UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELIETE MARCHIORO

UM ESTUDO SOBRE REJUVENESCIMENTO DE SOFTWARE EM SERVIDORES WEB APACHE

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Elizabeth Sueli Specialski
(Orientador)

Florianópolis, novembro de 2003

UM ESTUDO SOBRE REJUVENESCIMENTO DE SOFTWARE EM SERVIDORES WEB APACHE

ELIETE MARCHIORO

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração em Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

(Coordenador do Curso)

Banca Examinadora

_____

Profª. Elizabeth Sueli Specialski, Dra.

(Orientador)

_____

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

_____

Prof. Alexandre Ramos, Dr.

“Não basta ensinar ao homem uma especialidade,
porque se tornará assim uma máquina utilizável,
mas não uma personalidade. É necessário que
adquira um sentimento, um senso prático daquilo que
vale a pena ser empreendido, daquilo que é belo,
do que é moralmente correto. A não ser assim,
ele se assemelhará com seus conhecimentos profissionais,
mais a um cão ensinado do que a uma
criatura harmoniosamente desenvolvida.
Deve aprender a compreender as motivações dos homens,
suas quimeras e suas angústias, para determinar com exatidão
seu lugar preciso em relação a seus próximos e à comunidade”

Albert Einstein

Este trabalho é para ***Deus***,
Ser Superior, que tudo vê e a ninguém desampara.
À ***Joares Gaboardi***, por compreender-me.
Para ***Roberta Pasqualli***, amiga do coração, desabafo
nas horas tristes
e sorrisos nas boas horas.
Mario Romano, por
acreditar que eu sou capaz.
A ***Rivalino Matias Júnior,*** pelo apoio, dedicação e
incentivo dispensado no desenvolvimento deste trabalho.
E, a minha orientadora, ***Elizabeth Sueli Specialski***
pela sua compreensão e companheirismo.

Aos meus pais Angelo e Neida
pelo amor e carinho dedicado.

Sumário

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASP	*Active Server Pages*
CGI	*Commun Gateway Interface*
DLL	*Dynamic Link Libraries*
DNS	*Domain Name Server*
FTP	*File Transport Protocol*
HTML	*Hypertext Markup Language*
HTTP	*Hyper Text Transfer Protocol*
IBM	*International Business Machine*
IIS	*Internet Information Server*
ISAPI	*Internet Server Application Program Interface*
ISP	*Internet Service Provider*
LANs	*Local Area Networks*
LDAP	*Lightweight Directory Access Protocol*
MIME	*Multipurpose Internet Mail Extensions*
MMC	*Microsoft Management Console*
MS-IIS	*Microsoft Internet Information Server*
NIS	*Network Information Service*
NNTP	*Network News Transfer Protocol*

ODBC	*Open Database Connectivity*
PHP	*Hypertext Preprocessor*
RAS	*Remote Access Server*
RDBMS	*Relational Database Management System*
RFC	*Reques For Coments*
RPM	Rotação Por Minuto
SMB	*Server Message Block*
SMTP	*Simple Mail Transport Protocol*
SO	Sistema Operacional
SQL	*Structured Query Language*
SSL	Secure Sockets Layer
SSR	*Symptom-based selective Software Rejuvenation*
TSR	*Time-based Software Rejuvenation*
URI	*Uniform Resource Identifier*
URL	*Universal Resource Locator*
URN	*Uniform Resource Name*
WAIS	*Wide Area Information Server*
WANs	*Wide Area Networks*
XRN	*eXpandable Resilient Networking*

RESUMO

Este trabalho apresenta um estudo investigativo sobre o envelhecimento e rejuvenescimento de software, visando avaliar suas causas e efeitos em servidores WEB Apache no contexto de um ambiente de provedor Internet. Desta forma, investigou-se soluções de rejuvenescimento de software a serem aplicadas para minimizar os efeitos deste envelhecimento.

São apresentados os principais conceitos sobre o envelhecimento de software, abordando como este processo ocorre, bem como suas principais conseqüências. Algumas das soluções, adotadas para este problema, são apontadas evidenciando o conceito de rejuvenescimento de software como forma de minimizar os efeitos deste envelhecimento.

Considerações sobre o envelhecimento de software em servidores WEB Apache em um ambiente de provedor Internet são apresentadas, ressaltando-se as principais características desses servidores. Descreve-se, ainda, um ambiente de testes, onde foram realizados experimentos sobre rejuvenescimento de software, bem como os resultados obtidos.

ABSTRACT

This work presents an investigative study of software aging and software rejuvenation, aiming to evaluate its cause and effects on Apache WEB servers running in an ISP environment. Also, was verified some software rejuvenation solution to be applied due to minimize the aging effects.

The main concepts on software aging are presented broaching the subject of how this process occurs and its main consequences. Some of the solutions, adopted for this problem, are pointed out, substantiating the concept of software rejuvenation as a form of minimizing the effects of aging.

Considerations on the software aging in Apache Web servers for ISP environment are presented, highlighting the principal characteristics of these servers. It also describes an environment of tests, where experiments for the software rejuvenation were undertaken, the results obtained through the realization of these tests and the evaluation of these results.

1 INTRODUÇÃO

O crescimento, a integração, a sofisticação da informática e a comunicação mudaram a sociedade e a economia mundial. Consumidores usam redes de computadores para identificar vendedores, avaliar produtos e serviços, comparar preços, dentre outros. As empresas usam a rede para transmitir ou até mesmo para administrar a produção e processos de reengenharia, agilizar a obtenção de produtos e serviços, localizar novos clientes, enfim administrar as operações internas e externas à organização (MESENBOURG, 2001).

A Internet tornou-se indispensável na vida cotidiana tanto das pessoas, como de empresas e instituições públicas, pois através dela tem-se acesso a notícias, pode-se comprar e vender produtos (comércio eletrônico), pode-se trocar mensagens através de correio eletrônico, ou *chats*, realiza-se transações bancárias, dentre outros. Contudo, é necessário que as aplicações envolvidas nesses processos ofereçam segurança, confiabilidade e disponibilidade, tanto nos serviços oferecidos na Internet quanto nos dados envolvidos no processo (MESENBOURG, 2001).

As empresas precisam de softwares confiáveis e disponíveis, assim como exigem softwares funcionando continuamente, mesmo que isso aumente a carga dos sistemas de computador ou das redes de comunicação. MARSHALL apud HONG (HONG, 2002) descreve que caso um software falhe, este normalmente apresenta conseqüências que vão desde impactos relacionados à economia até perda de vidas humanas.

Recentemente, o fenômeno de envelhecimento de software (GARG, 1998) demonstrou que as condições de erro realmente acontecem com o tempo de execução de processos de software e/ou carga submetida aos mesmos, resultando assim na degradação da performance do software e/ou falhas (*crash/hang*), sendo que, em alguns casos, ambas situações se manifestam. A não liberação de memória, bloqueio de arquivos não atualizados, dados corrompidos, fragmentação de espaço de armazenamento e acúmulo de erros são algumas das causas típicas desta degradação. Segundo Garg (GARG, 1998), o envelhecimento de software não acontece apenas naqueles softwares que são utilizados em grande escala. Yurcik (YURCIK, 2002) descreve que inclusive os usuários de computadores pessoais (PCs), com aplicações

típicas, comumente sofrem deste tipo de problema. Um dos exemplos mais usados para ilustrar o envelhecimento de software são os problemas que, muitas vezes, afetam equipamentos com o sistema operacional Windows, quando em utilização por um longo período de tempo, sem reinicialização, ou submetidos à exaustivos processos de computação, manifestam tais problemas. Yurcik (YURCIK, 2002) descreve que é uma prática comum os usuários reiniciarem seus computadores à noite, pois o sistema operacional Windows freqüentemente não funciona mais do que algumas semanas sem um reinício.

1.1 Objetivos

1.1.1 Geral

- Apresentar um estudo sobre o impacto do envelhecimento do envelhecimento de software e os benefícios causados pelo rejuvenescimento de software em servidores WEB Apache.

1.1.2 Específicos

- Investigar o estado da arte sobre envelhecimento e rejuvenescimento de software;
- Realizar um diagnóstico sobre o envelhecimento de software em servidores WEB Apache em um ambiente de provedor Internet;
- Identificar possíveis soluções de rejuvenescimento de software a serem aplicadas para minimizar os efeitos deste envelhecimento;
- Realizar experimentos em laboratório, de forma a permitir uma avaliação do envelhecimento e rejuvenescimento de software para o caso de provedores Internet;

1.2 Justificativa

Atualmente, a sociedade moderna vive grandes transformações no que tange à geração, distribuição e processamento de seus dados e informações. A Internet oferece uma nova dimensão para estas atividades, ocupando papel fundamental na nova sociedade da informação e do conhecimento.

Nos últimos anos, observa-se uma crescente mudança de paradigmas na engenharia de software, onde sistemas de computação, a cada vez mais são concebidos com a WEB como centro de suas arquiteturas, tendo a Internet como infra-estrutura básica para seu funcionamento.

Neste sentido, este trabalho visa contribuir com esta evolução, apresentando um estudo sobre envelhecimento e rejuvenescimento de software em servidores WEB, haja vista ser este um dos principais componentes de qualquer arquitetura de software baseada na Internet.

Neste trabalho, todos os experimentos foram realizados utilizando o servidor WEB Apache, devido principalmente a este ser o servidor WEB mais utilizado na Internet, segundo pesquisa da Netcraft (NETCRAFT, 2003). A Fig. 1 ilustra os resultados dessa pesquisa.

Além disso, outro fator que influenciou na sua escolha, foi o fato deste ser um software de código aberto, o que facilitou a realização dos experimentos de laboratório.

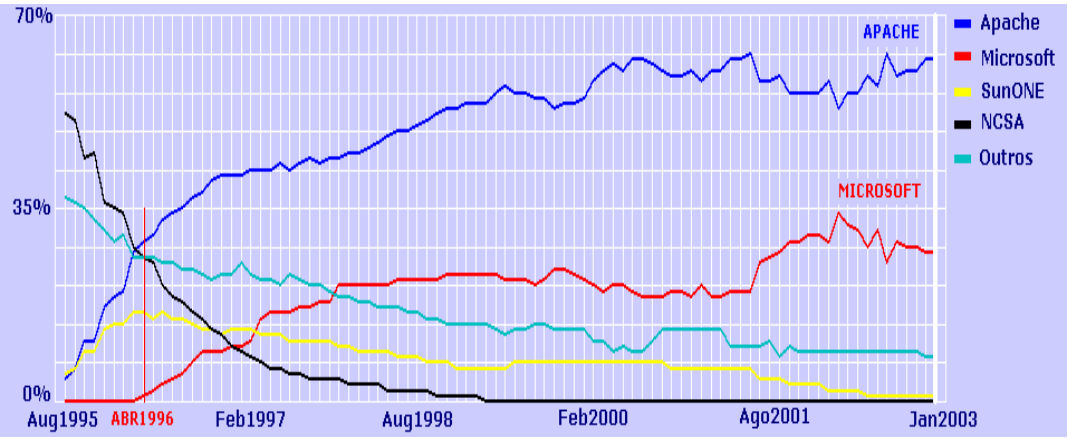

Figura 1. Percentual de utilização dos principais servidores WEB na Internet. Fonte: (NETCRAFT, 2003)

1.3 Metodologia

A metodologia utilizada para o desenvolvimento deste trabalho consiste em:

- Referencial teórico;
- O presente trabalho de pesquisa foi desenvolvido num provedor de internet ISP, localizado na cidade de Chapecó, denominado de SuperIP;
- Definição do ambiente de testes;
- Análise estatística.

1.4 Limite de Escopo

O desenvolvimento deste trabalho de pesquisa foi embasado em referencial teórico utilizando os procedimentos metodológicos específicos. O presente estudo apresenta limitações que são destacadas a seguir.

O presente estudo, embora embasado em fundamentos teóricos de estudiosos do assunto, é especifico para provedores de internet que utilizam as mesmas configurações de softwares listadas abaixo. Desta forma, os resultados obtidos não podem ser generalizados para outros provedores de Internet, que utilizem configurações diferentes das utilizadas no presente trabalho.

Os dados foram obtidos através análise documental nos logs do servidor WEB APACHE.

Como delimitação do estudo observou-se os logs do servidor WEB APACHE do SuperIP, no período de agosto/02 a junho/03. Através dos dados obtidos realizou-se uma análise estatística, a qual expressa a quantidade de requisições que deveriam ser utilizadas através da ferramenta de geração de tráfego de forma parametrizada.

No presente trabalho de pesquisa, na fase de testes, utilizou-se o seguinte cenário, uma máquina servidora e 3 máquinas clientes, conforme descrição a seguir:

- Sistema Operacional Linux, Conectiva 8.0;
- Servidor WEB Apache 1.3.27 (PHP+MySQL+SSL+SSH);

- Configurações do Hardware:
  - **Máquina Servidor** - Pentium III 500 MHZ, 256 MB RAM, HD 9 GB, Placa de Rede 3Com 10/100, Placa de rede Realtec 10/100;
  - **Máquinas Cliente**, ou seja, as geradoras de tráfego - K6 II 350 MHZ, 128 MB RAM, HD 4 GB, Placa de Rede Realtek 10/100;
  - 1 HUB/Switch 10/100;
  - Nobreak 1.5 KVA.

1.5 Organização do Trabalho

O capítulo 2 faz uma revisão dos conceitos de um ambiente WEB, além de uma descrição detalhada dos modelos mais utilizados na Internet, bem como uma descrição do protocolo de transferência de hiper texto o HTTP (*Hyper Text Transfer Protocol*) e suas versões.

No capítulo 3 são apresentados os principais conceitos sobre o envelhecimento de software, com o objetivo de se aprofundar neste assunto. Neste são detalhes do processo de envelhecimento de software, além de apontar as principais conseqüências e evidenciar algumas das soluções que são adotados para o problema. Além de apresentar o conceito de rejuvenescimento de software, o qual está intimamente ligado ao problema do envelhecimento de software.

O capítulo 4 apresenta uma descrição do ambiente de teste, o qual subsidiou os experimentos de implementação deste trabalho, e mostra o resultados obtidos.

As considerações finais são apresentadas no capítulo 5.

2 AMBIENTE WEB

Neste capítulo são apresentados os conceitos relacionados a um ambiente WEB, os modelos e arquiteturas utilizados, o protocolo de comunicação entre aplicações e as características dos servidores WEB.

2.1 Modelos e Arquiteturas

A seguir serão apresentadas as principais abordagens envolvendo aplicações baseadas na WEB.

2.1.1 Modelo Cliente-Servidor

A arquitetura WEB foi inicialmente concebida baseada no modelo cliente-servidor. O modelo cliente/servidor é muito usado em bases de dados, onde se tem o processamento entre duas entidades denominadas: cliente e servidor, cuja relação dar-se-á entre dois programas (aplicações) de computador no qual o cliente faz uma solicitação de serviço para o servidor e este responde a esta solicitação (pedido). Tais aplicações podem ser encontradas dentro de uma mesma máquina ou em máquinas distintas, dentro de uma rede de computadores (Internet ou Intranet) (FERREIRA, 2002 e EDUCARE, 2002).

Aplica-se esta arquitetura tanto para rede local (LANs – Local Area Networks), quanto para redes de longas distâncias (WANs – Wide Area Networks), visto a sua flexibilidade. Um computador qualquer pode ser, simultaneamente, cliente e servidor dependendo da circunstância, tem-se com exemplo a Internet que é baseada no paradigma de Cliente/Servidor. A Fig. 2 ilustra o modelo acima descrito, onde a aplicação servidor: aguarda novas mensagens, executa serviços e retorna resultados para a aplicação cliente. O cliente por sua vez estabelece conexão com a aplicação servidor, envia mensagens para o servidor e aguarda resposta das mensagens. (CALDEIRA, 2001).

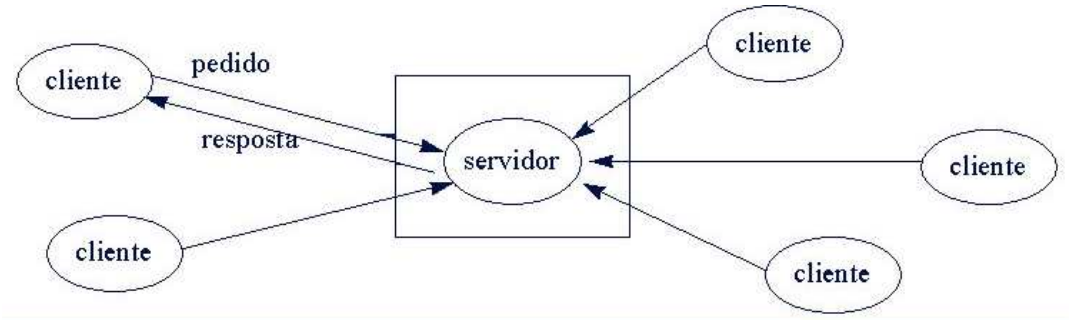

Figura 2. Modelo Cliente/Servidor. Fonte: (MESQUITA, 2002)

Mesquita (MESQUITA, 2002) descreve que o modelo Cliente/Servidor apresenta as seguintes características básicas:

- Como sendo o mais utilizado para aplicações distribuídas e não para aplicações paralelas;
- Um processo *servidor* está sempre à espera de comunicação;
- O processo *cliente* tem a iniciativa de começar a comunicação quando deseja algum serviço.

A Internet, como citado anteriormente, usa este modelo como base para suas operações. O computador cliente requisita uma operação a outro computador (servidor), que responde a solicitação, enviando o que foi pedido. A Internet apresenta vários tipos de servidores, tais como: *Mail Servers, WEB Servers, FTP Servers, File Servers*, etc (NOVISNET, 2000).

2.1.2 Arquitetura *N-tier*

Atualmente, uma abordagem que vem crescendo é cliente-servidor multicamadas (*N-tier*). Esta é uma arquitetura projetada para usar recursos de um ambiente em vários níveis ou camadas, aumentando principalmente a segurança e disponibilidade do sistema. A *N-tier* é uma evolução do modelo cliente-servidor, onde o processamento e a lógica são distribuídos entre as camadas física e lógica. Estas camadas podem estar

conceitualmente separadas, podendo ser localizadas fisicamente em servidores diferentes (GARRETT, 2001).

Aplicações que utilizam a tecnologia cliente-servidor com multi-camadas possuem capacidade para distribuir as cargas de trabalho de processamento por múltiplos servidores de dados e de negócios (INTERQUADRAM, 2002).

Grande parte das organizações que utilizavam a arquitetura *mainframe* migraram para a arquitetura cliente/servidor e posteriormente para a arquitetura *N-tier*. Contudo, em alguns casos, esta migração pode ocorrer diretamente para a *N-tier.*

Uma aplicação projetada para uma arquitetura *N-tier* é subdividida em três camadas: Apresentação, Lógica e Dados. Laboims (LABOLIMS, 2002) conceitua a camada de Apresentação como a que está em contato com o cliente, por exemplo, uma página da Internet. Esta pode conter código (*scripting*) que servirá para, por exemplo, verificar os dados introduzidos, evitando deste modo o processamento desnecessário na consulta da base de dados. A Camada Lógica é considerada a camada intermediária e esta contém o código, ou seja, os mecanismos necessários para o processamento das requisições. Já a Camada de Dados é onde encontra-se a base de dados do sistema.

No modelo *N-tier* a camada cliente nunca se comunica diretamente com a camada servidora, os dados são intermediados por uma camada central (*middle tier),* como ilustram as Fig. 3 e 4 (CHAPPLE, 2001 e LORRIMAN, 2000). Este modelo apresenta vários benefícios:

- A camada central (*middle tier)* fornece uma abstração às demais camadas. As aplicações clientes são desenvolvidas para se comunicarem com a camada central, uma vez que não dependem do tipo de servidor que armazena os dados. Neste caso, o servidor poderia ser mudado, por exemplo, de Oracle para um outro servidor SQL, por exemplo, MySQL e, com isso, apenas a camada central exigiria alterações.
- A base de dados desenvolvida no modelo de *N-tier* fornece um método que separa as responsabilidades dos desenvolvedores.

- Finalmente, as implementações desenvolvidas no modelo de *N-tier* podem ser projetadas para obter melhor desempenho nas operações em redes.

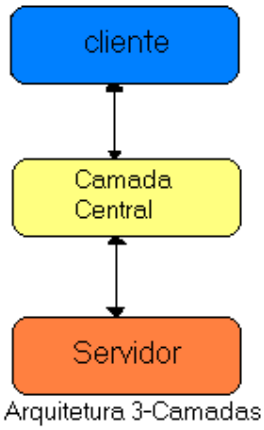

Figura 3. Arquitetura de *N-tier* proposto por Chapple. Fonte: (CHAPPLE, 2001)

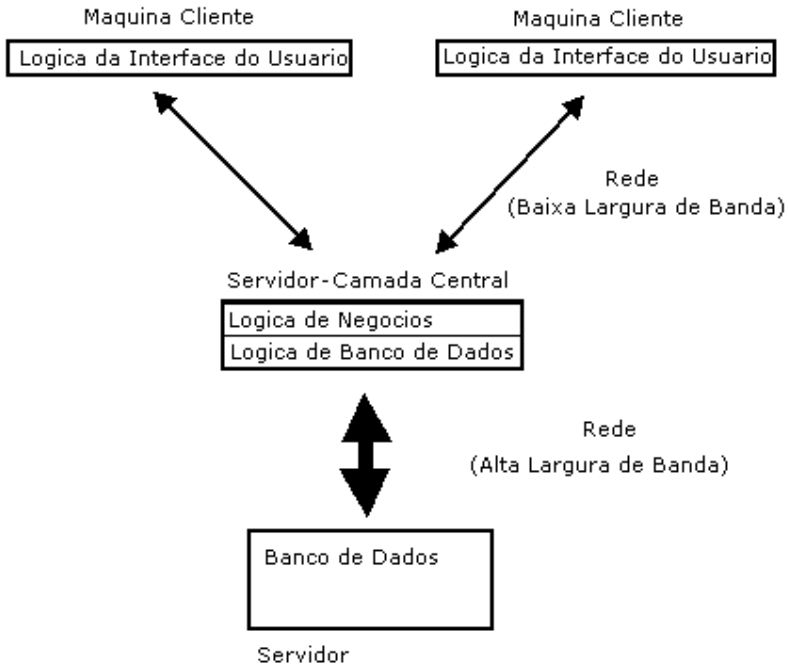

Figura 4. Arquitetura de *N-tier* proposto por Lorriman. Fonte: (LORRIMAN, 2000)

O ambiente de *N-tier* (multi-camadas) possui num todo sete camadas, como descrito abaixo: (NOORDERGRAAF, 2000).

1. Servidor WEB / Camada Externa

2. Camada Servidor de Aplicação
3. Camada Servidor de Banco de Dados
4. Extranet / Camada Provedor de Serviço
5. Camada de Armazenamento em Rede
6. Camada de Backup
7. Camada de Gerenciamento

Noordergraaf (NOORDERGRAAF, 2000) apresenta e descreve o diagrama das sete camadas ilustrado na Fig. 5.

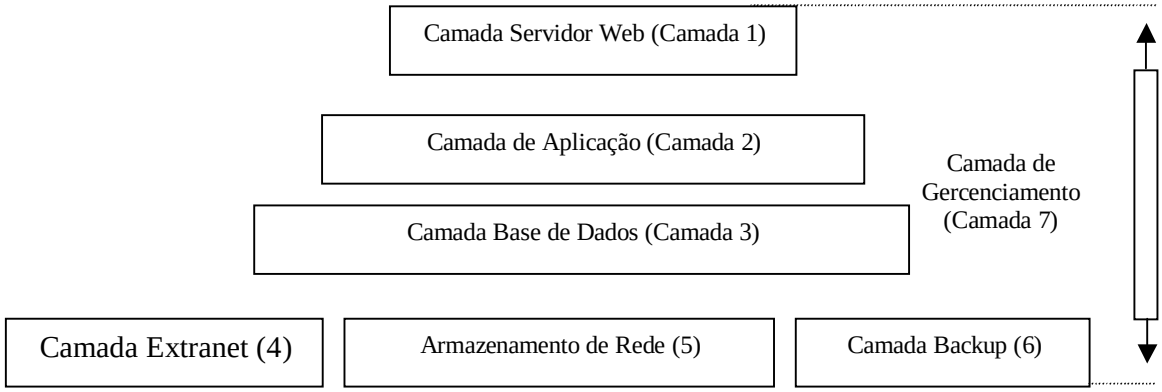

Figura 5. Diagrama multi-camadas. Fonte: (NOORDERGRAAF, 2000)

Camada do Servidor WEB - Esta camada é composta por serviços ligados diretamente a Internet, tais como: DNS, SMTP, HTTP e HTTPS. Os sistemas desta camada são suscetíveis a ataques, devido à vulnerabilidade a que estão expostos na Internet.

Camada do Servidor de Aplicação - Esta camada apresenta tanto os sistemas da camada Servidor WEB localizado entre a camada Servidor de Banco de Dados.

Camada Servidor de Banco de Dados – Esta camada possui o conteúdo principal do ambiente, mantido pelo Servidor de Banco de Dados, servidores de DNS internos, entre outros. Os servidores desta camada fornecem serviços a todas as solicitações advindas tanto do Servidor de Aplicação como do próprio Servidor de Banco de Dados.

Camada de Armazenamento de Rede – Esta camada foi incluída recentemente no modelo devido ao amplo crescimento de dispositivos de armazenamento de rede baseados em IP.

Camada de Backup – Esta camada fornece backup aos dados críticos para o suporte de sistema. Entretanto, somente os dados que sofreram alguma alteração, é que são incluídos no backup.

Extranet / Camada Provedor de Serviço – Informações e serviços para fontes externas são requeridas em quase todas as aplicações de comércio eletrônico (*e-commerce).* As informações podem ser do tipo simples, como uma informação contida na etiqueta que deve ser repassada para o estoque, ou complexa, como uma operação de inventário. Indiferente da informação requerida está é essencial para o sistema e este acesso deve ser seguro.

Camada de Gerenciamento – É considerada a mais importante e a mais vulnerável dentro desta arquitetura. Esta camada contém o software de gerenciamento de toda a rede, normalmente baseado em SNMP (*Simple Network Manegement Protocol)*, ela torna-se vulnerável porque oferece acesso de controle de todo ambiente.

2.2 Protocolo HTTP (*Hyper Text Transfer Protocol)*

Para o transporte da informação entre o servidor e o cliente WEB é utilizado um protocolo de comunicação denominado HTTP (*HyperText Transfer Protocol*) (YEAGER, 1996 e STEIN, 1995).

O protocolo HTTP é um protocolo do nível de aplicação que possui objetividade e rapidez necessária para suportar sistemas de informação distribuídos, cooperativos e de hipermídia. Este protocolo é usado na Internet. A primeira versão do protocolo HTTP foi HTTP/0.9, sendo esta uma versão de protocolo simples que transferia dados através da Internet. Logo após surgiu a versão HTTP/1.0, definida pela RFC 1945, esta versão

foi melhorada e o protocolo permitia mensagens no formato MIME. Devido à algumas limitações da versão HTTP/1.0 surgiu a versão HTTP/1.1 como descrita na RFC 2616.

A RFC 1945 descreve que Sistemas de informação requerem maior funcionalidade que a simples recuperação de informação, incluindo pesquisa, atualização *front-end*, autenticação, dentre outras. O HTTP permite um conjunto aberto de métodos para ser usado para indicar o propósito de uma requisição. Ele constrói na sua disciplina de referência provida pela *Uniform Resource Identifier* (*URI*), uma localização (*URL*) ou nome (*URN*) para indicar em qual recurso um determinado método deve ser aplicado. As mensagens são passadas em um formato similar ao usado pelo *Internet Mail* e o *Multipurpose Internet Mail Extensions (MIME).*

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

O protocolo HTTP possui os seguintes objetos de comunicação (RFC 2616):

***Connection* (conexão):** Circuito virtual na camada de transporte estabelecido entre dois programas aplicativos para o propósito de comunicação.

***Message* (mensagem)**: A unidade básica de comunicação HTTP, consistindo de uma seqüência estruturada de octetos de sintaxe definida e transmitida via conexão.

***Request* (pedido):** Uma mensagem de requisição HTTP.

***Response* (resposta):** Uma mensagem de resposta HTTP.

***Resource* (recurso):** Um objeto de dados da rede ou serviço que pode ser identificado por uma URL.

***Entity* (entidade):** Uma representação particular de um recurso de dados, ou resposta de um recurso de serviço, que pode ser incluído numa mensagem de pedido ou de resposta.

***Client* (cliente):** Um programa aplicativo que estabelece conexões para o propósito de enviar pedidos.

***User agent* (agente usuário):** O cliente que inicia o pedido. Normalmente "*browsers*", editores, "*spiders*" (robôs que pesquisam através da rede), ou outras ferramentas de usuário final.

***Server* (servidor):** Um programa aplicativo que aceita conexões para atender pedidos e enviar respostas.

***Origin server* (servidor de origem):** O servidor no qual um dado recurso reside ou é para ser criado.

***Proxy* (procuradores):** Um programa intermediário que atua duplamente como servidor e como cliente para fazer pedidos em nome de outros clientes. *Proxies* são geralmente usados como filtros de clientes e como auxiliares de aplicações para manipular pedidos via protocolos não implementados pelo agente usuário.

Gateway: Um servidor que atua como intermediário para outros servidores. Gateways são geralmente usados como portais clientes através de *firewalls* e como tradutores de protocolos em sistemas não-HTTP.

Cache: Um local de armazenamento de mensagens de resposta do programa e o subsistema que controla o armazenamento, recuperação e exclusão destas mensagens. O acesso fácil às mensagens armazenadas reduz o tempo de resposta e consumo de tráfego na rede no futuro, em pedidos equivalentes.

Qualquer programa pode ser capaz de ser cliente ou servidor. O uso destes dois termos refere-se apenas ao papel que está sendo executado pelo programa para uma conexão particular, ao invés de programas de uso geral. Assim também, cada servidor

pode atuar como servidor de origem, proxy ou gateway, mudando o comportamento baseado na natureza de cada pedido.

A principal característica do protocolo HTTP é ser um protocolo aberto e especializado na transmissão de documentos WEB sob a Internet. O protocolo HTTP é baseado no paradigma cliente; servidor, ou seja, um cliente estabelece uma conexão com um servidor e envia um pedido ao servidor, o mesmo analisa e responde a solicitação requerida. A conexão deve ser estabelecida antes de cada pedido de cliente e encerrada após a resposta. Este protocolo denomina-se como *stateless*, ao contrário de diversos outros protocolos *Internet* que são *stateful* (RFC 2616). Como descrito anteriormente, o cliente WEB envia o pedido de uma operação ao servidor WEB, este atende o pedido e logo em seguida é encerrada a conexão, ou seja, nenhuma informação sobre a solicitação do usuário é mantida no servidor WEB. A característica *stateless* do protocolo HTTP proporciona eficiência e velocidade necessárias a sistemas de informação *hipermídia* distribuídos como a WEB, no entanto desencadeia vários problemas relacionados ao desenvolvimento de aplicações que exijam, por exemplo, a identificação do usuário (SALLA SÁ, 2000). As mensagens seguem o formato da RFC822, atualizada pelas RFC1327 e RFC0987.

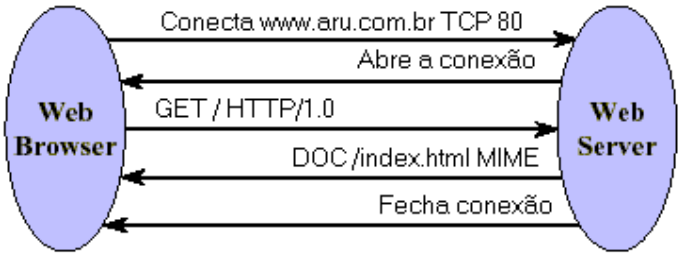

Figura 6. Etapas de uma conexão HTTP. Fonte: (GUEDES, 2001)

A Fig. 6 exibe as etapas da comunicação do protocolo HTTP entre WEB Browser (cliente) e WEB Server (servidor). LIMA apud SALLA (SALLA SÁ, 2000) descreve que inicialmente o cliente envia uma mensagem de sincronização para o servidor pedindo abertura de conexão. O servidor responde se pode ou não aceitar ao pedido. Caso aceite, a conexão é estabelecida. Uma negação do pedido pode indicar que o servidor está sobrecarregado e não consegue tratar novos pedidos de conexão que

chegam. Normalmente, existe um número máximo de pedidos pendentes. Um pedido de conexão pode ser recusado se este limite tiver sido atingido.

A partir do momento em que for estabelecida uma conexão, o cliente envia uma mensagem com a requisição feita pelo usuário, representada por uma *URL*. Após o servidor processar o pedido, uma mensagem de resposta é enviada ao cliente, na seqüência o servidor encera a conexão com o cliente.

Os principais métodos de manipulação de dados utilizados pelo protocolo HTTP são: POST e GET, conforme mostra a Tabela 1.

Tabela 1. Diferença entre os métodos GET e POST - Fonte: (CASTRO, 1997)

POST	GET
Os dados inseridos no formulário fazem parte do corpo da mensagem enviada para o servidor	Os dados de entrada fazem parte do URL associado à consulta enviada para o servidor (por exemplo, nas consultas a catálogos do tipo Yahoo e Google)
A cadeia de caracteres de entrada é lida como entrada padrão de comprimento CONTENT_LENGTH	A cadeia de entrada é colocada na variável de ambiente QUERY_STRING
É possível transferir grande quantidade de dados	Suporta somente até 128 caracteres
	Como os dados da consulta fazem parte do URL, ela pode ser encapsulada em um URL de link ou *bookmark*
	Os dados de entrada são copiados no registro de acessos ao site; portanto, informações que exigem segurança não devem ser manipuladas por este método

Um exemplo dos métodos apresentados acima pode ser expresso pela sintaxe apresentada a seguir:

GET: `"GET /home/index.php HTTP/1.1"`

`"GET /home/index.php HTTP/1.0"`

POST: `"POST /horde/imp/redirect.php HTTP/1.1"`

Relativo à versão 1.1 do protocolo HTTP permanecem as mesmas operações que na versão HTTP/1.0. O protocolo garante que, se *browsers* e servidores possuírem versões diferentes, mesmo assim eles se comunicarão corretamente. Por exemplo, se um servidor rodando HTTP/1.1 encontra uma versão inferior, ele ajusta sua resposta ao protocolo encontrado.

As novas características que foram incluídas ao protocolo HTTP/1.1 são listadas a seguir e estão descritas na (RFC 2616):

- A identificação de nomes de hosts;
- Negociação de conteúdo;
- Conexões persistentes;
- Transferências de arquivos particionados;
- Escalas de bytes (os *browsers* podem requisitar partes de documentos);
- Suporte para proxies e cachs;
- Novos métodos de requisição como OPTIONS, TRACE, DELETE, PUT;
- O uso de Autenticação de Acesso por Resumo Criptográfico (Digest Access Authentication).

Dentre as novas características da versão 1.1, vale destacar o uso de conexões persistentes, haja vista ser um conceito a ser explorado no desenvolvimento deste trabalho. Esta característica tem por objetivo enviar diversas solicitações sobre uma única conexão TCP.

Obter um documento HTML tipicamente envolve diversos pedidos HTTP para o servidor WEB (como por exemplo, busca de imagens embutidas). Nas versões HTTP/1.0 e anteriores, as conexões TCP eram fechadas depois que cada pedido e resposta eram realizados; com isso, cada recurso para ser acessado necessitava de sua própria conexão. A versão HTTP/1.0 suporta múltiplas solicitações para um mesmo

servidor, contudo criando uma conexão TCP para cada objeto (documento ou imagem). Abrir e fechar conexões TCP, entretanto, faz com que aumente consideravelmente o tempo de processamento, a largura de banda e a alocação de memória, (RFC 2616).

Na versão HTTP/1.1 existe a transferência múltipla de objetos sobre uma única conexão TCP [NETWORK, 2003], reduzindo, assim, a sobrecarga existente na versão anterior deste protocolo. Mogul (MOGUL, 1995) afirma que as conexões persistentes podem reduzir o tempo de resposta, a sobrecarga do servidor (*overhead*) e da própria rede.

2.3 Servidor WEB

Conforme ilustrado na Fig. 7, um servidor WEB é um software que implementa o protocolo HTTP e atua no papel de servidor, respondendo à requisições de clientes HTTP.

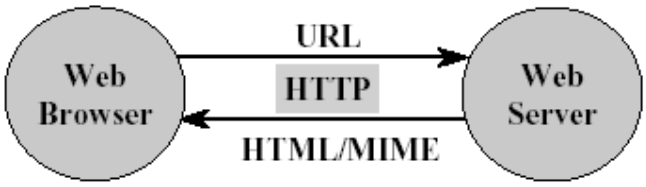

Figura 7. Modelo genérico Cliente/Servidor. Fonte: (GUEDES, 2001)

Exemplos de servidores WEB são o Apache, iPlanet, IIS, dentre outros (TELEPAC, 2003).

Segundo Hessel (HESSEL, 2001), um cliente HTTP (*browser* cliente WEB) se comunica com um servidor HTTP (Servidor WEB) requisitando arquivos. Normalmente estes arquivos encontram-se no formato HTML (*Hypertext Markup Language*) e, ao receber o arquivo HTML, o cliente verifica cada linha do arquivo, solicitando ao servidor HTTP os arquivos necessários. Esse é o modelo básico e estático de apresentação de páginas na Internet, ou seja, o usuário não faz nenhuma interação,

apenas as aplicações cliente e servidor é que são encarregadas do processo como um todo.

Além das páginas estáticas existe a possibilidade de interação e dinamismo executado sobre as páginas. Neste grupo encontram-se páginas baseadas em *script* como o Microsoft *Active Server Pages* e o PHP (*Hypertext Preprocessor*). Outras opções tais como as ferramentas *ColdFusion* e Macromedia *Ultradevem* também podem ser utilizadas. Através desses sistemas tornou-se possível um servidor WEB receber uma informação do cliente, processá-la, e retornar uma página HTML construída em tempo real no servidor WEB. O modelo de páginas dinâmicas é ilustrado na Fig. 8.

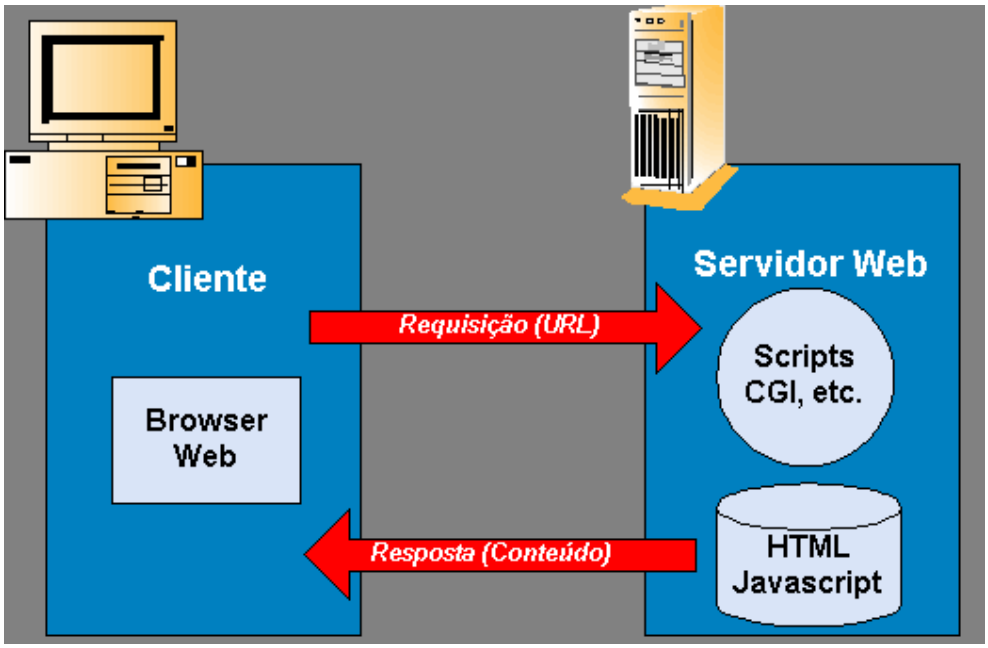

Figura 8. Modelo de um Servidor WEB com páginas dinâmicas. Fonte: (SILVA, 2000).

2.3.1 Características dos Servidores WEB

No trabalho de (COSTA, 2001), são apresentadas as características de dois servidores WEB, os quais são: o Servidor Apache e o Servidor MS-IIS. Estas características são apresentadas na Tabela 2 e as suas funcionalidades são apresentadas na Tabela 3.

Tabela 2. Características dos servidores WEB Apache e MS-IIS (COSTA, 2001)

Servidor Apache	**Servidor MS-IIS**
Modelo de desenvolvimento *open source*	Modelo de desenvolvimento proprietário, ou seja, código fechado
Elevada confiabilidade	Razoável confiabilidade (dependente do SO)
Excelente desempenho	Bom desempenho
Suportado em várias plataformas (NetBSD, Digital Unix, BSDI, AIX, OS/2, SCO, HPUX, Windows NT, Linux, FreeBSD, IRIX, Solaris)	Suporte somente para a plataforma Windows
Implementa o protocolo HTTP 1.1	Implementa o protocolo HTTP 1.1
Extensível através de módulos, ou seja, o servidor WEB Apache pode ser carregado por módulos visto que cada módulo consome memória, assim o uso de memória global utilizada pelo Apache será reduzido significativamente	Extensível através de DLL's
Administração através de linha de comandos	Administração e depuração através de interface gráfica (MMC)

Tabela 3. Funcionalidades dos servidores WEB mais utilizados na Internet

Funcionalidades	Apache	*MS IIS*
Linguagens internas suportadas	*Perl*, PHP, ASP (em curso), *Python*	ASP, *ActiveX*
CGI	Todo o tipo de linguagens suportadas pelo Sistema Operacional	PHP, *Perl*, linguagens suportadas pelo Sistema Operacional
Bases de dados	Todas através de SQL, ODBC	Através de SQL, ODBC
Controle de acesso e encriptação	Ficheiros *password*, NIS, LDAP, certificados e SSL	SMB, LDAP, certificados e SSL
Outros	Manipulação flexível de URL's	Uso de ISAPI DLL's para manipular URL's

2.4 Como funciona um Servidor WEB

De forma geral, os servidores WEB dedicam seu tempo em duas tarefas primordiais que são:

- Atendimento às conexões dos clientes;
- Processamento das requisições dos clientes.

2.4.1 Atendimento às conexões dos clientes

Como descrito anteriormente uma máquina cliente, ou seja um *browser*, necessita fazer uma série de conexões ao servidor WEB para montar localmente uma página. Dependendo da versão do HTTP deve ser feita uma conexão para cada elemento, como textos, figuras ou elementos de multimídia

A seqüência de estabelecimento de uma conexão TCP/IP entre um servidor WEB e seu cliente é mostrada na Fig. 9 e ocorre da seguinte maneira (BANGA, 1997):

1 - O servidor fica "ouvindo" (*listening*) por possíveis conexões HTTP, através de um *socket* na sua porta 80.
2 - O cliente envia uma requisição de conexão, iniciando o mecanismo de *three way handshake* através do envio de um pacote TCP SYN ao servidor.
3 - O servidor responde à requisição do cliente, devolvendo um pacote TCP SYNACK, criando um novo *socket* para a conexão e o insere numa fila SYNRCVD (*syn´s* recebidos), onde fica no aguardo da confirmação.
4 - Se tudo correr bem, o cliente envia a confirmação através de um pacote TCP ACK.
5 - Ao receber o pacote TCP ACK do cliente, o servidor remove o novo *socket* da fila SYN-RCVD e o insere na ACCEPT QUEUE (fila de *sockets* aceitos).

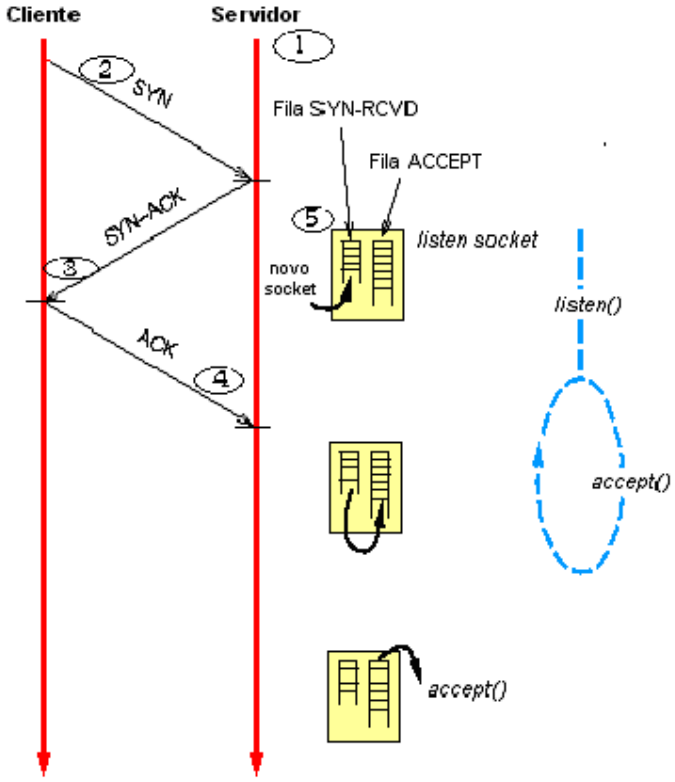

Figura 9. Atendimento a conexões num servidor WEB. Fonte: (BANGA, 1997)

Segundo Mogul (MOGUL, 1995), apesar da simplicidade e facilidade de implementação de aplicações através do protocolo HTTP, a adição de latência desnecessária pela criação de uma nova conexão TCP, para cada requisição, faz com que a utilização dos recursos de rede e dos servidores seja ineficiente. Isto se deve ao fato de as conexões serem *não-persistentes*, ou seja, não ficarem mantidas até o completo atendimento de uma requisição que deva retornar ao *browser* um número determinado de elementos de uma página *web*. O que contribui para a ineficiência são os chamados RTT ou *Round-Trip Times*: tempos de ida-e-volta das requisições do cliente até o recebimento das respostas. Uma das idéias é justamente trabalhar com conexões *persistentes*, visando reduzir a quantidade de períodos de RTT nas requisições dos clientes aos servidores, o que, conseqüentemente, reduziria a carga da rede e a utilização de recursos dos servidores, bem como melhoraria a performance percebida pelos usuários.

O atendimento das conexões de rede e seu conseqüente encaminhamento para que as requisições sejam atendidas, sempre terá como limite a capacidade dos servidores. Com efeito, um dos desafios mais críticos de servidores *web* é lidar com picos de carga. Sempre que for atingida a plena capacidade de atendimento de conexões, obviamente o servidor iniciará um processo de recusa de novas conexões até que seja restabelecida a sua capacidade de atendimento.

Em (BANGA, 1997) são analisados alguns problemas que podem ocorrer no tratamento de conexões por parte de servidores *web* UNIX. Nestes sistemas, a variável de *kernel*, denominada *somaxxcon*, determina o tamanho máximo do *backlog* em um *socket* de conexão. O *backlog* é limitado pela soma dos tamanhos das filas SYN-RCVD e ACCEPT. Quando a soma de pacotes SYN exceder 1,5 vezes o tamanho do *backlog*, o servidor TCP passa a recusar estes pacotes.

Do lado do cliente TCP, quando este não recebe o pacote SYN-ACK do servidor, inicia um processo de retransmissão de pacotes SYN até que receba a resposta SYN-ACK ou até que o tempo de estabelecimento de conexão seja atingido.

O tamanho médio da fila SYN-RCVD depende do tempo de ida-e-volta de uma mensagem (*RTT - round-trip time*) entre o servidor e seus clientes, e da taxa de requisição de conexões. Isto acontece devido ao fato de que um *socket* permanece nesta fila por um período de tempo igual ao RTT. Tempos de ida-e-volta muito longos e altas taxas de requisição contribuem para o aumento desta fila.

O tamanho da fila ACCEPT, por sua vez, depende da rapidez com que o servidor HTTP processa as chamadas de *accept()* - ou taxa com que disponibiliza conexões, e da taxa de requisições. Se um servidor estiver operando no máximo da sua capacidade, este não poderá executar chamadas *accept()* na velocidade suficiente para manter a taxa de requisição de conexões, fazendo com que a fila cresça.

O estado de cada *socket* é mantido em uma estrutura de dados denominada de Bloco de Controle do Protocolo - *PCB* (*Protocol Control Block*), e o protocolo TCP mantém uma tabela com todos os PCB´s ativos no sistema. Os PCB´s são criados juntamente com os *sockets*, como resultado de alguma chamada de sistema ou do

estabelecimento de alguma conexão. A existência de um PCB cessa no exato instante em que é executada uma chamada *close()* ou no momento em que chegar um pacote de controle FIN. Quando um pacote FIN é enviado, antes que retorne um pacote FIN-ACK e seja confirmado, o PCB é mantido por um intervalo igual ao tempo de *time-wait* da implementação. O propósito do *time-wait* é permitir retransmissões de ACK de finalização aos correspondentes FIN quando o ACK original é perdido, e também para permitir detecção de segmentos TCP duplicados ou atrasados.

Um dos problemas mais conhecidos das implementações tradicionais do protocolo TCP/IP, que limita a vazão dos servidores *web*, são os pequenos valores máximos atribuídos à variável *somaxconn*. Este limite, assim configurado, pode ser atingido rapidamente, fazendo com que as filas ACCEPT e SYN-RCVD lotem, provocando a recusa de conexões, sem necessidade. Se o limite de *somaxconn* for muito baixo, uma conexão poderá ser recusada mesmo se o servidor possuir suficientes recursos para atender à requisição. Mesmo no caso de uma longa fila de ACCEPT, é preferível aceitar a conexão, a menos que a fila já contenha carga suficiente para manter o servidor ocupado, no mínimo pelo intervalo inicial de retransmissão (cerca de 6 segundos, considerando o Unix FreeBSD 4.4). Muitos fornecedores de sistemas Unix aumentaram o tamanho da variável *somaxxcon*. No Digital Unix foi fixado um valor de 32.767, enquanto que o Solaris apresenta valor de 1.000 (BANGA, 1997).

No ANEXO 01 estão descritas as principais diretivas de configuração do Servidor WEB Apache, o qual faz partem do escopo deste trabalho.

2.4.2 Processamento das requisições dos clientes

À medida em que as conexões dos clientes vão sendo estabelecidas, outro processo no servidor, assíncrono, fica atendendo às requisições que chegam, da seguinte forma (BANGA, 1997):

1. O servidor passa a conexão para um processo auxiliar;
2. O processo auxiliar lê a requisição HTTP recebida do cliente na conexão;

3. A requisição é atendida (envio da página inicial HTML da página, execução de uma *query* em um banco de dados ou algum outro procedimento);
4. O processo encapsula a resposta em um pacote TCP e envia para o cliente;
5. O processo finaliza (fecha) a conexão, liberando o socket.

Na Fig. 10 é possível ver a seqüência de atendimento de requisições HTTP a partir do *browser* de um cliente.

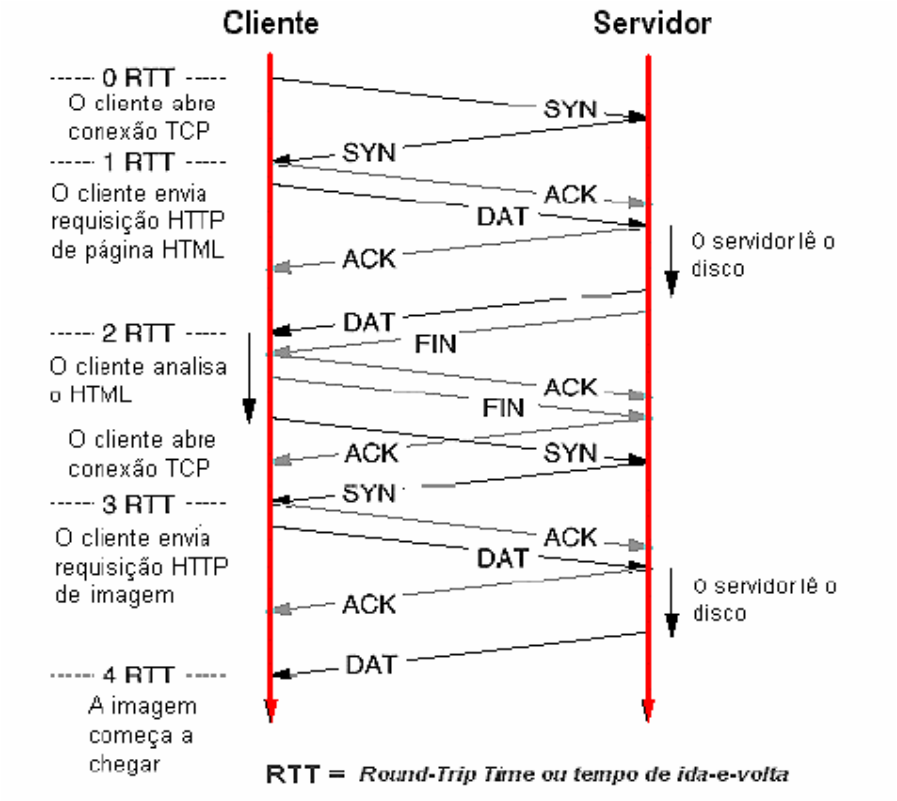

Figura 10. Seqüência completa das requisições HTTP. Fonte: (MOGUL, 1995)

Neste capitulo foi abordado conceitos envolvendo o ambiente WEB, o qual vem subsidiar o desenvolvimento dos próximos capítulos do presente trabalho de pesquisa. Onde serão apresentados os principais conceitos de envelhecimento e rejuvenescimento de software, em especial servidor WEB.

3 ENVELHECIMENTO E REJUVENESCIMENTO DE SOFTWARE

Neste capítulo são apresentados os conceitos envolvendo envelhecimento e rejuvenescimento de software, dando ênfase às causas e conseqüência do envelhecimento de software, soluções atuais e em desenvolvimento, envelhecimento de software em servidores WEB, rejuvenescimento de software em servidores WEB e as faltas encontradas no servidor WEB Apache.

3.1 Envelhecimento de Software

Atualmente, existem diversas pesquisas na área de envelhecimento de sistemas de software. Algumas destas enfocam o envelhecimento de sistemas de software em termos de suas funcionalidades, estando mais relacionadas com as diversas versões do software. Neste enfoque são considerados, principalmente, os aspectos de interface e de configurações.

Outra linha de pesquisa nesta área trata do envelhecimento dos processos de software em execução em um sistema computacional. Esta segunda linha de pesquisa trata o envelhecimento de software *(software aging*) como o estado de um sistema de software, em termos de seus processos[1] que degrada com o seu tempo de execução, conforme definido por (HUANG,1995).

Crowell (CROWELL, 2001) e Gross (GROSS, 2002a) abordam a existência das duas linhas de pesquisas citadas anteriormente, no entanto seus enfoques estão voltados para a linha de pesquisa apresentada por (HUANG, 1995) e (CASTELLI, 2001).

Este trabalho também adota o conceito de envelhecimento de software proposto por (HUANG, 1995).

[1] Neste trabalho **processo de software** significa uma instância de um programa executável, carregada em memória RAM, e sendo executado pelo processador.

O fenômeno de envelhecimento de software já foi relatado por diversos estudos (GARG, 1998, GARG, 1998a, HUANG, 1995 e VAIDYANATHAN, 2001). Nestes, observou-se que uma vez que o software é iniciado, muitas vezes as condições de faltas que ocorrem se acumulam gradualmente com o tempo, conduzindo à uma degradação do desempenho ou à faltas transientes[2], podendo em alguns casos ocorrerem ambos problemas. Estas faltas podem ser do tipo "queda" ou que resultam na inconsistência dos dados por causa do envelhecimento. As conseqüências típicas do envelhecimento são a degradação do sistema, consumo excessivo de memória, bloqueio indevido de arquivos, dados corrompidos, fragmentação do espaço de armazenamento, acumulação de redundância dos erros, dentre outras.

O envelhecimento de software foi observado não somente no software usado em grande escala, mas também em sistemas de software especializados, os quais exigem uma alta disponibilidade, por se tratarem muitas vezes, de aplicações de missão crítica (ex. Sistema de controle de sinais vitais em UTI de hospitais) (CASTELLI, 2001).

Castelli (CASTELLI, 2001) e GRAY apud GARG, SULLIVAN apud GARG (GARG, 1998) descrevem que as faltas em software são mais prováveis que as faltas em hardware. Entretanto, as faltas do software ocorrem normalmente depois de um certo período de tempo, causadas pelo crescente uso dos recursos do sistema, gerando muitas vezes, a corrupção de dados e o acúmulo de erros. Este fenômeno é denominado de **envelhecimento de software** e pode ser causado tanto por erros em uma aplicação, quanto no *middleware* [3] ou no próprio sistema operacional.

As faltas em softwares são estudadas por diversos autores, contudo existem praticamente três conceitos atualmente sendo explorados nesta área. HUANG apud TRIVEDI (TRIVEDI, 2002a), enfoca que as faltas no software não são uma entidade física e, portanto, não estão sujeitas aos fenômenos físicos transientes e, ao contrário do hardware, são permanentes por natureza. Outros classificam as faltas do software como

[2] Faltas *transientes* são de duração limitada, causadas por imperfeições do sistema ou em virtude de alguma interferência externa. Faltas transientes podem causar um erro, ou uma FALHA, somente naquele período de duração que ela existe. Conseqüentemente, um erro causado por faltas transientes podem existir somente naquele período. A detecção deste tipo de falta é bastante complexa e difícil. Uma Falta transiente pode ser do tipo intermitente. Neste caso, a falta aparece e desaparece várias vezes, repetidamente.

[3] Camada intermediária muito usada em arquiteturas de software em três camadas ou em cliente-servidor.

permanentes e transientes, em GRAY apud TRIVEDI (TRIVEDI, 2002a) e VAIDYANATHAN apud HONG (HONG, 2002) as faltas de softwares são classificadas como *Bohrbugs e Heisenbugs.* Segundo os autores, *Bohrbugs* são faltas de software de fácil reprodução e, portanto, podem ser facilmente removidas. Basicamente estas faltas deveriam ser removidas durante o processo de depuração do software, no final do seu processo de desenvolvimento.

Em (GRAY 1986), *Heisenbugs* são definidos como aquelas faltas de software que se manifestam apenas quando um conjunto de eventos, com o sistema em determinadas condições, ocorrem em uma determinada ordem. Por exemplo, uma determinada seqüência de operações pode levar o software para um estado que resultará em um possível erro durante sua próxima operação. Neste tipo de falta, uma nova tentativa da operação ou a reinicialização do sistema poderia resolver o problema.

Recentemente, uma classe de falha de software tem sido alvo de diversas pesquisas. Nesta, observa-se que a falha ocorre através do fenômeno da exaustão progressiva de recursos. Tipicamente, recursos do sistema operacional como, por exemplo, a memória virtual, são progressivamente afetados por problemas (faltas) originados pelos processos aplicativos. Por exemplo, vazamento de memória (*memory leaks*) e ausência de *cleanups* após o uso de recursos (ex. arquivos abertos), são situações presentes e que, muitas vezes, levam à progressiva exaustão de recursos do sistema operacional. Este tipo de problema ocorre não só no nível do sistema operacional, mas em qualquer categoria de sistema de software, sendo atualmente caracterizado como envelhecimento de software (TRIVEDI, 2002a).

A alta disponibilidade e confiabilidade, que os serviços WEB exigem, cresce a cada dia, devido a grande dependência que os novos sistemas vem tendo da WEB. Esta disponibilidade representa a porcentagem de tempo que um sistema está disponível durante um período de observação. Quanto maior o intervalo em que o sistema está ativo, melhor a disponibilidade. Já a confiabilidade do sistema é verificada pelo número de ocorrências de falhas durante o processamento do sistema (MARQUES, 2002).

Disponibilidade e confiabilidade são definidas pelos indicadores de falhas dos componentes dos sistemas observados, como por exemplo MTTF, MTTR e MTBF.

O *(mean-time-to-failure* - MTTF) representa o tempo médio em que ocorre uma falha. O (*mean-time-to-recover* - MTTR) representa o tempo médio para se reparar uma falha identificada. E o (*mean-time-between-failures* - MTBF) é o tempo médio entre falhas, que é igual a MTTF + MTTR. (MARQUES, 2002).

A disponibilidade de um sistema é representada por: MTTF / MTBF * 100%. Em contra partida a confiabilidade é expressa por MTBF = MTTF + MTTR, ou seja, é a probabilidade que os componentes de um sistema funcionarão conforme esperado por um longo período de tempo.

A partir destes indicativos, pode-se mensurar o quão disponível estão as aplicações, bem como identificar quais pontos devem ser melhorados para o incremento da sua disponibilidade.

3.1.1 Causas do envelhecimento de software

Segundo Parnas (PARNAS, 1994), programas, assim como os seres vivos, envelhecem. Não se pode prevenir o envelhecimento mas é necessário entender as suas causas, tomar medidas para limitar seus efeitos temporariamente e buscar soluções para os danos causados.

Segundo Castelli o envelhecimento ocorre porque software é algo extremamente complexo e nunca está totalmente livre de defeitos, visto que é praticamente impossível testar e verificar, totalmente, as partes integrantes de um software de médio ou grande porte. Esta situação é muitas vezes agravada, pelo fato de que o software tende a ter pouco tempo para o desenvolvimento, devido às exigências do mercado (CASTELLI, 2001).

As causas do envelhecimento de software são abordadas por diversos autores, sendo que para Trivedi (TRIVEDI, 2002b) a primeira causa do envelhecimento é a exaustão da operação do sistema e de seus recursos.

David Lorge Parnas tem investigado as causas e implicações do envelhecimento de software, afirmando que o envelhecimento do software ocorre em todos os produtos de software, inclusive naqueles bem sucedidos (PARNAS, 1994). Para ele, existem duas causas centrais que originam o problema do envelhecimento de software: a falta de atualizações e erros de projetos.

O primeiro resulta da falha dos usuários em atualizar ou mudar seu software para modificações necessárias, conseqüentemente resolvendo muitos dos seus problemas. O segundo é o resultado da mudança do software sem compreender os conceitos do projeto do sistema, amplificando problemas existentes e possivelmente introduzindo futuros problemas.

3.1.1.1 Ausência e/ou Inconsistência das atualizações

Segundo Parnas (PARNAS, 1994), o software evoluiu muito nas últimas décadas. Antes criavam-se programas e estes eram armazenadas sobre uma fita adesiva de papel, a qual era submetida a uma compilação e esperavam-se horas ou dias para compilar e funcionar o programa. A evolução do software trouxe a programação interativa e este problema foi resolvido.

Programas criados em 1960 poderiam ser utilizados até hoje, mas os usuários não os utilizariam, a menos que fosse feita uma atualização constante, pois o software daquela época estaria obsoleto. Normalmente, a atualização freqüente do software, prolongada por longos períodos de tempo, não compensa o custo/beneficio do produto e, portanto, os usuários preferem programas novos (PARNAS, 1994).

3.1.1.2 Erros de projeto

Embora a atualização do software seja essencial para impedir o seu envelhecimento, mudanças no software podem causar uma forma diferente de

envelhecimento. O projetista do software geralmente tem um conceito simples em mente ao escrever o programa. Dependendo da complexidade do programa, os conceitos pré-estabelecidos permitem que determinadas seções sejam alteradas ou corrigidas, conforme sua necessidade. Compreender o conceito implica também na compreensão do relacionamento existente entre as interfaces usadas com o sistema e entre o sistema e seu ambiente.

As mudanças são feitas, muitas vezes, por pessoas que não compreendem o conceito original do projeto e quase sempre causam a degradação da estrutura do programa. Algumas vezes os danos causados são pequenos mas, freqüentemente, são irreversíveis e às vezes quem fez as mudanças não havia participado do desenvolvimento anterior, o que amplia a probabilidade de erros.

As mudanças freqüentes do software induzem a outro problema: a atualização da documentação. Normalmente a documentação é relegada para modificação futura e nunca ocorre, deixando o software envelhecido (PARNAS, 1994).

3.1.2 Ocorrência e conseqüências do envelhecimento de software

O envelhecimento do software não ocorre somente em software com carga muito grande de uso, mas também em software de missão crítica como, por exemplo, o *Patriot Missile Defense System* (TRIVEDI, 2000 e GARG, 1998a). Em (HUANG, 1995), foi observado o envelhecimento no software de telecomunicação da AT&T, o qual apresentou perda de pacotes em suas comunicações. Além destes, existem outros exemplos onde pode-se observar o envelhecimento de software nos sistemas com exigências elevadas de disponibilidade.

Os sintomas do envelhecimento de software espelham-se no envelhecimento humano, sendo estes inevitáveis. Algumas conseqüências são:

- Os usuários de softwares envelhecidos acham cada vez mais difícil prosseguir com os seus produtos, caso estejam correndo o risco de perder clientes frente a novos produtos;
- O problema se manifesta freqüentemente e impacta nos sistemas correlacionados;
- O envelhecimento de software apresenta "*bugs*" devido aos constantes erros que são introduzidos toda vez que ocorre uma mudança, tornando o sistema instável.

Segundo Parnas (PARNAS, 1994) todo código novo, introduzido no software, nem sempre é bem compreendido e bem documentado. Visto que o programa aumenta gradativamente a cada atualização, as mudanças também tornam-se mais difíceis, além do custo que é agregado ao produto. Além disso, se um software aumenta de tamanho, este normalmente requer mais memória do computador. Caso não haja memória suficiente o programa responderá mais lentamente.

Parnas descreve ainda que, enquanto ocorre uma mudança para corrigir um erro no software, a literatura consta que, em média, mais um erro é introduzido neste software.

No próximo item serão apresentados detalhes sobre o mecanismo de rejuvenescimento de software.

3.2 Rejuvenescimento de software

O rejuvenescimento de software é um gerenciamento pró-ativo de sistemas de software, onde aplica-se uma técnica para renovar o estado interno do sistema monitorado, visando impedir a ocorrência de uma falha em detrimento do acúmulo de faltas neste sistema. Este processo envolve ocasionalmente terminar uma aplicação, limpando seu estado interno, a fim de reinicia-la posteriormente. As técnicas de gerenciamento pró-ativas de faltas, como o rejuvenescimento de software, podem ser

usadas para neutralizar o envelhecimento de software, caso este se manifeste. (GARG, 1998 e HUANG, 1995).

Também, de acordo com Dohi apud Gross (GROSS, 2002a), o rejuvenescimento de software é uma técnica de gerenciamento pró-ativo de faltas, o qual limpa o estado interno do sistema para prevenir a ocorrência futura de falhas mais críticas ou degradação de desempenho de sistema.

Huang (HUANG, 1995) descreve que se faltas de software (*bugs*) são encontradas durante as execuções de uma aplicação, elas poderiam evoluir e produzir falhas, muitas vezes graves ao ponto de paralisar ou inviabilizar a utilização da aplicação. Por exemplo, as faltas poderiam corromper um banco de dados, causar estouro de memória, bloqueio indevido de arquivos, ou poderiam induzir a degradação de outros recursos do sistema operacional, o que, em todas as situações citadas, pode levar a uma eventual falha e até à paralisação da aplicação.

As técnicas empregadas para recuperação de falhas são muitas vezes reativas, ou seja, elas consistem em ações que só são executadas depois que ocorreu a falha.

Um exemplo bem conhecido de rejuvenescimento é a reinicialização do hardware (DOHI, 2000a) através de *"reboots"*. Até pouco tempo, as soluções adotadas eram sempre executadas manualmente ou implementadas através de uma programação inteligente. Recentemente, plataformas e módulos de aplicações, tais como o *watchd*, *libft* e *nDFS/REPL* [DOHI, 2000a], são usados para recuperar aplicações de falhas depois que estas são detectadas. Alguns mecanismos reativos são: reinicialização, *rollback*, reordenação e repetição de comandos, dentre outros [DOHI, 2000a]. Esses mecanismos são recomendados para garantir a disponibilidade e integridade dos dados em uma determinada aplicação que executa continuamente e exige alta disponibilidade. Porém, as estratégias de recuperação reativas nem sempre garantem a longevidade da aplicação de forma imune a problemas.

Huang (HUANG, 1995) propõe um enfoque complementar para faltas transientes dos softwares, com enfoque para o envelhecimento de software. Esta técnica, pró-ativa, de rejuvenescimento de software, consiste em parar o software que está em execução,

remover o erro proveniente de uma falta, e reinicializar a execução da aplicação. No entanto, a utilização desta técnica não descarta a possibilidade de utilização de outros mecanismos, citados anteriormente, a fim de se ampliar a capacidade de resistência à falhas da aplicação. Nesse processo preventivo, Huang descreve que, se uma aplicação sempre apresenta uma falta no oitavo dia de execução, significa que a aplicação deve ser rejuvenescida a cada sete dias de execução, evitando assim que uma falha ocorra.

O desempenho da aplicação e sua disponibilidade são prejudicados durante o processo de rejuvenescimento, haja vista que a aplicação ficará indisponível por um curto espaço de tempo. No entanto, mesmo estando este tempo indisponível, ao se aplicar a técnica de rejuvenescimento, o custo da paralisação tende a ser menor do que se uma falha ocorrer e causar um maior período de inatividade (*downtime*), dentre outros efeitos indiretos que esta falha poderia causar como, por exemplo, a perda de dados.

Para (OKAMURA, 2001), o rejuvenescimento de software é uma das técnicas mais eficazes de manutenção preventiva de sistemas de software com exigências elevadas de confiabilidade, que vem sendo estudada extensivamente na atualidade.

O processo de rejuvenescimento software remove os erros acumulados e liberam recursos do sistema operacional. A ação preventiva pode ser feita em horários opcionais (por exemplo, quando a carga no sistema é baixa) de modo que a sobrecarga aplicada sobre o sistema no horário de reinicio seja mínima (TRIVEDI, 2000).

Segundo a IBM (IBM, 2000), as iniciativas da indústria que se detém aos problemas do envelhecimento do software focalizaram, até o momento, soluções reativas, tais como nos sistemas tolerantes à falhas. Já o rejuvenescimento do software busca ser uma tecnologia pró-ativa, onde a mesma é projetada para prevenir e controlar o envelhecimento do software antes que este acarrete em uma real falha do sistema. O rejuvenescimento do software não substitui outras técnicas de recuperação de falhas, mas as complementa, ajudando a reduzir a possibilidade de ocorrência da falha.

O rejuvenescimento de software não remove *bugs* que resultam no envelhecimento de software; no entanto, a sua aplicabilidade previne que eles se

manifestem ao ponto de causarem uma falha ou indisponibilidade do sistema (DOSS, 2001).

Segundo Dohi (DOHI, 2000a) confiabilidade e disponibilidade nos software são quesitos primordiais para as aplicações nos dias de hoje. As exigências dos requisitos em termos de acúmulo de tempo de reinício e falhas na operação do software são grandes e, em muitos casos, as conseqüências de falhas em software podem conduzir a enormes perdas econômicas ou podem arriscar vidas humanas. Contudo, Dohi (DOHI, 2000) enfatiza que é muito difícil de projetar e garantir tais exigências, principalmente em aplicações com um determinado grau de complexidade.

Candea (CANDEA, 2001) descreve que existem três motivos para se aplicar rejuvenescimento de software baseado na técnica de reinicialização da aplicação, como seguem abaixo:

- Primeiro: uma reinicialização de software renovará o estado interno da aplicação e esta voltará no seu estado inicial. Dohi e Garg (DOHI, 2000 e GARG, 1998a) descrevem que um exemplo de sistema crítico de segurança é o software do sistema militar *Patriot*[4], onde os erros acumulados com o sistema em operação conduziram a uma falha que resultou em perda de vidas humanas. Toich (TOICH, 1998) enfatiza que o problema apresentado pelo software *Patriot* relaciona-se com uma falha no software devido ao longo período de utilização do sistema, visto que o desenvolvimento do projeto original o incumbia como um sistema de defesa anti-aérea e não anti-míssil, o qual exige uma operação (execução) por um maior número de horas consecutivas. Toich descreve que as baterias *Patriot* devem executar continuamente por longos períodos de tempo ininterruptos;

- Segundo: as reinicializações (*reboots*) fornecem maior confiança na recuperação dos recursos que estão envelhecidos;

[4] O sistema de míssil Patriot, foi usado na primeira guerra do Golf entre USA e Iraq. Este se baseia em um sistema de bateria anti-aérea que automaticamente detecta um alvo inimigo e dispara contra ele. Toda infra-estrutura do sistema é controlada por sistemas de software.

- Finalmente, a terceira vantagem é que reinicializar é um método fácil de se compreender e empregar. Mecanismos simples beneficiam-se como por serem de fácil implementação.

Segundo pesquisas da IBM (IBM, 2003), ocorrem três vezes mais falhas de sistemas relacionadas ao software do que ao hardware. Tanto o sistema operacional, quanto o *middleware* e as aplicações envelhecem com o tempo.

A IBM afirma que atualmente está trabalhando num projeto tecnológico denominado Eliza (IBM, 2002) o qual tem como enfoque a implementação de uma ferramenta de rejuvenescimento de software, podendo-se predizer quando um sistema deixaria de funcionar, em função de problemas acarretados pelo envelhecimento de software, a fim de se aplicar eficientemente uma técnica de rejuvenescimento de software neste sistema.

Este trabalho da IBM visa, dentre outras áreas, as aplicações cliente-servidor. Neste ambiente, o servidor executa continuamente um ou diversos serviços para seus clientes. Rejuvenescer o(s) processo(s) servidor(es), durante um período em que este estiver ocioso, aumenta a disponibilidade dos seus serviços, prevenindo-se de futuros problemas de envelhecimento. Em aplicações de execuções intensas, rejuvenescer a aplicação periodicamente aumenta a sua probabilidade de sucesso de execução. Huang (HUANG, 1995) apresenta um modelo para analisar o rejuvenescimento de software em aplicações de funcionamento contínuo, obtendo como resultado o tempo de reinício e os custos oriundos da reinicialização através de parâmetros determinados neste modelo.

Castelli (CASTELLI, 2001) apresenta diversos exemplos de sistemas, onde está sendo aplicado o rejuvenescimento de software. Em um dos casos apresentados por Castelli, o rejuvenescimento de software foi implementado em sistemas de tempo real para correlacionar dados de faturamento da maioria das estações telefônicas dos Estados Unidos. A capacidade de restauração deste software é uma técnica similar ao rejuvenescimento e foi usada por Avritzer e Weyuker dentro de uma grande aplicação de software de telecomunicação.

Conforme Tai apud Gross (GROSS, 2002a), o rejuvenescimento de software obteve êxito pela NASA no software de sistemas de Vôo avançado X2000. Este foi decorrente da troca entre os componentes do sistema, reduzindo o processo de envelhecimento do sistema e aumentando confiança da missão.

O rejuvenescimento do software ameniza o envelhecimento do software, minimizando falhas com o seu *rollback* periódico e/ou reinicializações do sistema. A maneira mais direta, e atualmente mais utilizada, é recarregar periodicamente o sistema afetado. Apesar da reinicialização manual minimizar os problemas acima mencionados, o problema com este enfoque é que a aplicação pára de responder durante o processo de reinicialização. Estas técnicas são reativas e aplicam-se após apresentarem o problema de envelhecimento do software, como já descrito nesta seção. Uma forma de contornar este problema é aplicar conjuntamente o rejuvenescimento de software com uma configuração de *clustering*, evitando-se assim a indisponibilidade da aplicação. As Fig. 11 e 12 apresentam os benefícios do rejuvenescimento na redução do *downtime* do sistema.

Conforme relato de Donohue, (DONOHUE, 2002) o custo com *downtime* em determinados sistemas ultrapassam a 100 dólares por segundo. Empresas como Amazon.com tem um custo de $ 4,00/segundo (quatro dólares por segundo); Donohue relata em seu estudo que a empresa Wall Street tem um custo de $ 108,000/segundo (cento e oito mil dólares por segundo).

Scott apud Candea (CANDEA, 2003) estima que 60% (sessenta por cento) do tempo de manutenção (*downtime*) não planejado em ambientes computacionais estão relacionados à falhas na aplicação e a falhas de hardware, das quais 80% (oitenta por cento) são transientes, que são solucionadas com uma reinicialização.

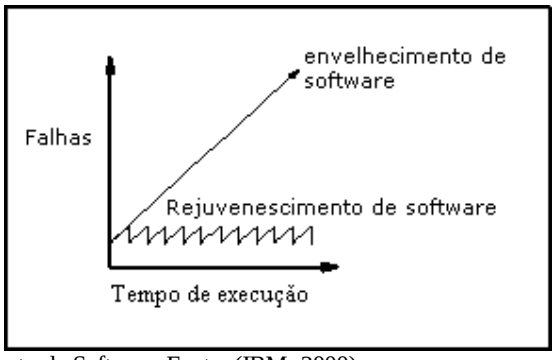

Figura 11. Rejuvenescimento de Software Fonte: (IBM, 2000)

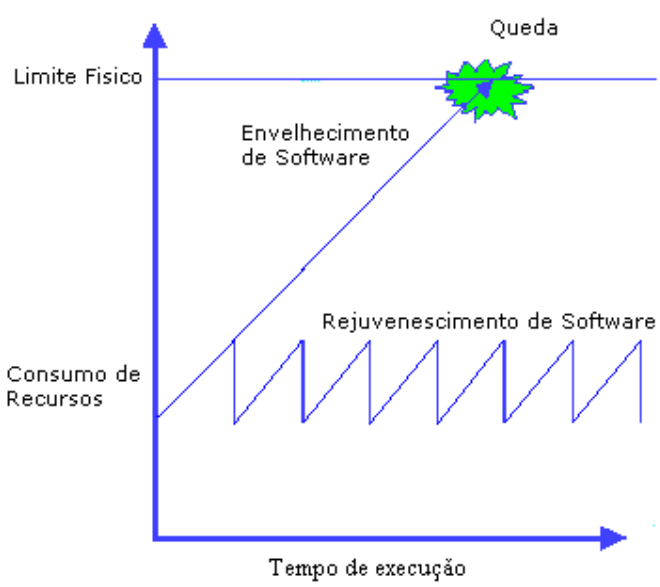

Figura 12. Rejuvenescimento de Software reduz *Downtime.* Fonte: (GROSS, 2002)

O rejuvenescimento pode ser aplicado em diferentes níveis, ou seja, tanto no nível de sistema como no nível de aplicação. Caso o hardware seja inicializado, aplica-se um rejuvenescimento em nível de sistema. Para Vaidyanathan (VAIDYANATHAN, 2001a) rejuvenescimento em nível de aplicação é quando uma aplicação, que se encontra em execução, é parada e inicializada, ou seja, os processos da mesma são reinicializados.

Um modelo proposto por Huang (HUANG, 1995), enfoca uma solução pró-ativa, ao invés de uma solução reativa. Contudo, Huang enfatiza que esta solução preventiva não garante completamente que os problemas sejam eliminados e que a possibilidade de ocorrer uma falha seja nula. Caso ocorra uma falha apesar de ter sido aplicado o

rejuvenescimento de software, os mecanismos reativos devem ser aplicados para recuperar a aplicação.

Para a IBM (IBM, 2000) as abordagens do rejuvenescimento de software são: rejuvenescimento baseado em tempo (TSR) e o rejuvenescimento baseado no sintoma seletivo (SSR). Ambos os enfoques fornecem um meio que pode ser usado para iniciar uma realocação programada e uma renovação do software.

A TSR confia na habilidade de um controlador inteligente para avaliar a freqüência aproximada dos processos antigos devido ao uso exaustivo de seus recursos.

Já o método SSR é mais sofisticado, pois estima quando um sistema ou uma aplicação necessita de rejuvenescimento, através de um agente de rejuvenescimento, o qual monitora o comportamento de um sistema de software para evidenciar a exaustão dos recursos. Como exemplo de utilização deste método pode-se citar o sistema operacional Microsoft Windows NT®, onde as informações são exportadas para o registro e o agente monitora os recursos do sistema tais como os eventos e a memória virtual.

Um ou mais parâmetros críticos podem ser monitorados por sinais de exaustão, onde podem preceder uma determinada falha do software. Caso um ponto ou uma área de alto risco seja alcançado, um alerta pode ser gerado. Neste caso agirá como um dispositivo para a recuperação do sistema antes que a aplicação ou o sistema sofra uma falha inesperada. Existem muitos algoritmos envolvidos neste processo, pois há uma ligação forte entre o sistema operacional, aplicação e carga do trabalho.

3.2.1 Soluções atuais e em desenvolvimento

Para neutralizar o fenômeno do envelhecimento de software, Trivedi sugere como solução o rejuvenescimento de software. Este consiste em remover os erros acumulados pelo sistema e liberar os recursos utilizados. A proposta visa conseguir um alto grau de disponibilidade do software, através de uma gerência pró-ativa de faltas. Recentemente,

Trivedi desenvolveu um modelo para detectar o envelhecimento de software de um servidor WEB (TRIVEDI, 2002).

Vaidyanathan (VAIDYANATHAN, 2001a) apresenta os enfoques que o rejuvenescimento de software pode ter, caracterizados por *open-loop* e *closed-loop*. Hong (HONG, 2002) afirma que a maioria das técnicas de rejuvenescimento de software se enquadram na categoria *open-loop*, isto é, a ação de rejuvenescimento é baseada em critérios predeterminados. Muitos métodos incluem cadeias de *Markov*, técnicas semi-Markov e redes de Petri, dentre outros apresentados em (HUANG, 1995, GARG, 1998, DOHI, 2000 e BOBBIO, 1999).

No entanto, uma outra linha de pesquisa está sendo estuda por CASTELLI apud HONG (HONG, 2002) e que vem de encontro com um dos enfoques apresentados por Vaidyanathan, onde o rejuvenescimento é tratado como *closed-loop*, ou seja, a ação de rejuvenescimento é baseada na medida do nível de degradação do software. Para reforçar o processo desta teoria, BOBBIO apud HONG (HONG, 2002) propõe uma política de rejuvenescimento baseada num modelo onde a ação de rejuvenescimento é ativada baseada num determinado sinal de alerta, o qual é definido de acordo com cada aplicação sendo analizada.

Gross (GROSS, 2002a) descreve a viabilidade de aplicar o rejuvenescimento software baseado em previsão. Em seu estudo utilizou-se de um modelo estatístico avançado de reconhecimento de sistema que foi usado pelo NASA's *Space Shuttle Main Engine* com uma validação notável, pois este mecanismo detectou o início de envelhecimento de software dentro dos servidores de missão-crítica deste sistema, conclusão obtida no estudo de Gross.

Marques (MARQUES, 2002) também apresenta uma pesquisa que tem como objetivo uma análise da interferência das variações de sobrecarga de trabalho, típicas em software do ambiente WEB, no processo de envelhecimento.

3.3 Envelhecimento de Software em Servidores WEB

Com o crescimento explosivo da tecnologia de Internet e o aparecimento de várias aplicações avançadas usadas na WEB, os usuários exigem, além de segurança nos sistemas, disponibilidade nos dados fornecidos. Portanto, um dos desafios é oferecer a disponibilidade desejada em um melhor desempenho possível.

O termo envelhecimento de software é principalmente aplicável a sistemas de software que são projetados para executar por um longo período de tempo como, por exemplo, uma aplicação servidora em um sistema cliente-servidor. Neste contexto pode-se citar o servidor WEB como um dos atuais sistemas que atualmente são exigidos em regime 24x7.

Usuários de computador, localizados no mundo inteiro, em parte estão familiarizados com os fenômenos apresentados pelo sistema operacional, tais como queda do software ou aplicação e paralização do sistema. O problema maior está nesta parada do sistema ou nas reinicializações das aplicações ou do sistema, de forma não programada e inesperada, o que normalmente resulta em perda de dados e indisponibilidade dos serviços oferecidos. (CROWELL, 2001).

Um servidor WEB é uma aplicação tipicamente de uso contínuo e intenso, sendo as duas variáveis (tempo e carga de trabalho) muito dependentes do ambiente de execução da aplicação. Por apresentar estas características, este tipo de aplicação é uma forte candidata a apresentar sintomas do envelhecimento de software.

Em (LI, 2002), comprovou-se que o desempenho de um servidor WEB degrada depois de um período de tempo de execução, devido, principalmente, a efeitos causados por envelhecimento do software. As ações tomadas pelos administradores vão desde o reinicio total do sistema ou parcial, reiniciando apenas a aplicação (processo) do servidor WEB. Contudo, os administradores de sistema determinam os intervalos entre os reinícios aleatoriamente, ou simplesmente reiniciam após uma parada total (*crash*) do sistema. O objetivo da pesquisa realizada por (LI, 2002) é predizer quando ocorre a exaustão do sistema, a fim de determinar o tempo apropriado para efetuar a aplicação da

técnica de rejuvenescimento de software. No trabalho de (LI, 2002) não foi abordada nenhuma técnica de rejuvenescimento em específico.

Desenvolver e manter sistemas sem falhas, principalmente para WEB, é praticamente impossível, devido ao tamanho e a constantemente evolução que ocorre neste tipo de ambiente. Os sistemas baseados em serviços oferecidos na Internet, por serem de uma estrutura heterogênea, podem variar tanto em número de componentes quanto em carga de trabalho a eles submetidos. Tais sistemas estão sujeitos a uma evolução constante. Isto significa que é muito complexo construir um modelo que permaneça consistente com o passar do tempo, o que aumenta enormemente as possibilidades de falhas acontecerem e conseqüentemente levarem ao envelhecimento de software.

Um enfoque apresentado por Candea possui como contexto um ambiente de serviços de Internet e é voltado a um mecanismo recursivo no qual, primeiramente, um subconjunto mínimo de componentes será recuperado. Se isso não resolver o problema, será recuperado um subconjunto maior de componentes. Os serviços de Internet exigem alta disponibilidade, no entanto a carga submetida ao servidor WEB é relativamente grande, mas as tarefas são normalmente pequenas e o impacto de uma micro-reinicialização é menor do que a recuperação de uma falha transiente. (CANDEA, 2003).

Um outro enfoque, utilizado por (CANDEA, 2003), é a aplicação de pequenos reinícios recursivos em um sistema de satélite. A estação ficou em operação por mais de 3 (três) anos. Depois desta experiência, foi proposto que as aplicações utilizassem um sistema recursivo, com pequenas reinicializações em outros sistemas de software similares.

3.4 Rejuvenescimento de software em servidores WEB

Como descrito no item 3.2 o conceito de rejuvenescimento de software está relacionado com a parada de uma aplicação e, na seqüência, a reinicialização da mesma,

sendo que este procedimento renovará o estado interno da aplicação. Por exemplo, em uma aplicação do tipo Cliente/Servidor, onde o servidor é destinado para rodar continuamente, com o objetivo de fornecer um determinado serviço para um de seus clientes, o rejuvenescimento no processo servidor se faz necessário. No entanto, durante este período, o processo servidor ficará inativo, ou seja, o serviço não estará disponível. (FORREST, 2002).

Segundo Kc (KC, 2002) as técnicas de rejuvenescimento de software envolvem diversas. Renovar o estado interno da aplicação para remover o acúmulo de *bugs* transientes, de erros numéricos, recursos bloqueados, estouro de memória e dados corrompidos, é fundamental para o conceito de rejuvenescimento.

Além disso, é fundamentel considerar o nível de rejuvenescimento que uma aplicação pode ser submetida. Neste caso, existem basicamente dois níveis:

Rejuvenescimento Total: É o rejuvenescimento de software aplicado através de reinicializações agendadas, buscando o menor impacto possível na aplicação, ou seja, o tempo de manutenção da aplicação deve ser menor do que aquele no caso da ocorrência de uma queda do sistema (*crash*). Além de causar o menor impacto este rejuvenescimento de software deve ser executado durante um horário de pouco uso, onde a interrupção seja a menor possível.

Rejuvenescimento Parcial: Este consiste em rejuvenescer de forma transparente os sub-componentes envolvidos no processo. Este rejuvenescimento dos sub-componentes é aplicado num componente de forma isolada, sem afetar os componentes que estejam em execução no momento do rejuvenescimento.

Ambos podem ser aplicados de forma recursiva (CANDEA, 2002), realizados progressivamente.

Forrest (FORREST, 2002) enfoca o conceito da maioria dos softwares, os quais são projetados para executar e depois de encontrar uma resposta e parar. No entanto, juntamente com os programas, existem sistemas operacionais e demais processos de software que rodam continuamente neste mesmo ambiente.

Conforme abordado por (CANDEA, 2003), a disponibilidade de um sistema global é representada pela combinação da função de reinicialização de componentes que apresentam falhas (renascimento) e a função de reinicializações de componentes (rejuvenescimento) o qual previne o estado de degradação que pode conduzir a um *downtime* da aplicação. Uma técnica é usar recursividade em pequenos reinícios para recuperar um subconjunto mínimo dos componentes de um sistema e, se isso não funcionar, aplica-se a recursividade de recuperação de um subconjunto progressivamente maior do que aplicado anteriormente. O micro-reinício é uma forma barata de reinicialização que é aplicado ao nível individual, ou seja, apenas aos componentes envelhecidos do software.

Brewer apud Candea (CANDEA, 2003), afirma que especialistas em portais da Internet, após verificarem a degradação de memória, habitualmente reinicializam o processo do servidor WEB.

Segundo Castelli (CASTELLI, 2001) o Apache funciona com um processo controlador (*controlling process*) e um conjunto de processos manipuladores (*handler process*). O processo controlador monitora os processos manipuladores para assegurar que estes estão executando corretamente. Cada processo manipulador, por sua vez, manipula as requisições dos clientes. Este modelo permite que caso um processo manipulador esteja com problemas, o mesmo possa ser reinicializado, não afetando o atendimento de requisições, haja vista a existência de outros processos manipuladores realizando este atendimento.

3.5 Faltas encontradas no APACHE

Chandra (CHANDRA, 2000) classifica as faltas de software baseadas na dependência que elas têm do ambiente operacional. A operação do ambiente está relacionada com os estados ou eventos que acontecem além dos que já foram estudados naquela aplicação. Portanto, este ambiente operacional inclui software e hardware de

infra-estrutura. Exemplos de software num ambiente operacional incluem dentre outros programas, o kernel do sistema operacional, servidor de nomes DNS, dentre outros.

Chandra (CHANDRA, 2000a) caracterizou as faltas para cada uma das aplicações dentro de três classes que compreendem: **independente do ambiente**, **não transiente dependente do ambiente** e **transiente dependente do ambiente**.

As faltas do tipo **independente do ambiente** acontecem independentemente do ambiente operacional. Em uma determinada carga de trabalho específica (por exemplo uma solicitação de uma operação de um usuário), as faltas podem acontecer.

As faltas do tipo **dependentes do ambiente** dependem do ambiente operacional, isto é, o ambiente operacional executa alguma ação que ativa um *bug* existente no programa. Por exemplo, o programa pode não tratar de uma rede com tempo de resposta alto, de um disco cheio, etc. Estes erros (faltas) podem ocorrer porque o software pode se comportar diferentemente a cada solicitação de uma operação (CHANDRA, 2000a).

Já para as faltas do tipo **não transiente dependente do ambiente** é improvável que o ambiente seja mudado a tempo para evitar o *bug* durante uma próxima execução. Por exemplo, o programa pode falhar se o disco estiver cheio. (CHANDRA, 2000a)

A Tabela 4 apresenta os resultados da classificação das 50 faltas contidas no Apache. A maioria das faltas não depende do ambiente operacional. Estes erros, determinísticos, são relativamente fáceis de serem reproduzidos.

Tabela 4. Classificação de faltas do Apache. Fonte: (CHANDRA, 2000a)

CLASSE	FALTA
Independente-do-Ambiente	36
Não-transiente-dependente-do-Ambiente	7
Transiente-dependente-do-Ambiente	7

No **independente-do-ambiente** o Apache apresenta alguns dos erros (bugs), como por exemplo quando uma URL muito longa é submetida ao servidor WEB Apache. Este problema ocorre como resultado de uma sobrecarga no cálculo *hash* Chandra (CHANDRA, 2000a) considera como bug os *SIGHUP* nas versões do Apache

em sistemas operacionais como Solaris e Unix. Normalmente, isto ocorre quando é aplicado um reinício/rejuvenescimento no apache.

Das 14 falhas que dependem do ambiente operacional, a metade delas são devido às condições que persistem durante a recuperação do sistema. As condições do ambiente operacional que provocam as 7 (sete) faltas do tipo **não-transientes-dependente-do-ambiente** são:

- Alta carga de trabalho conduz a um vazamento de recurso de memória. O vazamento do recurso (*leaks*) persistirá durante a recuperação da aplicação, no entanto retornando em níveis aceitáveis;
- Perda de referência para o descritor de arquivos;
- Falta de recursos para armazenamento temporário (swap);
- O tamanho do arquivo de log do Apache excede o tamanho máximo de arquivo permitido pelo SO;
- Falta de espaço em disco;
- Rede desconhecida e/ou exaustão de recurso de conexão.

As faltas restantes (**transiente-dependente-do-ambiente**) são ativadas devido às condições do ambiente operacional. As condições do ambiente operacional que ativam estas faltas são:

- Uma requisição de DNS retorna erro. Isto acontece quando ocorre mudança na configuração do DNS e este precisa ser reiniciado.
- O processo filho aborta sua execução durante um pico de carga e não libera os seus recursos alocados.
- Quando o usuário clica no botão parar do navegador no meio de uma página que está sendo carregada no seu browser. Esta falta depende do momento exato da solicitação e da carga da solicitação, sendo que não será possível novamente um reenvio depois do processo de recuperação;
- Resposta muito lenta do DNS. A causa da resposta do DNS ser lenta está relacionada com a recuperação de uma aplicação específica ou com o reinicio do servidor de nomes, ou ainda, devido a algum problema na rede.
- Conexão de rede lenta.

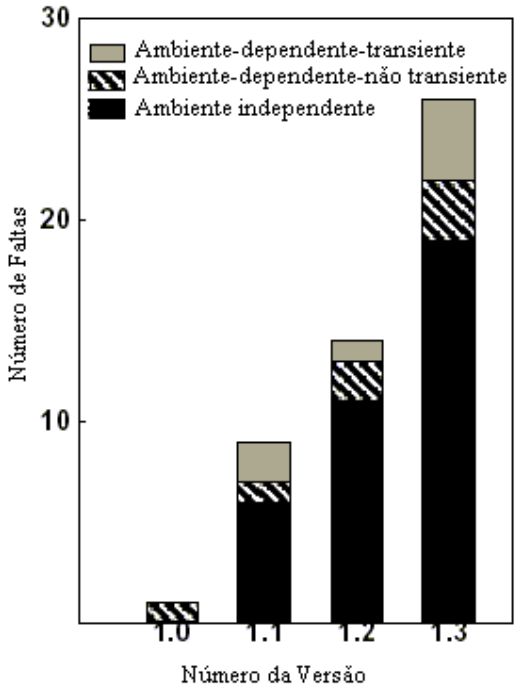

Figura 13. Distribuição de faltas para Apache sobre atualizações do software. Fonte: (CHANDRA, 2000a)

A Fig. 13 mostra a distribuição das faltas sobre liberações (atualizações) do software apache. A distribuição de faltas exibe duas propriedades distintas. Primeiro, a proporção relativa aos *bugs* do tipo **independente-do-ambiente** permanece praticamente inalterada, mesmo sob novas atualizações (lançamentos) do software. Segundo, o número total de *bugs* mostra o aumento de faltas relacionado com cada nova liberação do software. Isto provavelmente está relacionado a um aumento no número de usuários perante as novas atualizações (liberações) do software.

A partir da identificação das principais causas de envelhecimento de software em servidores WEB Apache, realizou-se um experimento em laboratório, com o objetivo de testar o desempenho deste tipo de servidor quando submetido à técnicas de rejuvenescimento. Os detalhes deste experimento são apresentados no próximo capítulo.

4 EXPERIMENTAÇÃO REALIZADA EM UM AMBIENTE DE ISP

Como ficou comprovado pelos estudos citados anteriormente, o servidor WEB Apache está sujeito a problemas de envelhecimento de software em diversas situações.

De forma a contribuir com os trabalhos citados, verificou-se a ausência de um estudo apontando resultados comparativos entre um ambiente WEB, baseado no Apache, com e sem a aplicação de um mecanismo de rejuvenescimento.

4.1 Mecanismo de Rejuvenescimento Utilizado no Servidor Apache

Como descrito no item 2.1.1, o servidor WEB Apache opera com o conceito de mestre e escravo. Um processo principal cria demais processos escravos, os quais atenderão todas as requisições dos clientes. O processo principal (mestre) monitora constantemente os processos escravos, aplicando as regras definidas em seu arquivo de configuração (ANEXO 01).

Estudando o seu comportamento de forma detalhada, foi identificado que ao receber um sinal `SIGUSR1`[5], o processo principal repassa o sinal para seus processos escravos. Cada processo escravo, apenas atenderá este sinal quando não estiver atendendo requisições dos processos clientes. Neste caso, o comportamento programado nestes processos é o de se reinicializarem. Desta forma, foi criado um processo *daemon*[6], chamado `sr` (*software rejuvenation*), o qual periodicamente (a cada 30 segundos) envia um sinal `SIGUSR1` ao processo principal do Apache, para fins de rejuvenescê-lo. A periodicidade de aplicar o software de rejuvenescimento definiu-se de forma *ah doc*, considerando que um único processo já havia atendido a 810 requisições, além de aumentar a precisão nos resultados.

[5] Sinal (*signal*) usado em sistemas UNIX como forma de comunicação entre processos (IPC).
[6] Termo utilizado para nomear processos residentes em memória e que executam em segundo plano (*background*) em sistemas baseados no UNIX.

Ao executar o comando `sr` é fundamental informar o `PID`[7] do processo Apache principal. A seguir um exemplo de sua execução.

```
% ./sr  /usr/local/var/httpd.pid
```

Em virtude da organização interna do Apache, este mecanismo implementa um rejuvenescimento parcial do servidor, haja vista que apenas alguns dos processos escravos, livres de conexão, serão reinicializados, estando os demais em pleno funcionamento, evitando assim a parada total do serviço. Para aqueles processos, que ao receberem o sinal, estejam atendendo conexões ativas no momento, assim que estas conexões sejam encerradas, eles também serão reinicializados.

4.2 Descrição dos Experimentos

Após selecionado o mecanismo de rejuvenescimento a ser utilizado, a fim de comparar o comportamento do servidor com e sem rejuvenescimento, buscou-se definir um ambiente de teste o mais próximo do ambiente real do ISP[8]. A infra-estrutura real do ISP em questão segue abaixo:

Servidores (dois):

- Sistema Operacional Linux, Conectiva 8;
- Servidor WEB Apache 1.3.27 (PHP 4.1.2 +MySQL+SSL+SSH);
- Servidor SMTP;
- Servidor DNS;
- Servidor FTP;
- Configurações do Hardware:
  - **Máquina Servidor 1** - Pentium IV 1.7 MHZ, 512 MB RAMBUS, HD 40 GB 7200 RPM, Placa de Rede 3Com 10/100;

[7] Identificador do Processo usado pelo sr para enviar o sinal ao processo desejado.
[8] ISP, Provedor de Internet localizado na cidade de Chapecó – SC, denominado como SuperIP.

- **Máquina Servidor 2** - Pentium III 900 MHZ, 512 MB RAM, HD 40 GB 7200 RPM, Placa de Rede 3Com 10/100;
- 1 HUB/Switch 10/100;
- 1 Roteador
- 1 RAS

Além das máquinas servidoras existem também 5 estações que são utilizadas no setor administrativo, conforme ilustra a Fig. 14.

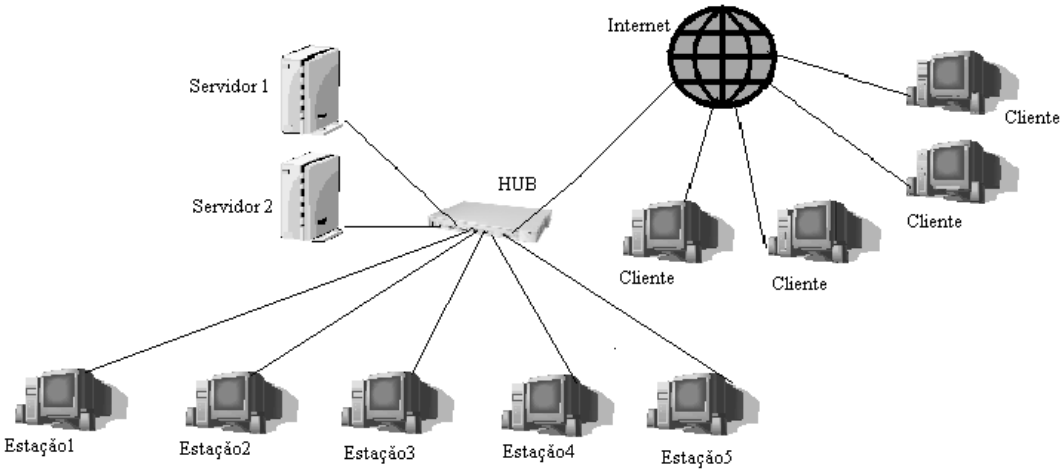

Figura 14. Ambiente Real do provedor SuperIP

Para o desenvolvimento dos experimentos foi utilizado um laboratório composto de 4 máquinas, sendo uma delas configurada como servidor e as demais configuradas como clientes para a geração parametrizada de tráfego.

Nas estações do tipo cliente foi instalado um aplicativo de geração de requisições HTTP, o qual gera estas requisições de forma parametrizada, permitindo se definir a carga desejada. Esta ferramenta gera o tráfego e contabiliza as suas respostas. A interconexão das máquinas foi feita por intermédio de um HUB. A Fig. 15 apresenta o ambiente de teste utilizado.

A configuração do ambiente onde foram realizados os experimentos está descrita a seguir:

- Sistema Operacional Linux, Conectiva 8.0;

- Servidor WEB Apache 1.3.27 (PHP+MySQL+SSL+SSH);
- Configurações do Hardware:
  - **Máquina Servidor** - Pentium III 500 MHZ, 256 MB RAM, HD 9 GB, Placa de Rede 3Com 10/100, Placa de rede Realtec 10/100;
  - **Máquinas Cliente**, ou seja, as geradoras de tráfego - K6 II 350 MHZ, 128 MB RAM, HD 4 GB, Placa de Rede Realtek 10/100;
  - 1 HUB/Switch 10/100;
  - Nobreak 1.5 KVA.

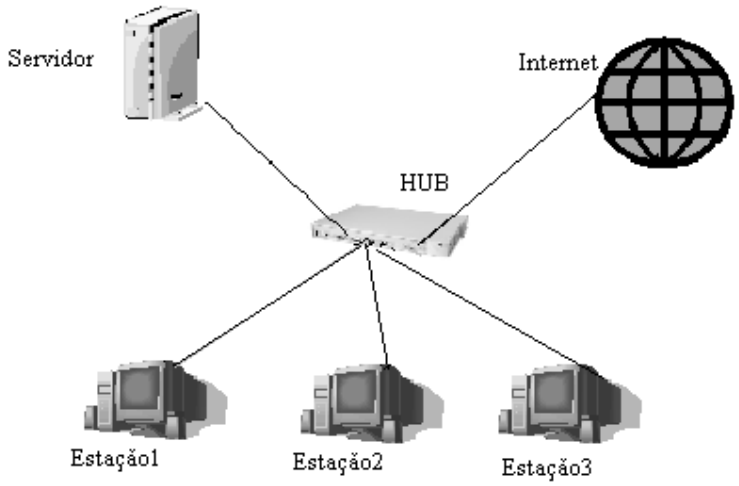

Figura 15. Ambiente de teste

A partir da montagem do ambiente de teste, iniciou-se o trabalho de caracterização do tráfego WEB do ISP, a fim de se gerar o mesmo padrão de tráfego no ambiente de teste. Esta caracterização foi baseada em análises dos dados reais coletados do servidor WEB Apache do provedor SuperIP.

4.2.1 Metodologia Utilizada

Inicialmente, realizou-se a coleta dos dados através do software *Webalizer* versão 2.01. Considerou-se a quantidade de solicitações geradas por mês, no período de 1º de agosto de 2002 até 30 de junho de 2003 (Tabela 5). Esta coleta foi realizada a partir dos

logs do servidor Apache do provedor. A Tabela 5 apresenta um resumo dos dados coletados no período.

Tabela 5. Resumo mensal do tráfego gerado no provedor SuperIP, no período de 01 de Agosto de 2002 à 30 de Junho de 2003.

Solicitações atendidas pelo servidor WEB Apache versão 1.3.27		
Mês	Média Diária de Solicitações	Total Mensal de Solicitações
Jun/2003	23.107	693.210
Mai/2003	24.340	754.569
Abr/2003	21.559	646.778
Mar/2003	14.255	441.933
Fev/2003	16.274	455.687
Jan/2003	17.996	557.878
Dez/2002	19.337	599.453
Nov/2002	18.055	541.656
Out/2002	18.140	562.346
Set/2002	26.509	795.292
Ago/2002	22.212	688.580
Total de solicitações do período	221.784	6.737.381

Além dos dados de requisições, foram identificadas as 10 (dez) páginas mais acessadas em ordem decrescente de solicitações, conforme relação abaixo:

1. http://www.superip.com.br/home
2. http://www.superip.com.br/horde/imp/mailbox.php
3. http://www.superip.com.br/horde/imp/message.php
4. http://www.superip.com.br/home/index1.php
5. http://www.superip.com.br/imp/compose.php
6. http://www.superip.com.br/~contabilidade/hps/hp.php
7. http://www.superip.com.br/horde/imp/login.php
8. http://www.superip.com.br/horde/imp/inicial.php
9. http://www.superip.com.br/home/noticias.php
10. http://www.superip.com.br/~contabilidade

Para o desenvolvimento do presente trabalho de pesquisa houve a necessidade de se identificar os valores (requisições) que mais se repetem no período analisado. Para filtragem dos dados, foi utilizado o banco de dados Interbase. Foi necessário o desenvolvimento de um aplicativo em linguagem Delphi para efetuar a importação dos dados armazenados no arquivo de *log* do Apache para o banco de dados. A partir de então, foi desenvolvida uma *query* SQL para determinar quantas solicitações por

segundo o servidor WEB Apache, do ambiente real, recebeu no período analisado. Essa rotina em SQL é apresentada a seguir:

```
select count(tmp_qtd), tmp_qtd from qtd
group by tmp_qtd
```

Esta *query* apresentou como resultado a quantidade de repetições de um determinado valor dentro do período compreendido entre 1º de abril de 2003 a 30 de junho de 2003, conforme mostra a Tabela 6.

Tabela 6. Quantidade de ocorrências de um determinado número de solicitações HTTP por segundo[9].

Qtd de repetições	Qtd Solicitações por segundo
1	47
1	42
1	41
1	40
1	39
8	38
5	37
9	36
12	35
24	34
34	33
38	32
36	31
56	30
73	29
85	28
104	27
138	26
133	25
201	24
266	23
402	22
524	21
745	20
884	19
1.155	18
1.371	17
1.840	16
2.133	15
2.597	14
3.169	13
3.847	12

[9] Baseado nos logs do servidor web Apache do provedor SuperIP.

Qtd de repetições	Qtd Solicitações por segundo
4.823	11
6.564	10
8.134	9
11.585	8
13.378	7
20.271	6
25.756	5
46.243	4
74.274	3
179.340	2
513.757	1

Juntamente com esses dados da Tabela 6 coletou-se a data e hora da ocorrência das requisições. Posteriormente, gerou-se uma lista com os 20 (vinte) maiores valores de requisições por segundo do período analisado. Estes valores são apresentados na Tabela 7.

Tabela 7. Quantidade de solicitações HTTP por segundo ao servidor WEB Apache do provedor SuperIP

Data	Horário	Solicitações
18/06/03	10:59:14	47
25/06/03	11:39:14	42
07/05/03	14:10:19	41
03/05/03	19:14:16	40
05/03/03	13:52:27	39
16/05/03	17:08:59	38
20/05/03	12:52:56	37
01/04/03	21:18:40	36
17/04/03	10:39:39	35
17/04/03	16:50:19	34
08/05/03	18:21:28	33
02/05/03	14:03:13	32
07/04/03	19:37:02	31
03/05/03	10:44:46	30
17/04/03	10:37:51	29
23/04/03	08:46:55	28
28/04/03	18:20:34	27
10/04/03	22:02:57	26
25/04/03	09:30:57	25
28/04/03	13:44:33	24

Com base na Tabela 7, extraiu-se os 10 (dez) valores que mais se repetiram conforme mostra a Tabela 8.

Tabela 8. Total de ocorrências do mesmo número de solicitações HTTP por segundo ao servidor WEB Apache do provedor SuperIP

Solicitações/segundo	Qtd de ocorrências no período de 01/04/2003 a 30/06/2003
33	34
31	36
32	38
30	56
29	73
28	85
27	104
25	133
26	138
24	201

A partir da análise dos dados coletados do servidor de produção, foi extraído um padrão de tráfego a ser utilizado nos experimentos de forma a simular o comportamento do servidor de produção no ambiente de testes. Para definir esse padrão utilizou-se o método de média ponderada, conforme Tabela 9.

Tabela 9. Caracterização do tráfego a ser gerado nos testes

Solicitações / segundo	Qtd de ocorrências no período de 01/04/2003 a 30/06/2003	Total de solicitações X qtd de ocorrências
33	34	33*34=1.122
31	36	31*36=1.116
32	38	32*38=1.216
30	56	30*56=1.680
29	73	29*73=2.117
28	85	28*85=2.380
27	104	27*104=2.808
25	133	25*133=3.325
26	138	26*138=3.588
24	201	24*201=4.824
Total ocorrências	898	24.176
	Solicitações por segundo	24.176 / 898 = 27

Baseando-se nos resultados encontrados, utilizou-se o valor de 27 requisições por segundo como parâmetro para gerar tráfego nas estações cliente.

4.2.2 Realização dos Experimentos e Resultados

Para o desenvolvimento dos experimentos foram usadas as seguintes ferramentas:

Ambiente Servidor:

Para monitoração do servidor WEB Apache foi usada a ferramenta `apachectl`, fornecida juntamente com o Apache. A `apachectl` recebe os parâmetros fornecidos pelo usuário e os converte para sinais que serão enviados para o servidor WEB Apache. Da mesma forma ele verifica os códigos de saída do Apache e os transforma em mensagens de erro legíveis para o usuário. Os seguintes comandos são aceitos pela `apachectl`:

`start` - Inicia o servidor Apache;

`stop` - Finaliza o servidor Apache (enviando um sinal `TERM`);

`restart` - Reinicia o servidor Apache (enviando um sinal HUP);

`graceful` - Recarrega a configuração do servidor Apache (enviando um sinal `USR1`);

`fullstatus` - Mostra o status completo do servidor `Apache`;

`status` - Mostra o status do processo do servidor `Apache` (requer o `lynx` e o módulo *mod_status* carregado);

`configtest` - Verifica se a sintaxe dos arquivos de configuração está OK (executa um `apache -t`);

Além da `apachectl`, no servidor foi criado um *shell script* o qual coleta dados do ambiente do sistema operacional e dos processos do Apache (httpd). O propósito deste *script* é monitorar as variáveis do ambiente servidor, registrando qualquer variação neste ambiente.

As variáveis monitoradas são:

- SO:
    - Qtde de memória RAM utilizada;
        - Buffers de I/O;
        - Memória *Cache* utilizada;
    - Número de processos do sistema;
    - Utilização da CPU;
    - Memória *Swap;*
- Servidor WEB:
    - Número de conexões HTTP ativas;
    - Número de processos Apache (httpd);

Ambiente Cliente (estações):

Em cada estação cliente utilizou-se a ferramenta `ab` que acompanha o Apache. Sua função é gerar tráfego HTTP de forma parametrizada. Dentre os diversos parâmetros dessa ferramenta foram utilizadas as seguintes opções:

- - n *requests*: define o número de requisições utilizadas nos testes;
- -c *concurrency*: define quantas requisições serão executadas concorrentemente;
- -t *timelimit:* gera a quantidade máxima de requisições no tempo estipulado;

Os testes foram realizados em baterias de 2 etapas cada. Na primeira etapa foram realizados testes sem o mecanismo de rejuvenescimento e na segunda com rejuvenescimento.

Na primeira bateria de testes foram utilizadas apenas duas estações e não três, conforme especificado na Fig. 15. A terceira estação, nesta ocasião, apresentou problemas de hardware e por isso não participou dos testes.

Em cada uma das estações programou-se um *script* que utilizava a ferramenta `ab`. A linha de comando do `ab` foi composta pelos seguintes parâmetros:

Estação1:

ab -n 50000 `http://192.168.1.1/infoeste/"dbpesquisa.php3?x=2&tipo=produto&` produtos=a" > saida_estacao_1.dat

Estação2:

ab -n 50000 -c 100 `http://192.168.1.1/` > report_estacao2.dat

O valor de 50.000 requisições foi escolhido de forma *ad hoc*. Já o número de 100 conexões concorrentes foi em virtude de ser a capacidade máxima de conexões concorrentes, configuradas para este teste. Desta forma, buscou-se levar o servidor Apache a uma condição de *stress* em termos de carga de trabalho.

A linha de comando da estação 1, se referindo a um arquivo `dbpesquisa.php`, visa realizar uma consulta no banco de dados MySQL residente no servidor. Esta pesquisa possui como chave principal o caractere ‘a’, propositalmente selecionado, a fim de criar uma condição adicional de *stress* no ambiente servidor. Neste caso, apenas uma conexão é aberta para o servidor.

Já no caso da estação 2, são abertas 100 conexões simultâneas para o servidor, a fim de distribuir as 50.000 requisições programadas para a *homepage* principal.

Durante este conjunto de testes, todas as variáveis do ambiente servidor foram monitoradas e os resultados estão descritos a seguir:

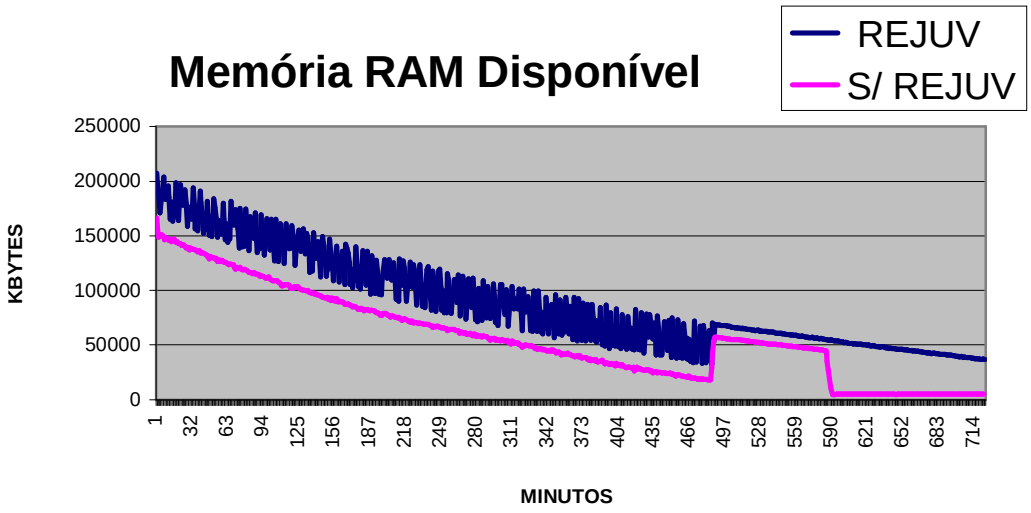

Figura 16. Memória RAM disponível (etapa 1)

É possível verificar um melhor aproveitamento de memória RAM quando usado o mecanismo de rejuvenescimento. Isto se deve ao fato de problemas de vazamento de memória, anteriormente citados no capitulo 3, os quais são acumulativos, comprometendo o ambiente do servidor como um todo, principalmente em condições de altas cargas de trabalho e longos períodos de tempo. Neste caso, os testes tiveram uma duração aproximada de 8 horas.

A estabilidade da disponibilização memória a partir dos 500 minutos dar-se-á pelo termino da execução das requisições geradas pela ferramenta `ab` configurada nas estações 1 e 2. O mesmo raciocínio aplica-se a utilização de CPU,considerando-se que em aproximadamente 500 minutos ocorre a finalização da execução dos *scripts* nas estações.

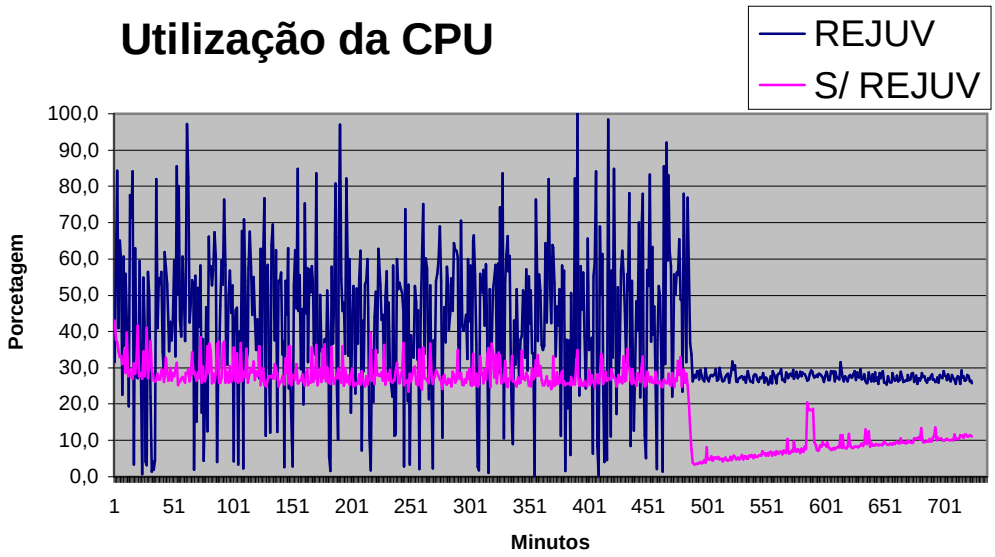

Figura 17. Utilização da CPU (etapa 1)

Como se pode observar, a CPU no ambiente com rejuvenescimento é mais utilizada, devido ao fato de que periodicamente os processos são submetidos à reinicializações, para fins de rejuvenescimento.

Nesta primeira bateria de testes, o propósito principal era a verificação da operacionalidade do ambiente de teste criado.

Com isso, a fim de validar os resultados obtidos nesta primeira bateria de testes, bem como submeter o sistema a uma carga de tráfego baseada no padrão caracterizado a partir do ambiente real, foram realizadas mais 2 baterias de testes. Ambas as baterias foram compostas de 2 etapas; a primeira sem rejuvenescimento e a segunda com o mecanismo ativado. A etapa 3 foi uma repetição da etapa 2 e foi realizada, basicamente, para se ter uma maior precisão nos resultados.

Baseando-se na caracterização de tráfego do servidor Web Apache do provedor, utilizou-se, como parâmetro, o valor de 27 requisições por segundo, gerado a partir da Tabela 5. A seguir, apresenta-se a configuração de cada estação geradora de tráfego nesta etapa.

Estação 1:

```
ab -n 19770 -c 50 http://192.168.1.1/  > saida_estacao_1.dat
```

Estação 2:

```
ab -n 19770 -c 50 http://192.168.1.1/ > saida_estacao2.dat
```

Estação 3:

```
ab -n 19770 -c 50 http://192.168.1.1/infoeste/"dbpesquisa.php3?x=2&tipo=produto& produtos=a"  > saida_estacao_3.dat
```

Tanto na estação 1 quanto na estação 2, os parâmetros utilizados foram `-n 19770`. Este parâmetro gera 19.770 solicitações distribuídas em 100 (2 vezes -c 50) conexões concorrentes. O tráfego que está sendo gerando pelo `ab` solicita a página inicial (*homepage*) que esta configurada no servidor WEB Apache. Já na estação 3, a diferença do *script* está na solicitação da página, pois o `ab` faz a solicitação de uma página em PHP, e o PHP faz uma consulta, com os parâmetros passados na solicitação, ao banco de dados MySQL.

Nesta etapa a capacidade máxima de conexões concorrentes suportadas, configurados no servidor, foi de 150 conexões. A seguir são apresentados os resultados obtidos nas etapas 2 e 3.

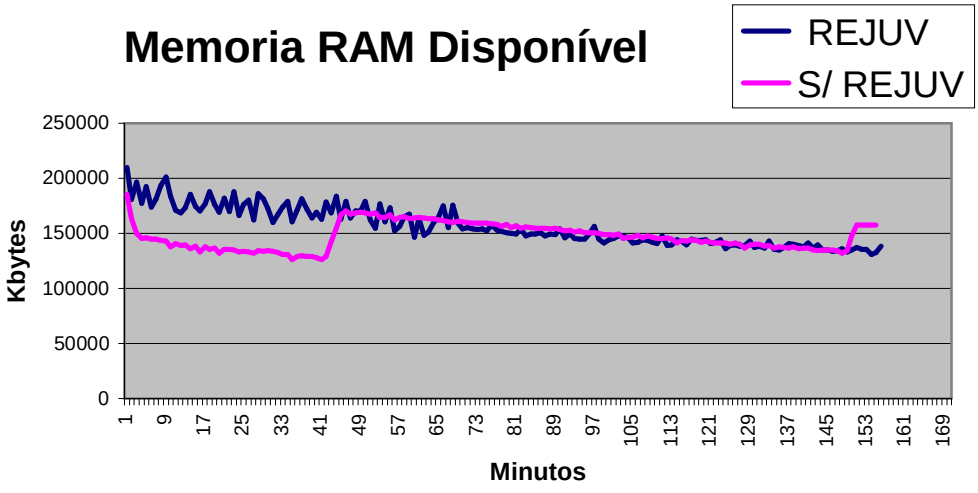

Figura 18. Memória RAM disponível (etapa 2)

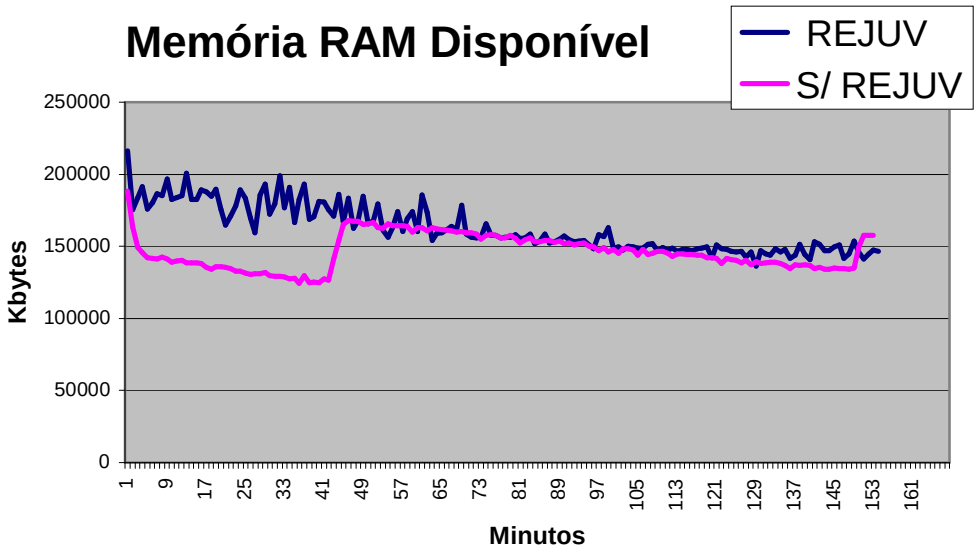

Figura 19. Memória RAM disponível (etapa3)

Nas Fig. 18 e 19 pode ser observado que nos primeiros 50 minutos todas as estações estavam em execução, ocasionando assim uma maior disponibilidade de memória RAM, conforme expressam os resultados dos testes com o mecanismo de rejuvenescimento ativo, onde tem-se uma maior disponibilidade de memória RAM. Como na primeira etapa, isso deve-se principalmente à redução pró-ativa de memória perdida em sucessivos processos de alocação e desalocação. Tais problemas foram detalhados na seção 2. A partir dos primeiros 50 minutos apenas a estação 3 continuou gerando requisições tanto com o mecanismo de rejuvenescimento como sem o mecanismo. Conclui-se que, em uma situação de *stress*, obtém-se um melhor aproveitamento de memória RAM.

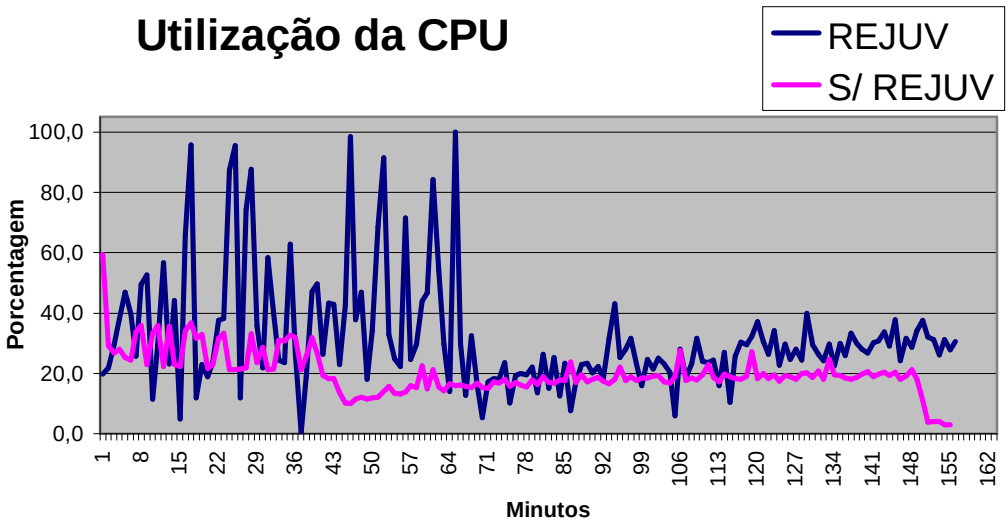

Figura 20. Utilização da CPU (etapa 2)

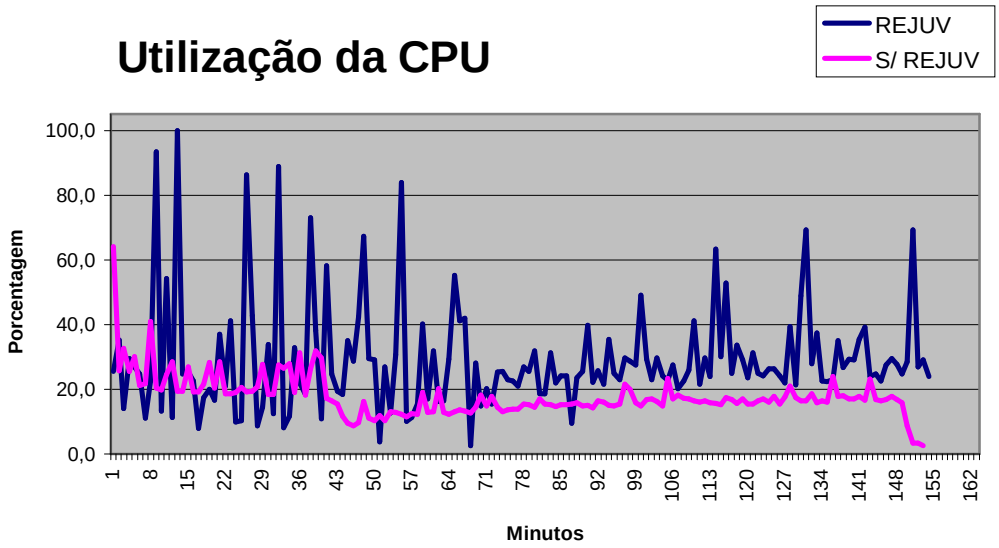

Figura 21. Utilização da CPU (etapa 3)

Em relação ao recurso utilização de CPU, a mesma análise realizada para a etapa 1 aplica-se para os gráficos mostrados nas Fig. 20 e 21.

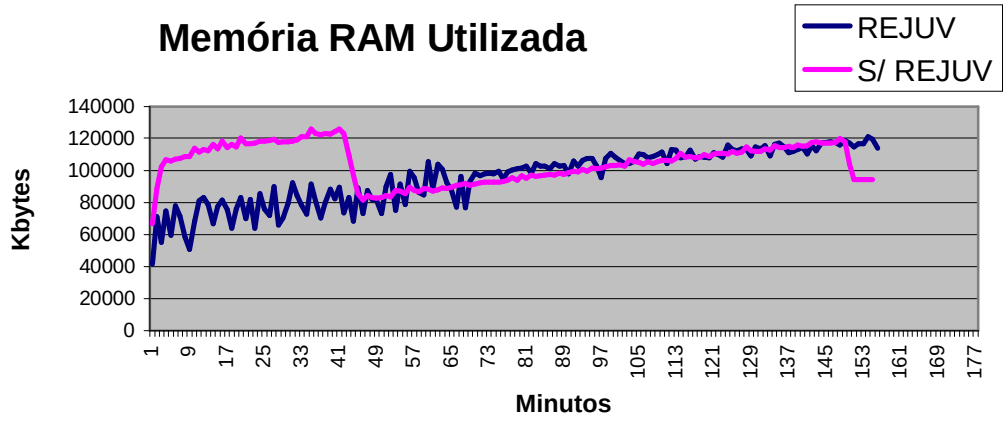

Figura 22. Memória RAM utilizada (etapa 2)

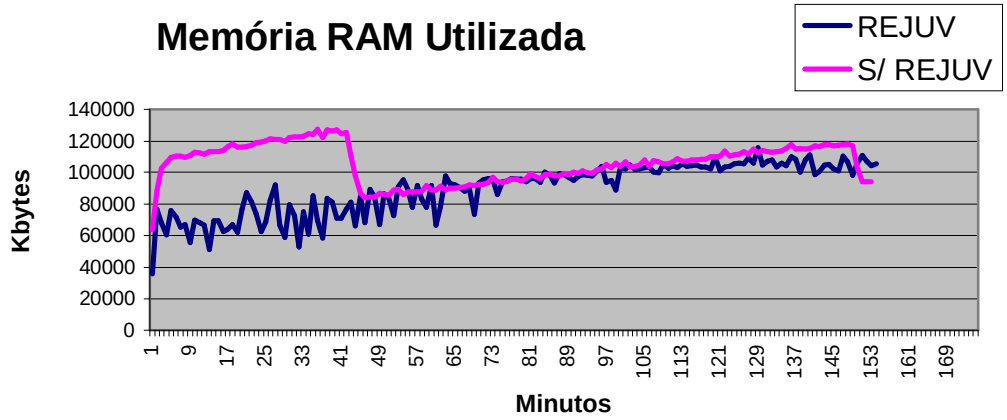

Figura 23. Memória RAM utilizada (etapa 3)

No recurso memória RAM utilizada, que representa a análise inversa das Fig. 18 e 19, pode-se observar nas Fig. 22 e 23 que enquanto o servidor WEB Apache está sob uma sobrecarga de solicitações (duração dos *scripts* das estações 1 e 2) o sistema utiliza melhor a memória RAM com o mecanismo de rejuvenescimento ativado. Depois de aproximadamente 50 minutos, ou seja, a partir do momento em que apenas a estação 3 permanece gerando tráfego, a utilização da memória RAM praticamente se iguala em ambos casos, visto não existir, neste caso, uma situação de *stress* em progresso.

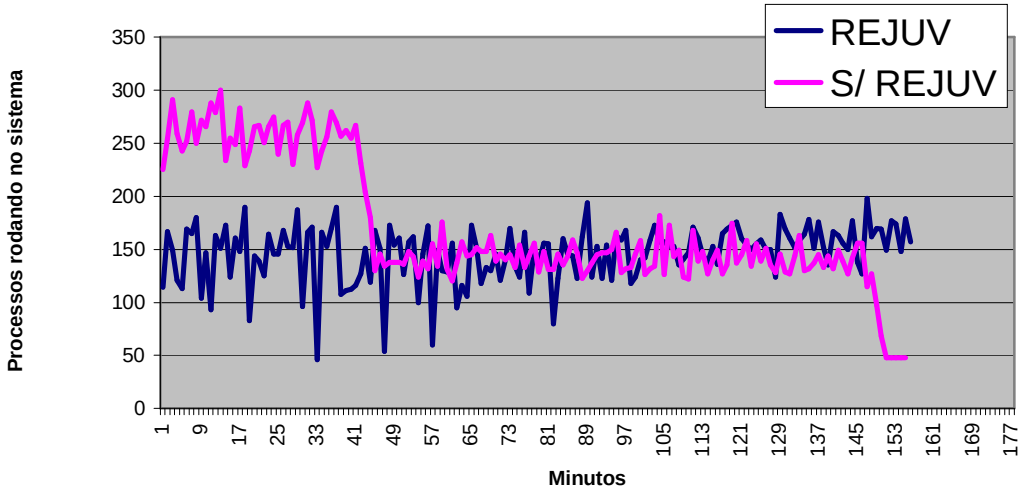

Figura 24. Quantidade de processos rodando no sistema (etapa 2)

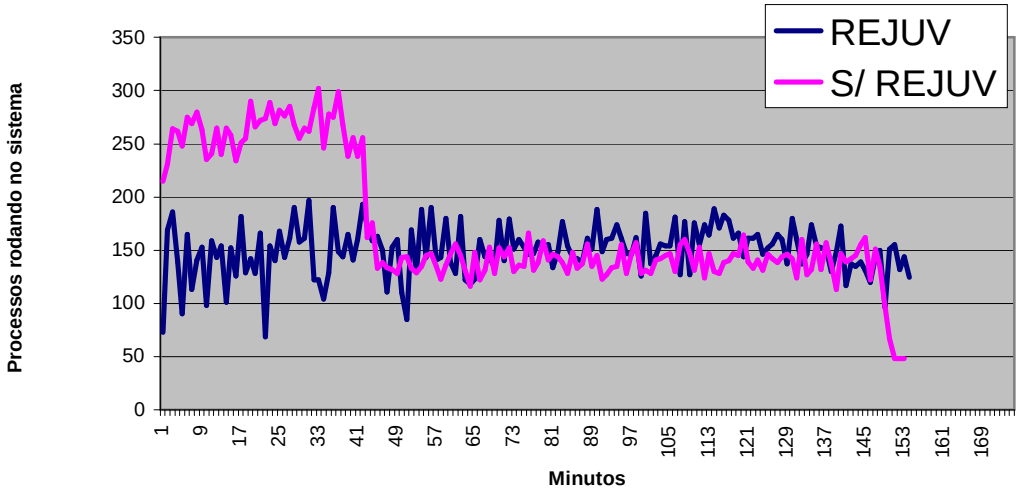

Figura 25. Quantidade de processos executando no sistema (etapa 3)

Em relação à quantidade de processos executando no sistema, pode-se observar nas Fig. 24 e 25 que os testes foram iniciados em três estações praticamente ao mesmo tempo, no entanto nota-se que em 1/3 (um terço) do teste a quantidade de processos rodando tem uma representatividade maior. Esta variação ocorre pelo fato das três estações estarem em fase de execução e, a partir do término das duas primeiras estações, a quantidade de processos se equipara. Este gráfico explica o porque do maior consumo de memória do sistema sem o mecanismo de rejuvenescimento, ou seja, durante este período do teste, existiam mais processos ativos nesta configuração.

4.3 Conclusões

Em todas etapas, os resultados do rejuvenescimento de software aplicado aos processos do Servidor WEB foram satisfatórios, principalmente no que se refere ao consumo de memória RAM.

A redução do número de processos no sistema, durante o período de maior carga (primeiros 50 minutos), demonstrou ser a causa principal para a redução do consumo de memória. Cada renovação dos processos, pelo mecanismo de rejuvenescimento de software, removia alocação excessiva de memória. Além disso, somente os processos necessários estavam carregados na memória, removendo aqueles processos bloqueados em conexões encerradas e aguardando o tempo necessário para encerrar a conexão (seção 2.4.1).

O maior consumo de CPU, como já mencionado, deve-se às reinicializações freqüentes, o que gera uma carga extra de criação e encerramento de processos no sistema. Em sistemas cuja CPU já se encontra em plena utilização, neste caso durante os primeiros 50 minutos, cargas adicionais, tais como o caso da introdução do mecanismo de rejuvenescimento, refletem significativamente na carga global do sistema.

Uma vantagem identificada no mecanismo de rejuvenescimento implementado foi quanto a inexistência de *downtime* durante suas execuções. Os processos clientes estavam sendo atendidos durante todo o período, visto que apenas os processos que encerravam suas conexões eram rejuvenescidos, antes de iniciar o tratamento de uma nova conexão.

5 CONSIDERAÇÕES FINAIS

O envelhecimento de software, dentro do contexto apresentado, é uma área bastante nova e, conseqüentemente, em franca expansão. Por se tratar de um fenômeno que pode estar presente em praticamente qualquer ambiente de execução de software, é de suma importância estudá-lo e entendê-lo, de forma a se criar mecanismos capazes de amenizar e/ou até evitar seus diversos efeitos. Por conseqüência, o rejuvenescimento atualmente é considerado o melhor mecanismo para prevenção das causas do envelhecimento, sendo mundialmente aplicado como técnica padrão para a solução de problemas nesta área.

Os resultados obtidos neste trabalho de pesquisa contribuem com a bibliografia atual, apresentando um comparativo de um ambiente WEB com e sem um mecanismo de rejuvenescimento de software. A escolha do servidor WEB Apache, atualmente o mais usado no mundo, amplia a abrangência das contribuições realizadas pelo trabalho.

O rejuvenescimento de software em servidores WEB Apache apresenta como principal objetivo a disponibilidade do serviço.

Observou-se que o envelhecimento de software em servidores WEB Apache normalmente causa paradas no sistema (servidor httpd) e para resolver este problema é comum os administradores aplicarem a técnica de reinicialização do sistema operacional, a fim de reativar o servidor WEB.

No desenvolvimento deste trabalho de pesquisa, identificou-se as seguintes técnicas de rejuvenescimento: **Total e Parcial,** que são aplicadas para minimizar os efeitos de envelhecimento de software. Os teste foram realizados utilizando-se a técnica parcial de rejuvenescimento de software, buscando assim evitar o downtime do servidor WEB em questão.

Foram realizadas três etapas de experimentos, com e sem o mecanismo de rejuvenescimento de software em um servidor WEB Apache, num período de um mês,

conforme resultados apresentados no capitulo anterior. Dentre os dados coletados para análise do consumo de recurso, o item que se mostrou com melhor desempenho foi a disponibilidade de memória RAM ao aplicar o mecanismo de rejuvenescimento.

5.1 Trabalhos Futuros

Neste trabalho aplicou-se o mecanismo de rejuvenescimento de forma *ad hoc*. Sugere-se um estudo no sentido de se determinar com precisão qual o mecanismo de rejuvenescimento mais adequado para se desenvolver técnicas que permitam determinar quando um processo deve ser rejuvenescido, isto é, quando o seu desempenho, confiabilidade ou disponibilidade podem começar a serem afetadas devido à necessidade de rejuvenescimento.

REFERÊNCIAS BIBLIOGRÁFICAS

(APACHE, 2003) **APACHE SOFTWARE FOUNDATION** - http://httpd.apache.org/

(BANGA, 1997) BANGA, G., DRUSCHEL, P. and MOGUL, J. C**. Measuring the Capacity of a Web Server.** Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997. http://www.cs.rice.edu/CS/Systems/Webmeasurement/paper/paper.html.

(BOBBIO, 1999) BOBBIO, A., GARG, S., GRIBAUDO, M., HORVATH, A., SERENO, M. and TELEK, M., - **Modeling Software Systems with Rejuvenation, Restoration and Checkpointing through Fliud Stochastic Petri Nets** - Intl. Conference on Petri Nets and Performance Models, PNPM99, September 1999. pp. 82-91.

(CALDEIRA, 2001) CALDEIRA, Carlos P., - **Redes de Computadores** Encontrado na URL: http://www.di.uevora.pt/~ccaldeira/Redes/Intro/sld001.htm Disponível em janeiro de 2003.

(CANDEA, 2001) Candea, G. and FOX, Armando - **Designing for High Availability and Measurability** - Appears in Proceedings of the 1st Workshop on Evaluating and Architecting System Dependability (EASY), July 2001

(CANDEA, 2002) CANDEA, G., CUTLER, J., FOX, A., DOSHI, R., GARG, P., GOWDA R. - **Reducing Recovery Time in a Small Recursively Restartable System** - Appears in Proceedings of the International Conference on

Dependable Systems and Networks (DSN-2002), June 2002

(CANDEA, 2003) CANDEA, G., CUTLER, J., FOX, A. - **Improving Availability with Recursive Micro-Reboots: A Soft-State System Case Study** - *Performance Evaluation Journal*, to appear Summer 2003.

(CASTELLI, 2001) CASTELLI, V., HARPER, R. E., HEIDELBERGER, P., HUNTER, S.W., TRIVEDI, K., VAIDYANATHAN, K. and ZEGGERT, W. P. – **Proactive management of software aging** - IBM Journal of Research and Development issue 45-2, Technology for xSeries Servers, 2001, pp. 311-331.

(CASTRO, 1997) CASTRO, M. A. S, – **Métodos de HTTP –**Encontrado na URL: http://www.icmc.sc.usp.br/manuals/HTML/metodos.html 1995-96-97

(CHANDRA, 2000) CHANDRA, Subhachandra - **AN EVALUATION OF THE RECOVERY-RELATED PROPERTIES OF SOFTWARE FAULTS -** A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Computer Science and Engineering) in the University of Michigan 2000

(CHANDRA, 2000a) CHANDRA, Subhachandra and CHEN, Peter M. - **Whither Generic Recovery from Application Faults? A Fault Study using Open-Source Software** Proceedings of the 2000 International Conference on Dependable Systems and Networks/Symposium on Fault-Tolerant Computing (FTCS)

(CHAPPLE, 2001) CHAPPLE, M. **Two-Tier or N-tier?** – Encontrado na URL: http://databases.about.com/library/weekly/aa040801a.htm - Join us in the forum to discuss *n*-tier databases.

(COSTA, 2001) COSTA, Antonio. **Servidores HTTP** - Encontrado na URL: http://www.dei.isep.ipp.pt/~acc/etc/ServHTTP.ppt - DEI-ISEP - Dezembro 2001.

(CROWELL, 2001) CROWELL, J. B., - **Multifractal Analysis of Memory Usage Patterns -** Thesis submitted to the College of Engineering and Mineral Resources at West Virginia University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science Morgantown, West Virginia 2001

(DOHI, 2000) DOHI, T., POPSTOJANOVA, K. G., TRIVEDI, K. - **Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule -** Pacific Rim International Symposium on Dependable Computing, PRDC 2000, December – 2000

(DOHI, 2000a) DOHI, T., GOSEVA-POPSTOJANOVA, K. and TRIVEDI, K. S. **Analysis of Software Cost Models with Rejuvenation -** IEEE Intl. Symposium on High Assurance Systems Engineering, HASE 2000, November 2000.

(DONOHUE, 2002) DONOHUE, S. K. - **Software Aging** - Department of Systems and Information Engineering. CS651 Presentation 2 December 2002. Encontrada na URL: http://www.cs.virginia.edu/~jck/cs651/slides/cs651.pres

entations/Software.aging.pdf. - Disponível em fevereiro de 2003.

(DOSS, 2001) DOSS, D., YURCIK, W., - **Achieving Fault-Tolerant** Software with Rejuvenation and Reconfiguration - 2001

(EDUCARE, 2002) EDUCARE – **O Modelo Cliente/Servidor** - http://www.educare.pt/DicioWeb/DicioWebGuiaNet_Cap2A.asp - 2002

(FERREIRA, 2002) FERREIRA, Everton. – **Dicionário de Informática e Internet –** Encontrado na URL: http://sites.uol.com.br/evertonferreira/c.htm Disponível em dezembro de 2002.

(FORREST, 2002) FORREST, S., BALTHROP, J., GLICKMAN, M. and ACKLEY, D. - **Computation in the Wild -** Dept. of Computer Science, University of New Mexico Albuquerque, NM 87131, Jul-2002

(FOX, 2001) FOX, Armando - **Capturing the Lessons of Internet Services: Recovery-Oriented Computing -** Stanford University - Encontrado na URL: http://swig.stanford.edu/pub/publications/nsf-0701-description.pdf Disponível em fevereiro de 2003.

(GARG, 1998) GARG, S., MOORSEL, A. V., VAIDYNATHAN, K. and TRIVEDI, K. - **A Methodology for Detection and Estimation of Software Aging**. In Proceedings of the 9th International Symposium on Software Reliability Engineering, pages 282–292, Paderborn, Germany, Nov 1998.

(GARG, 1998a) GARG, S., PULIAFITO, A., TELEK, M. e TRIVEDI, K. S. - **Analysis of Preventive Maintenance in**

Transactions Based Software Systems IEEE Transactions on Computers, Vol. 47 Nº 1, pp. 96-107. Jan 1998.

(GARRETT, 2001) GARRETT, Chris – **What is N-Tier? –** Encontrado na URL: http://www.aspalliance.com/chrisg/default.asp?article=43 2001.

(GROSS, 2002) GROSS, Kenny C. – **Advanced Pattern Recognition for Enhanced Dependanbility of Large-Scale Real-Time Systems and Networks –** Workshop on High Performance, Faukt Adaptive Large Scale Real - Time Systems – Sun Microsystems, 2002

(GROSS, 2002a) GROSS, K. C., BHARDWAJ, V., BICKFORD, R. **Proactive Detection of Software Aging Mechanisms in Performance Critical Computers** 27th Annual IEEE/NASA Software Engineering Symposium, Greenbelt, MD, USA, December 4-6, 2002.

(GUEDES, 2001) GUEDES, Dorgival Olavo - **WEB e HTTP** - Encontrado na URL: http://diamantina.dcc.ufmg.br/~dorgival/slides/redes/redes-llp-cap09b.pdf, UFMG, 2001 – Disponível em fevereiro 2003.

(HESSEL, 2001) HESSEL, L. S., - **ASP: UMA ALTERNATIVA PARA ACESSO A BASE DE DADOS CDS/ISIS VIA WEB** - http://www.oraculo.inf.br/ebib/anexos/orc00005.pdf, 2001

(HONG, 2002) HONG, Y., CHEN, D., LI, L., e TRIVEDI, K., - **Closed**

Loop Design for Software Rejuvenation - SHAMAN, June 2002.

(HUANG, 1995) HUANG, Y., et. al. - **Software Rejuvenation: Analysis, Modules and Applications** - *Proceedings of the 25th Symposium on Fault Tolerant Computing*, IEEE Computer Society, Pasadena, CA, pp. 381-390, June 1995.

(IBM, 2000) **IBM Director Software Rejuvenation –** Encontrado na URL: http://shannon.ee.duke.edu/Rejuv/swrejuv.pdf

(IBM, 2002) **IBM – BRASIL** – Encontrado na URL: http://www.ibm/Search?v=11&lang=pt&cc=br&q=eliza&Procurar.x=8&Procurar.y=7 Disponível em dezembro de 2002.

(IBM, 2003) **IBM xSERIES** - Encontrado Na URL: http://www.hplains.com/salesandservices/ibmxseries.php Disponível em janeiro de 2003.

(INTERQUADRAM, 2002) INTERQUADRAM - **Soluções integradas –** Encontrado na URL: www.interquadram.com.br/tecnologia.htm Disponível em dezembro de 2002.

(KC, 2002) KC, Gaurav S. - **Autonomic Systems –** September, 2002 - Encontrado na URL: http://www.cs.columbia.edu/~gskc/publications/candidacy.pdf Disponível em 14 março de 2003.

(LABOLIMS, 2002) LABOLIMS - **Aplicação Three-Tier? –** Encontrado na URL: http://www.labolims.com/estrutura.htm Disponível em dezembro de 2002.

(LI, 2002) LI, L., VAIDYANATHAN, K. AND TRIVEDI, K. - **An approach to estimation of software aging in a web server -** in *Intl. Symposium on Empirical Software Engineering (ISESE 2002)*, Nara, Japan, October 2002.

(LORRIMAN, 2000) LORRIMAN, G., - **Introduction to Multi-tier/N-tier/3-tier Architectures** – Encontrado na URL: http://www.undu.com/Articles/010131f.html Disponível em dezembro de 2002

(MARQUES, 2002) MARQUES NETO, Humberto Torres – **Envelhecimento de Software em Ambiente WEB –** Dissertação (Projeto de Mestrado) - Belo Horizonte, UFMG, 2002.

(MESENBOURG, 2001) MESENBOURG Thomas L. **Measuring the electronic economy -** U.S. Bureau of the Census, Suitland, Md. 20746. Encontrado na URL: (http://www.census.gov/eos/www/ebusiness614.htm). August, 2001.

(MESQUITA, 2002) MESQUITA, F. V., - **O Modelo Cliente/Servidor** Encontrado na URL: http://www.inf.puc-rio.br/~noemi/victal/mcs.html Disponível em dezembro de 2002.

(MOGUL, 1995) MOGUL, Jeffrey C. **The Case for Persistent-Connection HTTP.** WRL Research Report 95/4. Western Research Laboratory. 250 University Avenue. Palo Alto, California 94301 USA. 1995.

(NETCRAFT, 2003) **NETCRAFT** – Encontrado na URL: http://news.netcraft.com/ Disponível em fevereiro de 2003.

(NETWORK, 2003) **Network Performance Effects of HTTP/1.1, CSS1, and PNG (Comparing with HTTP/1.0) –** Encontrado na URL: http://www.ncnu.edu.tw/~u8321030/web_tech/HTTP1.1vs1.0.doc Disponível em fevereiro de 2003.

(NOORDERGRAAF, 2000) NOORDERGRAAF, Alex. **Building Secure N-Tier Environments** - Disponível na URL: www.sun.com/solutions/blueprints/1000/ntier-security.pdf Enterprise Engineering Sun BluePrints OnLine – October 2000

(NOVISNET, 2000) NOVISNET – **Aplicação Cliente/Servidor –** Encontrado na URL: http://www.novis.pt/article/articleview/269/1/168/ - Novis Telecom, S.A. 2000-2003

(OKAMURA, 2001) OKAMURA, H., MIYAHARA, S., DOHI, T., and OSAKI, S. **Performance Evaluation of Workload-Based Software Rejuvenation Scheme –** Issues, Vol.E84-D No.10 pp.1368-1375 2001/10.

(PARNAS, 1994) PARNAS, David Lorge – ***Software Aging*** - McMaster University, Canadá L8S 4K1, 1994.

(RFC 1945) BERNERS-LEE, T., FIELDING, R.,FRYSTYK, H., **Hypertext Transfer Protocol - HTTP/1.0 - RFC 1945** May 1996

(RFC 2616) FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T., **Hypertext Transfer Protocol -- HTTP/1.1 – RFC 2616** - June 1999

(SALLA SÁ, 2000) SALLA SÁ, Paulo Sérgio - **Gerador Automático de Arquivos HTML de Ajuda para Aplicação em Educação a Distância (GAAHA)** http://java.icmc.sc.usp.br/research/master/Paulo_Sa/tese.pdf Dissertação (Mestrado), USP - São Carlos, 2000.

(SILVA, 2000) SILVA, G. F., - **WAP – WIRELESS APPLICATION PROTOCOL –** Encontrado na URL: http://sites.uol.com.br/helyrosa/givawap/wprotocol1.html Monografia - 2000. Disponível em janeiro de 2003.

(STEIN, 1995) STEIN, D. L. - **How to set up and mantain a world wide web site: the guide for information provides**, Addison-Wesley, 1995.

(TELEPAC, 2003) TELEPAC – **Glossário –** Encontrado na URL: http://home.telepac.pt/glossario/glossario_W.html Disponível em janeiro de 2003.

(TOICH, 1998) TOICH, Shelley - **The Patriot Missile Failure in Dhahran: Is Software to Blame? -** Encontrado na URL: *http://shelley.toich.net/projects/CS201/patriot.html* February 9, 1998. Disponível em fevereiro de 2003.

(TRIVEDI, 2000) TRIVEDI, K. S., VAIDYANATHAN, K. and GOSEVA-POPSTOJANOVA, K. - ***Modeling and Analysis of Software Aging and Rejuvenation*** Proc. 33 rd Annual Simulation Symp., IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 270-279.

(TRIVEDI, 2002) TRIVEDI, K. - **Software Aging and Rejuvenation: Modeling and Analysis -** Center for Advanced Computing and Communication, Duque Universidade,

EUA, 2002. Encontrado na URL: http://renoir.csc.ncsu.edu/CACC/Events/S02/

(TRIVEDI, 2002a) TRIVEDI, K. – **Software Bugs –** Encontrado na URL: http://shannon.ee.duke.edu/Rejuv/ Disponível em dezembro de 2002.

(TRIVEDI, 2002b) TRIVEDI, K., VAIDYANATHAN, K. and MADAN, B. B. – **Software Rejuvenation: Analysis and Implementation –** Workshop on Self-Healing, Adaptive and self-MANged systems (SHAMAN), 16th Annual ACM International Conference on Supercomputing – New York, NY, June 23rd, 2002.

(TRIVEDI, 2003) TRIVEDI, K. S. - **Proactive Management of Software Systems -** Center for Advanced Computing and Communication. Dept. of Electrical and Computer Engineering Duke University.

(VAIDYANATHAN, 2001) VAIDYANATHAN, K., HARPER, R. E., HUNTER, S. W., TRIVEDI, K. S. - **Analysis and Implementation of Software Rejuvenation in Cluster Systems** - ACM SIGMETRICS 2001/Performance 2001, June 2001

(VAIDYANATHAN, 2001a) VAIDYANATHAN, K. and TRIVEDI, K. S. - **Extended Classification of Software Faults Based on Aging** – In 12th International Symposium on Software Reliability Engineering, Hong Kong, November 2001.

(VAIDYANATHAN, 2002) VAIDYANATHAN, K., SELVAMUTHU, D. AND TRIVEDI, K. S. - **Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems** - 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02) October 13 - 16, 2002

Osaka University, Suita, Japan

(YEAGER, 1996) YEAGER, N.J. e MCGRATH, R.E. - Web Server Technology - **The advanced guide for World Wide Web Information Providers**, Morgan Kaufmann Publishers, Inc., 1996.

(YURCIK, 2002) YURCIK, William and DOSS, David - **Achieving F.T. Software with Rejuvenation and Reconfiguration –** Encontrado na URL: http://www.guydavis.ca/seng/seng609.31/weekly.shtml Disponível em dezembro de 2002.

ANEXO 01

Principais diretivas do Servidor WEB Apache

Para o funcionamento de servidor WEB Apache, se faz necessário a configuração de determinadas diretivas que controlam a operação do daemon servidor, que estão dentro do arquivo httpd.conf, que é o arquivo de configuração do Apache . As diretivas apresentadas a seguir são relacionas com a versão do Apache 1.3.27 que foi utilizado para o desenvolvimento do trabalho. (APACHE, 2003)

Diretiva KeepAlive

Sintaxe: (Apache 1.2) KeepAlive on|off

Padrão: (Apache 1.2) KeepAlive on

Compatibilidade: KeepAlive só está disponível a partir da versão 1.1 do software apache.

A extensão Keep-Alive para HTTP/1.0 e a característica de conexão persistente de HTTP/1.1 possibilita sessões de HTTP duradouras, que permitem que múltiplas solicitações sejam enviadas em uma mesma conexão de TCP. Em alguns casos, esta pode mostrar um resultado de quase 50% de speedup em tempo de latência para documentos HTML com muitas imagens.

Para clientes que possuem a versão HTTP/1.0, conexões Keep-Alive só serão utilizadas caso seja especificado pelo cliente na hora da solicitação. Além disso, uma conexão Keep-Alive com um cliente HTTP/1.0 só pode ser usada quando a extensão do conteúdo for conhecida com antecedência. Isto inclui conteúdos dinâmicos como CGI e páginas SSI. Para clientes HTTP/1.1, as conexões são persistentes, a menos que estas estejam especificadas com outra versão do HTTP.

Diretiva KeepAliveTimeout

Esta diretiva determina quantos segundos o servidor apache esperará por uma solicitação subseqüente antes de fechar a conexão. Uma vez que uma solicitação tenha sido recebida, o valor de intervalo especificado pelo diretiva Timeout começa a ser contabilizado.

Caso a diretiva KeepAliveTimeout possuir um valor atribuído muito alto, isso poderá causar problemas de desempenho em servidores com uma sobrecarga de solicitações. Num alto Timeout, os processos do servidor ficarão esperando por uma conexão, com clientes inativos.

Diretiva MaxSpareServers

A diretiva MaxSpareServers estabelece o número máximo de processos no servidor de processos filhos inativos. Um processo inativo é um processo que não está controlando uma solicitação. Se existir mais processos filhos ociosos do que o número especificado na diretivaMaxSpareServers, o processo de pai exterminará os processos em excesso.

Esse procedimento só se faz necessário em sites com uma grande demanda de solicitações.

Diretiva MaxClients

A diretiva MaxClients é configurada para delimitar o número máximo de solicitações simultâneas que podem ser aceitadas pelos processos filhos do servidor WEB.

Diretiva StartServers

A diretiva StartServers determina o número de processos filhos que servidor criará ao ser inicializado.