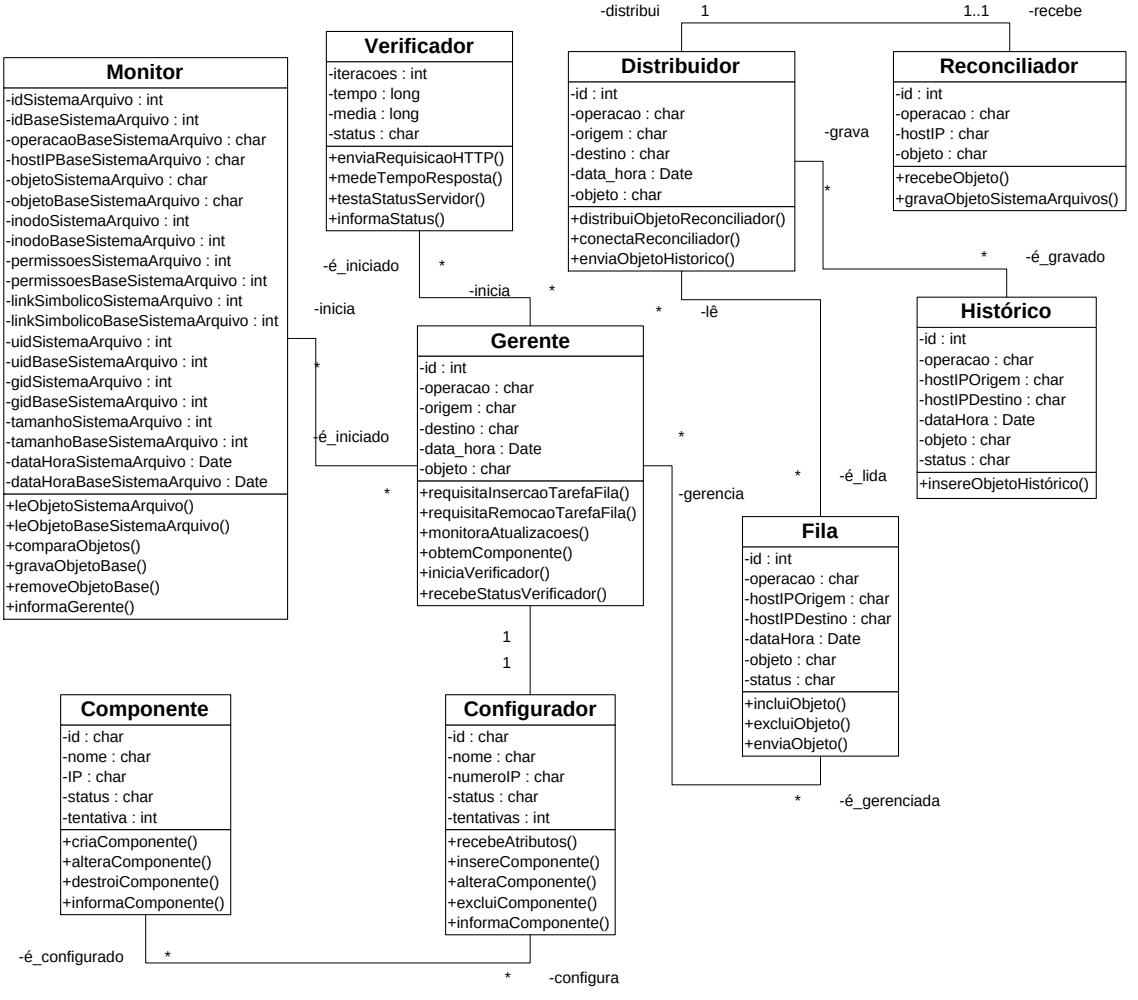

Figura 5.1: Diagrama de classes do modelo ORPIS

5.2 Classes

A figura 5.1 apresenta o diagrama de classes do modelo ORPIS. Esse diagrama foi elaborado utilizando-se a notação UML prevista na ferramenta Visio 2000. Dos

módulos que compõem o modelo ORPIS, apenas as classes Configurador e Verificador foram implementadas. As demais serão desenvolvidas futuramente. A seguir, são descritas as classes componentes do modelo ORPIS.

5.2.1 Classe configurador

A classe Configurador tem os seguintes atributos:

- `id` – identificador numérico único do servidor;
- `nome` – nome do servidor;
- `numeroIP` – número IP do servidor;
- `status` – estado do servidor, podendo valer 0 (inativo) ou 1 (ativo);
- `tentativas` – limite máximo de tentativas para restabelecer a conexão perdida.

O método `recebeAtributos()` recebe valores para as variáveis de instância da classe. O método `insereComponente()` inclui um novo servidor que faz parte da "comunidade ORPIS". Os métodos `alteraComponente()` e `excluiComponente()` alteram as informações de um componente e excluem um determinado componente, respectivamente. Por fim, o método `informaComponente()` retorna informações a respeito de um servidor.

O módulo Configurador tem, por natureza, uma característica distribuída e sua manipulação deve prever o gerenciamento remoto dos componentes do *cluster*. Portanto, a classe Configurador precisava ser implementada através do uso de ferramentas que facilitassem sua portabilidade e permitissem seu funcionamento em um ambiente distribuído. Foi escolhido o modelo *Common Gateway Interface* (CGI), com trechos do módulo executados no servidor *Web* e no cliente. O modelo CGI permite a separação da aplicação em três camadas: interface, regras e dados. A interface é proporcionada por elementos HTML, tais como campos, links e botões de submissão. As regras são armazenadas em módulos, também conhecidos como *server-side scripts*, localizados no servidor de aplicações. Os exemplos mais comuns de aplicações *server-side scripts* são *PHP: Hypertext Preprocessor* (PHP) e *Active Server Pages* (ASP). Os dados são manipulados por sistemas de gerência de banco de dados (SGBD). Os SGBD baseados em software livre mais difundidos são MySQL e PostgreSQL.

A classe Configurador foi implementada utilizando-se as seguintes premissas:

- uso de servidor *Web* – foi escolhido o Apache por ser baseado em código aberto;
- programação com linguagem de processamento de scripts – o PHP (*http://www.php.net*) foi eleito porque, além de possuir código aberto, é extremamente portável, possuindo versões para ambientes Linux e Windows 32 bits;
- utilização de SGBD – optou-se pelo MySQL (*http://www.mysql.com*) porque é muito difundido, com muita documentação disponível, baseado em código aberto e, como o PHP, é de fácil portabilidade.

A utilização do configurador do ORPIS deve ser iniciada através da execução de um navegador *Web*. Deve-se informar o endereço *http://nome_do_servidor/orpis/*. A primeira tela permite a inclusão de novos servidores, modificação ou remoção de servidores existentes, e ainda existe uma opção de visualizar todos os servidores cadastrados. A figura 5.2 apresenta a tela de inclusão de novos servidores.

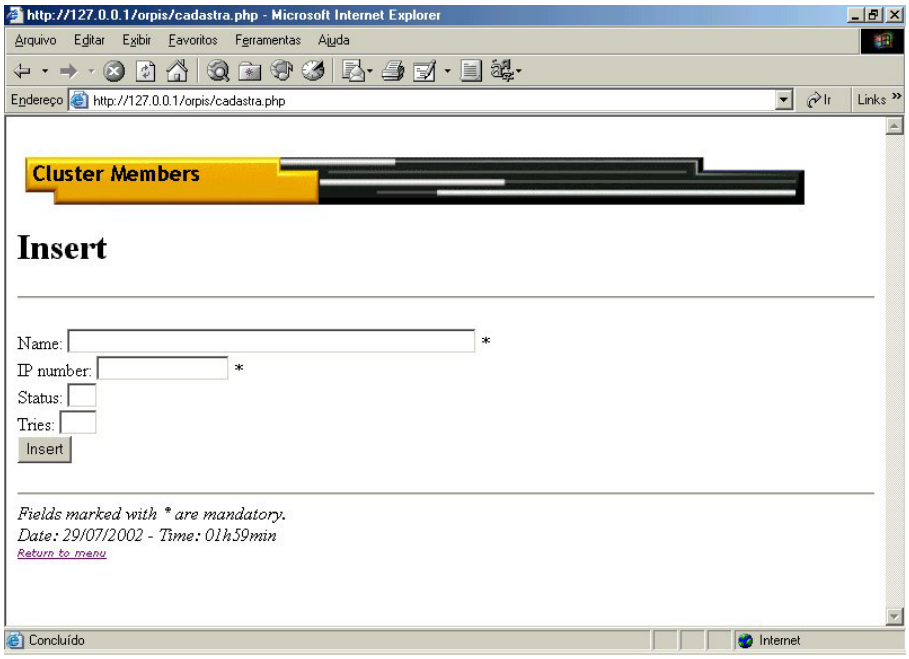

Figura 5.2: Tela da opção Inclusão do configurador do ORPIS

O configurador do ORPIS foi implementado através dos seguintes módulos escritos em PHP: `alterar.php`, `alterar_db.php`, `cadastra.php`, `controle.php`, `excluir.php`, `host.php`, `index.php` e `inserir.php`.

A figura 5.3 apresenta a tela com as opções de Alterar e Excluir do configurador do ORPIS.

A figura 5.4 apresenta a tela de consulta do configurador do ORPIS.

Os requisitos para instalação do protótipo do configurador ORPIS são os seguintes componentes configurados em um servidor Linux:

- servidor *Web* Apache, a partir da versão 1.3;
- servidor de banco de dados MySQL, a partir da versão 3.23;
- módulos do interpretador de scripts PHP, a partir da versão 4.04.

5.2.2 Classe distribuidor

Esta classe é responsável por informar aos diversos componentes do *cluster* as propagações que devem ser efetuadas. Ela tem a responsabilidade de conectar um nó propagador a um nó receptor e informá-lo sobre quais objetos deverão ser sincronizados. Os atributos desta classe são:

- `id` – identificador numérico único que caracteriza uma operação de propagação;
- `operacao` – tipo de operação que a ser realizada (1=inclusão de um novo objeto, 2=alteração de um objeto existente, 3=exclusão de um objeto);

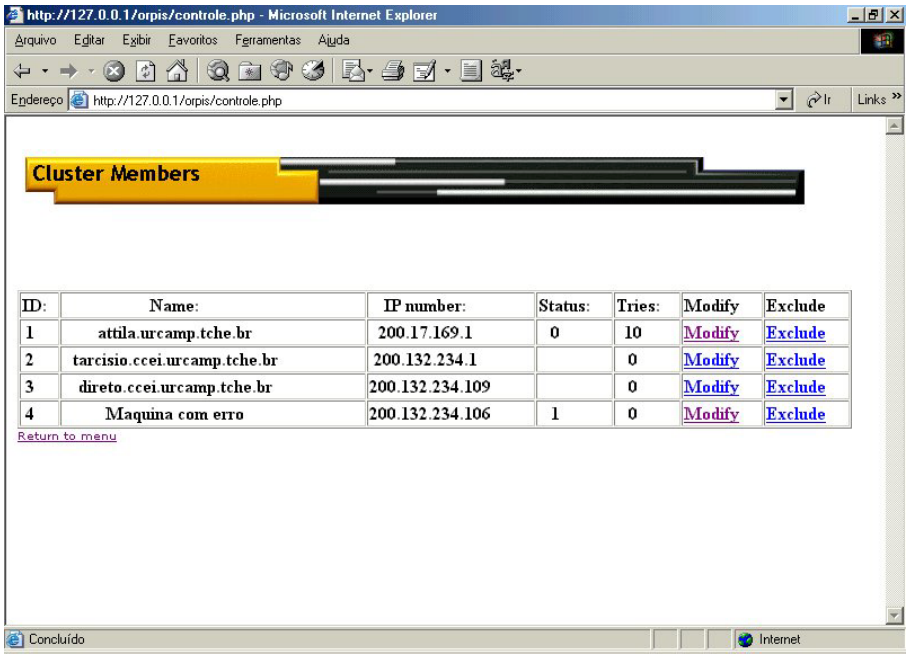

Figura 5.3: Tela das opções de Alterar e Excluir do configurador do ORPIS

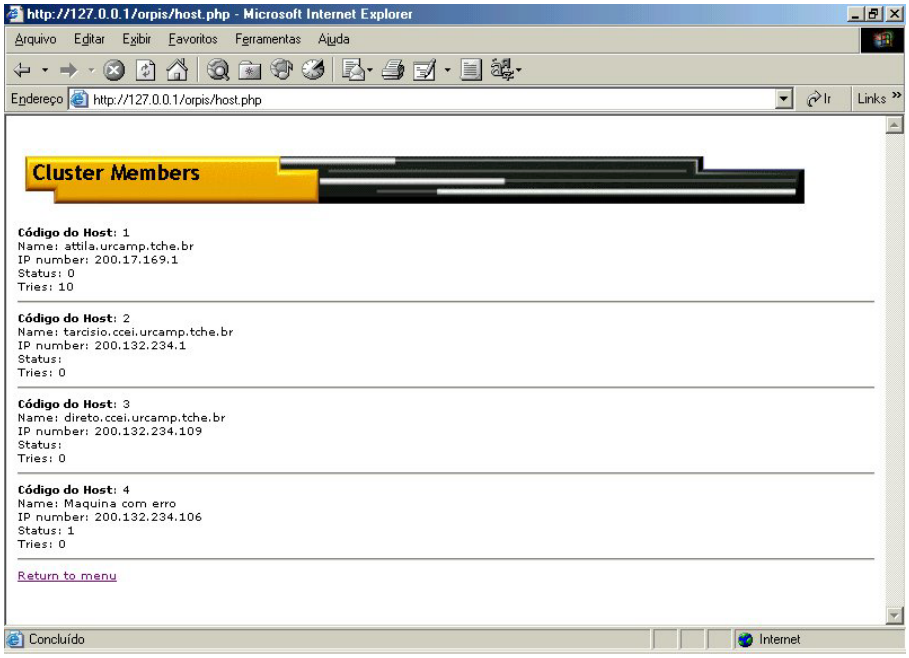

Figura 5.4: Tela de consulta do configurador do ORPIS

- `origem` – número IP do servidor de origem do objeto;
- `destino` – número IP do servidor de destino do objeto;
- `data_hora` – data e hora da propagação da tarefa ao nó receptor;
- `objeto` – caminho dentro do sistema de arquivos e nome do objeto a propagar.

A classe Distribuidor possui o método `conectaReconciliador()`, que é responsável por conectar o nó propagador com o módulo Reconciliador do nó receptor. Outro método é denominado `distribuiObjetoReconciliador()` e tem

a tarefa de enviar a operação a ser feita pelo nó receptor. Finalmente, o método `enviaObjetoHistorico()` tem a função de enviar a tarefa de propagação para a tabela Histórico de Propagações.

5.2.3 Classe gerente

Esta classe é responsável por gerenciar o processo de propagação do conteúdo para os demais componentes do *cluster*. Seus atributos são:

- `id` – identificador numérico único;
- `operacao` – tipo de operação que será realizada no objeto (1=inclusão de um novo objeto, 2=alteração de um objeto existente, 3=exclusão de um objeto);
- `origem` – número IP do servidor de origem do objeto;
- `destino` – número IP do servidor de destino do objeto;
- `data_hora` – data e hora da propagação da tarefa ao nó receptor;
- `objeto` – caminho dentro do sistema de arquivos e nome do objeto a propagar.

O método `requisitaInsercaoTarefaFila()` da classe Gerente envia uma mensagem para o método `incluiObjeto()` da classe Fila de Objetos a Propagar. O método `requisitaRemocaoTarefaFila()` envia uma mensagem para o método `excluiObjeto()` da classe Fila de Objetos a Propagar. O método `monitoraAtualizacoes()` da classe Gerente recebe mensagens do método `informaGerente()` da classe Monitor, informando que existem propagações a serem realizadas. Todos os métodos da classe Gerente se comunicam com os demais métodos das outras classes mencionadas, que estão localizadas no mesmo nó propagador.

O método `obtemComponente()` recebe mensagens do método `informaComponente()` da classe Configurador, informando qual o próximo componente a ser avisado de que existem propagações a serem feitas. O método `iniciaVerificador()` invoca a classe Verificador, que faz o teste de carga do servidor *Web*.

O método `recebeStatusVerificador()` recebe o estado do servidor *Web*, enviado pelo método `informaStatus()` da classe Verificador.

5.2.4 Classe reconciliador

Esta classe é encarregada de efetuar a comunicação nos nós receptores. Tem por objetivo enviar respostas de conexão e recepção de listas de atualizações, feitas pela classe Distribuidor do nó propagador. A comunicação entre essas duas classes é implementada através de mensagens via *sockets*.

Os atributos da classe Reconciliador são:

- `id` – identificador numérico único;
- `operacao` – tipo de operação que será realizada no objeto (1=inclusão de um novo objeto, 2=alteração de um objeto existente, 3=exclusão de um objeto);
- `hostIP` – número IP do componente com o qual está se comunicando;
- `objeto` – caminho dentro do sistema de arquivos e nome do objeto a propagar.

Os métodos da classe Reconciliador são:

- `recebeObjeto()` – faz a conexão via protocolo FTP para o nó propagador, solicitando o envio da tarefa a ser realizada no sistema de arquivos do nó receptor;
- `gravaObjetoSistemaArquivos()` – faz a operação de atualização do objeto no sistema de arquivos do nó receptor.

5.2.5 Classe verificador

Esta classe faz o teste de carga no servidor *Web* do componente do *cluster*. São seus atributos:

- `iteracoes` – número de iterações de envio de requisições HTTP;
- `tempo` – medição do tempo (em milissegundos) que o servidor *Web* levou para retornar a resposta;
- `media` – média aritmética simples dos tempos medidos anteriormente;
- `status` – estado do servidor *Web* (*false*= "leve", *true*=sobrecarregado).

Os métodos da classe Verificador são:

- `enviaRequisicaoHTTP()` – envia uma requisição HTTP ao servidor *Web*, utilizando a primitiva `GET`;
- `medeTempoResposta()` – cronometra o tempo (em milissegundos) que o servidor *Web* leva para enviar a resposta do método `enviaRequisicaoHTTP()`;
- `testaStatusServidor()` – compara a média obtida nas respostas e declara o servidor *Web* sobrecarregado caso seja uma média elevada;
- `informaStatus()` – envia o estado do servidor *Web* para o método `recebeStatusVerificador()` da classe Gerente.

5.2.6 Classe monitor

Esta classe tem a função de fazer a varredura passiva do sistema de arquivos, buscando atualizações que tenham sido feitas. Essa verificação é feita através da comparação dos objetos armazenados no sistema de arquivos com os que estão armazenados na base de dados do sistema de arquivos. Os atributos e métodos desta classe se referem ao sistema de arquivos de um ambiente Unix ou Linux.

Os atributos desta classe são:

- `idSistemaArquivo` – identificador numérico único do objeto no sistema de arquivos;
- `idBaseSistemaArquivo` – identificador numérico único do objeto na base de dados do sistema de arquivos;
- `operacaoBaseSistemaArquivo` – indicação da operação feita (Inclusão, Alteração ou Exclusão) na base de dados do sistema de arquivos;

- `hostIPBaseSistemaArquivo` – número IP do servidor;
- `objetoSistemaArquivo` – caminho dentro do sistema de arquivos e nome do objeto;
- `objetoBaseSistemaArquivo` – caminho dentro do sistema de arquivos e nome do objeto a propagar;
- `inodoSistemaArquivo` – *inodo* atual do objeto;
- `inodoBaseSistemaArquivo` – *inodo* do objeto;
- `permissoesSistemaArquivo` – número atual correspondente às permissões do objeto;
- `permissoesBaseSistemaArquivo` – número atual correspondente às permissões do objeto na base de dados do sistema de arquivos;
- `linkSimbolicoSistemaArquivo` – número atual indicativo de link simbólico do objeto;
- `linkSimbolicoBaseSistemaArquivo` – número indicativo de link simbólico do objeto na base de dados do sistema de arquivos;
- `uidSistemaArquivo` – número atual do usuário proprietário do objeto;
- `uidBaseSistemaArquivo` – número do usuário proprietário do objeto na base de dados do sistema de arquivos;
- `gidSistemaArquivo` – número atual do grupo proprietário do objeto;
- `gidBaseSistemaArquivo` – número do grupo do proprietário do objeto na base de dados do sistema de arquivos;
- `tamanhoSistemaArquivo` – tamanho atual do objeto em kilobytes;
- `tamanhoBaseSistemaArquivo` – tamanho do objeto na base de dados do sistema de arquivos, em kilobytes;
- `dataHoraSistemaArquivo` – data e hora atual de criação ou atualização do objeto;
- `dataHoraBaseSistemaArquivo` – data e hora de criação ou de atualização do objeto na base de dados do sistema de arquivos.

Os métodos da classe Monitor são:

- `leObjetoSistemaArquivo()` – lê um objeto do sistema de arquivos;
- `leObjetoBaseSistemaArquivo()` – lê um objeto da base de dados do sistema de arquivos;
- `comparaObjetos()` – faz a comparação entre os atributos dos objetos lidos;

- `gravaObjetoBase()` – cria ou atualiza o objeto na base de dados do sistema de arquivos;
- `removeObjetoBase()` – remove um objeto da base de dados do sistema de arquivos;
- `informaGerente()` – informa ao método `monitoraAtualizacoes()` da classe Gerente que existem propagações a serem realizadas.

A classe Monitor exigia uma implementação que a tornasse rápida e leve para o servidor, isto é, utilizasse poucos recursos de memória e processador. As linguagens disponíveis que foram analisadas para essa implementação foram C++ e *perl* (*Practical Extraction and Report Language*). A versão livre de C++ é denominada *gpp*. Devido à natureza de *perl* em ser otimizada para varredura de arquivos texto, extração de informações a partir destes arquivos, impressão de relatórios (PERLDOC.COM, 2002) e possuir código aberto, optou-se por *perl* para a implementação do protótipo da classe Monitor.

5.3 Tabelas

A seguir são apresentadas as estruturas das tabelas implementadas no protótipo do modelo ORPIS. A tabela 5.1 apresenta os atributos da tabela Componentes do *Cluster*. A tabela 5.2 apresenta os atributos da tabela Base de Dados do Sistema de Arquivos. A tabela 5.3 apresenta os atributos da tabela Fila de Objetos a Propagar e a tabela 5.4 apresenta os atributos da tabela Histórico de Propagações.

Tabela 5.1: Estrutura da tabela Componentes do *Cluster*

Atributo	Tipo	Tamanho	Índice	Descrição
Host_ID	Char	3	Sim	Identificador numérico único
Host_Nome	Char	255	Não	Nome do servidor
Host_IP	Char	15	Não	Número IP do servidor
Host_Status	Char	1	Não	Status do servidor
Host_Tentativa	Tinyint	2	Não	Número de tentativas de contato com o servidor

5.4 Avaliação funcional

A avaliação funcional do protótipo do modelo ORPIS foi realizada sobre os módulos implementados: Configurador e Monitor de Atualizações. A seguir, são apresentados os resultados obtidos.

5.4.1 Avaliação funcional do Configurador

A avaliação funcional do Configurador do ORPIS foi feita através da sua utilização desempenhando as tarefas de inclusões, consultas e exclusões de servidores de uma "comunidade ORPIS" composta por cinco servidores hipotéticos. Todas as operações foram desempenhadas com sucesso. É importante salientar que se trata

Tabela 5.2: Estrutura da tabela Base de Dados do Sistema de Arquivos

Atributo	Tipo	Tamanho	Índice	Descrição
Fsdb_ID	Int	10	Sim	Identificador numérico único
Fsdb_Operacao	Char	1	Não	Indicação da operação feita (Inclusão, Alteração ou Exclusão)
Fsdb_Host_IP	Varchar	15	Não	Número IP do servidor
Fsdb_Objeto	Text	65535	Não	Caminho dentro do sistema de arquivos e nome do objeto a propagar
Fsdb_Inode	Bigint	20	Não	Inodo do objeto
Fsdb_Permissoes	Int	10	Não	Número correspondente às permissões do objeto
Fsdb_NLinks	Int	10	Não	Número indicativo de links simbólicos do objeto
Fsdb_UID	Smallint	5	Não	Número de usuário do dono do objeto
Fsdb_GID	Smallint	5	Não	Número do grupo do dono do objeto
Fsdb_Tamanho	Int	10	Não	Tamanho do objeto em kilobytes
Fsdb_Data_Hora	Datetime	–	Não	Data e hora do sistema no formato ´YYYY-MM-DD HH:MM:SS´

Tabela 5.3: Estrutura da tabela Fila de Objetos a Propagar

Atributo	Tipo	Tamanho	Índice	Descrição
Fila_ID	Int	10	Sim	Identificador numérico único
Fila_Operacao	Char	1	Não	Indicação da operação a fazer (Inclusão, Alteração ou Exclusão)
Fila_Host_IP_Origem	Varchar	15	Não	Número IP do servidor de origem do objeto
Fila_Host_IP_Destino	Varchar	15	Não	Número IP do servidor de destino do objeto
Fila_Data_Hora	Datetime	–	Não	Data e hora do sistema no formato ´YYYY-MM-DD HH:MM:SS´
Fila_Objeto	Text	65535	Não	Caminho dentro do sistema de arquivos e nome do objeto a propagar
Fila_Status	Char	1	Não	Indicativo do status do objeto

de uma aplicação sendo executada através do servidor *Web*, baseada em uma linguagem interpretada, acessando um banco de dados cliente-servidor e cuja interface se encontra em uma máquina cliente. Nesse caso, existem vários elementos que podem provocar degradação do desempenho. Pode-se citar alguns mais significativos, tais como quantidade de memória da máquina cliente, *clock* do processador do computador cliente, versão do *browser* utilizado, número de pacotes transitando na rede e número de usuários simultâneos usando recursos do servidor ou da rede.

Tabela 5.4: Estrutura da tabela Histórico de Propagações

Atributo	Tipo	Tamanho	Índice	Descrição
Hist_ID	Int	10	Sim	Identificador numérico único
Hist_Operacao	Char	1	Não	Indicação da operação a fazer (Inclusão, Alteração ou Exclusão)
Hist_Host_IP_Origem	Varchar	15	Não	Número IP do servidor de origem do objeto
Hist_Host_IP_Destino	Varchar	15	Não	Número IP do servidor de destino do objeto
Hist_Data_Hora	Datetime	–	Não	Data e hora do sistema no formato ´YYYY-MM-DD HH:MM:SS´
Hist_Objeto	Text	65535	Não	Caminho dentro do sistema de arquivos e nome do objeto a propagar
Hist_Status	Char	1	Não	Indicativo do status do objeto

5.4.2 Avaliação funcional do Monitor de Atualizações

O objetivo da avaliação funcional do Monitor de Atualizações foi o de medir o tempo médio que este módulo leva para completar sua execução e que é um indicativo da degradação que pode ocorrer no desempenho do computador no qual está instalado.

O computador utilizado na avaliação foi um dos K6-II, com 500 MHz e com 64 Mb de RAM.

A avaliação funcional do Monitor de Atualizações foi realizada com base em quatro amostras do sistema de arquivos com as seguintes características:

- Amostra 1:
  - 3.119 arquivos;
  - 309 subdiretórios;
  - 307 Mb de tamanho total da estrutura de diretórios;
- Amostra 2:
  - 6.821 arquivos;
  - 748 subdiretórios;
  - 655 Mb de tamanho total da estrutura de diretórios;
- Amostra 3:
  - 48.526 arquivos;
  - 4.392 subdiretórios;
  - 1,4 Gb de tamanho total da estrutura de diretórios;

- Amostra 4:
  - 55.571 arquivos;
  - 5.169 subdiretórios;
  - 2,07 Gb de tamanho total da estrutura de diretórios.

As figuras 5.5, 5.6 e 5.7 apresentam, respectivamente, as características das quatro amostras selecionadas do sistema de arquivos.

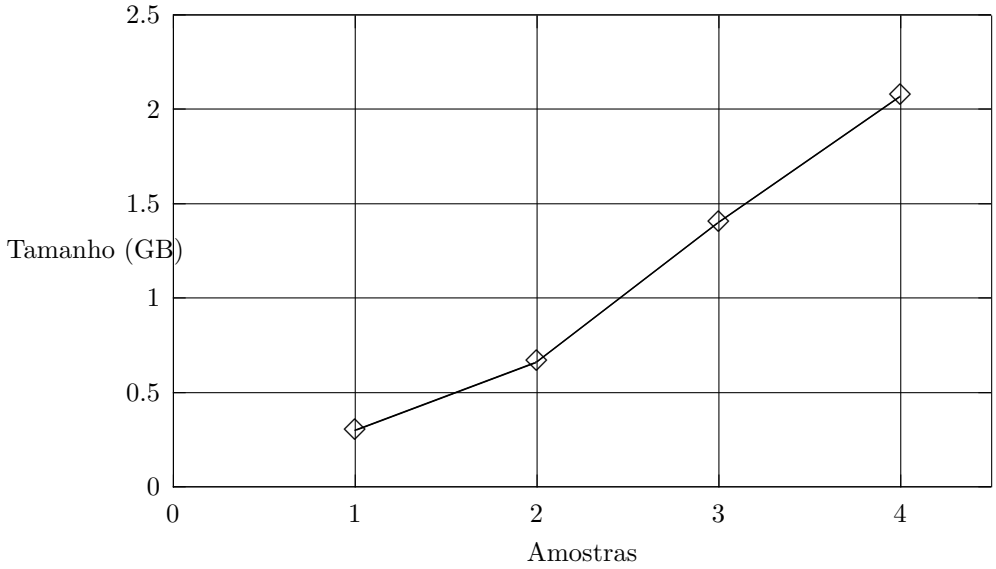

Figura 5.5: Tamanho das amostras, em gigabytes

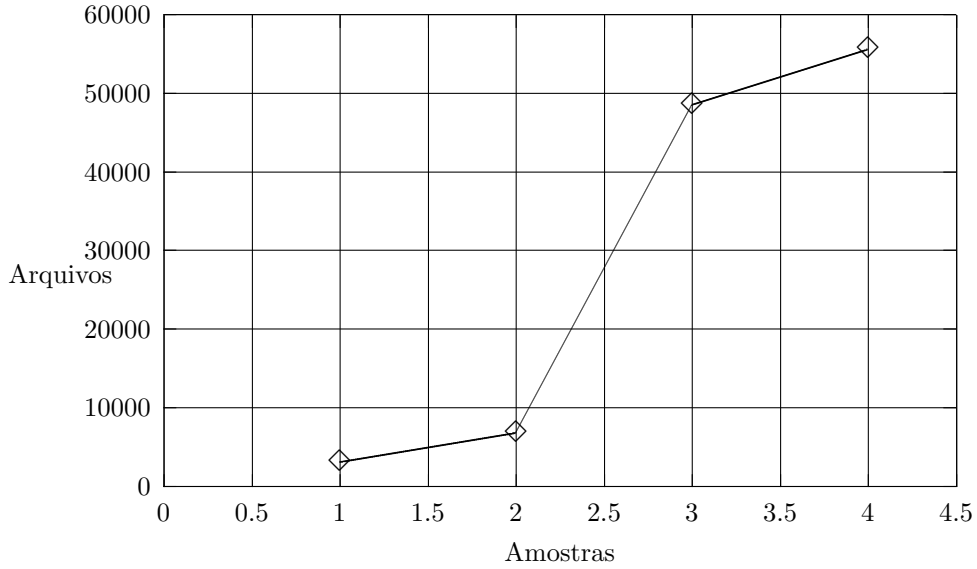

Figura 5.6: Número de arquivos das amostras

A figura 5.8 apresenta os tamanhos das bases do sistemas de arquivos que foram geradas pelo Monitor de Atualizações. O eixo vertical (Y) representa o tamanho

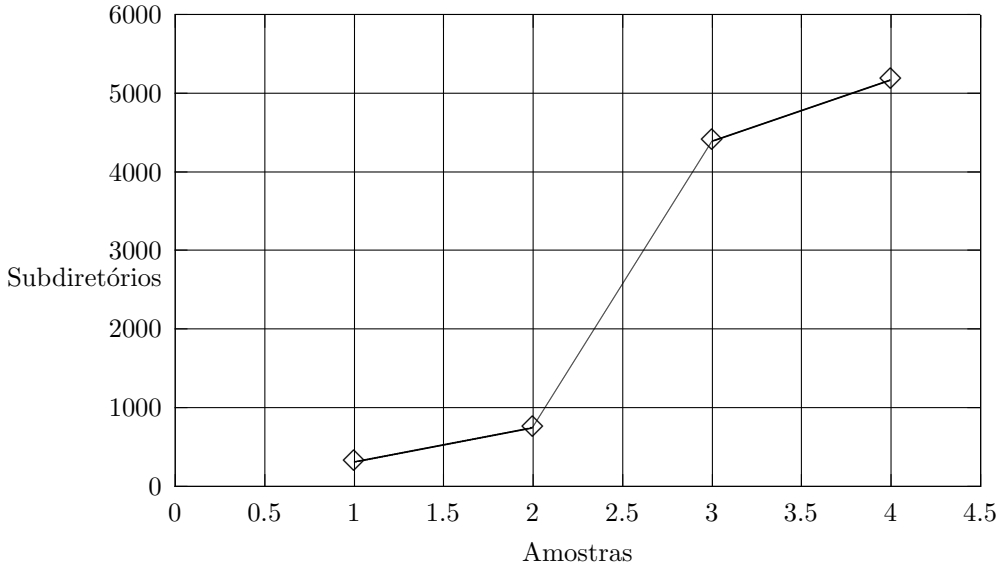

Figura 5.7: Número de subdiretórios das amostras

em megabytes (MB) e o eixo horizontal (X) representa a amostra a que esta base se refere.

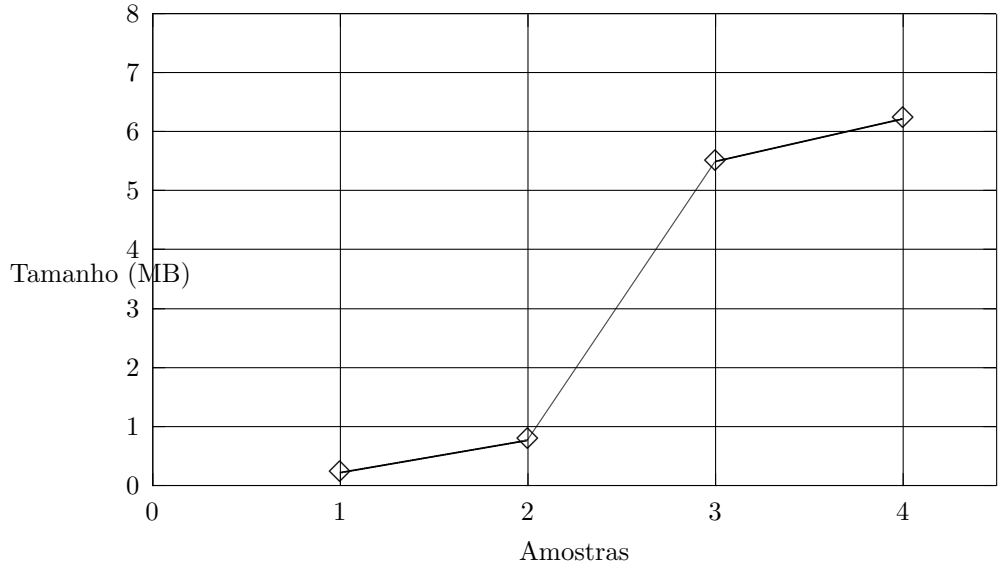

Figura 5.8: Tamanho das bases criadas

De acordo com Killelea (1998), existe uma forma simples de se medir o tempo de execução de um aplicativo, através do comando `time`, presente na maioria das distribuições Linux.

Foram realizados três experimentos. Em todos eles, tomou-se o tempo através do comando `time`. Esse comando apresenta três valores como resultado: *real*, *user* e *sys*. Foram considerados apenas os valores obtidos pelo valor *real*. Nos três gráficos seguintes, os eixos verticais representam os tempos tomados, em segundos, e os eixos horizontais representam os números de execuções (iterações).

O primeiro experimento baseou-se em vinte execuções do Monitor de Atualizações no modo de criação do banco de dados do sistema de arquivos. A figura 5.9 apresenta um gráfico com os resultados obtidos nesse experimento. Os tempos médios obtidos foram:

- Amostra 1 – 1,29 segundos;
- Amostra 2 – 3,54 segundos;
- Amostra 3 – 40,5 segundos;
- Amostra 4 – 48,65 segundos.

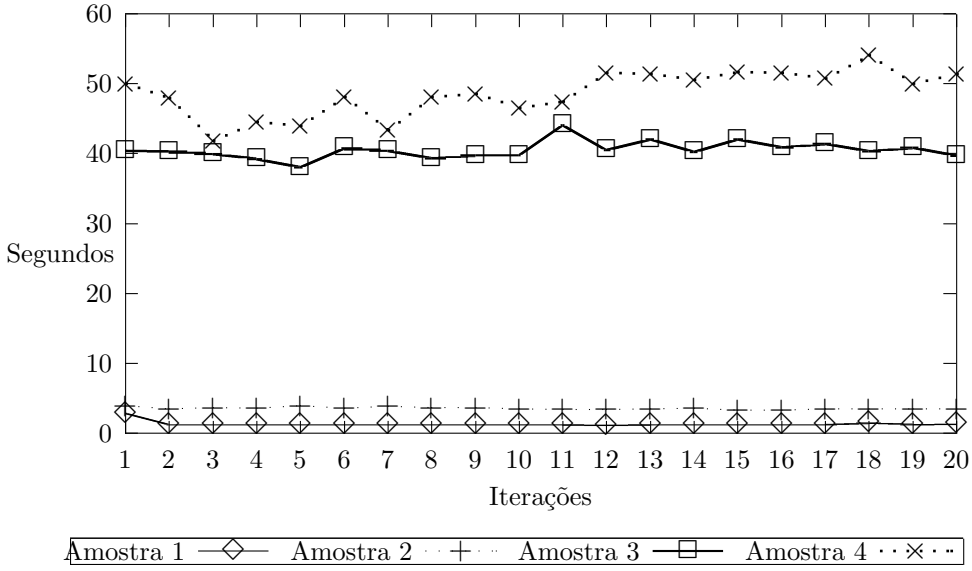

Figura 5.9: Modo de criação do banco de dados do sistema de arquivos

O segundo experimento baseou-se em vinte execuções do Monitor de Atualizações no modo de comparação do sistema de arquivos com o banco de dados. Não havia sido feita nenhuma atualização na estrutura de diretórios. A figura 5.10 apresenta um gráfico com os resultados obtidos nessa avaliação. Os tempos médios obtidos foram:

- Amostra 1 – 1,41 segundos;
- Amostra 2 – 4,32 segundos;
- Amostra 3 – 2 minutos e 32,85 segundos;
- Amostra 4 – 3 minutos e 11,62 segundos.

O terceiro experimento baseou-se em vinte execuções do Monitor de Atualizações no modo de comparação do sistema de arquivos com o banco de dados. Havia sido incluído um arquivo na estrutura de diretórios. A figura 5.11 apresenta um gráfico com os resultados obtidos nessa avaliação. Os tempos médios obtidos foram:

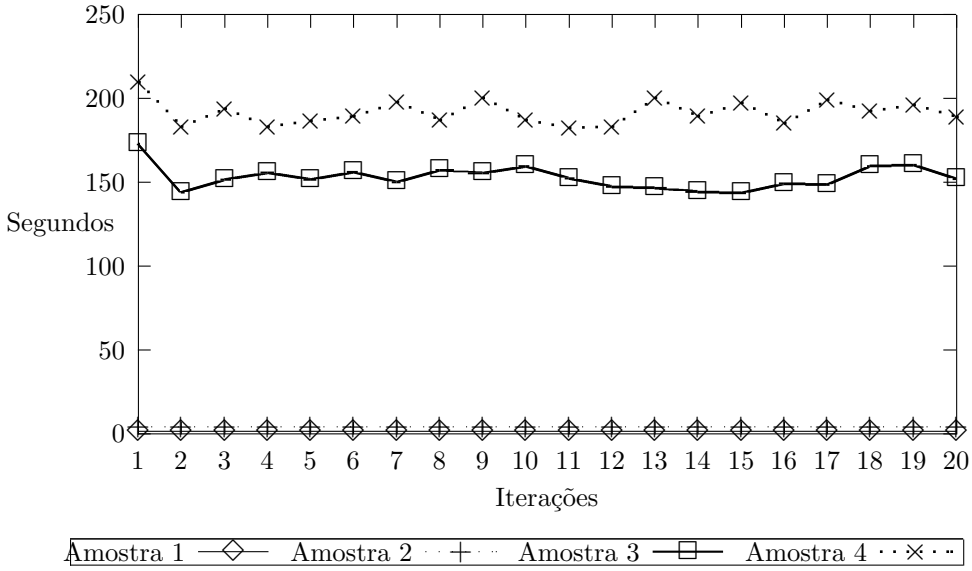

Figura 5.10: Modo de comparação do sistema de arquivos sem atualizações detectadas

- Amostra 1 – 1,43 segundos;
- Amostra 2 – 4,32 segundos;
- Amostra 3 – 2 minutos e 37,53 segundos;
- Amostra 4 – 3 minutos e 5,86 segundos.

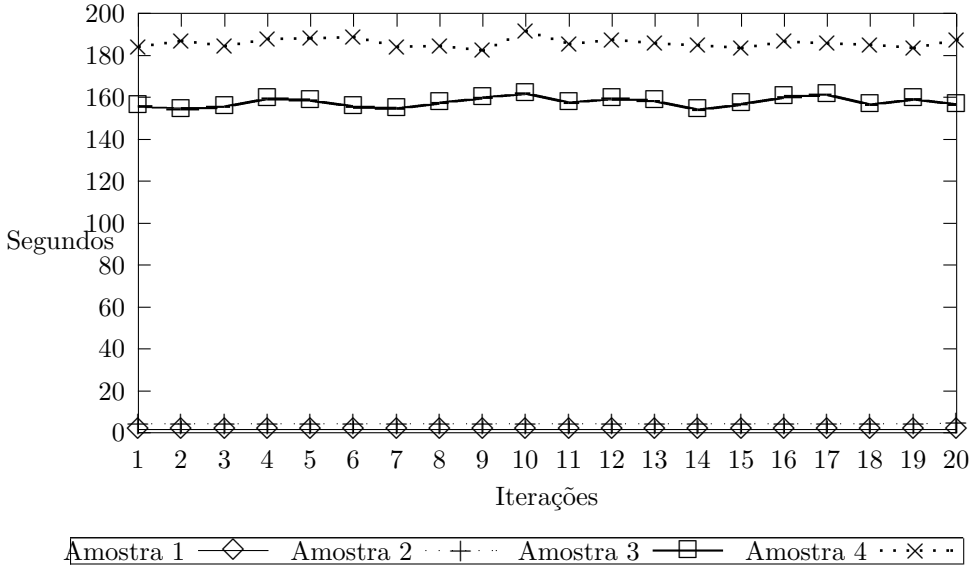

Figura 5.11: Modo de comparação do sistema de arquivos com uma atualização detectada

5.5 Análise dos resultados

Esses experimentos buscaram analisar o impacto no desempenho do servidor *Web* com a inclusão de mais uma aplicação sendo executada de tempos em tempos, a fim de detectar atualizações ocorridas.

Pôde-se perceber que a unidade de replicação é o arquivo, pois o tamanhos dos arquivos foi irrelevante no desempenho do Monitor de Atualizações. O comportamento das curvas das figuras 5.6, 5.7 e 5.8 é proporcional ao número de arquivos e desproporcional aos tamanhos das amostras (figura 5.5). Foi obtido um tempo médio considerado reduzido nos experimentos em relação às amostras 1 e 2, porém os tempos obtidos com as amostras 3 e 4 foram considerados elevados.

5.6 Considerações finais

Este capítulo descreveu a implementação e os testes feitos nos módulos do protótipo do modelo ORPIS. Foi feita a descrição do ambiente de desenvolvimento do protótipo e foram apresentados os detalhes da implementação. Enumeraram-se os atributos e métodos das classes do protótipo e foram definidas as estruturas das tabelas utilizadas. Ao final, foram apresentados os resultados obtidos da avaliação funcional dos módulos implementados no protótipo. O capítulo a seguir apresenta as conclusões deste trabalho.

6 CONCLUSÃO

O crescimento exponencial da Internet e a crescente demanda de serviços que a utilizam provocaram uma diminuição considerável no seu desempenho. Refletindo esse fato, alguns autores se referem à *World Wide Web* como *World Wide Wait*, isto é, espera de alcance mundial. Este trabalho se preocupa exatamente com isso e ataca de frente o problema de proporcionar mais desempenho aos servidores dos sítios *Web* que eventualmente recebam um número elevado de clientes em determinados momentos. Ele propõe um modelo que distribui a carga de requisições dos clientes entre diversos servidores *Web* distribuídos, atuando como um só, e associados a uma estratégia de replicação de conteúdo. Esse modelo foi denominado de ORPIS (***O****ne* ***R****eplication* ***P****rotocol for* ***I****nternet* ***S****ervers*).

O modelo ORPIS se insere no contexto de soluções acadêmicas que têm a preocupação de oferecer ferramentas portáveis, que possam ser facilmente implementadas em qualquer ambiente, e livres, com a finalidade de serem utilizadas por todos os interessados.

Através do estudo bibliográfico realizado, pôde-se perceber que esta é uma área de pesquisa ainda em pleno desenvolvimento, com diversas soluções atualmente sendo propostas. Outra constatação é que a literatura existente sobre técnicas de replicação é mais direcionada ao contexto de bancos de dados ou sistemas de arquivos distribuídos, com semântica de consistência pessimista.

Este trabalho também contribuiu, dentro do contexto do GPPD, com uma sistematização de técnicas de aumento de desempenho em servidores *Web*. Foram apresentados os componentes tecnológicos empregados na *Web*, além dos problemas causados pela escalabilidade e pela distribuição inerentes a esse ambiente. As principais técnicas de aumento de desempenho que atualmente vêm sendo empregadas na *Web* foram descritas. Selecionaram-se estas técnicas a partir de dois critérios: relevância prática e tempo disponível para escrita desta publicação.

Esta dissertação também apresentou conceitos fundamentais envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos. Foram apresentados detalhes sobre arquitetura de servidores *Web* distribuídos, manutenção da consistência em ambientes de servidores *Web* distribuídos, uso de replicação e formas de replicação. Além disso, foram citados alguns trabalhos correlatos ao propósito de manter réplicas consistentes em ambientes de servidores *Web* distribuídos.

Descreveu-se o modelo ORPIS, tendo sido elencados seus pressupostos e descritos os seus componentes. Os algoritmos de funcionamento do modelo também foram apresentados. Apresentaram-se informações relativas à implementação e aos testes realizados em alguns módulos do protótipo do modelo ORPIS. Foi descrito o ambiente de desenvolvimento do protótipo, e foram apresentados detalhes da imple-

mentação. Enumeram-se os atributos e métodos das classes do protótipo e definidas as estruturas de dados utilizadas. Além disso, apresentaram-se os resultados obtidos da avaliação de desempenho dos módulos implementados no protótipo.

Este trabalho de pesquisa, que resultou nesta dissertação, gerou três publicações que foram apresentadas à comunidade científica através dos seguintes meios:

- um resumo (GEYER; LIMA, 2001), na ERAD 2001 – 1ª Escola Regional de Alto Desempenho, realizada em janeiro de 2001 na cidade de Gramado, RS;

- um artigo (LIMA; ROYES MELLO; GEYER, 2002), publicado na Revista do CCEI, da Universidade da Região da Campanha, em Bagé, RS;

- um artigo (LIMA; GEYER, 2003), publicado no 1º Workshop GPPD/UFRGS, realizado em junho de 2003 em Porto Alegre.

O modelo ORPIS adota uma estratégia de propagação otimista porque não existe sincronismo no envio das atualizações para as réplicas. Outro ponto a ser salientado é a compatibilidade do modelo ORPIS aos servidores *Web* existentes, sem a necessidade de modificação em suas configurações. Essa característica também se aplica aos editores e atualizadores dos diretórios do servidor *Web*.

O modelo pode ser usado em ambientes de computação distribuída com estrutura não fixa (ex: grids computacionais) já que permite a configuração de diversos componentes de forma dinâmica.

O uso de software de código aberto no desenvolvimento do protótipo proporcionou um rápido acesso às ferramentas necessárias (sistema operacional, linguagens e gerenciador de banco de dados), com possibilidade de alteração nos códigos fonte como uma alternativa de *customização*.

Considera-se que o modelo descrito nesta dissertação ainda possui algumas limitações, já que o configurador do ORPIS ainda não permite a inclusão dos diretórios a serem replicados. Atualmente, esse processo deve ser feito manualmente, através da edição de um arquivo lido pelo monitor de atualizações. Além disso, o modelo não trata a serialização das operações de atualização das páginas.

Os vários componentes do modelo ORPIS ainda estão em desenvolvimento e várias funcionalidades podem ser incorporadas futuramente. Portanto, podem ser destacados como trabalhos futuros:

- desenvolvimento de um módulo anexado ao servidor *Web* que desvie as requisições dos usuários para o servidor replicado que apresente as melhores condições de acesso ou que esteja mais disponível, baseado em métricas relativas ao desempenho da rede e à carga imposta no momento;

- implementação dos demais componentes do modelo ORPIS que ainda não foram desenvolvidos;

- aperfeiçoamentos no protótipo, através da otimização do código ou até mesmo geração de código nativo da plataforma de hardware, proporcionando um melhor desempenho;

- testes funcionais mais apurados, utilizando ferramentas que proporcionem métricas mais precisas;

- geração de carga de trabalho sintética, simulando o acesso simultâneo de vários clientes e medição do desempenho do servidor *Web*, com o objetivo de simular situações possíveis de acontecerem em um ambiente de produção.

REFERÊNCIAS

ABDELZAHER, T. F.; BHATTI, N. Web Content Adaptation to Improve Server Overload Behavior. **Computer Networks**, Amsterdam, v.31, n.11–16, p.1563–1577, 1999.

ADLER, S. **The Slashdot Effect – an analysis of three internet publications**. 1999. Disponível em: ⟨http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html⟩. Acesso em: 17 jan. 2001.

AMIR, Y. **Replication Using Group Communication Over a Partitioned Network**. 1995. 95p. Tese (Doutorado em Ciência da Computação) — Hebrew University of Jerusalem, Jerusalem.

AUGUSTIN, I.; GEYER, C. F. R. **O Acesso aos Dados no Contexto da Computação Móvel**. Porto Alegre, 2000. 148f. Exame de qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BAKER, M. **Cluster Computing White Paper Version 2.0**. 2000. Disponível em: ⟨http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/final-paper.pdf⟩. Acesso em: 9 set. 2001.

BAVENDIEK, G. **Copying Files Using Mirror**. 1998. Disponível em: ⟨http://www.linuxgazette.com/issue24/bavendiek.html⟩. Acesso em: 30 jun. 2000.

BERNERS-LEE, T. et al. **RFC 2396 Uniform Resource Identifiers (URI)**: generic syntax. 1998. Disponível em: ⟨http://www.innosoft.com/rfc/rfc2396.html⟩. Acesso em: 22 fev. 2000.

BERNERS-LEE, T.; FIELDING, R. T.; FRYSTYK, H. **RFC 1945 – Hypertext Transfer Protocol – HTTP-1_0**. 1996. Disponível em: ⟨http://www.faqs.org/rfcs/rfc1945.html⟩. Acesso em: 15 nov. 1999.

BRAAM, P. J. **InterMezzo File System**: replicating http servers. 1999. Disponível em: ⟨http://www.inter-mezzo.org/docs/intermezzo-servrep.pdf⟩. Acesso em: 20 nov. 2000.

CHANDHOK, N. **Web Distribution Systems**: caching and replication. 1999. Disponível em: ⟨ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-99/web_caching.pdf⟩. Acesso em: 12 mar. 2000.

CLARKE, I. et al. **Protecting Freedom of Information Online with Freenet**. 2001. Disponível em: ⟨http://www.doc.ic.ac.uk/~twh1/longitude/papers/ieee.ps.gz⟩. Acesso em: 05 ago. 2001.

CLUSTER FILE SYSTEM, I. **The InterSync & InterMezzo HOWTO**. 2002. Disponível em: ⟨https://projects.clusterfs.com/intermezzo/HowTo⟩. Acesso em: 27 jun. 2003.

COOPER, I.; MELVE, I.; TOMLINSON, G. **Internet Web Replication and Caching Taxonomy**. 2000. Disponível em: ⟨ftp://ftp.nordu.net/internet-drafts/draft-ietf-wrec-taxonomy-05.txt⟩. Acesso em: 20 ago. 2000.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems**: concept and design. 3rd ed. Harlow: Addison-Wesley, 2001.

DÉFAGO, X.; SCHIPER, A.; SERGENT, N. Semi-Passive Replication. In: SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 1998, West Lafayette. **Proceedings...** West Lafayette: IEEE, 1998. p.43–50.

EKENSTAM, T.; MATHENY, C.; REIHER, P. L.; POPEK, G. J. The Bengal Database Replication System. **Distributed and Parallel Databases**, [S.l.], v.9, n.3, p.187–210, 2001.

FERRARI, D. N. **Um Modelo de Replicação em Ambientes que Suportam Mobilidade**. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

GARLAND, M. et al. **Implementing Distributed Server Groups for the World Wide Web**. Pittsburgh: Carnegie Mellon University, 1995. 12p. (Relatório Técnico, n. CMU-CS-95-114). Disponível em: ⟨http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/garland/Papers/CMU-CS-95-114.ps.gz⟩. Acesso em: 01 fev. 2000.

GEYER, C. F. R.; LIMA, C. C. Um Modelo de Replicação de Conteúdo em Servidores Web Distribuídos. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 2001, Gramado. **Anais...** Gramado: SBC, 2001.

GRAY, J. et al. The Dangers of Replication and a Solution. **SIGMOD Record**, New York, v.25, n.2, p.173–182, 1996. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, 1996, Montreal.

GUY, R. G. et al. Implementation of the Ficus Replicated File System. In: SUMMER USENIX CONFERENCE, 1990, Anaheim. **Proceedings...** Anaheim: USENIX, 1990.

GUY, R. G. et al. Rumor: mobile data access through optimistic peer-to-peer replication. In: INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND DATA MINING, 1998, Singapore. **Advances in Database Technologies:** proceedings**...** Berlin: Springer-Verlag, 1999. p.254–265. (Lecture Notes in Computer Science, v.1552).

HU, J. C. **JAWS 2**: refactorization framework design and utilization overview. 1998. Disponível em: ⟨http://www.cs.wustl.edu/~jxh/research/talks/jaws2.ps.gz⟩. Acesso em: 1 fev. 2000.

KILLELEA, P. **Web Performance Tuning**. Newton: O'Reilly, 1998.

KIM, G. H.; SPAFFORD, E. H. The Design and Implementation of Tripwire: a file system integrity checker. In: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 2., 1994, Fairfax. **Proceedings...** Fairfax: ACM Press, 1994. p.18–29.

LAQUEY, T.; RYER, J. C. **O Manual da Internet**. Rio de Janeiro: Campus, 1994.

LAVOIE, B.; NIELSEN, H. F. **Web Characterization Terminology & Definition Sheet**. 1999. Disponível em: ⟨http://www.w3.org/1999/05/WCA-terms/⟩. Acesso em: 22 fev. 2000.

LIMA, C. C.; GEYER, C. F. R. Um Modelo de Replicação de Conteúdo em Servidores *Web* Distribuídos. In: SEMANA ACADÊMICA DO PPGC, 2000, Porto Alegre. **Anais...** Porto Alegre: PPGC da UFRGS, 2000.

LIMA, C. C.; GEYER, C. F. R. ORPIS: uma ferramenta livre para sincronização de conteúdo em servidores *Web* distribuídos. In: WORKSHOP GPPD/UFRGS, 1., 2003, Porto Alegre. **Anais...** Porto Alegre: PPGC da UFRGS, 2003.

LIMA, C. C.; ROYES MELLO, F. de; GEYER, C. F. R. ORPIS: um modelo de consistência de réplicas em servidores web distribuídos. **Revista do CCEI**, Bagé, v.6, p.15–23, 2002.

MEIRA, W.; MURTA, C. D.; RESENDE, R. S. F. **Comércio eletrônico na WWW**. São Paulo: IME–USP, 2000. 167p.

MENASCÉ, D. A. Web Performance: the most obvious challenge for electronic commerce success. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, SEMISH, 1998, Belo Horizonte. **Anais...** Belo Horizonte: SBC, 1998.

MENASCÉ, D. A.; ALMEIDA, V. A. F. **Capacity Planning for Web Performance**: metrics, models, and methods. Englewood Cliffs: Prentice-Hall, 1998.

NOBLE, B. et al. A Case for Fluid Replication. In: NETWORK STORAGE SYMPOSIUM, NETSTORE, 1999, Seattle. **Proceedings...** Seattle: Internet2/UCAID, 1999.

PALPANAS, T.; KRISHNAMURTHY, B. **Reducing Retrieval Latencies in the Web**: the past, the present, and the future. Toronto: Department of Computer Science, 1999. (Relatório Técnico, n. CSRG-378). Disponível em: ⟨http://www.cs.toronto.edu/~themis/publications/latencysurvey.ps⟩. Acesso em: 18 mar. 2000.

PEDONE, F. L. **Um Servidor de Arquivos Multiversão Replicados**. 1995. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

PERLDOC.COM. **Perl 5.6 Documentation**. 2002. Disponível em: ⟨http://www.perldoc.com/perl5.6/pod/perl.html⟩. Acesso em: 30 jul. 2002.

PIERRE, G. et al. Differentiated Strategies for Replicating Web documents. **Computer Communications**, [S.l.], v.24, n.2, p.232–240, Feb. 2001.

PIERRE, G.; KUZ, I.; STEEN, M. van. **Adaptive Replicated Web Documents**. Amsterdam: Vrije Universiteit, 2000. Technical Report IR-477. Disponível em: ⟨http://www.cs.vu.nl/pub/papers/globe/IR-477.00.pdf⟩. Acesso em: 21 set. 2000.

PIERRE, G.; STEEN, M. van. Globule: a platform for self-replicating web documents. In: INTERNATIONAL CONFERENCE ON PROTOCOLS FOR MULTIMEDIA SYSTEMS, 6., 2001, Enschede. **Proceedings...** Enschede: University of Twente, 2001. p.1–11.

PIERRE, G.; STEEN, M. van; TANENBAUM, A. S. **Self-Replicating Web Documents**. Amsterdam: Vrije Universiteit, 2001. Technical Report. Disponível em: ⟨http://www.cs.vu.nl/pub/papers/globe/IR-486.01.pdf⟩. Acesso em 16 mar. 2001.

POLYSERVE. **Data Replication for High Availability Web Server Clusters**. 2000. Disponível em: ⟨http://www.polyserve.com/support/downloads/white_papers/data_replication_for_high_availability_web_servers.pdf⟩. Acesso em: 20 nov. 2000.

RAGGET, D. **HTML 4.01 Specification**. 1999. Disponível em: ⟨http://www.w3.org/TR/REC-html40/⟩. Acesso em: 22 fev. 2000.

ROBERTSON, A. Linux-HA Heartbeat System Design. In: ANNUAL LINUX SHOWCASE AND CONFERENCE, 4., 2000, Atlanta. **Proceedings...** Atlanta: USENIX, 2000.

SAITO, Y. **Optimistic Replication Algorithms**. 2000. Disponível em: ⟨http://128.95.4.112/homes/yasushi/general.ps⟩. Acesso em: 20 nov. 2001.

SATO, L. M. Evolução e Tendências da Programação Paralela. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 2001, Gramado. **Anais...** Gramado: SBC, 2001.

SCHLOSSNAGLE, T. Mod_backhand: a load balancing module for the apache web server. In: APACHECON, 2000, Orlando. **Proceedings...** Orlando: The Apache Software Foundation, 2000.

SUN. **The Java Tutorial**. 2001. Disponível em: ⟨http://java.sun.com/docs/books/tutorial/index.html⟩. Acesso em: 6 jun. 2001.

TANENBAUM, A. S. **Distributed Operating Systems**. Upper Saddle River: Prentice-Hall, 1995.

TRIDGELL, A.; MACKERRAS, P. **The Rsync Algorithm**. Canberra: The Australian National University, 1996. 8 p. (Relatório Técnico, n. TR-CS-96-05). Disponível em: ⟨ftp://samba.anu.edu.au/pub/rsync/tech_report.ps⟩. Acesso em: 07 ago. 2000.

WIESMANN, M.; PEDONE, F.; SCHIPER, A. A Systematic Classification of Replicated Database Protocols based on Atomic Broadcast. In: EUROPEEAN RESEARCH SEMINAR ON ADVANCES IN DISTRIBUTED SYSTEMS, 3., 1999,

Madeira Island (Portugal). **Proceedings...** Madeira Island: University of Lisboa, 1999.

WILKINSON, B. **Computer architecture**: design and performance. London: Prentice-Hall, 1996.