Modelo de servidor Web com conexões persistentes e carga orientada a sessão

Álvaro Sant'Anna Filho

Orientador: *Prof. Dr. Marcos José Santana*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Área: Ciências de Computação e Matemática Computacional.

USP - São Carlos
outubro/2004

A ***Comissão Julgadora:***

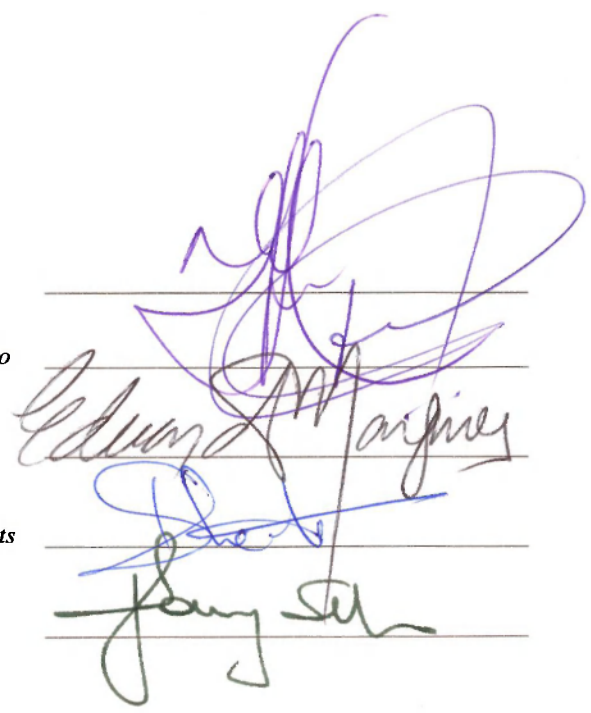
Prof. Dr. Marcos José Santana

Prof. Dr. José Carlos Maldonado

Prof. Dr. Eduardo Marques

Prof. Dr. Jan Frans Willem Slaets

Prof. Dr. Jorge Luiz e Silva



À Elenice, minha amada esposa, pois sem ela este trabalho não teria acontecido.

Agradecimentos

Agradeço ao professor Marcos José Santana pela paciência durante as diversas fases do curso e relativamente à orientação no trabalho. À professora Regina pelas críticas construtivas nas diversas etapas do curso.

Gostaria ainda de agradecer a todos colegas que me incentivaram e ensinaram a partir do Mestrado. Especialmente aos alunos dos cursos de Pós-Graduação em Ciência de Computação e Engenharia de Produção da UFSC (Florianópolis); aos alunos de Graduação em Ciência de Computação da UFOP (Ouro Preto); aos alunos de Pós-Graduação em Ciência de Computação da UFMG (Belo Horizonte) e finalmente, em particular, aos colegas do LaSDPC do ICMC da USP em São Carlos.

Agradeço também o apoio e ajuda de todos professores e funcionários da USP-São Carlos, em particular aqueles do ICMC.

Não vou citar nomes embora devesse, porque o apoio e auxílio de certas pessoas foram cruciais para prosseguir. Por outro lado, esses amigos e amigas especiais formariam uma bela lista e eu com certeza faria injustiça deixando de citar alguém.

Resumo

Esta tese de doutorado apresenta um modelo de servidor Web mostrando as componentes internas básicas do software servidor, representando a comunicação completa entre cliente e servidor.
O modelo construído pode ser usado para avaliação de desempenho e toma como base um modelo já existente no qual são feitas uma extensão e uma modificação inéditas.
As principais técnicas usadas foram a teoria das filas, parametrização do modelo, solução do modelo por simulação e a validação através de observações obtidas em experimentos controlados de laboratório.
Como contribuições citam-se aqui: (i)a extensão para conexões persistentes, o que diz respeito a uma característica já padronizada no protocolo HTTP 1.1, porém não considerada no modelo base anterior; (ii) a modificação quanto à carga de chegada, vindo esta segunda contribuição a constituir duas melhorias. A carga orientada a sessão é de modelagem mais fácil e a distribuição da chegada de sessões tem representação mais fidedigna do que a distribuição da chegada de documentos, conforme considerado no modelo base.

Abstract

This doctorate thesis presents a Web server model showing basic internal software components, regarding the complete communication between client and server.
The model is built from a base, already existing model in which an extension and modification are proposed.
The techniques used were queueing theory, model parametrization, model solution through simulation and validation by measures obtained in controlled laboratory experiments.
The main contributions are: (i) the extension for persistent connections, already a standard in HTTP 1.1, which had not been considered in the previous model; (ii) the modification of the arrival workload, bringing two improvements. The session oriented load is more easily modeled and the distribution of session arrival has a good representation, as compared to the distribution of document arrival considered in the base model.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAL ATM Adaptation Layer

ADSL Asymetric Digital Subscriber Line

AIX Advanced Interactive Executive

ASP Active Server Pages

ATM Asynchronous Transfer Mode

B-ISDN Broadband-Integrated Services Digital Network

BGP Border Gateway Protocol

CGI Common Gateway Interface

CISC Complex Instruction Set Computer

COM Component Object Model

CORBA Common Object Request Broker Architecture

DBMS Data Base Management System

DNS Domain Name System

DOM Distributed Object Model

EJB Enterprise Java Beans

ERP Enterprise Resource Planning

FDDI Fiber Distributed Data Interface

GUI Graphical User Interface

HTML Hypertext Message Language

HTTP Hypertext Transfer Protocol

IBM International Business Machines

ICMP Internet Control Message Protocol

IDL Interface Definition Language

IETF Internet Engineering Task Force

IGMP Internet Group Message Protocol

IIOP Internet Inter-ORB Protocol

IP Internet Protocol

ISDN Integrated Services Digital Network

ISO International Organization for Standardization

ITU International Telecommunications Union

LAN Local Area Network

LQM Layered Queueing Model

MFLOPS Million Floating-poin Operations Per Second

MIB Management Information Base

MIME Multipurpose Internet Mail Extensions

MIPS Million Instructions Per Second

MOL Method of Layers

MSS Maximum Segment Size

MVA Mean Value Analysis

NCSA National Center for Supercomputing Applications

NFS Network File System

NIS Network Information Service

NNTP Network News Transfer Protocol

NSAPI Netscape Server Application Program Interface

OMG Object Management Group

ORB Object Resource Broker

OSI Open System Interconnection

OSPF Open Shortest Path First

QNM Queueing Network Model

RAM Random Access Memory

RFC Request For Comments

RISC Reduced Instruction Set Computer

RPC Remote Procedure Call

SGML Extended Generalized Markup Language

SMDS Switched Multi-megabit Data Service

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SPEC Standard Performance Evaluation Corporation

SQL Structured Query Language

SRVN Stochastic Rendez-Vous Networks

SURGE Scalable URL Reference Generator

TCP Transmission Control Protocol

TP Transaction Processing

UDP User Datagram Protocol

URL Uniform Resource Locator

W3C World Wide Web Consortium

WAN Wide Area Network

XML Extensible Markup Language

XSL Extensible Stylesheet Language

CAPÍTULO

1

Introdução

1.1 Motivação

Com o advento da tecnologia Web os servidores Web se tornaram elementos do núcleo das redes de comunicação (van der Mei *et al.*, 2001). Assim, o estudo da configuração adequada e limitações de desempenho de servidores Web passaram a ser fundamentais.

O desempenho de um servidor Web resulta da interação entre uma variedade de componentes: a plataforma de hardware do servidor, o software servidor Web, o sistema operacional, a largura de banda da rede, os tamanhos dos arquivos requisitados, o sistema de cache existente, entre outras.

A experiência tem mostrado que o desempenho desses servidores pode ser bastante melhorado configurando-se apropriadamente os componentes do servidor; para isso é necessário entender-se como os componentes interagem, como influenciam no desempenho de um extremo a outro (do servidor ao cliente).

Fica mostrada desse modo a utilidade de um modelo de servidor Web considerando suas componentes internas, que auxilie, por exemplo, na tomada de decisões visando mudanças no hardware ou software servidor.

1.2 Problema objetivo a ser tratado

Muitos trabalhos importantes na literatura abordaram aspectos específicos do desempenho dos servidores Web (capítulo 4), embora sem considerar (i)um modelo para a completa comunicação entre cliente e servidor nem (ii)levando em conta as componentes internas do servidor, em particular quanto ao software servidor.

Sendo assim, tornou-se objetivo básico desta tese de doutorado a construção de um modelo original de servidor ou a modificação e/ou extensão de um modelo já existente, com essas duas características desejáveis, que pudesse servir adequadamente para avaliação de desempenho.

Uma extensa revisão bibliográfica da literatura disponível encontrou apenas um modelo já existente, com as características desejadas, sendo esse modelo descrito no capítulo 4 e tomado como base nesta tese.

O modelo existente, porém, considera que para cada documento Web requisitado, uma conexão é estabelecida e rompida, após o envio de cada documento ao cliente. Atualmente o protocolo HTTP em sua versão 1.1 já considera como padrão conexões persistentes (múltiplos documentos são enviados na mesma conexão), o que em geral acarreta vantagens de desempenho em termos de tempo de resposta do cliente, diminuindo concomitantemente o tráfego na rede entre cliente e servidor. Nesta tese é proposta a extensão do modelo para o caso de conexões persistentes.

Além da extensão do modelo base considerando conexões persistentes, uma modificação é proposta quanto à carga de chegada usada no modelo base original. No modelo original considerado, a carga de chegada (carga de trabalho do servidor Web modelado) é de documentos na unidade de tempo. A chegada de documentos é modelada como sendo um processo de Poisson, o que facilita a modelagem por um lado, mas não corresponde em geral à realidade, conforme diversos trabalhos na literatura. Sendo assim, propõe-se nesta tese que a carga de chegada seja de sessões de usuários (junto a um mesmo servidor Web), a qual é descrita na literatura (Liu *et al.*, 2001) como sendo geralmente um processo de renovação, podendo ser modelado em alguns casos como um processo de Poisson.

Contudo, como internamente o servidor Web (o modelo) enxerga o processamento (requisição e envio) de documentos, é feita uma transformação da carga de entrada de sessões por segundo para documentos por segundo, considerando as distribuições do número de páginas por sessão, do número de documentos por página e do tamanho dos documentos solicitados. Note-se que um conjunto de documentos irá constituir a página Web a ser visualizada pelo *browser* para o usuário e um conjunto de páginas (do mesmo sítio) irá constituir uma sessão.

Assim, a carga de chegada de sessões pode ser usada como entrada para o modelo, sendo a carga de sessões de mais fácil modelagem que a de documentos, conforme descrita, por exemplo em (Liu *et al.*, 2001).

1.3 Cenário global e áreas abrangidas no trabalho

O trabalho da tese se situa no contexto de Sistemas Distribuídos e, em particular, a WWW, incluindo diversos aspectos pertinentes, direta ou indiretamente relacionados. Também

está incluída a área de modelagem e avaliação de desempenho de servidores Web. O cenário global e outras áreas relacionadas à tese são vistos nos capítulos 2 e 3.

1.4 Técnicas consideradas

As principais técnicas consideradas no presente trabalho foram relativas a modelagem e avaliação de desempenho de servidores Web, especialmente as seguintes: uso de um dos *benchmarks* para servidores Web citados no capítulo 3 (subseção de *benchmarks* para servidores Web) para simular clientes Web requisitando sessões a um servidor Web em laboratório; teoria das filas; parametrização de modelos de servidores Web; solução de modelos usando simulação e validação usando observações em experimentos sob controle em laboratório.

Essas técnicas citadas são abordadas inicialmente no capítulo 3, de uma maneira geral; posteriormente, nos capítulos seguintes, são utilizadas em cada contexto específico correspondente e, em especial, no capítulo 6 (solução e validação do novo modelo proposto).

1.5 Organização do trabalho

Esta tese de doutorado está organizada em sete capítulos e neste primeiro se faz uma introdução expondo a motivação, objetivos e técnicas principais usadas.

No segundo capítulo são abordados redes de computadores, sistemas distribuídos e a WWW. É descrito o cenário global, sendo citados alguns aspectos importantes relativos à tese, com observações e dados que possam vir a ser utilizados ou citados em capítulos posteriores.

No terceiro capítulo são descritos aspectos gerais de modelagem e avaliação de desempenho de sistemas computacionais inicialmente, sendo em seguida abordado especificamente o caso particular referente a servidores Web.

No capítulo 4, primeiramente são revisados importantes trabalhos da literatura que abordam aspectos específicos de modelagem e desempenho de servidores Web. Em seguida é descrito um modelo existente (proprietário) de servidor Web com as características desejáveis, conforme citado na segunda seção deste capítulo.

No capítulo 5 são apresentadas e descritas em mais detalhes as principais contribuições para o novo modelo proposto (construído com base no modelo descrito no capítulo 4): uma modificação e uma extensão inéditas em relação ao modelo anteriormente descrito.

No capítulo 6 o novo modelo resultante é resolvido por simulação e validado através de medições experimentais em laboratório.

No capítulo 7 são apresentadas as conclusões e comentários finais, juntamente com sugestões de trabalhos futuros.

CAPÍTULO

2

Redes de Computadores, Sistemas Distribuídos e a WWW

2.1 Introdução

As redes de computadores usadas atualmente tiveram sua origem na ARPANET, uma rede de computadores de chaveamento de pacotes, desenvolvida a partir do final dos anos 60 sob liderança e patrocínio da ARPA (Advanced Research Projects Agency), do Departamento de Defesa (DoD) dos Estados Unidos (Menasce & Almeida, 1998).

A ARPA foi criada em resposta ao lançamento do Sputnik em 1957 pela URSS, tendo como missão desenvolver tecnologia com fins militares (Tanenbaum, 1996). Em meados da década de 60, época do apogeu da Guerra Fria, o DoD desejava uma rede de controle e comando que pudesse sobreviver a uma guerra nuclear. As redes de telefonia tradicionais, com chaveamento de circuitos, foram consideradas muito vulneráveis e a idéia de comutação de pacotes, sugerida por Paul Baran (Tanenbaum, 1996) numa série de artigos publicados pela RAND Corporation (organização também encarregada de planejar estratégia na guerra do Vietnam) veio a ser adotada. Assim, decidiu-se que a rede que o DoD precisava seria uma rede de comutação de pacotes, consistindo de uma subrede e computadores hospedeiros (*hosts* ou *end systems*).

Preocupações relativas a desempenho estiveram presentes desde o início. Leonard Kleinrock (University of California at LA) liderou a pesquisa que desenvolveu modelos de filas de redes de chaveamento de pacotes e estabeleceu instalações de medições e gerência de redes para coleta e interpretação de um grande número de dados extremamente úteis, que foram usados para compreender e reprojetar a rede e seus protocolos (Kleinrock, 1975), (Kleinrock, 1976). O termo Internet surgiu em 1983 quando a ARPANET foi dividida em duas redes, uma militar (a MILNET) e uma versão reduzida da ARPANET, de onde se originou a Internet. Hoje ela é um

grande conjunto de WANs interligadas mundialmente. A base da Internet é seu protocolo de rede, IP, e o protocolo de comunicação entre processos, TCP, desenvolvido por (Cerf & Kahn, 1974).

Os sistemas distribuídos surgiram na década de 1980 e a WWW foi concebida em 1990, apresentando a partir de 1995 uma expansão sem precedentes que continua até hoje.

2.2 Redes de Computadores

As redes de computadores são usualmente classificadas como WANs (*Wide Area Networks*) ou LANs (*Local Area Networks*) de acordo com o tamanho da área coberta pela rede (Menasce & Almeida, 1998). Alguns autores citam MANs (*Metropolitan Area Networks*), que são consideradas aqui um caso particular de LANs.

2.2.1 Redes de Longa Distância (WANs)

As WANs podem abranger uma cidade, várias cidades, países, ou continentes (Comer, 1997). A tecnologia básica usada é a de comutação de pacotes. As mensagens transmitidas entre dois computadores são divididas em unidades chamadas pacotes, tendo um tamanho máximo e cabeçalho com campos contendo a informação de endereçamento necessária para que um pacote seja roteado de sua origem ao destino; e também a seqüência da informação, necessária para reagrupar a mensagem no destino a partir de seus pacotes constituintes. Os blocos construtivos de uma WAN são os comutadores de pacotes (*packet switches*, *packet switched nodes*, ou *intermediate systems*) e linhas de ligação de alta velocidade, por exemplo de 45 Mbps a vários Gigabits/s, conectando os comutadores de pacotes. Estes são computadores de comunicação que armazenam pacotes que chegam, examinam os cabeçalhos e fazem busca em tabelas de roteamento para tomar decisões sobre o próximo *packet switching node* (ou roteador) para onde deve ser enviado um pacote a caminho de seu destino; em seguida coloca o pacote na fila de saída para o *link* selecionado. Esse é o chamado método *store-and-forward*. Algumas técnicas para se construir WANs são: o X.25 que é um padrão do ITU (International Telecommunications Union) seguido por muitas redes públicas mais antigas, especialmente fora dos Estados Unidos; ISDN e B-ISDN, esta última visando um sistema (já em execução) para integração futura total de serviços diversos; *frame relay*, para transmissão com velocidade razoável e custo baixo, SMDS (*Switched Multi-megabit Data Service*), primeiro serviço de comutação de banda larga oferecido e ATM, que usa pacotes pequenos de tamanho fixo, para prover chaveamento rápido para voz, vídeo e dados em WANs (Comer, 1997).

2.2.2 Redes Locais (LANs)

As LANs são distintas por algumas características: seu tamanho, restrito a alguns quilômetros, faz com que o tempo de transmissão no pior caso seja limitado e conhecido de antemão. Sua tecnologia de transmissão, freqüentemente usa um único cabo ao qual todas as máquinas são conectadas. As LANs tradicionais transmitem dados com velocidades da ordem de 10 a centenas de Mbps (ou até mais, Gbps), têm retardo baixo e apresentam pouquíssimos erros. As tecnologias mais usadas para LANs são: a *Ethernet* com velocidades de transmissão de 10 Mbps, 100Mbps a alguns Gbps (*Fast Ethernet*) na atualidade; *Token Ring* de 4 ou 16 Mbps; e FDDI (*Fiber Distributed Data Interface*).

2.2.3 Conexão LAN-WAN e Conexão Doméstica a uma WAN

As LANs usualmente se conectam às WANs através de linhas alugadas dedicadas a velocidades T1 (1,544Mbps) ou T3 (45Mbps) (Menasce & Almeida, 1998). Por outro lado, existem correntemente muitas alternativas para se conectar computadores domésticos, ou mesmo LANs domésticas a uma WAN. O mais simples e barato é usar modens analógicos de discagem a velocidades (baixas) variando de 14,4 a 56Kbps, ou ainda além. A alternativa mais rápida seguinte, ISDN BRI (*Basic Rate Interface*), requer um modem digital de discagem e provê velocidades de 128Kbps. Se velocidades mais altas são requeridas dos serviços ISDN, pode-se usar ISDN PRI (*Primary Rate Interface*) que transfere 1,544Mbps. A mesma velocidade pode ser obtida alugando-se uma linha T1. Velocidades como a de T1 podem ser também obtidas com a HDSL (*High-Bit-Rate Digital Subscriber Line*) mas com mais flexibilidade. Uma versão assimétrica de HDSL, a ADSL, provê 640 Kbps de saída e 6 Mbps de entrada. Essa assimetria é vantajosa para acesso à Web, já que os requisitos de largura de banda para *downloads* rápidos de imagem e vídeo são mais altos do que para se enviar requisições a servidores Web.

2.2.4 Protocolos

Introdução

O software de redes é estruturado através de camadas construídas uma sobre a outra. O número de camadas, nome, conteúdo e função de cada camada diferem de uma rede para outra, mas o propósito de cada camada é oferecer certos serviços às camadas superiores, escondendo dessas camadas os detalhes de como esses serviços são implementados (Tanenbaum, 1996). As regras e convenções usadas na comunicação entre entidades pares de uma mesma camada constituem o protocolo dessa camada, ou seja, um protocolo é um acordo entre entidades comunicantes sobre como a comunicação deve proceder.

Entre cada par de camadas adjacentes (superior e inferior) existe uma interface, que

define quais operações primitivas e serviços a camada inferior oferece para a superior. Um conjunto de camadas e protocolos constitui uma arquitetura de rede.

No caso de protocolos sem conexão, as mensagens são independentes uma da outra e podem chegar ao destino numa ordem distinta da que foram transmitidas. Esse tipo de protocolo é bom quando os dados a serem trocados entre as duas entidades se ajustam à máxima unidade de dados permitida pelo protocolo. Desse modo a mensagem não tem que ser fragmentada em mais de uma unidade de dados e a seqüência não é uma preocupação.

Protocolos orientados a conexão são usados quando devem ser transmitidas mensagens maiores do que a unidade de dados máxima. Nesse caso, a seqüência e recuperação de erros são importantes. Antes que duas entidades comunicantes iniciem a troca de mensagens, uma conexão entre eles tem que ser estabelecida. Os protocolos orientados a conexão são úteis quando grandes arquivos precisam ser transferidos entre os comunicantes porque a seqüência de mensagens, recuperação de erros e controle de fluxo são considerações importantes para esse tipo de aplicação.

Principais Modelos de Referência para Arquiteturas de Rede

Os dois modelos principais de referência para arquiteturas de rede são conhecidos como o modelo OSI (*Open Systems Interconnection*) da ISO (*International Organization for Standardization*) e o modelo TCP-IP. A Figura 2.1, apresentada em (Tanenbaum, 1996) mostra o modelo OSI.

O modelo OSI é baseado numa proposta desenvolvida pela ISO como uma primeira etapa rumo à padronização internacional dos protocolos usados nas várias camadas; possui 7 camadas (a IBM, na época dominando a indústria de computadores, possuía uma arquitetura proprietária com 7 camadas) conforme a figura mostrada. Cada entidade duma camada *n* se comunica somente com entidades remotas da enésima camada. Uma entidade na camada *n* usa serviços providos por entidades da camada *n-1*.

O modelo TCP-IP é nomeado em função de seus dois principais protocolos, TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*), os dois mais importantes protocolos na Internet.

A Figura 2.2 compara os modelos OSI e TCP-IP.

O modelo TCP-IP pode ser considerado como tendo 4 camadas. Abaixo da camada de rede (internet) o modelo apenas prevê que o hospedeiro tem que se conectar à rede usando algum protocolo para envio de pacotes IP através dela; porém esse protocolo não é definido.

Os modelos OSI e TCP-IP têm algo em comum, sendo ambos baseados no conceito de uma pilha de protocolos independentes. A funcionalidade das camadas é grosseiramente similar. Por exemplo, em ambos modelos as camadas até (e inclusive) a camada de transporte existem

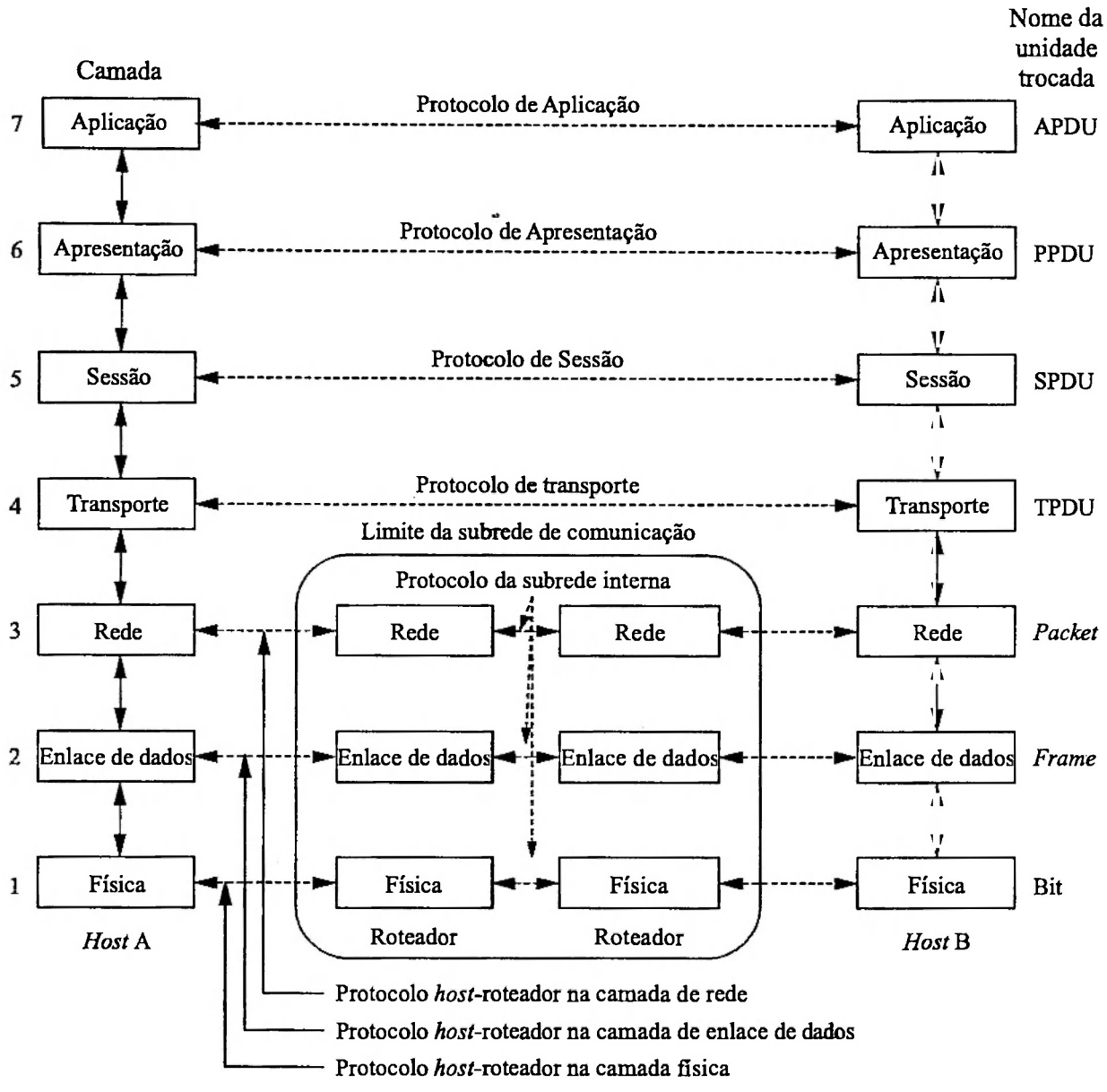


Figura 2.1: O modelo de referência OSI da ISO (Tanenbaum, 1996)

para prover um serviço de transporte fim-a-fim, independente da rede, para processos querendo se comunicar. Essas camadas formam o provedor de transporte. Em ambos modelos, as camadas acima da de transporte são usuárias do serviço de transporte, orientadas para uma aplicação. Apesar das similaridades, os dois modelos têm muitas diferenças (Tanenbaum, 1996).

Internet Protocol

O protocolo IP especifica os formatos de pacotes enviados através da Internet e os mecanismos usados para levar adiante esses pacotes através de um conjunto de redes e roteadores, da fonte ao destino.

Cada *host* conectado à Internet tem um endereço chamado endereço IP, o qual é único para toda Internet. Esse endereço, um número de 32 bits, é usualmente representado por uma

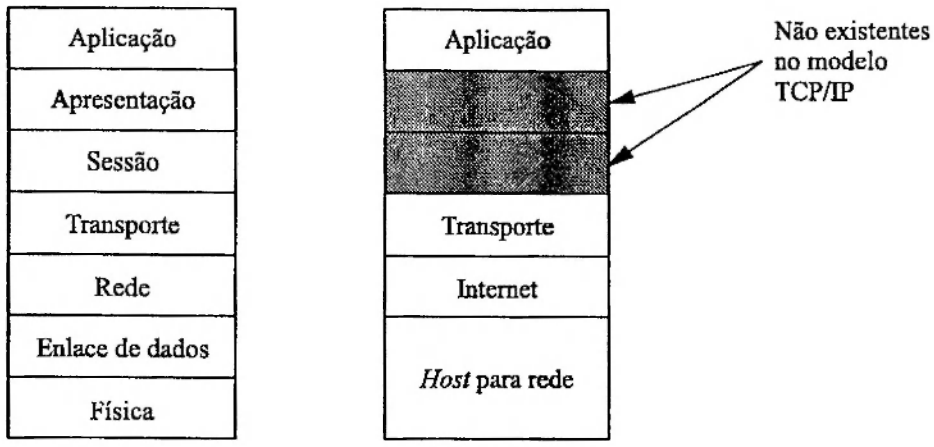


Figura 2.2: Comparação entre os modelos OSI e TCP-IP (Tanenbaum, 1996)

notação decimal com pontos, por exemplo, 129.2.0.37, onde cada um dos quatro números, na faixa de 0 a 255, representa o valor de 8 bits no endereço de 32 bits. Os 32 bits são divididos em 2 partes: um prefixo e um sufixo. O prefixo é usado para indicar uma rede e o sufixo um computador ***host*** dentro da rede. O número de bits alocados ao prefixo determina a quantidade (o número) de números de redes únicas (singulares) e o número de bits no sufixo determina o número de ***hosts*** por rede (Menasce & Almeida, 1998).

A unidade de dados transportada pelo IP é chamada de datagrama IP. O IP é um protocolo sem conexão, que não fornece garantia de entrega de um extremo a outro: os datagramas podem ser perdidos. Além disso, dois datagramas IP enviados em sequência de uma máquina fonte para o mesmo destino são roteados independentemente e podem chegar fora de ordem no destino (Stevens, 1994). O IP encarrega o protocolo TCP de cuidar da recuperação, no caso de erros. Todo ***host*** e roteador na Internet implementam o protocolo IP. A implementação em um roteador mantém uma tabela de roteamento na memória que é usada para procurar pelo próximo roteador ou ***host*** (se for o destino final) para enviar adiante o datagrama, conforme a figura 2.3.

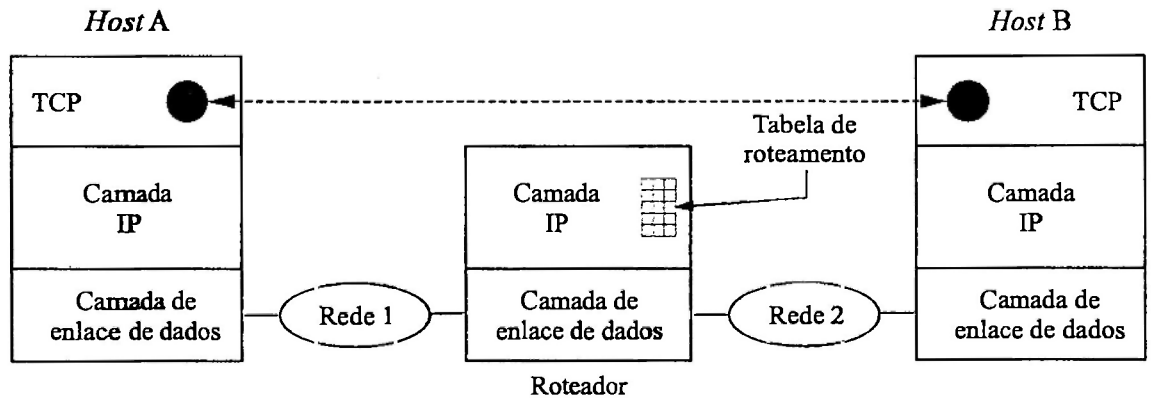


Figura 2.3: Protocolo IP em *hosts* e roteadores (Menasce & Almeida, 1998)

A versão corrente IPv4, com seu campo de endereçamento de 32 bits está alcançando suas limitações, conforme o número de *hosts* ligados à Internet cresce extensivamente.

A partir de 1990, o IETF (Internet Engineering Task Force) passou a trabalhar numa nova versão do IP, IPv6, onde não faltariam números de endereços; resolveria ainda uma variedade de outros problemas e seria mais flexível e eficiente. Os principais objetivos foram (Tanenbaum, 1996): dar suporte a bilhões de hosts, mesmo com alocação de espaço de endereçamento insuficiente; reduzir o tamanho das tabelas de roteamento; simplificar o protocolo, para permitir aos roteadores processar os pacotes mais rapidamente; fornecer melhor segurança (autenticação e privacidade) do que o IP corrente; dar mais atenção ao tipo de serviço, particularmente para dados de tempo real; auxiliar comunicação em grupo (*multicasting*) permitindo formatos a serem especificados; tornar possível a um host mudar de local sem mudar seu endereço; permitir ao protocolo evoluir no futuro e fazer com que o antigo e os novos protocolos coexistam por anos.

O IPv6 atinge os objetivos mantendo as boas características do IP e adicionando novas características onde necessário. Em geral, o IPv6 não é compatível com IPv4 mas sim com todos outros protocolos Internet incluindo TCP, UDP, ICMP (Internet Control Message Protocol, IGMP (Internet General Message Protocol), OSPF (Open Short Path First), BGP (Border Gateway Protocol), e DNS. O IPv6 tem endereços mais extensos que o IPv4, com 16 bytes, o que resolve o problema de prover quantidade ilimitada de endereços Internet. A segunda grande melhoria do IPv6 é a simplificação do cabeçalho, contendo apenas 7 campos (em relação a 13 no IPv4). Essa mudança permite aos roteadores processar pacotes mais rapidamente e dessa forma melhorar o *throughput*. A terceira importante melhoria foi melhor suporte para opções. Essa mudança foi essencial com o novo cabeçalho porque campos que eram requeridos anteriormente agora são opcionais. Em acréscimo, o modo com que as opções são representadas é diferente, tornando simples para os roteadores deixar de lado as opções não intencionadas a eles. Essa característica melhora o tempo de processamento de pacotes. Uma quarta área em que o IPv6 apresenta um grande avanço é em segurança, onde autenticação e privacidade são características chave do novo IP. No IPv6 deu-se mais atenção ao tipo de serviço do que antes. O IPv4 tem um campo de 8 bits dedicado a isso, mas com o crescimento esperado de tráfego multimídia, muito mais é necessário.

Várias escolhas feitas pelo IPv6 foram controversas especialmente no que tange a aspectos de segurança. Dado o investimento maciço em roteadores IPv4 usados no presente, o processo de conversão ainda levará anos devendo ser feito de forma a não causar problemas.

Transmission Control Protocol

O TCP provê um serviço de comunicação de um extremo a outro com fluxo controlado, confiável, orientado a conexão, entre processos executando em hosts ligados através de uma internet. O TCP garante que os dados serão entregues na ordem transmitida, sem dados perdidos. Permite que dois pontos terminais de uma conexão troquem dados simultaneamente, ou seja em modo *full-duplex*. Fornece uma interface de fluxo, o que significa que ele aceita um fluxo contínuo de bytes da aplicação a ser enviado através da conexão. A PDU (*Protocol Data Unit*) a

ser trocado no nível TCP é chamada de segmento. O cabeçalho de um segmento TCP ocupa 20 bytes. O TCP envia segmentos dentro de datagramas IP.

Antes que dados possam ser enviados entre 2 *hosts*, uma conexão deve ser estabelecida entre eles. O TCP implementa um mecanismo de estabelecimento de conexão confiável chamado *three-way handshake* (Comer, 1997), (Stevens, 1994). A figura 2.4 mostra a troca de segmentos que ocorre para estabelecimento de conexão, transmissão de dados e término de conexão.

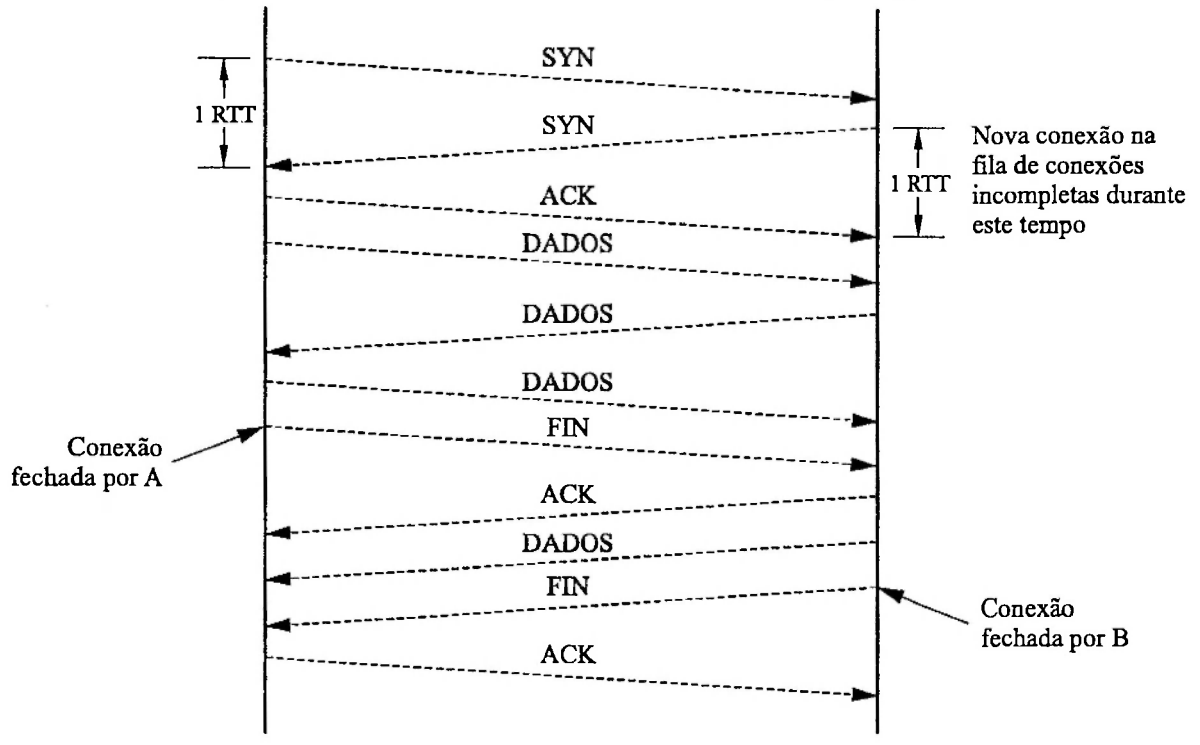


Figura 2.4: Estabelecimento e término de conexão TCP

As linhas verticais são eixos de tempo nos *hosts* A e B, com o tempo crescendo de cima para baixo. As setas diagonais indicam segmentos sendo enviados entre A e B. O *host* A inicia enviando um segmento de sincronização (SYN) para B, que responde com outro SYN. Essa troca de segmentos de SYN leva 1 RTT (tempo de viagem de ida e volta). A conexão no *host* B não é considerada completa até que um segmento de ACK (*acknowledgement*) de A chegue. Durante o intervalo de tempo desde que B envia seu SYN para A até que receba o ACK de A, a conexão permanece numa fila de conexões incompletas (permanece nessa fila durante 1 RTT). Se um host recebe muitas requisições para abrir conexões de servidores remotos, a fila de conexões incompletas pode ficar cheia, impedindo-o de aceitar novas conexões. Esse problema é encarado por servidores HTTP ocupados que recebem muitas conexões de clientes remotos (Stevens, 1996). Quando o ACK é recebido por B, o processo de *three-way handshake* se completa e a conexão é estabelecida. A partir disso, dados podem ser trocados em ambas direções.

O término de uma conexão TCP é chamado *half-close* porque quando um *host* fecha

uma conexão, está indicando que não enviará mais dados ao outro *host*, mas ainda aceita receber dados. Por exemplo, na figura 2.4 o *host* A fecha a conexão de A para B enviando um segmento FIN para B, cuja recepção é indicada por um ACK. Quando o *host* B deseja terminar a conexão de B para A, envia um segmento FIN para A, que acusa a recepção com um segmento ACK. Assim, 3 segmentos são necessários para estabelecer uma conexão TCP e 4 para terminá-la em ambas direções.

O TCP usa *acknowledgements*, *time-outs* e retransmissões para controle de erro. O controle de fluxo é implementado pelo TCP através de um mecanismo de *sliding window* (janela deslizante). O tamanho da janela é o número máximo de bytes que pode ser enviado antes que um ACK seja recebido.

Não se pode aumentar o tamanho da janela obtendo-se um *throughput* sempre crescente, porque a largura de banda da rede limita o fluxo máximo alcançado na conexão. Assim, $X_w = min(B, w \times X_1)$, onde X_w é o *throughput* da conexão TCP em bytes por segundo com um tamanho de janela w, e B é a largura de banda, em bytes por segundo, da rede. Seja w^* o tamanho de janela para o qual o ganho de throughput X_{w^*}/X_1 é máximo. Então, $w^* = \lceil B/X_1 \rceil$. Em (Menasce & Almeida, 1998) é mostrada uma tabela com o ganho de *throughput* máximo X_{w^*}/X_1, o valor de w^*, e o *throughput* máximo X_{w^*} em bytes por segundo para vários tipos de redes, sendo os valores de RTT e largura de banda de (Heidemann *et al.*, 1997); verifica-se assim que os benefícios de tamanhos de janela grandes são mais evidentes para redes de baixa latência e alta largura de banda. Como o TCP não sabe qual o RTT e a largura de banda efetivos para a rede, ele usa um mecanismo de partida lenta (*slow start*) para obter uma estimativa das características de desempenho da rede. O TCP inicia a conexão com uma janela pequena e ajusta o tamanho de acordo com a taxa com que os ACKs são recebidos pelo outro lado (Stevens, 1994). Isso traz um problema para conexões de curta duração que podem nunca atingir o *throughput* máximo sob um tamanho ótimo de janela (Heidemann *et al.*, 1997).

2.3 Sistemas Distribuídos

2.3.1 Introdução

Até a década de 80 os computadores eram considerados grandes e caros. A partir de meados dessa década, com microprocessadores mais poderosos e LANs de alta velocidade, surgiram sistemas computacionais compostos de múltiplas CPUs ligadas por uma rede de alta velocidade, chamados sistemas distribuídos. Seu software é diferente daquele de sistemas centralizados e não existe na literatura uma definição com concordância geral para o que seja um sistema distribuído.

Pode-se definir um sistema distribuído como um conjunto de computadores autônomos ligados por uma rede e equipado com software de sistema distribuído, o que capacita os computadores a coordenarem suas atividades e compartilhar os recursos do sistema: hardware, software

e dados (Coulouris *et al.*, 1994). A transparência é uma característica chave: os usuários de um sistema distribuído devem perceber um único recurso de computação integrada, muito embora ele possa ser implementado por diversos computadores em diferentes localizações. A figura 2.5 mostra como exemplo as componentes de um sistema distribuído simples, baseado em rede local.

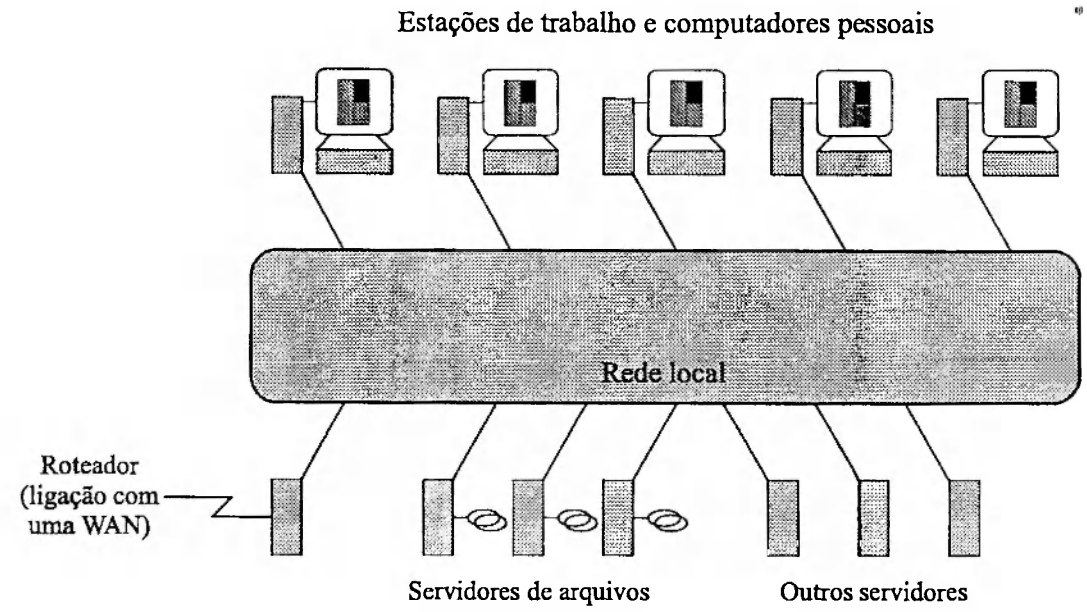


Figura 2.5: Componentes de um Sistema Distribuído simples baseado em rede local (Souza, 1996)

Cada usuário é provido de uma máquina poderosa o suficiente para executar os programas que queira; há servidores que fornecem acesso a uma variedade de recursos integrados ao sistema.

2.3.2 Exemplos de sistemas Distribuídos e Aplicações

Como exemplos de sistemas distribuídos existentes podem-se citar as várias implementações de UNIX distribuído, incluindo a que levou ao desenvolvimento do NFS, com suas componentes associadas, RPC e NIS. Embora o desenvolvimento de sistemas distribuídos UNIX ofereça vantagens, uma nova geração de sistemas operacionais distribuídos surgiu, cujos objetivos de projeto vão além de UNIX. Distintamente dos sistemas operacionais tradicionais, os sistemas operacionais distribuídos são modulares e extensíveis, com novas componentes podendo ser acrescidas em resposta às necessidades de novas aplicações, estando algumas componentes na fronteira entre o sistema operacional e software de aplicações.

Outros exemplos de sistemas distribuídos ocorrem em diversas aplicações comerciais, por exemplo, bancárias, as quais exigem os seguintes requisitos: alto nível de confiabilidade, privacidade e segurança, acesso concorrente à base de dados para muitos usuários, tempos de resposta garantidos, pontos de acesso a serviços amplamente distribuídos em termos geográficos, crescimento em escala e interoperabilidade. A maioria das implementações de tais aplicações

comerciais é baseada em hardware, software e redes de comunicação dedicadas. Aplicações distribuídas usando WANs também se tornaram viáveis, embora as LANs de alta velocidade constituam uma melhor base inicial, já que seu desempenho possibilita que uma faixa muito mais ampla de sistemas e aplicações distribuídas seja desenvolvida de maneira transparente, explorando a velocidade da rede para esconder o efeito de distribuição física do hardware e da informação. Finalmente, podem-se citar aplicações em NoWs (*Network of Workstations*), *Grids* e *Clusters* de computadores.

2.3.3 Características Principais de Sistemas Distribuídos

As principais características requeridas para um sistema distribuído são: compartilhamento de recursos, abertura, concorrência, crescimento em escala, tolerância a falhas, transparência e desempenho.

Uma abordagem muito comum para sistemas distribuídos formados por um conjunto de estações de trabalho numa LAN é o fornecimento de um sistema de arquivos global compartilhado, passível de ser acessado a partir de todas as estações. O sistema de arquivos é implementado por uma ou mais máquinas servidoras de arquivos. Os servidores de arquivos aceitam requisições de programas de usuários executando em outras máquinas, para ler e escrever em arquivos. O sistema operacional usado nesse tipo de ambiente deve gerenciar as estações individuais, servidores de arquivos e cuidar da comunicação. Não se requer que todas as máquinas executem o mesmo sistema operacional. Numa situação como essa, em que cada máquina tem elevado grau de autonomia e há poucos requisitos para o sistema global, tem-se o chamado "sistema operacional de rede". O exemplo mais comum desse caso é aquele representado pelo uso do NFS. Nesse caso, além do sistema de arquivos compartilhado, é bastante aparente aos usuários que o sistema consiste de vários computadores, entre os quais não há praticamente coordenação, exceto a regra de que o tráfego C/S (cliente-servidor) deve obedecer aos protocolos do NFS.

Já em um sistema distribuído verdadeiro, os usuários devem ter a ilusão de que a rede de computadores é um único sistema *time-sharing*, ou seja, o conjunto de máquinas age como um uniprocessador virtual (*single system image*). Aqui será normal que *kernels* idênticos executem em todas CPUs do sistema, para facilitar a coordenação global de atividades. Por exemplo, quando um processo tiver que ser iniciado, todos *kernels* devem cooperar para achar o melhor lugar onde executar o processo. Em acréscimo, um sistema global de arquivos também é necessário, porém com semântica melhor definida do que no caso do NFS. Cada *kernel* pode, contudo, ter controle considerável sobre seus próprios recursos locais. Por exemplo, como não há memória compartilhada, é lógico permitir que cada *kernel* gerencie sua própria memória.

2.3.4 Requisitos e Modelos de Sistemas Distribuídos

Num sistema distribuído o termo gerente de recursos é às vezes usado para denotar um módulo de software que gerencia um conjunto de recursos de um tipo particular, oferecendo uma interface de comunicação fazendo com que o recurso seja acessado, manipulado e atualizado com confiabilidade e consistência. Cada tipo de recurso requer algumas políticas e métodos de gerência em separado, mas há requisitos comuns.

Esses requisitos incluem: o fornecimento de um esquema de atribuição de nomes para cada classe de recurso, capacitando recursos individuais a serem acessados a partir de qualquer localização; o mapeamento de nomes de recursos a endereços de comunicação; e a coordenação de acessos concorrentes que mudam o estado de recursos compartilhados, de modo a assegurar sua consistência.

Citam-se agora três modelos usados em sistemas distribuídos: o cliente/servidor, mais conhecido e amplamente usado, um modelo alternativo baseado em objetos e o modelo *peer to peer*.

O Modelo Cliente-Servidor

O modelo cliente/servidor é baseado na divisão do trabalho entre dois processos: o cliente e o servidor (Coulouris *et al.*, 1994). A idéia é estruturar o sistema operacional como um grupo de processos cooperativos chamados servidores, que oferecem serviços a usuários, os processos clientes. As máquinas cliente e servidor executam todas o mesmo micro kernel. Cada processo servidor age como gerente de recursos para um conjunto de recursos de determinado tipo. Alguns tipos de recursos devem permanecer localmente em cada computador: a RAM, os processadores centrais e a interface de rede local são usualmente considerados como o conjunto mínimo de tais recursos. Esses recursos chave são gerenciados em separado por um sistema operacional em cada computador; eles devem ser compartilhados apenas entre processos localizados no mesmo computador.

O servidor é uma entidade passiva que nunca inicia a troca de mensagens; ele executa usualmente numa máquina mais poderosa do que aquela em que o cliente executa. A máquina em que o servidor executa usa um sistema operacional multiprogramado, tal como UNIX ou Windows NT. O servidor executa um conjunto de serviços relacionados funcionalmente que requerem em geral uma componente especializada de hardware e software (Menasce & Almeida, 1998). O protocolo implementado entre clientes e servidores é um protocolo *request-reply*, conforme a figura 2.6.

Os clientes e servidores podem executar sobre o TCP ou um protocolo sem conexão como UDP. Como um servidor pode receber requisições de muitos clientes, uma fila de requisições se forma no servidor. Se apenas uma requisição é atendida de cada vez, os recursos na máquina

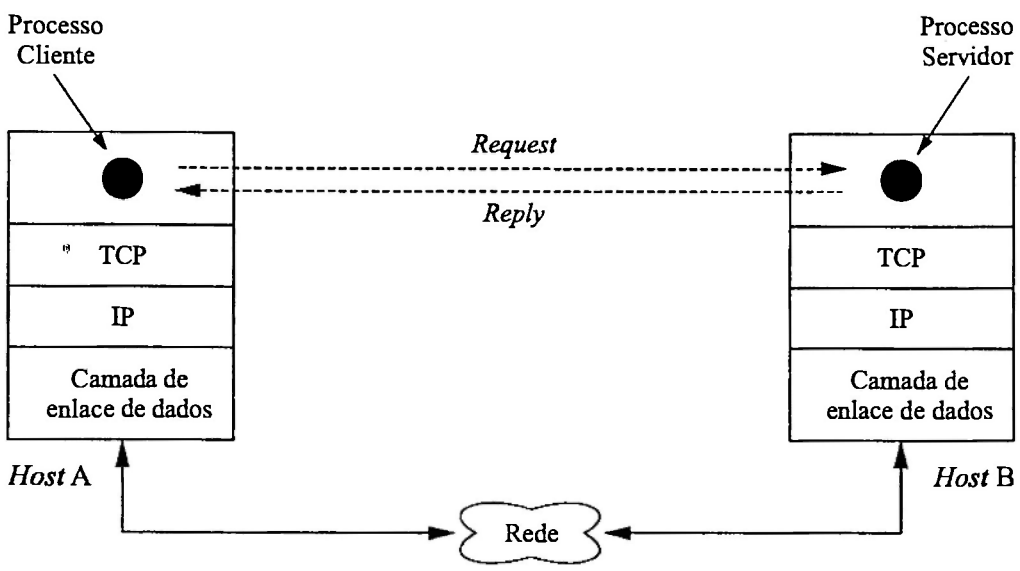


Figura 2.6: Protocolo cliente-servidor: *request-reply*

servidora podem ser subutilizados, o *throughput* do servidor (número de requisições atendidas por unidade de tempo) pode ser baixo, e o tempo de resposta aos clientes crescerá à medida que a carga no servidor aumenta. A maioria dos servidores cria múltiplos processos ou múltiplas *threads* de execução num único processo para manipular a fila de requisições chegando. O modelo C/S não satisfaz todos requisitos de sistemas distribuídos; apesar disso o modelo tornou-se uma base efetiva adequada para a maioria das aplicações correntes.

Modelo Baseado em Objetos

O modelo baseado em objetos não é distinto do tradicional modelo de programação orientada a objetos, em que cada entidade em um programa executando é vista como um objeto, com uma interface de manipulação de mensagens fornecendo acesso a suas operações. No modelo baseado em objetos para sistemas distribuídos, cada recurso compartilhado é visto como um objeto. Os objetos são identificados de forma única e podem ser movidos para qualquer lugar na rede sem mudar suas identidades.

O modelo de objetos capacita todos recursos compartilhados a serem vistos de modo uniforme por usuários de recursos. Como no modelo C/S, os objetos podem agir conjuntamente como usuários de recursos e gerentes de recursos. No modelo C/S, o esquema de nomes usados para recursos varia dependendo do serviço que os gerencia. Mas no modelo orientado a objetos, os usuários de recursos podem se referir a todos recursos de modo uniforme.

Modelo *peer-to-peer* (P2P)

Quando se usa o modelo *peer-to-peer*, a computação não é feita apenas por um servidor central, mas cada membro (ou *peer*) pode desempenhar o papel de cliente ou servidor para outro

peer, diretamente. O sucesso das aplicações baseadas nesse modelo se deve a vários fatores, principalmente o aumento na capacidade de processamento dos PCs, em particular quanto à troca de arquivos na Internet e as próprias deficiências no modelo C/S, quanto a crescimento em escala e existência de pontos centrais de falha, entre outras.

As primeiras aplicações surgiram no final da década de 90 (ICQ e Napster) usando o modelo C/S para alguns serviços como pesquisa, localização e *buffering* (armazenamento temporário) de mensagens. O modelo P2P era usado para outros serviços, por exemplo, transferência de arquivos (da Fonseca, 2002). Na seqüência surgiram as primeiras redes P2P totalmente descentralizadas, vindo a ser posteriormente estruturadas, após o surgimento de problemas de desempenho.

2.4 WWW

2.4.1 Introdução

A Web, basicamente um sistema cliente-servidor, é uma estrutura arquitetural para acessar documentos interligados espalhados em milhares de máquinas de toda a Internet. Do ponto de vista dos usuários, a Web consiste de uma vasta coleção de documentos, chamados páginas, cada qual contendo *links* (apontadores) para outras páginas relacionadas. As páginas podem ser vistas usando o programa chamado *browser*, que solicita a página requisitada a um servidor, interpreta os comandos de texto e formatação que a página contém e visualiza a página na tela. Os *hyperlinks*, *strings* de texto que são *links* para outras páginas, são destacados de alguma maneira especial.

2.4.2 Evolução da Web

A Figura 2.7 mostra 3 fases na evolução da Web (Orfali *et al.*, 1999).

No eixo vertical está representada a funcionalidade e no horizontal a interação C/S. Conforme a figura 2.7, pode-se dividir a evolução nas seguintes 3 fases: a fase de Hipertexto, a fase interativa e a fase de objetos distribuídos. A divisão é feita considerando as aplicações que se pode executar sobre a estrutura oferecida na Web, ou seja, que tipo de facilidades ou serviços os usuários podem obter. Em cada fase se acrescenta mais funcionalidade e interação com o usuário.

Fase hipertexto (requisições estáticas)

Na fase hipertexto a Web é apenas um meio unidirecional para publicar e difundir documentos eletrônicos estáticos, funcionando como um servidor de arquivos gigante baseado no

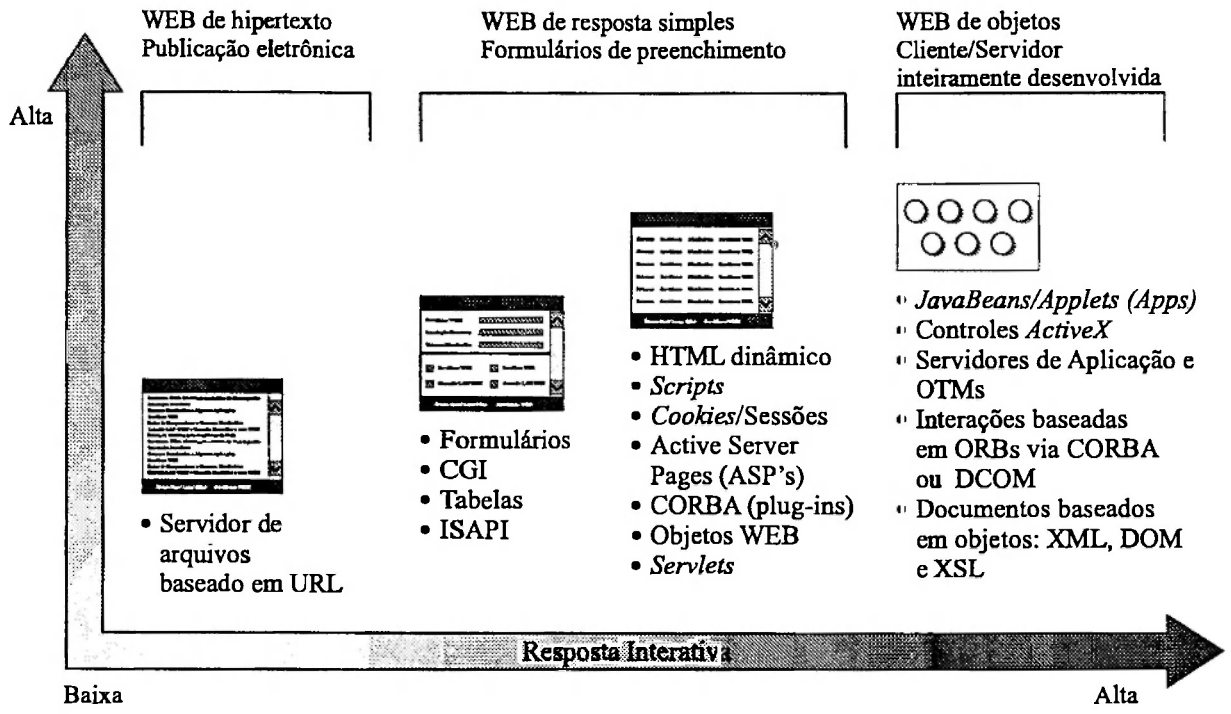


Figura 2.7: Evolução da Web em 3 fases

protocolo URL. Nessa fase o único modo com que se pode interagir com servidores Web é clicando em *hiperlinks* para se navegar entre documentos; as interações C/S são em duas camadas (*tiers*), clientes e servidores, conforme a figura 2.8 .

O *browser* ao receber a resposta do servidor, examina o conteúdo do arquivo, interpreta os comandos HTML (*Hypertext Message Language*) e mostra o documento formatado ao usuário solicitante. O arquivo enviado pode ser de um tipo especial não suportado pelo *browser* e nesse caso é chamado um *plug-in* para correta visualização.

Fase interativa (requisições dinâmicas)

A partir de 1995 a Web evoluiu para um meio mais interativo com a introdução do modelo C/S utilizando agora 3 níveis (ou camadas), usando o protocolo CGI (*Common Gateway Interface*); este permite a servidores Web rotearem o conteúdo de formulários HTML para aplicações de servidores *back-end*. A figura 2.9 mostra a 2a. fase (interativa) da Web (Orfali *et al.*, 1999).

Um formulário Web (*form*) é uma página HTML com um ou mais campos de entrada de dados e um botão de *Submit*. Ao clicá-lo, o conteúdo de dados do formulário é enviado ao servidor Web. Este, por sua vez, invoca o programa ou recurso nomeado na URL para que se encarregue da requisição. O servidor passa a requisição do método e seus parâmetros ao servidor *back-end*, obedecendo ao protocolo CGI. O servidor *back-end* executa a requisição e devolve os resultados em formato HTML para o servidor Web. O servidor Web trata o resultado como um documento normal que ele devolve ao cliente. Aqui o servidor Web age como um conduto entre

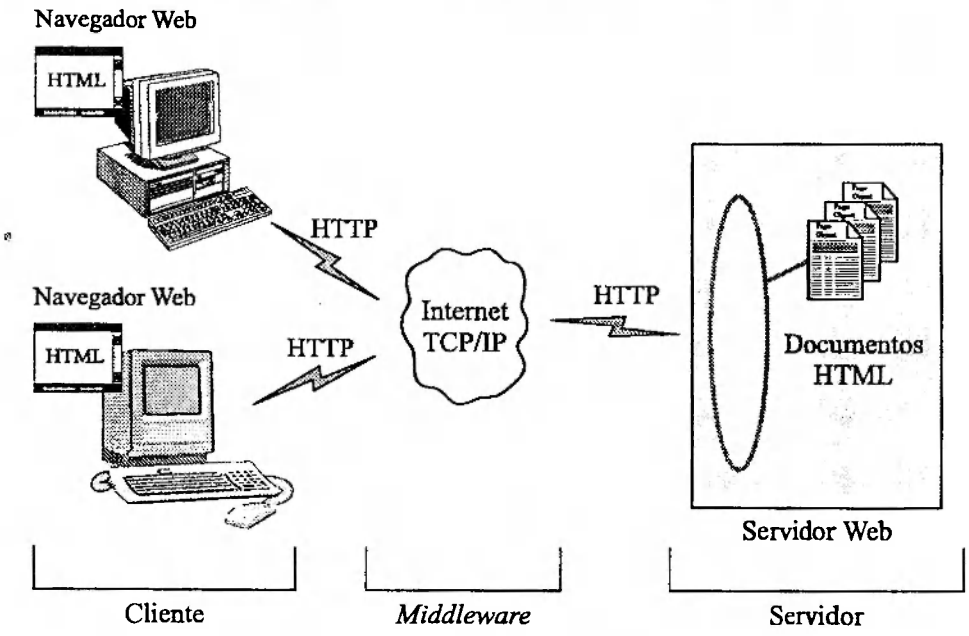


Figura 2.8: Modelo cliente-Servidor na fase hipertexto

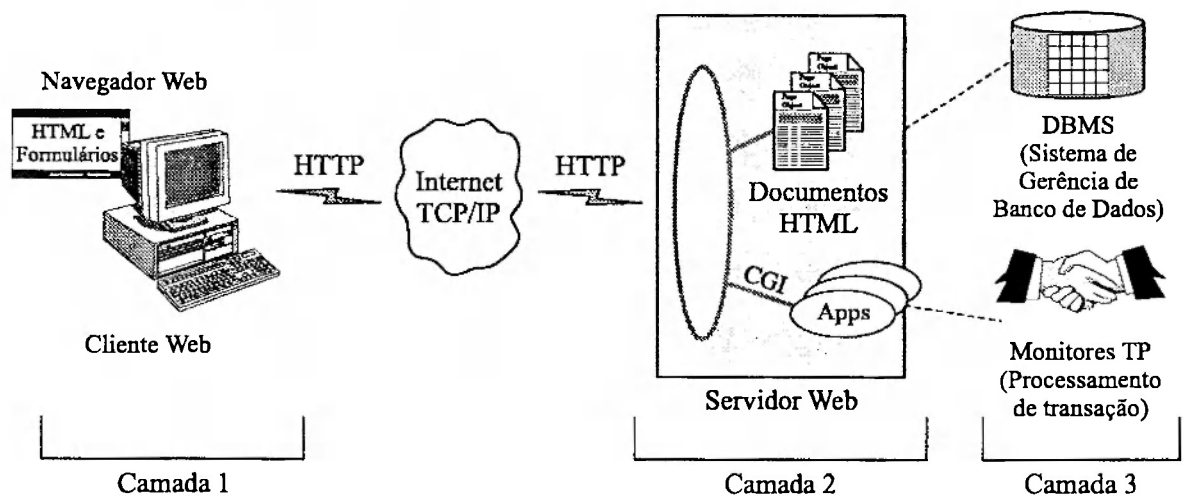


Figura 2.9: Modelo Cliente-Servidor na fase interativa

o cliente e o programa *back-end* que faz o trabalho real.

Com esse uso do CGI se tornou possível para clientes fazerem atualizações em bases de dados, com servidores *back-end*, sendo essas atualizações e inserções parte essencial do comércio eletrônico. A maioria dos provedores de serviços *on-line* requer para isso, contudo, alguma forma de mecanismo de segurança; é aqui que protocolos como SSL (*Secure Sockets Layer*), S-HTTP (*Secure-HTTP*), IP-Sec e *firewalls* Internet se tornam atuantes. O protocolo CGI não mantém informações de um formulário para o formulário seguinte (é *stateless*). Para se manter o estado de uma transação ao longo de invocações seguidas de formulários, existem soluções como, por exemplo, *cookies*. *Cookies* se constituem numa pequena parte de dados que é armazenada no cliente, a cargo do servidor.

A dupla de protocolos HTTP/CGI não é um *middleware* totalmente adequado. *Mid-*

dleware se refere a um conjunto de serviços comuns que permitem aplicações e usuários trocarem informações através de redes. Esses serviços se situam no meio, ou seja, sobre o sistema operacional e o software de rede e abaixo das aplicações distribuídas. O CGI como plataforma de aplicações tem duas limitações: não mantém estado entre conexões e o programa alvo deve ser carregado na memória para cada invocação. Nunca foi pensado como uma plataforma para executar componentes escaláveis do lado do servidor.

Pelos motivos citados, entre outros, foram surgindo com o tempo algumas soluções que procuravam minimizar as desvantagens do CGI em relação à falta de estado do servidor, compartilhar processos em memória entre as invocações de serviços e introduzir mais interatividade no cliente. Exemplos dessas soluções são: o ASP (Active Server Pages) da Microsoft, NSAPI (Netscape Server API), Java Servlets e HTML dinâmico. A página do servidor tipicamente executa um *script* do lado do servidor que sabe como instanciar e invocar objetos. Em adição, o lado cliente tem agora suficiente inteligência para interagir com o usuário sem ir ao servidor toda vez, usando *applets* ou *scripts*. Por exemplo, com HTML dinâmico, cada elemento numa página HTML é um objeto passível de escrita.

Essas últimas técnicas citadas constituem uma segunda parte da fase interativa, em que as soluções, embora alcançando algum sucesso nos propósitos, não são universais, mas proprietárias.

Fase de objetos distribuídos

A 3a. fase na evolução da Web vem com as tecnologias de objetos (componentes) distribuídos. O problema principal com as abordagens citadas no final da subseção anterior é que elas requerem o HTTP e o servidor Web para intermediar objetos executando no cliente e no servidor. Não há maneira de um objeto cliente invocar diretamente um objeto servidor. O formulário HTTP que se submete é ainda a unidade básica na interação C/S, o que não é adequado para aplicações C/S requerendo conversações altamente interativas entre componentes. Também quanto a crescimento em escala, fica-se muito a desejar.

A partir de 1996 surgiram na Web objetos na forma de *applets* Java e *browsers* capacitados para Java. Em 97 fornecedores de software apresentaram produtos nos lados cliente e servidor usando ORBs (*Object Request Brokers*: agentes intermediários para requisição de objetos) (CORBA/Java e CORBA/C++). Ao final de 98 havia mais de uma dúzia de servidores de aplicações Web que suportavam CORBA/IIOP (IIOP: Internet InterORBs Protocol, protocolo que possibilita a intercomunicação entre ORBs pela Internet) e Java do lado do servidor. Em 1998 o W3C recomendou dois padrões importantes para a Web baseados em objetos: o XML 1.0 e o DOM (*Level 1*). Em acréscimo, o HTML versão 4.0 forneceu um modo consistente para se embutir componentes dentro de páginas Web. Com isso a teia de objetos (*Object Web*) passou a ser efetivamente formada. A figura 2.10 mostra a 3a. fase da Web, de objetos distribuídos.

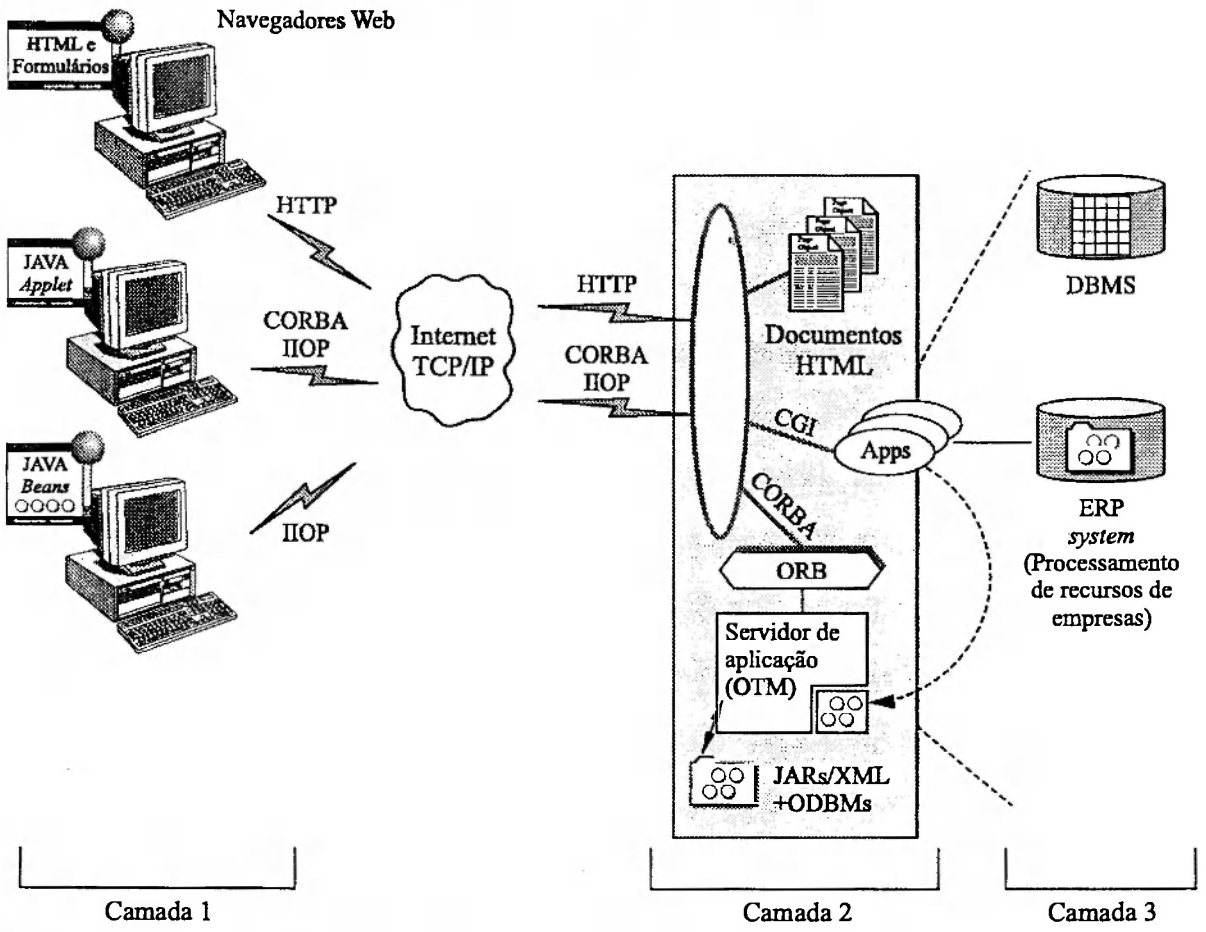


Figura 2.10: O Modelo Cliente-Servidor na fase de Objetos Distribuídos: Web de Objetos com 3 camadas CORBA/Java

Java foi o primeiro passo para se criar uma Web com objetos, sendo acompanhada de uma infraestrutura de objetos distribuídos dada pela arquitetura CORBA, criada pela OMG (Object Management Group). CORBA lida com transparência em relação à rede enquanto que Java com transparência em relação à implementação. A maioria dos servidores de aplicações Web está seguindo o padrão CORBA/EJB (*Enterprise Java Beans*) para modelo de componentes do lado do servidor, enquanto produtos Microsoft seguem o padrão COM+.

O padrão para documentos na Web, XML, (*Extensible Markup Language*) recomendado pelo W3C em 1998 tem suas raízes em SGML, sendo considerado como uma linguagem para criar estruturas de documentos. Ao final de 98 o W3C recomendou também o padrão DOM (*Document Object Model*) que provê um modelo de objetos conjuntamente para HTML e XML. Esse modelo de objetos define, em IDL CORBA, interfaces que permitem controle de cada *tag* e estrutura dentro de um documento XML ou HTML. Possibilita geração em tempo de execução de documentos Web dinâmicos. Os programas clientes e servidores podem usar qualquer linguagem suportada por CORBA para acessar as interfaces DOM.

Ocorreram assim várias melhorias e vantagens na integração de CORBA com a Web.

2.4.3 Protocolos Web

Protocolos de Suporte

As camadas abaixo da camada de aplicação, numa arquitetura de redes em camadas, fornecem transporte confiável, mas não fazem qualquer trabalho real para os usuários. Mesmo na camada de aplicação há necessidade de mecanismos e/ou protocolos de suporte para permitir que as aplicações reais funcionem. Pode-se considerar a existência de três desses protocolos de suporte (Tanenbaum, 1996).

Primeiramente, em relação à segurança, não há um único protocolo, mas um grande número de conceitos e protocolos a serem usados para assegurar privacidade onde necessário.

O segundo protocolo de suporte é o DNS (Domain Name System). Consiste basicamente em um esquema de nomeação hierárquica baseada em domínios e um sistema de base de dados distribuído para implementar esse esquema de nomeação. Ele é usado primariamente para mapear nomes de *hosts* e destinos de *e-mail* em endereços IP.

O terceiro protocolo suporte é para gerencia de redes. Quando a Internet surgiu derivada da ARPANET, com múltiplas operadoras e *backbones*, tornaram-se necessárias melhores ferramentas para gerência de redes. Em maio de 1990, uma RFC (*Request For Comments*) foi publicada, definindo a versão 1 do SNMP (*Simple Network Management Protocol*). O modelo SNMP de uma rede gerenciada consiste de 4 componentes: nós gerenciados, estações de gerência, informação de gerência e um protocolo de gerência. Os nós gerenciados podem ser hosts, roteadores, pontes, impressoras ou quaisquer outros mecanismos capazes de se comunicar com o mundo externo. Para ser gerenciado diretamente pelo SNMP, um nó deve ser capaz de executar um processo de gerencia SNMP chamado agente SNMP. Cada dispositivo mantém uma ou mais variáveis descrevendo seu estado. Na literatura SNMP essas variáveis são chamadas objetos, embora não sejam objetos no sentido de um sistema orientado a objeto, porque elas têm apenas estado e nenhum outro método que não seja leitura e escrita de seus valores. A coleção de todos objetos possíveis em uma rede é dada numa estrutura de dados chamada MIB (Management Information Base).

A Web usa a Internet como seu *backbone* e a Internet já tem da ordem de centenas de milhares de redes interconectadas, que se estendem tanto dentro de corporações como também em residências, com os protocolos de aplicações Web executando sobre o TCP-IP; hoje, a maioria das pessoas que acessam a Internet usa a Web como seu *front-end*. A Web engloba a maioria dos protocolos de aplicação da Internet - incluindo SMTP (*Simple Mail Transfer Protocol*), Telnet, FTP e NNTP (*Network News Transfer Protocol*). O modelo Web de C/S é também a base para *intranets*, redes privadas que usam os protocolos TCP-IP, podendo ou não ter conexões para a Internet.

URL

O protocolo URL provê um esquema de nomes para identificar todos recursos Web, incluindo documentos, imagens, áudio e programas (Tanenbaum, 1996). Os URLs descrevem completamente onde um recurso se encontra e como consegui-lo. Os URLs suportam os protocolos Web mais novos, como o HTTP, assim como protocolos Internet mais antigos, como FTP, Gopher e outros.

HTTP

O protocolo padrão para transferências na Web é (similar a uma RPC) o HTTP (*Hypertext Transfer Protocol*), usado para acessar os recursos que se encontram no espaço definido por URLs. O protocolo HTTP é uma RPC *stateless* que basicamente: (i) estabelece uma conexão C/S; (ii) transmite e recebe parâmetros incluindo um arquivo retornado; (iii) encerra a conexão C/S. Os clientes e servidores HTTP usam a representação de dados MIME (*Multipurpose Internet Mail Extensions*) usada na Internet para descrever (e negociar) o conteúdo de mensagens. Os servidores HTTP (servidores Web) produzem conteúdo (as páginas HTML) independente da plataforma do cliente. Os *browsers* se encarregam dos detalhes específicos a cada plataforma.

Embora o uso do TCP para conexão de transporte seja muito comum, não é formalmente requerido. Com as redes ATM se tornando confiáveis o suficiente, as requisições e respostas HTTP podem ser também transportadas em mensagens AAL5 (*ATM Adaptation Layer 5*). Uma requisição HTTP inclui várias partes: o método que especifica a ação legal (por exemplo, *GET*, *HEAD*, *PUT* e *POST*), o URL que identifica o nome da informação requisitada, e outras informações adicionais, como por exemplo, o tipo de documento que o cliente está querendo aceitar, autenticação e autorização. O servidor, após receber a requisição, faz o *parsing* (filtragem) da requisição e executa a ação especificada pelo método. A resposta consiste de uma *status line* indicando sucesso ou falha da requisição, meta-informação descrevendo o tipo de objeto retornado e a informação requisitada, e um arquivo ou *output* gerado por uma aplicação do lado do servidor (por exemplo, CGI). Os principais passos envolvidos em uma interação HTTP são: mapear o nome do servidor a um endereço IP; estabelecer uma conexão TCP/IP com o servidor; transmitir a requisição (URL, método, outras informações); receber a resposta (texto HTML, imagem, outra informação); fechar a conexão TCP/IP. Cada uma dessas etapas tem um custo inerente que depende do desempenho do servidor e da rede.

O protocolo HTTP é chamado de *stateless* porque não inclui o conceito de sessão ou interação além da entrega do documento solicitado. No protocolo HTTP original, uma conversação é restrita à transferência de um documento ou imagem. Cada transferência é totalmente separada da anterior ou seguinte. A natureza sem estado do protocolo pode trazer vantagem, pois o servidor não tem que manter registro de quais são os clientes ou que requisições foram providas no passado. Porém a versão original do HTTP tem várias ineficiências. Por exemplo, uma nova

conexão é estabelecida por requisição. Uma página com texto e muitas imagens pequenas geram muitas conexões em separado para o texto e para cada imagem. Já que a maioria dos objetos Web é pequena, uma fração alta de pacotes trocados entre cliente e servidor é simplesmente de pacotes TCP de controle usados para abrir e fechar conexões, como mostrado na figura 2.11, que ilustra os pacotes trocados entre um cliente e um servidor para uma interação HTTP sobre o TCP (Comer, 1997), (Padmanabhan & Mogul, 1995).

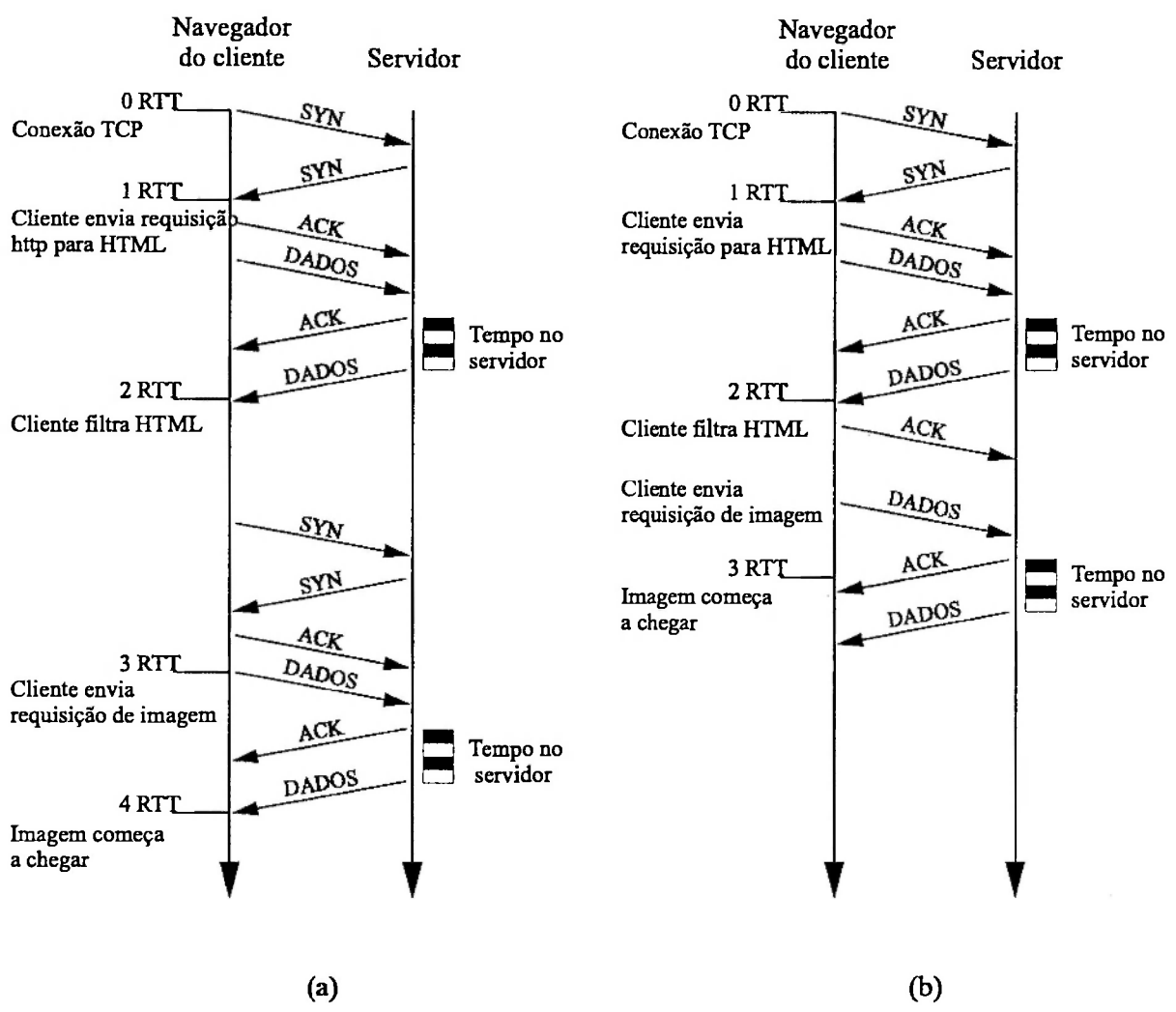


Figura 2.11: (a)Interação HTTP no HTTP 1.0; (b)Interação HTTP no HTTP 1.1

Um elemento chave para se compreender o desempenho do par HTTP-TCP/IP é a latência ou retardos obrigatórios impostos pelos protocolos. A latência nesse caso consiste de retardo na conexão e retardo na requisição. O primeiro é o tempo (medido em RTT) para se estabelecer uma conexão. O último é o tempo para se completar uma transferência de dados sobre uma conexão já estabelecida. A figura 2.11(a) mostra a troca de pacotes que ocorre na versão 1.0 e a figura 2.11(b) a troca equivalente para a versão do HTTP 1.1 com conexão persistente, que mantém as conexões TCP durante várias requisições HTTP. As linhas horizontais pontilhadas no lado cliente representam os tempos obrigatórios de RTT impostos pela combinação dos protocolos TCP/IP e HTTP. Os retardos obrigatórios envolvidos em uma interação HTTP de

requisição-resposta são:

- O cliente abre uma conexão TCP que resulta numa troca de pacotes SYN e ACK.
- O cliente envia uma requisição HTTP ao servidor, que faz o *parsing*, executa a ação requisitada, e envia de volta a resposta. No caso de uma operação de recuperação, o servidor tem que mover dados do disco ou cache para o cliente, através da rede. Logo em seguida, o servidor é responsável por fechar a conexão. Os pacotes de FIN trocados por cliente e servidor não estão representados na figura 2.11 porque o cliente não tem que esperar o término da conexão para continuar.
- O cliente faz o *parsing* da resposta HTML para procurar pela URL da imagem *inline*. O cliente então abre uma nova conexão TCP, o que envolve um outro *three-way handshake*.
- O cliente envia uma requisição HTTP para a primeira imagem *inline* e o processo se repete.

Para cada requisição adicional, uma nova conexão TCP deve ser estabelecida e o sistema tem de novo uma sobrecarga para estabelecimento de conexão. Como uma consequência da combinação HTTP e TCP/IP, um cliente tem que esperar no mínimo quatro RTTs de rede antes que um documento com imagem embutida seja visualizado pelo *browser*. Cada imagem adicional requer no mínimo dois outros RTTs; uma para estabelecer uma conexão TCP e uma outra para obter a imagem. Além disso, quando uma conexão TCP é aberta pela primeira vez, o TCP emprega um algoritmo conhecido como *slow start*. Ele usa vários primeiros pacotes de dados para testar a rede, visando determinar a taxa de transmissão ótima. Novamente pelo fato dos objetos Web serem pequenos, a maioria dos objetos são transferidos antes que sua conexão TCP complete o algoritmo de *slow start*. Em resumo, a maioria das operações HTTP 1.0 usa o TCP ineficientemente, resultando em problemas de desempenho devido a congestionamento e sobrecarga.

HTTP 1.1

A versão HTTP 1.1 ou HTTP-NG inclui características que resolvem parte dos problemas de desempenho da versão 1.0. O HTTP 1.1 deixa a conexão TCP aberta entre operações consecutivas (figura 2.11(b)). Com isso uma conexão TCP é usada para fazer múltiplas requisições HTTP, eliminando o custo de vários *open* e *close* e minimizando o impacto do *slow start*. Desse modo múltiplas requisições e respostas podem estar contidas em um único segmento TCP. Essa característica também evita muitos retardos de ida e volta, melhorando o desempenho e reduzindo o número de pacotes trocados. A figura 2.11 mostra como o novo protocolo HTTP afeta as latências de rede. No caso de uma requisição para um documento HTML e uma imagem *inline*, o documento está disponível para o *browser* após 3 RTTs, ao invés de 4 necessários na versão original do HTTP. Uma outra característica do HTTP relevante para desempenho é conhecida como

pipeline de requisições. O *pipeline* no HTTP permite que múltiplas requisições sejam enviadas pelo cliente durante uma única conexão TCP, antes que respostas de requisições prévias sejam recebidas. Resultados experimentais de (Nielsen *et al.*, 1997) indicam que uma implementação do HTTP 1.1 com *pipeline* ultrapassou em desempenho o HTTP 1.0 mesmo quando a versão original do protocolo usou múltiplas conexões em paralelo sobre várias conexões TCP.

2.4.4 Servidores Web e outros Servidores de Rede

Tipos de Servidores

Há vários tipos de servidores em ambientes computacionais baseados em rede, como por exemplo, servidores de arquivos, servidores de bases de dados, servidores de aplicação, servidores de objetos, servidores Web e servidores de software (Orfali *et al.*, 1996).

Servidores de arquivos: provêm aos computadores de rede acesso a um sistema de arquivos compartilhado. Os clientes fazem requisições de busca em diretórios, recuperam atributos de arquivos e lêem (de) ou escrevem (em) blocos de arquivos. Um dos servidores de arquivos mais populares é o NFS que permite um fácil compartilhamento de dados em sistemas distribuídos heterogêneos. O NFS pode funcionar com os protocolos UDP e TCP, mas a maioria usa UDP.

Servidores de bases de dados: fornecem acesso a uma ou mais base de dados compartilhados. As requisições de clientes estão na forma de assertivas ou chamadas (*statements*) SQL (*Structured Query Language*), uma linguagem padrão para acessar bases de dados relacionais (O'Neil, 1994). O servidor de base de dados recebe *statements* SQL e os submete a uma máquina de base de dados que, ou faz consultas à base de dados ou atualiza a base de dados, dependendo do tipo de requisição. Mesmo que a máquina de base de dados tiver que ler centenas de registros para satisfazer uma consulta do cliente para recuperar apenas alguns registros, somente o resultado é enviado de volta ao cliente. Isso reduz significativamente o tráfego de rede e o número de interações C/S.

Servidores de aplicação: fornecem acesso a procedimentos remotos invocados pelos clientes através de mecanismos de RPC. Esses procedimentos remotos implementam a lógica da aplicação e emitem chamadas SQL a uma máquina de base de dados *back-end*. Assim, ao invés de uma interação cliente-servidor por chamada SQL, essa abordagem requer uma única interação por invocação de procedimento.

Servidores de objetos: dão suporte à invocação remota de métodos no desenvolvimento de aplicações orientadas a objetos distribuídos. Os ORBs agem como a "cola"entre clientes e objetos remotos.

Servidores Web: provêm acesso a documentos, imagens, som, executáveis, e aplicações passíveis de *download* (por exemplo, *applets* Java) através do protocolo HTTP. Estão sendo hoje em alguns casos usados como servidores de aplicação.

Servidores de software: são usados para prover executáveis a computadores conectados à rede que não têm discos rígidos, obtendo os executáveis de pacotes de softwares comerciais a partir de um servidor de software; este sempre tem a versão mais atual do software.

Aspectos Comparativos de Servidores Web

Um servidor de arquivos efetua leitura e escrita em arquivos. Sob esse aspecto, um servidor Web é mais simples pois efetua basicamente operações de leitura apenas. Assim, visto por esse aspecto, servidores Web poderiam ser considerados como um caso particular de servidores de arquivo, com operações apenas de leitura. Observe-se que boa parte do estudo de servidores de arquivos é feita considerando-se operações em LANs (NFS o mais usado, opera usualmente em LANs); já os serviços Web pressupõem, em geral, o uso de WANs.

Por outro lado, servidores Web exibem características peculiares, merecendo estudo em separado. Por exemplo, a carga de trabalho para um servidor de arquivos será uma mistura de dados de entrada de operações (leitura e escrita) em servidores de arquivos. Já para um servidor Web, essa carga será relativa às requisições feitas, (páginas estáticas de vários tipos e diversas requisições dinâmicas a considerar). Pode-se ainda citar que um servidor Web possui um grande grupo de aplicações *back-end* executando, ao contrário de servidores de arquivos. Os documentos armazenados em um servidor Web apresentam maior variabilidade em termos de tamanho e tipos de objetos; a distribuição dos tamanhos de arquivos na Web, incluindo tanto arquivos armazenados nos servidores ou arquivos requisitados pelos clientes, apresenta cauda pesada. Isso, em termos práticos significa que valores muito grandes de tamanhos de arquivos são possíveis com probabilidade não desprezível. Com isso, valores médios terão grande variabilidade, reduzindo o significado estatístico de medições efetuadas, para as quais, o efeito de uma grande variância tem que ser levado em consideração.

2.4.5 Considerações Finais

A Web é responsável pela maioria do tráfego na Internet e engloba um grande número de aplicações de diversos tipos.

O protocolo HTTP executa a partir dos protocolos TCP-IP, que não foram concebidos originalmente para lhe dar suporte.

Os servidores Web são componentes centrais nos serviços Web; seu desempenho é fundamental já que passaram a fazer parte do núcleo das redes de comunicação contemporâneas, podendo vir a se constituir no gargalo do atendimento de requisições. A avaliação de desempenho desses servidores passou a ser alvo de intensas pesquisas. Conclusões sobre o desempenho desses servidores podem ser úteis também para outros tipos de servidores sendo utilizados.

O capítulo seguinte focará no início a modelagem e avaliação de desempenho de sistemas

computacionais em geral, abordando após o caso específico de servidores Web, em particular quanto a aspectos de interesse da presente tese.

CAPÍTULO

3

Modelagem e Avaliação de Desempenho de Servidores Web

3.1 Introdução

A avaliação de desempenho é uma arte pois é algo que não pode ser feito mecanicamente (Jain, 1991). A avaliação de um sistema envolve conhecimento íntimo do sistema modelado e uma seleção cuidadosa de metodologia, carga de trabalho e ferramentas. O problema deve ser definido e convertido para uma forma em que as ferramentas e técnicas existentes possam ser usadas, considerando uma série de restrições (entre elas o tempo disponível). Cada analista tem seu estilo: dado o mesmo problema, podem-se escolher métricas e métodos de avaliação diferentes. E, com os mesmos dados obtidos, pode-se chegar a conclusões diferentes, e até opostas. As principais técnicas para avaliar desempenho são técnicas de aferição e modelagem. As técnicas de aferição podem utilizar *benchmarks*, monitoramento ou coleta de dados diretamente no sistema. As técnicas de modelagem podem usar (entre outras) Redes de Petri, Redes de Filas e *Statecharts*. Os modelos podem ser resolvidos por simulação ou analiticamente (Frances, 1998). Muitas vezes é usada uma combinação dessas técnicas.

3.2 Modelagem e Avaliação de Desempenho de Sistemas Computacionais

3.2.1 Carga de Trabalho e Métricas

Carga de Trabalho

A carga de trabalho de um sistema se refere às solicitações feitas pelos usuários, entidades que fazem pedidos de serviços no sistema. Exemplos de carga são instruções enviadas a uma CPU, consultas feitas a uma base de dados ou transações num sistema bancário. Uma carga real é aquela observada em operações normais, não pode ser repetida de modo controlado e, em geral, não é adequada para uso como teste. Assim, frequentemente é desenvolvida uma carga sintética (cujas características devem ser similares àquelas da carga real) que pode ser aplicada repetidamente e de forma controlada para estudos. Essa carga sintética é chamada de *benchmark* e se constitui de programas ou partes de programas relativos às aplicações mais utilizadas.

Métricas

Métricas de desempenho são medidas ou critérios para avaliação de desempenho (Jain, 1991).

Do ponto de vista de um usuário, a principal métrica é o tempo de resposta, que é o intervalo entre o fim de um pedido e o fim da resposta correspondente, sendo essa a definição usual, embora se possa considerar também o início da resposta.

Do ponto de vista do sistema a principal métrica é o *throughput*: o número médio de pedidos processados por unidade de tempo.

Para CPUs, o desempenho é medido comumente em MIPS (Milhões de Instruções por Segundo) ou MFLOPS (Milhões de Operações de Ponto Flutuante por Segundo) (observe-se que não faz sentido comparar MIPS de diferentes arquiteturas de CPUs, por exemplo, RISCs e CISCs, já que os tipos de instruções são distintos).

Para redes o *throughput* é medido em pps (pacotes por seg) ou bps (bits por seg). Observe-se que o tempo de resposta obtido com o *throughput* máximo pode ser alto demais para ser aceitável. Em muitas aplicações o joelho da curva do *throughput* versus tempo de resposta é considerado o ponto ótimo de operação. É o ponto além do qual o tempo de resposta aumenta rapidamente com a carga, mas o ganho em *throughput* é pequeno. O *throughput* máximo alcançável sob condições ideais de carga de trabalho é chamado de capacidade nominal do sistema; para redes, isso é denominado *bandwidth*. A razão da capacidade utilizável (*throughput* máximo alcançável sem exceder um limite de tempo de resposta pré-especificado) para a capacidade nominal é chamada de eficiência.

A taxa de utilização de um recurso é a porcentagem de tempo que o recurso está ocupado, para determinada carga. O recurso do sistema que primeiramente atingir 100% de utilização (o que ficar saturado em primeiro lugar) é chamado de gargalo. A otimização no desempenho desse recurso é a que oferece o maior retorno em termos de investimento. Por isso, achar a utilização de vários recursos no sistema é uma parte importante na análise global.

Outras medidas de desempenho que devem ser citadas são: (1) a confiabilidade: medida pela probabilidade de erros ou pelo tempo médio entre erros; (2)a disponibilidade: é a fração de tempo que o sistema está disponível para utilização.

3.2.2 Métodos de Avaliação de Desempenho

Benchmarks

Benchmarks são programas ou partes agrupadas de programas, de diferentes tipos, por exemplo, relativos a aplicações comerciais ou científicas, e que devem representar a carga real a ser submetida ao sistema; são usados em testes para comparações ou em estudos de projeto ou planejamento. Têm baixo custo, porém apresentam limitações como influência do compilador, da biblioteca em tempo de execução e tamanho do cache. Além disso, não representam em geral sistemas interativos. Alguns *benchmarks* podem abordar basicamente a CPU, a Unidade de Ponto Flutuante e um subsistema de memória; outras podem ser direcionadas para comparação de I/O e outros subsistemas

Monitoramento

Um monitor é uma ferramenta usada para observar as atividades em um sistema (Jain, 1991). Em geral, os monitores observam o desempenho de sistemas, coletam estatísticas de desempenho, avaliam os dados, e mostram resultados. Alguns também identificam áreas de problemas e sugerem soluções. Um gerente de sistemas pode usar um monitor para medir as utilizações de recursos e achar o gargalo, ou também para ajustar os parâmetros do sistema visando melhoria no desempenho. Um analista pode fazer um monitoramento para caracterizar a carga de trabalho, sendo os resultados usados para planejamento de capacidade ou para criar cargas de trabalho para testes. Um programador pode usar o monitor para achar os segmentos do software freqüentemente usados e otimizar seu desempenho.

Os monitores são classificados com base em características como nível de implementação, mecanismo de disparo e capacidade de apresentar resultados. Considerando o nível de implementação podem-se ter monitores de hardware, software, ou híbridos.

Os monitores de software são usados para monitorar sistemas operacionais e softwares de nível mais alto como redes e bases de dados. Em cada ativação várias instruções são executadas e em geral provocam sobrecarga maior que os monitores de hardware.

Os monitores de hardware são dispositivos de hardware específicos para coletar e avaliar dados do objeto em estudo. Não devem ser intrusos em relação ao sistema. Os monitores de hoje são inteligentes e programáveis contendo seu próprio processador, memória e dispositivos de I/O. Consistem de vários elementos, como por exemplo: *probes* - para observar sinais em pontos desejados do hardware do sistema; contadores - incrementados quando um certo evento ocorre; temporizadores - para marcar o tempo ou disparar uma operação de amostragem.

Simulação

Uma simulação é um programa que representa o comportamento de um sistema real. No caso de sistemas computacionais, um hardware ou software pode ser simulado. Para se construir um programa de simulação relativo a um sistema em estudo, deve-se inicialmente especificar os componentes e subcomponentes: hardware, software, aplicativos e todos mecanismos e dispositivos de interesse no caso. Em seguida se constrói um modelo da estrutura do sistema. Após, transforma-se o modelo num programa, representando-se a ocorrência de eventos consecutivos, tais como: atualizações, transações, consultas, interrupções, retardos, esperas. Neles se colocam as características pertinentes: taxas, prioridades, comprimento de mensagens, origens e destinos entre outras. A verificação é a etapa em que se analisa se o programa implementa (representa) corretamente o modelo. E na validação se analisa se as suposições feitas no modelo correspondem à realidade.

A figura 3.1 mostra as diversas etapas para construção de uma simulação.

As variáveis que definem o estado do sistema são chamadas variáveis de estado. Se uma simulação pára na metade, pode ser reiniciada mais tarde, se e somente se os valores de todas variáveis de estado forem conhecidas. Um modelo no qual o estado do sistema é definido em todos instantes é chamado um modelo contínuo no tempo. Se o estado do sistema for definido apenas em instantes particulares do tempo, o modelo é chamado de discreto no tempo. Um modelo é chamado de contínuo ou discreto em relação a estado se as variáveis de estado forem contínuas ou discretas. Um modelo de estados discretos é também chamado um modelo de eventos discretos (idem para estado contínuo). Observe-se que a continuidade do tempo não implica na continuidade de estados e vice-versa, existindo assim 4 combinações distintas possíveis.

Uma simulação pode ser seqüencial ou distribuída. Numa simulação seqüencial, os eventos não podem ser executados eficientemente em sistemas paralelos ou de multiprocessadores, porque existem duas variáveis globais que são compartilhadas por todos processos: o relógio da simulação e a lista de eventos.

Quanto à escolha da linguagem de simulação, pode ser usada uma linguagem de propósito geral, uma extensão dessa linguagem de propósito geral adaptada para simulação, uma linguagem específica para simulação ou ainda um pacote de simulação. Outras considerações

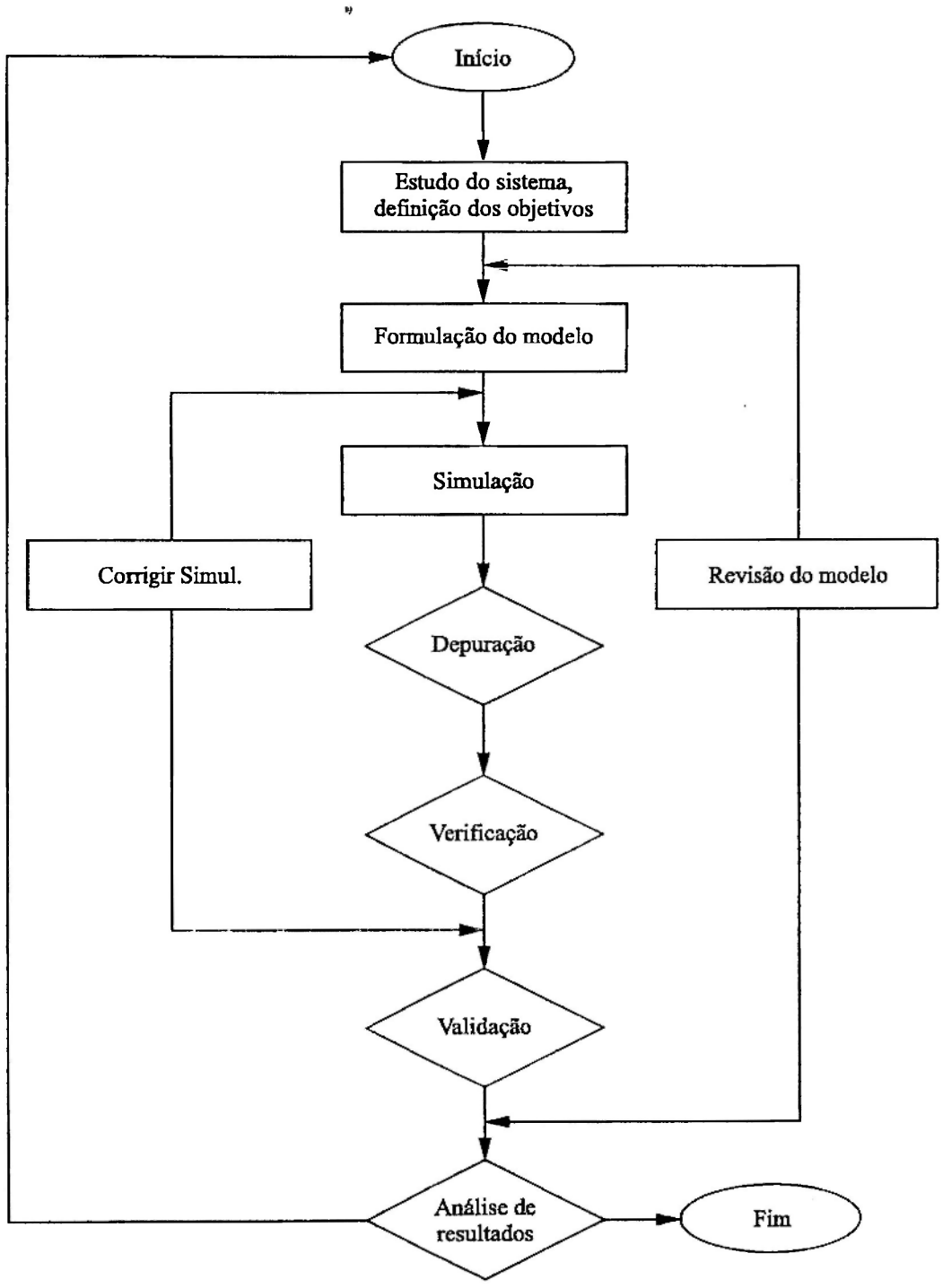


Figura 3.1: Seqüência de etapas para construção de uma simulação

comumente feitas em simulações são verificar se a simulação atingiu um estado permanente (o início da simulação é em geral descartado), o tempo total da implementação da simulação, geração de números aleatórios e escolha de sementes.

Solução analítica (teoria das filas)

Em sistemas computacionais muitas tarefas compartilham os recursos do sistema tais como CPU, discos e outros dispositivos. Em geral apenas uma tarefa pode usar o recurso num determinado instante, e as outras têm que esperar em fila pelo recurso. A teoria de filas auxilia em determinar o tempo que as tarefas gastam em várias filas no sistema. Esses tempos podem ser combinados para se prever o tempo de resposta.

O uso de modelos matemáticos baseados em teoria de filas pode se tornar difícil em sistemas complexos, mas apresenta vantagens como baixo custo e tempo para a avaliação; algumas das características que devem ser especificadas para avaliação com filas são: (i) processo de chegada (muitas vezes se considera como um processo de Poisson, onde os tempos entre chegadas tem distribuição exponencial). (ii) tempo de serviço: muitas vezes considerado como tendo distribuição exponencial; (iii) número de servidores; (iv) capacidade do sistema: máximo no. de clientes que podem permanecer dentro do sistema (v) tamanho da população (de clientes); (vi) disciplina de serviço, sendo alguns exemplos: FCFS (*First Come, First Served*); LCFS-PR (*Last Come, First Served with Preempt and Resume*); RR (*Round-Robin*).

3.2.3 Comentários e Observações

De nada adianta um sistema ser confiável, consistente, econômico, entre outras características desejáveis, se o desempenho (por exemplo, tempo de resposta) não for satisfatório. As componentes de um sistema distribuído trabalham em separado e concorrentemente. Todas as componentes de um sistema distribuído devem ter bom desempenho, quando consideradas em separado, mas isso não garante bom desempenho do conjunto, já que há outros fatores que influem no conjunto, como a distribuição dos servidores de arquivos, configuração correta de protocolos e um escalonamento adequado de processos. O desempenho não se traduz apenas em termos de tempo de resposta e *throughput*, estando também associado a tolerância a falhas, confiabilidade e consistência. Para melhorar o desempenho de um sistema comumente se procura o recurso que corresponde ao gargalo para otimização ou *upgrade*.

A avaliação de desempenho é algo complexo, devido ao ambiente heterogêneo, grande volume de dados que se pode extrair, necessidade de conhecimentos sobre sistemas computacionais e análise estatística. Na avaliação se deve levar em conta os objetivos almejados, ferramentas, orçamento e tempo disponíveis; e ainda integração com o gerenciamento do sistema. Para se avaliar protocolos de comunicação é comum se fazer uso de simulações, também utilizadas, juntamente com modelagem analítica (matemática), para se avaliar desempenho de servidores.

Comumente se usam combinações de diversas técnicas em avaliação de desempenho. Exemplos: (i) um modelo analítico para se achar a faixa de parâmetros juntamente com um simulador para se analisar o desempenho na faixa considerada; (ii) validação de técnica analítica com uso de simulação ou medições reais (monitoramento).

3.3 Estrutura Geral de Serviços Web

3.3.1 Introdução

Servidores, clientes e redes são os principais elementos de ambientes Web. Os serviços Web são providos por servidores muito diferentes usando uma larga variedade de elementos de hardware e software. A explosão de conteúdo na Web criou uma situação onde a maioria do tráfego Internet é relacionado à Web. A partir de 1995, o tráfego Web se tornou a maior carga na Internet. Apesar da conscientização dos problemas relacionados a esse crescimento, muitos sítios Web populares são ainda atendidos a partir de um único local, o que torna necessário freqüentes transferências de dados em WANs, e de forma repetida. Isso resulta em altos tempos de resposta aos usuários e desperdício de largura de banda de rede. Em acréscimo, esse freqüente uso de local único para prover serviços, cria um único ponto de falha entre o *site* Web e seu provedor Internet (Shaw, 1998). O uso de cache e replicação são métodos comuns usados para aliviar os problemas relacionados a respostas lentas.

3.3.2 Cache

Os caches na Web podem ser do lado do cliente (no próprio cliente ou próximo a ele), do lado do servidor ou em outro ponto estratégico no *backbone*. Os itens de dados em caches normalmente têm um tempo de vida (TTL) o que impede o cache de servir dados ultrapassados. Conforme é freqüentemente implementado na Internet hoje em dia, caches na Web usam um *proxy* especial no lado do cliente ou do servidor. Esse *proxy* age como intermediário entre o *browser* e o servidor.

Por exemplo, o SQUID (SQUID, 2004) usa um sistema de cache hierárquico onde os caches pais são colocados próximo a pontos de trânsito principais na Internet. Se um cache filho não tem um objeto, a requisição é passada adiante ao pai, que recupera o objeto do servidor Web principal, coloca o objeto em seu próprio cache e o envia ao filho.

O cache do Apache (Apache, 2004), outro exemplo de cache do lado servidor, é integrado ao popular servidor Apache. É uma solução comumente usada para sítios que não precisam participar de uma grande hierarquia de cache. Em caches do lado servidor, um ou mais servidores de cache agem como *front-end* em relação ao servidor principal. Quando um usuário solicita um documento, o servidor de cache tenta buscar o documento em seu próprio cache ou em um outro

servidor de cache.

Os métodos de cache nos lados cliente e servidor não se conflitam um com outro e podem ser usados isoladamente ou juntos, conforme necessário. Uma limitação intrínseca de cache surge quando o cliente envia informação juntamente com uma requisição para algo mais do que uma página estática (por exemplo, *scripts* CGI ou outro programa similar do lado servidor). Como a resposta do servidor à requisição é dependente dos parâmetros enviados junto com a requisição, não pode ser colocada em cache tanto do lado cliente como servidor. A atualização dos dados também limita a eficiência de caches. Dado um documento em particular, os dados podem ser encontrados em qualquer ou todos os vários caches entre cliente e servidor. Isso cria um problema de consistência pois nenhum cache pode garantir completa atualidade de dados sem mecanismo de invalidação embutido, atualização de entradas inválidas de cache, ou espera pelo término de dados inválidos de cache. Se essa falta de conhecimento da consistência entre os dados de cache e do servidor principal for inaceitável, termina por derrubar a funcionalidade de um sistema de cache, pois toda requisição terá que ser servida, embora indiretamente, do servidor principal (Shaw, 1998).

Historicamente, o uso de cache tem sido uma resposta fraca ao problema do excesso de carga porque as capacidades de gerencia de cache do protocolo HTTP 1.0 eram fracas, o que foi retificado na versão 1.1; com essa última, os autores de documentos e gerentes de servidores Web tem agora a capacidade de exercer mais controle.

3.3.3 Replicação

Para dar suporte a uma alta carga de tráfego, a solução imediata seria colocar mais servidores a trabalhar (Shaw, 1998). A replicação ou espelhamento é a duplicação do servidor Web, incluindo seus dados e quaisquer capacidades auxiliares ou acessórias. Um usuário pode acessar qualquer das réplicas, que qualquer delas dará uma resposta válida e presumivelmente idêntica. A replicação aborda alguns dos problemas criados com requisições ao servidor que não podem utilizar cache, incluindo aquelas que requerem interação C/S (*scripts* CGI, buscas em bases de dados e páginas geradas pelo servidor). Há várias técnicas para se dividir o tráfego entre vários servidores, mas dois métodos comuns são uso de DNS com *round-robin* e redirecionamento dinâmico.

A abordagem de DNS permite que um site implemente o conceito de *server clustering* (agrupamento de servidores). Quando o serviço de DNS recebe um pedido de mapeamento, ele seleciona o endereço IP de um dos servidores do agrupamento. Nos sistemas DNS *round-robin*, o nome do servidor Web é associado a uma lista de endereços IP. Cada endereço IP na lista mapeia para um servidor diferente e cada servidor contém uma versão replicada do servidor Web principal ou acessa um sistema de arquivos comum. Sempre que uma requisição é recebida, o nome do servidor Web é traduzido para o próximo endereço IP da lista. Assim se tenta balancear

a carga entre servidores.

O redirecionamento dinâmico é uma técnica em que o servidor responde a um cliente com um novo endereço de servidor para o qual o cliente irá resubmeter a requisição. O URL redirecionado poderia estar no mesmo computador do servidor principal, ou qualquer um entre vários computadores *back-end*espelhados. O servidor principal redireciona o tráfego para os servidores *back-end* de acordo com algum mecanismo de balanceamento de carga. Embora a técnica seja transparente para usuários, ela adiciona uma conexão extra à requisição original e pode, em alguns casos, aumentar o tempo de resposta e o tráfego em rede.

3.4 Componentes, Evolução Funcional, Interação com Sistemas Operacionais, Modelos de Execução

3.4.1 Componentes e Capacidade

As componentes de um servidor Web são a plataforma de hardware, o sistema operacional, o software servidor e o conteúdo, como ilustrado na figura 3.2 .

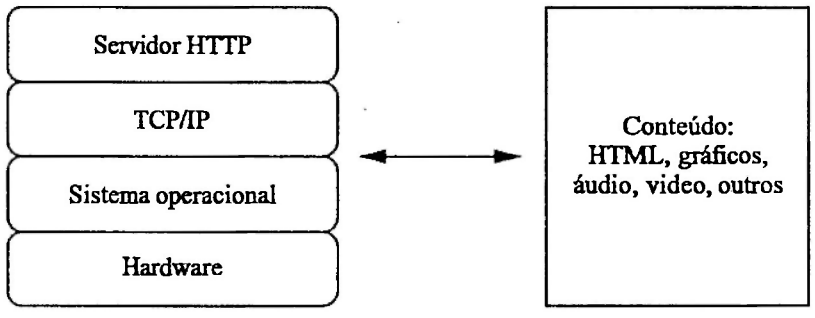


Figura 3.2: Elementos de um servidor Web

A capacidade de um servidor Web é função dessas componentes, juntamente com a conexão de rede e a carga de trabalho. A combinação da capacidade do servidor e a largura de banda da rede determinam a capacidade de um sítio Web (Menasce & Almeida, 1998).

Hardware e Sistema Operacional

Do ponto de vista do hardware, o desempenho de um servidor é uma função das seguintes partes principais: processadores (velocidade e número), memória (capacidade), subsistema de disco (velocidade e capacidade) e placa de interface de rede (largura de banda). Quanto ao Sistema Operacional, os servidores podem executar sobre sistemas operacionais multiusuário de tempo compartilhado, tais como UNIX, Windows NT e outros. Ao se decidir por um sistema se consideram características tais como confiabilidade, desempenho, crescimento em escala e robustez. A implementação do TCP pelo Sistema Operacional é também um fator chave para o desempenho do HTTP.

Conteúdo

Um site Web entrega conteúdo de muitas formas, como por exemplo documentos HTML, imagens, som e trechos de vídeo . O HTML permite que se integre vídeo, aplicações de software e objetos multimídia na Web. Porém, a maioria dos documentos HTML consiste de texto e imagens embutidas. E um documento HTML com imagens é uma combinação de múltiplos objetos, o que gera várias requisições em separado ao servidor. O usuário vê só um documento (página Web), mas o servidor vê uma série de requisições (para documentos) em separado para constituir a página.

Servidor Web

O servidor Web (software servidor), às vezes citado como *HTTP daemon*, é um programa que controla o fluxo de dados de entrada e de saída em um computador ligado à Internet, basicamente escutando e atendendo requisições. Para aumentar a velocidade do serviço os servidores Web manipulam mais de uma requisição por vez. Isso pode ser feito de três modos diferentes (Yeager & McCrath, 1996): fazendo uma cópia (usando *fork* do processo HTTP para cada requisição; através de uso de múltiplas *threads* para executar o programa HTTP; espalhando as requisições entre um *pool* de processos executando. Supondo o uso de uma API de tipo UNIX (embora a maioria das discussões seguintes seja válida também para servidores baseados em Windows NT), segue-se uma descrição sucinta da execução no servidor.

Um servidor HTTP recebe requisições de seus clientes através de conexões TCP. Na versão HTTP 1.1 várias requisições podem ser enviadas serialmente numa conexão. O servidor escuta em uma porta bem conhecida para novas requisições de conexão. Quando uma nova solicitação de conexão chega, o sistema entrega a conexão à aplicação do servidor através da chamada ao sistema *accept*. O servidor então espera que o cliente envie uma requisição para dados nessa conexão, faz o *parsing* da requisição, e a seguir retorna a resposta na mesma conexão. Os servidores Web tipicamente obtêm a resposta do sistema de arquivo local, enquanto *proxies* obtém respostas de outros servidores; contudo, ambos tipos de servidores podem usar um cache para aumentar a velocidade de recuperação. (Stevens, 1996) descreve as operações básicas de servidores HTTP em mais detalhe.

3.4.2 Evolução Funcional

Processamento de requisições estáticas

A arquitetura de servidores Web tem passado por mudanças radicais (Banga & Mogul, 1998). Os servidores primitivos faziam a cópia de um novo processo (usando *fork* para manipular cada conexão HTTP, seguindo o modelo UNIX clássico. A sobrecarga usando *fork* rapidamente se tornou um problema, e os servidores subseqüentes tais como o httpd do NCSA (do qual se

derivou o Apache), usavam um conjunto de processos previamente gerados via *fork*.

Nesse modelo, mostrado na figura 3.3, um processo mestre aceita novas conexões e as passa para os processos trabalhadores já gerados anteriormente usando sockets do domínio UNIX.

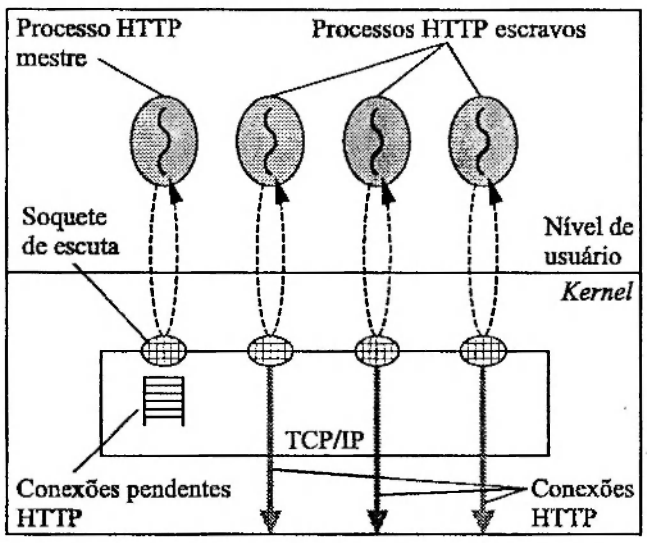


Figura 3.3: Um servidor HTTP de um processo por conexão com um processo mestre

A inovação seguinte elimina o processo mestre. Em vez disso, cada servidor previamente gerado chama *accept* diretamente para aceitar novas requisições de conexões. O servidor Apache tem essa arquitetura (Banga & Mogul, 1998). Os servidores de processos múltiplos podem sofrer grande sobrecarga para mudança de contexto e comunicação entre processos. Por causa disso muitos servidores recentes usam uma arquitetura dirigida por eventos, de um único processo.

No modelo dirigido por eventos, visto na figura 3.4 , o servidor usa uma única *thread* para gerenciar todas conexões no servidor.

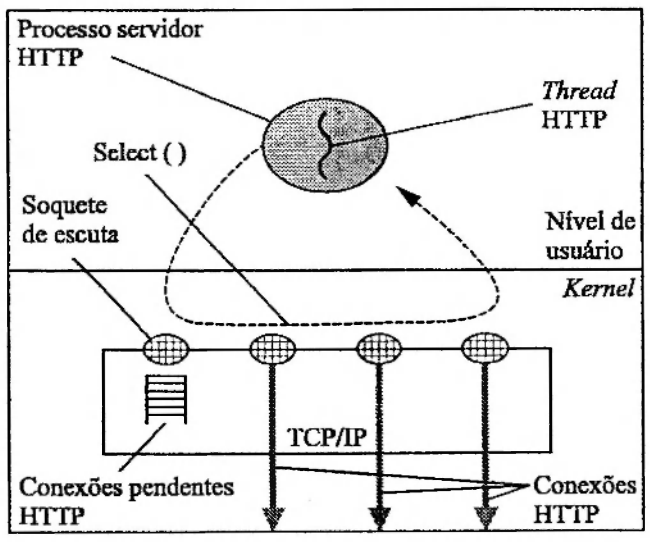


Figura 3.4: Um servidor dirigido por eventos, de um único processo

Os servidores dirigidos por evento projetados para multiprocessadores usam uma *thread* ou processo por processador. Um servidor dirigido por eventos usa a chamada de sistema *select* (ou a *poll*) para esperar simultaneamente por eventos em todas conexões manipuladas pelo servidor. Quando a *select* entrega um ou mais eventos, o laço principal do servidor invoca *handlers* para cada conexão pronta. O Squid (Chankhunthod *et al.*, 1996), o Zeus (Zeus, 2004), o thttpd [Thttpd] e vários servidores de pesquisa (Banga & Druschel, 1997), (Kaashoek *et al.*, 1996) e (Pai *et al.*, 1997) todos usam uma arquitetura dirigida por eventos.

Uma outra alternativa é a arquitetura de um único processo com múltiplas *threads*, como mostra a figura 3.5 .

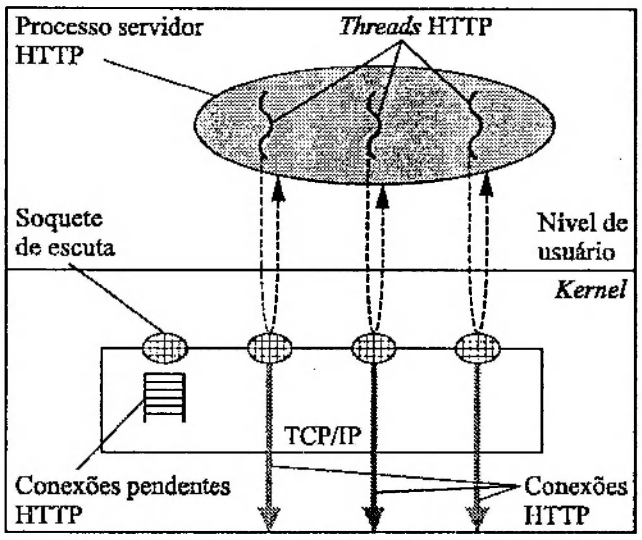


Figura 3.5: Um servidor de um único processo com múltiplas *threads*

Em um servidor simples de *threads* múltiplas, cada conexão é atribuída a uma *thread* dedicada. O escalonador de *threads* é responsável por fazer o compartilhamento de tempo da CPU entre as várias *threads* de servidores. Já que há apenas um processo servidor, a sobrecarga de chaveamento de contexto é muito menor do que em uma arquitetura de um processo por conexão. Contudo, o suporte eficiente para grandes grupos de *threads* não está sempre disponível, e assim alguns servidores usam uma abordagem híbrida, na qual um conjunto de tamanho moderado de *threads* é multiplexado entre muitas conexões, usando eventos para controlar a atribuição de conexões a *threads*.

Processamento de requisições dinâmicas

A discussão acima supôs requisições para documentos estáticos. O HTTP também dá suporte a requisições de documentos dinâmicos, cujo conteúdo depende dos parâmetros particulares e do instante da chegada da requisição. Os documentos dinâmicos são tipicamente criados por programas auxiliares que constituem uma terceira camada e executam como processos em separado. Essas páginas são transferidas subseqüentemente para o cliente através do servidor

Web.

Para tornar mais fácil a construção desses programas auxiliares, várias interfaces padrão governando a comunicação entre servidores Web e os programas auxiliares foram definidas: exemplos incluem CGI e o FastCGI. A interface mais antiga, o CGI, cria um novo processo para manipular cada requisição de documento dinâmica. O FastCGI, mais recente (Pinheiro, 2001), permite processos servidores de documentos dinâmicos persistentes.

Conforme (Banga & Mogul, 1998) os servidores Internet estão se movendo em direção a uma arquitetura onde um pequeno conjunto de processos implementa a funcionalidade do servidor. Existe um processo servidor principal, que implementa a funcionalidade para servir todos documentos estáticos. Os documentos dinâmicos são criados ou por código de biblioteca dentro do processo servidor principal, ou por processos auxiliares cujo código precisa ser mantido em separado das outras componentes do servidor por razões de isolamento de falhas. Em um sentido, isso é ideal porque a sobrecarga de chavear contexto entre domínios de proteção ocorre apenas se absolutamente necessário. Contudo, estruturar um servidor como um pequeno conjunto de processos leva a certos problemas relacionados ao sistema operacional, a serem abordados na próxima subseção.

3.4.3 Interação com Sistemas Operacionais

Os sistemas operacionais correntes deixam a desejar em relação a um suporte eficiente e passível de crescimento em escala tanto para servidores dirigidos por evento como para os de *threads* múltiplas. Na próxima subseção são descritas algumas dessas faltas de adequamento com implicações para o desempenho do servidor.

Servidores Dirigidos por Evento

O desempenho de servidores dirigidos por evento depende criticamente de 3 fatores. Primeiramente, os mecanismos de entrega e manipulação dos eventos devem ser eficientes e escaláveis. Em segundo lugar, a única *thread* de controle nunca deve bloquear ao manipular o evento. Se o manipulador de eventos bloqueasse, isso retardaria a entrega e a manipulação de eventos subsequentes. Finalmente, quaisquer mudanças no ambiente de um servidor deveriam ser informadas ao servidor de modo assíncrono; um servidor nunca deveria ter que recorrer a uma verificação de estado nos recursos que ele gerencia para operação correta (Banga & Mogul, 1998).

Servidores com *Threads* Múltiplas

Uma pesquisa anterior expôs as limitações de suporte a *threads* em sistemas operacionais correntes (Anderson *et al.*, 1995). *Threads* do kernel puras impõem a sobrecarga de uma chamada

de kernel para cada operação de mudança de contexto e sincronização. As *threads* em nível de usuário têm sincronismo e mudança de contexto eficiente, mas se qualquer *thread* em nível de usuário bloqueia num evento de I/O, todas *threads* no processo param. Isso é análogo ao problema de bloqueio de I/O em servidores dirigidos por eventos.

Infelizmente, muito pouco tem sido publicado sobre o uso e o desempenho de servidores puramente de *threads* múltiplas, embora alguns servidores de alto desempenho importantes como o *front-end* do Alta Vista tenham adotado com sucesso essa abordagem.

Ineficiências de Modelos Correntes quanto a Gerencia de Recursos e Escalonamento

A maioria dos sistemas operacionais trata um processo, ou uma *thread* dentro de um processo, como a entidade escalonável. Os processos também são as entidades usadas para a alocação de recursos tais como tempo de CPU e memória. As políticas de escalonamento e alocação de memória do sistema tentam prover equidade entre essas entidades e comportamento adequado do sistema sob várias condições de carga.

Mas, na maioria dos sistemas operacionais o *kernel* em geral não controla ou leva em conta adequadamente os recursos consumidos durante o processamento do tráfego de rede. A maioria dos sistemas faz o processamento do protocolo no contexto de interrupções de software. As interrupções de software têm prioridade estritamente mais alta do que a execução de qualquer código no nível de usuário. Isso pode levar a anomalias de escalonamento, *throughput* diminuído e *starvation* ou *livelock*, (Druschel & Banga, 1996), (Mogul & Ramakrishnan, 1996), o que é particularmente importante para servidores porque eles, por sua natureza particular, demandam serviços intensos de rede.

Em um modelo centrado em processo, o *kernel* não distingue, geralmente, entre subatividades independentes dentro de um processo. Por exemplo, um processo não pode especificar as prioridades relativas de suas várias conexões de rede, e a prioridade de estabelecer novas conexões relativas ao serviço das conexões existentes. Dessa forma, muito embora uma aplicação possa priorizar a manipulação de conexões em código em nível de usuário, isso não vai muito além em controlar as taxas de progresso relativo de conexões.

Essa falta de controle sobre recursos do *kernel* torna difícil construir um servidor Web para prover qualidade de serviço diferenciada a seus clientes (Menasce & Almeida, 1998). Por exemplo, considere-se um servidor Web que gostaria de fornecer aos clientes serviço diferenciado dependendo de uma variedade de fatores, tais como a diferença em taxas de acesso pagas por clientes corporativos e usuários domésticos, ou as diferenças em taxas pagas pelos proprietários de conteúdos variados providos pelo servidor. Infelizmente, embora tais situações sejam cada vez mais abundantes na Internet, os servidores executando em sistemas operacionais tradicionais não podem prover o suporte de qualidade de serviço necessário.

3.4.4 Modelos de Execução

A escolha entre modelos de execução permanece complexa. Uma boa implementação dirigida por eventos pode executar melhor do que uma baseada em *threads*. Essa expectativa pode ser baseada em razões dadas por (Ousterhout, 1996), incluindo a necessidade para uso de trancas e mudanças de contexto em um sistema baseado em *threads*. Também, um programa baseado em eventos pode usar uma única pilha de execução; um programa baseado em *threads* usa pilhas múltiplas, colocando mais pressão nos caches de dados e no TLB.

(Ousterhout, 1996) admite que *threads* são uma abstração mais poderosa, porém o argumento primário contra elas é a complexidade de programação. Conforme (Banga & Mogul, 1998), a distinção de desempenho entre servidores baseados em *threads* e dirigidos por evento é provavelmente secundária em relação a muitas outras decisões de implementação e projeto, tais como a ordem na qual os eventos são manipulados. Em virtude de os sistemas operacionais existentes não suportarem de forma otimizada servidores em grande escala usando *threads* ou eventos, não se pode ainda usar experimentos para decidir qual modelo provê inerentemente melhor desempenho. Certamente as *threads* são necessárias para explorar o poder total de um multiprocessador. Um modelo híbrido, usando um número moderado de *threads* e um mecanismo de notificação baseado em eventos pode ser o melhor.

3.5 Modelagem e Avaliação de Servidores Web

3.5.1 Desempenho em serviços Web

Os problemas de desempenho na Internet (sobre a qual estão assentados os serviços Web) são exacerbados pela natureza imprevisível da recuperação da informação e de requisição de serviços através da Web. Muitos sítios na Web têm aumento imenso de requisições em certos períodos, em relação a períodos "calmos"no restante do tempo. Outras características distinguem os sistemas baseados na Web de sistemas C/S tradicionais. O tamanho dos objetos (por exemplo, gráficos, vídeo, áudio) recuperado dos servidores Web é variável (Menasce & Almeida, 1998). Incluído no problema, há um grande conjunto de aplicações *back-end* executando nos servidores Web. A combinação dessas características únicas na Web gera vários problemas de desempenho, tais como sobrecarga no servidor e gargalo na rede, os quais influem diretamente no desempenho visto por um usuário final. Este desempenho é usualmente restrito pelo desempenho de alguns componentes, por exemplo, servidor, ligações de rede e roteadores, ao longo do caminho entre cliente e servidor.

O desempenho na Web pode ser analisado sob diferentes pontos de vista.

A visão de desempenho de um usuário tem relação com tempo de resposta rápido e nenhuma conexão recusada. Os usuários vêem a qualidade do serviço prestado através de tempo

de resposta, disponibilidade, confiabilidade, previsibilidade e custo. A disponibilidade representa a porcentagem de tempo que um sítio está disponível durante um período de observação. A confiabilidade mede a ocorrência de falhas durante o processamento de serviços. O problema da qualidade de serviço é muito aumentado pela natureza imprevisível da busca dos usuários por serviços Web.

Já o desempenho visto por um *Webmaster* é orientado para alto *throughput* de conexão e alta disponibilidade.

O que é comum a todas percepções de desempenho é a necessidade para medidas quantitativas que descrevam o comportamento do serviço.

As duas características seguintes têm profundo impacto no comportamento dos servidores Web (Almeida *et al.*, 1996).

- Distribuições de cauda pesada: estudos recentes têm mostrado que as distribuições de tamanhos de arquivos na Web exibem caudas pesadas, incluindo os arquivos armazenados nos servidores, solicitados pelos clientes e aqueles transmitidos pela rede. Uma variável aleatória X tem distribuição de cauda pesada se $P(X > x) \sim x^{-\alpha}, 0 < \alpha < 2$, onde $f(x) \sim a(x)$ significa que $\lim_{n\to\infty} f(x)/a(x) = c$ para alguma constante positiva c. A distribuição de cauda pesada mais conhecida é a distribuição de Pareto. As distribuições de cauda pesada teóricas tem variância infinita, o que, em termos práticos, significa que observações muito grandes são possíveis com probabilidade não desprezível. Uma possível explicação é a presença de arquivos de multimídia grandes que contribuem para aumentar a cauda da distribuição.

- Processos de curta duração: muitas implementações (as mais antigas) de servidor Web usam uma nova conexão TCP para quase toda requisição. Várias referências noticiam que mais de 90% das requisições de clientes são para arquivos de imagem ou HTML pequenos. A combinação desses dois fatos traz a criação de um grande número de processos de curta duração, fato observado em servidores ocupados. Isto traz novos desafios a alguns sistemas operacionais que não estão regulados para manipular um grande número de processos curtos. Esses processos também representam novos problemas para monitoração de desempenho. Os autores em (Somin *et al.*, 1996) apontam os problemas ao se tentar medir tempo de CPU usado por processos individuais de curta duração.

3.5.2 Métricas de Desempenho e Parcelas do Tempo Gasto no Servidor

A latência e o *throughput* são duas das métricas mais importantes para sistemas Web. A taxa com que as requisições Web são atendidas representa o *throughput* da conexão. É usualmente expressa em operações HTTP por segundo. Devido à grande variabilidade no tamanho dos objetos Web requisitados, o *throughput* é também medido em termos de bits/seg. A latência

no servidor, tempo requerido para se completar uma requisição, é uma componente do tempo de resposta do cliente. A latência média no servidor é o tempo médio que ele leva para manipular uma requisição. Assim, as medidas mais comuns de desempenho de servidor Web são conexões por segundo, megabits por segundo, tempo de resposta e erros por segundo.

Conforme (Almeida *et al.*, 1996) o tempo gasto apenas no servidor pode ser dividido em três parcelas: (i) o tempo de *parsing* que é o intervalo que começa logo após o estabelecimento da conexão e acaba quando o cabeçalho da requisição foi filtrado e está pronto para ser processado; (ii) o tempo de processamento é o tempo gasto realmente processando a requisição. Não inclui o tempo de *logging* do servidor. Leva em conta o tempo gasto lendo a URL e o tempo preciso para mover o arquivo da memória ou disco para a rede; (iii) o tempo de *logging* é o tempo gasto executando *logs* (registros) padrão no HTTP. Após fazer o *logging*, um processo servidor está pronto para manipular uma nova requisição. Note-se que as rotinas de temporização do Linux não são acuradas o suficiente para levar em conta as três componentes do tempo de execução de uma requisição curta. A resolução de tempo é da ordem de 10 milissegundos (Welsh, 1994). Alguns estudos como (Almeida *et al.*, 1996) e (Banga & Mogul, 1998) enfatizaram o papel importante do sistema operacional e da implementação do protocolo de rede no desempenho final do servidor Web.

3.5.3 Carga de Trabalho

Para avaliar o desempenho de servidores Web é preciso um bom conhecimento da carga de trabalho, que engloba as requisições submetidas ao sistema pelos usuários durante qualquer período de tempo. É difícil saber de antemão quantas pessoas acessarão um sítio, em que instantes no tempo e quais serão seus padrões de uso. Contudo, estudos experimentais de (Arlitt & Williamson, 1996), (Crovella & Bestavros, 1996) identificaram algumas propriedades e invariantes da carga de trabalho.

Impacto de Páginas Dinâmicas no Desempenho

As páginas dinâmicas oferecem interação e efeitos especiais; essas páginas podem ser obtidas usando-se programas do lado cliente ou servidor (Menasce & Almeida, 1998). Por exemplo, Java, Active X e HTML dinâmico são técnicas usadas do lado cliente na Web. Com qualquer uma delas, efeitos especiais podem ser adicionados a documentos Web sem se depender de programas do lado servidor. Com isso se reduzem tempos de resposta e se evitam retardos de rede e do servidor. No lado servidor a interação pode ser alcançada emitindo-se *scripts* CGI.

Um programa CGI (por exemplo, Perl, C ou Java) executa como um processo em separado fora do servidor e devolve dados (por exemplo, uma página gerada dinamicamente ou o resultado de uma busca) ao servidor. Como um processo é iniciado cada vez que um CGI é necessário, essa técnica pode ser muito custosa. No caso de servidores executando sobre

Windows NT, o *scripting* do lado servidor pode ser implementado com a ISAPI (Blaszczak, 1996), que usa *threads* para reduzir a sobrecarga. A carga de CGI pode ter um impacto imenso no desempenho do servidor, dado que os *scripts* criam processos que demandam serviços do sistema operacional e consomem tempo de CPU, memória e operações em disco. Se um servidor tem só um processador, o servidor Web e os *scripts* CGI compartilharão os mesmos recursos, o que pode tornar mais lenta a execução de requisições HTTP. Em alguns sítios, os programas CGI, incluindo máquinas de busca, representam a maior parte da carga de trabalho.

Representação da Característica de Cauda Pesada

Uma maneira de representar a característica de cauda pesada de cargas de trabalho em modelos é usar a noção de classes (Menasce & Almeida, 1998). Cada classe compreende requisições similares com respeito ao tamanho de documentos a serem buscados. Assim, requisições semelhantes em termos de uso de recursos são agrupadas. Isso reduz a variabilidade das medições, melhorando seu significado estatístico. Por exemplo, se poderia agrupar as requisições Web de uma carga de trabalho em três classes: pequenas páginas (até 5KB), médias (de 5 a 50KB) e grandes (acima de 50KB).

Carga em rajadas

Alguns novos fenômenos foram observados em sistemas distribuídos de porte como a Internet, intranets e a Web, a partir da década de 1990. Vários estudos (Arlitt & Williamson, 1996), (Crovella & Bestavros, 1996), (Leland *et al.*, 1994) revelaram novas propriedades do tráfego de rede, tal como a auto-similaridade. Um processo auto-similar se mostra com rajadas em várias escalas de tempo. Em (Crovella & Bestavros, 1996) é mostrado que o tráfego Web contém rajadas observáveis em 4 ordens de grandeza, significando que as rajadas são observáveis quando, a partir de um gráfico inicial do tráfego Web (dado em bytes) como uma função de intervalos de tempo de 1000 segundos, os períodos de tempo são mudados para 100, 10 e 1 segundo, respectivamente. A auto-similaridade do tráfego de rede em LANs e WANs, causada, entre outros fatores, pela predominância do tráfego Web, implica então em rajadas observáveis em várias escalas de tempo. A ocorrência de rajadas em um período observado pode ser representada por um par de parâmetros (a, b) (Menasce & Almeida, 1998). O parâmetro a é o quociente entre a taxa máxima observada de requisições e a taxa média durante o período observado. O parâmetro b é a fração de tempo durante a qual a taxa de chegada instantânea excede a taxa de chegada média. Os modelos analíticos podem assim incorporar o efeito de rajadas. Em (Banga & Druschel, 1997) se mostra que as taxas de pico durante as rajadas excedem a taxa média em fatores de 5 a 10 e podem facilmente ultrapassar a capacidade do servidor. Mesmo uma pequena quantidade de rajadas (a=6, b=5%) pode degradar o *throughput* de um servidor Web em 12 a 20%.

3.5.4 *Benchmarks* para Servidores Web

Introdução

Aqui serão descritos alguns *benchmarks* populares de servidores Web: Webstone (Webstone, 2004), SPECWeb (SPECWeb, 2004), S-clients (S-clients, 2004), SURGE (SURGE, 2004) e Httperf (Mosberger & Jin, 1998) (todos simulam clientes Web). Esses programas geram requisições a um servidor, de acordo com algumas características de carga de trabalho especificadas, recebem as respostas devolvidas pelo servidor e coletam as medições. É importante observar que os *benchmarks* de servidores Web são em geral levados a efeito em LANs isoladas, pequenas, sem quaisquer erros de transmissão. Ao contrário, os serviços Web oferecidos no mundo real são acessados através da Internet ou grandes intranets, o que envolve conexões em WANs, *gateways*, roteadores, pontes e *hubs*, os quais tornam o ambiente de rede com ruído e tendência a erros. Dessa forma, a análise de resultados em *benchmarks* Web deve levar isso em conta.

Webstone

O Webstone é um *benchmark* configurável para servidores Web que usa parâmetros de caracterização de carga e processos cliente para gerar tráfego HTTP visando testar um servidor de diferentes maneiras. Foi projetado para medir o *throughput* máximo do servidor e o tempo de resposta médio para conectar com o servidor. Ele faz um número de requisições GET para documentos específicos do servidor Web em estudo e coleta dados de desempenho. O Webstone é um *benchmark* distribuído, de processos múltiplos, composto de processos cliente e mestre. O processo mestre, local ou remoto, gera um número predefinido de processos cliente que começam a gerar requisições Web no servidor. Depois que todos processos cliente acabam de executar, o processo mestre reúne os dados coletados pelos processos cliente e gera um relatório resumido de desempenho. O usuário pode especificar a duração do teste ou o número total de iterações.

SPECWeb

Trata-se de um *benchmark* padronizado desenvolvido pela SPEC (*Systems Performance Evaluation Consortium*) para medir a capacidade de um sistema para agir como um servidor Web. A arquitetura foi inicialmente baseada na estrutura do *benchmark* LADDIS para gerar carga. O LADDIS mede o desempenho do servidor NFS gerando uma carga sintética de operações NFS e a seguir medindo o tempo de resposta da requisição ao servidor para diferentes níveis de carga. O SPECWeb96 foi o primeiro benchmark padronizado para medir desempenho de servidores Web. Utilizava como carga somente páginas estáticas. Posteriormente foi apresentado o SPECWeb99. Neste, 30% das requisições que ele produz são para conteúdo dinâmico. Ao invés de uma métrica de desempenho convencional tal como fluxo ou tempo de resposta, a métrica primária usada pelo SPECWeb99 é o no. (máximo) de conexões simultâneas solicitando a carga

sintética(*benchmark*) pré-definida que um servidor Web é capaz de lidar enquanto ainda suporta requisitos específicos de taxa de erros e *throughput* (Nahum, 2002).

S-clients

Esse gerador de carga sintética introduziu a noção de geração de carga com *open-loop*. Muitos geradores de carga são *closed-loop*, no sentido de que são fortemente acoplados ao servidor que eles estão medindo, esperando para gerar uma nova requisição apenas depois que se tenha recebido uma resposta. Gerando requisições a uma taxa ortogonal à (independente da) capacidade do servidor, o S-clients pode gerar cargas que sobrecarregam o servidor.

SURGE: Scalable URL Reference Generator

O SURGE foi o primeiro gerador de carga que dedicou especial atenção a reproduzir características da carga de trabalho do servidor tais como popularidade de documentos, referências embutidas e tamanho dos arquivos transferidos. O SURGE originalmente considerava apenas tráfego HTTP 1.0, sendo depois estendido para implementar conexões persistentes e *pipelining*. A intenção do SURGE é representar uma carga de trabalho real na Web através de características que podem ser divididas em duas categorias principais (Barford & Crovella, 1998).

A 1a. categoria é referente ao conceito de *user equivalents* que diz respeito à intensidade de demanda gerada por uma população de algum número conhecido de usuários. Ou seja, a intensidade de demanda de serviço gerada pelo SURGE pode ser medida em *user equivalents*. Um *user equivalent* (UE) é definido como um único processo em um laço sem fim que alternadamente (i)faz requisições de arquivos Web e (ii) permanece ocioso. Cada UE é portanto um processo ON/OFF.

A 2a. categoria se refere a modelos de distribuições de probabilidade para representar características encontradas em cargas reais na Web, com relação aos seis itens seguintes. (1) tamanho dos arquivos armazenados no servidor; (2) tamanho dos arquivos requisitados pelos clientes ao servidor; (3) popularidade dos arquivos, o que é relacionado ao item anterior, pois representa o no. relativo de requisições feitas a arquivos individuais no servidor, o que pode ser considerado como seguindo a lei de Zipf; (4) o no. de documentos embutidos em um objeto (uma página) Web; (5) localidade temporal: refere-se à probabilidade de que um arquivo, tendo sido requisitado, será requisitado novamente em um futuro próximo. Um modo de se medir a localidade temporal é usando a noção de *stack distance*, conforme segue. Dado uma série de requisições, sua seqüência de *stack distances* correspondentes pode ser gerada do seguinte modo. Considere-se que os arquivos (requisitados) sejam armazenados em uma pilha, estando o último no topo da pilha. A profundidade na pilha em que cada arquivo requerido é encontrado é sua *stack distance*; (6) tempos em OFF: modelar os tempos em que se fica ocioso é necessário para se capturar a natureza de rajadas de requisições de um usuário Web individual.

Modelos para cada uma das seis distribuições são necessários bem como métodos para combinar essas distribuições em um único fluxo de saída, para funcionamento correto da ferramenta.

Httperf

O httperf (Mosberger & Jin, 1998) é uma ferramenta livre para avaliar o desempenho de servidores Web, desenvolvida pelo laboratório de pesquisas da HP. Essa ferramenta também fornece recursos para gerar cargas Web variadas. O httperf é dividido logicamente em 3 partes diferentes: o núcleo da máquina httperf, a geração de carga e a coleta de estatísticas. A máquina httperf manipula toda comunicação com o servidor e assim toma conta da gerência de conexões, da geração de requisições e da manipulação das respostas. A geração de carga é responsável por iniciar as chamadas HTTP especificadas, nos instantes adequados, tal que uma carga de trabalho em particular é induzida no servidor. A 3a. parte, coleta de estatísticas, é responsável por medir várias quantidades e produzir estatísticas de desempenho relevantes.

As interações entre essas três partes ocorrem através de um mecanismo de sinalização de eventos geral, embora simples. A idéia é que sempre que algo interessante ocorre dentro do httperf, um evento é sinalizado. As partes interessadas em observar um evento em particular podem registrar um *handler* para o evento. Esses *handlers* são invocados sempre que o evento seja sinalizado, isto é, o *handler* é uma rotina que assume o controle após a sinalização. Por exemplo, o coletor de estatísticas básicas mede o tempo que se leva para estabelecer uma conexão TCP registrando o manipulador de eventos para os eventos que sinalizam o início e o estabelecimento de uma conexão, respectivamente. Similarmente, um gerador de carga de trabalho responsável por gerar um padrão de acesso de URL em particular, pode registrar um *handler* para o evento, indicando a criação de uma nova chamada. Sempre que esse *handler* é invocado, o gerador de URL pode inserir a URL apropriada na chamada sem ter que se preocupar ele próprio com os outros aspectos de manipulação e criação de chamadas.

Serão abordadas agora algumas limitações com que se deve preocupar ao usar o httperf. É interessante considerar o que limita a carga oferecida que um cliente pode sustentar. Deixando de lado o limite óbvio que a CPU do cliente impõe, há uma variedade de recursos que pode se tornar o gargalo de primeira ordem (quanto ao número de requisições que o cliente pode gerar). É importante manter esses limites à vista evitando a armadilha de confundir limites de desempenho no cliente com limites no servidor. Os três gargalos mais importantes no cliente, descritos em (Mosberger & Jin, 1998) são:

(i)Tamanho do espaço de portas TCP: Os números de portas TCP são de 16 bits. Dos 64K números de porta disponíveis, 1024 são tipicamente reservados para processos privilegiados. Isso significa que uma máquina cliente executando Httperf pode usar no máximo 64.512 números de portas. Já que um dado número de porta não pode ser reutilizado até que o estado *TIME-WAIT*

do TCP expire, isso pode limitar a taxa oferecida sustentável do cliente. Especificamente, com um *timeout* de 1 minuto (comum para SOs derivados do BSD) a taxa máxima sustentável por cliente é cerca de 1075 reqs/seg. Com o valor recomendado de 4 minutos da RFC-793, a taxa máxima cairia para só 268 reqs/seg.

(ii)Número de descritores de arquivo abertos: a maioria dos sistemas operacionais limita conjuntamente o número total e por processo de descritores de arquivo que podem ser abertos. O número de arquivos abertos para o sistema global não é normalmente um fator limitante. Já se o número de descritores de arquivo por processo se torna o gargalo do cliente, é possível evitá-lo regulando o sistema operacional para permitir um número maior de descritores abertos ou decrescendo o valor de *timeout* do httperf.

(iii)Memória do *buffer* de *sockets*: cada conexão TCP contém um *buffer* de *socket* de *receive* e *send*. Por *default* o httperf limita os *buffers* de *send* a 4KB e os *buffers* de *receive* a 16 KB. Com limites na faixa de Kbytes, esses *buffers* são tipicamente os custos dominantes por conexão no que diz respeito a consumo de memória httperf. A carga oferecida que um cliente pode sustentar é por isso limitada também pela quantidade de memória disponível para *buffers* de *socket*.

O httperf possui três geradores de carga: orientado a requisição, orientado a seção e mixto. Nesta tese usou-se especificamente o gerador orientado a sessão, com o uso do comando *wsesslog*, tempos entre chegadas de sessões probabilísticos e arquivos representando sessões bem definidos, para representar sessões com diferentes médias de tamanhos de documentos, no. de documentos por sessão e taxas de chegada.

O httperf foi utilizado neste trabalho por sua facilidade de uso, ter código que pode ser estendido e oferecer uma variedade de opções e configurações para testar diferentes cenários no servidor; a ferramenta também contém o modelo de *open-loop* usado pelo S-Clients e se adequou perfeitamente às nossas necessidades experimentais na validação (capítulo 6).

3.5.5 Comentários Adicionais

A Web baseada em WANs tem características de rede que diferem das LANs nas quais os servidores Web são usualmente avaliados. Os aspectos de desempenho de um servidor que são dependentes de tais características de rede não são avaliados. Em particular, o método simples não modela retardos elevados e variáveis em WANs, que, sabe-se, provocam longas filas de SYN-RCVD na *socket* de escuta do servidor. Perdas de pacotes devido a congestionamento também estão ausentes em plataformas baseadas em LANs. (Maltzahn *et al.*, 1997) descobriram uma grande diferença no desempenho do *proxy* Squid dos números idealizados noticiados em (Chankhunthod *et al.*, 1996). Boa parte dessa degradação é atribuído a tais efeitos de WANs, que tendem a manter recursos do servidor, tais como memória, comprometidos por períodos extensos de tempo.

Ao gerar requisições sintéticas HTTP a partir de um pequeno número de máquinas cliente, deve se tomar cuidado para que as restrições de recursos nessas máquinas não distorçam acidentalmente o desempenho do servidor medido. Com um número crescente de clientes simulados por máquina cliente, a disputa por memória e CPU no lado do cliente tende a surgir. Eventualmente se alcança um ponto onde o gargalo em uma transação Web não é mais o servidor, mas o cliente. Projetistas de *benchmarks* de servidores Web comerciais também observaram essa armadilha. O *benchmark* Webstone avisa explicitamente sobre esse problema em potencial, mas não dá método sistemático para evitá-lo.

O fator primário para se impedir que gargalos de cliente afetem resultados de desempenho de servidor é limitar o número de clientes simulados por máquina cliente. Em acréscimo, é importante usar uma implementação eficiente de TCP-IP (em particular, uma implementação da tabela PCB eficiente (Mogul, 1995) nas máquinas cliente), e evitar operações de I/O nos clientes simulados que poderiam afetar a taxa de transações HTTP de modo descontrolado. Por exemplo, escrever informação de *log* em disco pode afetar o comportamento do cliente de modo indesejável.

3.5.6 Exemplo de modelo do lado servidor

Modelos do lado servidor mostram a visão de desempenho conforme percebida pelo servidor. Podem ser usados para, por exemplo, se saber qual deve ser a largura de banda do link que conecta o servidor à Internet tal que níveis de serviço aceitáveis sejam providos, ou para se comparar soluções para o atendimento, por exemplo, pode-se considerar redirecionamento ou sistema de arquivos montado localmente, compartilhado por vários servidores Web (Menasce & Almeida, 1998).

Os modelos de dimensionamento de um servidor Web são centrados em torno de quatro gargalos potenciais de sistema: rede, memória, disco e CPU. Os projetistas de servidores Web não têm, em geral, controle dos ambientes externos ao servidor Web ou das conexões de redes locais à Internet. Assim, os tempos de resposta de clientes não podem ser garantidos. O dimensionamento de um servidor Web é muitas vezes limitado a configurar o sistema para acomodar um requisito de *throughput* alvo.

Exemplo: modelo com um único servidor Web

A figura 3.6 mostra um ambiente com um único servidor Web no sítio. Esse servidor é conectado a uma LAN, que é conectada a um roteador, o qual conecta o sítio ao ISP (Provedor de Acesso à Internet), e daí é feita a conexão à Internet. O conjunto de documentos (*document tree*) atendidos pelo servidor Web é armazenado na mesma máquina onde o servidor Web executa.

O modelo de rede de filas correspondente à figura 3.6 é mostrado na figura 3.7.

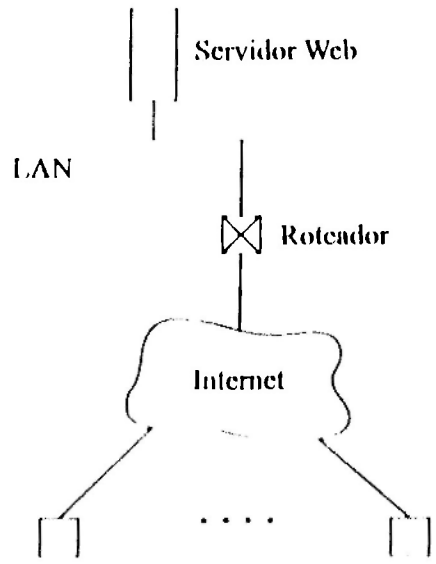


Figura 3.6: Ambiente com um único servidor Web

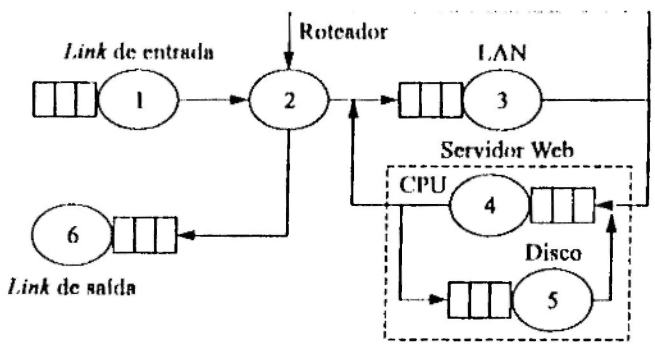


Figura 3.7: Modelo de redes de filas para o servidor da figura 3.6

Na figura, cada número representa um recurso (ou fila) Q. Q1: representa o *link* de entrada como um recurso independente de carga; Q2: representa o roteador como um recurso de retardo, devido à sua pequena latência, comparada a outros retardos envolvidos; Q3: representa a LAN como um recurso independente de carga; Q4: representa CPU do servidor como um recurso independente da carga; Q5: representa o disco do servidor como um recurso independente de carga; Q6: representa o *link* de saída, como um recurso independente de carga. Os recursos são aqui modelados como recursos de retardo ou recursos independentes da carga para simplificação. Aqui é suposto um servidor Web com disponibilidade pública na Internet, existindo assim uma população muito grande de clientes com acesso ao servidor. Dessa forma pode-se apenas caracterizar a taxa de chegada de requisições para tamanhos de documentos variados. O servidor Web pode ser modelado como um modelo aberto multiclasse de rede de filas. Outros exemplos de modelos do lado servidor podem ser achados em (Menasce & Almeida, 1998).

3.6 Considerações finais

Neste capítulo foram descritos e analisados diversos aspectos de modelagem e avaliação de desempenho, em particular aqueles pertinentes a servidores Web.

A técnica de modelagem tem sido amplamente utilizada tanto para avaliação de sistemas existentes, onde a validação de modelos é facilitada, como para os casos em que o sistema em estudo é ainda uma especulação de projeto.

O próximo capítulo apresentará um modelo de servidor Web que mostra as componentes internas de software, representando a comunicação completa entre cliente e servidor. Tal modelo será descrito em mais detalhe, já que foi tomado como base para esta tese de doutorado.

CAPÍTULO

4

Um Modelo de Servidor Web mostrando componentes internas de software

4.1 Introdução

Muitos trabalhos na literatura fornecem conclusões importantes quanto a aspectos específicos do desempenho dos servidores Web; quase todos, porém, não abordam um modelo para o desempenho da completa comunicação entre cliente e servidor, de um extremo a outro e levando em conta as componentes internas do software servidor. Alguns dos principais trabalhos relacionados a modelos de servidores Web são citados na próxima seção deste capítulo, com uma descrição sucinta de seu conteúdo. Na seqüência deste capítulo é descrito um modelo que foi estudado e tomado como base para o trabalho desenvolvido nesta tese.

4.2 Modelos para servidores Web: o estado da arte

Serão revisados nesta seção alguns trabalhos da literatura sobre modelos para servidores Web.

Em (Hu *et al.*, 1999) é estudado o comportamento do servidor Apache executando sobre o sistema operacional AIX da IBM. Nesse estudo é feita a identificação dos gargalos e são fornecidas informações detalhadas a respeito dos eventos ocorrendo no núcleo do sistema operacional. As cargas sintéticas são geradas pelas ferramentas SPECWeb e Webstone. O trabalho fornece dados sobre as porcentagens de tempo gastas executando em nível de usuário, em chamadas de sistema no núcleo e em manipulação de interrupções[1]. Com relação aos gargalos, a con-

[1]Em média, 20 a 25% em código de usuário, 35 a 50% em chamadas de sistema no núcleo e 25 a 40% em manipulação de interrupções

clusão no trabalho é que para sistemas com memória principal (RAM) de pequeno tamanho, o desempenho é limitado pela largura de banda do disco. Para sistemas com memória principal de tamanho razoavelmente grande, os gargalos são os softwares que processam os protocolos TCP/IP e o manipulador de interrupções da rede. O comportamento é semelhante para os sistemas uniprocessador e SMP(Symetric Multiprocessor). Após identificar.gargalos, são propostas e implementadas diversas técnicas, com as quais se consegue um aumento de cerca de 60% no desempenho do servidor, conforme relatado.

(Hu *et al.*, 1997a)[2] e (Hu *et al.*, 1997b)[3] são trabalhos que estudam diversos servidores Web, especificamente, Apache, Roxen, Jigsaw, Netscape Enterprise, PHTTPD e Zeus, executando sobre redes ATM de alta velocidade em plataformas UNIX. O autor estuda o desempenho relativo entre esses servidores e identifica também os gargalos em questão. A conclusão no trabalho é que, quando as sobrecargas relativas à rede e I/O de disco são diminuídas para fatores constantes desprezíveis (por exemplo, através de caches na memória), os principais fatores determinantes no desempenho são as estratégias de concorrência e de despacho de eventos usadas no servidor Web; além disso, verifica-se que nenhuma dessas estratégias funciona de modo ótimo para todas condições locais e tipos de tráfego. São descritas otimizações para desenvolver um servidor de alto desempenho, e conclui-se que, para prover um desempenho otimizado, os servidores Web devem ser adaptativos, escolhendo diferentes mecanismos para manipular requisições de arquivos grandes e pequenos distintamente. As características consideradas são portabilidade e flexibilidade. O servidor proposto pode se autoconfigurar em tempo de execução ou instalação para usar a estratégia de concorrência mais eficiente, para uma dada plataforma de sistema operacional; pode ser também configurado para dar suporte a múltiplos protocolos de transporte.

(Almeida *et al.*, 1996) descrevem a concepção e implementação de uma ferramenta para gerar uma carga sintética. A carga de trabalho utiliza uma distribuição de tamanho de arquivos de cauda pesada para capturar o fato de que o servidor Web deve manipular concorrentemente algumas requisições para grandes arquivos (por exemplo, áudio e vídeo) e um grande número de requisições para documentos pequenos contendo texto ou imagens. Nesse estudo é verificado que, em um servidor Web saturado por requisições, 90% do tempo para manipular essas requisições HTTP é gasto no kernel. Manter as conexões abertas, conforme requerido pelo protocolo TCP, provoca um aumento de fator dois no tempo decorrido para atender uma requisição HTTP. Os resultados do estudo mostram o importante papel da implementação do sistema operacional e do protocolo de rede para determinação do desempenho.

Em (Banga & Druschel, 1997), inicialmente se descreve a dinâmica de um servidor Web, com uma implementação TCP/IP, executando em uma máquina Unix; na seqüência são examinados os problemas para se medir a capacidade de um servidor Web usando carga sintética. Os

[2]relatório técnico
[3]artigo publicado

autores propõem e avaliam um método de geração de carga buscando contornar os problemas em *benchmarks* existentes; analisam também os efeitos de sobrecarga de requisições no desempenho do servidor. Quanto a esse comportamento do servidor sob excesso de carga, verifica-se que o *throughput* diminui com o aumento da taxa de requisições na sobrecarga, devido a recursos da CPU gastos no processamento de protocolos para requisições que chegam (pacotes SYN), que são finalmente descartados, devido à fila cheia em *accept*.

O trabalho apresentado em (Banga & Mogul, 1998) estuda a arquitetura dirigida por eventos de servidores Web, juntamente com as razões para um baixo desempenho de tais servidores. Descobre-se que os retardos típicos em WANs fazem com que servidores ocupados gerenciem um grande número de conexões simultaneamente. É mostrada a implementação de uma outra versão (que apresenta crescimento em escala) para a chamada de sistema *select* e para o algoritmo de alocação dos descritores de arquivos, levando a cerca de 60% de melhoria no *throughput* de servidores Web e *proxy* analisados, bem como aumentando a capacidade de crescimento em escala do sistema.

(Wallace & Jr., 1997) apresentam um estudo que mostra algumas formas simples para instrumentar um servidor Web, usando recursos e/ou mecanismos existentes, já implementados em sistemas de gerência da rede ou no sistema operacional, para obter estatísticas. Esses dados coletados e analisados, são usados em alguns modelos simples e genéricos de dimensionamento para componentes do servidor Web, descritos pelos autores. Os modelos são centrados em 4 gargalos comumente considerados: rede, memória, disco e CPU (o *throughput* de rede pode ser descartado já que isso não é, em geral, uma preocupação do servidor Web em si, pois a largura de banda de rede pode ser aumentada, configurando-se o servidor Web com mais (ou mais rápidas) placas de rede.

(Barford & Crovella, 1999) comparam o desempenho entre as versões 1.0 e 1.1 do protocolo HTTP, em termos de *throughput* do servidor e latência de transferência no cliente. Examina como os gargalos na rede, CPU e sistema de disco afetam os desempenhos relativos das duas versões entre si. As demandas de rede sob o HTTP 1.1 são um tanto menores que sob o 1.0. Essas diferenças são quantificadas em termos de pacotes transferidos, tamanho de janela de congestionamento do servidor e bytes de dados por pacote. No trabalho é mostrado que, se a CPU é o gargalo, há pouca diferença no desempenho entre o 1.0 e o 1.1. Quando o disco é o gargalo, usar o 1.1 pode ser (surpreendentemente) pior. Baseado em suas observações, os autores propõem uma política de gerência de conexões para aumentar o *throughput*, baixar a latência e manter baixo o tráfego de rede quando o gargalo está no sistema de disco.

(Liu *et al.*, 2001) propõem um modelo de tráfego no nível de sessão, o qual descreve quando os usuários chegam e como eles fazem solicitações para o servidor. Os autores desenvolvem um *benchmark* (para servidores Web) e noticiam os resultados obtidos com seu uso, juntamente com as análises de resultados pertinentes, relativos ao servidor Apache. É feita a comparação quanto ao desempenho das versões 1.0 e 1.1. Quando o 1.1 é usado, deveria ser

usado com *pipeline* de requisições do cliente. O trabalho tira conclusões dos lados cliente e servidor. No lado servidor, conclui-se, por exemplo, que o servidor deveria configurar valores de *timeout* pequenos para conexões persistentes, se um mecanismo de controle de temporização fixo for usado (como no Apache) ou se um mecanismo de controle dinâmico for usado e a carga de trabalho medida for elevada.

(Kant, 1999) apresenta uma metodologia para determinar requisitos de largura de banda para diversos componentes de hardware de um servidor Web, supondo uma arquitetura de SMP tradicional para o servidor. O trabalho propõe fórmulas para demandas de largura de banda para memória, barramento de dados do processador, adaptadores de rede e de disco, entre outros componentes de hardware considerados. Os resultados apontam algumas conclusões gerais em relação a cargas Web: por exemplo, os esforços para diminuir a sobrecarga relativa ao processamento TCP/IP podem requerer um espaço maior na largura de banda do subsistema de I/O do que no subsistema processador-memória.

(Kant & Sundaram, 2000) descrevem um modelo de redes de filas para um sistema SMP executando uma carga sintética, como em (Kant, 1999). O modelo leva em conta detalhes arquiteturais do servidor Web em termos de hierarquia de cache multinível e barramento do processador entre outros; é baseado em medidas detalhadas a partir de um sistema *baseline* e algumas de suas variantes. O modelo opera no nível de transações Web, e não modela explicitamente o núcleo da CPU e a hierarquia de cache. Contudo, prevê o impacto no desempenho de características de baixo nível, como por exemplo número e velocidades de processadores, tamanhos e latências de cache, latências de memória, e pré-busca (*prefetching*) de setores entre outros. A solução *default* do modelo é por simulação, dada a quantidade de características com dificuldade para solução analítica. O modelo apresentado em (Kant, 1999) e (Kant & Sundaram, 2000) é dirigido para o hardware e considera os principais componentes de desempenho internos do hardware do servidor.

Além destes dois últimos trabalhos citados, apenas alguns poucos na literatura têm como foco a modelagem dos servidores Web visando a avaliação de desempenho.

O primeiro modelo de servidor Web publicado foi proposto por (Slothouber, 1995) e modela o servidor Web através de uma rede de filas aberta. O modelo ignora detalhes essenciais de baixo nível dos protocolos TCP/IP e HTTP, embora eles tenham forte influência no desempenho do servidor; além disso, considera o servidor como uma caixa preta.

(Heidemann *et al.*, 1997) apresentam modelos analíticos para a interação do HTTP com vários protocolos de transporte tais como TCP, T-TCP e UDP, incluindo o impacto de algoritmos do tipo *slow start*. Nesse trabalho é apresentado um modelo analítico para cada um desses protocolos, sendo o modelo usado para avaliar a sobrecarga de rede conduzindo tráfego HTTP, usando características de rede variadas. O modelo é validado comparando resultados com traços de pacotes de redes medidos com dois protocolos (HTTP e HTTP persistente) em redes locais

e de grande distância. Mostra-se que o modelo tem precisão de 5% quanto ao desempenho medido para WANs, mas pode subestimar a latência quando a largura de banda é elevada e o retardo baixo. O modelo é usado para comparar os custos de estabelecimento de conexão desses protocolos, limitando o possível aumento de desempenho. Esses custos são avaliados para uma gama de características de rede, concluindo-se que as otimizações de *setup* são relativamente sem importância para os usuários correntes (artigo escrito em 1996) de modem, ISDN e LANs, mas podem prover aumento de desempenho de moderado a substancial em WANs de alta velocidade. O modelo é também usado para prever desempenho relativo a futuras características de redes.

(Dilley *et al.*, 1998) apresentam um modelo de alto nível em camadas de um servidor HTTP, e constrói uma estrutura ferramental para coletar e analisar dados empíricos. Os modelos de filas em camadas (LQMs:*Layered Queuing Models*) foram propostos para estudar sistemas de aplicações distribuídos (Rolia & Sevcik, 1995) (Woodside *et al.*, 1995), sendo estendidos e aplicados em (Dilley *et al.*, 1998) para abordar servidores Web. Os LQMs são extensões dos modelos de redes de filas (QNMs) (Lazowska *et al.*, 1984) que consideram disputa por processos ou recursos físicos como processadores e discos, ou seja, disputa por recursos de hardware e software.

Um LQM pode ser usado para responder ao mesmo tipo de perguntas de planejamento de capacidade, como um QNM, mas também estima os retardos do cliente em filas nos servidores. Quando o número de clientes é grande, esses retardos do cliente no servidor podem aumentar muito mais rapidamente do que os tempos de resposta do servidor e são uma medida melhor da qualidade de serviço do usuário final (incluindo o tempo de resposta do cliente).

As técnicas do Método de Camadas (MOL) (Rolia & Sevcik, 1995) e do SRVN (Stochastic Rendez-Vous Network) (Woodside *et al.*, 1995) foram propostas como técnicas de avaliação de desempenho que estimam o comportamento de LQMs, sendo ambas baseadas em MVA (Análise do Valor Médio). As estimativas de desempenho para cada qual dos QNMs são achadas e usadas como parâmetros de entrada para outros QNMs. O propósito do MOL é achar um ponto fixo onde os valores previstos para os tempos médios de resposta e utilização dos processos sejam consistentes no que diz respeito a todos os submodelos. Intuitivamente, esse é o ponto em que os tempos de ociosidade e a utilização de processos previstos estão equilibrados, tal que cada processo no modelo tem o mesmo ***throughput***, independentemente de ser considerado um cliente ou um servidor no QNM.

Embora esses estudos forneçam observações importantes sobre o desempenho dos servidores Web, bem como modelos de partes do sistema servidor, nenhum deles fornece um modelo para o desempenho da comunicação completa entre cliente e servidor, mostrando componentes internas do servidor.

Mais recentemente, (Andersson *et al.*, 2003) apresentaram um modelo de filas para um servidor Web que usa tráfego de chegada em rajadas. Supõe-se que a chegada de requisições

Tabela 4.1: Comparação entre algumas Características dos Modelos de Desempenho de Servidores Web apresentados por (van der Mei *et al.*, 2001) e (Kant & Sundaram, 2000)

Características	Mei et al	Kant e Sundaram
OBJETIVOS (parâmetros de saída)	*Throughput* e Tempo de Resposta do Servidor	*Bandwidth* para componentes do hardware servidor
CARGA DE TRABALHO	Transações Web	SpecWeb; proxy (via *trace*)
ARQUITETURA DE HARDWARE	Geral	SMP, *Clusters*
REPRESENTAÇAO ABSTRATA	Camadas de Redes de Filas	Fluxo sobre o hardware SMP; Redes de Filas
MÉTODO DE SOLUÇÃO	Simulação, Analítica	Simulação

HTTP seja um Processo de Poisson Modulado Markoviano (MMPP). Tal modelo considera contudo o servidor Web como uma caixa "preta".

Outro trabalho recente em modelagem de servidores Web é (Cao *et al.*, 2003), onde o processo de chegada de requisições HTTP é modelado pela distribuição de Poisson e a disciplina de serviço é "*processor sharing*", ou seja, o modelo de servidor Web consiste de um nó compartilhado por processadores com uma fila ligada ao nó. O no. total de tarefas (que podem ser processadas ao mesmo tempo) no sistema é limitada a K. A distribuição do tempo de serviço é arbitrária (G).

Em (van der Mei *et al.*, 2001) foi proposto um modelo fim-a-fim para servidores Web, englobando os impactos de características de carga de trabalho do cliente, configuração de hardware e software, protocolos de comunicação e topologias de interconexão; esse último trabalho será abordado com mais detalhe na próxima seção. A Tabela 4.1 apresenta uma comparação das características dos modelos apresentados em (van der Mei *et al.*, 2001) e (Kant & Sundaram, 2000).

4.3 Um modelo de servidor Web mostrando componentes internas

Esta seção apresenta um modelo para servidor Web, baseado em filas, representando a comunicação completa entre servidor Web e cliente, englobando os impactos de características de carga, configuração de hardware e software do servidor, protocolos de comunicação e topologias de interconexão; o modelo é descrito numa série de trabalhos por pesquisadores, principalmente dos laboratórios da ATT. O modelo básico inicial é apresentado em (van der Mei *et al.*, 2001), sendo solucionado por simulação. Uma solução analítica do mesmo modelo é apresentada em

(Reeser *et al.*, 1999). Nesses dois estudos são consideradas apenas requisições para documentos estáticos no modelo e conexões não persistentes. (van der Mei *et al.*, 1999), (Hariharan *et al.*, 2000) e (Ehrlich *et al.*, 2000) consideram também documentos dinâmicos bem como distintos tipos de geração desses documentos, em acréscimo ao modelo básico inicial. Os fluxos de transações no servidor Web são descritos por um modelo de filas em série, que consiste de 3 sub-modelos (ou fases), que são descritos a seguir.

4.3.1 Fase 1: estabelecimento de conexão TCP

A primeira fase representa o estabelecimento da conexão TCP. Antes que os dados possam ser transmitidos entre servidor e cliente, uma conexão TCP *two-way* (uma *socket* TCP) deve ser estabelecida. Nessa fase (subsistema) existe a fila de escuta TCP (TCP *Listen Queue*: TCP-LQ) que é atendida por um *daemon* servidor. Uma conexão TCP é estabelecida através do procedimento conhecido como *three way handshake*, que obedece aos seguintes passos:

- O cliente envia um pacote SYN de pedido de conexão ao servidor.
- Se há um lugar disponível na TCP-LQ, então a solicitação ocupa um lugar e o *daemon* servidor envia um *acknowledgement* (SYN-ACK) ao cliente; caso contrário, o pedido de conexão é rejeitado.
- Após receber o SYN-ACK, o cliente envia ao servidor um ACK juntamente com um pedido de transação (por exemplo, um GET). Após a chegada desse pedido, o lugar na fila TCP é liberado.

Logo após o estabelecimento da *socket* TCP, o *daemon* servidor envia o pedido de transação para o sub-sistema HTTP, que constitui a fase 2, a ser descrita. Ao término do processamento da transação, o servidor tipicamente envia uma mensagem de FIN para encerrar a *socket* TCP. A fase de estabelecimento de conexão pode ser modelada como um modelo bloqueante com os servidores sendo representados pelos lugares na fila TCP; os clientes são representados pelas requisições. O tempo de serviço na fase 1 é o tempo entre (1)a chegada do pedido de conexão na fila TCP e (2)o instante em que o pedido de transação (juntamente com o SYN-ACK) chega. Desse modo, um tempo de serviço corresponde ao tempo de viagem de ida e volta (*Round Trip Time* - RTT) pela rede, entre servidor e cliente. O tamanho da fila TCP é configurável.

4.3.2 Fase 2: processamento HTTP

Após o estabelecimento de conexão o pedido de transação está apto a ser filtrado e interpretado pelo sub-sistema HTTP, que consiste de uma fila de escuta HTTP (HTTP *Listen Queue*: HTTP-LQ) servida por um ou mais *daemons* HTTP. Segue-se a dinâmica do sub-sistema HTTP:

- Se uma *thread* HTTP está disponível, então a *thread* busca o arquivo solicitado (ou de um sistema de arquivos ou da memória cache) e coloca o arquivo em um *buffer* de I/O (se houver um disponível). A *thread* é então liberada para processar o próximo pedido de transação.

- Se todos *buffers* de I/O estiverem ocupados nesse instante, então a *thread* HTTP permanece ociosa até que um *buffer* fique disponível. Se houver mais de uma *thread* esperando por um *buffer*, alguma regra (específica da implementação) de atribuição é usada para determinar em que ordem os *buffers* de I/O liberados serão atribuídos às *threads* em espera.

- Se não há *thread* HTTP disponível (estão todas ocupadas), então o pedido de transação entra na HTTP-LQ (se houver lugar disponível), e espera até que lhe seja designada uma *thread* para manipular a requisição.

- Se a fila HTTP estiver cheia, o pedido é rejeitado, a conexão rompida, e o cliente recebe uma mensagem de conexão recusada.

O tamanho do arquivo pedido pode exceder o tamanho do *buffer* de I/O. Esse tamanho, configurável, é tipicamente regulado para que a maioria dos arquivos solicitados (por exemplo, 95 a 99%) caiba no *buffer*. Em caso contrário, o arquivo é dividido em um no. de partes, $P_1, P_2, \ldots\ldots, P_k$, cada qual preenchendo um *buffer*, exceto a última parte P_k. Todos segmentos de um dado arquivo devem usar o mesmo *buffer*, ou seja, não se permite que o servidor coloque as diferentes partes do mesmo arquivo em *buffers* distintos. Quando $k > 1$, após P_1 ser colocado no *buffer*, a *thread* responsável pela transação tem que permanecer ociosa (bloqueada) até que o *buffer* seja esvaziado, antes que possa colocar P_2 no *buffer*, e assim por diante. Exceção é feita para P_k (a parte final): logo que P_k é colocada no *buffer*, a *thread* está pronta a atender um outro pedido de transação, sem ter que esperar o esvaziamento de P_k. A Figura 4.1 mostra a colocação no *buffer* de um arquivo particionado em 3.

O sub-sistema HTTP pode ser modelado por um sistema bloqueante com um *buffer* finito, que representa a fila HTTP (HTTP-LQ) e um certo número de servidores, N_{HTTP}, que representam as *threads* HTTP. Os clientes representam os pedidos de transações. Se um servidor está disponível, então o cliente é atendido imediatamente. Caso contrário, o cliente entra na fila HTTP; se a fila estiver cheia, então o cliente é rejeitado. O tempo de serviço de um cliente, τ_{trans}, é o intervalo de tempo entre (1)o instante em que a *thread* começa a buscar o arquivo solicitado e (2)o instante em que todas partes do arquivo tenham sido colocadas no *buffer* de I/O. Esse tempo de serviço pode ser desmembrado nas seguintes partes:

$$\tau_{trans} = \tau_{fetch} + \tau_{wait} + \tau_{drain}$$

τ_{fetch} representa o tempo necessário para buscar o arquivo, τ_{wait} representa o tempo que o servidor tem que esperar para obter acesso a um *buffer* de I/O e τ_{drain} representa o tempo para

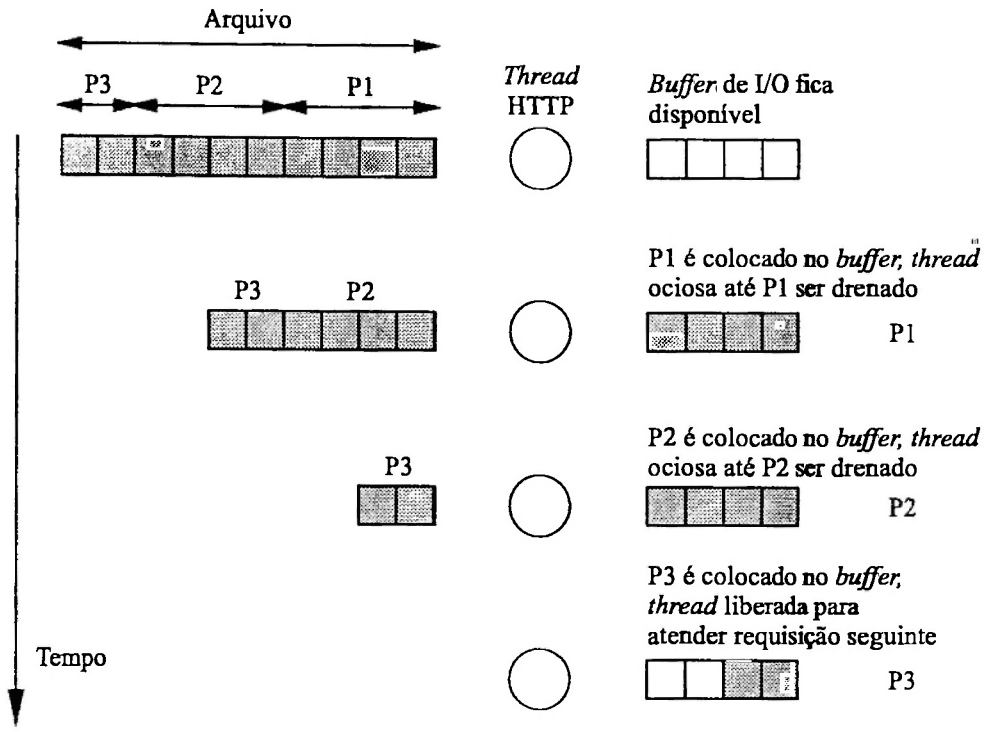


Figura 4.1: Particionamento de arquivo durante a fase de processamento HTTP

se colocar todo o arquivo (possivelmente em partes) em um *buffer* de I/O. Considerando que um arquivo pode vir a ser particionado, para envio, em k partes, $k = N_{file}$, é o número de *buffers* de I/O que o arquivo precisa. Denotando o tamanho do arquivo pedido por F e o do *buffer* de I/O por B_{IO}, tem-se que:

$$N_{file} = \lceil F/B_{IO} \rceil$$

Tem-se ainda que, $\tau_{drain} = \tau_{drain}^{(1)} + \cdots + \tau_{drain}^{(k-1)}$, onde $\tau_{drain}^{(i)}$ é o tempo necessário para colocar P_i em um *buffer* de I/O e drenar P_i $(i = 1, \ldots, k-1)$. Supõe-se como desprezível o tempo para se colocar uma parte do arquivo em um *buffer* de I/O, após esse *buffer* ter sido designado para a *thread* HTTP responsável. A Figura 4.2 mostra os componentes do tempo de serviço no subsistema HTTP para o caso de partição de um arquivo em 3, sendo colocadas essas partes no mesmo *buffer*.

Em geral, τ_{fetch} é uma variável aleatória independente, enquanto que as distribuições de probabilidade das variáveis τ_{wait}, τ_{drain} e N_{file}, e sua estrutura de correlação, são parâmetros de saída que em geral dependem do desempenho do sub-sistema de I/O. O tamanho do *buffer* (B_{IO}), o número de *buffers* de I/O (N_{IO}), o tamanho da fila HTTP (B_{HTTP}), e o número de *threads* são configuráveis.

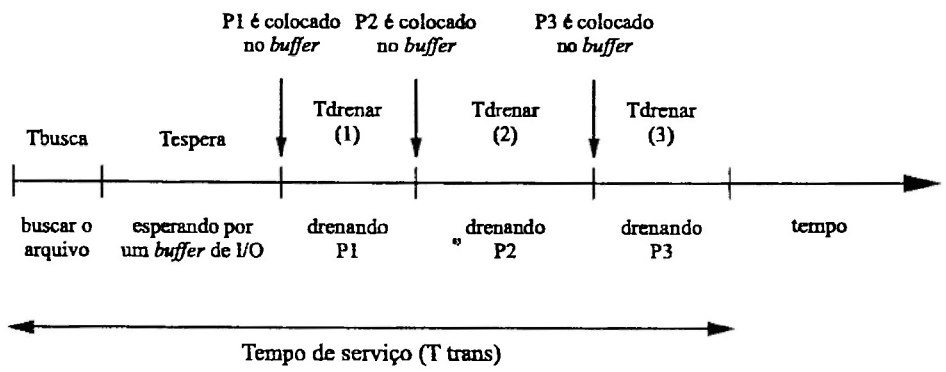


Figura 4.2: Componentes de tempo de serviço, na fase HTTP, para um arquivo particionado em três partes

4.3.3 Fase 3: Processamento de I/O

Os diferentes *buffers* de I/O são esvaziados através de uma conexão de rede comum, para a rede (por exemplo, a Internet ou uma intranet). O escalonamento do acesso para os diferentes *buffers* de saída de dados é feito pelo chamado controlador de I/O, que visita os *buffers* em alguma ordem (por exemplo, *round-robin*). O TCP-IP é controlado por um mecanismo de janela. A unidade de transmissão para a conexão de rede baseada em TCP/IP é o MSS (*Maximal Segment Size*),i.e., a maior quantidade de dados que o TCP enviará ao cliente em um segmento.

Por isso, os arquivos no *buffer* de I/O são (virtualmente) particionados em blocos de um MSS (exceto a última parte do arquivo). O mecanismo de janela implica em que um bloco de um arquivo residindo em um *buffer* de saída só pode ser transmitido se a janela TCP estiver aberta; isto é, se os blocos ainda podem ser transmitidos antes de se receber um ACK. Observe-se que a chegada de ACKs depende em geral da congestão na rede. Assim, a taxa em que cada um dos *buffers* de I/O pode esvaziar seu conteúdo é afetada pela congestão na rede.

A dinâmica do subsistema de I/O pode ser modelada com um modelo *polling* com um único servidor, com N_{IO} filas, cada qual de tamanho B_{IO}. O servidor representa o controlador de I/O e as filas representam os *buffers* de I/O. Os tempos de serviço representam o tempo para esvaziar (transmitir e receber o ACK) um bloco de arquivo.

O tempo total para esvaziar o conteúdo de um *buffer* de I/O pode ser expresso como:

$$\tau_{drainI/Obuffer} = \sum_k \tau_{I/Oblock}^{(k)}$$

sendo que $\tau_{I/Oblock}^{(k)}$ pode ser decomposto como:

$$\tau_{I/Oblock}^{(k)} = \tau_{RC}^{(k)} + \tau_{DB}^{(k)}$$

onde $\tau_{RC}^{(k)}$ denota o tempo até que o servidor visite o *buffer* de I/O em questão (esse tempo, chamado de tempo de ciclo residual (RC), é o tempo até que o controlador volte a visitar o *buffer* em questão, fazendo o *polling* nos *buffers*. $\tau_{DB}^{(k)}$ denota o tempo para esvaziar (drenar) o bloco do arquivo e pode ser decomposto como:

$$\tau_{DB}^{(k)} = \tau_{link}^{(k)} + \tau_{inet}^{(k)} + \tau_{client}^{(k)}$$

onde $\tau_{link}^{(k)}$ é o tempo para colocar o bloco no *link* de saída (conexão com uma LAN), $\tau_{inet}^{(k)}$ é o tempo para enviar o bloco e receber um ACK, e $\tau_{client}^{(k)}$ é o tempo requerido pela interface do cliente (e.g. modem ou placa de LAN) para ler o bloco. A figura 4.3 ilustra a dinâmica do modelo para o caso em que o *buffer* de I/O contém 3 blocos.

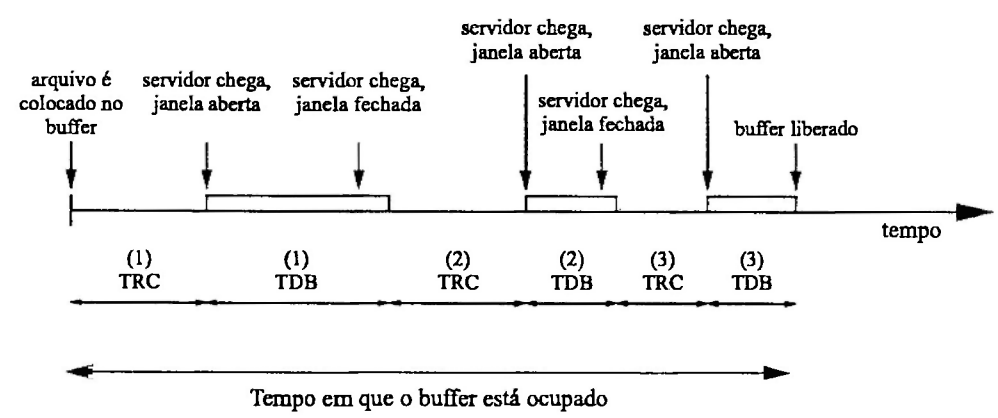


Figura 4.3: Componentes de tempo na fase de processamento de I/O, para o caso do *buffer* conter três blocos

O tempo para colocar um bloco de arquivo de tamanho $B^{(k)}$ em uma conexão de rede é dado pela seguinte expressão:

$$\tau_{link}^{(k)} = \frac{B^{(k)}}{\mu_{NC}}$$

onde μ_{NC} é a velocidade de linha da conexão à rede. O tempo requerido pelo cliente para ler um bloco de arquivo de tamanho $B^{(k)}$ é dado por

$$\tau_{client}^{(k)} = \frac{B^{(k)}}{\mu_{client}}$$

onde μ_{client} é a taxa com que o cliente lê os dados que chegam (por exemplo, 56Kbits/seg para um determinado modem). $\tau_{inet}^{(k)}$ é uma variável aleatória com a mesma distribuição de um RTT de rede entre o servidor e o cliente.

4.3.4 Requisição de documentos dinâmicos

O modelo apresentado foi ampliado para representar também o atendimento de requisições para documentos gerados dinamicamente (Hariharan *et al.*, 2000). Para tanto, se na fase HTTP o arquivo requer processamento de *script*, o pedido é enviado para um sub-sistema SE (*Script Engine*), servido por *threads SE*. A dinâmica da SE depende em geral da implementação do servidor Web e do SO (Sistema Operacional); o sub-sistema SE consiste basicamente de uma fila de escuta SE (SE-LQ) e um conjunto de *threads* dedicadas para interpretar as declarações do

script e executar código embutido (e.g., C++ e Java). Durante a execução, pode ser necessário uma comunicação com um servidor remoto *backend* (por exemplo, para fazer consulta a uma base de dados), com o estabelecimento (e posterior ruptura) de uma conexão TCP com o servidor remoto. Segue-se uma descrição do sub-sistema SE:

- Se uma *thread* SE está disponível, então a *thread* executa o código embutido.
  - Se a implementação HTTP-SE opera no modo bloqueante (ou seja, a *thread* HTTP bloqueia esperando uma resposta da *thread* SE (como no caso de JSP), então a *thread* SE encaminha a transação de volta à *thread* HTTP para manipulação posterior.
  - Caso contrário, se a implementação HTTP-SE opera em modo não bloqueante, isto é, a *thread* HTTP é liberada logo que a requisição é encaminhada ao subsistema SE (como no caso do IIS usando ASP), então a *thread* SE encaminha a transação diretamente para o subsistema de I/O. Se todos *buffers* de I/O estiverem ocupados nesse instante, então a *thread* SE permanece ociosa até que um *buffer* fique disponível.
- Se não há *thread* disponível, então o pedido de transação entra na SE-LQ (se houver lugar nessa fila de escuta), e espera até que uma *thread* seja designada para manipular o pedido.
- Se a SE-LQ estiver cheia, então o pedido de transação é recusado, e o cliente recebe uma mensagem de conexão recusada, sendo a conexão TCP rompida.

Observe-se que a implementação IIS/ASP opera no modo bloqueante, enquanto que a maioria das outras implementações (por exemplo, JSP) operam tipicamente no modo bloqueante. A Figura 4.4 mostra o modelo descrito, incluindo o caso de requisição dinâmica.

4.4 Considerações finais

Este capítulo descreveu o único modelo encontrado na literatura que apresenta o software servidor Web explicitando suas componentes internas. Esse modelo é tomado como base para o modelo proposto nesta tese.

Porém, conforme se pode verificar em (van der Mei *et al.*, 2001), o modelo base considera que: (1)uma conexão é estabelecida e rompida para envio de cada documento Web para o cliente (conexões não persistentes); (2)além disso, no modelo base se supõe que a carga de chegada de documentos ao servidor segue a distribuição de Poisson, para facilidade de modelagem. Entretanto tal suposição não é corroborada por diversos trabalhos da literatura, alguns entre os quais foram citados na revisão do estado da arte feita no início deste capítulo.

O próximo capítulo abordará novamente esses dois aspectos citados e as melhorias propostas no modelo base, referentes aos mesmos.

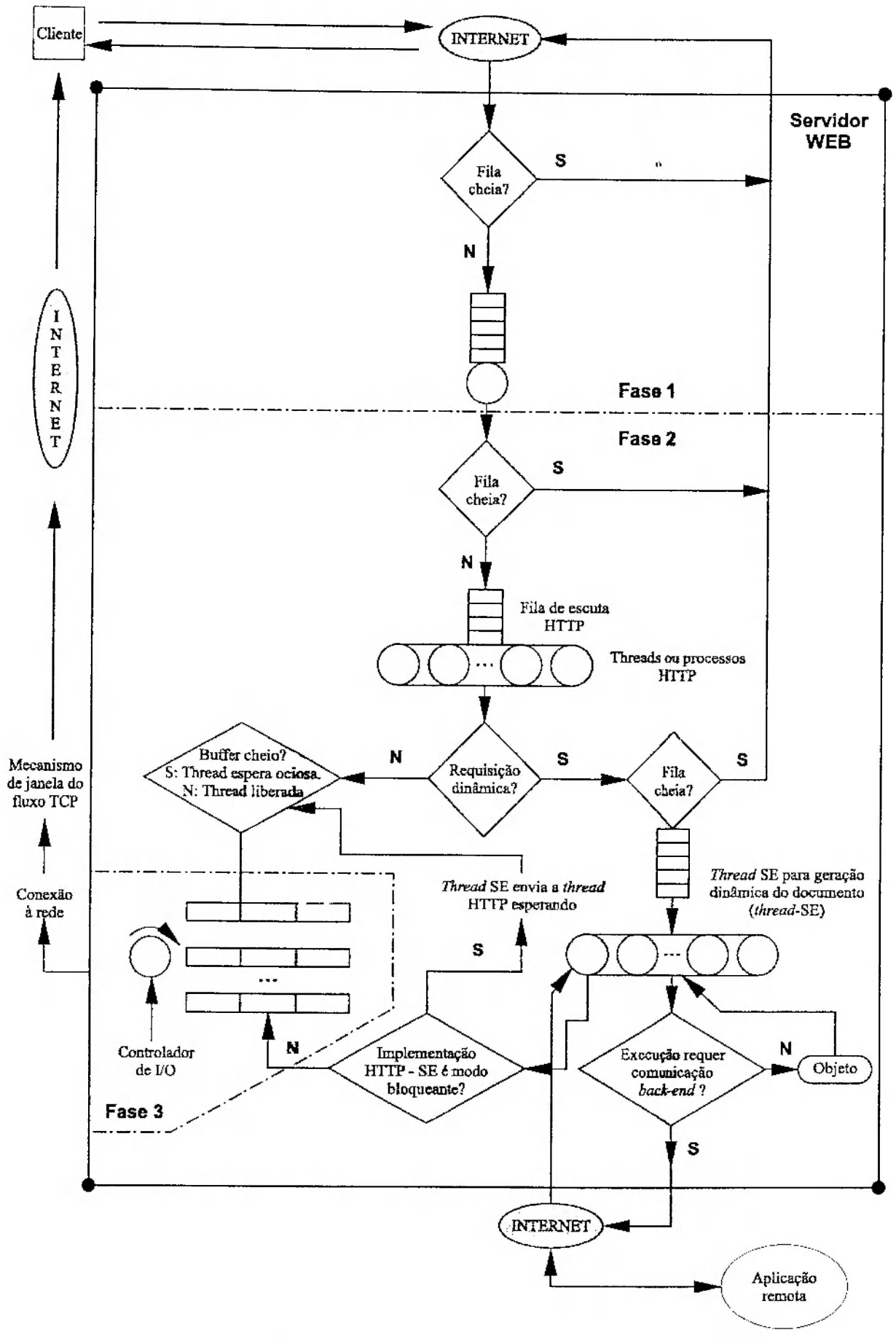


Figura 4.4: Modelo de servidor Web

CAPÍTULO

5

Melhorias propostas para o modelo base

5.1 Introdução

Este capítulo aborda com mais detalhes as duas contribuições incorporadas nesta tese ao modelo base descrito no capítulo 4. A primeira contribuição (melhoria) no modelo base diz respeito à extensão aqui proposta com relação a conexões persistentes. A segunda contribuição diz respeito à modificação da carga de chegada do modelo base, originalmente orientada a documentos, e que se propõe nesta tese que seja orientada a sessões.

5.2 Extensão do modelo para conexões persistentes

No modelo base citado no presente trabalho, uma conexão TCP é estabelecida para cada transferência de arquivo armazenado no servidor, conforme a versão 1.0 do protocolo HTTP. Isso ocasiona sobrecarga de tráfego na rede e também, muitas vezes, sobrecarga no servidor, conforme já foi observado em diversos trabalhos da literatura, alguns deles citados no capítulo 4. Sendo assim, é proposto aqui que o modelo seja estendido para o caso de conexões persistentes, que é o padrão da versão 1.1 do mesmo protocolo, amplamente utilizada no presente.

No caso de conexões persistentes, uma conexão TCP é estabelecida no primeiro *click* do cliente junto ao sítio onde o servidor está executando, no início de uma sessão do usuário. Essa conexão é mantida até que todos os documentos solicitados sejam enviados, para que o *browser* possa montar a página, ou até que ocorra algum ***timeout*** (do cliente ou do servidor), ou outro evento, por exemplo, de ordem física ou resultante de configuração do servidor, que provoque o rompimento da conexão.

É interessante observar que a extensão do modelo para conexões persistentes foi sugerida

como trabalho futuro em (van der Mei *et al.*, 2001), já que o modelo ali apresentado havia sido concebido mais de dois anos antes, quando o padrão do protocolo HTTP 1.0 era o uso de conexões não persistentes.

5.3 Modificação da Carga de Chegada

Além da extensão citada na seção anterior, é também proposto nesta tese uma modificação relativa ao tipo de carga de chegada considerada no modelo base descrito, onde uma série de suposições foram feitas. Em particular, a carga de chegada (o processo de chegada de pedidos de documentos ao servidor Web) no modelo original é modelada como um processo de Poisson. A literatura, contudo, não é totalmente conclusiva com relação à natureza do processo de chegada de pedidos de documentos para um servidor Web. Vários resultados porém, como por exemplo, (Liu *et al.*, 2001) e (Paxson & Floyd, 1995), mostram que as chegadas de *sessões* iniciadas por usuários a um (mesmo) servidor Web podem ser bem modeladas por um processo de Poisson, ao passo que fluxos de tráfego de nível mais baixo (documentos ou pacotes) não são bem representados por um processo de Poisson.

Na verdade, diversos estudos de medição de tráfego na literatura revelaram que os fluxos de tráfego em nível de pacotes em redes de computadores podem ter uma estrutura de auto-correlação complexa, exibindo dependência ao longo prazo, conforme (Leland *et al.*, 1994), (Paxson & Floyd, 1995) e (Crovella & Bestavros, 1996). Do mesmo modo, a caracterização da chegada dos pedidos de documentos ao servidor Web tem indicado uma distribuição híbrida, que considera o corpo e a cauda (pesada)da distribuição. Ao adotar uma carga de chegada de documentos/unidade de tempo, a caracterização correta dessa carga pode se tornar complicada.

Propõe-se aqui que essa carga de trabalho (carga de chegada no modelo) seja uma carga de sessões, a qual, conforme será visto neste capítulo: (i)pode ser mais facilmente modelada, a partir de características do usuário; (ii)a carga de chegada orientada a sessões, constitui sempre um processo de renovação (*renewal*), sendo muitas vezes um processo de Poisson.

Um processo de Poisson pode ser caracterizado como um processo de contagem para o qual os tempos entre sucessivos eventos são variáveis aleatórias independentes, identicamente distribuídas, com distribuição exponencial. Um processo de renovação nada mais é do que uma generalização do processo de Poisson, ou seja, onde os tempos entre sucessivos eventos não mais obedecem a uma distribuição exponencial, podendo ter uma distribuição qualquer.

Definição (Allen, 1990): Seja $\{N(t), t \geq 0\}$ um processo de contagem e X_1 o tempo de ocorrência do primeiro evento. Seja X_n a variável aleatória que mede o tempo entre o evento de ordem $n-1$ e o enésimo evento desse processo para $n > 2$. Se a seqüência de variáveis aleatórias não negativas $\{X_n, n \geq 1\}$ é independente e identicamente distribuída então $\{N(t), t \geq 0\}$ é um processo de *renovação*.

Quando um evento contado por $N(t)$ ocorre, diz-se que uma renovação aconteceu. A soma $W_0 = 0, W_n = X_1 + X_2 + ... + X_n, n \geq 1$ é denominada de tempo de espera até a enésima renovação. O processo de contagem $\{N_t, t \geq 0\}$ e o processo $\{W_n, n \geq 0\}$ são conjuntamente chamados de processo de renovação.

Um exemplo comumente usado para descrever um processo de renovação é o da *lâmpada*. Uma lâmpada é instalada no instante $W_0 = 0$. Quando ela queima, no instante $W_1 = X_1$, ela é imediatamente substituída por uma lâmpada nova, que queima no instante $W_2 = X_1 + X_2$. Esse processo continua indefinidamente; quando cada lâmpada queima é substituída por uma perfeita. Supõe-se que os tempos de vida das sucessivas lâmpadas $\{X_n, n \geq 1\}$ sejam independentes e tenham a mesma distribuição. $N(t)$ é o número de substituições de lâmpada que ocorrem até o instante t e $\{N(t), t \geq 0\}$ é um processo de renovação.

Os eventos de renovação, também chamados de eventos recorrentes, correspondem nesse caso ao início de sessões de usuários em um mesmo servidor Web. Em outros casos práticos, os eventos de renovação correspondem a falha (e troca instantânea) de um componente em um ambiente com disponibilidade sem limites de outros componentes para substituição. Ou então de reparos nos componentes a serem efetuados, em lugar das substituições.

Em (Liu *et al.*, 2001) foi proposto um modelo estocástico de tráfego HTTP no nível de sessão, onde se consideram requisições HTTP ao *mesmo* servidor Web (ao mesmo sítio). O interesse é no comportamento típico de usuários através do hipertexto de um servidor Web, ou seja, quando eles chegam e como as requisições são geradas. Isso pode ser bem descrito pela noção de sessão: uma seqüência de *clicks* de um usuário nos *hyperlinks* do mesmo servidor.

Os instantes da chegada correspondem aos instantes de início de sessões. Cada chegada é associada com as seguintes variáveis:

- O número de *clicks* durante a sessão;
- Os tempos ociosos entre os *clicks* (ou tempos de pensar dos usuários), ou seja, os tempos decorridos entre o término da transferência da página requisitada previamente e o começo da transferência da página corrente.
- Os custos de transferência e de CPU dos *clicks* sucessivos.

A figura 5.1 mostra uma sessão de usuário, onde τ_j denota o tempo para se recuperar páginas Web (incluindo documentos embutidos), e γ_j representa os tempos entre chegadas de *clicks*.

Como se verifica, a sessão é constituída por várias páginas Web solicitadas pelo usuário, sendo cada página constituída por um conjunto de documentos enviados pelo servidor ao *browser*, que montará a página Web a ser visualizada pelo usuário.

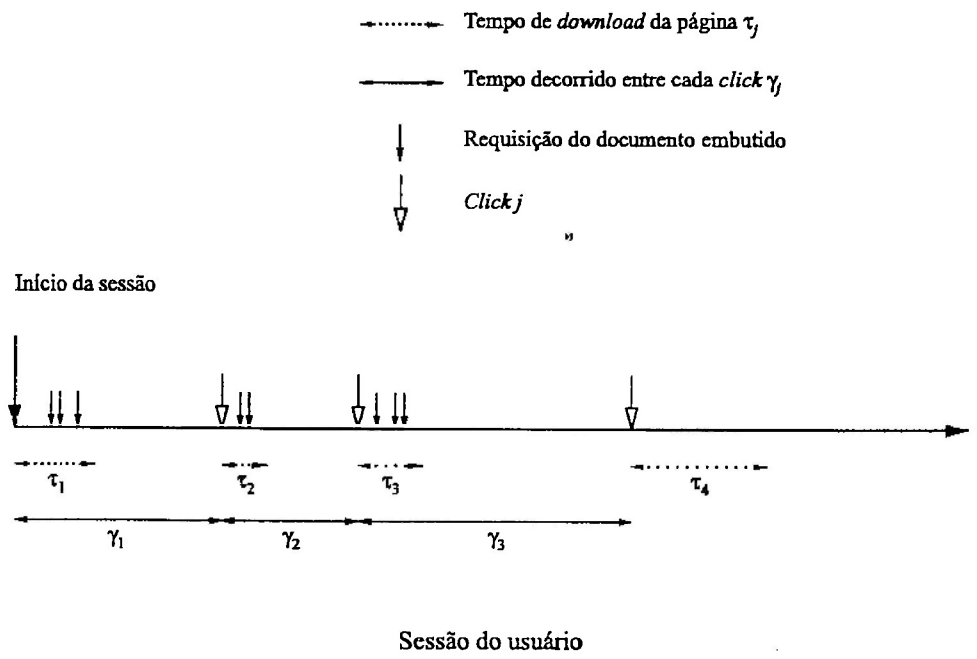


Figura 5.1: Sessão do usuário

As variáveis usadas no modelo de sessões citado, como o número de *clicks* em uma sessão e os tempos ociosos entre *clicks*, são fáceis de interpretar e relativamente fáceis de se medir. Verifica-se também em (Liu *et al.*, 2001) que as chegadas de sessões são bem descritas por um processo de renovação , e em muitos casos são simplesmente um processo de Poisson. Essa propriedade permite um modo mais simples de se modelar a carga de chegada bem como de se parametrizar a intensidade de tráfego.

Isso está em contraste com as chegadas de requisições HTTP (de documentos) que usualmente formam um processo de dependência de longo prazo, sendo, dessa forma, de análise mais difícil (Liu *et al.*, 2001).

Contudo, embora se possa considerar a carga de chegada no modelo como de sessões por unidade de tempo, a carga de trabalho conforme efetivamente vista pelo servidor, continua sendo de documentos e essa é a que deve ser considerada internamente no modelo desta tese.

Adotando a carga de chegada orientada a sessões, deve-se transformar essa carga de chegada de sessões por segundo, para uma carga de trabalho de documentos por segundo, para aplicação no modelo. Isso será possível, desde que se disponha das distribuições do número de páginas por sessão, do número de documentos por página Web e aquela relativa ao tamanho dos documentos requisitados.

A modificação da carga de chegada facilita e melhora a modelagem em vários aspectos. Por exemplo, a chegada de sessões será muitas vezes caracterizada por um processo de Poisson, sendo em *todos* os casos um processo de renovação; além disso, as distribuições de páginas por sessão e documentos por página poderão ser conhecidas, a partir de registros (*logs*) feitos no *site*, cujo servidor se pretende modelar. A modelagem da carga orientada a sessões pode ser feita

considerando-se as páginas mais acessadas, tempo de permanência em cada página bem como o tamanho da página (soma dos tamanhos dos documentos que constituem a página).

5.4 Considerações finais

Este capítulo apresentou as principais contribuições desta tese, que são melhorias a serem incorporadas no novo modelo proposto, a partir do modelo base, constituindo-se principalmente de uma extensão e uma modificação a serem feitas no modelo base descrito no capítulo 4. Em particular, a modificação proposta quanto à carga de chegada tomou como importante referência o trabalho de (Liu *et al.*, 2001), entre outros de seu grupo de pesquisa.

Observe-se que em (Liu *et al.*, 2001) o modelo de tráfego apresentado (baseado em sessões) é proposto como carga de chegada para um servidor Web, porém esse servidor é visto como uma caixa preta, distintamente do considerado na presente tese.

Com relação a desvantagens que o novo modelo pode apresentar, pode-se citar o fato de que para usar sessões como carga de entrada em um modelo que requer processamento de documentos internamente, sempre há a preocupação da transformação da carga de sessões na carga equivalente de documentos. Isso pode ser considerado como acrescentando complexidade ao modelo, ou seja, representando uma desvantagem.

O próximo capítulo apresentará a solução e validação do novo modelo, contendo as melhorias identificadas neste capítulo.

CAPÍTULO

6

Solução e Validação do novo modelo

6.1 Introdução

O modelo básico original é um software proprietário, com seus direitos pertencentes à ATT (*American Telephone and Telegraph*). Dessa forma, não foi possível ter-se acesso a todos detalhes do modelo base, apesar de consultas realizadas com seu principal idealizador, Prof. Dr. Robert van der Mei; optou-se por simplificar o modelo em alguns aspectos.

Considerou-se na modelagem presente apenas documentos estáticos, embora a extensão para documentos dinâmicos haja sido apresentada no modelo base. Isso porque, por exemplo, no caso de documentos dinâmicos o servidor é muito mais exigido em termos de CPU e os processos dinâmicos subjacentes (CGI, requisições a base de dados, Java Servlet, etc.) podem representar um papel preponderante no desempenho do servidor Web (Liu *et al.*, 2001).

Presentemente resumiu-se a parte de envio dos documentos pela rede para o cliente (dentro da fase 3 do modelo base), supondo um tempo de ida e volta total entre cliente e servidor (envio pelo servidor e *acknowledgements* do cliente) modelado em relação ao tamanho total do documento a ser enviado, a latência e a largura de banda relativas ao tipo de rede em questão (local ou de longa distância). Essa parte pode sofrer maior detalhamento e expansão na modelagem, sempre que isso se fizer necessário.

Ainda em relação à fase 3, foi feita outra simplificação, no que diz respeito ao funcionamento do *buffer* de I/O, conforme representado no programa de simulação, o que será abordado na próxima seção que contém exemplos de parâmetros para essa fase.

A calibragem do modelo é feita usando diversos parâmetros, a serem usados no simulador, como os que são apresentados na próxima subseção, com fases que se referem ao modelo original.

6.2 Parâmetros usados nas componentes de processamento relativas às três fases do modelo base

6.2.1 Fase 1

Tamanho da fila de escuta TCP (esse tamanho equivale ao número de servidores, N_{TCP} do centro de serviço considerado no modelo para a fase 1, no estabelecimento de conexão TCP).

$N_{TCP} = 1024$. Esse valor pode ser verificado no *kernel* do sistema operacional, referindo-se à implementação do protocolo TCP.

6.2.2 Fase 2

Aqui se encontram incluídos os parâmetros relativos à transformação de sessões para documentos, conforme se acha representado no simulador, além dos parâmetros relativos à fase de processamento HTTP propriamente dita.

Número de páginas por sessão (exemplo de valores que podem ser considerados): 1 a 3. Número de documentos por página (exemplo de valores calculados a partir de (Liu *et al.*, 2001): 6 a 16.

Tamanho de documentos: a caracterização da distribuição de tamanho de documentos depende do sítio a modelar, podendo a caracterização se tornar trabalhosa, dependendo do caso. Adotou-se uma distribuição log-normal com valores típicos de $\mu = 7$ e $\sigma = 1$ como em (Crovella *et al.*, 1999); considerou-se no momento, apenas a representação do corpo da distribuição no simulador (a falta da representação da cauda dessa distribuição não constitui deficiência relativa ao modelo em si).

Número de *threads* (ou processos) HTTP. O valor *default* máximo no servidor Web Apache é de 256, para o Apache versão 2.0. Na versão anterior (1.3) era de 150.

Tempo de CPU da *thread* HTTP. Podem ser tomados valores na faixa de 2,5 a 15ms/bloco de KByte, dependendo da capacidade da CPU considerada.

Tempo para buscar o arquivo no disco: valor da ordem de um milissegundo (ms) para documentos estáticos.

6.2.3 Fase 3

Número de *buffers* de I/O (N_{IO}): 128, 256 ou 512 são exemplos de valores a considerar.

Tamanho dos *buffers* de I/O (B_{IO}). Este tamanho é configurado para que possa caber integralmente no *buffer* o documento solicitado, em 95 a 99% dos casos. Dessa forma, valores típicos seriam na faixa de 20 a 100KB. A representação do *buffer* de I/O no simulador considerou

o seguinte: se o documento couber inteiramente no *buffer*, o que ocorre, por exemplo, em 95% dos casos, o documento é enviado para rede. Senão, o documento vai ter um retardo no envio, dependendo do quociente da divisão do tamanho do que restou para ser enviado pelo tamanho do *buffer*.

RTT (Round Trip Time: tempo de viagem de ida e volta através da rede): para redes locais, 1ms; para WANs, da ordem de 250ms.

Outros parâmetros usados no modelo base original, tais como por exemplo, o MSS (Maximum Segment Size), a velocidade de conexão da rede (μ_{NC}), a taxa (ou velocidade) com que o cliente lê os dados que chegam (μ_{client}), bem como tamanho máximo da janela TCP, não serão abordados aqui em virtude da simplificação presentemente feita no subcomponente do modelo relativa ao envio do documento pela rede (do servidor para cliente). Serão considerados no caso de se fazer a expansão desse subcomponente.

Observe-se que em muitos casos se adotou a distribuição normal por simplificação (embora tal representação possa ser inexata), não vindo tal consideração de um detalhe na modelagem, a comprometer a proposta do modelo global (aqui proposto) em si.

6.3 Solução do modelo por simulação

O modelo aqui proposto é resolvido por simulação. A Figura 6.1 representa o modelo transcrito para o simulador. Usou-se até o presente o software proprietário ARENA na simulação por já se estar familiarizado com o uso do mesmo, por ele apresentar alguns recursos já embutidos e porque se encontrava disponível uma licença para seu uso, apesar de limitações que ele apresenta.

O software de simulação (simulador) ARENA apresenta uma abordagem da simulação orientada a processos. Nesse tipo de abordagem, entidades no modelo passam por todos processos seqüenciais descritos no modelo, até completar todo processamento. Na presente tese as entidades são as sessões dos usuários, que são completamente processadas no modelo transcrito para o simulador.

Vem a seguir uma descrição de etapas referentes ao modelo simulado.

1ª Etapa (Taxa de Sessões): nessa etapa é introduzido o parâmetro relativo ao intervalo de tempo entre a chegada sessões, que consideramos regido por uma distribuição exponencial, com (por exemplo) média $\mu = 2$ segundos.

2ª Etapa (Conexão TCP): Processo que acrescenta um atraso proveniente do tempo de se estabelecer a conexão TCP entre o servidor e o cliente, tendo como agente ativo o *daemon* TCP; o atraso tem distribuição exponencial de $\mu = 0,25$ segundos (por exemplo, no caso de uma rede de grande distância).

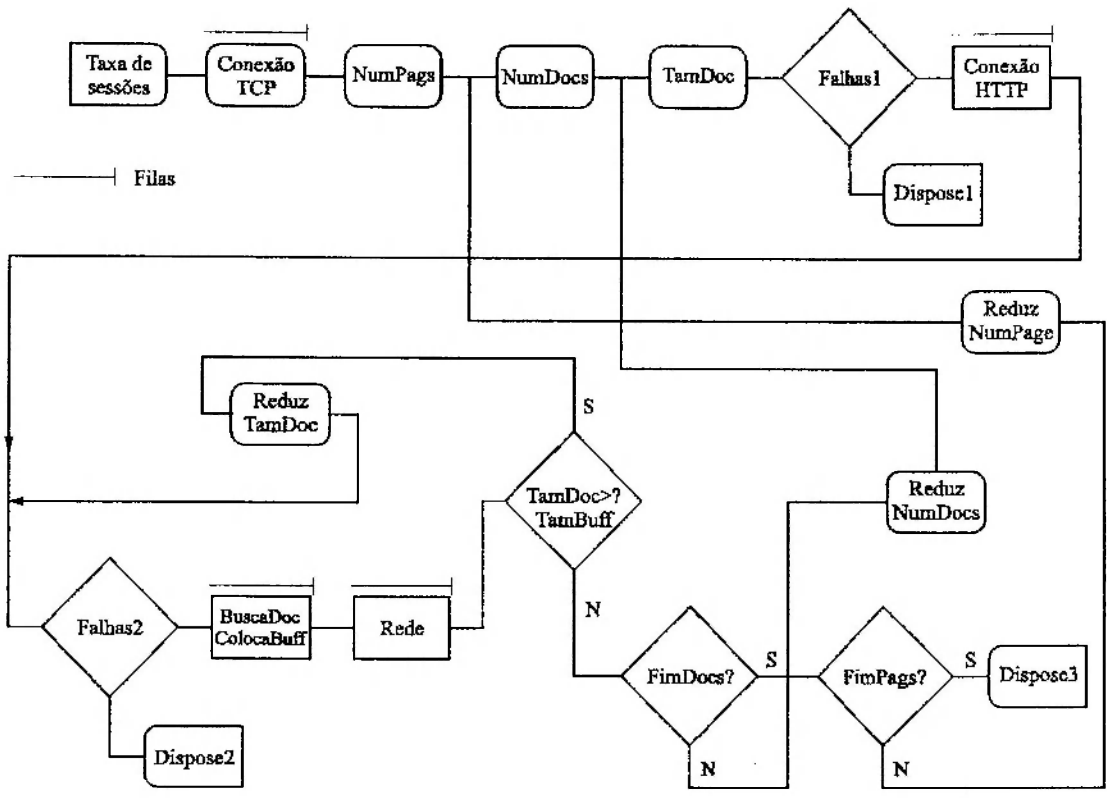


Figura 6.1: Modelo transcrito para o simulador ARENA

3ª Etapa (NumPags): Determina o valor do atributo numPag que representa o número de páginas que constituem uma determina sessão, é regido por uma distribuição normal, por exemplo, com $\mu = 4$ e $\sigma = 2$.

4ª Etapa (NumDocs): Atribui um valor que segue uma distribuição normal de $\mu = 10$ e $\sigma = 3$ para representar o número de documentos que formam uma determinada página.

5ª Etapa (TamDoc): Determina o valor do atributo TamDoc que representa o tamanho de um determinado documento que está contido em uma página, é regido por uma distribuição log-normal, com (por exemplo)$\mu = 7$ e $\sigma = 1$ (valores estes em Bytes).

6ª Etapa (Falhas1): Um desvio condicional que descarta 5% das requisições, simulando assim falhas inerentes ao sistema, como falhas físicas, *time out*, entre outros eventos.

7ª Etapa (Dispose1): Repositório que armazena as requisições tidas como falhadas.

8ª Etapa (ConexãoHTTP): Acrescenta um atraso de tempo na simulação que corresponde ao processamento HTTP propriamente dito; o agente ativo são as *threads* HTTP, que tem seu tempo de processamento regido por uma distribuição normal com $\mu = 20$ e $\sigma = 5$ milissegundos, por exemplo, para o caso de documentos estáticos.

9ª Etapa (Falhas2): Idem a falhas 1.

10ª Etapa (Dispose2): Idem a Dispose 1.

11ª Etapa (BuscaDoc / ColocaNoBuffer): Etapa que representa o processo de busca do

arquivo em disco, e armazenamento do mesmo no *buffer* para ser enviado para a rede; o tempo para esta etapa tem distribuição normal de $\mu = 5$ e $\sigma = 2$ (dados em milissegundos).

12ª Etapa (Rede): Acrescenta um atraso na simulação relativo ao tempo de se enviar o documento pela rede, tendo seu valor relacionado com o quociente da divisão do tamanho do documento pela largura de banda da rede.

13ª Etapa (TamanhoDoBuffer<TamDoc): Desvio condicional que avalia se o tamanho do documento é menor que o tamanho do buffer. Caso seja, isso significa que o documento foi enviado integralmente para o cliente; caso contrário, a porção de documento já enviada é subtraída do tamanho total do documento e o fluxo da simulação é desviado para a etapa Falhas2 para continuar a simulação do envio do restante do documento pela rede ao cliente.

14ª Etapa (FimDocs): Desvio condicional que avalia se o valor do atributo numDoc é igual a zero, ou seja, se todos os documentos de uma página foram enviados; caso seja verdadeiro, o atributo numPag é subtraído de 1 (uma página foi enviada com sucesso); caso contrário (é falso) o fluxo é desviado para o processo TamDoc que atribuirá um novo tamanho de documento para ser enviado ao cliente.

15ª Etapa (FimPags): Desvio condicional que avalia se o valor do atributo numPag é igual a zero, ou seja, se todos as páginas de uma sessão foram enviadas com sucesso; caso seja verdadeiro, a entidade vai para o repositório Dispose3, significando que todas as páginas daquela sessão foram enviadas com sucesso; caso seja falso então o fluxo é desviado para o processo NumDocs que atribuirá um novo número de documentos, que formarão uma nova página a ser vista pelo usuário.

Resumindo, a carga de entrada no simulador é constituída por uma taxa de sessões por segundo e após se estabelecer a conexão TCP (no início de uma sessão) é feita a transformação da sessão para o número de documentos equivalente, considerando as distribuições do número de páginas/sessão, do número de documentos/página e do tamanho dos documentos.

Em seguida é feita a atribuição de um processo HTTP (para atender à requisição dos documentos), podendo haver algum tipo de falha nesse ponto (como se poderia considerar também por ocasião da conexão TCP). Após ser alocado um processo HTTP, este irá buscar o documento e colocá-lo no *buffer* de I/O, para envio à rede, podendo também nesse caso haver algum tipo de falha. Caso o tamanho do documento seja maior do que o *buffer*, o tamanho do documento será reduzido, e o que restar será enviado ao *buffer* pela mesma *thread* HTTP que permaneceu ociosa esperando.

O restante da lógica do modelo no simulador considera simplesmente os documentos que faltam ser enviados para completar a página e as outras páginas que faltam para completar a sessão.

São apresentados a seguir alguns parâmetros de entrada usados no simulador, na ta-

bela 6.1, cujo conjunto chamaremos de configuração 1 (config.1).

Tabela 6.1: Parâmetros de entrada do modelo implementado no simulador

Parâmetros de entrada e fases	Distribuição	Valores	Unidade
Tempo médio entre chegada de sessões	Exponencial	$\mu = 2$	seg
Conexão TCP	Exponencial	$\mu = 0,25$	seg
Proc. HTTP	Normal	$\mu = 0,020; \sigma = 0,005$	seg
Buffer	Normal	$\mu = 0,05; \sigma = 0,02$	seg
Proc. Rede	Normal	$\mu = 0,4; \sigma = 0,2$	seg
No.de páginas/sessão	Normal	$\mu = 4\ ; \sigma = 2$	—
No.de documentos/página	Normal	$\mu = 10\ ; \sigma = 3$	—
Tamanho de documentos	Lognormal	$\mu = 7 \cdot \sigma = 1$	Byte
Tamanho do Buffer	—	50	KByte

Seguem alguns resultados do modelo simulado para cinco configurações diferentes, na tabela 6.2, com comentários vindo após. Todos os tempos são dados em segundos.

Tabela 6.2: Resultados obtidos no simulador

Métricas	Config.1	Config.2	Config.3	Config.4	Config.5
Throughput (sessões/seg)	0.486	1.910	0.461	0.472	9.325
Tempo(medio) total	0.767	1.575	0.902	0.021	0.031
T.medio(Conexão TCP)	0.250	0.248	0.250	0.001	0.001
T.medio (Proc. HTTP)	0.010	0.009	0.010	0.009	0.015
T.medio (Buffer)	0.050	0.051	0.102	0.010	0.014
T.medio (Rede)	0.457	1.267	0.54	0.001	0.001

Comentários:

Com a configuração 1, pode-se inicialmente verificar o funcionamento básico do modelo no simulador. O tempo médio entre chegada de sessões (na distribuição exponencial) é igual a 2 segundos, o que equivale a uma taxa média de 0,5 sessões por segundo (distribuição de Poisson), sendo o *throughput* do servidor obtido, de 0,485 sessões por segundo (aproximadamente igual).

Na configuração 2 aumenta-se a carga de um fator de 4 (o tempo médio entre chegada de sessões passou de 2 segundos para 0,5 segundos), mantendo-se todos outros parâmetros iguais. Nas configurações 1 e 2 a rede se mostra como o componente mais sobrecarregado (com os maiores tempos de 0,457 e 1,267). Fila de espera aparece somente no componente Rede, sendo o tempo médio de espera de 0.052 segs. (não apresentado na tabela) bem como um número médio de 0,02 requisições na fila.

Na configuração 3, retorna-se à carga de chegada da configuração 1, mas o parâmetro da média dos arquivos, na distribuição lognormal, foi aumentado de 7 para 20 e se reduziu o tamanho do *buffer* de 50 para 20KB. Com isso, se observa que o tempo médio referente ao componente *Buffer* passou de 0,05 para 0,102.

Na configuração 4, toma-se a carga de chegada da configuração 1, a média dos arquivos de 7, mantendo-se o tamanho do *buffer* de 20KB, mas considera-se a rede agora uma LAN, com RTT de um milissegundo. Observe-se que nesse caso as maiores parcelas do tempo de resposta se referem às componentes HTTP e *Buffer*.

Na configuração 5 foram mantidos todos os parâmetros da configuração 4, aumentando-se a carga de entrada para o maior valor adotado, com tempo médio entre chegada de sessões de 0,1 segundos (chegada de 10 sessões por segundo). Verifica-se que o *throughput* torna-se quase 10% menor do que a carga de entrada, o que sugere a possibilidade da ocorrência de algum gargalo no servidor, como um todo.

6.4 Validação

6.4.1 Introdução

Em diversos testes efetuados o modelo exibiu comportamento realista e razoável quanto à continuidade, consistência e degeneração (Pegden *et al.*, 1990).

Quanto à continuidade, verificou-se que pequenas mudanças nos parâmetros de entrada causam pequenas mudanças nos dados de saída e nas variáveis de estado do sistema. Por exemplo, um pequeno aumento na taxa de chegada a um centro de serviço resultou em um pequeno aumento no número médio de elementos na fila, no caso de esta se haver formado.

Em relação à consistência, execuções similares do modelo produziram essencialmente resultados similares. Por exemplo, não houve resultados amplamente diferentes simplesmente com a mudança da semente de número aleatório para os geradores de variáveis aleatórias.

Quando algumas características do modelo foram removidas os dados de saída refletiram essa remoção (degeneração).

Os principais testes quanto à validação do modelo foram feitos comparando-se valores obtidos com o modelo implementado no simulador, com valores experimentais obtidos em um laboratório, usando uma LAN com 100 Mbps. Essa técnica de validação (por comparação com observações experimentais) foi utilizada nesta tese por ser aceita cientificamente, além de ter sido adotada diversos casos semelhantes de modelagem existentes na literatura, por exemplo em (van der Mei *et al.*, 2001).

6.4.2 Geração de carga orientada a sessão com o httperf

Foi usado o gerador de carga orientado a sessão do httperf, ferramenta descrita na seção relativa a *benchmarks* para servidores Web do capítulo 3. A chegada de sessões é gerada probabilisticamente com uso do comando *wsesslog*, o qual é acompanhado de três parâmetros, N,X

e F. N é o número total de sessões a serem geradas no teste; X é o *think-time* do cliente a ser usado como padrão entre sessões, caso não seja especificado no arquivo F; F é o arquivo que irá especificar as sessões a serem solicitadas. Nesse arquivo constam as sessões com documentos de tamanhos variados, que venham a resultar nas médias desejadas de tamanho, *think-time* e ainda outras características existentes em documentos requisitados em sessões, como rajadas.

6.4.3 Ambiente de testes

O ambiente laboratorial de testes foi constituído de máquinas clientes e servidor executando em ambiente isolado, para não haver interferências nos testes realizados.

As máquinas cliente e servidor usadas nos testes de laboratório foram Pentium 4, 1,8 GHz, com 256 MB de RAM e HD de 40 GB. A placa de rede foi de 100 Mbits e foi utilizado um *switch* de 100 Mbps, interligando clientes e servidor. A distribuição Linux usada foi o Fedora Core 1.0.

Foram feitas algumas mudanças na configuração do kernel do Linux das máquinas cliente, por exemplo, o número de descritores de arquivo disponíveis foi aumentado de 1024 para 32768. Os processos não essenciais no cliente foram desabilitados para minimizar o consumo de recursos por processos não relacionados à geração de carga. Ambos esses cuidados, entre outros, são no sentido de se evitar que o próprio cliente se torne um gargalo nos testes de laboratório.

Inicialmente foram feitos alguns experimentos para validar o ambiente teste (Titchkosky *et al.*, 2003), tendo se verificado o número máximo de requisições para documentos com 10K de tamanho, conforme a figura 6.2, na qual foi suficiente o uso de apenas uma máquina cliente.

Conforme a figura 6.2, verifica-se que o número máximo de requisições para arquivos de 10K é de cerca de 1200 requisições por segundo, o que equivale a um fluxo máximo na rede próximo ao máximo possível de 100Mbps. Isso significa que, nesse caso, ocorre um gargalo na rede antes do servidor. A possibilidade de o próprio cliente ter se tornado o gargalo é descartada, por não haverem ocorrido erros que apontassem para isso nos resultados de saída do httperf.

O gráfico da figura 6.2 foi gerado usando o software Autobench (Midgley, 2004), um *script* que executa a partir do httperf.

Foi usado o servidor Web Apache versão 2.0; nessa versão do Apache deve-se escolher um MPM (***Multi-Processing Modules***)(modelo de concorrência). Para plataformas UNIX em geral são citados dois MPMs, sendo que a escolha pode afetar a velocidade e crescimento em escala do *httpd*. O *worker* MPM usa processos escravos com várias *threads* cada. Cada *thread* manipula uma conexão por vez. O *pre-fork* MPM usa múltiplos processos escravos com uma ***thread*** cada. Cada processo atende uma conexão por vez. Observação: em muitos sistemas, o *pre-fork* é comparável em velocidade ao *worker*, mas ele usa mais memória. O projeto sem ***threads*** do *pre-fork* tem vantagens em algumas situações, por exemplo, é mais fácil

para depuração em plataformas com suporte fraco na depuração para *threads*.

Foi considerado aqui somente o caso do MPM *pre-fork*, no qual cada processo (uma só *thread*) atende apenas a uma conexão.

O valor default (utilizado) do parâmetro MaxClients (o número máximo de processos servidores) no Apache 2.0 foi de 256.

6.4.4 Experimentos realizados

Para a validação foram realizados experimentos com taxas de chegada de sessões e parâmetros iguais àqueles usados no simulador. Os parâmetros incluíram:

- O número médio de páginas por sessão. Na validação foi considerado o valor de 1 página na sessão
- O número médio de documentos por página. Foram considerados valores de 5 e 15
- O tamanho médio dos documentos (nas sessões consideradas). Foram considerados valores de 10 e 100K.
- A *bandwidth* da rede local (100Mbps)
- O RTT entre cliente e servidor utilizado, correspondente à LAN utilizada (para estabelecimento de conexão e envio pela rede a clientes). Foi considerado valor da ordem de unidade de milissegundos.
- O número máximo de processos HTTP do servidor Web Apache (*MaxClients*)

Na simulação foram usadas 10 réplicas para obtenção de médias, tendo sido descartados os resultados iniciais relativos à fase (inicial) de transição, antes de se chegar à condição estável no sistema (Jain, 1991). Para os testes de laboratório também foram adotadas réplicas, sendo estas em menor número (3). Contudo, foi verificado não ter havido variabilidade significativa nos resultados observados para obtenção das médias amostrais.

6.4.5 Comparação de Resultados Experimentais e Simulados

Na validação foram considerados três valores para a carga de entrada de sessões por segundo, respectivamente, 2, 10 e 20 (sobrecarga); mediram-se os resultados para o caso de 5 e 15 (número médio de documentos por página) e médias de tamanho de documentos de 10 e 100KB. As métricas observadas foram o *throughput* (sessões por segundo) e o tempo de resposta médio por documento (milissegundos). Os resultados obtidos se encontram na tabela 6.3. O I/O de rede mede a largura de banda média de rede utilizada em Mbps.

Tabela 6.3: Comparação de resultados obtidos na simulação e em laboratório

Carga	No. docs	Tam. docs	SIMULADOR		LABORATÓRIO		
			throughput	Tresposta	*throughput*	Tresposta	I/O Rede
2	5	10	1,94	1,5	2,04	1,9	0,9
2	5	100	1,90	14,4	2,04	15,2	8,4
2	15	10	1,76	1,8	1,87	2,0	2,6
2	15	100	1,73	16,9	1,85	17,6	25,4
10	5	10	9,97	1,8	9,95	1,9	-
10	5	100	9,65	18,5	9,74	20,1	-
10	15	10	9,47	2,0	9,68	2,1	-
10	15	100	9,20	24,1	9,53	23,2	-
20	5	10	14,70	2,8	16,13	2,2	6,8
20	5	100	12,11	75,1	16,10	63,6	66,7

Os valores relativos ao caso de 15 documentos por sessão não são apresentados para o caso da carga de chegada ser de 20 sessões por segundo pois mais de 30% das sessões falharam, em laboratório, nesse caso; aqui ocorreu diminuição do *throughput* do servidor e grande aumento no tempo de resposta, o que indica sobrecarga. Com efeito, o caso da carga de 20 sessões por segundo equivale à saturação da rede entre cliente e servidor (a capacidade máxima declarada é de 100Mbps, porém na prática se obtém uma capacidade real utilizável abaixo desse valor).

Por outro lado, no simulador, quando as intensidades de carga requisitadas levaram à sobrecarga, ocorreram tempos de aumento na simulação dificultando a obtenção prática dos resultados nesse caso, em relação ao tempo demandado para os experimentos.

Pode ser observado na tabela 6.3 que a discrepância na maioria dos casos não é significativa, situando-se em geral abaixo de 20%.

Foram levados a efeito testes de hipótese. O teste foi o de diferença das médias observadas, para o caso de dados emparelhados, em relação aos *throughputs* e tempos de resposta observados no simulador e em laboratório.

Os valores relativos à carga de 20 sessões por segundo, no caso do número médio de documentos por página ser 5 e a média do tamanho dos documentos ser 100KB, foi desconsiderado como sendo um "outlier", ou seja, um valor discrepante em relação aos outros dados obtidos (considerou-se que este valor discrepante resulta da geração de carga de trabalho já na faixa relativa à sobrecarga). Observe-se que foram mantidos os valores relativos à carga de 20 sessões por segundo, média de 5 documentos por página e média de tamanho de documentos de 10KB, na presente análise estatística.

Os valores usados na análise são mostrados na tabela 6.4.

A tabela 6.4 apresenta os dados obtidos a partir dos dados amostrais, de acordo com o teste que considera as seguintes hipóteses (tanto para o caso de *throughputs* como para o de

Tabela 6.4: Valores utilizados para a análise Estatística

Carga	ndocs	tamdocs	SIMULADOR		LABORATÓRIO			
			tput	Tresp	tput	Tresp	dif(tput)	dif(Tresp)
2	5	10	1,94	1,5	2,04	1,9	0,10	0,4
2	5	100	1,90	14,4	2,04	15,2	0,14	0,8
2	15	10	1,76	1,8	1,87	2,0	0,11	0,2
2	15	100	1,73	16,9	1,85	17,6	0,12	0,7
10	5	10	9,97	1,8	9,95	1,9	-0,02	0,1
10	5	100	9,65	18,5	9,74	20,1	0,09	1,6
10	15	10	9,47	2,0	9,68	2,1	0,21	0,1
10	15	100	9,20	24,1	9,53	23,2	0,33	-0,9
20	5	10	14,70	2,8	16,13	2,2	1,43	-0,6
						Σdif	2,51	2,4
						$d = \Sigma dif/9$	0,2788	0,2627
						s^2_{dif}	0,1879	0,555
						s_{dif}	0.4355	0,745
						$t_{n-1}(obs)$	1,891	1,074
						$t_{8;0,025}$	2,306	2,306

tempos de resposta observados):

H_0 : A média das diferenças é igual a zero (não é significativamente diferente de zero).

H_1 : A média das diferenças é diferente de zero.

A variável teste é a t de *Student* com $n - 1 = 8$ GL (Graus de Liberdade). O valor observado (t_{obs}) é dado por:

$t_{obs} = \bar{d}/(s_{dif}/\sqrt{n})$, $(n = 9)$.

Como se observa em ambos os casos, não se rejeita a hipótese nula, ou seja, conforme os dados amostrais não há, ao nível de significância considerado ($\alpha = 5\%$), dados estatísticos que apontem para a rejeição de H_0.

A figura 6.3 apresenta o gráfico dos *throughputs* obtidos na simulação e em laboratório.

A figura 6.4 apresenta o gráfico dos tempos de resposta obtidos na simulação e em laboratório.

6.5 Considerações finais

Este capítulo apresentou a solução e validação do novo modelo proposto nesta tese, incluindo os parâmetros utilizados e outros aspectos importantes relativos ao modelo apresentado.

Após a validação, o modelo pode ser usado para previsão de resultados em avaliação de desempenho. Assim, suponha-se que os dados de tempo de resposta, obtidos no modelo, conforme a tabela 6.3 sejam tomados como da (variável) resposta (C5) em um modelo de regressão

linear simples onde as variáveis explicativas são a carga de entrada em sessões por segundo (C1), o número médio de documentos por sessão (C2) e o tamanho médio de documentos (C3). Os resultados abaixo foram obtidos a partir do software MINITAB (MINITAB, 2004):

Regression Analysis: C5 versus C1; C2; C3

The regression equation is

$C5 = -14,3 + 1,55C1 - 0,06C2 + 0,309C3$

Predictor	*Coef*	*SECoef*	*T*	*P*
Constant	$-14,29$	$16,13$	$-0,89$	$0,410$
$C1$	$1,5548$	$0,8090$	$1,92$	$0,103$
$C2$	$-0,056$	$1,097$	$-0,05$	$0,961$
$C3$	$0,3091$	$0,1122$	$2,76$	$0,033$

$S = 15,9635$

$R - Sq = 66,4\%$

$R - Sq(adj) = 49,6\%$

Analysis of Variance

Source	*DF*	*SS*	*MS*	*F*	*P*
Regression	3	3022,7	1007,6	3,95	0,072
ResidualError	6	1529,0	254,8		
Total	9	4551,7			

Source	*DF*	*SeqSS*
$C1$	1	1087,1
$C2$	1	0,7
$C3$	1	1934,9

Unusual Observations

Obs	*C1*	*C5*	*Fit*	*SEFit*	*Residual*	*StResid*
10	20,0	75,10	47,44	11,18	27,66	$2,43R$

R denotes an observation with a large standardized residual.

Conforme se observa pelos dados acima, as variáveis explicativas mais significantes (em ordem crescente) para o modelo de regressão linear obtido são a carga (C1), com P=0,103 e o tamanho médio de documentos com P=0,033; observe-se ainda o valor considerado *outlier* anteriormente (seção 6.4.5), (R) conforme efetivamente detectado na análise.

O próximo capítulo conterá as conclusões finais, revisando as contribuições principais da tese já expostas no capítulo 5; indicará ainda trabalhos futuros de interesse, relacionados ao novo modelo proposto.

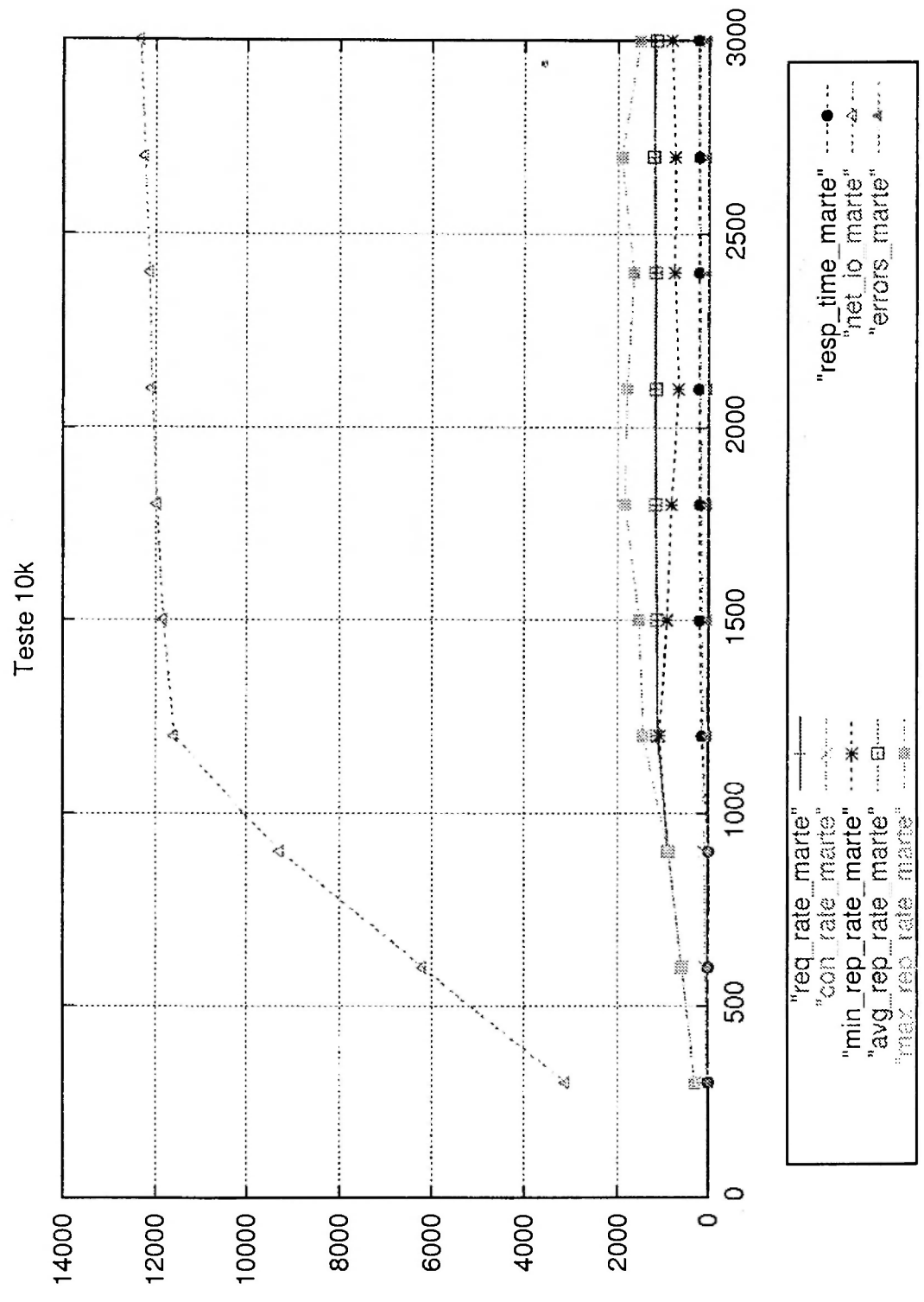


Figura 6.2: Saturação da rede

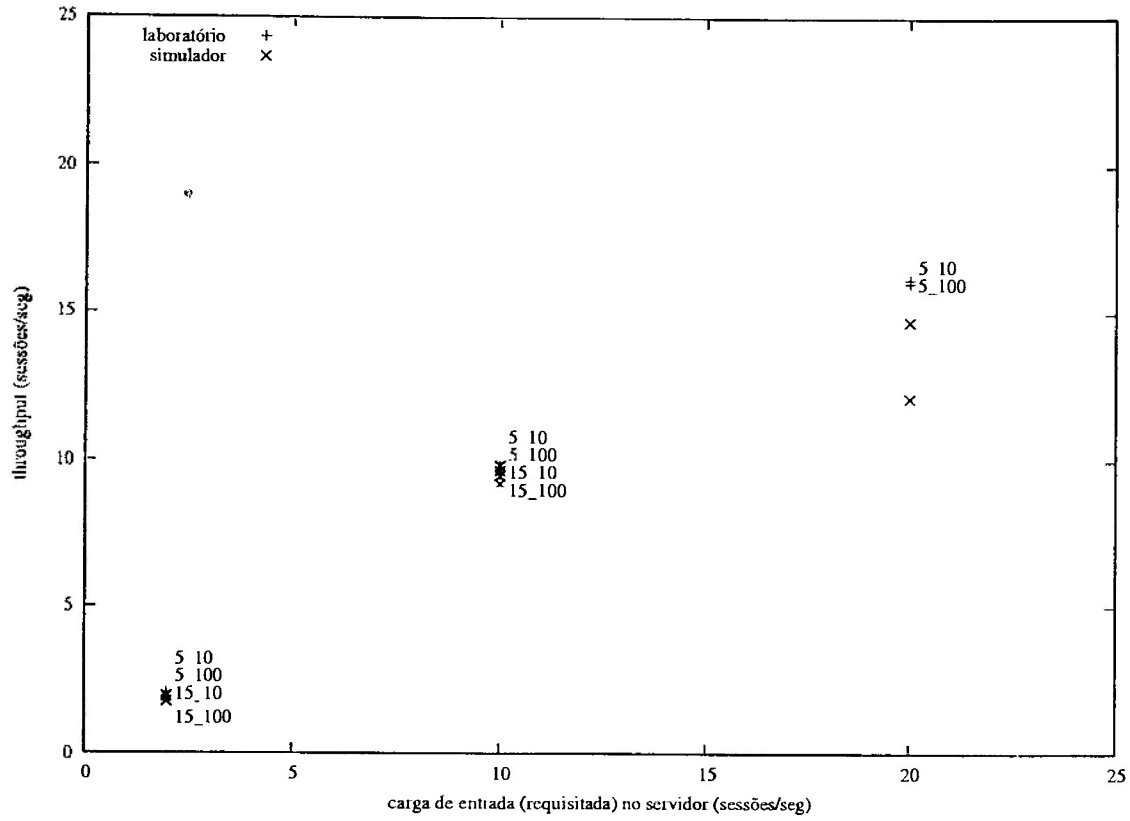


Figura 6.3: *Throughputs* obtidos na simulação e em laboratório

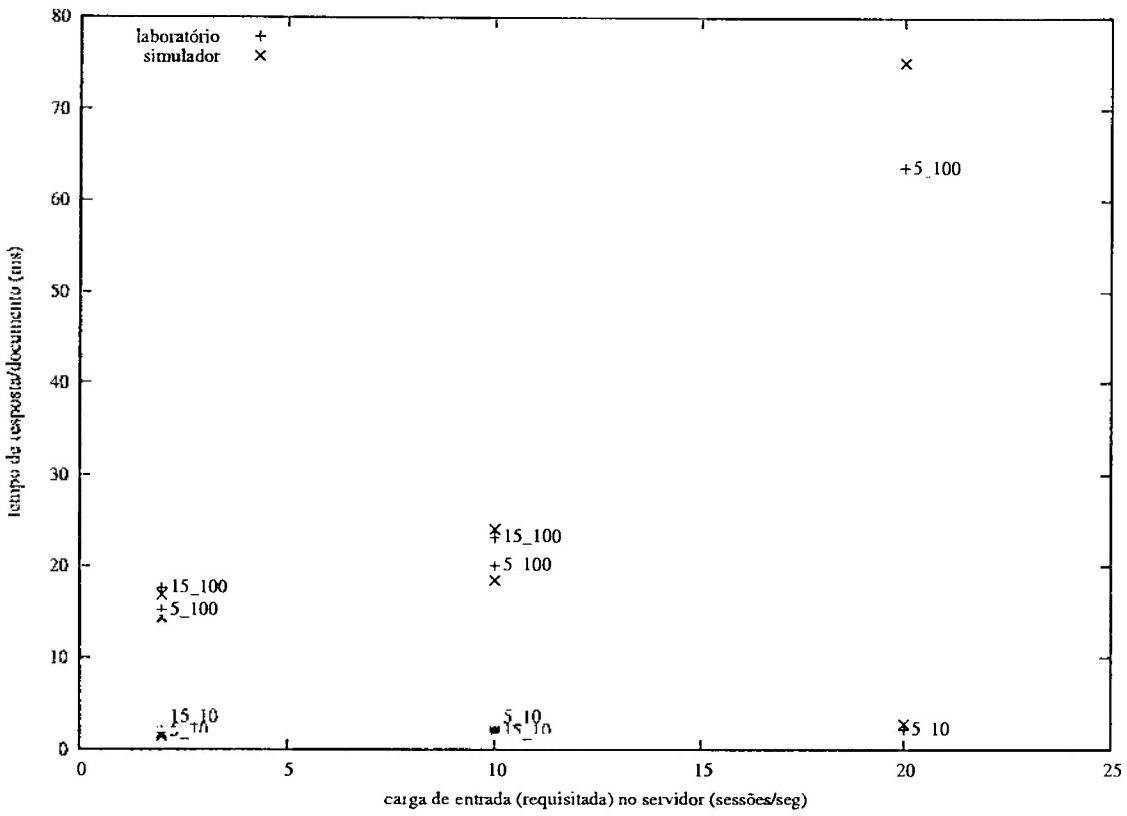


Figura 6.4: Tempos de resposta obtidos na simulação e em laboratório

CAPÍTULO

7

Conclusões finais: contribuições e trabalhos futuros

O modelo apresentado por (van der Mei *et al.*, 2001) é o único modelo encontrado que mostra explicitamente as componentes internas do software servidor, para modelar a completa comunicação entre cliente e servidor; o modelo busca descrever os aspectos reais de processamento internamente ao servidor Web e na interação com o cliente. Por isso esse modelo foi tomado como base nesta tese

Na presente tese existem duas contribuições inovadoras no novo modelo proposto (conforme já exposto com mais detalhes no capítulo 5), relativas à extensão e modificação efetuadas no modelo base, que visam a melhoria do modelo. Será abordada inicialmente a extensão para conexões persistentes.

O modelo base original publicado em 2001, porém, cuja concepção se originou mais de dois anos antes, considera o caso da existência de uma conexão para cada requisição de documento, conforme visto pelo servidor (caso de conexões não persistentes); esse era o padrão usado no HTTP 1.0.

A extensão do modelo base original para conexões persistentes é sugerida explicitamente em (van der Mei *et al.*, 2001) como trabalho futuro; posteriormente, no início de 2003, o próprio autor principal do modelo, gentilmente informou não saber da existência, ainda, de tal extensão; sendo assim, é inovadora a extensão para conexões persistentes proposta na presente tese.

Será revisto agora porque se adotou aqui uma mudança na carga de entrada para o modelo e porque isso constitui também uma contribuição. Em (van der Mei *et al.*, 2001), onde o modelo é resolvido por simulação, bem como em (Reeser *et al.*, 1999), onde o (mesmo) modelo base é resolvido analiticamente, a carga de chegada (carga de trabalho) para o modelo é constituída por

documentos. O processo da chegada desses documentos é admitido como sendo um processo de Poisson. Porém tal suposição não é confirmada pela maioria de trabalhos pertinentes na literatura; a chegada de sessões, sim, constitui-se sempre em um processo de renovação, podendo em alguns casos ser um processo de Poisson, conforme descrito e comentado no capítulo 5. Além disso, a carga de chegada de documentos é mais difícil de ser modelada (Liu *et al.*, 2001) do que a carga de sessões.

Assim, a carga de chegada de sessões, mais fidedigna do que a de documentos, conforme considerado no modelo base original, facilitará a modelagem de um servidor Web nas condições atuais, já que resulta de modelagem mais fácil como carga de chegada do que a carga orientada a documentos.

Mais simplificado se comparado ao modelo base, o novo modelo proposto pode contudo ser considerado como tendo quatro fases (uma a mais do que o modelo base - que tem três): conexão TCP, processamento HTTP, *buffer* e rede. Isso porque a fase que chamamos de rede nesta tese, no modelo base está embutida, com todos detalhes, na fase do *buffer* de I/O, conforme se pode verificar no modelo original.

Com relação a trabalhos futuros, além da consideração na modelagem de documentos dinâmicos gerados dentro ou fora do servidor, existe ainda o problema da transformação matemática da distribuição relativa à carga orientada a sessão, para a distribuição da carga orientada a documentos, a ser usada internamente no modelo. Essa transformação, que conjeturamos ser homotópica, é uma transformação que utiliza como funções intermediárias (na transformação), as distribuições de páginas por sessão, de documentos por página e de tamanho de documentos.

Uma ferramenta de geração de carga orientada a sessões, incorporando a transformação para documentos, e onde o usuário pudesse especificar as diversas características das sessões, bem como dos documentos, por exemplo, de um tipo que pudesse reunir as vantagens de SURGE e httperf, facilitaria a utilização do modelo para avaliação de desempenho, sendo também um trabalho a se considerar para o futuro.

Referências

Allen, A. O. (1990). *Statistics, Probability and Queueing Theory with Computer Science applications*. Computer Science and Scientific Computing. Academic Press Inc., 24-28 Oval Road NW1 7DX, London, 2a. edição.

Almeida, J. M.; Almeida, V. A. F.; Yates, D. J. (1996). Measuring the Behavior of a World-Wide Web Server. Relatório Técnico TR-CS-96-025, Department of Computer Science, Boston University, Boston, USA.

Anderson, T. E.; Bershad, B. N.; Lazowska, E. D.; Levy, H. M. (1995). Scheduler activations: effective kernel support for the user-level management of parallelism. *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, p. 95–109. Association for Computing Machinery SIGOPS.

Andersson, M.; Cao, J.; Kihl, M.; Nyberg, C. (2003). Performance modeling of an Apache Web server with bursty arrival traffic. *International Conference on Internet Computing (IC)*.

Apache (2004). Apache Software Foundation. *http://www.apache.org*.

Arlitt, M.; Williamson, C. (1996). Web server workload characterization: the search for invariants. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, p. 126–137.

Banga, G.; Druschel, P. (1997). Measuring the Capacity of a Web Server. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, USA.

Banga, G.; Mogul, J. C. (1998). Scalable kernel performance for internet servers under realistic loads. *Proceedings of the USENIX Annual Technical Conference*.

Barford, P.; Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, p. 151–160.

Barford, P.; Crovella, M. (1999). A performance evaluation of hyper text transfer protocols. *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, p. 188–197, Boston, MA, USA.

Blaszczak, M. (1996). Writing interactive Web applications is a piece of cake with the new ISAPI classes in MFC 4.1. *Microsoft Systems Journal.*

Cao, J.; Anderson, M.; Nyberg, C.; Kihl, M. (2003). Web server performance modeling using an $M/G/1/K^*PS$ queue. *International Conference on Telecommunication (ICT).*

Cerf, V.; Kahn, R. (1974). A protocol for packet network interconnection. *IEEE Transactions on Communications*, v.COM-22, p.637–648.

Chankhunthod, A.; Danzig, P. B.; Neerdaels, C.; Schwartz, M. F.; Worrel, K. J. (1996). A hierarchical Internet object cache. *USENIX Technical Conference.*

Comer, D. (1997). *Computer Networks and Internets.* Upper Saddle River - New Jersey.

Coulouris, G.; Dollimore, J.; Kindberg, T. (1994). *Distributed Systems: Concepts and Design.* Addison Wesley, 2a. edição.

Crovella, M.; Bestavros, A. (1996). Self-similarity in World Wide Web: evidence and possible causes. *Proceedings of the ACM Sigmetrics*, p. 160–169, Philadelphia, USA.

Crovella, M.; Frangioso, R.; Harchol-Balter, M. (1999). Connection scheduling in Web servers. *Proceedings of the USENIX Symposium on Internet Technologies and Systems.*

da Fonseca, N. M. (2002). Um estudo sobre a topologia de redes P2P. Dissertação (Mestrado), Universidade Federal de Minas Gerais, DCC-UFMG, Belo Horizonte - Brasil.

Dilley, J.; Friedrich, R.; Jin, T.; Rolia, J. (1998). Web server performance measurements and modeling techniques. *Performance Evaluation*, v.33, p.5–26.

Druschel, P.; Banga, G. (1996). Lazy receiver processing LRP: A network subsystem architecture for server systems. *2nd Symposium on Operating Systems Design and Implementation*, Seattle, WA.

Ehrlich, W. K.; Hariharan, R.; Reeser, P. K.; van der Mei, R. D. (2000). Performance of web servers in a distributed computing environment. Relatório técnico.

Frances, C. R. L. (1998). Uma extensão estocástica para os Statecharts. Dissertação (Mestrado), ICMC-USP, São Carlos/SP, Brasil.

Hariharan, R.; Ehrlich, W. K.; Cura, D.; Reeser, P. K. (2000). End to end performance modeling of Web server architectures. *Proceedings of the PAWS 2000 Conference.*

Heidemann, J.; Obraczka, K.; Touch, J. (1997). Modeling the Performance of HTTP over Several Transport Protocols. *IEEE/ACM Transactions on Networking*, v.5, n.5, p.616–630.

Hu, J. C.; Mungee, S.; Schmidt, D. C. (1997a). Principles for Developing and Measuring High-performance Web Servers over ATM. Relatório Técnico WUCS-97-09, Department of Computer Science, Washington University, St. Louis, MO, USA.

Hu, J. C.; Pyarali, I.; Schmidt, D. C. (1997b). Measuring the Impact of Event Dispatching and Concurrency Models on Web Server Performance over High-speed Networks. Phoenix, Arizona, USA.

Hu, Y.; Nanda, A.; Yang, Q. (1999). Measurement, Analysis and Performance Improvement of the Apache Web Server. *Proceedings of the 18th IEEE International Performance Computing and Communications Conference*, Phoenix, Arizona, USA.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, New York.

Kaashoek, M. F.; Engler, D. R.; Ganger, G. R.; Wallach, D. A. (1996). Server operating systems. *SIGOPS European Workshop*.

Kant, K. (1999). Server capacity planning for Web traffic workload. *IEEE Transactions on Knowledge and Data Engineering*, v.11, n.5, p.731–747.

Kant, K.; Sundaram, C. R. M. (2000). A server performance model for static Web workloads. Relatório técnico, Intel Corporation (Server Architeture Lab).

Kleinrock, L. (1975). *Queueing Systems, Vol1: Theory*. Wiley, New York, USA.

Kleinrock, L. (1976). *Queueing Systems, Vol2: Computer Applications*. Wiley, New York, USA.

Lazowska, E.; Zahorjan, J.; Graham, G.; Sevcik, K. (1984). *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*. Prentice Hall, Englewood Cliffs, NJ, USA.

Leland, W.; Taqqu, M.; Willinger, W.; Wilson, D. (1994). On the self-similar nature of Ethernet traffic. *IEEE Transactions on Networking*, v.2, p.1–15.

Liu, Z.; Niclausse, N.; Jalpa-Villanueva, C. (2001). Traffic model and performance evaluation of Web servers. *Performance Evaluation*, , n.46, p.77–100.

Maltzahn, C.; Richardson, K. J.; Grunwald, D. (1997). Performance issues of enterprise level Web proxies. *Proceedings of the ACM SIGMETRICS Conference*, Seattle, WA.

Menasce, D.; Almeida, V. (1998). *Capacity Planning for Web Performance*. Prentice Hall, Upper Saddle River, New Jersey, 1a. edição.

Midgley, J. T. J. (2004). http://www.xenoclast.org.

MINITAB (2004). http://www.minitab.com.

Mogul, J. C. (1995). Network behavior of a busy Web server and its clients. Relatório Técnico TR-WRL 95/5, Western Research Laboratory, Palo Alto, CA.

Mogul, J. C.; Ramakrishnan, K. K. (1996). Eliminating receive livelock in an interrupt-driven kernel. *Usenix Technical Conference*, p. 99–111.

Mosberger, D.; Jin, T. (1998). "httperf: A tool for measuring web server performance". *ACM Performance Evaluationa Review*, v.26, n.3, p.31–37.

Nahum, E. M. (2002). Deconstructing SPECWeb99. *IBM T.J. Watson Research Center.*

Nielsen, H.; Gettys, J.; Bair-Smith, A.; Prud'Hommeaux, E.; Lie, H.; Lilley, C. (1997). Network performance effects of HTTP/1.1 CSS1 and PNG. *Proceedings of the ACM SIGCOMM'97*, Cannes, France.

O'Neil, P. (1994). *Database: Principles, Programming, Performance.* Morgan Kaufmann, San Francisco, CA.

Orfali, R.; Harkey, D.; Edwards, J. (1996). *The Essential Distributed Object Survival Guide.* Wiley, New York, 2a. edição.

Orfali, R.; Harkey, D.; Edwards, J. (1999). *The Client-Server Survival Guide.* Wiley, 3a. edição.

Ousterhout, J. (1996). Why threads are a bad idea (for most purposes). Invited Talk at USENIX Technical Conference.

Padmanabhan, V.; Mogul, J. (1995). Improving HTTP latency. *Computer Networks and ISDN Systems*, v.28, n.1,2.

Pai, V. S.; Druschel, P.; Zwaenepoel, W. (1997). Io-lite: A unified I/O buffering and caching system. Relatório Técnico TR97-294, Rice University, CS Dept, Houston TX.

Paxson, V.; Floyd, S. (1995). Wide area traffic: the failure of Poisson. *IEEE Transactions on Networking*, v.3, p.226–244.

Pegden, C. D.; Shannon, R. E.; Sadowski, R. P. (1990). *Introduction to Simulation using SIMAN.* McGraw and Hill, Inc.

Pinheiro, J. C. (2001). Avaliação de políticas de substituição de objetos em caches na Web. Dissertação (Mestrado), ICMC- USP, Universidade de São Paulo - São Carlos.

Reeser, P. K.; van der Mei, R. D.; Hariharan, R. (1999). An analytic model of a Web server. *Proceedings of the 16th International Teletraffic Congress*, p. 1199–1208. Elsevier Science.

Rolia, J.; Sevcik, K. (1995). The method of layers. *IEEE Transactions on Software Engineering*, v.21, n.8, p.689–700.

S-clients (2004). http://www.cs.rice.edu/cs/systems/web-measurements.

Shaw, D. M. W. (1998). A low latency, high throughput Web service using Internet-wide replication. Dissertação (Mestrado), The John Hopkins University.

Slothouber, L. P. (1995). A Model of Web Server Performance. *Starmine Technologies*.

Somin, Y.; Agrawal, S.; Forsyth, M. (1996). Measurement and analysis of process and workload CPU times in UNIX environments. *Proceedings of the CMG*.

Souza, P. S. (1996). Máquina paralela virtual em ambiente Windows. Dissertação (Mestrado), ICMC-USP, São Carlos/SP - Brasil.

SPECWeb (2004). SPECweb99 benchmark. *http://www.specbench.org/osg/web99*.

SQUID (2004). http://www.squid.org.br.

Stevens, W. (1994). *TCP /IP Illustrated*, v. 1. Addison Wesley, Reading, MA.

Stevens, W. (1996). *TCP/IP Illustrated*, v. 3. Addison Wesley, Reading, MA.

SURGE (2004). Scalable URL Reference Generator. *http://www.cs.bu.edu/groups/cwealth*.

Tanenbaum, A. (1996). *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, USA.

Titchkosky, L.; Arlitt, M.; Williamson, C. (2003). Performance Benchmarking of Dynamic Web technologies. *Proceedings of the IEEE MASCOTS 2003*.

van der Mei, R.; Ehrlich, W.; Reeser, P. K.; Francisco, J. P. (1999). A decision support system for tuning Web servers in distributed object oriented network architectures. *Proceedings of the 2nd WISP*, Atlanta, GA. ACM SIGMETRICS.

van der Mei, R.; Hariharan, R.; P.K.Reeser (2001). Web Server Performance Modeling. *Telecommunications Systems*, v.16, p.361–378.

Wallace, R. B.; Jr., T. E. M. (1997). A performance monitoring and capacity planning methodology for Web servers. Relatório técnico, NCR Corporation.

Webstone (2004). http://www.mindcraft.com/webstone.

Welsh, M. (1994). *The Linux Bible*. Yggdrasil Computing Incorporated, 2a. edição.

Woodside, C.; J.E.Neilson; Petriu, D.; Majumdar, S. (1995). The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, v.44, n.1, p.20–34.

Yeager, N.; McCrath, R. (1996). *Web Server Technology*. Morgan Kaufmann, San Francisco, CA.

Zeus (2004). http://www.zeus.com/products/zws.