Suporte a serviços diferenciados em servidores web: modelos e algoritmos

Mário Antonio Meireles Teixeira

Suporte a serviços diferenciados em servidores web: modelos e algoritmos

Mário Antonio Meireles Teixeira

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.

"VERSÃO REVISADA APÓS A DEFESA"

Data da Defesa:	13.05.2004
Visto do Orientador:	

USP – São Carlos
Julho de 2004

Dedico este trabalho a minha família, que sempre esteve a meu lado, dando-me o amor e o incentivo necessários para buscar novos caminhos.

Agradecimentos

Ao Prof. Marcos Santana, pela orientação prestada e pela confiança demonstrada em meu trabalho.

À Profa. Regina Santana, por sua disponibilidade e opiniões seguras.

Aos demais professores e funcionários do Instituto, por seu exemplo de profissionalismo e dedicação. Em especial, ao pessoal da Seção de Pós: Beth, Laura e Ana Paula.

A minha mãe Mimi, minha tia Ana, meu irmão Jorge, minha cunhada Michelline e também aos tios Dulce, Paulo e Maria das Dores e aos primos Terezinha e Carlos Arlindo. Sem esquecer dos que deixaram saudades: meu pai Nazareno e meus avós Mario e Zezé.

Ao amigo Renato Bulcão, companheiro de todas as horas durante esse período, pelas conversas, brincadeiras, conselhos e incentivo. Igualmente à sua Taciana, pela simpatia e generosidade.

Aos amigos e colegas do LaSDPC: Douglas, Hermes, Renato (Japa), Luciano e Lilian, Márcio, Ricardo, Sarita. E também a Álvaro, Caio, Célia, Edmilson, Jacqueline, Juliano, Kalinka, Michel, Renato Francês, Roberta Spolon, Tatiana, Thaís e outros.

Aos amigos do Maranhão: Auxiliadora, Alexandre, Caracas, Erika, Larissa e Eduardo, Paulo Sérgio, Valeska.

Aos integrantes da "colônia" maranhense em São Carlos: Bruno, João Carlos, Nilson, Omar e Elaine, Tanaka.

Aos demais amigos e colegas da USP: Adenilso, Camila e Humberto, Elisângela, Luciana e Richard, Marisa, Rogério Garcia, Rudinei e Flávia, Toño e Alessandra e tantos outros.

Aos colegas e funcionários do Departamento de Informática da UFMA.

À CAPES, pelo apoio financeiro dado a este trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AF	Assured Forwarding
API	Application Programming Interface
ASP	Active Server Pages
CBQ	Class-Based Queueing
CGI	Common Gateway Interface
CLF	Common Log Format
CPU	Central Processing Unit
DNS	Domain Name Service
DS	Differentiated Services Field
DSCP	Differentiated Services Codepoint
EF	Expedited Forwarding
FCFS	First-Come First-Served
FIFO	First In First Out
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
LaSDPC	Laboratório de Sistemas Distribuídos e Programação Concorrente
MIME	Multi-Purpose Internet Mail Extensions
MPLS	Multi-Protocol Label Switching
NFS	Network File System
NSAPI	Netscape Server API
OSI	Reference Model of Open Systems Interconnection
PHP	Hypertext Preprocessor
QoS	Quality of Service
PHB	Per-Hop Behavior
PRIAdap	Mecanismo de Prioridades Adaptativo
RED	Random Early Detection
RFC	Request For Comments
RR	Round Robin
RSV	Mecanismo de Reserva de Recursos
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SQF	Shortest Queue First

SWDS	Servidor Web com Diferenciação de Serviços
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WFQ	Weighted Fair Queueing

Resumo

Atualmente, há uma grande diversidade de aplicações que funcionam sobre a infra-estrutura da Internet, as quais apresentam diferentes necessidades. Como conseqüência, seu modelo de serviços de melhor esforço tem sido incrementado, de modo a permitir o fornecimento de diferentes níveis ou classes de serviço aos clientes. Contudo, de nada adianta garantir uma qualidade de serviço diferenciada na rede, se os elementos finais dessa cadeia, os servidores, não estiverem habilitados a reconhecê-la. Nesse contexto, este trabalho propõe uma arquitetura para um servidor web capaz de fornecer serviços diferenciados a seus clientes, segundo suas características de demanda. Esta arquitetura é verificada por meio de um modelo de simulação e são utilizados *logs* de acesso a servidores web como carga de trabalho. Foram implementados três mecanismos de diferenciação de serviços na arquitetura, os quais correspondem a duas abordagens distintas: enfileiramento baseado em classes e escalonamento baseado em prioridades. Dentre eles, destaca-se o mecanismo de prioridades adaptativo, que realiza uma sintonia fina da qualidade de serviço fornecida, determinando quão rigoroso será o esquema de prioridades empregado. O sistema adquire, então, capacidade de adaptação a variações na carga de trabalho, característica essencial em um ambiente altamente dinâmico como a Web. A arquitetura proposta contempla também o controle de admissão de requisições, a fim de evitar a sobrecarga do sistema, caso a demanda dos usuários atinja níveis elevados. Foram implementados três mecanismos de controle de admissão, os quais utilizam diferentes parâmetros como referência para a tomada de decisão. O controle de admissão revelou-se de fundamental importância para a estabilidade do sistema, bem como para a garantia da qualidade do serviço fornecido aos clientes.

Abstract

Nowadays, there are several types of applications running on the Internet infrastructure with different necessities. Consequently, the current best-effort service model has been enhanced so as to allow the provision of different levels or classes of service to the clients. However, it is useless to sustain a differentiated quality of service in the network if the end elements, the web servers, are not enabled to deal with it. Therefore, this work proposes an architecture for a web server capable of providing differentiated services to its clients according to their demand characteristics. The architecture is validated by means of a simulation model and real web server traces are used for workload generation. Three service differentiating mechanisms have been implemented in the architecture, which correspond to two different approaches: class-based queueing and priority based scheduling. Among them, the adaptive priority mechanism has shown the best results: it allows the tuning of the quality of service provided and determines how strict the use of priorities will be. The system can then adapt itself to various workloads, an essential feature in a highly dynamic environment such as the Web. The proposed architecture also deals with admission control issues, in order to avoid system overload when user demand rises unexpectedly. Three admission control mechanisms have been implemented using different evaluation parameters. The admission control module has proved to be of fundamental importance to system stability as well as to assure the quality of service provided to the clients.

Capítulo

1

Introdução

1.1 Contextualização

A Internet vem experimentando um crescimento intenso nos últimos anos e não se vêem sinais de que tal tendência vá se reverter em um futuro próximo. A World Wide Web tem sido um dos principais fatores que motivaram esse crescimento, pois foi a responsável pela popularização do uso da Internet, de tal forma que se confunde com a própria Internet e com serviços disponíveis há mais tempo, como o *e-mail* e a transferência de arquivos.

Com isso, a Internet se transformou, de uma rede para fins de pesquisa, restrita a instituições acadêmicas e órgãos governamentais, há apenas algumas décadas atrás, em um dos mais importantes meios de comunicação atuais, sendo utilizada para fins informativos, educacionais, de entretenimento e comerciais. Nota-se também a tendência de convergência de outros tipos de rede, como as de telefonia, rádio e TV para a Internet, o que tende a aumentar ainda mais as demandas sobre a mesma (Stardust, 1999b).

Entretanto, originalmente a Internet não foi projetada para o uso observado atualmente, nem para dar suporte à quantidade de carga que lhe é imposta. Ela surgiu como uma rede comutada por pacotes, de grande abrangência, que objetivava o transporte de dados convencionais entre computadores localizados em diferentes partes do mundo (Comer, 2000). Sua administração era descentralizada, até mesmo por exigência do governo dos Estados Unidos, que determinava que a Internet deveria continuar a operar mesmo em situações de guerra. Para funcionar sobre a infra-estrutura da rede, foram projetadas aplicações cliente-servidor, como o correio eletrônico, transferência de arquivos, grupos de notícias e, mais recentemente, a Web.

Circulavam inicialmente pela Internet dados em formato textual, oferecendo-lhe pouca carga. Na última década, impulsionado principalmente pelo surgimento da Web e pelo crescimento dos provedores comerciais, o tráfego aumentou em algumas ordens de grandeza, numa tendência que se mantém até hoje. A mudança não se deu apenas na quantidade de tráfego, mas também na natureza do mesmo, pois o uso de imagens está hoje disseminado na Web e aplicações multimídia já são uma realidade. A Web também se transformou em uma plataforma para a realização de transações comerciais, envolvendo indivíduos e empresas, o chamado comércio eletrônico (Xiao & Ni, 1999).

O fato é que a Internet e, em especial, a Web estão atingindo seu ponto de estrangulamento e a solução não parece ser simplesmente adicionar à rede mais largura de banda, pois esta tem sido sistemática e rapidamente consumida pela demanda dos usuários. É preciso introduzir novos modelos de serviço na rede, que sejam capazes de atender às exigências colocadas pelo crescimento da demanda e pelos novos tipos de aplicações.

O serviço oferecido atualmente na Internet baseia-se em um modelo de melhor esforço (*best-effort*), ou seja, a rede procura, de todas as maneiras, transportar os dados que lhe são confiados, no menor tempo possível, de preferência sem erros ou inconsistências, contudo não são dadas às aplicações garantias de que isso realmente ocorrerá (Comer, 2000). Na maior parte das vezes, considerando-se uma rede com pouco tráfego e aplicações não muito exigentes, este modelo se mostra satisfatório. Os problemas surgem quando a rede está muito sobrecarregada ou quando se pretende executar sobre a mesma aplicações que apresentam requisitos para os quais ela não foi projetada, tais como aplicações multimídia e transações comerciais.

1.2 Motivação

Uma das conseqüências da adoção do modelo de melhor esforço na Internet é o fato de que todo o tráfego é tratado de maneira uniforme, sem nenhum tipo de diferenciação ou priorização. Essa tendência se reflete até mesmo no projeto de serviços críticos da Internet, como a Web, cujos servidores, em sua grande maioria, tratam as requisições dos clientes segundo uma disciplina em que o primeiro a chegar será o primeiro a ser atendido[1] (Yeager & McGrath, 1996).

Entretanto, verifica-se que nem todos os tipos de tráfego e transações são equivalentes ou têm a mesma prioridade para os usuários (Dovrolis & Ramanathan, 1999). Comparem-se, por exemplo, requisições de documentos HTML, disparadas por um servidor *proxy*, com aquelas originadas como resultado de uma busca feita por um usuário. No primeiro caso, trata-se de um tráfego meramente especulativo, que poderia ser atendido com uma prioridade inferior, enquanto que, no segundo caso, o tráfego originado é de real interesse

[1] *First-Come, First-Served* (FCFS)

para quem o solicitou. Outro exemplo é o caso de um usuário que esteja simplesmente navegando por um site de comércio eletrônico, comparado com outro em processo de finalização de uma compra. Evidentemente, é desejável que as transações do segundo indivíduo recebam prioridade e confiabilidade mais altas que as do primeiro. Infelizmente, não é o que ocorre atualmente, pois o tráfego na rede é tratado, em geral, de maneira igualitária, sem que seja considerada sua prioridade relativa aos usuários e aplicações. A mesma situação se dá em nível de aplicação, em que as solicitações que chegam aos servidores web recebem um tratamento uniforme, sem distinção.

Dessa forma, é essencial fornecer diferenciação de serviços com diferentes níveis de qualidade para diferentes tipos de requisições (Kant & Mohapatra, 2000). Considerando-se a infra-estrutura de rede, já existem algumas soluções com esse objetivo, elaboradas sob a coordenação da IETF (*Internet Engineering Task Force*). Estão disponíveis diversas especificações para a provisão de qualidade de serviço sobre redes IP, com destaque para as arquiteturas de Serviços Integrados (Braden *et al.*, 1994) e Diferenciados (Blake *et al.*, 1998).

Entretanto, em nível de aplicação, ainda são limitados os esforços nesse sentido, não sendo encontradas características de diferenciação de serviços na grande maioria dos servidores web atualmente em operação (Vasiliou & Lutfiyya, 2000). Não é difícil perceber que qualquer esforço para o fornecimento de uma qualidade de serviço diferenciada na Web não poderá ter sucesso se apenas mecanismos em nível de rede e sistema operacional forem utilizados, pois, em última instância, são os servidores web os responsáveis pelo atendimento das solicitações dos usuários. Portanto, precisam estar preparados para reconhecer os diferentes tipos de demanda existentes, adaptando-se para melhor atender cada classe de aplicação.

1.3 Objetivos

O presente trabalho tem como objetivo geral a investigação de alternativas para o fornecimento de serviços diferenciados na Internet em nível de aplicação, particularmente no contexto de servidores web.

Dentre seus objetivos específicos, destacam-se:

a) Modelagem de uma arquitetura global para um servidor web com provisão de serviços diferenciados, que deverá ser capaz de oferecer diferentes níveis de serviço a seus clientes;

b) Identificação de critérios para a classificação dos clientes que chegam a um servidor web, a fim de subdividi-los em classes de serviço;

c) Desenvolvimento de algoritmos para a diferenciação de serviços em servidores web;

d) Desenvolvimento de algoritmos para o controle de admissão de requisições, a fim de evitar a sobrecarga do servidor;

e) Avaliação do impacto da introdução de características de qualidade de serviço em servidores web;

f) Identificação de cenários para a utilização de serviços diferenciados na Web;

g) A partir da experiência adquirida, identificar novos rumos para a pesquisa na área de serviços diferenciados, dando origem a novos projetos.

Esta tese contribui para um melhor entendimento dos aspectos relacionados ao fornecimento de uma qualidade de serviço diferenciada na Web, em especial nos servidores. O uso da modelagem como ferramenta de análise permite abstrair as principais características dos servidores web, com ênfase no suporte a serviços diferenciados, a fim de definir uma arquitetura global para esse fim. Essa arquitetura é incrementada com diversos algoritmos, voltados tanto para a diferenciação de serviços quanto para o controle de sobrecarga do sistema. São identificados também alguns cenários para a utilização de serviços diferenciados na Web.

1.4 Estrutura

O Capítulo 2 desta tese apresenta a infraestrutura da Internet, seus protocolos e principais serviços, destacando-se a Web, com especial atenção ao protocolo HTTP e às aplicações que podem ser executadas sobre a mesma. Discute-se, ainda, a aplicação de técnicas de avaliação de desempenho à Web. São apresentadas as principais formas de organização dos servidores e feita também uma revisão sobre estudos recentes relativos à caracterização de carga na Web. Conclui-se com a apresentação de alguns modelos em redes de filas para servidores web convencionais.

No Capítulo 3, aborda-se o tópico de diferenciação de serviços na Internet e discutem-se as limitações do seu modelo atual de atendimento a clientes. São apresentados conceitos da área de qualidade de serviço e detalhadas as arquiteturas de serviços integrados e diferenciados. É introduzida a necessidade de se fornecer qualidade de serviço na Internet, em nível de aplicação, particularmente nos servidores web, sendo feita uma revisão dos principais trabalhos na área.

O Capítulo 4 apresenta um modelo global para um servidor web com provisão de serviços diferenciados. Descrevem-se os principais componentes do modelo e destacam-se também sua parametrização, forma de validação e a problemática de geração de uma carga de trabalho próxima da realidade. É fornecida uma visão geral dos algoritmos de diferenciação de serviços e de controle de admissão implementados. Finalmente, são apontados alguns cenários de utilização de serviços diferenciados na Web.

O Capítulo 5 inicia com a implementação de alguns algoritmos conhecidos de balanceamento de carga no modelo do servidor diferenciado, comparando seus resultados. Também propõe e implementa um algoritmo de diferenciação de serviços que divide os recursos em partições e os aloca às diferentes classes de serviço, a fim de isolar as questões de desempenho relativas a cada categoria de clientes. São apresentados os principais resultados obtidos e feitas algumas considerações sobre o desempenho e limitações do algoritmo proposto.

No Capítulo 6, o foco reside no emprego de prioridades para a diferenciação de serviços. Implementa-se um algoritmo de prioridades rigoroso nas filas do servidor e são avaliados os efeitos colaterais do mesmo no desempenho das várias classes de serviço. É também proposto e implementado um algoritmo de prioridades adaptativo, o qual permite fazer uma sintonia fina do uso de prioridades, regulando o nível de qualidade de serviço empregado pelo servidor.

O Capítulo 7 aborda os aspectos de controle de sobrecarga em servidores web, motivando sua aplicação também no contexto de serviços diferenciados, como uma forma de melhorar as garantias de qualidade de serviço oferecidas. São propostos e implementados três mecanismos de controle de admissão de requisições no modelo do servidor diferenciado e analisados seus efeitos no atendimento dispensado aos clientes.

Finalmente, o Capítulo 8 apresenta as conclusões e principais contribuições desta tese, bem como aponta caminhos para trabalhos futuros.

Capítulo

2

Infra-estrutura e Desempenho da Web

2.1 Introdução

Dentre as aspirações da humanidade, uma delas seria a capacidade de comunicação com seus semelhantes, a qualquer tempo, a qualquer hora, em qualquer lugar. Assim, surgiram os primeiros mensageiros, os correios, o telégrafo, o sistema telefônico, a televisão. Cada um desses meios atendeu a uma necessidade específica de sua época, com a tecnologia então disponível e muitos deles ainda são largamente utilizados e indispensáveis no mundo atual. Contudo, se é possível apontar uma "falha" nesses meios tradicionais de comunicação, esta seria a não integração das diversas formas de comunicação. Os correios e telégrafos prestam-se à transmissão da palavra escrita; o telefone, à transmissão da palavra falada (e mais recentemente dados); a televisão, à transmissão de áudio e vídeo (pecando, porém, por sua não interatividade). Faltava alguma coisa que realmente revolucionasse a forma das pessoas se comunicarem e conduzirem suas vidas e negócios. Este algo mais, a Internet, finalmente surgiu no final do século passado, causando uma transformação na vida de todos nós.

Embora o uso da Internet tenha se disseminado apenas recentemente, principalmente em termos comerciais, as pesquisas visando a criação de tecnologia que permitisse a interligação de redes e computadores começaram bem antes disso, no início dos anos 70. Atualmente, a Internet tornou-se muito maior do que seus criadores poderiam prever e cada vez mais vem sendo usada para fins comerciais e não apenas para pesquisas acadêmicas. Ela tem crescido exponencialmente desde 1983, dobrando de tamanho a cada ano e, no ano 2000, já contava com mais de 50 milhões de computadores conectados,

distribuídos por mais de 95 mil redes, em mais de 200 países (Comer, 2000; Hunt, 1997). Entretanto, o crescimento da Internet não se deu impunemente. Hoje, nota-se que a rede está sobrecarregada, apresentando sérios problemas de desempenho.

A avaliação de desempenho da Web, mais do que de qualquer outro sistema computacional, representa um grande desafio, tanto em termos do número de elementos que a compõem, quanto da diversidade dos mesmos. Talvez um dos problemas mais críticos da Web, atualmente, seja seu baixo desempenho em muitas situações, embora muito se esteja pesquisando no sentido de melhorar a percepção geral dos usuários quanto à qualidade dos serviços prestados. O presente trabalho objetiva justamente a melhoria dos serviços oferecidos pelos servidores web, que freqüentemente se constituem em um dos gargalos do sistema.

No caso da Web, a avaliação de desempenho deve ser feita em cima de um sistema já existente, de proporções gigantescas, ainda que de invenção recente. Outro fator complicador é o dinamismo da Web, a qual tem mudado bastante e de maneira muitas vezes imprevisível em sua curta história, principalmente em relação à quantidade de tráfego produzido e às aplicações que é possível executar sobre a mesma. Isso faz com que as "verdades absolutas" de hoje não mais se sustentem em um futuro próximo (Floyd & Paxson, 2001).

Neste capítulo, será dada uma visão geral da Internet, destacando-se seus serviços mais comuns e os protocolos da arquitetura TCP/IP que constituem sua infra-estrutura. Será abordada também a World Wide Web, sua forma de organização, os tipos de aplicações que é possível executar sobre a mesma e o protocolo HTTP. Aborda-se, ainda, a aplicação de algumas técnicas de avaliação de desempenho à Web, com destaque para as arquiteturas mais comumente utilizadas na organização de servidores web. É dada especial atenção à caracterização de carga, analisando-se a utilização de carga real e sintética na avaliação de desempenho. São também apresentados alguns modelos para servidores web convencionais propostos na literatura.

2.2 A Internet

2.2.1 Protocolos TCP/IP

Os protocolos TCP/IP foram criados juntamente com a Internet, como solução para a interligação dos computadores nessa rede. O sucesso da Internet fez com que o TCP/IP se tornasse um padrão de fato para a interligação de computadores, tanto em redes locais quanto de longa distância e demonstrou também a viabilidade do uso da arquitetura TCP/IP em larga escala.

Os protocolos TCP/IP se destacam por funcionar sobre uma grande variedade de

arquiteturas de rede, desde cabos coaxiais até redes sem fio, passando por fibras óticas e linhas telefônicas. O TCP/IP permite que as aplicações da Internet sejam escritas com um elevado grau de abstração em relação à forma como a comunicação realmente ocorre, através do uso da API de *sockets* (Stevens *et al.*, 2003), o que facilita o trabalho dos programadores e aumenta a portabilidade das aplicações.

Como exemplo de características que distinguem o TCP/IP de outros conjuntos de protocolos, podem-se citar (Comer, 2000; Hunt, 1997):

- *Uso de padrões abertos.* O TCP/IP está disponível gratuitamente e não está preso a nenhuma plataforma de hardware ou software específica. As especificações dos protocolos TCP/IP estão livremente disponíveis na Internet, através de RFCs (*Request for Comments*), promulgadas pela IETF (*Internet Engineering Task Force*).

- *Independência da tecnologia de rede.* O TCP/IP usa a comutação de pacotes para a comunicação entre as partes e é capaz de funcionar sobre uma variedade de protocolos das camadas de enlace e física, de forma transparente para as aplicações. Por essa razão, é a solução mais escolhida para a interligação de hardware e software diversos.

- *Protocolos padronizados.* Os protocolos TCP/IP são padronizados não apenas no nível de rede e de transporte, mas também em nível de aplicação. Existem, por exemplo, padrões para *e-mail*, *login* remoto e transferência de arquivos, o que torna mais fácil a confecção de novas aplicações.

- *Interconexão total.* Cada dispositivo conectado a uma rede TCP/IP recebe um endereço que o identifica de maneira unívoca e permite que ele se comunique e seja acessível de qualquer outro ponto da rede.

- *Confirmações fim-a-fim.* O TCP/IP fornece um mecanismo de *acknowledgements* entre a origem e o destino final de uma comunicação, em vez de simplesmente entre pares de máquinas ao longo do caminho, o que torna a entrega de dados, como um todo, mais confiável.

2.2.2 Arquitetura TCP/IP

É uma prática comum organizarem-se os protocolos de comunicação em camadas, onde cada camada é responsável por funções bem específicas e possivelmente define um conjunto de protocolos para esse fim. A abstração que se faz é a de cada camada se comunicando virtualmente com a camada de mesmo nível na outra máquina (o seu *peer*), de modo que, em cada nível, a representação dos dados seja equivalente. Uma camada não tem conhecimento de como operam as camadas abaixo e acima de si mesma, precisando apenas conhecer o formato para entrega e recepção dos dados.

A proposta mais conhecida para a organização de software de comunicação é o modelo OSI (*Reference Model of Open Systems Interconnection*) da ISO (*International Organization for Standardization*), que propõe uma arquitetura dividida em sete camadas: Aplicação, Apresentação, Sessão, Transporte, Rede, Enlace de dados e Física (Tanenbaum, 2002).

Os protocolos TCP/IP, embora não sejam um padrão oficial, também apresentam uma arquitetura dividida em quatro camadas: *Aplicação*, *Transporte*, *Internet* e *Acesso à rede*, conforme a Figura 2.1.

Figura 2.1: As camadas da arquitetura TCP/IP

Camada de Acesso à Rede

Esta camada encontra-se no nível mais baixo da hierarquia e fornece a base para a entrega dos dados entre computadores diretamente conectados. Ela utiliza vários protocolos de acesso às camadas inferiores, um para cada tipo de rede disponível. É esta camada que confere a tão conhecida flexibilidade aos protocolos TCP/IP, permitindo-lhes operar em diversos meios físicos. Ela é responsável por funções como o encapsulamento dos datagramas IP nos *frames* usados na rede, assim como a conversão dos endereços IP para o formato da rede. Particularmente em relação à Ethernet, as RFCs 894 (Hornig, 1984) e 826 (Plummer, 1982) tratam, respectivamente, desses dois tópicos.

Camada de Internet

A Camada de Internet é responsável pela comunicação entre uma máquina e outra da rede. Nesta camada, está definido o protocolo IP (*Internet Protocol*), conforme a RFC 791 (Postel, 1981b), o qual é o principal elemento da família TCP/IP, fornecendo o serviço de entrega de pacotes sobre o qual todas as redes TCP/IP são construídas (Hunt, 1997). Toda e qualquer informação em uma rede TCP/IP circula dentro de um datagrama IP, o qual é usado pelos protocolos das camadas superiores.

O protocolo IP, portanto, define o formato do datagrama, que é a unidade básica de transmissão de dados em uma rede TCP/IP, como é o caso da Internet. Também é responsável por definir o esquema de endereçamento da rede, pelo roteamento dos datagramas entre os *hosts* e pela fragmentação e remontagem dos datagramas quando os mesmos passam de uma camada para outra da pilha TCP/IP.

O protocolo IP é do tipo não orientado à conexão, ou seja, cada datagrama carrega informações que lhe permitem ser roteado independentemente dos demais e não é preciso haver uma troca inicial de informações de controle (*handshaking*) para que seja iniciada a comunicação. O IP também é dito *não-confiável*, pois ele não traz nenhum mecanismo de detecção e controle de erros em sua concepção. Dessa forma, serviços de transporte orientados à conexão, bem como controle de erros, seqüência, fluxo e similares deverão ser fornecidos por camadas superiores na arquitetura.

Outro protocolo presente na camada de Internet é o ICMP (*Internet Control Message Protocol*), RFC 792 (Postel, 1981a), que é responsável pela troca de mensagens referentes a controle de fluxo, detecção de erros, redirecionamento de rotas, dentre outras.

Camada de Transporte

A Camada de Transporte fornece um serviço de comunicação fim-a-fim entre dois *hosts* ou, mais precisamente, entre dois processos executando em *hosts* conectados à Internet. Define dois protocolos: o TCP e o UDP.

O protocolo TCP (*Transmission Control Protocol*), descrito na RFC 793 (Postel, 1981c), implementa um serviço de transporte orientado à conexão sobre o protocolo IP, isto é, antes de se iniciar a comunicação, é preciso estabelecer uma conexão lógica fim-a-fim entre os dois *hosts*, o que é feito através de uma troca inicial de mensagens de controle, o chamado *three-way handshake* (Comer, 2000). O TCP fornece um serviço confiável, garantindo que os dados chegarão ao seu destino sem erros e na ordem em que foram enviados. Também faz controle de fluxo, impedindo que uma fonte muito rápida sobrecarregue um destino.

Na visão do TCP, os dados transmitidos são um fluxo (*stream*) de bytes, sem nenhum tipo de delimitador entre eles. Este tipo de abstração é o mesmo encontrado na manipulação de arquivos em linguagens de programação de alto nível e facilita a implementação de certos tipos de aplicações cliente-servidor. O protocolo TCP, por sua confiabilidade, é largamente utilizado na Internet, por ser esta justamente um ambiente propício à ocorrência de erros. Protocolos da camada de aplicação, como o FTP, o TELNET e o HTTP fazem uso dos serviços do protocolo TCP.

O outro protocolo da camada de transporte, o UDP (*User Datagram Protocol*), definido na RFC 768 (Postel, 1980), implementa um serviço de transporte não orientado à conexão sobre o protocolo IP. Aqui, a transmissão de dados pode se iniciar tão logo

seja necessário, eliminando-se a sobrecarga inicial para o estabelecimento da conexão, requerida pelo protocolo TCP. Contudo, não há nenhum tipo de controle de erro, por isso diz-se que o serviço de transporte é do tipo não-confiável. O que de fato acontece é que esses controles são geralmente implementados em nível de aplicação ou, dependendo do caso, são dispensáveis. Os dados transmitidos pelo protocolo UDP são vistos como datagramas (formalmente, são denominados pacotes), que se mapeiam diretamente sobre os datagramas IP, na camada imediatamente inferior.

O UDP é indicado para a construção de aplicações cliente-servidor, principalmente em redes locais (Coulouris *et al.*, 2000). Mensagens DNS (*Domain Name Service*) e SNMP (*Simple Network Management Protocol*), bem como de algumas aplicações de transmissão de áudio e vídeo também circulam utilizando o protocolo UDP.

Camada de Aplicação

Na camada mais alta da arquitetura TCP/IP, encontram-se todos os processos que se utilizam dos serviços da camada de transporte para a entrega dos dados. Esses processos optam por um serviço não orientado à conexão, baseado em mensagens individuais (UDP) ou por um serviço orientado à conexão, que fornece a abstração de um fluxo contínuo de bytes (TCP). Esta camada compreende as aplicações dos usuários e os diferentes protocolos da camada de aplicação, tais como: TELNET, FTP, DNS, HTTP, NFS (*Network File System*), SMTP (*Simple Mail Transfer Protocol*) e outros.

Como foi visto, os protocolos TCP/IP têm a capacidade de interligar redes e computadores baseados em tecnologias diversas, de forma transparente aos usuários, o chamado *internetworking*. A base para o transporte e roteamento dos dados é fornecida pelo protocolo IP e, sobre ele, funcionam protocolos de transporte que proporcionam abstrações de alto nível, bastante convenientes para o desenvolvimento de aplicações. A viabilidade do emprego da arquitetura TCP/IP em larga escala é demonstrada pela Internet e, dada a constante evolução e flexibilidade desses protocolos, a tendência é que eles se firmem cada vez mais como um padrão de fato na área de redes.

De todos os serviços que funcionam sobre a Internet, é de particular interesse para esta tese a World Wide Web, abordada a partir da próxima seção.

2.3 Organização da Web

A World Wide Web, WWW ou simplesmente Web, desde o seu surgimento, no início dos anos 90, tornou-se rapidamente o serviço mais utilizado em toda a Internet. Por volta de 1995, a Web tornou-se responsável pela maior parte do tráfego na Internet (Comer,

2000), superando todos os outros serviços, inclusive aplicações tradicionais como FTP e *e-mail*. Para muitos usuários, principalmente os mais leigos, a Internet é a Web.

Do ponto de vista de sua arquitetura, a Web é um gigantesco sistema de hipertexto em escala global. As informações encontram-se armazenadas em repositórios, denominados servidores web ou HTTP, espalhados ao redor do mundo e os usuários têm acesso às mesmas a partir de programas denominados *browsers* ou navegadores (Orfali *et al.*, 1999). Para as interações entre seus componentes, a Web emprega o paradigma cliente-servidor sobre a infra-estrutura da Internet (Seção 2.4).

Na Web, é clara a separação entre o software de armazenamento da informação e o software de visualização da mesma e, desde seu início, a Web foi concebida para funcionar em ambientes heterogêneos, considerando-se hardware e software. Para um servidor web, é indiferente a plataforma em que se está executando o *browser* cliente e vice-versa; o servidor apenas retorna a informação que lhe é solicitada e o *browser* se encarrega de apresentá-la ao usuário. Esta característica de independência de plataforma, aliás, é observada nas aplicações da Internet de uma maneira geral.

A Web funciona sobre dois padrões que garantem a sua portabilidade: a linguagem HTML e o protocolo HTTP. A HTML (W3C, 1999) é o idioma universal falado na Web: as *tags* HTML descrevem a estrutura do documento, fornecem informações sobre sua formatação e estabelecem os *links* com outros documentos ou recursos da Web. Mais recentemente, surgiu o padrão XML (W3C, 2003) e outros dele derivados, cujo estudo foge ao escopo do presente trabalho. As páginas da Web podem estar fisicamente armazenadas em um servidor ou ser geradas em tempo de execução por meio de algum programa específico (Seção 2.4). Para a comunicação entre os *browsers* e os servidores web, é empregado o protocolo HTTP (Seção 2.5), o qual funciona sobre o TCP.

Nomenclatura dos Recursos

Outro ponto importante na forma como a Web está organizada é seu sistema de nomenclatura baseado em URLs (*Uniform Resource Locators*), as quais identificam de forma não ambígua um recurso ou objeto qualquer existente na Internet, não somente páginas HTML.

Uma URL apresenta a seguinte estrutura, conforme a Figura 2.2:

- *Protocolo*. A parte inicial da URL informa o protocolo utilizado para o transporte dos dados (HTTP, FTP, MAILTO, NNTP e outros). O protocolo escolhido influencia a maneira como o restante da URL é interpretado.
- *Nome do servidor*. É um nome de *host* válido, podendo também ser utilizado um endereço IP.

- *Número da porta.* Especifica a porta em que um processo no servidor está aguardando por mensagens. Caso não seja informado, é utilizado algum valor default (porta 80 para HTTP, 21 para FTP e assim por diante).
- *Localização do recurso.* É o caminho (*path*) até o recurso. Em se tratando de interações HTTP, representa o caminho para o arquivo solicitado na árvore de diretórios do servidor web.

Figura 2.2: Estrutura de uma URL

2.4 Interações Cliente-Servidor na Web

A Internet e, em particular, a Web pode ser considerada a maior plataforma cliente-servidor da atualidade. Elas demonstram a viabilidade do emprego dos protocolos TCP/IP e do paradigma cliente-servidor em larga escala, numa rede mundial.

Um servidor web é um processo que fica permanentemente aguardando por solicitações vindas dos clientes. Ao chegar uma requisição, ele a interpreta e, caso a mesma seja válida, retorna ao cliente o objeto solicitado (em geral, um documento HTML). Este pode estar fisicamente armazenado no sistema de arquivos do servidor ou ser gerado dinamicamente por um programa ou *script* invocado no servidor.

Nesta seção, pretende-se analisar a Web sob o ponto de vista das aplicações que é possível executar sobre a sua estrutura, destacando-se as funcionalidades encontradas em cada caso.

2.4.1 Páginas Estáticas

No seu início, a Web nada mais era que um sistema de hipertexto em escala mundial, contendo milhares de documentos interligados, alguns deles constituídos por outras mídias, além de simples texto. À época, os principais elementos que forneciam o suporte para o funcionamento da Web eram os *browsers*, o protocolo HTTP, os servidores web, os documentos HTML e o sistema de nomenclatura de *hosts* e documentos (as URLs). Para

efeito de simplificação, pode-se encarar a Web, neste seu primeiro momento, como um servidor de arquivos universal baseado em URLs (Orfali *et al.*, 1999).

A Figura 2.3 apresenta uma interação ou transação HTTP típica. Nela, um *browser* contacta um servidor web quando o usuário especifica uma URL a ser alcançada, implícita ou explicitamente(1). Então, o *browser* faz uma requisição ao servidor(2), via protocolo HTTP. O servidor, ao recebê-la através de um *socket* escutando numa porta pré-definida (geralmente a de número 80), estabelece uma conexão com o cliente; em seguida, procura em seu disco o arquivo solicitado e o envia ao cliente, fechando a conexão(3). O *browser*, ao receber a resposta, examina o conteúdo do arquivo e, se for o caso, interpreta os comandos HTML e mostra o documento formatado ao usuário que o solicitou(4).

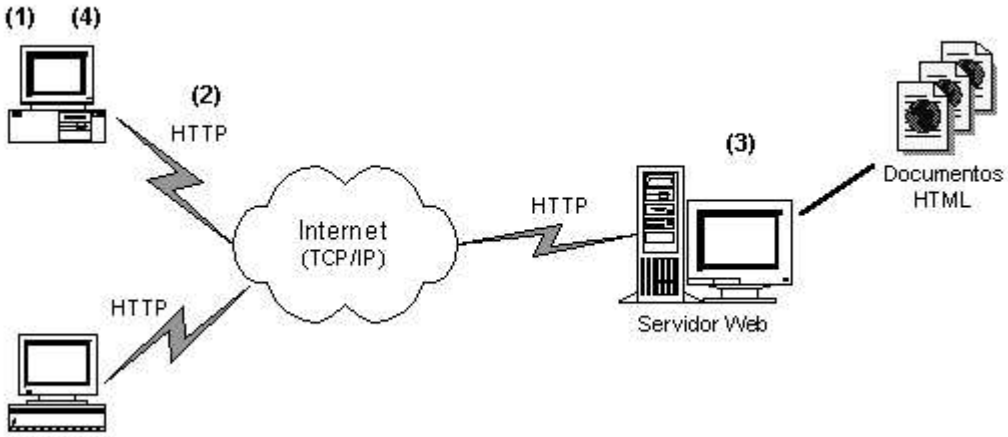


Figura 2.3: Interação cliente-servidor na Web

As interações cliente-servidor, neste caso, são de duas camadas (*two-tier*), com clientes "leves", os *browsers*, fazendo acesso a uma infra-estrutura de servidores de documentos HTML. Esta fase inicial da Web apresenta uma baixa funcionalidade e pouca interatividade com o usuário, já que os recursos solicitados ainda são, essencialmente, documentos HTML fisicamente armazenados no sistema de arquivos do servidor web.

A abordagem aqui descrita baseia-se no que é chamado de *páginas estáticas*, as quais ainda estão presentes em boa parte das interações que ocorrem na Web atualmente.

2.4.2 Páginas Dinâmicas

No final de 1995, surgiu o protocolo CGI (*Common Gateway Interface*), o qual introduziu uma maior interatividade na Web, permitindo que, a partir de um *browser*, se pudesse iniciar uma aplicação do lado servidor (Yeager & McGrath, 1996).

Toda a comunicação entre o *browser* e o servidor web continua acontecendo em formato HTML, o que garante a independência de plataforma. O que o CGI faz, na verdade,

é transferir o pedido de execução de aplicação para o programa apropriado, também localizado no lado servidor, agindo como um tradutor entre o código HTML enviado pelo cliente e os requisitos específicos de cada aplicação, que pode ser desde um servidor de *e-mail* ou FTP até um banco de dados ou um complexo sistema empresarial. A porta de entrada para as aplicações em CGI são os formulários HTML, que recebem os parâmetros digitados pelo usuário, no *browser*.

A Figura 2.4 apresenta uma transação HTTP envolvendo CGI (Orfali *et al.*, 1999). Nela, o programa servidor, ao receber a solicitação(1), a executa e retorna os resultados ao módulo CGI(2), em formato HTML, o qual os repassa ao cliente(3). Para o usuário, o efeito é o de ter acessado uma página HTML estática armazenada no servidor, só que, na verdade, a mesma foi gerada dinamicamente a partir de um processo iniciado sob o comando do servidor web. Este processo, denominado de programa ou *script* CGI, é independente do servidor, ou seja, executa em um outro espaço de endereçamento e com escalonamento próprio, sob o ponto de vista do sistema operacional. O que o protocolo CGI define é justamente o formato das informações trocadas entre o *browser* e o programa que foi acionado. Esta abordagem emprega o modelo de *páginas dinâmicas*, cuja utilização tem crescido cada vez mais na Web.

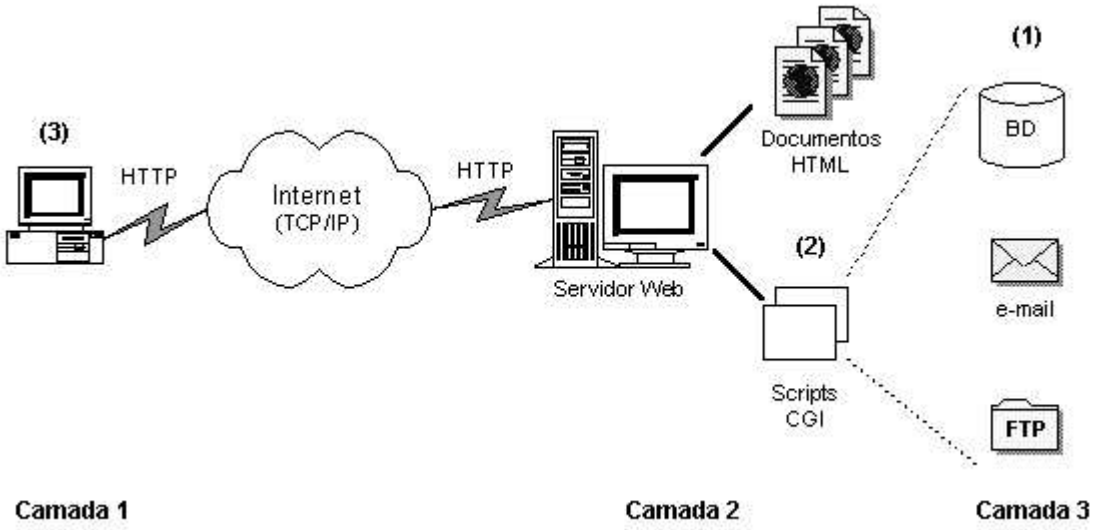


Figura 2.4: Interação cliente-servidor baseada em páginas dinâmicas

As interações cliente-servidor descritas acima são de três camadas (*three-tier*), com a camada intermediária constituída pelo servidor web (acrescido do módulo CGI) e as camadas externas representadas pelo *browser* cliente e pelas aplicações servidoras de propósitos específicos.

Embora o CGI torne possível a geração de páginas dinâmicas na Web e seja uma das soluções mais utilizadas atualmente para o suporte a aplicações nesse meio, ele é um protocolo *stateless*, que apresenta um alto tempo de resposta, além de ter a tendência de

sobrecarregar o servidor, pois, para cada solicitação feita, é iniciado um novo processo para tratá-la (Orfali *et al.*, 1999). Com o tempo, foram surgindo algumas soluções que procuram minimizar as desvantagens do CGI, preocupando-se com a falta de estado no servidor, tentando compartilhar processos em memória entre as invocações de serviços e procurando introduzir mais interatividade no cliente. Exemplos delas são: NSAPI (*Netscape Server API*), ASP (*Active Server Pages*), *Cold Fusion*, PHP (*Hypertext Preprocessor*), *servlets* Java, objetos distribuídos, *scripts* do lado cliente, HTML dinâmico e outras. Entretanto, algumas dessas novas soluções pecam por serem proprietárias, portanto não universais.

Para concluir, cabe acrescentar que a divisão aqui apresentada teve apenas finalidade didática, pois nenhuma das técnicas analisadas necessariamente exclui as outras e o que se tem atualmente, na Web, é o uso de várias soluções em conjunto, de acordo com a necessidade. O conhecimento dessas técnicas é importante para este trabalho, pois cada uma delas tem implicações particulares no desempenho geral de um servidor web.

2.5 Protocolo HTTP

2.5.1 Visão Geral

O protocolo HTTP (*Hypertext Transfer Protocol*) é o elemento que une as partes que compõem a Web. Toda e qualquer informação que trafega entre os *browsers* e os servidores web o faz dentro de uma mensagem HTTP. A exceção seriam as interações baseadas em objetos distribuídos (Dos Santos *et al.*, 2002), mas que, por constituírem uma parcela pequena das transações na Web, não serão comentadas neste trabalho.

O HTTP é o protocolo usado para fazer acesso aos recursos armazenados no servidor web. Ele é um protocolo em nível de aplicação, que assume a existência de um serviço de transporte confiável, orientado à conexão, como o TCP (Comer, 2000). Uma transação HTTP se desenrola segundo um mecanismo do tipo requisição/resposta (*request/reply*), conforme explicado na Seção 2.4.1. O HTTP é um protocolo do tipo *stateless*, ou seja, o servidor não guarda nenhuma informação em relação ao estado dos clientes; se houver alguma falha na execução da tarefa, é responsabilidade do cliente submeter novamente a transação, fornecendo todas as informações necessárias para completá-la.

É importante destacar que, embora sob o ponto de vista do usuário, uma solicitação de página HTML se resuma, muitas vezes, a apenas um clique do mouse, na verdade ela freqüentemente dá origem a várias solicitações HTTP que são enviadas do *browser* para o servidor web (seguidas de suas respectivas respostas no sentido contrário). Isso ocorre porque, para cada objeto contido em uma página HTML, é gerada uma requisição independente ao servidor. No protocolo HTTP 1.0, isso implica em estabelecer uma nova conexão TCP para cada objeto solicitado, o que pode levar a uma sobrecarga desnecessária

no servidor e na rede (Seção 2.5.3).

A representação dos dados no protocolo HTTP é feita segundo o padrão MIME (*Multi-Purpose Internet Mail Extensions*), definido na RFC 1521 (Borenstein, 1993), também utilizado no protocolo SMTP para a codificação dos dados nas mensagens de correio eletrônico. Antes de iniciar uma transação, o *browser* e o servidor web geralmente negociam os tipos de dados que serão trocados, o que também contribui para a sobrecarga inerente ao HTTP.

2.5.2 Mensagens de Requisição e Resposta

O HTTP define um formato padrão para as mensagens de requisição e resposta. Uma requisição tipicamente consiste de uma linha informando a ação a ser executada no servidor, seguida de uma ou mais linhas de cabeçalho (os parâmetros) e pelo corpo da mensagem (opcional), como descrito a seguir:

- *Request line.* Contém o método HTTP invocado (a ação), a localização do objeto no servidor e a versão do protocolo HTTP utilizada.
- *Request header.* O cabeçalho contém campos que o cliente pode utilizar para enviar informações ao servidor, por exemplo, para informar os tipos de dados que ele é capaz de aceitar, numa espécie de negociação.
- *Corpo da mensagem.* Às vezes é utilizado quando o cliente precisa passar dados adicionais ao servidor, como no caso do método POST.

No exemplo da Figura 2.5, o cliente solicita (GET) o arquivo `/docs/arq.html` do servidor `www.empresa.com` usando o protocolo HTTP 1.1. O campo Accept no cabeçalho informa que o cliente sabe como tratar texto em formato HTML (text/html) e imagens em formato JPEG (image/jpeg). O campo User-Agent informa ao servidor qual o tipo do *browser*.

```
GET /docs/arq.html HTTP/1.1
Host: www.empresa.com
Accept: text/html, image/jpeg
User-Agent: Mozilla
```

Figura 2.5: Exemplo de uma requisição HTTP

Para as respostas, o HTTP determina que as mensagens devem conter uma linha de status, seguida de um ou mais campos e do corpo da mensagem, precedido por uma linha em branco, como descrito a seguir:

- *Response header*. Contém a versão do protocolo, o código de status e uma explicação do mesmo.
- *Request header*. Vários campos que informam as características do servidor e do objeto retornado para o cliente.
- *Corpo da mensagem*. Contém o objeto retornado, na maioria das vezes um documento HTML.

A Figura 2.6 mostra um exemplo de uma resposta HTTP 1.1 válida. O código 200 reporta que a requisição foi bem-sucedida. O campo Server informa o tipo do servidor web (CERN 3.0) e o campo Content-Type diz que o objeto retornado é um documento HTML, cujo tamanho é de 150 bytes (Content-Length). Finalmente, tem-se uma linha em branco e o documento propriamente dito.

```
HTTP/1.1  200  OK
Server: CERN/3.0 libwww/2.17
MIME-Version: 1.0
Content-Type: text/html
Content-Length: 150

<HTML> . . . </HTML>
```

Figura 2.6: Exemplo de uma resposta HTTP

Métodos HTTP

O protocolo HTTP define um conjunto de métodos que o cliente pode invocar, que funcionam como comandos enviados ao servidor web. O protocolo HTTP 1.0, descrito na RFC 1945 (Berners-Lee *et al.*, 1996), define os métodos GET, HEAD e POST. O HTTP 1.1, RFC 2616 (Fielding *et al.*, 1999), acrescenta a esse conjunto os métodos OPTIONS, PUT, DELETE, TRACE e CONNECT. A Tabela 2.1 apresenta um resumo dos métodos e suas finalidades.

Códigos de status

As requisições HTTP devem sempre retornar um código de status que informa o resultado de sua execução. Desse modo, têm-se o código 200 (OK), o 404 (*Not Found*), o 503 (*Service Unavailable*), dentre outros. Esses códigos são divididos em classes, conforme mostra a Tabela 2.2.

Método	Finalidade
GET	Solicita que seja retornado o recurso identificado pela URL
HEAD	Obtém informações sobre o recurso sem que o mesmo seja retornado ao cliente. Testa validade de *links*, acessibilidade e a data da última modificação
POST	Envia informações adicionais do cliente para o servidor no corpo da mensagem, por exemplo, dados digitados em formulários HTML
OPTIONS	Obtém as opções de comunicação disponíveis ou os requisitos associados ao recurso solicitado, sem necessariamente iniciar sua recuperação
PUT	Permite criar ou modificar um recurso no servidor web
DELETE	Solicita que o recurso identificado pela URL seja apagado do servidor web
TRACE	É usado para enviar uma mensagem de teste, do tipo *loopback*, ao servidor
CONNECT	Reservado para comunicação com servidores *proxy*

Tabela 2.1: Métodos do protocolo HTTP

Classe	Descrição
1xx	Finalidade informativa
2xx	Sucesso
3xx	Redirecionamento
4xx	Erro do cliente
5xx	Erro no servidor

Tabela 2.2: Classes de resposta do protocolo HTTP

2.5.3 HTTP 1.0 e 1.1

O protocolo HTTP 1.0 foi introduzido juntamente com a Web, em 1990. Nesta versão inicial, ele nada mais era que um modo simples de recuperar dados através da Internet. Com o crescimento da Web, surgiram novos requisitos e algumas de suas fraquezas foram aparecendo.

A primeira delas é que o HTTP 1.0 exige o estabelecimento de uma nova conexão TCP para cada objeto solicitado. No início, em que os documentos da Web eram constituídos basicamente de texto, isso não chegava a ser um problema, porém, atualmente, em que uma simples página HTML pode conter dezenas de pequenas imagens, isso tende a causar uma grande sobrecarga no tráfego da Internet, bem como nos servidores. O protocolo HTTP 1.1, padronizado em 1999 pelo W3C (*World Wide Web Consortium*), usa como *default* o esquema de conexões persistentes, que permite que uma mesma conexão TCP seja usada por várias transações HTTP, o que é bem mais eficiente.

O HTTP 1.1 também permite fazer o *pipelining* de requisições. Neste caso, várias requisições são enviadas em seqüência, sem aguardar pelas respostas. Isso é bastante útil, por exemplo, para recuperar as imagens de uma página, principalmente em ambientes que possuam uma alta latência para o estabelecimento de conexões TCP.

Ultimamente, tem se disseminado o uso de caches na Web, como forma de diminuir a latência no acesso aos servidores, bem como o tráfego na rede devido a transferências de arquivo desnecessárias. Em reconhecimento a isso, nesta nova versão do HTTP também foram incluídos comandos específicos para a manipulação de caches, tanto pelos servidores (web e *proxy*) quanto pelos clientes.

2.6 Arquitetura de Servidores Web

Em uma visão de alto nível, pode-se considerar a Web como sendo composta por três elementos principais: os *browsers*, a rede e os servidores. No contexto deste trabalho, é muito importante um bom entendimento da arquitetura dos servidores web, pois, em última análise, são eles que atendem as requisições dos usuários, podendo se tornar um dos gargalos do sistema quando sobrecarregados.

Na seção 2.4, foram apresentadas as ações tipicamente tomadas por um servidor web para atender uma requisição de um documento HTML, seja utilizando páginas estáticas ou dinâmicas. Um servidor web está sempre em um laço infinito, permanentemente aguardando por requisições dos clientes. Nesta espera, existem alguns atrasos que são inevitáveis, como a espera pela transmissão dos dados na rede, o acesso ao disco do servidor, o escalonamento dos processos pelo sistema operacional, entre outros. O servidor, portanto, deve ser projetado de modo a atender o maior número de requisições que lhe seja possível. As diversas arquiteturas apresentadas a seguir procuram justamente melhorar o nível de concorrência que é possível alcançar no servidor.

2.6.1 Formas de Organização

Servidor Iterativo

Este seria o tipo de servidor mais simples, que apenas fica aguardando as requisições dos clientes e as trata uma por vez, na ordem de sua chegada, sem nenhum tipo de concorrência. Sua inclusão aqui se dá apenas por finalidade didática, pois, na prática, seria um servidor muito ineficiente.

Processo por Requisição

Esta é uma das formas mais comuns para a construção de servidores concorrentes. Neste caso, existe um processo principal (o "pai") que fica aguardando requisições em uma porta pré-definida. Ao chegar uma nova requisição, ele cria uma cópia de si mesmo e passa a esse "filho" a responsabilidade de tratar a solicitação, retornando em seguida ao seu laço de espera. No UNIX, a criação dos processos filhos é feita por meio de uma chamada à *system call* fork() (Stevens *et al.*, 2003), a qual gera um novo processo que é uma cópia do pai, possuindo o mesmo código executável e o mesmo conteúdo da memória, sendo-lhe passado um identificador da conexão de rede que foi estabelecida com o cliente. Este novo processo é executado e escalonado independentemente do processo pai e, ao terminar de atender a solicitação do cliente, encerra sua execução e deixa de existir.

Nessa arquitetura, em um determinado momento, existirão inúmeras cópias do servidor web executando concorrentemente, limitadas apenas pela capacidade da máquina que as abriga. Evidentemente, existe um limite sensato para o número de processos concorrentes no servidor, sob pena de bloquear sua execução, pois cada processo filho ocupa memória da máquina e ciclos de CPU, além de possuir uma conexão de rede aberta com um cliente. Por essa razão, alguns servidores web limitam o número de processos filhos que podem executar concorrentemente (Yeager & McGrath, 1996).

A criação de um novo processo também é uma operação cara em termos computacionais e, de certa forma, desnecessária neste caso, pois as transações web são simples e não precisariam de um processo individual para sua execução. Estudos mostram que um fork() pode representar até 15 por cento do tempo total de execução no caso de requisições de documentos pequenos (Yeager & McGrath, 1996). Entretanto, a simplicidade e naturalidade de programação desta solução, principalmente em ambientes UNIX, explica sua popularidade na construção de servidores dos mais diversos tipos.

Pool de Processos

Esta solução tenta aproveitar a simplicidade da abordagem anterior, ao mesmo tempo em que procura eliminar suas desvantagens. Neste caso, o servidor, ao iniciar sua execução, cria um número mínimo de processos, que constituem o *process pool* (Hu *et al.*, 1997). Um processo *dispatcher* fica permanentemente aguardando solicitações dos clientes e, ao receber uma requisição, escolhe algum dos processos ociosos para tratá-la, retornando ao seu laço de espera. Os processos do *pool*, após atenderem uma requisição, não encerram sua execução; eles ficam de prontidão, esperando até que sejam convocados para atuar novamente. Esta solução é usada por servidores web como o Apache (Apache Software Foundation, 2002).

Com isso, consegue-se eliminar a sobrecarga da criação dos processos, porém o *dispat-*

cher pode se tornar o gargalo do servidor, além de um ponto crítico de falha do mesmo, pois todas as requisições têm que passar inicialmente por ele. Além disso, pode ser difícil determinar o número ótimo de processos no *pool* e, no caso de a carga ser muito alta, o servidor pode ser forçado a criar novos processos, recaindo-se no caso da abordagem anterior.

Thread por Requisição

Nesta abordagem, para cada nova requisição que chega é criada uma nova *thread* (linha de execução) para tratá-la. A vantagem é que uma *thread* consome muito menos recursos da máquina que um processo, pois diferentes *threads* podem compartilhar o mesmo espaço de endereçamento, incluindo código e dados globais. E, como as *threads* se encontram dentro de um mesmo processo, a mudança de contexto entre elas também se dá de forma mais rápida.

A utilização de *threads* é bastante indicada para o caso de tarefas que freqüentemente precisam esperar por I/O, como acontece com os servidores web, pois é preciso aguardar pela leitura dos dados no disco e também pela recepção e envio dos pacotes pela rede. Enquanto uma *thread* espera por I/O, o controle é passado para outra *thread*, responsável por uma requisição diferente, o que acaba resultando numa utilização bem mais eficiente dos recursos da máquina. O servidor PHTTPD usa esta abordagem de *thread* por requisição (Hu *et al.*, 1998).

De maneira geral, o uso de *threads* não só torna o servidor web mais rápido, mas também permite que ele execute mais requisições concorrentemente, o que torna esta solução bastante atrativa para a construção dos servidores atuais. Entretanto, a programação com *threads* é bem mais complexa do que com processos e seu código, por vezes, faz uso de bibliotecas específicas de um dado sistema operacional, o que pode comprometer sua portabilidade (Yeager & McGrath, 1996).

Pool de Threads

Finalmente, outra alternativa possível é a utilização de um *pool* de *threads*, que são criadas quando o servidor web é iniciado. Dessa forma, elimina-se a sobrecarga de criação de uma nova *thread* para cada requisição que chega ao servidor, a exemplo do caso da utilização do *pool* de processos, além de permitir um controle refinado do nível de multiprogramação utilizado. O servidor JAWS (Hu *et al.*, 1997) emprega esta abordagem.

2.6.2 Características Especiais

À primeira vista, pode-se sentir tentado a considerar um servidor web como um mero servidor de arquivos, com mais algumas funcionalidades incorporadas. Esta, entretanto, seria uma visão simplificada, podendo levar a inferências errôneas sobre seu comportamento e desempenho. O bom desempenho de um servidor web, como de qualquer outro tipo de servidor, depende da combinação de hardware e software escolhida. Por software, entenda-se o sistema operacional da máquina, o software servidor propriamente dito e o conteúdo à disposição dos usuários.

Alguns pontos chamam a atenção na avaliação de desempenho de servidores web. Por exemplo, a população de usuários de um servidor web é potencialmente ilimitada e, a priori, as solicitações podem vir de qualquer lugar do mundo, principalmente no caso de servidores de provedores comerciais ou de sites de comércio eletrônico. Os usuários também fazem acesso aos servidores segundo padrões muitas vezes imprevisíveis: um servidor pode passar por longos períodos em que é submetido a uma carga relativamente baixa e, repentinamente, ser alvo de uma carga muito superior a sua capacidade, prejudicando o atendimento das solicitações.

A variabilidade no tamanho dos objetos armazenados em um servidor web também é muito grande, podendo oscilar entre uma centena e alguns milhões de bytes (Menascé & Almeida, 1998). O perfil de um servidor otimizado para servir documentos pequenos deve ser diferente de outro voltado, por exemplo, para o fornecimento de conteúdo multimídia. Este, aliás, é outro aspecto a ser considerado: a diversidade dos tipos de objetos existentes em um servidor web, que podem abranger desde simples arquivos texto e páginas HTML até figuras, áudio e vídeo, casos que exigem um tratamento diferenciado. Os usuários, muitas vezes, nem têm consciência de que, ao solicitar uma página HTML, estão, na verdade, requisitando indiretamente todos os outros objetos contidos na mesma, os quais são solicitados individualmente ao servidor web. Dependendo do caso e da versão do protocolo HTTP utilizada, o desempenho do servidor pode ser comprometido de diferentes formas.

Para complicar ainda mais a situação, os servidores web vêm, cada vez mais, sendo usados como servidores de aplicação, principalmente em sites de comércio eletrônico. Este é o caso da utilização de páginas dinâmicas (seção 2.4.2), em que o servidor tem que dar suporte à execução de programas de aplicação que irão gerar dinamicamente as respostas às requisições dos clientes.

Por essas e outras razões, novos modelos para servidores web têm sido propostos na literatura e diversas pesquisas têm sido feitas visando caracterizar a carga presente na Web atualmente.

2.6.3 Métricas de Desempenho

Antes de se prosseguir com a apresentação de modelos de servidores web e de carga, é preciso definir bem quais métricas de desempenho se aplicam neste caso e quais os critérios para se afirmar que um dado servidor tem um bom desempenho.

Em primeiro lugar, a idéia do que vem a ser um bom desempenho para um sistema não é um conceito exato. O que para uns pode parecer uma característica favorável, para outros pode não ter tanta relevância: tudo depende de quem observa. Menascé & Almeida (1998) apontam que, para um usuário da Web, a percepção do desempenho de um sistema está relacionada com respostas rápidas e ausência de erros de conexão. Para o administrador do sistema, por outro lado, o desempenho relaciona-se com um alto *throughput* e uma alta disponibilidade do sistema. Uma boa arquitetura de servidor web deverá ser capaz de atender às expectativas de ambos os grupos.

As principais métricas que buscam quantificar o desempenho de um servidor web são a latência e o *throughput* (Menascé & Almeida, 1998). A latência é definida como o tempo necessário para completar uma requisição ou serviço (Jain, 1991). No âmbito da Web, por exemplo, o tempo de resposta visto pelo cliente inclui o tempo de transmissão da requisição pela rede (latência de rede), o seu processamento pelo servidor (latência do servidor), o retorno dos dados ao cliente e, finalmente, a apresentação do resultado ao usuário final (latência do cliente). Em diferentes situações, um desses recursos pode se tornar o gargalo do sistema (geralmente, o servidor ou a rede), tendo a sua latência um papel preponderante no cômputo da latência global. É importante identificar o recurso que representa o gargalo do sistema, pois ele é o primeiro ponto sobre o qual se deve agir para melhorar o desempenho do mesmo (Eggert & Heidemann, 1999).

O *throughput* de um servidor é geralmente expresso em número de transações HTTP completadas por segundo. Também se pode analisá-lo segundo o número de conexões estabelecidas com o servidor por segundo ou o número de bits transferidos por segundo, conforme a situação. Observe-se que os objetivos buscados podem ser conflitantes: um servidor que tenha um alto *throughput*, atendendo vários clientes concorrentemente, provavelmente o alcançará às custas de uma maior latência média para completar as requisições.

Finalmente, há que se considerar o número de erros por segundo, pois uma alta taxa de erros geralmente indica que o servidor está sobrecarregado e, conseqüentemente, seu desempenho tende a cair bruscamente. Não é incomum, na Web, que servidores sejam submetidos a cargas muito mais altas do que são capazes de suportar, o que resulta numa degradação considerável da qualidade do serviço oferecido aos clientes.

2.7 Caracterização de Carga da Web

O desempenho da Web não depende apenas da maneira como são construídos os servidores web. Como em qualquer sistema distribuído, seu desempenho está bastante relacionado com as características da carga colocada sobre o mesmo. Para se fazer uma avaliação de desempenho de um sistema, seja ele qual for, é de suma importância caracterizar-se bem o tipo de carga a que ele é normalmente exposto.

É possível analisar o comportamento de um sistema submetendo-o a uma carga real, por exemplo, um *trace* da demanda pelos recursos armazenados no servidor. Contudo, algumas vezes, é mais interessante construir um modelo da carga de trabalho do sistema, da mesma forma que se constrói um modelo do mesmo, pois assim é possível sumarizar e reproduzir a informação necessária para descrever a carga de trabalho, sem a necessidade de continuamente realizar experimentações sobre o sistema real. A escolha de quais características representar na carga de trabalho vai depender do enfoque que se pretende dar à avaliação de desempenho.

Calzarossa & Serazzi (1993) discutem algumas diretrizes gerais que se podem aplicar no processo de análise e caracterização da carga de trabalho de um sistema. Em primeiro lugar, segundo eles, é preciso definir qual carga de trabalho se pretende analisar, pois, em um ambiente distribuído, encontram-se diferentes tipos. Para uma máquina cliente, em que um usuário manipula um *browser*, a carga apresentada são os cliques dados pelo usuário e as respostas vindas dos servidores web. Já do ponto de vista do servidor, a carga são as requisições HTTP que chegam dos clientes. Na visão da rede, a carga são os pacotes que circulam na mesma.

É igualmente importante definir o nível de detalhamento empregado na descrição da carga. Por exemplo, a análise do número de transações web completadas por segundo ou do número médio de ciclos de CPU para atender uma requisição implicam em visões diferentes do mesmo problema. A escolha feita neste ponto vai determinar que tipos de medidas de desempenho serão colhidas.

Também é fundamental identificar os componentes básicos da carga de trabalho de um sistema, ou seja, qual a unidade genérica de trabalho que chega ao sistema a partir do ambiente externo (Menascé & Almeida, 1998). Esta pode ser um *job*, uma transação de banco de dados, um comando digitado por um usuário ou, no caso da Web, uma requisição HTTP.

Os recursos do sistema considerados na caracterização de carga devem ser aqueles cuja utilização tenha o maior impacto no desempenho do sistema (Ferrari *et al.*, 1983). No caso de um servidor web, a CPU e o sistema de I/O são os principais recursos a serem analisados, pois é sobre eles que recai a maior parte do processamento.

Finalmente, precisam ser determinados os parâmetros que melhor caracterizam cada

componente da carga de trabalho. Há parâmetros que caracterizam a intensidade da carga (taxa de chegada, número de clientes, número de processos concorrentes) e os que se referem à demanda de serviço colocada sobre o sistema (tempo de CPU, de I/O). Uma vez determinados os parâmetros, deve-se proceder a coleta dos dados, fase que pode se tornar uma das mais demoradas e trabalhosas, principalmente no caso de sistemas muito complexos, como a Web, o que pode dificultar a sumarização dos dados. Uma das estratégias para lidar com isso consiste em dividir a carga em grupos com características semelhantes, tentando-se, assim, minimizar os efeitos da heterogeneidade na concepção do modelo.

2.7.1 Partição da Carga de Trabalho

Vários critérios podem ser utilizados com o intuito de dividir a carga de trabalho da Web. A idéia consiste em organizar a carga em classes com características similares, de modo que se possa estudar cada classe como um todo homogêneo, diminuindo-se a variabilidade das medidas e aumentando-se a significância estatística dos resultados.

Menascé & Almeida (1998) destacam algumas possibilidades, comentadas a seguir. Um critério a ser considerado pode ser o consumo de recursos do sistema: sob este prisma, seria possível dividir as requisições HTTP segundo seu consumo esperado de CPU e I/O.

A carga de trabalho também pode ser organizada em classes de acordo com o tipo de aplicação que a originou. Na Internet, têm-se desde requisições de simples páginas web e sessões de *ftp* até a recuperação de complexos conteúdos multimídia e acesso a bancos de dados. Cada uma dessas aplicações impõe diferentes demandas sobre o sistema e gera, como resultado, quantidades variadas de tráfego.

Os tipos (ou tamanhos) dos objetos solicitados são outro critério interessante. Na Web, encontram-se requisições de objetos que vão desde simples texto ou páginas HTML a imagens, áudio, vídeo, documentos formatados, páginas dinâmicas e outros. Uma requisição de uma página HTML de alguns poucos kilobytes impõe uma demanda sobre um servidor web que é muito diferente daquela originada, por exemplo, pelo *download* de um vídeo com alguns megabytes. Em geral, essas situações não devem ser analisadas em conjunto.

Outro ponto que se pode considerar é a origem geográfica das requisições que chegam a um servidor, o que pode ser determinado analisando-se os endereços IP dos clientes. Enfim, inúmeros critérios de classificação podem ser propostos; cabe ao analista identificar e implementar aqueles que sejam significativos para o estudo em questão.

2.7.2 A Busca de Invariantes

Como já comentado, para que se possa realizar uma avaliação de desempenho de um sistema é fundamental que se conheçam as características da carga de trabalho colocada sobre o mesmo. Isso é especialmente verdadeiro no caso de sistemas distribuídos complexos, como é o caso da Web. Pela sua enormidade e heterogeneidade, a Web se apresenta como um formidável desafio para aqueles que tentam compreendê-la e, principalmente, fazer previsões sobre seu futuro tomando como base seu passado recente.

Um dos primeiros trabalhos que se preocupou em identificar aquilo que se convencionou chamar de *invariantes* na carga de trabalho da Web se deve a Arlitt & Williamson (1996). Uma invariante é uma característica da carga de trabalho que potencialmente representaria uma verdade universal, aplicável a todo um sistema.

No estudo citado, os autores utilizaram seis registros (*logs*) de acesso a servidores web: três de ambientes acadêmicos, dois de instituições de pesquisa e um de um provedor comercial. Os conjuntos de dados representam três ordens de grandeza diferentes na utilização de servidores web, variando de menos de mil requisições por dia a mais de 350 mil requisições por dia e os tempos considerados para análise variaram de uma semana a um ano de atividade. Foram encontradas dez invariantes, as quais são apresentadas na Tabela 2.3.

Pode-se notar que o grau de sucesso nas requisições ao servidor é grande, pois em quase 90% dos casos o documento procurado de fato existe. A variabilidade de tipos de arquivo foi levada em conta no estudo: os arquivos foram classificados nas categorias HTML, imagens, som, vídeo, formatados e dinâmicos, segundo os sufixos usados nos seus nomes. Ainda assim, a segunda invariante mostra que mais de 90% das referências são para arquivos HTML ou imagens. Note-se que este estudo foi feito em 1996, quando o número de requisições dinâmicas na Web ainda era bem reduzido. O trabalho de Pinheiro (2001), mais recente, embora tenha analisado registros de servidores diferentes, concluiu que as referências para documentos HTML e imagens caíram para menos de 70% do total, tendo havido um aumento significativo nas requisições de páginas dinâmicas, que chegam a mais de 15%, em média.

Outra conclusão interessante é que a distribuição do tamanho dos arquivos obedece a uma distribuição de Pareto (comentada a seguir), fato também observado em outros trabalhos. Também se concluiu que os tempos entre as referências aos arquivos são exponencialmente distribuídos e independentes, em todos os seis registros analisados, com a média dependente da carga no servidor.

Nota-se também que um terço dos arquivos são acessados uma única vez e que 10% dos arquivos respondem por 90% das requisições e dos bytes transferidos, fato que não deixa de ser surpreendente, mas que é explicado pela Lei de Zipf (Zipf, 1949). A concentração geográfica das origens das requisições também é curiosa: a maior parte delas vem de sites

Invariante	Nome	Descrição
1	Taxa de sucesso	Taxa de sucesso para requisições ao servidor foi de aproximadamente 88%
2	Tipos de arquivo	Arquivos HTML e imagens representam 90 a 100% das requisições
3	Tamanho médio de transferência	Tamanho médio menor que 21 kbytes
4	Requisições distintas	Menos de 3% das requisições são para arquivos distintos
5	Referências únicas	Aproximadamente um terço dos arquivos e bytes são acessados uma única vez
6	Distribuição dos tamanhos	A distribuição do tamanho dos arquivos é Pareto com $0,40 < \alpha < 0,63$
7	Concentração de referências	10% dos arquivos acessados correspondem a 90% das requisições ao servidor e a 90% dos bytes transferidos
8	Tempos entre referências	Os tempos entre as referências aos arquivos são exponencialmente distribuídos e independentes
9	Requisições remotas	Sites remotos representam mais de 70% dos acessos ao servidor e mais de 60% dos bytes transferidos
10	Uso na WAN	10% dos domínios dão origem a mais de 75% dos acessos

Tabela 2.3: Invariantes na carga de trabalho da Web (Arlitt & Williamson, 1996)

remotos e cerca de 10% dos domínios originam três quartos das requisições.

Os resultados de estudos como o de Arlitt servem como diretrizes para caracterizar melhor o tipo de carga de trabalho imposta sobre os servidores web, proporcionando um melhor entendimento sobre a mesma e auxiliando na construção de programas geradores de carga sintética. Entretanto, há que se analisar os resultados com cuidado, uma vez que nenhum levantamento feito será capaz de englobar toda a Web, ainda que trabalhos independentes tenham chegado a resultados muito semelhantes.

Para finalizar esta seção, serão apresentadas a seguir algumas conclusões, obtidas nos últimos anos, sobre o comportamento de alguns aspectos da Web, tais como: tamanho dos arquivos, característica do tráfego e popularidade dos arquivos na Web.

Distribuição do Tamanho dos Arquivos

A Web apresenta uma grande variabilidade nas características da sua carga de trabalho, sendo difícil definir um modelo de carga que se aplique a todas as situações. Particularmente em relação à distribuição do tamanho dos arquivos, tanto aqueles armazenados nos servidores quanto os efetivamente transmitidos pela rede, foi observado que a mesma apresenta características híbridas, com seu corpo obedecendo a uma distribuição log-

normal e sua extremidade superior a uma distribuição de cauda pesada, governada por Pareto (Arlitt & Williamson, 1996; Barford *et al.*, 1998). Uma distribuição de *cauda pesada* significa que valores realmente muito grandes para os tamanhos dos arquivos podem ocorrer, com uma probabilidade não desprezível (Menascé & Almeida, 1998). Além disso, os valores médios apresentam uma grande instabilidade, sendo a mediana, nestes casos, uma métrica muito mais confiável do que a média (Barford *et al.*, 1998; Crovella & Lipsky, 1997). A existência de uma característica lognormal para o corpo da distribuição significa que o logaritmo da variável aleatória que representa o tamanho dos arquivos obedece a uma distribuição normal.

Para tratar a característica de cauda pesada, uma solução é agrupar os arquivos com tamanhos semelhantes em classes (Seção 2.7.1), de modo que requisições que consumam uma quantidade semelhante de recursos do sistema possam ser estudadas em conjunto. A evidência de cauda pesada não é exclusividade da Web, podendo ser observada em outras situações, como no tempo de CPU consumido por processos no UNIX, no tamanho dos arquivos em sistemas UNIX, em *frames* de vídeo comprimidos e também em rajadas geradas pela atividade de redes Ethernet ou de sessões de FTP (Floyd & Paxson, 2001).

A propriedade de cauda pesada é importante, pois ela é um dos indicativos da presença de auto-similaridade (*self-similarity*) no tráfego gerado pelas requisições na Web, característica analisada a seguir.

Características do Tráfego

Tradicionalmente, ao se modelar o tráfego em redes locais, assumia-se como válido o modelo de Poisson para caracterizá-lo. Mais recentemente, alguns estudos indicaram a presença de outro tipo de característica para o tráfego, tanto em redes locais (Leland *et al.*, 1994) quanto de longa distância (Paxson & Floyd, 1995).

Se o modelo de Poisson fosse adotado, a presença de rajadas (*bursts*) no tráfego tenderia a ser suavizada, desde que se considerasse uma escala de tempo suficientemente grande, mas, ao contrário, medições feitas mostraram que o tráfego na rede apresenta-se em rajadas e com uma variância significativa em diferentes escalas de tempo (Crovella & Bestavros, 1997). Esta propriedade pode ser caracterizada estatisticamente usando-se a noção de *auto-similaridade*, que é uma propriedade geralmente associada a fractais, objetos que têm a mesma aparência independentemente da escala em que são observados. Em termos estatísticos, isto significa que a característica da distribuição de probabilidade permanece inalterada, não importando a duração do período de observação.

Particularmente, o estudo de Crovella & Bestavros (1997) mostra que o tráfego na Web possui a característica de auto-similaridade, apresentando-se em rajadas em diferentes escalas de tempo. Outra característica notada é que o tráfego possui uma dependência de longa duração, com valores em qualquer instante correlacionados com todos os valores

futuros. Foi também observada a presença de tráfego em rajadas em quatro ordens de grandeza, considerando-se intervalos de 1, 10, 100 e 1000 segundos.

Em períodos de grande atividade, o tráfego da Web pode facilmente exceder a capacidade do servidor, com taxas de pico de 8 a 10 vezes maiores que a média (Mogul, 1995; Stevens, 1996). Por essa razão, a característica de tráfego em rajadas precisa ser incorporada nos modelos e nas ferramentas de geração de carga sintética, a fim de que se possa criar um tráfego mais condizente com a realidade da Web (Floyd & Paxson, 2001).

Popularidade dos Arquivos

Uma terceira propriedade notada no uso da Web refere-se à popularidade dos arquivos, a qual mede a distribuição das requisições pelo universo dos arquivos armazenados nos servidores (Barford & Crovella, 1998). Foi observado que alguns arquivos são extremamente populares e tendem a concentrar a maior parte das referências, enquanto outros são acessados uma única vez, se tanto.

Vários estudos mostram que a popularidade dos arquivos na Web é governada pela *Lei de Zipf* (Zipf, 1949; Almeida *et al.*, 1996; Cunha *et al.*, 1995; Huberman *et al.*, 1998), a qual originalmente era aplicada para relacionar a popularidade de uma palavra com a freqüência de sua utilização. No contexto da Web, Zipf permite concluir que, se os arquivos forem ordenados em ordem decrescente de popularidade, o número de referências a um arquivo tende a ser inversamente proporcional à sua posição na classificação, embora a razão disso ainda não esteja muito clara (Barford & Crovella, 1998). Esta distribuição é muito importante no estudo do comportamento de *caches*.

2.8 Benchmarks para a Web

Existem várias técnicas que podem ser empregadas a fim de avaliar o desempenho de um sistema computacional. Em geral, os dados mais exatos são obtidos quando se analisa um sistema real em operação, com uma carga de trabalho também real. Entretanto, isso muitas vezes é impraticável, seja porque o sistema ainda não existe ou porque o mesmo é muito grande ou complexo (o caso da Web), inviabilizando um estudo deste porte.

Uma alternativa que se apresenta é a utilização de *benchmarks*, que são um conjunto de programas que procuram simular um determinado cenário de utilização de um sistema computacional, em termos de consumo de CPU, operações de I/O, tráfego na rede e outras características, a fim de obter uma medida ou conjunto de medidas representativas do desempenho do sistema. Em geral, *benchmarks* são utilizados com o intuito de comparar diferentes sistemas, podendo servir também como ferramentas de monitoramento e diagnóstico ou, ainda, como auxílio na tomada de decisões (Menascé & Almeida, 1998).

Os *benchmarks* permitem avaliar um sistema em diferentes situações, com uma carga de trabalho que pode ser reproduzida quantas vezes for necessário. Contudo, há que se tomar cuidado na interpretação dos resultados obtidos, pois um *benchmark* extrai informações sobre o desempenho de um sistema, submetido a uma carga particular e com uma dada configuração. É neste contexto específico que seus resultados devem ser entendidos.

A seguir, são comentados dois *benchmarks* tradicionalmente utilizados na avaliação de desempenho de servidores web.

2.8.1 WebStone

O WebStone (Trent & Sake, 1995) é um dos mais conhecidos *benchmarks* para servidores web. Ele permite que a carga de trabalho seja configurada segundo diferentes parâmetros a fim de testar o servidor HTTP em diversas situações. O WebStone envia uma série de comandos GET (seção 2.5.2) ao servidor solicitando documentos previamente armazenados; permite também realizar requisições dinâmicas (CGI) e fazer chamadas à API do servidor web, tanto para o *Internet Information Server* (ISAPI) quanto para o *Netscape Enterprise Server* (NSAPI).

O WebStone é um *benchmark* distribuído, composto de um processo mestre e vários processos clientes. O mestre se encarrega de gerar os clientes (via fork()) e os mesmos fazem as requisições ao servidor, da maneira mais rápida que lhes for possível. Após cada cliente terminar sua sessão, este retorna os resultados ao mestre, que os recolhe e compõe um sumário geral.

O usuário do WebStone possui um bom controle sobre o andamento do teste, podendo definir o número de clientes, a duração do teste, o número de iterações, os tipos de arquivos pedidos, a freqüência de ocorrência de cada um deles numa sessão e o número de máquinas clientes a serem utilizadas no teste.

As principais métricas que podem ser obtidas são o *throughput* do servidor e o tempo de resposta médio (latência). O *throughput* é medido em bytes por segundo e corresponde ao número de bytes retornados pelo servidor dividido pelo tempo de duração do teste. A latência compõe-se de três partes: o tempo para o estabelecimento da conexão, o tempo de resposta da requisição no servidor e o tempo para transmissão dos dados na rede. O WebStone produz várias outras estatísticas, inclusive relacionadas a erros, que podem ser combinadas de diversas formas para permitir a avaliação de desempenho do servidor.

2.8.2 SPECweb

O SPECweb é um *benchmark* padronizado, produzido pela SPEC, que se destina a medir o desempenho de um servidor web em condições de sobrecarga, em plataformas UNIX ou Windows NT.

Um teste com SPECweb consiste de um servidor web e um conjunto de máquinas clientes que irão gerar uma carga progressivamente mais alta a fim de levar o servidor ao seu limite, quando, então, o seu tempo de resposta começa a se degradar de forma significativa. O ponto em que o servidor web se satura representa o número máximo de conexões HTTP por segundo que o servidor suporta e é esta a métrica que é fornecida pelo SPECweb96 (Levitt, 1997).

A arquitetura do benchmark consiste de um processo cliente principal (*prime*) que cria uma série de processos filhos, os quais irão, de fato, gerar a carga de trabalho para o servidor. A carga de trabalho do SPECweb96 é fixa e extraída de registros de acesso a sites populares da Web, não podendo ser modificada pelos usuários. A distribuição do tamanho dos arquivos tenta reproduzir as características de cauda pesada encontradas na Web (Seção 2.7.2).

Na sua versão inicial, a SPECweb96, este *benchmark* somente permitia avaliar o desempenho do servidor usando-se páginas estáticas, porém, na versão mais recente, a SPECweb99, esta lacuna já foi preenchida, sendo agora possível gerar carga dinâmica com os métodos GET e POST (SPEC, 2001). Também já foi introduzido suporte para as conexões persistentes do protocolo HTTP 1.1 (Seção 2.5.3) e para *cookies*. O SPECweb99 não mais mede o número de operações HTTP por segundo, como na versão anterior, mas sim o número de conexões simultâneas que é possível estabelecer com o servidor quando o mesmo está saturado.

Além dos benchmarks discutidos, existem outras ferramentas que permitem a geração de carga sintética para servidores web, as quais são resultado de pesquisas recentes e procuram incorporar na carga produzida as características estudadas na Seção 2.7.2. Banga & Druschel (1998) levantam a necessidade da existência de um software que seja capaz de gerar e sustentar uma sobrecarga em um servidor web, simulando, na prática, uma base de clientes infinita. Eles descrevem uma ferramenta, denominada *S-Clients*, que produz efetivamente uma carga de trabalho com essas características. Mosberger & Jin (1998) também criaram uma ferramenta, chamada *httperf*, com propriedades semelhantes aos *S-Clients* e que possui ainda a flexibilidade de poder gerar diferentes tipos de carga e coletar diferentes estatísticas, facilitando a criação de *benchmarks* em vários níveis. Finalmente, Barford & Crovella (1998) realizaram uma pesquisa que deu origem à ferramenta *Surge*, a qual produz uma carga de trabalho com propriedades de cauda pesada, auto-similaridade, popularidade de arquivos segundo Zipf e outras.

2.9 Modelagem de Servidores Web

O uso de abordagens como *benchmarks* e coleta de dados na avaliação de desempenho pertence ao domínio das técnicas de aferição. Outra possibilidade muito comum é o uso da modelagem, a qual se mostra bastante atrativa quando o objeto de estudo são sistemas complexos ou ainda inexistentes (Jain, 1991). Um modelo é uma abstração de uma situação real, em que se procuram incorporar suas características mais relevantes.

No caso da Internet, devido ao seu tamanho e constante evolução, muitas vezes torna-se economicamente inviável ou até impossível realizar uma experimentação em larga escala, portanto a avaliação de desempenho baseada em modelos se impõe como a alternativa mais adequada. Entretanto, é bom se ter cuidado para não cair em algumas armadilhas, entre elas a de criar modelos por demais simplificados, que não incorporem aspectos fundamentais do comportamento da Internet (Floyd & Paxson, 2001).

Um dos objetivos deste trabalho é o de produzir modelos para servidores web, particularmente de servidores que dêem suporte a serviços diferenciados. Na seção a seguir, serão apresentados alguns modelos para servidores convencionais coletados na literatura, os quais poderão servir de base para esta pesquisa.

2.9.1 Exemplos de Modelos

Slothouber (1995) apresenta um modelo que é uma visão de alto nível, simplificada, de um servidor web, o qual é modelado como uma rede de filas aberta. O autor argumenta que seu objetivo era produzir um modelo de aplicabilidade geral, que abstraísse todos os detalhes de hardware e software, mas que, ainda assim, permitisse estudar o relacionamento entre as velocidades do servidor e da rede. Embora o modelo esteja defasado em relação a alguns pressupostos, ele serve como uma primeira aproximação didática do assunto.

O modelo é apresentado na Figura 2.7 e nele a rede de filas consiste de quatro nós ou centros de serviço: dois modelam o servidor web e os outros, a comunicação via rede. As solicitações de arquivos chegam ao servidor com uma freqüência A e toda a inicialização é feita no centro de serviço S_I. Então, a tarefa segue para o centro de serviço S_R, onde um *buffer* de dados é lido do disco e passado para a rede. No centro de serviço S_S, este bloco de dados é transmitido pela Internet e chega até o *browser* cliente (S_C). Se a transmissão do arquivo ainda não se completou, a tarefa volta a S_R com uma probabilidade $1-p$; caso contrário, a tarefa se completa e deixa o sistema. A taxa de chegada em S_R (A') é a soma entre a taxa de chegada que vem da rede e a taxa das tarefas ainda não completadas, que vêm de S_C.

Pode-se perceber que várias simplificações estão presentes no modelo. Por exemplo, o mesmo só considera as solicitações de arquivos efetivamente armazenados no servidor,

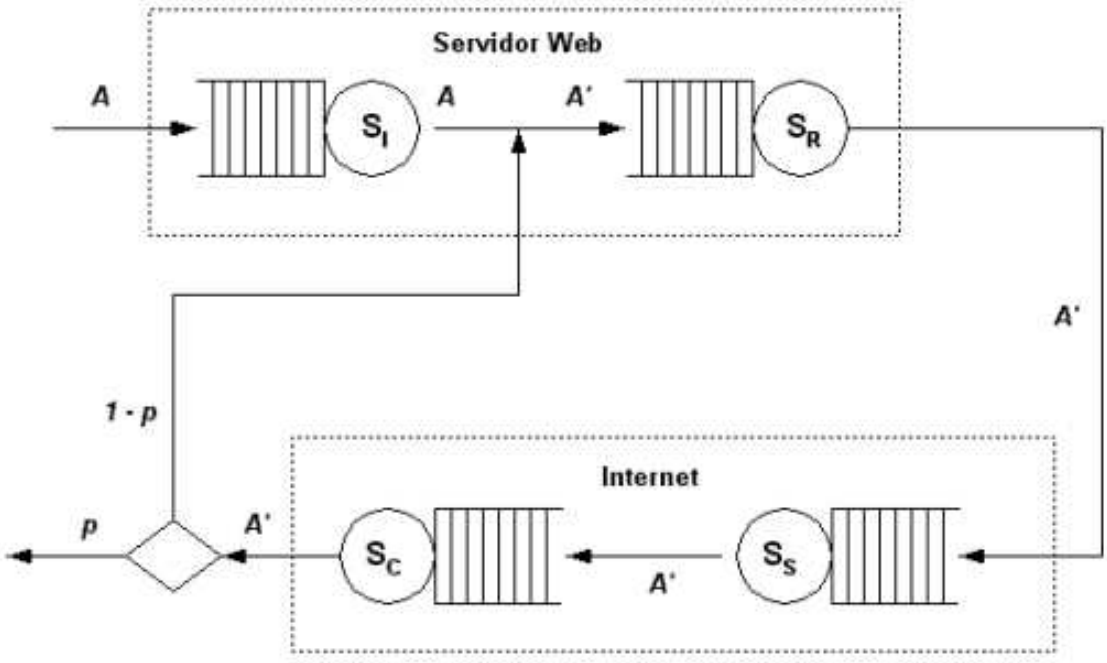


Figura 2.7: Modelo de rede de filas para um servidor web

ou seja, somente requisições de páginas estáticas são modeladas (Seção 2.4). Também é assumido que a distribuição do tamanho dos arquivos é exponencial, fato que é desmentido por estudos recentes que mostram que a mesma segue um modelo híbrido, cujo corpo é aproximado por uma distribuição lognormal e cuja distribuição dos valores extremos por Pareto (Seção 2.7.2). Contudo, como o próprio autor coloca, seu modelo assume pressupostos conservadores, o que não tira seu valor, ainda mais levando-se em conta que o mesmo foi produzido em 1995, quando ainda não era disseminado o uso de páginas dinâmicas na Web e resultados como as invariantes de distribuições de cauda pesada para o tamanho dos arquivos estavam ainda em discussão. Também por razões históricas, a análise feita considera apenas uma rede Ethernet a 10 Mbps.

Em Menascé & Almeida (1998), modelos mais elaborados são apresentados, como o da Figura 2.8. Observa-se que o modelo inclui seis centros de serviço: o primeiro deles é um nó de retardo e representa os clientes. O tempo de permanência neste nó corresponde ao tempo de "pensar" dos clientes (*think time*), ou seja, o tempo entre a recepção de um arquivo e o início da próxima solicitação. O centro de serviço 2 representa a rede local e o centro 3, um roteador, que é modelado como um nó de retardo porque sua latência é muito pequena em relação às dos demais centros de serviço. Os *links* de saída e chegada do provedor de Internet são modelados nos centros de serviço 4 e 6. O centro de serviço 5 representa o provedor, seus *links* para a Internet e seus servidores web, numa visão de alto nível. Pela rede de filas, considera-se que circulam requisições HTTP.

Os autores apresentam outro modelo (Figura 2.9), o qual leva em conta apenas o lado

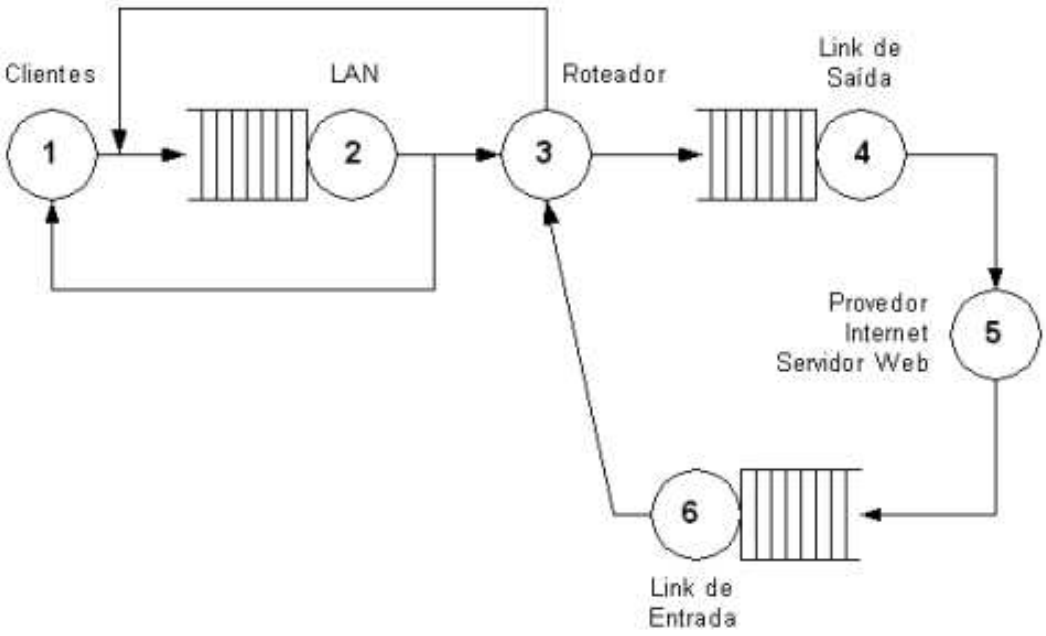


Figura 2.8: Modelo de rede de filas para a Web

do servidor web, que é o principal foco do presente trabalho. Este modelo procura representar um servidor web típico, que pode ser acessado por qualquer cliente na Internet. Os *links* de entrada e saída (1 e 6) são modelados separadamente, o roteador (2) é modelado como um nó de retardo e o servidor web é dividido em dois centros de serviço, a CPU (4) e o disco (5). Este modelo poderia ainda ser estendido para incluir um *cluster* de servidores web ou um servidor *proxy*. Da mesma forma, o servidor web ou a rede (3) poderiam estar mais detalhados, a fim de melhor caracterizar as filas que se formam no acesso à CPU, ao disco e ao meio de comunicação.

Essas e outras alternativas são encontradas em Menascé & Almeida (1998, 2003), onde os autores desenvolvem soluções analíticas para os modelos propostos. No presente trabalho, pretende-se usar a simulação como ferramenta para solução dos modelos.

2.9.2 Técnicas de Aferição vs. Modelagem

Para finalizar, cabe fazer algumas considerações a respeito da utilização das *técnicas de aferição* ou da *modelagem* na avaliação de desempenho da Web. A experimentação de um ambiente real, em que se pode obter resultados diretamente a partir de um sistema em funcionamento, em geral tem um grande apelo, pois normalmente fornece dados mais precisos e dá a impressão de que se está trabalhando com o mundo "de verdade" e não com uma "mera" abstração, um modelo do mesmo.

Entretanto, devido ao tamanho da Web e sua rápida evolução, torna-se difícil, senão impossível, realizar esforços, por exemplo, de coleta de dados, pois corre-se o risco de,

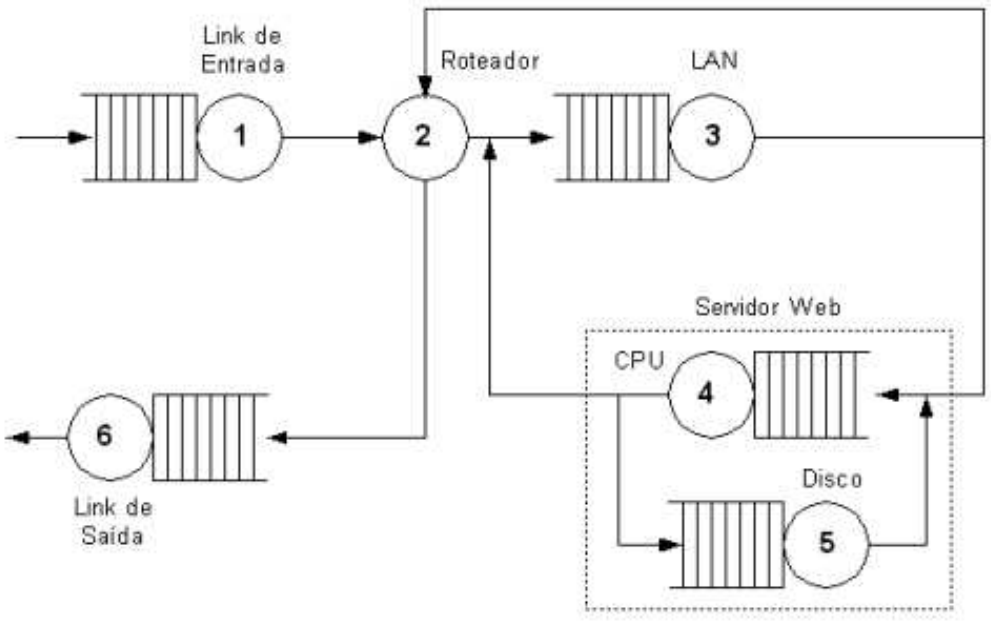


Figura 2.9: Modelo de rede de filas para um servidor único

ao terminar o trabalho, o mesmo já estar defasado. É neste contexto que se impõe a necessidade da construção de modelos que permitam compreender a complexidade da Web atual e futura de uma maneira geral, seja qual for a forma escolhida para solucioná-los.

Não se quer dizer com isso que técnicas como o uso de *benchmarks* ou coleta de dados devam ser desprezadas. Pelo contrário, a coleta de dados é essencial para verificar se a realidade corresponde àquilo que implicitamente se assume como verdadeiro (Floyd & Paxson, 2001), pois freqüentemente os dados coletados desafiam o senso comum e as verdades estabelecidas, como já foi comentado neste capítulo. Só para citar outra aplicação, *logs* de atividade em servidores web ou do tráfego na rede podem servir para alimentar simulações baseadas em *traces*, desde que se tomem os devidos cuidados quanto à acuidade dos dados levantados. E os resultados de *benchmarks*, por exemplo, podem ser utilizados para estimar os parâmetros de entrada para os modelos (Menascé & Almeida, 1998; Chen & Mohapatra, 1999).

2.10 Considerações Finais

Este capítulo apresentou um panorama geral da Web, abordando inicialmente a infra-estrutura da Internet e particularmente os protocolos TCP/IP e sua aplicação. A organização atual da Web foi discutida e também foram apresentadas as principais alternativas para a comunicação entre clientes e servidores. Foi estudado o protocolo HTTP, seu funcionamento e impacto no comportamento da Web. Em seguida, deu-se uma visão geral da

área de avaliação de desempenho aplicada à Web. Foram comentadas as principais alternativas para a construção de servidores web e as métricas de desempenho mais utilizadas. A seção de caracterização de carga apresentou os mais importantes resultados obtidos nos últimos anos quanto às características do tráfego na Web e apontou possíveis soluções para lidar com sua complexidade. Também foram discutidos os problemas inerentes à geração de carga sintética na Web, comentando-se sobre alguns *benchmarks*. Finalmente, argumentou-se sobre o potencial da aplicação de técnicas de modelagem e simulação à Web, sendo apresentados alguns modelos em redes de filas para servidores web.

O próximo capítulo aborda os problemas e limitações do modelo atual de serviços da Internet e a necessidade da introdução de qualidade de serviço na mesma, apresentando as principais arquiteturas para tal. Discute também a provisão de uma QoS diferenciada em nível de aplicação, particularmente nos servidores web.

Capítulo

3

Serviços Diferenciados

3.1 Introdução

O modelo atual de serviços da Internet tem suas raízes em pesquisas que datam de três décadas atrás. Embora tenha funcionado e ainda funcione muito bem para vários tipos de aplicações, já são notados sinais de que o mesmo possa estar atingindo o seu limite. Isso gera a necessidade de se pensar em um novo paradigma de serviços para a Internet e, conseqüentemente, para a Web, conduzindo-a a novos níveis de funcionalidade.

A Internet é uma rede global que depende do protocolo IP para o transporte dos dados, o qual foi projetado para ser bastante flexível, podendo funcionar sobre uma gama de tecnologias de rede diferentes, fato que contribuiu para a disseminação da Internet e das redes IP por todo o mundo. Com isso, serviços como o correio eletrônico e a Web tornaram-se parte da vida das pessoas, seja para fins de entretenimento, educativos ou comerciais. Além disso, o que se nota atualmente é a convergência de outros tipos de redes, já há mais tempo estabelecidas em nossa rotina, como as redes de telefonia, rádio e TV, para a Internet (Stardust, 1999b), a qual pretende ser o principal meio de transmissão de informações, condução de negócios e entretenimento no futuro. A questão que se coloca é se a mesma está preparada para responder a essas novas demandas.

Este capítulo aborda os problemas atuais da Internet e como a introdução da noção de qualidade de serviço pode torná-la mais eficiente. Em particular, destacam-se a abordagem de serviços diferenciados sobre redes IP e a viabilidade de seu emprego em nível de aplicação, particularmente em servidores web, tema que vem a ser a razão desta tese.

3.2 Limitações da Internet Atual

O tráfego da Internet tem crescido enormemente nos últimos anos, principalmente depois do surgimento da Web e não se vê sinais de que essa tendência vá se reverter em um futuro próximo. Este crescimento não se deve apenas ao aumento do número de usuários e de aplicações, mas também se dá em decorrência do aparecimento de novos tipos de aplicações, principalmente multimídia e de tempo real, que exigem da Internet uma resposta para a qual ela não foi projetada.

O fato é que uma rede como a Internet, baseada no protocolo IP, somente é capaz de fornecer um serviço de "melhor esforço" (*best-effort*), ou seja, a rede tentará, de todas as formas, entregar os dados que lhe foram confiados, no menor tempo possível e sem erros, mas não será dada nenhuma garantia às aplicações de que isso realmente ocorrerá. Este comportamento é decorrente da própria concepção do protocolo IP, o qual foi projetado para ser simples, eficiente e flexível, fornecendo um serviço de entrega de datagramas do tipo não-confiável. Na verdade, em situações nas quais o tráfego na rede é inferior a sua capacidade, o protocolo IP e seus congêneres conseguem realizar suas funções com bastante rapidez e eficiência. O problema está nos momentos de congestionamento, quando o serviço na rede pode apresentar níveis inconsistentes ou até mesmo imprevisíveis, devido à variação do atraso na entrega dos pacotes e à perda de dados (Vasiliou, 2000).

Tradicionalmente, a abordagem empregada pelas redes IP, desde o seu início, foi tornar os elementos extremos da rede (*hosts*), complexos e os elementos internos (roteadores), mais simples, destinando-se estes meramente a verificar o endereço IP dos datagramas em uma tabela de rotas a fim de determinar qual o próximo "salto" (*hop*) a ser dado (Stardust, 1999b). Esta solução tem sido suficiente para a transmissão de dados convencionais (dados provenientes de *e-mail*, transferência de arquivos, navegação na Web), comumente encontrada na Internet, mas se revela inadequada para os novos tipos de aplicações que se espera colocar sobre a rede.

O tráfego atual na Internet não é apenas intenso, mas se origina de aplicações que possuem requisitos operacionais variados, que a infra-estrutura existente na rede não está preparada para suportar. Aplicações multimídia, por exemplo, requerem uma alta largura de banda e, em geral, podem suportar pequenas perdas de pacotes em favor de uma melhor sincronização na entrega dos mesmos. Por outro lado, aplicações de tempo real, como a telefonia sobre IP, colocam pouca demanda sobre a largura de banda, mas, em compensação, possuem requisitos de temporização bastante rígidos (Stardust, 1999b) que, se não forem atendidos, podem inviabilizar a comunicação entre as partes. Aplicações de transferência de arquivos, por sua vez, demandam uma certa largura de banda e não podem tolerar a perda de nenhum bit de informação. Há também aplicações que necessitam de um serviço *multicast* como, por exemplo, videoconferência e transmissão de rádio e TV via Web. Finalmente, a Web vem sendo cada vez mais utilizada para a

condução de negócios, uma área que exige alta confiabilidade e disponibilidade do meio de transmissão e cujos usuários estão dispostos a pagar mais por um serviço mais previsível e de melhor qualidade (Xiao & Ni, 1999).

Possíveis Soluções

Do exposto acima, conclui-se que não existe um tipo de rede que possa satisfazer a toda a gama de requisitos colocados. À primeira vista, o problema da Internet poderia parecer de largura de banda, ou seja, supondo-se que fosse possível acrescentar capacidade de transmissão indiscriminadamente à rede, teoricamente seria possível acolher todo o tráfego existente, satisfazendo a todos.

Infelizmente, a experiência mostra que, não importa a quantidade de largura de banda que seja disponibilizada, a tendência é, em pouco tempo, tal quantidade ser completamente exaurida. Além disso, como foi comentado, para algumas aplicações o ponto crítico não está na quantidade de largura de banda disponível, mas sim na latência de transmissão. Há que se considerar também que a Internet é uma rede composta por redes, compreendendo as mais diferentes tecnologias, portanto, não é tão simples assim incrementar a capacidade dos seus canais de comunicação uniformemente. Outro problema é a existência de tráfego em rajadas, que pode levar a demanda por largura de banda a níveis muito superiores aos experimentados usualmente.

Sendo assim, vê-se que dotar a rede da maior capacidade possível, além de economicamente inviável, não é a solução adequada para alguns casos. O que se precisa é de um gerenciamento ativo da largura de banda disponível, de modo que se possa fornecer aos usuários e aplicações diferentes classes ou níveis de serviço (Xiao & Ni, 1999). Para tanto, os serviços IP precisam ser incrementados e isto pode ser alcançado agregando-se alguma "inteligência" aos elementos internos da rede a fim de diferenciar o tráfego que passa pelos mesmos. É sobre isso precisamente que trata a qualidade de serviço, assunto da próxima seção.

3.3 Qualidade de Serviço

3.3.1 Conceitos Básicos

Qualidade de Serviço (QoS) pode ser definida como a capacidade de fornecer a um elemento da rede (aplicação, *host* ou roteador) algum nível de segurança de que seus requisitos de tráfego e serviço serão satisfeitos (Stardust, 1999b). Está relacionada com a garantia de um atraso de entrega (*delay*) e uma perda de pacotes suficientemente baixos para certos tipos de aplicações ou tráfego (Zhao *et al.*, 2000). Os requisitos de qualidade podem ser determinados por fatores humanos, por exemplo, limites máximos para o atraso

a fim de permitir um diálogo entre duas pessoas; por razões comerciais, por exemplo, a necessidade de concluir uma certa tarefa em um tempo mínimo; ou por aplicações críticas cujo funcionamento fique comprometido caso o atraso ou *jitter* (variação do atraso) superem certos limites.

Fornecer QoS não é uma tarefa trivial, mesmo em redes pequenas e proprietárias, muito menos em uma rede global como a Internet. Para que se tenha QoS, faz-se necessária a cooperação de todos as camadas da rede, de cima a baixo, assim como de todo e qualquer elemento da rede, de fim-a-fim. Qualquer garantia de QoS será tão forte quanto o mais frágil elemento nessa cadeia entre o emissor e o receptor (Stardust, 1999b).

É importante ressaltar que o emprego de QoS não é capaz de criar largura de banda, ou seja, a rede nunca poderá fornecer aquilo que ela não tem. O que a QoS faz é administrar a largura de banda existente segundo a demanda das aplicações e dentro de certos parâmetros de gerenciamento e desempenho da rede. Uma rede habilitada para fornecer QoS continuará dando suporte ao tráfego de melhor esforço, contudo parte da largura de banda será reservada para as aplicações de mais alta prioridade. Também faz parte das atividades de gerenciamento de QoS garantir que essas aplicações menos "exigentes" não serão anuladas pelas aplicações mais "nobres".

A QoS pode ser descrita de forma absoluta ou relativa. Uma especificação de QoS em termos absolutos fornecerá métricas a serem cumpridas, relacionadas, em geral, ao atraso ou perda de pacotes, por exemplo, uma meta como "nenhum pacote poderá ter um atraso maior que x milissegundos". Caso a rede não seja capaz de garantir um serviço com essa característica, então a aplicação não terá acesso ao meio de comunicação.

A QoS relativa, por sua vez, trabalha com a idéia de diferenciação de serviços, comparando o tratamento recebido por uma classe de pacotes com aquele dado a outra classe (Zhao *et al.*, 2000). Neste caso, a rede garante que uma aplicação em uma classe de mais alta prioridade nunca receberá um serviço pior que o de qualquer classe inferior. Neste modelo, não é possível dar garantias rígidas às aplicações, em termos de métricas, como no caso anterior, pois a QoS resultante dependerá da carga atual na rede, em cada classe (Vasiliou, 2000).

Segundo Zhao *et al.* (2000), é possível identificar dois aspectos importantes dentro do tema de QoS: *garantia de desempenho* e *diferenciação de serviços*. A primeira se relaciona com o gerenciamento de largura de banda, perda de pacotes, atraso e *jitter*. A largura de banda é o recurso mais importante e a sua disponibilidade e forma de alocação afetam as outras características. A diferenciação de serviços consiste em fornecer diferentes níveis de QoS para diferentes aplicações segundo seus requisitos. Esta diferenciação pode ser feita em um grupo de pacotes individual ou em um agregado de grupos de pacotes relacionados (Vasiliou, 2000). O primeiro caso se remete à definição de fluxo, conceito explorado a seguir.

Um fluxo é definido como um fluxo contínuo de dados (*stream*), individual, unidirecional entre duas aplicações (emissor e receptor) (Vasiliou, 2000). Tradicionalmente, um fluxo é caracterizado por uma quíntupla composta por: endereço IP de origem, número de porta de origem, endereço IP de destino, número de porta de destino e protocolo de transporte. Garantir QoS em uma granulosidade tão fina quanto a de um fluxo permite isolá-lo de outras aplicações possivelmente mal-comportadas, mas, por outro lado, compromete a escalabilidade desta solução em redes globais como a Internet, que podem, em um certo instante, apresentar centenas de milhares de fluxos (Zhao *et al.*, 2000). Por essa razão, às vezes é mais vantajoso realizar o gerenciamento de QoS em um nível mais alto, agrupando-se os pacotes (fluxos) em classes com requisitos de QoS similares, cada uma tratada de modo diferente. Assim, resolve-se o problema de escalabilidade, embora as garantias de desempenho dadas não possam ser tão rígidas quanto aquelas da abordagem por fluxos individuais.

3.3.2 Caracterização do Tráfego

Como já foi visto, diferentes aplicações possuem diferentes requisitos operacionais e demandam da rede diferentes serviços para que possam funcionar satisfatoriamente.

O tráfego gerado pelas aplicações tem uma influência direta nas demandas de QoS impostas à rede e pode ser caracterizado segundo duas dimensões: previsibilidade da taxa de dados e sensibilidade ao atraso de entrega e *jitter* (Stardust, 1999b; Magalhães & Cardozo, 1999). Considerando-se a previsibilidade (Tabela 3.1), o tráfego pode ser do tipo fluxo contínuo (*stream*) ou em rajadas. Tráfego de fluxo contínuo é característico de aplicações que geram mídia contínua, como áudio e vídeo. Tráfego em rajadas é típico de aplicações como transferência de arquivos e interações cliente-servidor.

Tipo da Taxa	**Descrição**
Fluxo Contínuo	Entrega de dados a uma taxa relativamente constante (CBR — *Constant Bit Rate*)
Em Rajadas	Blocos de dados são entregues de forma imprevisível, a uma taxa de dados variável (VBR — *Variable Bit Rate*), que pode chegar a utilizar toda a largura de banda se não for empregado algum tipo de controle

Tabela 3.1: Caracterização do tráfego segundo a taxa de dados (Stardust, 1999b)

O tráfego também pode ser caracterizado, segundo sua tolerância ao atraso e *jitter*, em cinco categorias (Tabela 3.2). O tráfego assíncrono, também denominado "elástico", não possui restrições temporais e admite bem o serviço de melhor esforço. Aplicações tradicionais como *ftp* e *e-mail* estão incluídas nesta categoria. O tráfego síncrono possui restrições temporais, mas é possível compensar as variações observadas através de esquemas de sincronização no receptor. Aqui se enquadra o tráfego gerado por aplicações de áudio e vídeo

sob demanda. O tráfego interativo impõe valores máximos ao atraso e *jitter*, sob pena de comprometer a interatividade da comunicação. É característico de situações em que duas ou mais pessoas interagem em tempo real, por exemplo, aplicações de vídeo-conferência e telefonia sobre IP. No tráfego isócrono, atraso e *jitter* altos comprometem a qualidade da informação transmitida e podem afetar a usabilidade da aplicação. É gerado por aplicações que necessitam de um baixo atraso, como difusão de rádio e TV em redes comutadas por pacotes. Por fim, o tráfego de missão-crítica é gerado por aplicações cuja funcionalidade fica comprometida caso o atraso e *jitter* ultrapassem certos limites, por exemplo, em aplicações de tele-comando, tele-supervisão e de automação industrial (Magalhães & Cardozo, 1999).

Tolerância a Atrasos	**Tipo de Tráfego**	**Descrição**
alta	Assíncrono	Tráfego não possui restrições temporais
	Síncrono	Dados são sensíveis ao tempo, mas flexíveis
	Interativo	Atrasos podem ser notados pelos usuários, mas não afetam a usabilidade e funcionalidade da aplicação
	Isócrono	Sensível ao tempo de uma forma que pode comprometer a usabilidade da aplicação
baixa	Missão-crítica	Atrasos podem comprometer a funcionalidade da aplicação

Tabela 3.2: Caracterização do tráfego segundo a sensibilidade ao atraso (Stardust, 1999b)

3.3.3 Arquiteturas para QoS

Em resumo, pode-se afirmar que o objetivo da introdução de QoS na Internet atual é fornecer algum nível de previsibilidade e controle, além do serviço de melhor esforço das redes IP (Stardust, 1999a), pois somente assim será possível atender às crescentes demandas das aplicações atuais e futuras. O tema de QoS na Internet tem sido objeto de intensas pesquisas nos últimos anos e várias abordagens foram propostas nesse sentido, sempre tendo em mente que o sucesso da Internet se explica, em grande parte, pela simplicidade dos protocolos que funcionam sobre a rede, portanto assim ela deverá permanecer.

Destacam-se dois tipos básicos de arquiteturas para QoS na Internet (Stardust, 1999a; Zhao *et al.*, 2000):

- *Reserva de Recursos.* Os recursos da rede são atribuídos às aplicações segundo suas demandas de QoS e submetidos à política de gerenciamento de largura de banda. Esta é a abordagem empregada pela arquitetura de Serviços Integrados (*IntServ*).

- *Priorização.* O tráfego na rede é classificado de acordo com suas características de demanda e recebe os recursos segundo a política de gerenciamento vigente. As clas-

ses de tráfego mais exigentes recebem tratamento preferencial, abordagem adotada pela arquitetura de Serviços Diferenciados (*DiffServ*).

No primeiro caso, a *aplicação* receberá os recursos de que necessita antes da transmissão dos dados, sendo necessária a reserva de caminhos (*paths*) e recursos ao longo da rede. No segundo caso, o *tráfego* é dividido em classes e os pacotes pertencentes a uma certa classe são marcados diferentemente a fim de receber o serviço adequado (Xiao & Ni, 1999).

As arquiteturas de Serviços Integrados e Serviços Diferenciados serão discutidas com mais detalhes a seguir.

3.4 Serviços Integrados

A abordagem de Serviços Integrados, definida na RFC 1633 (Braden *et al.*, 1994), foi a primeira das soluções propostas para dar suporte a QoS na Internet, no âmbito da IETF. Tinha como objetivo inicial atender certas garantias exigidas por determinados tipos de tráfego, como transmissão de áudio e vídeo. A arquitetura *IntServ* visa fornecer, em uma rede comutada por pacotes, como a Internet, o serviço mais próximo possível da abstração de circuitos virtuais.

A idéia principal subjacente ao *IntServ* é a de reserva de recursos. Antes de iniciar a transmissão dos dados, as aplicações precisam encontrar um caminho até o receptor que satisfaça suas demandas de QoS, reservando, ao longo do mesmo, os recursos necessários (Xiao & Ni, 1999). Para tanto, as aplicações fazem uso do protocolo RSVP (*Resource Reservation Protocol*), um protocolo de controle e sinalização que atua na camada de rede, sendo responsável por reservar caminhos e recursos na sub-rede de comunicação.

O protocolo RSVP é considerado a mais elaborada e complexa das soluções para suporte a QoS e representa uma grande ruptura com a abordagem tradicional de melhor esforço das redes IP (Stardust, 1999a), na qual, antes de se enviar um pacote pela rede, não é feita nenhuma reserva de recursos de qualquer tipo, nem se procura encontrar um caminho disponível. A abordagem de serviços integrados trabalha sobre fluxos individuais usando o protocolo RSVP para reservar recursos nos roteadores da rede, de fim-a-fim, com o objetivo de atender às demandas de QoS dos fluxos (Vasiliou, 2000). Melhoramentos recentes no RSVP têm procurado estendê-lo para trabalhar também com a noção de agregação de fluxos.

3.4.1 Classes de Serviço

A arquitetura de serviços integrados define duas classes de serviço, além do modelo de melhor esforço, tradicionalmente encontrado nas redes IP. São elas:

- *Serviço Garantido*[1]. Especificado na RFC 2212 (Shenker *et al.*, 1997), fornece um limite superior rígido para o atraso fim-a-fim, além de garantir a disponibilidade de largura de banda. Este serviço se destina a aplicações que possuem requisitos estritos de tempo real para funcionar, atingindo um alto nível de QoS na Internet.

- *Serviço de Carga Controlada*[2]. Especificado na RFC 2211 (Wroclawski, 1997), fornece um serviço equivalente ao modelo de melhor esforço em uma rede pouco utilizada, com quase nenhuma perda ou atraso. Em situações de sobrecarga, esta abordagem será capaz de compartilhar a largura de banda entre múltiplos fluxos, de uma maneira controlada, garantindo um serviço melhor que o usual. Entretanto, este modelo não oferece garantias de atraso máximo, apenas um limiar probabilístico, assim como também não pode assegurar que pacotes não serão perdidos.

3.4.2 Protocolo RSVP

O protocolo RSVP, especificado na RFC 2205 (Braden *et al.*, 1997), constitui-se no protocolo de controle e sinalização usado por aplicações para reserva de recursos ao longo da rede. Apresenta algumas características interessantes, analisadas a seguir (Stallings, 2002):

- *Protocolo de controle.* Embora pertença à camada de rede, o RSVP não é um protocolo de roteamento. Ele não transporta dados, apenas informações de controle e sinalização referentes à reserva de recursos ao longo de um caminho ou árvore de distribuição (*spanning tree*).

- *Unicast e Multicast.* As reservas feitas pelo RSVP podem ser para transmissão *unicast* ou *multicast*, tendo a capacidade de se adaptar dinamicamente a mudanças no número de membros do grupo e a modificações nas rotas.

- *Unidirecional.* O RSVP faz reservas para fluxos unidirecionais, apenas. Assim, para trocas de dados entre dois sistemas, é preciso fazer reservas separadas nos dois sentidos.

- *Reserva iniciada pelo receptor.* O receptor é quem toma a iniciativa de fazer a reserva de recursos, especificando seus requisitos de largura de banda, atraso e *jitter*.

- *Soft State.* O protocolo RSVP usa a abordagem de *soft state*, ou seja, a informação sobre a reserva de recursos para os fluxos fica armazenada temporariamente nos roteadores. Dessa forma, é necessário renová-la periodicamente, tarefa que é responsabilidade dos *hosts* finais (*end hosts*).

[1] *Guaranteed Service*
[2] *Controlled-Load Service*

- *Diferentes estilos de reserva.* Os usuários de um mesmo grupo *multicast* podem especificar como as reservas devem ser agregadas nos roteadores intermediários, a fim de economizar recursos da rede.

- *Operação transparente através de roteadores não-RSVP.* Como as reservas feitas são independentes do protocolo de roteamento, o tráfego pode atravessar roteadores não habilitados para RSVP. Neste caso, é usado o modelo de melhor esforço, não sendo possível assegurar a QoS nesse trecho do caminho.

- *Suporte para IPv4 e IPv6.* O RSVP pode explorar o campo *Type-of-Service* no cabeçalho do protocolo IPv4 e o campo *Flow Label* do IPv6.

A Figura 3.1 ilustra o funcionamento do protocolo RSVP: o emissor envia uma mensagem do tipo PATH a um ou mais receptores informando as características do tráfego(1). À medida que a mensagem PATH se propaga pela rede, cada roteador grava informações sobre o caminho na mensagem(2) (por exemplo, o roteador de origem da mesma). O receptor, ao receber a mensagem PATH(3), responde com uma mensagem do tipo RESV(4), requisitando recursos ao longo do caminho gravado na mensagem PATH, em ordem inversa, do receptor para o emissor. Quando um roteador intermediário recebe a mensagem RESV(5), ele verifica se pode atender a solicitação de recursos. Caso seja possível, ele faz a reserva de largura de banda e espaço em buffers e passa a mensagem RESV adiante; caso contrário, envia uma mensagem de erro ao receptor. Se o emissor recebe a mensagem RESV(6), isto significa que a reserva de recursos foi bem sucedida e é possível iniciar a transmissão dos pacotes(7).

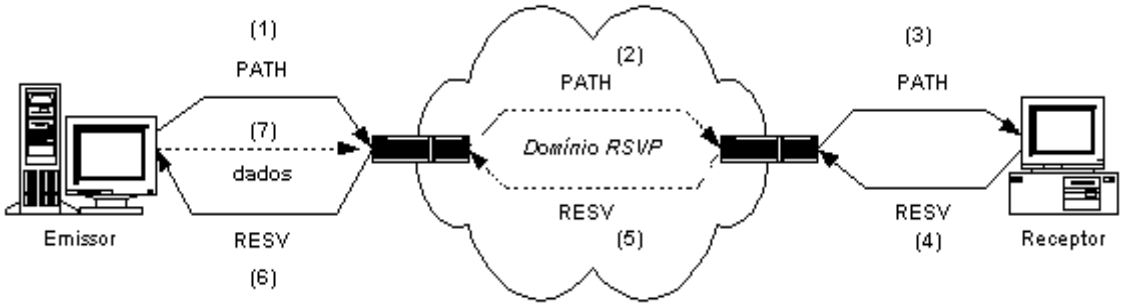


Figura 3.1: Funcionamento do protocolo RSVP

Com o uso do protocolo RSVP, consegue-se atingir a maior QoS possível na Internet (Stardust, 1999a), pois ele permite fazer o gerenciamento em uma granulosidade bem fina, em nível de fluxo, conseguindo dar excelentes garantias de qualidade às aplicações. Entretanto, há um preço a se pagar por isso: o RSVP possui sérios problemas de gerenciamento e escalabilidade (Vasiliou, 2000). Cada roteador ao longo do caminho precisa dar suporte a RSVP para que se possa assegurar a QoS, sendo necessário manter as informações de estado e fazer o escalonamento e enfileiramento dos pacotes para cada fluxo.

Na Internet, em que dezenas de milhares de fluxos concorrentes podem passar por um mesmo roteador, o gerenciamento dessas informações parece quase impossível, até por limitações de memória e capacidade de processamento nos roteadores. Com isso, vê-se que o RSVP introduz uma complexidade significativa no núcleo da Internet, o que representa uma ruptura com o seu modelo tradicional de serviços, que sempre procurou manter a rede simples e levar a complexidade para os *hosts* finais (Stardust, 1999a).

A abordagem de serviços integrados, portanto, aplica-se melhor a um ambiente de rede local, fornecendo uma QoS intra-domínio para aplicações que dela necessitem. No âmbito da Internet, a abordagem de serviços diferenciados, tratada a seguir, revela-se mais adequada.

3.5 Serviços Diferenciados

Os problemas de escalabilidade da arquitetura *IntServ* e a dificuldade de sua implantação em uma rede com as proporções da Internet motivaram os esforços para o desenvolvimento da arquitetura de Serviços Diferenciados, no âmbito da IETF, conforme proposto na RFC 2475 (Blake *et al.*, 1998).

Enquanto o *IntServ* realiza as reservas de recursos por fluxo, exigindo dos roteadores a manutenção de informações de estado para cada fluxo, fim-a-fim, a abordagem *DiffServ* baseia-se na idéia de agregação de fluxos em umas poucas classes de serviço (Magalhães & Cardozo, 1999). Dessa forma, o *DiffServ* fornece diferenciação de serviços local para grandes agregados de tráfego, enquanto o modelo *IntServ* dá garantias de performance fim-a-fim para fluxos individuais (Vasiliou, 2000).

A arquitetura de serviços diferenciados usa a classificação de pacotes como mecanismo para atingir a QoS. Para tanto, é redefinido o layout do octeto *Type-of-Service* do cabeçalho do protocolo IPv4 (ou o campo *Traffic Class* do IPv6), que passa a ser chamado de campo DS (*Differentiated Services*), conforme a RFC 2474 (Nichols *et al.*, 1998). Até o momento, somente os seis primeiros bits do campo DS são usados, recebendo a denominação de campo DSCP (*Differentiated Services Codepoint*), como mostrado na Figura 3.2. Seu objetivo é especificar o tratamento dado ao encaminhamento de pacotes em cada roteador, o chamado PHB (*Per-hop Behavior*). Os PHBs são mecanismos de priorização que permitem a agregação de fluxos gerados por diferentes aplicações, definindo uma classe de serviço (Magalhães & Cardozo, 1999).

O modelo *DiffServ*, portanto, opera através da marcação de pacotes de forma distinta, o que permite que os mesmos sejam tratados internamente à rede de maneira diferenciada, segundo a classe de serviço à qual pertencem. A abordagem empregada pelo *DiffServ* baseia-se em um esquema de prioridades relativas, ou seja, ele garante que o tráfego gerado por uma aplicação, com um certo nível de prioridade, receberá um tratamento

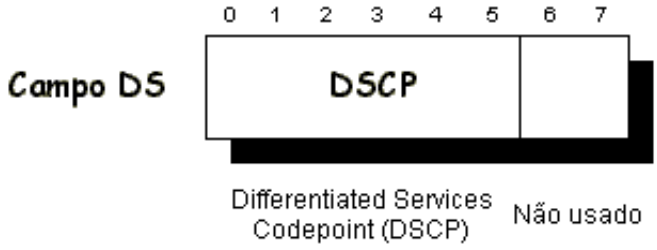


Figura 3.2: Layout do campo DS

melhor que o gerado por qualquer outra que possua uma prioridade inferior (Xiao & Ni, 1999).

A marcação dos pacotes se dá nos pontos de ingresso na rede, como os *hosts* finais e os roteadores de borda. Dessa forma, retém-se um dos princípios básicos do projeto da Internet, que é colocar a complexidade na fronteira da rede, ao contrário da abordagem *IntServ*, que exige complexidade fim-a-fim. O modelo *DiffServ* não necessita de um protocolo próprio (como no caso do RSVP, usado no *IntServ*), pois aqui se utiliza um campo do próprio datagrama IP. Tudo que um roteador precisa fazer é examinar o campo DSCP de cada pacote para determinar qual o tratamento a ser dado ao mesmo. Assim, pacotes marcados da mesma forma recebem tratamento igual. Não é necessário manter nenhum tipo de informação relacionada a fluxos, pois os roteadores só precisam ser capazes de distinguir entre um certo número de classes de serviço pré-definidas.

3.5.1 Classes de Serviço

Embora outras alternativas sejam possíveis, existem atualmente duas classes de serviço principais definidas na arquitetura de serviços diferenciados, que são abordadas a seguir: Encaminhamento Expresso e Encaminhamento Garantido (Kilkki, 1999).

Encaminhamento Expresso

O serviço denominado de Encaminhamento Expresso[3] (*Expedited Forwarding* — EF), definido na RFC 2598 (Jacobson *et al.*, 1999), permite a adaptação do modelo de serviço garantido da arquitetura *IntServ* à arquitetura de serviços diferenciados. Ele oferece garantias de QoS absoluta, com baixos valores de perda, atraso e *jitter*, fornecendo o equivalente a uma linha privada virtual com largura de banda fixa entre dois *hosts*. É indicado para aplicações de telefonia sobre IP, videoconferência e para a criação de linhas dedicadas em redes privadas virtuais (VPNs).

Sua vantagem sobre o serviço equivalente na arquitetura *IntServ* está na simplici-

[3]Usa-se também a denominação *Premium Service*.

dade de implementação, pois não é necessário manter nos roteadores nenhuma informação relativa a fluxos. Os equipamentos que implementam este comportamento (PHB) simplesmente devem escalonar o tráfego de maneira a manter descongestionadas as filas de saída, a fim de que o tráfego passe o menor tempo possível no equipamento (Magalhães & Cardozo, 1999). Em geral, os pacotes pertencentes a esta classe são colocados em uma fila de maior prioridade que a do tráfego de melhor esforço e são os primeiros a serem encaminhados em qualquer situação. Em relação à política de descarte, este serviço evita a todo custo descartar os pacotes para o tráfego em conformidade com o perfil contratado. Já para o tráfego sem conformidade, ele é implacável: os pacotes simplesmente são descartados. Uma conseqüência disso é que os usuários não podem exceder a taxa de pico solicitada; caso contrário, o tráfego em excesso será descartado. Em contrapartida, a arquitetura *DiffServ* garante que a largura de banda contratada estará disponível quando o tráfego for enviado (Xiao & Ni, 1999).

Encaminhamento Garantido

A classe de serviço Encaminhamento Garantido[4] (*Assured Forwarding* — AF), definida na RFC 2597 (Heinanen *et al.*, 1999), destina-se a aplicações que demandem da rede um serviço mais confiável que aquele de melhor esforço, mas sem todas as garantias de QoS dadas pelo encaminhamento expresso. Este serviço não oferece limites superiores para o atraso e *jitter*, mas garante um tratamento preferencial ao tráfego que dele se utilize. Ele é indicado quando se deseja obter da rede um serviço de entrega de pacotes mais consistente, por exemplo, a fim de oferecer uma melhor QoS a agregados de tráfego consistindo de rajadas de curta duração com destinos diferentes (o tráfego na Web) (Stoika & Zhang, 1998).

O princípio aqui é a divisão do tráfego em N classes, cada uma com alguns níveis de precedência de descarte (M). A especificação atual define $N = 4$ e $M = 3$ (Figura 3.3), embora, em uma situação real, nem todas elas possam vir a ser necessárias. O serviço fornecido por uma certa classe independe do serviço das demais classes, sendo função apenas dos recursos alocados para cada classe pelo sistema (Magalhães & Cardozo, 1999).

Um usuário pode contratar de um provedor um dos quatro serviços de encaminhamento diferenciado, cada um com três níveis de prioridade de descarte. Em situações de sobrecarga, o tráfego de uma classe superior tem menor probabilidade de sofrer congestionamento que o tráfego de uma classe inferior. O mecanismo de diferenciação aqui utilizado se baseia na prioridade de descarte: quando este for inevitável, primeiro se descartam os pacotes pertencentes ao serviço de melhor esforço e, só então, passa-se para os pacotes associados ao serviço de encaminhamento garantido, segundo sua classe e nível de precedência na classe. Portanto, os pacotes pertencentes ao serviço AF são os

[4]Usa-se também a denominação *Assured Service*.

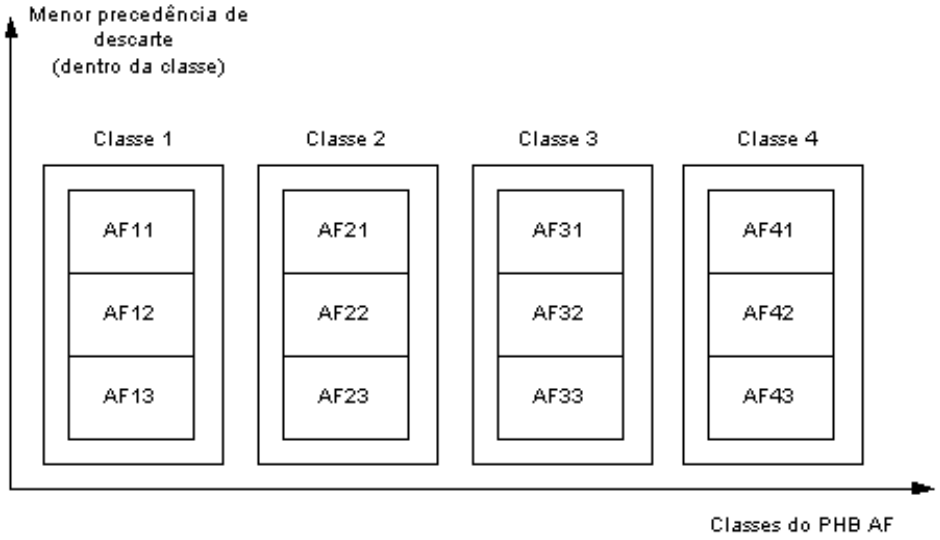


Figura 3.3: Classes do serviço de Encaminhamento Garantido (Kilkki, 1999)

últimos a serem descartados em situações de congestionamento.

É possível estabelecer uma relação entre os serviços das arquiteturas de serviços integrados e diferenciados. Em geral, pode-se mapear o Serviço Garantido (*IntServ*) para o Serviço de Encaminhamento Expresso (*DiffServ*) e o Serviço de Carga Controlada (*IntServ*) para o Serviço de Encaminhamento Garantido (*DiffServ*). Esta possibilidade de mapeamento é especialmente interessante em situações em que se utiliza as duas arquiteturas em conjunto para a provisão de uma QoS fim-a-fim (Seção 3.5.4).

3.5.2 *Service Level Agreements*

A fim de que um usuário possa receber serviços diferenciados de um provedor, é necessário que se estabeleça entre as partes um "acordo de serviço", conhecido como *Service Level Agreement* (SLA). Um SLA estabelece os critérios das políticas de QoS e define o perfil esperado do tráfego gerado pelas aplicações. Em outras palavras, define as classes de serviço contratadas e a quantidade de tráfego permitida em cada classe.

Ao ser enviado um tráfego, o domínio de origem tem a responsabilidade de policiá-lo (*policing*) e suavizá-lo nos pontos de saída (*egress points*), pois um tráfego fora do perfil contratado não receberá nenhum tipo de garantia de QoS ao chegar ao próximo ponto de ingresso (*ingress point*), em outro domínio. Quando um pacote deixa um domínio e segue para outro, às vezes pode ser necessário remarcar o seu campo DS, como resultado de um SLA estabelecido entre os dois domínios, embora o ideal seja que o tráfego experimente o mesmo nível de QoS ao longo de todo o percurso, da origem até o destino, o que nem

sempre é possível.

Os critérios empregados para a aplicação das políticas de QoS podem ser: data e hora, endereços de origem e destino, números de portas ou qualquer outra informação que possa ser extraída do conteúdo do tráfego, inclusive a contida nos cabeçalhos (Stardust, 1999b). Além desses aspectos, um SLA também pode especificar: procedimentos de tarifação e cobrança, serviços de criptografia e autenticação, procedimentos de renegociação dos parâmetros do SLA, ações a serem tomadas para o tráfego fora de perfil, entre outros (Vasiliou, 2000).

Um SLA pode ser estático, no sentido de que ele é negociado em bases mensais ou anuais, por exemplo, ou pode ser dinâmico, caso em que se faz necessário o uso de um protocolo de sinalização (como o RSVP) para a requisição de serviços de QoS sob demanda.

3.5.3 Protocolo MPLS

Uma abordagem alternativa aos serviços diferenciados, mas que tem recebido bastante atenção ultimamente, graças a sua simplicidade e eficiência, é o protocolo MPLS (*Multi-Protocol Label Switching*), especificado na RFC 3031 (Rosen *et al.*, 2001). Assim como o *DiffServ*, o MPLS marca os pacotes nos pontos de ingresso na rede e os desmarca nos pontos de saída, contudo, ao contrário do *DiffServ*, em que a marcação é apenas uma forma de atribuir prioridade aos pacotes dentro dos roteadores, o MPLS a utiliza efetivamente para fazer o roteamento, determinando qual o próximo *hop* a ser dado conforme o rótulo (*label*) colocado no pacote. O protocolo MPLS funciona integralmente nos roteadores, sem nenhum componente nos sistemas finais.

O rótulo de um pacote MPLS determina completamente qual caminho o mesmo deverá seguir na rede, permitindo se estabelecer canais (*pipes*) com largura de banda fixa, nos moldes dos circuitos virtuais de uma rede ATM ou *Frame Relay*. Como o MPLS opera entre as camadas de rede e enlace, ele pode funcionar sobre diferentes tipos de protocolos além do IP (Vasiliou, 2000).

A Figura 3.4 mostra o formato de um cabeçalho MPLS, o qual possui um rótulo de 20 bits. É este rótulo que será examinado pelos roteadores habilitados para MPLS (*Label Switching Routers* — LSRs), funcionando como um índice para uma tabela que especifica o próximo *hop* a ser dado e um novo rótulo para o pacote. Há também um campo de três bits reservado para fins experimentais, um bit usado para permitir o aninhamento de rotas MPLS (*Stack Flag*) e um campo de 8 bits para estabelecer um tempo de vida para o pacote (*Time-to-Live* — TTL).

Um aspecto importante na implantação do MPLS está na forma de gerenciamento e distribuição dos rótulos entre os roteadores MPLS, a fim de garantir um significado comum para todos eles.

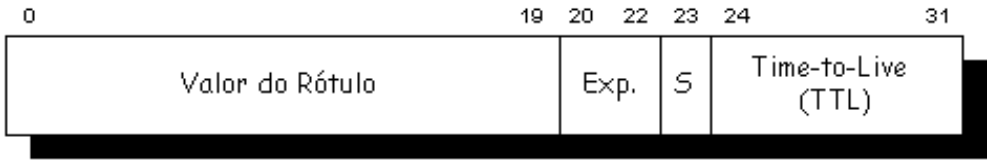


Figura 3.4: Layout do cabeçalho do protocolo MPLS

3.5.4 Comparação *IntServ* vs. *DiffServ*

As arquiteturas de serviços integrados e diferenciados não devem ser consideradas como competidoras, ao contrário, elas são complementares e podem ser usadas em conjunto, explorando-se as melhores características de uma e de outra. Entretanto, existem diferenças significativas entre as duas abordagens, as quais são comentadas a seguir.

Em primeiro lugar, destaca-se o nível de granulosidade em que cada mecanismo trabalha: o modelo *IntServ* tem seu foco em fluxos individuais fim-a-fim, entre uma origem e um destino; já o modelo *DiffServ* lida com o conceito de agregação de fluxos, os quais constituem as classes de serviço. Daí, tem-se que o gerenciamento das informações de estado associadas aos fluxos é muito mais complexo no *IntServ*, pois, em uma rede como a Internet, podem-se ter centenas de milhares de fluxos concorrentes, o que ultrapassa a capacidade de memória e processamento dos roteadores atuais. No futuro, com links da ordem de gigabits ou terabits, esse gerenciamento se tornará ainda mais difícil (Dovrolis & Ramanathan, 1999). No modelo *DiffServ*, a quantidade de informações de estado é proporcional ao número de classes de serviço definidas, que é muito inferior ao número de fluxos (Xiao & Ni, 1999). Por essa razão, considera-se que a arquitetura *IntServ* é mais adequada para a provisão de uma QoS intra-domínio, em redes locais e a *DiffServ*, com uma melhor escalabilidade, presta-se mais a redes do porte da Internet (contudo, é possível usar as duas abordagens em conjunto, como será comentado).

A arquitetura *IntServ* representa uma ruptura com a abordagem tradicionalmente empregada nas redes IP, pois ela leva complexidade para o núcleo da rede; já a arquitetura *DiffServ* é mais natural, pois coloca a complexidade nos pontos externos da rede e deixa os roteadores simples, preservando as diretrizes do projeto original da Internet. O *IntServ* necessita de um protocolo de sinalização como o RSVP para funcionar, o qual precisa incluir informações de estado nos roteadores ao longo do caminho; o *DiffServ* apenas faz uma redefinição de um campo do cabeçalho IP, o que agiliza sua implantação. Portanto, a coordenação da QoS, no *IntServ*, é feita fim-a-fim e, no *DiffServ*, as decisões são tomadas localmente, a cada *hop*.

A classificação dos pacotes no *IntServ* é feita a partir de vários campos, como os endereços de origem e destino, números de portas e o protocolo utilizado, individualmente para cada fluxo; já o *DiffServ* usa apenas o campo DS do protocolo IP para este fim, sobre um conjunto de fluxos relacionados (agregação).

O gerenciamento da rede, nos serviços integrados, é feito de forma similar a redes comutadas por circuito, como a ATM; nos serviços diferenciados, o mesmo é semelhante ao já encontrado em redes IP.

O modelo *IntServ* está mais voltado para as demandas de serviço de cada aplicação, procurando satisfazê-las fim-a-fim e, por causa disso, exige uma reserva de recursos antes do início de qualquer transmissão de dados. Em contrapartida, o modelo *DiffServ* visualiza mais as propriedades do tráfego gerado, usando um mecanismo de priorização para classificá-lo segundo suas características de demanda de QoS. É responsabilidade das aplicações (ou de algum roteador de borda) marcar seus pacotes adequadamente a fim de que recebam o serviço de que precisam.

Por outro lado, a QoS fornecida pela arquitetura *IntServ* é a maior possível em uma rede baseada nos protocolos TCP/IP, sendo oferecidas garantias deterministas de perda, atraso e *jitter*. Já a arquitetura *DiffServ*, embora possa fornecer QoS absoluta, baseia-se mais na idéia de diferenciação de serviços relativa entre as classes.

Cooperação entre as Arquiteturas

Para concluir esta seção, cabem alguns comentários acerca da utilização dessas arquiteturas de provisão de QoS em conjunto, o que parece bastante promissor. Na prática, é improvável que qualquer uma das soluções apresentadas (RSVP, *DiffServ*, MPLS) seja utilizada independentemente para fornecer QoS fim-a-fim. O que geralmente ocorre é a combinação dessas arquiteturas no caminho entre o emissor e o receptor a fim de que se possa extrair o melhor de cada uma delas (Stardust, 1999a).

A recomendação geral é que o modelo *IntServ* (RSVP) pode ser melhor empregado nos pontos de ingresso na rede, com uma alta granulosidade, a fim de especificar as demandas de QoS das aplicações em termos de largura de banda, perda de pacotes, atraso e *jitter*. Os roteadores de borda nesses pontos poderão, então, mapear essas demandas para as classes de serviço do *DiffServ*, através da marcação adequada dos pacotes. Nesta solução híbrida, o RSVP se beneficiaria dos aspectos de agregação da arquitetura *DiffServ* e esta, por sua vez, se utilizaria do mecanismo de sinalização de QoS do RSVP, a fim de proporcionar uma QoS absoluta fim-a-fim (Magalhães & Cardozo, 1999). Assim, o *DiffServ* teria seu melhor emprego no núcleo (*backbone*) da Internet, por ser uma solução mais leve e de melhor escalabilidade, enquanto o RSVP ficaria restrito às bordas da mesma. É para esta direção que se dirigem os estudos atuais dentro do grupo de trabalho *DiffServ*, na IETF (Stardust, 1999a).

Para finalizar, outra possibilidade é o uso de MPLS com *DiffServ*, o que não parece muito complicado, pela similaridade das duas abordagens. A solução seria mapear um valor particular do campo DS (uma classe de serviço) para um rótulo MPLS, tendo-se o cuidado de reservar recursos em cada roteador MPLS a fim de emular as prioridades do

DiffServ. O RSVP também teria seu lugar neste cenário, reservando recursos para uso dedicado ao longo dos caminhos estabelecidos pelo MPLS (Vasiliou, 2000).

3.6 Serviços Diferenciados em Nível de Aplicação

Até este ponto, foram estudadas as formas de garantir QoS em nível de rede, particularmente as abordagens de serviços diferenciados e integrados. Contudo, como já foi comentado, qualquer esforço no sentido de fornecer QoS terá que ter a cooperação de todos os elementos do sistema, de cima a baixo, de fim a fim. Sendo assim, os *hosts* finais constituem-se em elementos essenciais neste cenário, uma vez que são eles, em última análise, que vão atender os usuários.

A discussão de QoS em nível de rede, nas seções anteriores, não foi sem motivo, uma vez que se pode aproveitar parte do conhecimento já adquirido e testado na área, desta feita na construção de servidores web. Pode-se perceber que um servidor web não preparado para oferecer QoS anulará quaisquer esforços que tenham sido empreendidos pela rede nesse sentido, pois ele tratará todas as solicitações que receber por igual, ignorando a prioridade relativa das mesmas. Infelizmente, a maior parte dos servidores atuais ainda trata todas as solicitações uniformemente, sem nenhum tipo de diferenciação, segundo uma disciplina FCFS (*First-Come First-Served*), o que muitas vezes pode transformá-los no ponto de estrangulamento do sistema. Por essa razão, a provisão de QoS na Web, de uma maneira ampla, não será possível se não se considerar o servidor web como um elemento essencial nesta cadeia, dotando-o dos mecanismos necessários para tal.

Nesta seção, serão examinados alguns trabalhos que se preocupam com o fornecimento de QoS em nível de aplicação, particularmente no contexto de servidores web, assunto que é o objetivo do presente projeto de pesquisa. Esse é um tema dos mais relevantes, na medida em que a presença das empresas na Web cada vez mais se confunde com sua imagem no mundo real. Não se está tratando aqui apenas de aplicações multimídia, como o tema de QoS pode levar a pensar, mas da utilização da Web como um meio para a condução de negócios, colocando-se o servidor web como o elemento central deste cenário, do qual serão exigidos requisitos de qualidade de serviço, disponibilidade, confiabilidade e segurança.

Serviços Diferenciados em Nível de Aplicação para Servidores Web

Os esforços atuais no sentido de fornecer múltiplos níveis de serviço na Internet concentram-se principalmente na rede e nos sistemas operacionais. Embora essas abordagens tendam a fornecer os resultados mais expressivos, na prática sua implantação em larga escala é difícil, pela impossibilidade de se fazer a atualização dos roteadores e sistemas operacionais por toda a Internet.

Eggert & Heidemann (1999) propõem mecanismos do lado servidor, tão somente em nível de aplicação, que, segundo os autores, mostram benefícios significativos, apesar de sua aparente simplicidade. A idéia principal é dividir as solicitações que chegam ao servidor em *foreground* e *background*, onde as solicitações de *background* recebem um serviço de "menor esforço" (*less effort*), ou seja, elas só serão processadas e transmitidas aos clientes caso haja recursos ociosos disponíveis no servidor. Se não houver, poderão ser postergadas indefinidamente ou até mesmo descartadas.

Dessa forma, criam-se, na prática, duas classes de serviço: uma classe de *foreground* e outra de *background*. Os autores apontam algumas situações em que esta solução pode ser aplicada: a primeira é no atendimento ao tráfego especulativo na Web, resultante de operações de cache antecipado (*prefetching*) ou de *pushes* feitos pelo servidor. Este tipo de tráfego, argumentam, não é primordial e pode ser descartado se necessário. Outra possibilidade é atribuir diferentes prioridades às requisições segundo o tipo de objeto requisitado, por exemplo, o código HTML poderia ter prioridade sobre imagens e *applets*. Finalmente, é sugerido que a QoS diferenciada poderia ser resultado de políticas externas ao servidor, no caso favorecendo-se usuários pagantes em detrimento dos não-pagantes.

Esses mecanismos foram implementados sobre o código do servidor Apache, usando-se estratégias como a limitação do número de processos em *background*, a diminuição da prioridade de processos em nível de sistema operacional e da sua taxa de transferência na rede. São mostrados resultados experimentais que validam os mecanismos implementados, sendo feita também uma comparação com o desempenho do servidor Apache não modificado.

Provisão de Diferentes Níveis de Serviço em *Web Hosting*

O trabalho de Almeida *et al.* (1998) está voltado à área de *web hosting*, ou seja, o caso de um provedor de Internet que possui vastos recursos e abriga sites de diferentes empresas, instituições e indivíduos. A diferenciação de QoS se dá entre os diferentes sites hospedados no provedor. Evidentemente, nesta situação, a existência de diferentes níveis de serviço é mais do que necessária, pois as empresas esperam que as requisições dos seus clientes sejam atendidas com uma qualidade compatível com a quantia que foi paga pelos serviços de hospedagem do site.

O mecanismo usado para a provisão de QoS é o escalonamento baseado em prioridades, tanto no modo de usuário quanto de kernel, diferentemente do trabalho de Eggert & Heidemann (1999), que só emprega mecanismos em nível de aplicação. São oferecidos dois níveis de QoS: alta prioridade e baixa prioridade. A primeira classe de serviço é para clientes que pagam uma taxa de hospedagem e a segunda, para aqueles que a tem de graça.

O primeiro passo consiste em classificar as requisições dos clientes em categorias, algo

que normalmente não é feito pelos servidores web e que demandou a modificação do código do servidor Apache. No nível de usuário, um processo escalonador decide a ordem em que as requisições serão atendidas, limitando também o número de processos em cada categoria. Num segundo momento, as prioridades das requisições são mapeadas nas prioridades dos processos HTTP que as estão atendendo, o que requereu alterações no kernel do Linux. Foram implementadas duas políticas de escalonamento: uma conservadora (*work-conserving*), que permite que processos de baixa prioridade executem como processos de alta prioridade, caso haja recursos disponíveis na categoria superior e uma não-conservadora (*non-work-conserving*), que não permite a migração de processos de uma categoria para outra.

Os resultados obtidos mostram que restringir o número de processos concorrentes revela-se um mecanismo eficiente para a obtenção de QoS diferenciada, com a política não-conservadora apresentando os melhores resultados.

Suporte a Qualidade de Serviço em Servidores HTTP

Os trabalhos de Eggert e de Almeida realizam a diferenciação de serviços em alto nível, considerando apenas processos de alta e baixa prioridade. Já Pandey *et al.* (1998) propõem uma arquitetura de serviços web que permite uma granulosidade mais fina no escalonamento das requisições. Nela, é possível personalizar como um servidor HTTP deve responder às requisições dos clientes, através da atribuição de prioridades e da alocação dos recursos do servidor às requisições de páginas.

As páginas web são modeladas como objetos e as requisições às mesmas são consideradas invocações de métodos. O servidor, portanto, torna-se um sistema que gerencia a execução de várias invocações de métodos, dentro das restrições de QoS vigentes. É especificada uma linguagem, denominada WebQoSL, que permite ao administrador do sistema informar detalhes de alocação dos recursos do sistema às páginas, determinar a disponibilidade de conjuntos de páginas e especificar garantias quanto à taxa de transferência das mesmas.

A arquitetura implementada consiste de cinco servidores web distribuídos e um *daemon* controlador de QoS. Quando um servidor recebe uma requisição, ele envia uma mensagem ao *daemon* perguntando qual a ação a ser tomada. O *daemon* pode responder de três formas: processar a requisição, negar o processamento ou redirecionar a requisição. Essas decisões são tomadas a partir das informações mantidas pelo *daemon* a respeito da utilização dos recursos do sistema, com dois objetivos em mente: satisfazer as restrições de QoS e otimizar o uso dos recursos do sistema. Vários experimentos foram realizados a fim de validar a arquitetura proposta.

Provisão de Serviços Diferenciados a partir de um Servidor Web

Chen & Mohapatra (1999) apresentam uma solução para um servidor web com diferenciação de serviços que consiste de um servidor web distribuído com roteamento de tarefas centralizado. O servidor é formado por quatro componentes principais: um iniciador de tarefas, um escalonador, um conjunto de servidores de tarefas e o canal de comunicação. As requisições são recebidas pelo iniciador, que pode aceitá-las ou rejeitá-las, caso a capacidade do sistema tenha sido excedida. Uma vez aceita, a requisição recebe um nível de prioridade, dado pelo escalonador e é atendida por um dos servidores de tarefas. O canal de comunicação modela a carga na rede, que é um dos parâmetros levados em conta para o escalonamento.

A abordagem aqui empregada não é a experimentação, como nos trabalhos anteriores. Neste caso, foi construído um modelo de rede de filas para o sistema, o qual é resolvido por simulação, utilizando-se, como entrada, *traces* de acessos a servidores web comerciais. Seus resultados demonstraram, mais uma vez, a eficácia do escalonamento baseado em prioridades na diferenciação de serviços, pois a degradação do desempenho das tarefas de mais alta prioridade aconteceu a um nível de utilização do servidor bem maior do que no caso das tarefas de mais baixa prioridade. Além disso, as requisições de alta prioridade experimentaram um baixo atraso mesmo quando o sistema se aproximou da utilização completa.

Análise das Arquiteturas

Os trabalhos apresentados representam abordagens diferentes à questão de como fornecer QoS diferenciada em servidores web. Embora apresentem algumas similaridades, cada um tem mecanismos e detalhes de implementação próprios.

O trabalho de Eggert & Heidemann (1999) mostra claramente que é possível fornecer serviços diferenciados em servidores web empregando-se apenas técnicas em nível de aplicação e foi a motivação inicial para este projeto de pesquisa. Almeida *et al.* (1998) apresentam uma solução mais sofisticada, com o uso de mecanismos para diferenciação de serviços em nível de kernel do sistema operacional, obtendo bons resultados. O trabalho de Pandey *et al.* (1998) propõe uma abordagem com o uso de uma sintonia fina para a provisão de QoS diferenciada, definindo até mesmo uma linguagem de especificação para este fim. Finalmente, Chen & Mohapatra (1999) desenvolvem um modelo em rede de filas para um servidor web diferenciado e o validam através de simulação, ao contrário dos outros trabalhos que se baseiam em experimentação.

Além dos artigos comentados nesta seção, vários outros abordando a diferenciação de serviços em servidores web foram objeto de estudo nesta revisão bibliográfica, como os trabalhos de Abdelzaher & Bhatti (1999), Banga *et al.* (1999), Bhatti & Friedrich

(1999), Cardellini *et al.* (2001b), Casalicchio & Colajanni (2000), Dovrolis & Stiliadis (1999), Kanodia & Knightly (2000), Rao & Ramamurthy (2001) e Vasiliou & Lutfiyya (2000). De uma maneira geral, pôde-se notar que os trabalhos estudados utilizam apenas mecanismos do lado servidor com o intuito de realizar a diferenciação de serviços. Nenhum deles contempla o mapeamento entre a QoS no servidor e os mecanismos em nível de rede, abordados neste capítulo.

3.7 Considerações Finais

Este capítulo apresentou uma visão geral da área de qualidade de serviço, abordando seus conceitos básicos. Deu-se especial atenção à possibilidade de utilização de QoS na Internet, a fim de que a mesma possa dar suporte a novos tipos de aplicações, oferecendo níveis de serviço superiores aos do modelo atual de melhor esforço. Foram destacadas as arquiteturas de serviços integrados e diferenciados, com ênfase nesta última, concluindo-se com uma revisão de alguns trabalhos na área de diferenciação de serviços em servidores web.

O próximo capítulo detalha uma das principais propostas desta tese: uma arquitetura para um servidor web com suporte à diferenciação de serviços.

Capítulo

4

Servidor Web com Diferenciação de Serviços (SWDS)

4.1 Introdução

Esta tese de doutorado visa estudar o fornecimento de serviços diferenciados na Web, em nível de aplicação, particularmente no contexto de servidores web. A motivação para este tema nasce da constatação de que o modelo atual de serviços na Internet e na Web, em particular, trata todas as requisições de forma equivalente, sem nenhum tipo de diferenciação ou priorização. Contudo, verifica-se que nem todos os tipos de tráfego e transações são equivalentes ou têm a mesma prioridade para os usuários (Dovrolis & Ramanathan, 1999). Assim, é essencial fornecer diferenciação de serviços com diferentes níveis de qualidade para diferentes tipos de requisições (Kant & Mohapatra, 2000).

Entretanto, como comentado anteriormente, a provisão de uma QoS abrangente, em todos os níveis, não é algo que se possa alcançar com esforços isolados. É necessário que se tenha um comprometimento fim-a-fim, com a colaboração de todos os elementos envolvidos no processo. Daí, a necessidade de que os servidores web sejam também capazes de reconhecer as diferentes demandas dos usuários e aplicações, de modo a estarem aptos a fornecer-lhes múltiplos níveis ou classes de serviço, conforme a sua necessidade.

A introdução de características de diferenciação de serviços em servidores web tem apresentado resultados promissores e faz-se mais do que necessária, pois são os servidores, em última instância, que irão atender as requisições dos usuários. Pouco adianta ter-se uma rede com provisão de QoS, se os elementos finais não forem capazes de reconhecê-la.

Este capítulo propõe um modelo para um Servidor Web com Diferenciação de Serviços, descrevendo em detalhes seus principais componentes, parametrização e forma de validação, feita através de simulação dirigida por *traces*. Para a geração de carga de trabalho, são utilizados *logs* de acesso a servidores web reais, os quais são aqui discutidos. É fornecida também uma visão geral dos algoritmos de controle de admissão e diferenciação de serviços propostos nesta tese. Finalmente, são discutidos alguns cenários de uso para serviços diferenciados.

4.2 Modelo do Servidor Web com Diferenciação de Serviços

Os servidores web atuais tratam as requisições que recebem segundo uma disciplina FCFS (*First Come, First Served*), ou seja, é mantida uma única fila de espera onde cada requisição aguarda o momento de ser atendida, de acordo com sua ordem de chegada. Embora diferentes esquemas de controle de concorrência possam ser implementados no servidor, visando agilizar o atendimento das requisições, este ocorre em geral de maneira uniforme, sem considerar as particularidades e a urgência de cada tipo de requisição.

Esta seção descreve um modelo para um *Servidor Web com Diferenciação de Serviços* (SWDS) (Teixeira *et al.*, 2004b). Esse modelo, uma contribuição original desta tese, representa uma arquitetura de servidor web cujo objetivo é fornecer serviços diferenciados a seus clientes, de acordo com suas necessidades e segundo requisitos de QoS previamente acertados. A arquitetura do servidor SWDS compõe-se dos seguintes módulos: um Classificador, um módulo de Controle de Admissão e um *cluster* de processos ou servidores web (Figura 4.1).

O *Classificador* é o elemento responsável por receber as requisições que chegam ao sistema e subdividi-las em classes de serviço, segundo critérios pré-estabelecidos (Seção 4.4). Após esta fase, a nova requisição entra no sistema em uma categoria determinada e recebe um tratamento condizente com a classe à qual pertence.

O *Controle de Admissão* recebe as requisições já classificadas e gerencia a sua aceitação pelo servidor, levando em conta as políticas de atendimento vigentes e as informações da carga de trabalho do sistema. Caso o mesmo esteja sobrecarregado, uma requisição poderá ser rejeitada (Descarte) ou ter suas exigências de qualidade de serviço relaxadas (Negociação), a fim de que possa ser aceita em uma classe de prioridade inferior.

Após ser admitida no sistema, a requisição é atribuída a um dos nós do *cluster de servidores web*, sendo atendida conforme o algoritmo de escalonamento ou diferenciação de serviços vigente. Após a conclusão do processamento, os resultados são retornados ao cliente que originou a solicitação.

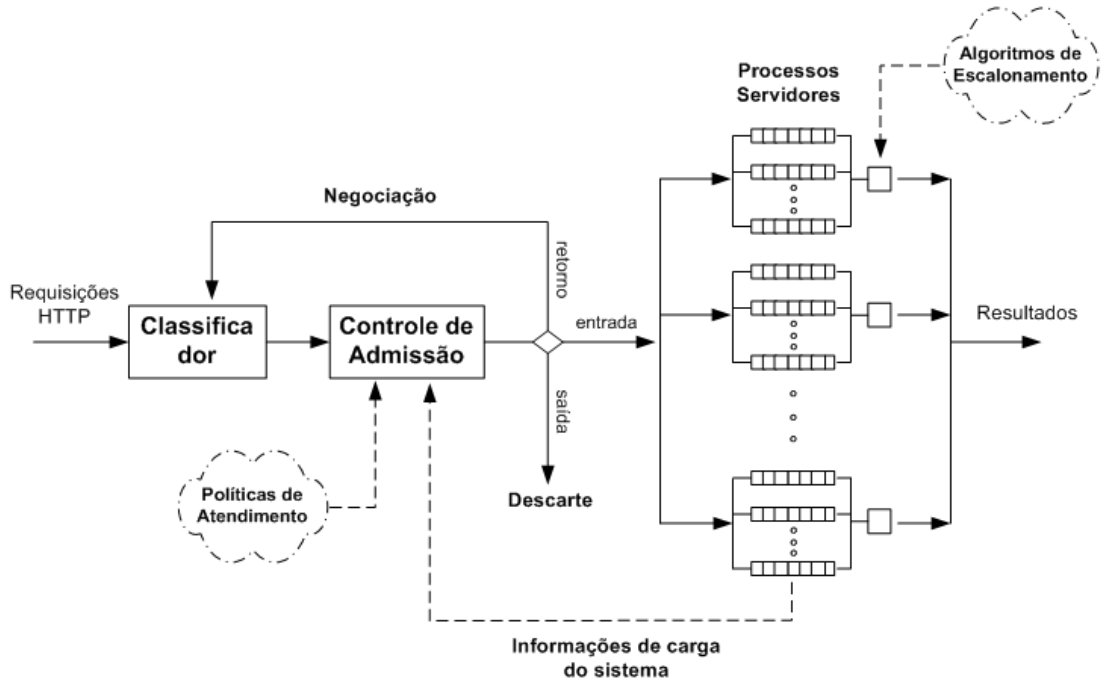


Figura 4.1: Servidor Web com Diferenciação de Serviços (SWDS)

Para os propósitos deste trabalho, cada nó do *cluster* é considerado como um servidor web convencional, composto de CPU, disco e interface de rede. Cada nó possui múltiplas filas de prioridade, a fim de acomodar as diferentes classes de serviço. Considera-se que o *cluster* é formado por servidores web, mas nada impede que se faça uma abstração dos mesmos para processos ou até mesmo CPUs em um computador paralelo, pois o modelo não exige que a arquitetura seja necessariamente formada por máquinas dispostas em um sistema distribuído. Igualmente, não é suposta nenhuma plataforma de hardware ou sistema operacional em particular.

O modelo do servidor SWDS constitui-se em uma proposta inovadora para o fornecimento de serviços diferenciados em servidores web, integrando características de QoS às arquiteturas de servidores tradicionalmente encontradas. Comparado aos trabalhos discutidos na Seção 3.6, destaca-se por sua flexibilidade, a qual é conferida pelos variados mecanismos de controle de admissão e diferenciação de serviços. Dessa forma, o servidor SWDS consegue adaptar-se a diferentes perfis de carga de trabalho e distribuições dos clientes pelas classes. O modelo tanto pode ser usado como ponto de partida para a implantação de infra-estruturas de servidores web distribuídos, voltadas ao fornecimento de serviços diferenciados, quanto para o desenvolvimento de um programa de servidor web com características de diferenciação de serviços, caso em que os nós do *cluster* seriam abstraídos para processos ou *threads*, executando em um ou múltiplos processadores.

4.3 Experimentação do Modelo

4.3.1 Parametrização

O foco dos experimentos realizados neste trabalho é no desempenho do servidor SWDS e na sua capacidade de fornecer serviços diferenciados aos seus clientes, portanto não são modelados detalhes da rede externa, nem de interconexão entre os componentes do servidor. Assume-se que o meio de comunicação não representa um gargalo do sistema, havendo largura de banda suficiente na rede. Cada nó do *cluster* é modelado separadamente, com sua própria CPU, disco e interface de rede. Adicionalmente, o servidor SWDS utiliza os módulos de Classificação e Controle de Admissão, os quais têm sua própria influência no desempenho global do sistema.

A capacidade de processamento do Classificador é definida como 8000 requisições/seg e a do módulo de Controle de Admissão como 4000 requisições/seg. Para o cálculo do tempo de serviço das requisições estáticas, os discos dos servidores são parametrizados com uma taxa de transferência de 37 MBps e uma latência de 8,5 ms, tomando-se como referência um disco IBM Deskstar 75GXP (IBM, 2003). Assume-se também que cada servidor do *cluster* possui uma interface de rede Fast Ethernet com capacidade real de 80 Mbps. O tempo de serviço das requisições dinâmicas é assumido como 10 ms. A parametrização do modelo tomou por base observações próprias, feitas a partir de *benchmarks* realizados na rede do LaSDPC[1], bem como exemplos encontrados em outros trabalhos da área (Cardellini *et al.*, 2001b; Casalicchio & Colajanni, 2000; Chen & Mohapatra, 1999; Hu *et al.*, 1999; Menascé & Almeida, 2003). A Tabela 4.1 resume a parametrização do modelo.

Parâmetro	**Valor**
Capacidade do Classificador	8000 req/seg
Capacidade do Controle de Admissão	4000 req/seg
Taxa de transferência do disco	37 MBps
Latência do disco	8,5 ms
Capacidade da interface de rede	80 Mbps
Tempo de serviço das requisições dinâmicas	10 ms

Tabela 4.1: Parâmetros do modelo (SWDS)

4.3.2 Validação do Modelo

A abordagem escolhida para a validação do modelo do servidor SWDS foi a simulação, por se tratar de uma técnica adequada e devido à larga experiência do Grupo de Sistemas Distribuídos e Programação Concorrente na área. A simulação é uma abordagem bastante flexível, permitindo verificar diferentes configurações do modelo prontamente. Os

[1]Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP.

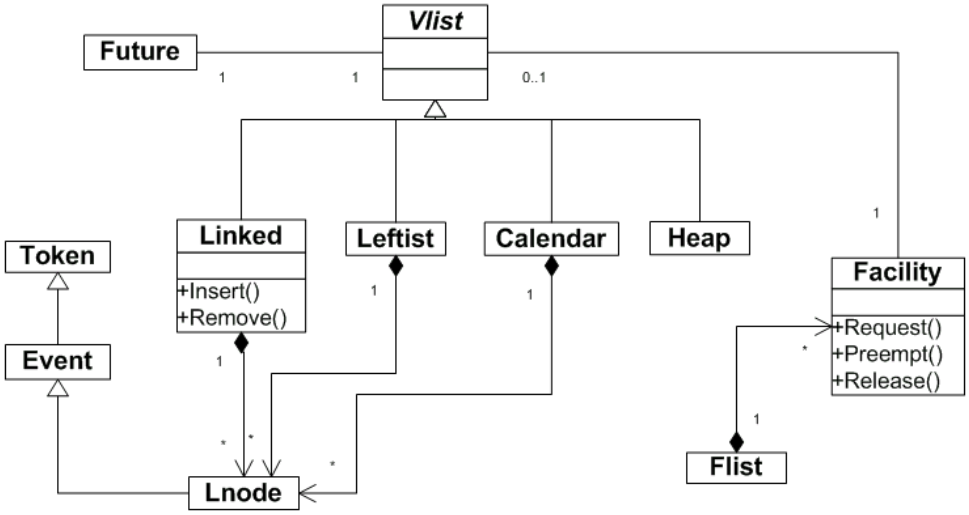


Figura 4.2: Diagrama de Classes (SimPack)

resultados podem ser obtidos com mais rapidez do que se fossem realizados experimentos tradicionais, por exemplo, com *benchmarks* como WebStone ou HTTPerf e softwares de servidor web como o Apache, os quais certamente precisariam ser modificados para incorporar as características de diferenciação de serviços.

O simulador utilizado foi o `SimPack` (Cubert & Fishwick, 1995), desenvolvido a partir da biblioteca `SMPL` (MacDougall, 1987). O `SimPack` possui uma biblioteca escrita em C++, denominada `Sim++`, composta por uma ampla gama de métodos voltados à *simulação de redes de filas, orientada a eventos discretos*. O simulador é distribuído com seu código fonte completo, incluindo inúmeros exemplos de aplicações. Ele foi escolhido para este trabalho pela experiência anterior do Grupo com o `SMPL`, o que proporcionou uma rápida curva de aprendizado, já que a nova biblioteca em C++ é bastante semelhante à anterior, escrita em C. Como o código fonte da biblioteca `Sim++` está disponível livremente, é possível modificá-lo se necessário, característica decisiva para a sua escolha.

Embora a documentação da API do simulador seja bastante útil para o desenvolvimento de programas, são fornecidas poucas informações acerca de sua organização interna. Sendo assim, foi preciso fazer uma análise detalhada do código fonte a fim de levantar o diagrama de classes em UML da biblioteca `Sim++`, mostrado na Figura 4.2. Existe uma classe base, denominada VList (*Virtual List*), que é uma classe abstrata da qual são derivadas todas as outras classes que necessitam fazer processamento de listas. Dentre elas, a classe Future, que implementa a lista de eventos futuros (LEF) do simulador e a classe Facility, que representa os recursos (centros de serviço).

Quando a LEF é criada no início da simulação, é preciso definir de qual tipo ela será: Linked, Leftist, Calendar ou Heap. A classe Linked implementa uma lista dinâmica duplamente encadeada com um ponteiro adicional para o último elemento; a classe Heap, um *heap* através de um vetor estático; a classe Leftist, uma árvore binária alocada dinamica-

mente; e a classe Calendar, uma tabela *hash*. Em todas as simulações realizadas, optou-se por usar uma LEF definida segundo a classe Linked, cujo acesso tem complexidade linear. Os objetos da classe Facility têm obrigatoriamente que usar uma lista do tipo Linked.

Cada nó de uma das listas acima (com exceção do tipo Heap) é um objeto da classe Lnode, a qual é derivada por herança da classe Event que, por sua vez, consiste em um Token com mais alguns atributos, tais como o instante de ocorrência do evento e sua identificação. A classe Lnode é uma classe aninhada, definida de acordo com o tipo de lista implementado.

Informações detalhadas sobre o simulador `SimPack` e a biblioteca `Sim++` estão disponíveis em Cubert & Fishwick (1995).

4.3.3 Geração de Carga de Trabalho

Um dos problemas existentes na simulação de sistemas relacionados à Web consiste na geração de carga de trabalho para os experimentos. Estudos anteriores mostraram que a Web apresenta uma grande variabilidade nas características da sua carga de trabalho, sendo difícil definir um modelo de carga que se aplique a todas as situações, como comentado na Seção 2.7. Foi observado, por exemplo, que a distribuição dos tamanhos de arquivo possui características de cauda pesada (Arlitt & Williamson, 1996; Barford *et al.*, 1998) e também que o tráfego na rede se apresenta em rajadas, em diferentes escalas de tempo, tendo um comportamento auto-similar (Crovella & Bestavros, 1997). Portanto, dada a dificuldade de produzir sinteticamente uma carga com essas características, decidiu-se empregar neste trabalho uma *simulação dirigida por traces*, sendo utilizados *logs* de acesso a servidores web reais para a geração de carga. Os arquivos de *log* foram coletados durante a Copa do Mundo 98, na França e estão livremente disponíveis para consulta em (Internet Traffic Archive, 1998).

Nos experimentos, foi usado o *log* correspondente a um período do dia 11 de junho de 1998, o qual registrava acessos a 27 servidores diferentes, com 7 milhões de registros. A vantagem de se usar este *log* nos experimentos está no fato de que ele representa uma carga de trabalho atualizada da Web, além de refletir um padrão de acesso a um site bastante solicitado (`www.france98.com`). Durante a simulação, os registros do *log* são lidos seqüencialmente e usados como entrada para o modelo. Cada registro do *log* corresponde a uma requisição HTTP emitida por um certo cliente e é processado segundo a parametrização descrita na Seção 4.3.1.

Caso a requisição HTTP seja estática, seu tempo de serviço (S_{estat}) é computado tomando-se como base o tamanho do arquivo solicitado, informado em cada registro do *log*. Para tanto, utilizam-se a taxa de transferência e a latência do disco (Tabela 4.1), segundo a fórmula:

$$S_{disco} = latencia + \frac{tam\_arq}{taxa\_disco} \tag{4.1}$$

Neste caso, considera-se que o tempo de processamento na CPU é desprezível se comparado ao tempo gasto para a leitura do arquivo no disco, não sendo incluído no cálculo do tempo de serviço total. Sendo assim, o tempo de serviço de uma requisição estática é aproximado por S_{disco}.

Quanto às requisições dinâmicas, estas têm a característica de ocupar por mais tempo a CPU do servidor web, devido às diversas operações necessárias para gerar dinamicamente uma página em HTML. Assume-se que o tempo de serviço para as requisições dinâmicas, S_{dinam}, é igual a 10 ms, conforme a parametrização da Tabela 4.1. Este tempo de serviço é genérico e engloba também o tempo de acesso a disco.

O resultado de uma requisição HTTP, quer seja ela estática ou dinâmica, precisa ser enviado de volta ao cliente. O cálculo do tempo de serviço na interface de rede (S_{nic}) leva em consideração a capacidade da interface, 80 Mbps, informada na Tabela 4.1, a qual corresponde à largura de banda útil de uma rede *Fast Ethernet*. Portanto, o tempo de serviço para a transferência de um arquivo pela interface de rede é dado por:

$$S_{nic} = \frac{tam\_arq}{taxa\_rede} \tag{4.2}$$

Note-se que os tempos de serviço (S_i) acima referem-se ao tempo de processamento em cada dispositivo i (CPU, disco ou rede), não sendo incluído no cômputo dos mesmos os tempos de espera em fila (W_i), que são função da taxa de chegada de requisições a cada dispositivo e de sua taxa de serviço (Jain, 1991, Cap. 31). Dessa forma, o tempo de residência R_i em um dispositivo i é dado genericamente por:

$$R_i = S_i + W_i \tag{4.3}$$

O tempo de residência ou atendimento é calculado pelo próprio simulador `SimPack` e depende da carga imposta ao dispositivo.

Concluindo, para as requisições estáticas o tempo de atendimento de uma requisição HTTP corresponde à:

$$R_{estat} = R_{disco} + R_{nic} \tag{4.4}$$

onde $R_{disco} = S_{disco} + W_{disco}$ e $R_{nic} = S_{nic} + W_{nic}$.

Para as requisições dinâmicas, o tempo de atendimento é dado por:

$$R_{dinam} = R_{dinam} + R_{nic} \tag{4.5}$$

onde $R_{dinam} = S_{dinam} + W_{dinam}$.

Vale acrescentar que os tempos de atendimento acima referem-se apenas à parcela de processamento nos nós do *cluster* de servidores web, pois os tempos de processamento

no Classificador e no Controle de Admissão são calculados a partir dos parâmetros da Tabela 4.1 e independem do fato de uma requisição ser estática ou dinâmica.

Finalmente, a distribuição dos tempos de chegada é obtida a partir dos próprios *timestamps* das requisições HTTP contidas no *log*. Um *log* HTTP tem uma resolução de um segundo, ou seja, esta é a menor fração de tempo com que se registra a ocorrência de um evento, portanto pode ocorrer coincidência de *timestamps* dos registros. Neste caso, assume-se que as chegadas estão uniformemente distribuídas em um intervalo entre 0 e 1 segundo. O valor superior do intervalo pode ser variado a fim de gerar diferentes taxas de chegada para as requisições HTTP.

Análise do Log

Para dar suporte ao tráfego previsto para a Copa do Mundo, foi montada na época uma estrutura composta por 30 servidores web, distribuídos em quatro cidades diferentes, uma na França e três nos EUA. Os *logs* coletados, inicialmente em formato CLF (*Common Log Format*), foram convertidos para um formato binário, por razões de espaço de armazenamento. Seus registros representam a atividade de todos os servidores do *site* no período observado, entre 30 de abril e 26 de julho de 1998, num total de cerca de 1,3 bilhão de registros, constituindo-se na maior carga de trabalho da Web analisada até hoje. A base de dados, disponível em Internet Traffic Archive (1998), está dividida em arquivos de 7 milhões de registros cada, correspondentes a partes de um dia de coleta.

A grande maioria das requisições usa o método GET para obtenção dos dados, seguido por um percentual ínfimo de métodos HEAD e POST (Tabela 4.2), o que denota um predomínio de requisições estáticas de dados previamente armazenados nos servidores. A análise dos códigos de resposta mostra que a grande maioria dos acessos resultou em sucesso (200), enquanto o segundo código de resposta mais comum foi o 304, correspondente a solicitações de objetos que já se encontram em *cache* e não foram modificados desde o último acesso. Encontrou-se também um número insignificante de erros devido a URLs incorretas (404) (Tabela 4.3).

Método	**Requisições (%)**	**Dados Transferidos (%)**
GET	99,88	99,62
HEAD	0,10	0,30
POST	0,02	0,08

Tabela 4.2: Distribuição das requisições por método HTTP
(Arlitt & Jin, 1999)

A análise da distribuição das requisições por tipo de arquivo (Tabela 4.4) revela que quase todas as requisições foram de arquivos em HTML ou imagens. A maioria das

Código de Resposta	**Requisições (%)**	**Dados Transferidos (%)**
200 (Sucesso)	80,52	97,86
206 (Conteúdo Parcial)	0,09	2,08
304 (Não Modificado)	18,75	0,00
4xx (Erro do Cliente)	0,64	0,06
5xx (Erro do Servidor)	0,00	0,00

Tabela 4.3: Distribuição das requisições por código de resposta (Arlitt & Jin, 1999)

requisições restantes foram arquivos em Java. Uma parcela ínfima das requisições foi de natureza dinâmica. Em relação à quantidade de bytes transferidos, houve um predomínio de arquivos em HTML sobre as imagens, pois aqueles eram em geral bem maiores que estas. Além disso, muitas requisições de imagens resultaram em respostas sem dados, do tipo 304 (*Not Modified*), ou seja, o arquivo, por não ser modificado com freqüência, já se encontrava em alguma *cache* ao longo do caminho e não precisou ser enviado novamente pelo servidor. Nota-se também que os arquivos compactados, embora respondam por uma parcela muito pequena das requisições, corresponderam à cerca de um quinto do total de bytes transferidos. As requisições de arquivos de áudio e vídeo foram normalmente do tipo *streaming*, tendo sido atendidas por outros servidores, dos quais não se têm informações disponíveis. Uma descrição detalhada do *log* utilizado pode ser encontrada em Arlitt & Jin (1999).

Tipo de Arquivo	**Requisições (%)**	**Dados Transferidos (%)**
HTML	9,85	38,60
Imagens	88,16	35,02
Áudio	0,02	0,10
Vídeo	0,00	0,82
Compactados	0,08	20,33
Java	0,82	0,83
Dinâmicos	0,02	0,38
Outros	1,05	3,92

Tabela 4.4: Distribuição das requisições por tipo de arquivo (Arlitt & Jin, 1999)

4.4 Classificação das Requisições

O fornecimento de serviços diferenciados aos clientes tem como base a noção de classe de serviço. Cada classe agrupa um conjunto de clientes com necessidades de atendimento

bem definidas, que podem ser expressas, por exemplo, em termos de tempo de resposta médio, *throughput* a ser atingido, percentual de requisições completadas, utilização do sistema, dentre outros parâmetros.

A população de clientes de um servidor web é bastante variada, com diferentes necessidades e expectativas. Ao módulo Classificador, cabe a atribuição de identificar as características das requisições HTTP que chegam ao servidor SWDS e subdividi-las em classes de serviço. Cada classe supõe um determinado nível de serviço oferecido aos clientes, os quais podem contratar um serviço específico, de forma explícita, ou deixar que o próprio servidor decida em que classe alocar as requisições recebidas, segundo suas características.

Para a divisão em classes de serviço, podem ser usados os seguintes critérios:

- *Urgência de atendimento*: o tráfego que chega a um servidor web pode ser classificado como real ou especulativo. Tráfego real é aquele que surge como resultado de requisições emitidas, de fato, por um cliente. Tráfego especulativo são requisições emitidas geralmente de forma automática, por exemplo, a partir de um servidor *proxy* ou originadas por pré-buscas (*prefetching*);

- *Mecanismo de geração de conteúdo*: as requisições HTTP podem ser divididas em estáticas ou dinâmicas, segundo a forma de geração do conteúdo retornado ao cliente. As requisições dinâmicas podem ainda envolver sessões convencionais ou seguras;

- *Conteúdo solicitado*: o tipo de arquivo requisitado também pode ser um critério para a classificação das requisições, por exemplo: texto, código HTML, imagens, *applets*, *scripts*, áudio/vídeo, dentre outros;

- *Origem da requisição*: as solicitações podem ser classificadas entre locais (pertencentes ao mesmo domínio do servidor web) ou remotas (provenientes de outros domínios);

- *Políticas de atendimento diferenciadas*: os clientes podem ser divididos em classes de serviço segundo políticas externas, por exemplo, usuários pagantes/não-pagantes. Outra possibilidade é o caso de múltiplos *sites* armazenados no mesmo servidor (*web hosting*), onde os clientes são atendidos de acordo com o contrato de hospedagem firmado pelo proprietário do *site* (Eggert & Heidemann, 1999).

Os critérios comentados acima não são exaustivos e outros parâmetros poderiam ter sido considerados. Note-se que não cabe ao módulo Classificador decidir qual classe receberá atendimento preferencial, ele apenas designa uma dada requisição para uma certa classe. A diferenciação de serviços é, de fato, realizada pelos mecanismos descritos na Seção 4.6.

No estágio atual, as requisições que chegam ao servidor SWDS são classificadas através de uma distribuição de probabilidade acumulada e não pelos critérios aqui abordados. Isso é feito sem perda de generalidade do modelo, uma vez que, para os mecanismos de diferenciação de serviços definidos, é indiferente o critério utilizado para a classificação das requisições. O modelo do SWDS, porém, é flexível o bastante para permitir a utilização de mecanismos mais sofisticados de categorização.

4.5 Controle de Admissão

4.5.1 Visão Geral

O controle de admissão das requisições é essencial para manter a carga do sistema dentro de níveis aceitáveis. Caso o servidor fique sobrecarregado, a diferenciação de serviços será comprometida, não sendo possível atingir os níveis de QoS pretendidos.

O Controle de Admissão do servidor SWDS foi projetado de modo a ser ampliado conforme a necessidade. É dotado de um módulo de coleta de informações de carga, uma área de variáveis globais e um módulo de mecanismos de controle, o qual pode ser incrementado com novos algoritmos a qualquer tempo. As políticas de atendimento norteiam a aplicação desses mecanismos, a fim de que sejam seguidas as diretrizes estabelecidas pelo administrador do sistema.

Atualmente, estão implementados os seguintes mecanismos de controle no SWDS:

- *Tamanho das Filas*: impõe um tamanho máximo para as filas dos servidores do *cluster*, recusando novas requisições quando esse limite for atingido;
- *Tempo de Resposta*: tenta manter o tempo de resposta abaixo de um certo limiar. Utiliza, para tanto, *buffers* independentes para cada classe de serviço, que controlam o número de clientes presentes no servidor;
- *Utilização do Sistema*: emprega uma média exponencialmente ponderada da utilização do *cluster* para o controle de admissão, a qual pode ser ajustada para considerar valores atuais ou o histórico da carga do sistema.

O módulo de Controle de Admissão é abordado detalhadamente no Capítulo 7.

4.5.2 Trabalhos Relacionados

Nesta seção, serão destacadas algumas soluções empregadas em trabalhos da área para o controle de admissão de requisições. Alguns dos projetos aqui comentados já foram

abordados com maiores detalhes na Seção 3.6, onde foi dada maior ênfase ao fornecimento de serviços diferenciados.

No trabalho de Chen & Mohapatra (1999), a admissão de uma requisição é decidida assim que ela chega ao sistema. Somente após sua aceitação, o escalonador atribuirá a ela um certo nível de prioridade. Caso a carga do sistema esteja acima de um limiar pré-estabelecido, a requisição será rejeitada, independentemente de sua classe.

Cardellini *et al.* (2001b) afirmam ser impossível honrar qualquer tipo de acordo de nível de serviço (SLA) firmado com os usuários se não for utilizado algum mecanismo para controlar a carga oferecida ao sistema. Sua proposta consiste em rejeitar a aceitação de novas requisições da classe baixa caso a carga exceda um certo limiar. As requisições da classe alta também são passíveis de rejeição, mas somente em situações altamente críticas. Todas as requisições que chegam ao sistema são colocadas em uma fila de espera global, de onde serão removidas posteriormente caso venham a ser recusadas. O mecanismo implementado utiliza como métrica o número máximo de conexões que pode ser estabelecido pelo *cluster* como um todo, sem que ocorra degradação do seu desempenho. Como sempre, a escolha correta do valor desse limiar é crítica para o bom funcionamento do controle de admissão.

Zhu *et al.* (2001) propõem um algoritmo de diferenciação de serviços capaz de se adaptar dinamicamente a variações na demanda de recursos, modificando o particionamento atual do *cluster* de servidores web caso necessário. Não são fornecidas garantias rígidas de QoS, porém é utilizado um controle de admissão para impedir que o servidor torne-se excessivamente sobrecarregado devido às requisições dos usuários. A alocação de recursos é formalizada como um problema de otimização com restrições, em que o sistema tenta estimar, a partir do histórico das taxas de chegada e das taxas de processamento em cada classe, qual o impacto de admitir uma dada requisição no sistema. Uma das restrições do problema de otimização diz respeito aos fatores de elasticidade (*stretch factors*) de cada classe, os quais não podem ultrapassar um limiar global previamente estabelecido. As requisições que violarem essa restrição serão rejeitadas, independentemente de sua classe.

Nessa mesma linha, seguem Chen *et al.* (2001) com seu algoritmo PACERS, o qual também realiza a alocação de recursos baseada no histórico da taxa de chegada e dos requisitos de processamento das tarefas. O trabalho ratifica que a utilização de prioridades é um mecanismo eficiente para a provisão de serviços diferenciados em servidores web, mas não é suficiente para garantir um tempo de resposta máximo para as requisições de maior prioridade em situações de carga excessivamente alta. O algoritmo proposto é capaz de estabelecer limites para o tempo de resposta das tarefas de diferentes classes, assegurar a disponibilidade de serviço para as tarefas de alta prioridade e preservar o *throughput* do sistema, em diferentes perfis de carga de trabalho. Os autores procuram explorar o comportamento cíclico da carga na web, subdividindo o tempo em intervalos de uma hora. Ao receber uma requisição, o sistema poderá aceitá-la desde que a soma do tempo de

serviço para as tarefas aceitas não exceda a capacidade do sistema no próximo intervalo. O tempo de serviço é estimado tomando-se por base a distribuição dos tipos de requisições feitas ao servidor, uma vez que as demandas de processamento tendem a ser semelhantes para requisições do mesmo tipo.

A proposta de Kanodia & Knightly (2000) consiste em um esquema que permite oferecer qualidade de serviço a múltiplas classes satisfazendo suas restrições de latência. É definido um servidor virtual de alto nível, baseado em mecanismos de controle de admissão, isolamento de desempenho e diferenciação de serviços, a partir do qual pode-se incrementar servidores web já existentes com características de QoS. Os autores utilizam um conceito de envelope, que representa estatisticamente a carga de requisições que chega a um servidor e a sua capacidade de serviço, evitando-se, dessa forma, a necessidade de modelar e avaliar cada componente do servidor (como CPU, disco, memória) individualmente. O controle de admissão, ao decidir pela aceitação de uma nova requisição em uma classe de serviço, procura determinar os efeitos dessa escolha não apenas na classe pretendida, mas também em todas as outras. Pretende, com isso, atingir um alto nível de isolamento de desempenho entre as classes, a fim de respeitar os limites de latência previamente estabelecidos.

De um modo geral, os trabalhos acima, assim como vários outros existentes, enfatizam a importância de um mecanismo de controle de admissão a fim de que se possa dar garantias de QoS aos usuários. Em muitos casos, principalmente quando a carga oferecida ao sistema torna-se muito elevada, a eventual recusa de novas requisições é necessária para honrar os compromissos de níveis de serviço firmados com os clientes. Alguns estudos lançam mão de históricos de uso e esquemas de predição para determinar o efeito da aceitação de uma nova requisição pelo servidor, além de preocupar-se com o *throughput* do sistema e com a justa distribuição dos recursos entre as classes de serviço. Um controle de admissão eficiente certamente contribui para uma maior estabilidade do sistema como um todo, permitindo-lhe oferecer níveis mais refinados de diferenciação de serviços a seus clientes.

4.6 Mecanismos de Diferenciação de Serviços

4.6.1 Visão Geral

Os mecanismos de escalonamento/diferenciação de serviços determinam qual será o tratamento dispensado pelo servidor SWDS aos seus clientes. Tanto podem interferir diretamente na disciplina das filas do *cluster* (por exemplo, introduzindo prioridades) quanto gerenciar a alocação dos nós às diferentes classes de serviço. Genericamente, podem ser considerados como algoritmos de escalonamento de requisições, com a característica adicional de favorecer determinadas classes de clientes consideradas privilegiadas.

Inicialmente, foram implementados dois algoritmos tradicionais de balanceamento de carga, apenas para se adquirir uma familiaridade maior com o simulador `SimPack` e o modelo do SWDS, pois esses algoritmos não visam fornecer serviços diferenciados:

- *Round Robin*: atribui as requisições aos servidores do *cluster* segundo um esquema de rodízio;
- *Shortest Queue First*: direciona as requisições para o servidor que tiver o menor número de processos em fila.

Finalmente, foram implementadas duas variantes de algoritmos de diferenciação de serviços no modelo:

- *Reserva de Recursos*: subdivide o *cluster* em partições de tamanho fixo, alocando-as às diferentes classes de serviço;
- *Escalonamento Baseado em Prioridades*: foram desenvolvidos dois algoritmos que realizam a diferenciação de serviços utilizando prioridades nas filas do *cluster*. Um deles emprega um mecanismo rigoroso e o outro, uma abordagem adaptativa.

Os mecanismos de diferenciação de serviços são apresentados com detalhes nos Capítulos 5 e 6.

4.6.2 Trabalhos Relacionados

Reserva de Recursos

Nesta seção, serão comentados alguns trabalhos que buscam fornecer serviços diferenciados através do particionamento de recursos entre as classes.

Banga *et al.* (1999) assumem que o servidor web está localizado em um único nó e propõem um mecanismo para o gerenciamento do uso dos componentes da máquina. Eles introduzem o conceito de repositório de recursos (*resource containers*), uma abstração de sistema operacional que contém logicamente todos os recursos do sistema (CPU, espaço de memória, *sockets* abertos, *buffers* de rede, etc.) necessários para realizar uma certa atividade. O *kernel* do sistema operacional contabiliza os recursos do sistema por repositório e usa essa informação no escalonamento das *threads* associadas com o mesmo. Nesse esquema, múltiplas *threads*, não necessariamente contidas em um mesmo processo, são associadas com um dado repositório e o escalonamento dos recursos é feito em relação às atividades, independentemente de como elas possam estar mapeadas sobre *threads* específicas. A classe, portanto, vem a ser a atividade ou o conjunto de atividades relacionadas.

O trabalho de Bhatti & Friedrich (1999) detalha uma arquitetura de servidor web para o fornecimento de serviços diferenciados, a qual propõe duas políticas de escalonamento

que alocam uma parcela fixa ou variável da capacidade de processamento do servidor (*Fixed and Shared Capacity*) para uma certa classe de serviço.

Pandey *et al.* (1998) apresentam uma versão modificada do servidor `httpd` do NCSA, habilitada para fornecer QoS. Eles empregam uma abstração de dutos (*pipes*), onde cada duto tem sua capacidade determinada em termos de bytes transferidos por segundo, podendo se subdividir em canais (*channels*), a fim de alcançar uma granulosidade mais fina na atribuição dos recursos do servidor. É definida também uma linguagem de especificação de restrições de QoS (WebQosL), que permite alocar porções fixas dos recursos do servidor a determinados tipos de aplicação. Assim, se não mais de 20% dos servidores devem ser alocados para jogos de computador, por exemplo, então não mais que 20% dos canais serão associados com requisições relativas a jogos.

Cardellini *et al.* (2001b) introduzem o conceito de Qualidade de Serviços Web (*Quality of Web Services* — QoWS), o qual consiste na aplicação de princípios conhecidos de QoS, utilizados em nível de rede, na construção de novas arquiteturas para servidores web. A arquitetura proposta consiste em um *cluster* de servidores web, formado por um conjunto de nós localmente distribuídos. Todas as requisições que chegam até o *cluster* têm que passar primeiramente por um *Web Switch*, o qual funciona como um despachante global para os nós do *cluster*, além de representá-los perante seus clientes. Equipamentos desse tipo são comumente encontrados em redes, sendo denominados de comutadores da camada 7 (*layer-7 switches*), por trabalharem na camada de aplicação do modelo OSI. Eles estabelecem uma conexão com os clientes, recebem as requisições HTTP e analisam seu conteúdo a fim de decidir qual nó atenderá a requisição. Os autores integram características de QoWS ao comutador, particularmente os aspectos de classificação, isolamento de desempenho, utilização de recursos em caso de sobrecarga do sistema e controle de admissão de requisições. O algoritmo proposto no artigo consiste em um mecanismo de particionamento dinâmico do *cluster* de servidores web, a fim de melhor acomodar as variações na carga de trabalho.

Zhu *et al.* (2001) propõem uma arquitetura em camadas, semelhante ao trabalho de Cardellini *et al.*, em que também há um despachante responsável por dividir a carga de trabalho entre as partições do *cluster* de servidores web. É implementado um mecanismo de escalonamento dinâmico que se adapta às variações na demanda de serviços e reparticiona automaticamente o *cluster* quando necessário. A alocação dos recursos é modelada como um problema de otimização.

Escalonamento Baseado em Prioridades

Nesta seção, serão discutidos alguns trabalhos que utilizam prioridades a fim de fornecer serviços diferenciados em servidores web.

Eggert & Heidemann (1999) propõem mecanismos tão somente do lado servidor para a provisão de serviços diferenciados na Web e os justificam pela sua facilidade de implantação na Internet atual. Os autores dividem as requisições HTTP em duas classes e mostram, através de experimentos, que mecanismos aparentemente simples, como a limitação do número de processos no servidor e a atribuição de prioridades aos mesmos, podem fornecer resultados significativos. O trabalho de Almeida *et al.* (1998) faz uso de escalonamento baseado em prioridades, porém demanda a alteração do *kernel* do sistema operacional. Seus resultados, obtidos através de experimentação, são bastante promissores, tendo sido implementadas duas políticas de escalonamento.

Chen & Mohapatra (1999) propõem um modelo para um servidor de Internet com diferenciação de serviços e o validam através de simulação dirigida por *traces*, utilizando *logs* de acesso a servidores web, a exemplo do presente trabalho. Os autores também empregam escalonamento baseado em prioridades e analisam o controle de admissão de requisições em caso de sobrecarga do servidor. Entretanto, o *log* usado como entrada para a simulação está defasado, datando de 1995 e pode não ser representativo da carga de trabalho na Web atual, o que não invalida as idéias apresentadas.

O trabalho de Bhatti & Friedrich (1999) também detalha uma arquitetura de servidor web para o fornecimento de serviços diferenciados, destacando a importância da adoção de técnicas de suporte a QoS em nível de aplicação. Uma das políticas de escalonamento proposta faz uso de prioridades para a diferenciação de serviços.

Rao & Ramamurthy (2001) apresentam um módulo de programa que implementa dois mecanismos em nível de aplicação do lado servidor. O módulo *DiffServer* intercepta as requisições vindas dos clientes, classifica-as segundo certos parâmetros e as repassa ao servidor Apache para processamento, ordenadas segundo suas prioridades. Os resultados demonstram que o tempo médio de espera para as requisições de alta prioridade diminui consideravelmente quando comparado com uma abordagem do tipo FIFO.

Na literatura sobre o assunto, encontram-se diversas implementações de algoritmos de diferenciação de serviços. Foram discutidos aqui apenas alguns exemplos, com ênfase naqueles relacionados às abordagens empregadas nesta tese, isto é, o particionamento dos recursos do servidor e a utilização de prioridades nas filas.

4.7 Cenários de Utilização

A necessidade de fornecimento de serviços diferenciados em servidores web, como comentado anteriormente, advém da constatação de que o modelo de serviço de melhor esforço, utilizado atualmente na Web, tornou-se inadequado e restritivo (Dovrolis & Ramanathan, 1999). Cada vez mais, faz-se necessário reconhecer as diferenças entre os usuários, a fim de fornecer a cada um o serviço de que necessita e, por que não dizer, pelo

qual pode pagar. Nesta seção, são discutidos alguns possíveis cenários de uso de serviços diferenciados na Web, motivando sua aplicação.

Uma possibilidade seria em um *Cenário Acadêmico*, em que se podem encontrar diferentes tarefas, tais como: acesso a páginas estáticas, *download* de material didático, acesso a correio eletrônico via Web (*web mail*), ambientes de ensino a distância, etc. Com os serviços diferenciados, torna-se possível, por exemplo, dar prioridade às requisições de usuários que estejam lendo sua correspondência diária ou usando algum ambiente de ensino a distância, ambas tarefas importantes em um ambiente acadêmico. Outra possibilidade (não muito popular) seria considerar como prioritárias as requisições emitidas por alunos de pós-graduação e professores.

Outro cenário propício para o emprego de QoS diferenciada são os *Provedores de Acesso à Internet*. Nesse ambiente, tem-se um alto nível de concorrência entre os usuários, com destaque para o acesso a notícias e correio eletrônico via Web. Naturalmente, a abordagem, nesse caso, seria dividir os usuários em grupos, tais como pagantes e não-pagantes (os primeiros poderiam ainda ser classificados em *premium* e *standard*). A cada grupo, caberia um tratamento diferenciado, segundo um contrato de serviço previamente acertado entre as partes.

Lojas Virtuais também são um excelente meio para o emprego de serviços diferenciados. Caracterizam-se pelo uso intenso de páginas dinâmicas e imagens, alto nível de concorrência entre os usuários, uso de mecanismos de busca e também sessões seguras. Nesse tipo de ambiente, poderia ser dada preferência aos clientes em processo de conclusão de uma compra, em lugar daqueles que estejam simplesmente passeando pelo *site*. Note-se que esta é uma decisão polêmica, pois usuários ocasionais podem também se tornar clientes, portanto não é interessante negligenciar demasiadamente seu atendimento.

Outros exemplos não exaustivos de aplicação de serviços diferenciados são *sites* de empresas, de bancos, serviços de *download*, disponibilização de conteúdo mulmídia e outros. Em qualquer caso, deverá sempre haver um acompanhamento do serviço fornecido, não só para garantir que o mesmo esteja dentro dos padrões de QoS desejados, mas também para propiciar um atendimento justo às diversas classes de usuários.

4.8 Considerações Finais

Este capítulo apresentou uma arquitetura para um Servidor Web com Diferenciação de Serviços, o SWDS, cujo objetivo é atender seus clientes com uma qualidade diferenciada, fornecendo a cada classe de serviço um tratamento condizente com suas características de demanda.

O servidor SWDS é capaz de reconhecer os diferentes grupos de clientes e subdividi-los em classes, segundo critérios previamente estabelecidos. Em seguida, à luz da carga de

trabalho presente no sistema e das políticas de atendimento vigentes, o servidor decidirá pela aceitação ou rejeição dos novos clientes, conforme o caso. Finalmente, realiza a diferenciação de serviços por meio dos vários algoritmos disponíveis. A arquitetura proposta pode servir como base para a implantação de infra-estruturas de servidores web, voltadas ao atendimento diferenciado de clientes, ou como referência para o desenvolvimento de um programa de servidor web com características de diferenciação de serviços. O modelo proposto não impõe nenhuma organização específica, adaptando-se tanto a um sistema distribuído quanto a uma arquitetura mono ou multiprocessada.

A abordagem escolhida para a solução do modelo foi a simulação orientada a eventos, dirigida por *traces*. A parametrização do modelo baseou-se em valores de desempenho encontrados em dispositivos de hardware reais, assim como em medições feitas em laboratório e nos resultados mais atuais encontrados em artigos da área. Para a geração da carga de trabalho, foram utilizados *logs* de acesso a servidores web, coletados durante a Copa do Mundo de 98.

Foi também apresentada uma visão geral do módulo de controle de admissão e dos mecanismos de diferenciação de serviços, discutindo-se alguns trabalhos relacionados. Finalmente, foram comentados alguns cenários de aplicação de serviços diferenciados.

Os capítulos seguintes discutem em detalhes a implementação e os resultados dos mecanismos de diferenciação de serviços, bem como do módulo de controle de admissão do servidor SWDS.

Capítulo

5

Mecanismo de Reserva de Recursos

5.1 Introdução

O Capítulo 4 apresentou o modelo do servidor SWDS, no qual se destacam os seguintes módulos: o Classificador, o Controle de Admissão e o *cluster* de processos servidores. A diferenciação de serviços é realizada através de algoritmos específicos, os quais são discutidos neste e no próximo capítulo.

Os algoritmos implementados neste trabalho possuem duas variantes: podem ser aplicados ainda na fase de despacho das requisições HTTP, determinando a forma de distribuição das mesmas entre os nós do *cluster*, ou agir diretamente sobre a disciplina de fila dos nós, a fim de alterar a ordem de processamento das requisições. No primeiro caso, o módulo de Controle de Admissão assume também o papel de escalonador e atribui as requisições a grupos de processos servidores segundo a classe a que pertencem. Realiza-se, portanto, uma forma de Enfileiramento Baseado em Classes (*Class-Based Queueing* — CBQ), abordagem na qual é reservada parte da capacidade de processamento do servidor para cada classe de requisições.

Este capítulo descreve a implementação de dois algoritmos conhecidos de balanceamento de carga no modelo do SWDS, considerando um *cluster* homogêneo e heterogêneo. Adicionalmente, um algoritmo de diferenciação de serviços, elaborado segundo os princípios do CBQ, também é proposto e implementado no modelo. São analisadas sua aplicabilidade, limitações e possíveis aperfeiçoamentos.

5.2 Modelo Simplificado do Servidor SWDS

Para a verificação dos algoritmos apresentados neste capítulo, foi adotado um modelo simplificado do servidor SWDS, conforme apresentado na Figura 5.1 (Teixeira *et al.*, 2003b). Este modelo consiste de um Classificador, um módulo de Escalonamento e um *cluster* de servidores web. O módulo escalonador é responsável por fazer a distribuição das requisições entre os nós do *cluster*, assumindo as funções de um despachante. Pode-se considerar cada nó como um servidor web convencional, com uma fila de espera independente, além de CPU, disco e interface de rede.

Para a validação do modelo, foi utilizada uma simulação dirigida por *traces*, sendo a carga de trabalho gerada a partir de *logs* de acesso ao *site* da Copa do Mundo 98 (Seção 4.3.3). A parametrização do modelo segue o exposto na Seção 4.3.1. Nos experimentos, optou-se por não utilizar o controle de admissão das requisições HTTP em caso de sobrecarga do sistema, a fim de não interferir no comportamento dos algoritmos. Dessa forma, consideram-se as filas dos servidores como ilimitadas. Assume-se que todos os nós têm acesso aos mesmos documentos. Os resultados apresentados neste capítulo correspondem à média de cinco simulações com um intervalo de confiança de 95%.

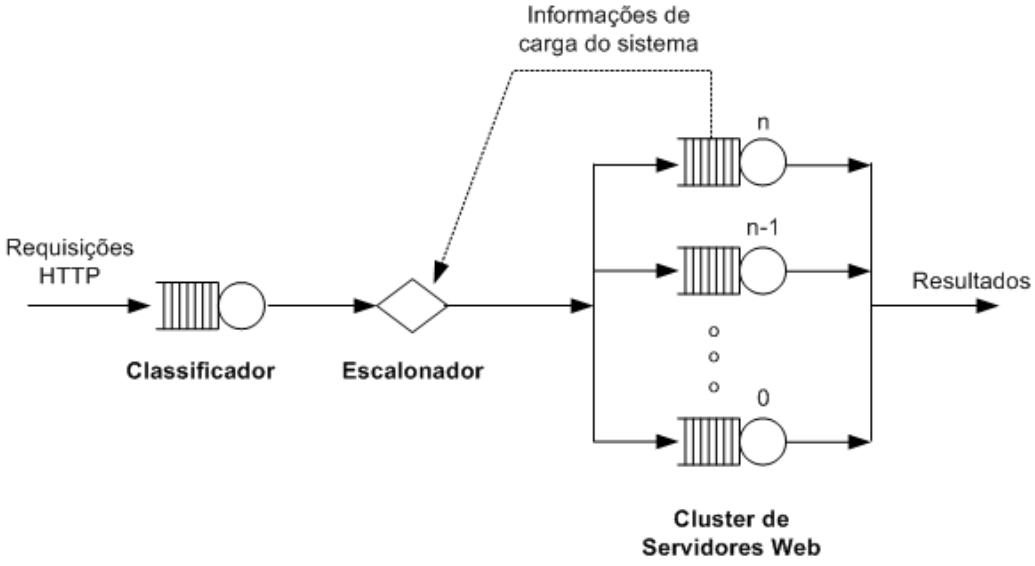


Figura 5.1: Modelo simplificado do servidor SWDS

5.3 Algoritmos de Balanceamento de Carga

Inicialmente, analisou-se o emprego de dois algoritmos de balanceamento de carga conhecidos, *Round Robin* (RR) e *Shortest Queue First* (SQF), na alocação das tarefas ou requisições aos nós do *cluster*. Embora o balanceamento de carga não seja o objetivo

principal deste trabalho, procurou-se, com a implementação desses algoritmos, cumprir as seguintes etapas: adquirir uma maior familiaridade com o modelo desenvolvido para o servidor; sistematizar os testes a serem realizados; definir um esqueleto básico para o programa de simulação; validar a parametrização do modelo e testar a geração da carga de trabalho. Além disso, os experimentos permitiram determinar o ponto de saturação da configuração utilizada para o *cluster*, tanto em relação à taxa de chegada de requisições quanto à utilização do sistema.

O algoritmo RR faz a atribuição das requisições aos nós do *cluster* segundo um esquema de rodízio, um após o outro. O algoritmo SQF, por sua vez, direciona a requisição para o nó que possuir o menor número de processos em fila, procurando, assim, atingir um melhor balanceamento de carga. Este último algoritmo necessita de informações precisas sobre a carga do sistema a cada momento, o que introduz uma sobrecarga maior no escalonamento.

Como métrica de desempenho, foi utilizado o tempo de resposta necessário para completar cada requisição, que é tomado entre o instante em que a requisição é aceita pelo sistema e o momento em que a mesma é completada, ou seja, enviada de volta ao cliente. O tempo de resposta compreende o tempo de serviço propriamente dito e o tempo de espera em fila. No caso de servidores muito sobrecarregados, o tempo de fila tende a aumentar rapidamente, constituindo-se na parcela dominante no cálculo do tempo de resposta total experimentado pelos clientes.

Nos experimentos relatados a seguir, foi simulado um *cluster* com quatro servidores web, numerados de 0 a 3, conforme a Seção 5.2. O objetivo é verificar o comportamento dos algoritmos RR e SQF no balanceamento de carga, para um *cluster* homogêneo e heterogêneo.

5.3.1 Cluster Homogêneo

O primeiro caso estudado foi o de um *cluster* homogêneo, aquele em que todos os nós exibem a mesma configuração de hardware e software. Nesta situação, não foi notada uma diferença de desempenho significativa entre os algoritmos RR e SQF. A Figura 5.2 mostra a variação do tempo médio de resposta das tarefas em função da taxa de chegada de requisições ao sistema, quando se utiliza o algoritmo *Round Robin*. Observa-se que, para taxas menores que cerca de 430 requisições por segundo, o tempo de resposta situa-se em um patamar inferior a 1 segundo, enquanto que, para taxas superiores a 450 requisições por segundo o tempo de resposta começa a subir rapidamente, chegando a mais de cinco vezes seu valor original a 490 requisições por segundo. O aumento no tempo de resposta deve-se ao crescimento do número de processos esperando nas filas dos nós do *cluster*, já que as medições feitas mostram que o tempo de serviço médio permanece baixo.

O comportamento do algoritmo RR também foi observado sob outro prisma,

considerando-se a variação do tempo de resposta em função da utilização do sistema (Figura 5.3). Pode-se notar que o *cluster* começa a saturar-se com cerca de 80% de utilização e, a partir desse ponto, o tempo de resposta sobe rapidamente, à medida que a utilização se aproxima de 100%.

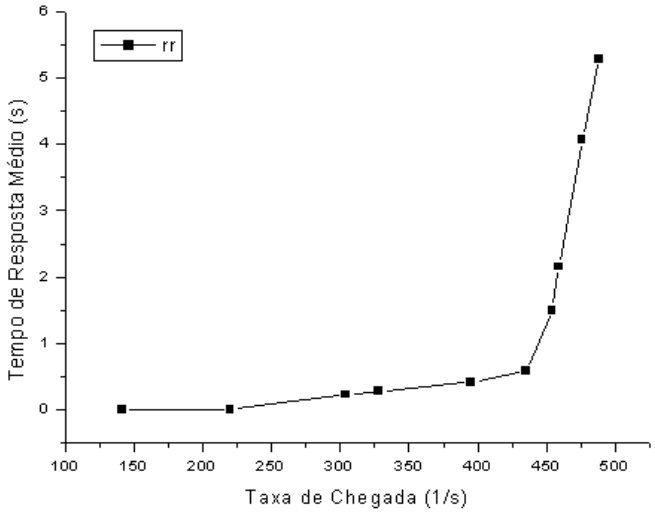


Figura 5.2: Tempo de resposta vs. Taxa de chegada (RR)

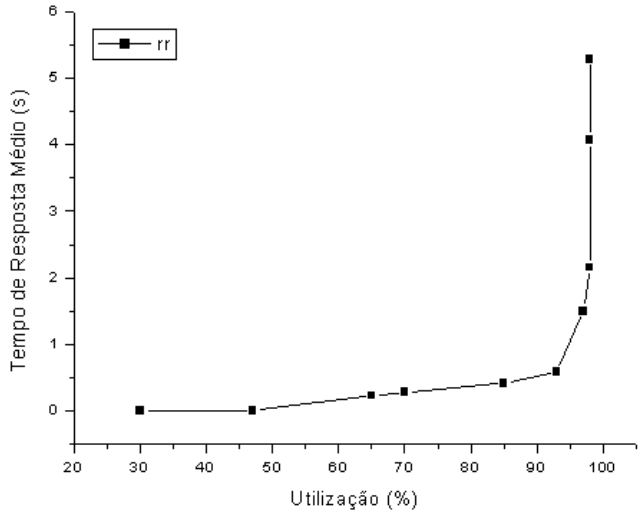


Figura 5.3: Tempo de resposta vs. Utilização (RR)

5.3.2 Cluster Heterogêneo

Em seguida, foi estudada a aplicação dos dois esquemas de balanceamento de carga acima em um *cluster* heterogêneo, a fim de verificar-se a eficácia do algoritmo SQF. O *cluster* considerado é formado por quatro servidores web, sendo que um deles apresenta um disco dez vezes mais lento que o dos demais, o que, na ausência de um bom balanceamento de carga, pode torná-lo um gargalo do sistema.

A análise do comportamento do tempo de resposta quando se utiliza o algoritmo RR para a atribuição das tarefas (Figura 5.4) mostra que, para uma carga de trabalho reduzida (taxa de chegada inferior a 400 requisições por segundo), o algoritmo RR comporta-se de maneira satisfatória. Contudo, à medida que o sistema atinge a saturação, as tarefas designadas para o servidor mais lento (nó 3) passam a apresentar tempos de resposta cada vez maiores, comparativamente às tarefas atribuídas aos servidores mais rápidos, cujos tempos de resposta comportam-se de modo semelhante ao observado no caso homogêneo (Seção 5.3.1). O algoritmo RR, portanto, não produz um bom balanceamento de carga em servidores web distribuídos heterogêneos, visto que lhe faltam informações sobre a carga presente em cada nó, a cada instante.

Em seguida, simulou-se novamente a mesma configuração do *cluster* utilizando o algoritmo SQF para a atribuição das tarefas, o qual designa as tarefas ao servidor que possui a menor fila, distribuindo a carga de maneira mais homogênea. A Figura 5.5 mostra os resultados obtidos e observa-se que os tempos de resposta apresentam comportamento praticamente igual, tanto para as requisições atribuídas ao servidor mais lento quanto para aquelas atribuídas aos demais servidores, mesmo a taxas de chegada bem elevadas, o que comprova o bom balanceamento de carga conseguido. Isso ocorre porque o algoritmo SQF tende a manter aproximadamente iguais os tamanhos das filas em cada servidor, homogeneizando os tempos de resposta médios. Vale ressaltar que, para requisições com tamanhos muito divergentes, este algoritmo não se revelaria tão eficiente, o que não é o caso da carga de trabalho aqui utilizada.

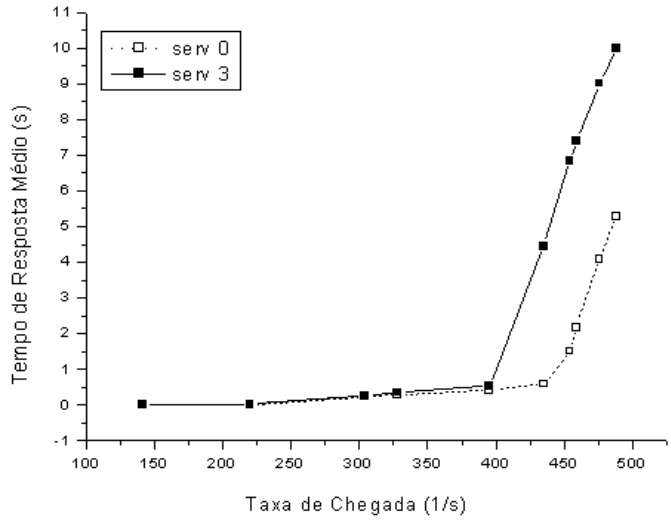


Figura 5.4: Tempo de resposta utilizando um *cluster* heterogêneo (RR)

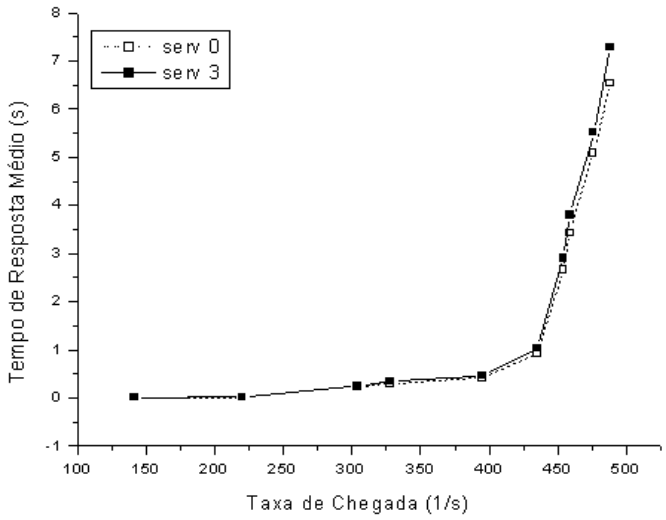


Figura 5.5: Tempo de resposta utilizando um *cluster* heterogêneo (SQF)

5.4 Algoritmo de Diferenciação de Serviços

Nesta segunda fase, procurou-se verificar a adequação da arquitetura proposta para a diferenciação de serviços. Para tanto, foi desenvolvido um algoritmo que implementa uma forma de CBQ no modelo, visando garantir a cada classe de serviço uma parcela pré-determinada da capacidade de processamento total do servidor. Esquema semelhante a esse é encontrado no nível da camada de rede, em roteadores e se caracteriza pela alocação estática das filas de saída às diferentes classes de tráfego (Kilkki, 1999). A seção a seguir discute algumas particularidades desse mecanismo de alocação de recursos.

5.4.1 Conceitos

Ferguson & Huston (1998) afirmam que o principal objetivo do enfileiramento baseado em classes é impedir que seja negado o acesso aos recursos do sistema a qualquer classe de serviço. Dessa forma, evita-se que as requisições de uma determinada classe sofram *starvation*, embora seja difícil prever a qualidade do serviço fornecido a cada grupo individualmente, já que esta será dependente da carga presente em cada classe e da sua importância relativa. As requisições são divididas em grupos ou agregados razoavelmente grandes, por exemplo, segundo critérios de requisitos das aplicações, preço pago pelo cliente e organização à qual ele pertence. A parcela de recursos destinada a cada classe pode ser estática, como no caso do algoritmo RSV, ou dinâmica, refletindo a mudança no perfil da carga de trabalho experimentada pelo sistema.

Quando a divisão em classes se dá segundo as características de cada aplicação, isso implica que cada classe poderá ser compartilhada por diferentes grupos de usuários, os quais têm em comum o tipo de aplicação que estejam utilizando. Se uma determinada

classe começa a exigir uma quantidade cada vez maior de largura de banda (ou de capacidade de processamento, no caso de um servidor web), devido à alta atividade de seus usuários, então este agregado poderá sofrer uma limitação nos recursos que lhe foram alocados. Dessa forma, nenhum grupo de usuários poderá utilizar toda a capacidade do sistema, mesmo que isso seja necessário. Uma conseqüência dessa abordagem é que a importância de uma tarefa individual será função do número de tarefas presentes na sua classe. Quanto maior o número de solicitações existentes, menor a importância relativa de cada uma delas.

Em um sistema desse tipo, a qualidade do serviço fornecido por cada classe depende da importância dessa classe em relação às demais e da carga gerada por seus componentes. É muito difícil, portanto, fazer uma previsão exata sobre o atendimento dispensado aos usuários, algo como "o tempo de resposta máximo para os clientes da Classe 2 nunca será superior a 1 segundo". Particularmente, usuários mal-comportados podem vir a monopolizar os recursos dentro da classe à qual foram alocados (Kilkki, 1999). A implementação de um controle de admissão permite gerenciar a aceitação de novos usuários pelo sistema, a fim de evitar que o mesmo fique sobrecarregado e possibilitar um maior controle sobre a qualidade do serviço em cada classe.

Certamente, um fator de controle do número de indivíduos interessados em um dado nível de serviço será sempre o preço. Um serviço de maior qualidade implica em um preço maior a ser pago. Como, em geral, poucos estão dispostos ou podem pagar por um luxo muitas vezes desnecessário, este talvez seja o maior limitante no número de clientes privilegiados presentes em um sistema. Uma abordagem simples, portanto, é adotar diferentes preços para diferentes classes de serviço, como sugerido em Odlyzko (1999) e aguardar que ocorra a seleção natural dos usuários.

5.4.2 Descrição do Algoritmo

O algoritmo proposto, denominado de *Reserva de Recursos* (RSV), subdivide o *cluster* de servidores em partições e associa cada partição a uma classe ou conjunto de classes de serviço, de maneira estática. Dessa forma, reserva-se efetivamente uma parcela da capacidade de processamento do servidor SWDS para cada categoria de requisições, garantindo-lhes o atendimento.

Um algoritmo desse tipo possui várias aplicações. Com um simples redirecionamento das requisições para os servidores apropriados, é possível, por exemplo, dar um tratamento preferencial às requisições feitas por usuários *premium* de um provedor de Internet ou, então, privilegiar usuários que estejam em fase de conclusão de uma compra em uma loja virtual. A diferenciação de serviços também pode ser feita levando-se em conta outros fatores, tais como: origem das requisições (tomando-se o endereço IP como referência), estimativa de consumo de recursos por uma tarefa (supondo que se tenha um histórico de

uso dos clientes), tipo de objeto solicitado (arquivos HTML, imagens, áudio, vídeo, requisições dinâmicas, etc.), requisitos de qualidade de serviço, mapeamento sobre protocolos de diferenciação de serviços em nível de rede e outros.

Nos experimentos relatados a seguir, considera-se um *cluster* formado por quatro servidores web, numerados de 0 a 3, conforme descrito na Seção 5.2. As requisições que chegam são divididas em alta e baixa prioridade pelo módulo Classificador da arquitetura e o escalonamento é feito segundo o algoritmo RSV. Neste estudo, o servidor de mais alta ordem do *cluster* (nó 3) é reservado somente para as requisições de alta prioridade, enquanto que as requisições de baixa prioridade são direcionadas aos outros servidores da arquitetura segundo um esquema de rodízio (*round robin*). Uma vez atribuída a um servidor, a tarefa tem que permanecer nele até que seja atendida pelo sistema. A métrica de desempenho utilizada é o tempo de resposta médio para o atendimento das requisições.

5.4.3 Resultados Experimentais

No primeiro experimento realizado, 80% das requisições pertencem à classe de baixa prioridade e 20% delas, à de alta prioridade. A análise dos resultados obtidos (Figura 5.6) mostra que, para cargas não muito elevadas, as tarefas de ambas as classes exibem tempos de resposta aproximadamente iguais, porém, para uma taxa de chegada superior a 400 requisições por segundo, o tempo de resposta das requisições de alta prioridade mantém-se praticamente estável (curva c), enquanto que o das de baixa prioridade começa a subir rapidamente (a). Isso ocorre devido à saturação dos servidores alocados à classe de menor prioridade, o que provoca o surgimento de longas filas de espera.

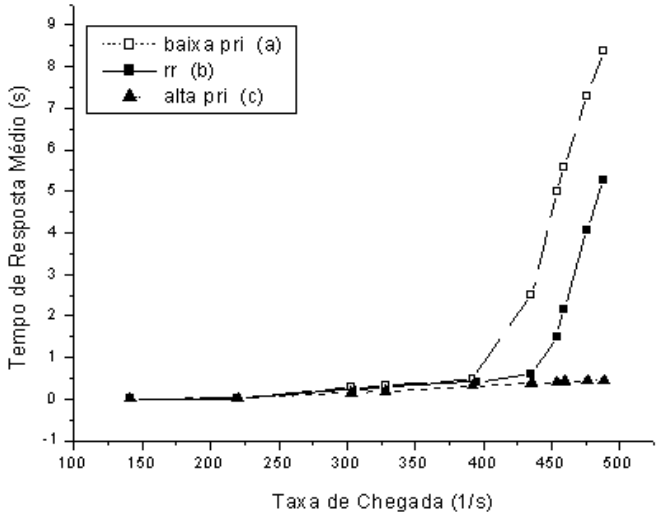


Figura 5.6: Tempo de resposta usando o algoritmo RSV (20% de requisições de alta prioridade)

Para efeito de comparação, foi também traçada no gráfico a curva do tempo de resposta obtida com o algoritmo RR (Seção 5.3.1), cujo escalonamento não leva em conta a

diferenciação por classes de serviço. Comparando-se as curvas, observa-se que a reserva de um servidor para as requisições de alta prioridade faz com que estas realmente tenham um atendimento preferencial (c), sem uma degradação visível no seu desempenho, mesmo a taxas de chegada elevadas. Já no caso sem diferenciação de serviços (b), a queda no desempenho ocorre bem mais cedo, pois todas as requisições recebem um tratamento uniforme, sem distinção de classe. Essa é precisamente a forma de operação de um servidor web convencional, o que não é desejável em algumas situações. As requisições de baixa prioridade, por outro lado, sofrem uma acentuada degradação no seu desempenho, como mostra a inclinação abrupta da curva do seu tempo de resposta (a).

O algoritmo RSV também foi testado com outras configurações de requisições de alta e baixa prioridade. O próximo caso analisado foi aquele em que 75% das requisições pertencem à classe de baixa prioridade e 25%, à de alta prioridade. A Figura 5.7 mostra que o comportamento do tempo de resposta das requisições de ambas as classes é virtualmente igual. A razão é fácil de ser entendida: como existem 4 servidores no *cluster*, cada um receberia aproximadamente 25% da carga total se não fosse usada a diferenciação de serviços. Na divisão em classes escolhida para o experimento, 25% das requisições são de alta prioridade e vão para um servidor específico e as outras 75% são distribuídas entre os três servidores restantes (25% para cada um). Nada mudou em relação ao caso sem diferenciação de serviços, portanto esta perde sua finalidade, devido à divisão das classes e à configuração adotada para o *cluster*. Ressalte-se que esta é uma limitação do algoritmo de reserva de recursos implementado e não do modelo do servidor SWDS, como será analisado mais adiante.

Para finalizar, estudou-se ainda mais um caso, em que 30% das requisições são de alta prioridade. Desta vez, a situação se inverte em relação ao primeiro caso: como se pode ver na Figura 5.8, agora são as requisições da classe de alta prioridade que experimentam uma acentuada degradação no seu desempenho, enquanto que as requisições de baixa prioridade são privilegiadas, exatamente o contrário do que se pretendia inicialmente. Tal fato ocorre porque o algoritmo RSV, ao reservar um único servidor para as requisições da classe prioritária, está, na verdade, sobrecarregando-o. Observe que, no caso sem diferenciação de serviços, esse servidor receberia apenas 25% da carga total e, aqui, ele recebe 30%, um acréscimo de 5 pontos percentuais que é suficiente para saturá-lo. Novamente, esta é uma falha na alocação de recursos feita pelo algoritmo RSV, como comentado na seção a seguir.

5.4.4 Considerações sobre o Algoritmo

O algoritmo RSV, como foi visto, consegue fornecer uma boa diferenciação de serviços sob certas condições, porém não pretende ser uma solução definitiva para esta questão. Seu objetivo era a validação inicial do modelo proposto para o servidor SWDS, bem como

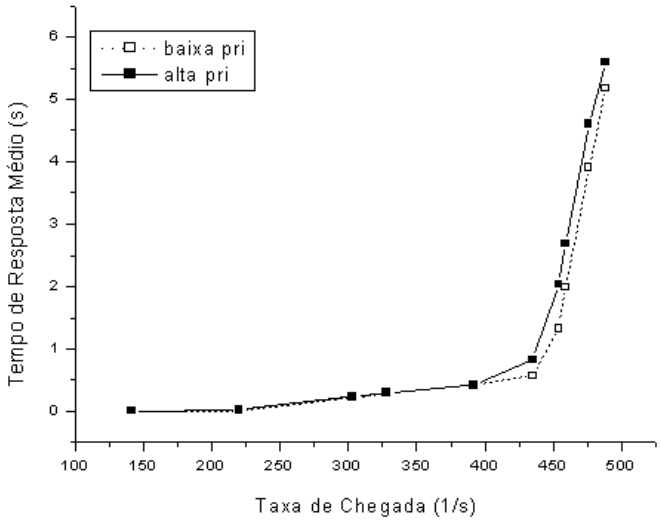


Figura 5.7: Tempo de resposta usando o algoritmo RSV (25% de requisições de alta prioridade)

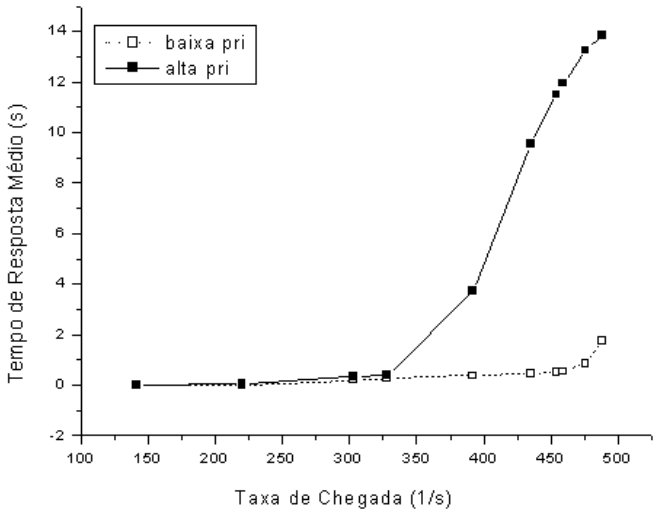


Figura 5.8: Tempo de resposta usando o algoritmo RSV (30% de requisições de alta prioridade)

apontar possibilidades de pesquisa na área.

O particionamento estático do *cluster* de servidores web, com a reserva de subconjuntos de servidores para certas classes de serviço, aplica-se melhor a casos em que o padrão da carga de trabalho é razoavelmente conhecido e previsível, quando se tem uma população de usuários controlada (por exemplo, uma *intranet* ou um site restrito a uma comunidade específica). Contudo, em ambientes absolutamente livres como a Web, este esquema apresenta algumas desvantagens, dentre elas o fato de que variações na carga de trabalho, mesmo que pequenas, podem tornar inapropriada a alocação de recursos feita inicialmente, produzindo resultados contrários ao esperado (Teixeira *et al.*, 2003a).

O particionamento estático não consegue se adaptar a taxas de chegada variáveis e

também não considera a carga presente em cada nó ao fazer o escalonamento das requisições. Isso pode levar ao desperdício de recursos, uma vez que uma determinada partição pode estar subutilizada enquanto outra está sobrecarregada (Cardellini *et al.*, 2001b; Zhu *et al.*, 2001). No esquema em funcionamento atualmente no SWDS, também não há como migrar recursos de uma partição para outra, mesmo que estes estejam ociosos, o que é uma característica de algoritmos do tipo CBQ com alocação estática.

Dessa forma, a implementação de um algoritmo que realize o particionamento dinâmico do *cluster* torna-se atrativa. Tal algoritmo deverá inicialmente fazer um particionamento arbitrário e ajustá-lo periodicamente, adaptando-se às características da carga de trabalho e às condições de utilização do SWDS, obtidas a partir do módulo de Controle de Admissão. Para tanto, será necessário conhecer a carga presente no sistema em um dado instante e ser capaz de prever seu comportamento futuro, reagindo rapidamente a mudanças. Uma estratégia de adaptação como essa implica, em geral, na necessidade de migração de requisições de um servidor para outro. Essas questões, como se pode ver, não são triviais, ainda mais no caso da Web, em que as características da carga de trabalho podem variar rapidamente e de modo imprevisível.

Em resumo, o emprego da abordagem de reserva de recursos para a diferenciação de serviços em servidores web distribuídos, embora adequado em alguns casos, pode também se revelar ineficiente, como demonstrado pelos experimentos descritos neste capítulo. Os resultados obtidos motivam a investigação adicional da área a fim de dotar o algoritmo RSV de características de adaptabilidade.

5.5 Considerações Finais

Este capítulo tratou sobre a validação inicial do modelo proposto para o servidor SWDS, apresentando seus primeiros resultados. Foram implementados dois algoritmos de balanceamento de carga conhecidos no modelo (RR e SQF) e analisado seu comportamento sob diferentes configurações. Os resultados obtidos ratificam que o algoritmo SQF é mais eficiente que RR na alocação de recursos em um *cluster* heterogêneo.

Além disso, implementou-se um algoritmo de reserva de recursos (RSV) a fim de avaliar a aplicação da arquitetura proposta na diferenciação de serviços. Os experimentos realizados comprovam que a mesma é efetivamente capaz de fornecer uma QoS diferenciada aos seus clientes, conseguindo bons resultados mesmo a cargas elevadas. O algoritmo RSV, contudo, mostrou-se não muito flexível, tendo sido apontados caminhos para aperfeiçoamentos futuros, dentre eles a alocação dinâmica do *cluster* de servidores web às classes de serviço.

O capítulo seguinte discute a introdução de prioridades nas filas dos servidores a fim de permitir a implementação de outros algoritmos de diferenciação de serviços, inclusive

alguns já empregados em roteadores, em nível de rede. Assim, será possível utilizar esquemas mais refinados de alocação de recursos na arquitetura do SWDS.

Capítulo

6

Escalonamento Baseado em Prioridades

6.1 Introdução

O capítulo anterior estudou o uso de esquemas do tipo *Class-Based Queueing* para a diferenciação de serviços, mostrando algumas de suas limitações, principalmente quando se faz a alocação estática dos servidores (ou processos) às classes de usuários, o que pode produzir resultados contrários ao esperado. A pouca flexibilidade daquela solução, aliada à necessidade de se ter um conhecimento mais detalhado da carga que será imposta ao sistema, podem gerar alguns problemas em um ambiente com as características da Web, em que o comportamento da carga de trabalho pode variar rapidamente e de modo imprevisível (Floyd & Paxson, 2001).

Neste capítulo, são apresentados os resultados obtidos com a aplicação de um Escalonamento Baseado em Prioridades na arquitetura do Servidor Web com Diferenciação de Serviços (SWDS), utilizando-se como carga de trabalho os *traces* de acesso a servidores web já discutidos. Em particular, destacam-se as vantagens do uso de prioridades visando alcançar a diferenciação de serviços, bem como as limitações desse mecanismo. Relatam-se também os resultados obtidos com a introdução de um mecanismo de prioridades adaptativo na arquitetura, destacando-se sua adequação à natureza instável do tráfego na Web.

6.2 Metodologia de Teste

Para a validação dos mecanismos de diferenciação de serviços aqui propostos, recorreu-se à simulação. É implementado um modelo do servidor SWDS conforme descrito na Seção 4.2, com quatro servidores web homogêneos formando um *cluster*, os quais obedecem à parametrização da Seção 4.3.1. Cada nó do *cluster* pode ser interpretado como um servidor web convencional, composto de CPU, disco e interface de rede, com múltiplas filas de prioridade, a fim de acomodar as diferentes classes de serviço. Para a geração da carga de trabalho, utilizam-se os *logs* de acesso aos servidores web da Copa do Mundo 98 (Seção 4.3.3).

Nos experimentos, as requisições que chegam ao sistema são divididas em duas classes, de alta e baixa prioridade, pelo módulo Classificador da arquitetura, sendo atribuídas aos nós do *cluster* segundo um esquema de rodízio. A disciplina de fila de cada nó é determinada pelos algoritmos de diferenciação de serviços descritos nas Seções 6.3 e 6.4. As métricas de desempenho utilizadas são o tempo de resposta médio para o atendimento das requisições e o percentual de requisições completadas com sucesso, para ambas as classes de serviço consideradas. Os resultados apresentados foram obtidos a partir da média de cinco simulações com um intervalo de confiança de 95%.

Assim como no Capítulo 5, não foi utilizado o módulo de Controle de Admissão, a fim de não interferir no desempenho dos algoritmos, portanto as filas dos servidores são ilimitadas. Assume-se que todos os nós têm acesso aos mesmos documentos.

6.3 Mecanismo de Prioridades Rigoroso

6.3.1 Conceitos

Sistemas de filas baseados em prioridades são utilizados em diversas áreas da Computação, como sistemas operacionais e redes de computadores e caracterizam-se por tratar os indivíduos de modo diferenciado, dando tratamento preferencial a alguns em detrimento de outros (Allen, 1990). Esse tipo de comportamento é precisamente o que se deseja quando se busca a diferenciação de serviços, daí a razão de sua escolha para implementação no modelo do servidor SWDS.

O Mecanismo de Prioridades Rigoroso (*Strict Priority Queueing*) aqui utilizado exige que o atendimento das requisições siga uma disciplina de prioridades rígida, ou seja, somente serão atendidas requisições de prioridade inferior se não houver nenhuma requisição de prioridade superior aguardando em fila. Nos experimentos relatados neste capítulo, as classes de serviço em que se dividem os clientes do sistema (portanto, as requisições HTTP) são mapeadas nas diferentes classes ou níveis de prioridade oferecidos pelo servidor, nu-

meradas de 0 a n. Quanto maior o número da classe, maior a sua prioridade no sistema, ou seja, os clientes pertencentes a uma classe i receberão um tratamento preferencial em relação aos clientes da classe j, se $i > j$. Os clientes que possuírem a mesma prioridade serão atendidos, dentro de sua classe, segundo uma disciplina FCFS.

Teoricamente, existem dois esquemas de controle possíveis para o caso em que um cliente de prioridade superior chega a um centro de serviço e encontra um cliente de menor prioridade sendo atendido. Se o esquema adotado for do tipo *não-preemptivo*, o cliente que chega deve esperar até que o cliente em serviço termine de ser atendido, independentemente de suas respectivas prioridades. No esquema *preemptivo*, o cliente em atendimento tem seu serviço interrompido e o sistema passa a atender aquele que acabou de chegar. O cliente que foi interrompido retorna, então, ao início da fila de espera da classe à qual ele pertence. Mais tarde, ao ser novamente selecionado para atendimento, o serviço poderá ser retomado do ponto em que parou (*preemptive-resume*) ou recomeçar desde o início (*preemptive-repeat*).

Nos experimentos, assume-se que o atendimento é do tipo não-preemptivo e, ainda, que uma requisição atribuída a um processo servidor deve permanecer nele até a conclusão do serviço. A disciplina de fila é *work conserving*, isto é, um servidor nunca ficará ocioso enquanto houver clientes esperando para ser atendidos e um cliente não deixará o sistema sem ter sido atendido (Almeida *et al.*, 1998).

Os resultados aqui discutidos demonstram que o uso de prioridades permite realizar a diferenciação de serviços de uma maneira mais simples e flexível do que no esquema de Reserva de Recursos apresentado no Capítulo 5, porém é preciso ter cuidado na atribuição de prioridades aos clientes, a fim de evitar que uma determinada classe de serviço monopolize o uso dos recursos do sistema.

6.3.2 Resultados Experimentais

Inicialmente, considerou-se o caso em que 50% das requisições que chegam ao sistema pertencem à classe de alta prioridade. Para a geração de carga de trabalho, foram utilizados os sete servidores mais ativos do *log*, o que é aqui referido como *carga média*.

A Figura 6.1 mostra o comportamento do tempo de resposta médio em relação à utilização do sistema, para ambas as classes de serviço. A curva (b) corresponde ao padrão do tempo de resposta quando não se utiliza nenhum mecanismo de diferenciação de serviços, o caso de um servidor web convencional, como o Apache. As curvas (a) e (c) representam o comportamento das requisições de baixa e alta prioridade, respectivamente.

Quando não são utilizadas prioridades (b), o tempo de resposta sobe com o aumento da utilização do sistema, sofrendo uma acentuada degradação à medida que o sistema se aproxima da utilização completa. Por outro lado, quando a diferenciação de serviços é

empregada, as requisições de alta prioridade (c) apresentam um tempo de resposta estável mesmo em altos níveis de utilização, sem sinais de queda no seu desempenho. Além disso, o tratamento dado às requisições de baixa prioridade (a) permanece próximo daquele oferecido por um servidor web convencional, o que também é uma característica atrativa do mecanismo de prioridades rigoroso.

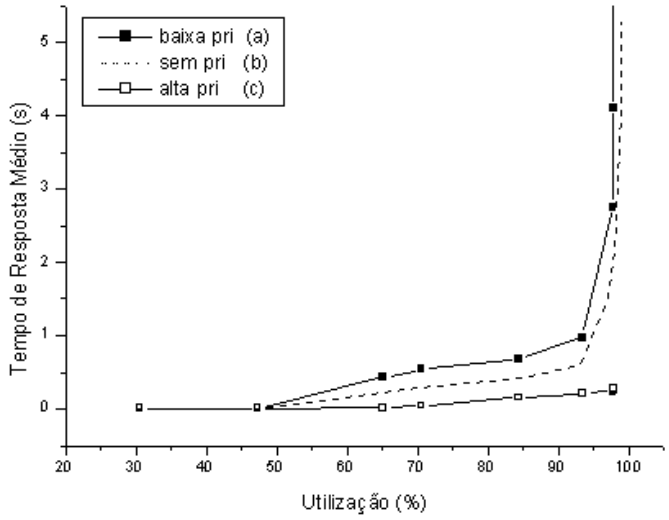


Figura 6.1: Tempo de resposta usando prioridades (utilização)

Analisou-se também o comportamento do tempo de resposta em relação à taxa de chegada (Figura 6.2). As requisições de alta prioridade apresentam um tempo médio de atendimento baixo, menor que 0,3 s, mesmo a taxas elevadas (Tabela 6.1). Quanto às tarefas de baixa prioridade, nota-se uma acentuada degradação no seu desempenho a partir de 435 requisições por segundo, conseqüência do tratamento preferencial dado à classe superior.

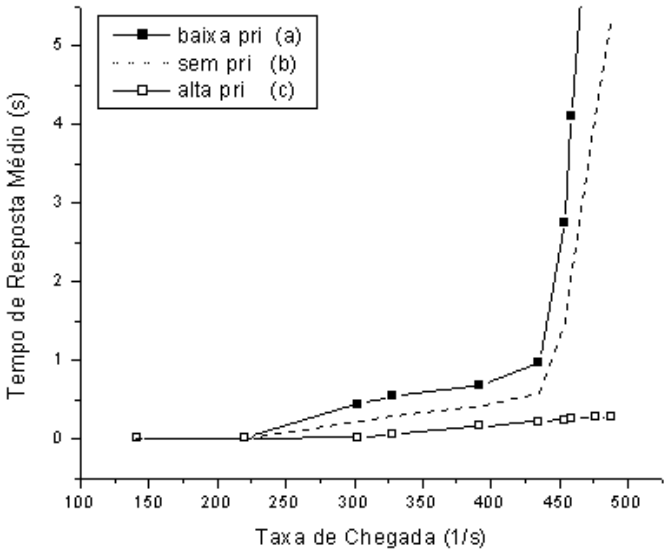


Figura 6.2: Tempo de resposta usando prioridades (taxa de chegada)

Taxa Cheg (1/s)	Baixa Prioridade		Alta Prioridade	
	Util (%)	Tempo Resp (s)	Util (%)	Tempo Resp (s)
141	30,5	0,01	30,5	0,01
220	47,2	0,01	47,2	0,01
303	65,1	0,44	65,1	0,02
328	70,5	0,54	70,5	0,05
392	84,3	0,68	84,3	0,16
435	93,5	0,97	93,5	0,22
454	97,7	2,75	97,7	0,24
459	97,8	4,10	97,8	0,25
476	97,9	7,96	97,9	0,27
488	97,9	10,44	97,9	0,28

Tabela 6.1: Tempo de resposta com 50% de requisições na classe de alta prioridade

Variação dos Percentuais de Clientes nas Classes de Serviço

Em seguida, foram realizados novos experimentos com diferentes distribuições dos clientes nas classes de serviço, a fim de verificar sua influência no tempo de atendimento das requisições. A Figura 6.3 mostra o comportamento do tempo de resposta das requisições de alta prioridade, em relação à utilização do sistema, para os casos em que se têm 20, 50, 70 e 90% dos clientes nesta classe. Observa-se que, à medida que se aumenta o percentual de clientes na classe de alta prioridade, a inclinação da curva do tempo de resposta ocorre em níveis cada vez mais baixos de utilização, o que denota a saturação do sistema. Dessa forma, há um limite razoável para o número de clientes que podem ser considerados de alta prioridade, sob pena de não ocorrer a diferenciação de serviços.

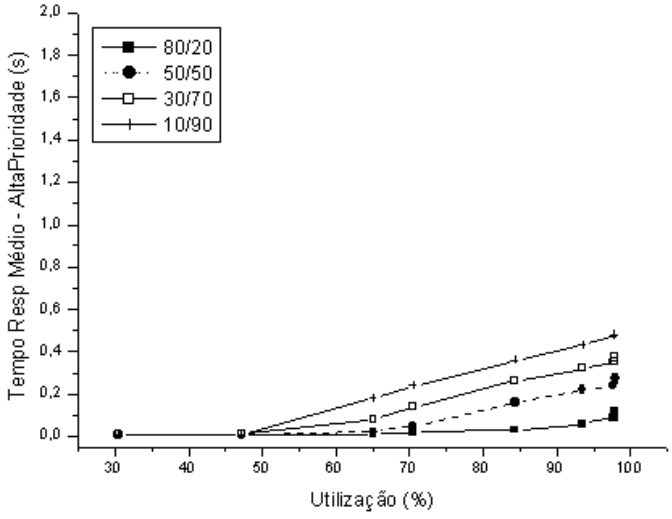


Figura 6.3: Variação dos percentuais de clientes (alta prioridade)

Na Figura 6.4, vê-se a mesma situação acima sob o ponto de vista das requisições de baixa prioridade, para os casos em se têm 80, 50, 30 e 10% dos clientes na classe de menor prioridade. Novamente, a inclinação da curva do tempo de resposta ocorre em níveis de

utilização progressivamente menores, à medida que mais clientes são alocados à classe de alta prioridade, os quais acabam por monopolizar os recursos do sistema. Nessa situação, as requisições de baixa prioridade têm pouca influência no desempenho global do servidor e recebem pouca ou nenhuma atenção do mesmo.

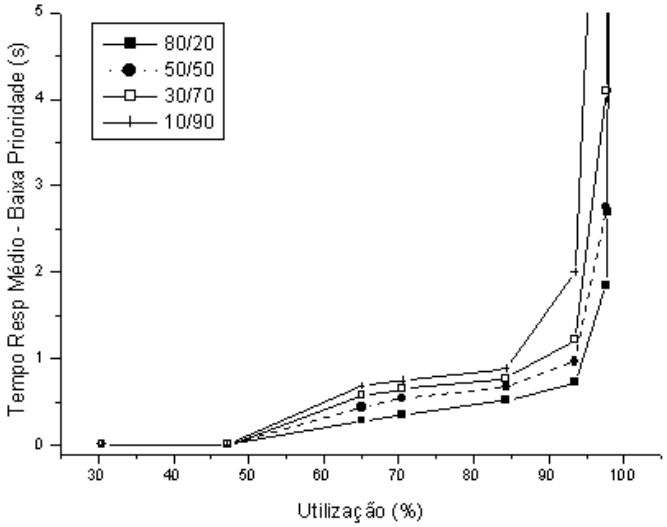


Figura 6.4: Variação dos percentuais de clientes (baixa prioridade)

Pode-se notar que, em todos os casos analisados, o tratamento recebido pelas requisições da classe de alta prioridade é nitidamente superior ao experimentado pelas requisições de baixa prioridade.

6.3.3 Avaliação de Desempenho

Como demonstrado pelos experimentos realizados, o escalonamento de prioridades rigoroso revela-se um excelente mecanismo para a provisão de serviços diferenciados em servidores web. Sua implementação é simples e lança mão de conceitos corriqueiros na área de sistemas operacionais e redes, como as filas de prioridades. Além disso, esse mecanismo permite alcançar, a cargas altas, praticamente 100% de utilização do sistema, pois um processo servidor nunca ficará ocioso enquanto houver requisições aguardando para ser atendidas. Também apresenta boa escalabilidade, sendo capaz de atender a diferentes classes de serviço com apenas alguns poucos ajustes no servidor SWDS, ao contrário do algoritmo de Reserva de Recursos (RSV) proposto no Capítulo 5.

Deve-se ter cuidado, entretanto, na atribuição das prioridades às requisições, uma vez que um excesso de clientes em uma determinada classe pode anular a diferenciação de serviços pretendida, além de provocar o não atendimento das requisições das classes de prioridade inferior, como demonstrado a seguir.

Baixa/Alta	Baixa Prioridade		Alta Prioridade	
(%)	Términos (%)	Tempo Resp (s)	Términos (%)	Tempo Resp (s)
90/10	92,58	5,87	100,00	0,07
80/20	91,64	6,61	100,00	0,12
70/30	90,46	7,52	100,00	0,17
60/40	88,84	8,77	100,00	0,23
50/50	86,66	10,44	100,00	0,28
40/60	83,49	12,91	100,00	0,33
30/70	78,24	17,05	99,99	0,38
20/80	67,62	25,94	99,99	0,44
10/90	38,41	53,64	99,99	0,68

Tabela 6.2: Percentual de requisições completadas (carga média)

Experimentos com Carga Média

Foram feitos experimentos visando analisar o percentual de requisições de uma certa classe que conseguem ser concluídas, em relação ao total de requisições que chegam ao sistema durante o período simulado (términos/chegadas). São consideradas duas classes de serviço e o percentual de clientes na classe de alta prioridade varia de 10 até 90%, com incremento de 10 pontos. Analisando-se os dados na Tabela 6.2, pode-se perceber que, à medida que mais requisições são alocadas na classe de alta prioridade, o tratamento dado às requisições de baixa prioridade piora consideravelmente, até o ponto em que pouco mais de um terço delas são concluídas. As requisições de alta prioridade, por sua vez, não apresentam alterações, perceptíveis pelos clientes, em seu desempenho. A Figura 6.5 mostra a variação do percentual de requisições que conseguem ser completadas em cada classe, nos diversos casos estudados.

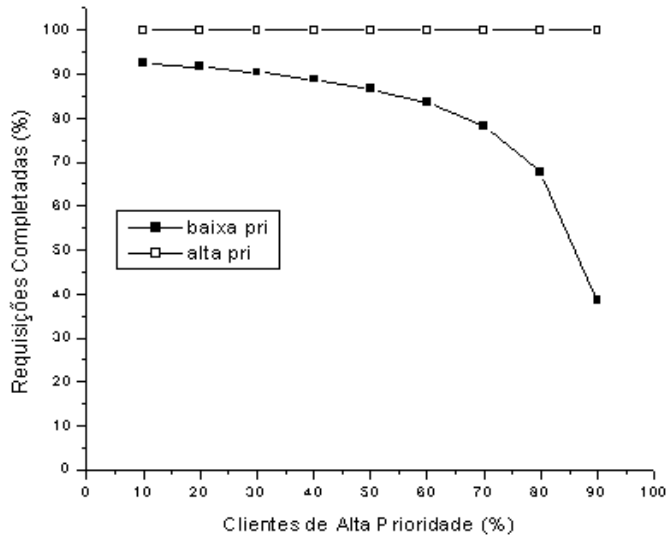


Figura 6.5: Requisições completadas (carga média)

É possível analisar a mesma situação sob o prisma do tempo de resposta médio das

Baixa/Alta	Baixa Prioridade		Alta Prioridade	
(%)	Términos (%)	Tempo Resp (s)	Términos (%)	Tempo Resp (s)
90/10	73,27	17,26	99,99	0,08
80/20	69,95	19,45	99,99	0,14
70/30	65,70	22,22	99,99	0,21
60/40	59,94	26,17	99,99	0,27
50/50	52,04	31,69	99,99	0,34
40/60	40,35	40,50	100,00	0,41
30/70	21,52	58,79	99,99	0,51
20/80	1,52	0,38	95,46	4,55
10/90	1,39	0,34	85,12	10,35

Tabela 6.3: Percentual de requisições completadas (carga total)

requisições. As requisições de alta prioridade mantêm seu tempo de resposta sempre abaixo de 1 s, embora o mesmo aumente quase uma ordem de grandeza entre o caso em que se têm 10% de clientes de alta prioridade no sistema e aquele em que há 90% desses clientes. O tempo de resposta da classe de baixa prioridade, por outro lado, sobe de maneira proibitiva, atingindo quase 1 minuto, o que pode levar os clientes não prioritários a desistirem de submeter suas solicitações ao servidor web, fato não desejável.

Experimentos com Carga Total

Foi feita uma nova série de experimentos, desta vez utilizando-se a *carga total* do *log*, ou seja, todos os servidores web. Neste caso, a situação de *starvation* das requisições da classe inferior se agrava ainda mais, como mostrado na Tabela 6.3. Nos casos extremos, em que há mais de 80% dos clientes na classe de alta prioridade, menos de 2% das requisições de baixa prioridade chegam a ser completadas, caracterizando uma situação de negação de serviço (Figura 6.6). Os clientes de alta prioridade, portanto, acabam por monopolizar os recursos do sistema e, como conseqüência, têm também o seu próprio desempenho prejudicado. No caso mencionado acima, algumas de suas requisições nem chegam a ser concluídas pelo servidor e os tempos de resposta sofrem uma acentuada elevação, contrariando a diferenciação de serviços. Novamente, esta é uma situação que deve ser evitada.

A análise do tempo de resposta médio também revela uma situação anômala: para o caso em que mais de 80% dos clientes são de alta prioridade, os clientes de menor prioridade aparentemente apresentam o melhor desempenho entre as duas classes, o que vai contra o objetivo inicial. Na verdade, seu tempo de resposta médio apresenta-se baixo porque as poucas requisições atendidas o foram bem no início da simulação. Os testes realizados mostram que, depois de um certo tempo, somente requisições de alta prioridade são servidas pelo sistema e as requisições não prioritárias aguardam em uma fila de espera interminável, sem nunca receberem uma parcela de processamento. Essa situação é resultado de uma alocação equivocada das requisições nas classes de serviço.

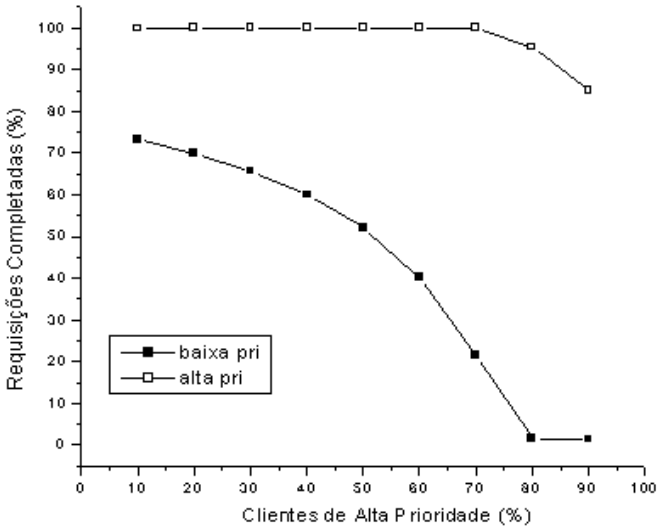


Figura 6.6: Requisições completadas (carga total)

Concluindo, quando se utilizam prioridades, ou qualquer outro mecanismo para realizar diferenciação de serviços, evidentemente pretende-se favorecer uma determinada classe de clientes, garantindo-lhes um atendimento melhor que o convencional, dentro de certos requisitos de QoS previamente acertados. Contudo, os experimentos realizados mostram que é preciso prestar atenção ao comportamento de todas as classes de requisições, uma vez que não é, em geral, desejável que as classes de menor prioridade sejam por demais penalizadas. Isso acabaria por afastar clientes que, embora aparentemente de "menor" importância, ainda assim são potenciais geradores de receitas para o negócio representado por um dado site na Web. Além disso, se a maioria dos clientes que chega a um sistema for classificada como de alta prioridade, na prática fica difícil conseguir alguma forma de diferenciação de serviços entre eles, qualquer que seja o mecanismo empregado.

6.4 Mecanismo de Prioridades Adaptativo

O mecanismo de prioridades rigoroso, relatado na Seção 6.3, é bastante eficiente ao fornecer diferenciação de serviços entre diferentes classes de requisições. Contudo, apresenta algumas deficiências, as quais podem ser minimizadas com o uso de um esquema mais flexível de priorização das requisições, como descrito nesta seção.

6.4.1 Descrição do Algoritmo

O mecanismo de prioridades adaptativo aqui proposto (PRIAdap) tem o objetivo de fazer uma sintonia fina do emprego das prioridades, relaxando ou intensificando a sua utilização conforme o caso. Dessa forma, pode-se atribuir uma maior ou menor importância às requisições de alta prioridade, a fim de evitar que estas venham a monopolizar

o uso dos recursos do sistema. O mecanismo adaptativo reconhece que as requisições de menor prioridade não podem esperar indefinidamente nas filas (como às vezes ocorre no mecanismo rigoroso), por isso introduz uma certa flexibilidade na escolha da próxima requisição a ser atendida.

No algoritmo da Figura 6.7, cada processo servidor é definido com uma fila de espera única, na qual as requisições são inseridas por ordem de chegada, independentemente de sua prioridade. É utilizado um parâmetro k, denominado *look-ahead*, que determina o número máximo de posições da fila de espera que serão percorridas a partir do início, à procura de requisições de uma determinada prioridade. Caso não encontre nenhuma requisição do tipo especificado, o algoritmo será repetido para o nível de prioridade imediatamente inferior e assim por diante. Em último caso, será escolhida a primeira requisição da fila. Quanto maior o valor de k, tanto melhor será o tratamento dispensado às requisições de maior prioridade. Para $k = 1$, as requisições serão atendidas segundo sua ordem de chegada, sem diferenciação (Teixeira *et al.*, 2004b).

```
int i = 0;
Lnode *cliente = início da fila;

Faça
    Enquanto ((cliente->prioridade != classe) && (i < look-ahead))
        cliente = próximo cliente da fila;
        i++;
    fim-enquanto;

    Se (não encontrou cliente da classe procurada) então
        classe--;
        i = 0;
        cliente = início da fila;
    fim-se;
enquanto ((classe > 0) && (cliente->prioridade != classe));

evento = *cliente;
```

Figura 6.7: Algoritmo de Prioridades Adaptativo (PRIAdap)

A implementação do mecanismo de prioridades adaptativo demandou modificações no código do simulador `SimPack`, pois o mesmo disponibiliza originalmente apenas o mecanismo de prioridades rigoroso. Como mostrado na Figura 4.2, todas as classes derivadas de VList devem implementar três métodos: *Insert()*, *Remove()* e *Display()*. Para o desenvolvimento do algoritmo PRIAdap, foi preciso alterar o método Insert() a fim de que ele permitisse, de uma maneira mais simples, a inserção de novos *tokens* no final da fila de um recurso (Facility), usando-se o parâmetro `AT_REAR`. O simulador, no seu modo padrão, faz a inserção dos *tokens* nas filas seguindo uma ordem de prioridades decrescente, sendo FIFO para *tokens* de mesma prioridade (`BEHIND_PRIORITY_KEY`).

Foi implementado também um novo método na classe Linked, denominado *RemoveSel()* (Figura 6.7), que remove um elemento da fila de um recurso somente se ele pertencer a uma determinada classe (`MATCHING_CLASS`). O comportamento padrão do simulador é remover sempre o primeiro elemento da fila do recurso (`FROM_FRONT`). O método Release(), definido na classe Facility, é o responsável por fazer a chamada aos métodos Remove() e RemoveSel() e teve que ter sua lista de argumentos alterada a fim de dar suporte à classe procurada e ao parâmetro de *look-ahead* utilizado no algoritmo.

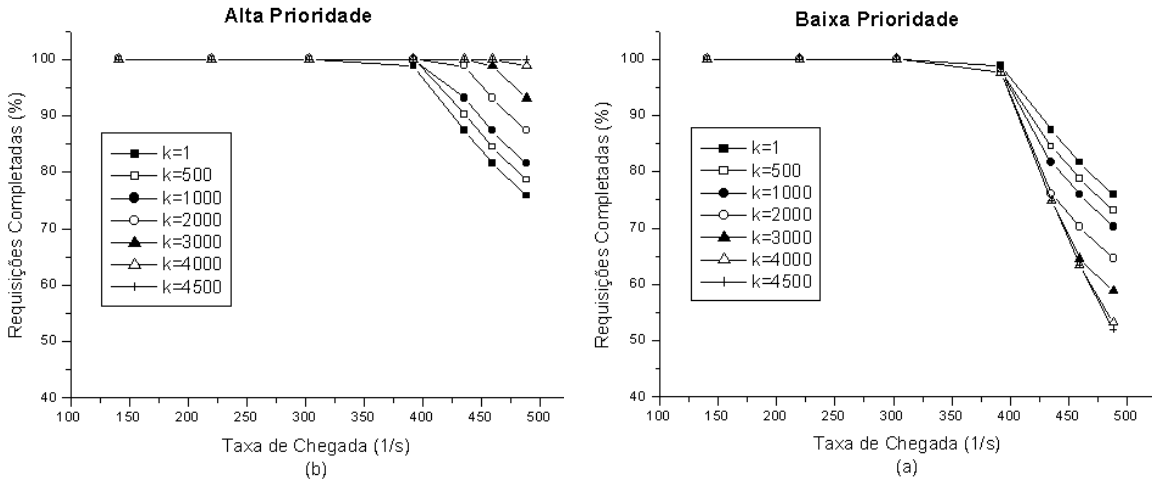
O uso do *look-ahead* como parâmetro de controle do algoritmo permite regular o nível de priorização do sistema ou, em outras palavras, determina quão rigoroso será o esquema de prioridades empregado. O algoritmo PRIAdap introduz características de adaptabilidade no modelo do servidor SWDS, o qual pode ser ajustado para fornecer diferentes níveis de serviço de acordo com a carga presente no sistema. Evidentemente, existe uma relação de custo-benefício entre o rigor da qualidade de serviço oferecida e o desempenho global do sistema, em termos do tempo de resposta das requisições. Outra vantagem do mecanismo proposto é que ele é distribuído por natureza. Muitos dos esquemas encontrados atualmente (Cardellini *et al.*, 2001b; Casalicchio & Colajanni, 2000; Chen & Mohapatra, 1999; Rao & Ramamurthy, 2001) dependem de algum componente central (um despachante) para realizar a diferenciação de serviços. O algoritmo adaptativo proposto transfere a carga associada com a diferenciação de serviços para os nós do *cluster*, o que por sua vez alivia a carga no despachante e melhora a escalabilidade e a confiabilidade do sistema (Teixeira *et al.*, 2004a).

6.4.2 Resultados Experimentais

Inicialmente, foram realizadas simulações buscando avaliar a influência do valor escolhido para o *look-ahead* no percentual de requisições completadas com sucesso. Consideram-se 50% de clientes na classe de alta prioridade. Primeiramente, foi executada uma simulação com $k = 1$, a fim de determinar o maior valor a ser utilizado para o *look-ahead*. O tamanho máximo observado para as filas dos nós, neste caso (denominado max_fila), foi de 4.300 requisições.

Os experimentos seguintes procuram analisar a variação do percentual de requisições completadas com sucesso, para as diferentes classes de serviço, com valores do *look-ahead* variando de 500 até 4.500, a diferentes taxas de chegada. Pode-se notar que valores progressivamente maiores de k aumentam gradativamente o percentual de requisições de alta prioridade que conseguem ser completadas (Figura 6.8(a)), ao mesmo tempo em que pioram sensivelmente o atendimento recebido pelas requisições de baixa prioridade (Figura 6.8(b)).

Para $k = 1$, o atendimento recebido pelas requisições de ambas as classes é virtualmente o mesmo. Nessa situação, para taxas de chegada superiores a 400 requisições por



(a) Alta prioridade (b) Baixa prioridade

Figura 6.8: Percentual de requisições completadas com diferentes valores do *look-ahead*

segundo, algumas requisições de alta prioridade nem mesmo chegam a ser completadas.

Por outro lado, para $k > max\_fila$, o algoritmo adaptativo comporta-se como o mecanismo de prioridades rigoroso comentado anteriormente. Nesse caso, as requisições de baixa prioridade têm o seu pior desempenho, com menos de 50% delas sendo completadas com sucesso. Para $500 \leq k \leq 4000$, constatam-se os outros níveis de diferenciação de serviços atingidos, conforme a proposta inicial do algoritmo PRIAdap, que é a de regular o nível de priorização empregado pela arquitetura.

O ajuste do parâmetro de *look-ahead* é crucial para o desempenho do servidor SWDS. Um valor de k por demais elevado pode levar à negação de serviço para as requisições de menor prioridade. Por outro lado, um valor muito baixo prejudica a diferenciação de serviços. O *look-ahead* pode ser ajustado manualmente pelo administrador do sistema, ou dinamicamente a partir das informações da carga de trabalho, obtidas em tempo real.

Para concluir, também foi analisado o comportamento do tempo de resposta médio das requisições, mostrado na Figura 6.9. Observa-se que, para $k = 1$, as curvas se sobrepõem, pois o mesmo tratamento é dispensado a ambas as classes. Entretanto, para $k = 3000$, a diferenciação de serviços já se torna evidente e o atendimento recebido pelas requisições de alta prioridade é visivelmente melhor.

Os experimentos realizados permitem concluir que a variação controlada do valor do *look-ahead* constitui-se em um mecanismo apropriado para fornecer diferenciação de serviços entre diferentes classes de requisições. O algoritmo PRIAdap evita os problemas de negação de serviço evidenciados no mecanismo de prioridades rigoroso e permite ao administrador do sistema um controle mais efetivo sobre a qualidade do serviço oferecida aos clientes. O servidor SWDS pode, então, ser ajustado segundo as características da carga de trabalho e as políticas vigentes na organização.

O *look-ahead* também pode ser atualizado automaticamente pelo sistema, a partir

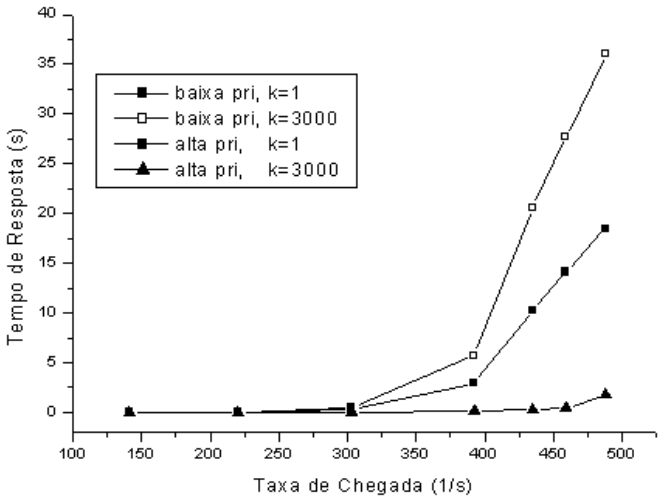


Figura 6.9: Tempo de resposta (PRIAdap)

de informações obtidas do módulo de Controle de Admissão. Este parâmetro seria, então, ajustado periodicamente em função de certos objetivos colocados pelo administrador como, por exemplo, o tempo de resposta máximo esperado para as requisições de alta prioridade ou o *throughput* médio que se busca atingir para as requisições de uma determinada classe de serviço. Em um cenário real, esses objetivos seriam, de fato, estabelecidos em Acordos de Níveis de Serviço (*Service Level Agreements* — SLAs) firmados com os clientes do sistema. Neste trabalho, não se chegou a implementar a atualização automática do *look-ahead*.

A introdução de prioridades no modelo do servido SWDS, como se pôde constatar, abre um leque de possibilidades, pois permite o desenvolvimento de diversos algoritmos de diferenciação de serviços, como é o caso dos mecanismos rigoroso e adaptativo aqui descritos. Outro encaminhamento deste trabalho seria a adaptação de algoritmos encontrados em nível de rede, por exemplo, o esquema *Weighted Fair Queueing* (WFQ) (Demers *et al.*, 1990; Parekh & Gallager, 1993), que consegue garantir largura de banda para certos serviços de rede, permitindo que várias fontes compartilhem o mesmo meio de comunicação. No caso de um servidor web, o desafio consiste em transportar o algoritmo WFQ (projetado segundo uma visão de pacotes) para o nível de aplicação. Nesse caso, seria preciso considerar o tamanho das requisições HTTP no cálculo da parcela de processamento alocada a cada classe de serviço e ajustar periodicamente os pesos atribuídos a cada uma delas. Este e outros exemplos são possíveis desdobramentos futuros do presente projeto.

6.5 Considerações Finais

Este capítulo tratou sobre a implementação de dois mecanismos de escalonamento baseados em prioridades, rigoroso e adaptativo, no modelo do Servidor Web com Diferenciação de Serviços. O mecanismo rigoroso, comumente usado em sistemas operacionais e dispositivos de rede, revelou-se uma abordagem bastante eficiente para o fornecimento de serviços diferenciados também em servidores web, funcionando satisfatoriamente com diferentes níveis de carga de trabalho e configurações do servidor. Os experimentos mostraram, contudo, que este mecanismo opera melhor quando o sistema não está muito sobrecarregado e também que é importante atentar para que a distribuição dos clientes pelas classes de serviço seja adequada, a fim de evitar a monopolização dos recursos pelas requisições de maior prioridade.

O mecanismo de prioridades adaptativo utiliza um parâmetro de *look-ahead* nas filas de espera do *cluster* que determina o nível de priorização empregado pelo sistema, conferindo uma maior ou menor importância às requisições de alta prioridade. Para valores elevados do *look-ahead*, este algoritmo comporta-se como o caso rigoroso e, para valores próximos de um, as requisições são atendidas segundo sua ordem de chegada, anulando-se a diferenciação de serviços. O mecanismo adaptativo evita os problemas observados no esquema rigoroso e permite que o administrador do sistema tenha um controle mais efetivo sobre o atendimento dispensado aos clientes. Com isso, introduz-se uma certa adaptabilidade no servidor SWDS, tornando-o capaz de oferecer diferentes níveis de QoS a seus clientes de acordo com a carga presente no sistema. Além disso, o algoritmo adaptativo realiza a diferenciação de serviços de forma distribuída, transferindo essa atribuição para os nós do *cluster*, o que aumenta a escalabilidade do sistema. Este mecanismo mostrou-se mais apropriado para um ambiente altamente dinâmico, como é o caso da Web.

O capítulo a seguir trata da implementação do módulo de Controle de Admissão no modelo.

Capítulo

7

Controle de Admissão

7.1 Introdução

Nos dois capítulos anteriores, estudaram-se os algoritmos de diferenciação de serviços implementados no modelo do servidor SWDS. Aqueles algoritmos correspondem à funcionalidade de escalonamento de tarefas do sistema e respondem a duas questões:

- A qual processo servidor devem ser alocadas as requisições aceitas?
- Em que ordem devem ser atendidas as requisições?

Entretanto, o fornecimento de serviços diferenciados aos clientes pode tornar-se seriamente comprometido, caso não haja uma forma eficiente de controle da sobrecarga do sistema. Nesse caso, a pergunta a ser respondida consiste em:

- Quais requisições descartar quando o servidor está sobrecarregado?

Este capítulo descreve o módulo de Controle de Admissão proposto para o servidor SWDS, destacando seus principais componentes, métricas utilizadas e arquitetura. São descritos três mecanismos de controle implementados no modelo: o primeiro deles consiste em limitar o tamanho das filas dos nós. O seguinte procura controlar o tempo de resposta por meio de *buffers* de tamanho fixo alocados às classes de serviço. Finalmente, tem-se um mecanismo que se baseia em uma média ponderada da utilização do sistema para a tomada de decisão. As soluções propostas são comparadas em diferentes situações e configurações de carga de trabalho.

7.2 Arquitetura do Módulo de Controle de Admissão

7.2.1 Componentes

O módulo de *Controle de Admissão* presente no modelo do servidor SWDS (Figura 4.1) tem a função de gerenciar a aceitação de novas requisições pelo servidor, impedindo que o mesmo atinja a sobrecarga. Para tanto, utiliza informações atualizadas da carga presente no sistema e leva em conta as políticas de atendimento vigentes.

A arquitetura do Controle de Admissão é mostrada na Figura 7.1. Destacam-se os seguintes componentes:

a) *Área de Variáveis Globais*: guarda informações referentes ao estado do sistema e às métricas relevantes para o Controle de Admissão. Atualmente, as métricas usadas são o tamanho das filas do *cluster*, o tempo de resposta das requisições (instantâneo e médio) e a utilização do sistema. Esta área pode ser ampliada de acordo com a necessidade;

b) *Área de Buffers*: compreende *buffers* utilizados, por exemplo, para limitar o número de clientes de uma certa classe presentes no servidor, número de clientes em fila e outros;

c) *Coleta de Informações*: este componente recolhe informações atualizadas da carga do sistema, que servem como base para a tomada de decisão;

d) *Mecanismos de Controle*: são os mecanismos de controle de admissão implementados no servidor SWDS, discutidos posteriormente.

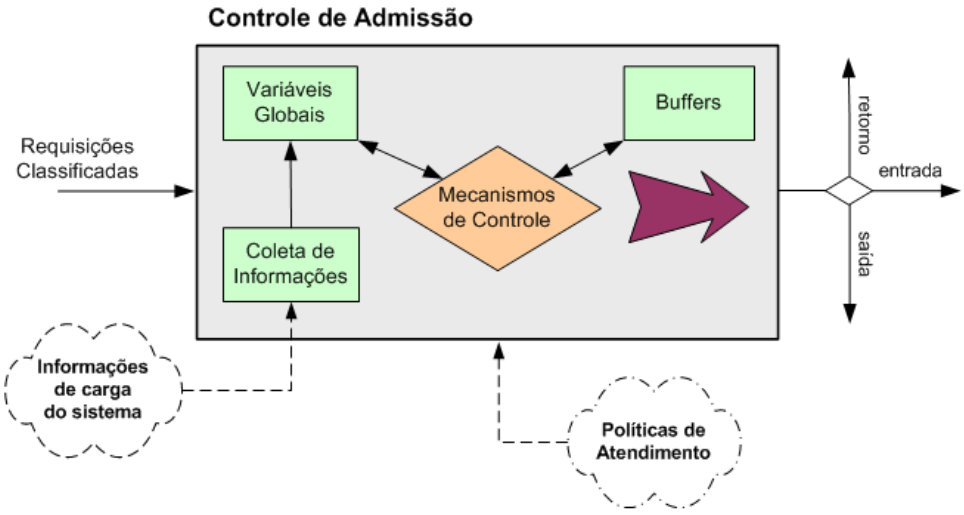


Figura 7.1: Módulo de Controle de Admissão

O Controle de Admissão foi projetado de forma flexível, podendo-se ampliá-lo para abrigar diferentes mecanismos de controle de sobrecarga. Na versão atual do servidor

SWDS, estão em funcionamento três mecanismos distintos: o primeiro deles limita as filas dos processos servidores a um tamanho máximo e recusa novas requisições quando esse limite for atingido. O segundo faz o controle a partir do tempo de resposta das requisições de maior prioridade e aloca *buffers* de tamanho fixo para as diferentes classes, como uma forma de limitar a carga no sistema. O terceiro mecanismo baseia suas decisões de descarte numa média exponencialmente ponderada da utilização do sistema, podendo-se ajustá-lo para ser mais ou menos sensível a mudanças no perfil da carga de trabalho. A Seção 7.3 apresenta esses mecanismos e seus resultados detalhadamente.

Além de coletar informações da carga atual do sistema, o Controle de Admissão também recebe como entrada as *Políticas de Atendimento* estabelecidas pelo administrador do sistema, as quais norteiam a escolha e aplicação dos mecanismos de controle. Uma política de atendimento é uma meta a ser atingida, podendo ser algo tão genérico quanto garantir que as requisições de alta prioridade tenham um tratamento preferencial (o que é óbvio) ou algum objetivo mais específico, por exemplo, o tempo de resposta da Classe 1 nunca será superior a um segundo ou o percentual de requisições completadas com sucesso deverá ser sempre superior a 90%. Diferentes políticas podem se mapear para um ou mais mecanismos de controle. Pode até mesmo não haver mecanismos disponíveis que satisfaçam a uma certa política, levando à necessidade de criação de novos algoritmos. A Seção 7.3 apenas descreve os mecanismos implementados até o momento e seu funcionamento, sem relacioná-los especificamente a qualquer política de atendimento a clientes, cuja definição será feita em trabalhos futuros.

No estágio atual do projeto, não foi implementada a funcionalidade de *Negociação* prevista no modelo do servidor SWDS. Portanto, uma requisição, após passar pelo Controle de Admissão, será apenas aceita ou descartada pelo servidor. Futuramente, no caso de uma requisição ser recusada, o servidor SWDS poderá trocar informações com o cliente que a emitiu e propor a aceitação da requisição em uma classe de serviço inferior. Outra possibilidade é fazer uma espécie de negociação forçada, em que a nova requisição é remarcada para a classe de serviço imediatamente inferior e, então, tenta-se novamente sua admissão no sistema. No pior caso, a requisição acabaria sendo admitida na classe de melhor esforço, embora isso não seja conveniente para alguns tipos de aplicações.

7.2.2 Seleção das Métricas

A eficiência do controle de admissão está relacionada à disponibilidade de informações precisas sobre o estado do sistema, as quais servirão de base para a tomada de decisões. Normalmente, a decisão de descarte baseia-se na análise da carga do sistema, o que, em tese, parece simples. Entretanto, às vezes é difícil chegar a um consenso sobre quando a sobrecarga é atingida. Além disso, deve-se ser capaz de aferir o nível de carga sem afetar negativamente o desempenho do sistema. Outra questão muito importante está na seleção

apropriada das métricas a serem utilizadas pelo controle de admissão.

Existem várias maneiras de medir a carga de um sistema, de acordo com o objetivo que se tenha em mente. Exemplos de métricas comumente utilizadas são o tempo de resposta, a utilização do sistema, a taxa de chegada, o tamanho das filas, o nível de ocupação dos *buffers*, dentre outras. Uma vez selecionada a métrica, precisa-se determinar sua periodicidade. Esta pode variar desde medições instantâneas, feitas a cada segundo, até um acompanhamento ao longo de um período extenso. Quanto mais freqüente for a medição, mais fielmente irá refletir o estado do sistema. Contudo, é sabido que medições muito freqüentes podem interferir no desempenho global.

Outra questão a ser considerada é a abrangência da métrica, a qual pode se referir a uma classe de serviço específica ou ao comportamento global do sistema. Métricas que dizem respeito a uma determinada classe permitem acompanhar de perto o comportamento de suas requisições, porém perde-se a visão sistêmica. Métricas globais, por outro lado, fornecem um panorama geral, mas perdem-se os detalhes de cada classe de serviço individualmente, informação essencial para honrar os acordos de serviço (SLAs) previamente estabelecidos. E ainda, as decisões de descarte podem ser tomadas a partir de limiares rigidamente estabelecidos ou usando-se mecanismos mais flexíveis, como funções de probabilidade e históricos de observação.

A seleção da métrica a ser utilizada, sua periodicidade, abrangência e limiares pré-estabelecidos vão depender dos objetivos colocados para o controle de admissão. Para cada caso, o administrador do sistema deverá ser capaz — supostamente — de selecionar um conjunto de parâmetros que possa levar a uma decisão de aceitação ou descarte satisfatória.

Para o Controle de Admissão do servidor SWDS, foram escolhidas como métricas o tamanho das filas do servidor, o tempo de resposta da classe de maior prioridade e uma média exponencialmente ponderada da utilização do sistema.

7.3 Mecanismos de Controle

7.3.1 Metodologia de Teste

A simulação foi a abordagem escolhida para validar os mecanismos de controle de admissão propostos neste Capítulo. Foi implementado um modelo do servidor SWDS conforme a Figura 4.1, em que o *cluster* é formado por quatro servidores web homogêneos. A parametrização do modelo segue o exposto na Seção 4.3.1. Nos experimentos, as requisições que chegam ao sistema são divididas em duas classes, de alta e baixa prioridade (Classes 1 e 0), pelo módulo Classificador da arquitetura, sendo atribuídas aos nós segundo um esquema de rodízio. Consideram-se 50% de clientes na classe de alta prioridade. Para a diferenciação de serviços, emprega-se o mecanismo de prioridades adaptativo

já discutido.

A carga de trabalho é gerada a partir de *traces* de acesso aos servidores web da Copa do Mundo 98 (Seção 4.3.3). Assume-se que todos os nós têm acesso aos mesmos documentos. Os experimentos realizados pretendem verificar a eficiência dos mecanismos de controle de admissão propostos e as vantagens, advindas da sua utilização, para o desempenho do servidor SWDS e para a diferenciação de serviços entre as classes.

7.3.2 Configuração de Referência

Inicialmente, foi feito um experimento sem o emprego do controle de admissão, a fim de que se pudesse ter uma referência com a qual comparar os resultados obtidos com os mecanismos de controle. A faixa do *log* escolhida para a geração de carga foi entre 3,6 e 4,6 milhões de registros, usando-se todos os servidores do *log*. Nesta faixa, ocorre um súbito aumento da carga de trabalho, situação ideal para se avaliar a eficiência do controle de admissão. O algoritmo de diferenciação de serviços empregado foi o PRIAdap, com um valor de *look-ahead* igual a 300. O modelo foi configurado como descrito na Seção 7.3.1.

A Figura 7.2 mostra o comportamento do tempo de resposta no decorrer da simulação. Constata-se uma situação caótica, pois o mesmo sobe continuamente, acompanhando o aumento da carga de trabalho que chega ao servidor. A partir de cerca de 500 segundos de tempo de simulação, quando ocorre um súbito incremento na carga, as novas requisições tendem a esperar cada vez mais para ser atendidas. Embora a diferenciação de serviços seja respeitada, observa-se que o algoritmo PRIAdap sozinho não é capaz de garantir uma qualidade mínima de atendimento para nenhuma das duas classes. No pior caso, o tempo de resposta chega a quase 90 segundos, o que inviabiliza o atendimento aos clientes.

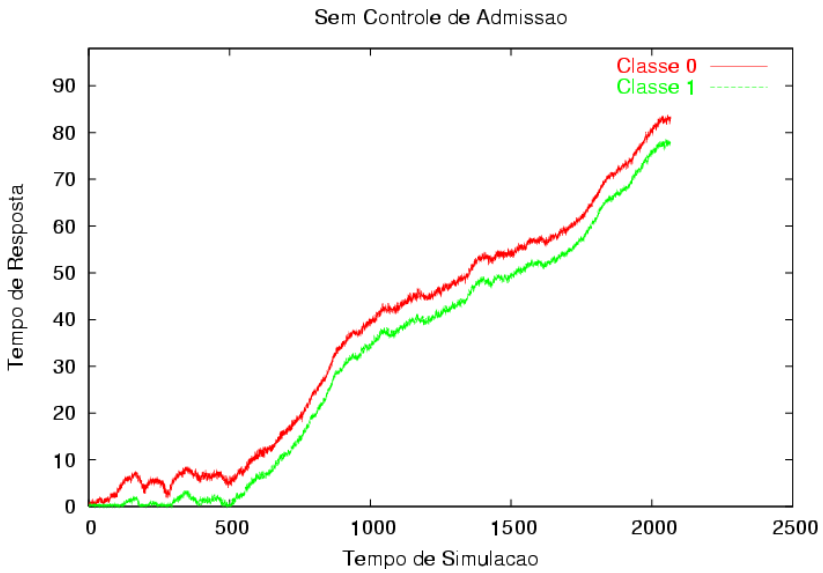


Figura 7.2: Tempo de resposta sem utilização do Controle de Admissão

A observação do tamanho das filas dos servidores explica o excessivo aumento no

tempo de resposta (Figura 7.3). Como nenhuma requisição HTTP que chega ao sistema é descartada, as filas atingem mais de 9.700 clientes e a utilização do *cluster* chega a quase 100%, o que denota sua sobrecarga. O tempo de resposta médio é superior a 30 segundos. Além disso, as estatísticas por classe de serviço mostram que cerca de 4% das requisições de ambas as classes nem chegam a ser completadas durante o tempo da simulação, portanto o esforço de admiti-las no sistema terá sido inútil.

```
Estatisticas por Recurso
------------------------
F 1 (CLASSIF): Idle: 94.0%, Util: 6.0%, Preemptions: 0, LongestQ: 624
F 2 (CTRLADM): Idle: 100.0%, Util: 0.0%, Preemptions: 0, LongestQ: 0
F 3 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 9761
F 4 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 9778
F 5 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 9795
F 6 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 9708

Estatisticas por Classe (0:baixa, 1:alta)
-----------------------
Chegadas  (0/1): 503763 495166
Admissoes (0/1): 503763 495165
Terminos  (0/1): 483813 476510
Tempo Resid Medio (0/1): 36.24 31.37
```

Figura 7.3: Resultados da simulação (sem controle de admissão)

Em resumo, numa situação como esta, o emprego de alguma forma de controle de admissão é essencial para que se possa garantir uma qualidade de serviço mínima aos clientes. Caso contrário, até mesmo as classes de maior prioridade podem vir a ter seu desempenho seriamente comprometido, como mostrado na Seção 6.3.3.

7.3.3 Admissão segundo o Tamanho das Filas

Descrição do Mecanismo

Este mecanismo estabelece um tamanho máximo para as filas dos servidores do *cluster*, de acordo com o valor do parâmetro `MAXFILA`. Se uma requisição for atribuída a um nó cuja fila tenha atingido esse limiar, então ela será sumariamente recusada, independentemente de sua classe.

Abordagem semelhante a essa é encontrada em nível de rede e consiste na rejeição dos pacotes mais novos que chegam a um determinado dispositivo, por exemplo, um roteador[1]. O servidor web Apache também lança mão de um mecanismo desse tipo, rejeitando novas requisições HTTP caso o tamanho de sua fila ultrapasse 1.024 clientes.

Embora simples, este mecanismo foi o primeiro a ser implementado no servidor SWDS, objetivando-se adquirir uma familiaridade inicial com a tarefa de controle de admissão.

[1]Na literatura sobre o assunto, recebe o nome de *tail drop*, significando que serão descartados os pacotes que estiverem no final do fila, isto é, os mais recentes.

Resultados Experimentais

Nos experimentos, o valor do parâmetro `MAXFILA` foi fixado em 1.024 posições, a exemplo do servidor Apache. Para a geração de carga, foi utilizada a faixa do *log* entre 3,6 e 4,6 milhões de registros, usando-se o algoritmo PRIAdap com *look-ahead* igual a 300. Os experimentos foram conduzidos como na Seção 7.3.2, para que se pudesse fazer comparações.

No caso sem controle de admissão, o tempo de resposta crescia proibitivamente a partir de 500 segundos de simulação, devido ao súbito aumento na carga de trabalho. Entretanto, com a limitação do tamanho das filas em 1.024 clientes, o tempo de resposta tende a se estabilizar em torno de 11 segundos para a classe de baixa prioridade e cerca de 6 segundos para as requisições de alta prioridade, como mostrado na Figura 7.4. A diferenciação de serviços fornecida pelo algoritmo PRIAdap é respeitada. Testes realizados com valores maiores do *look-ahead* mostraram que estes provocam o aumento da distância entre as curvas.

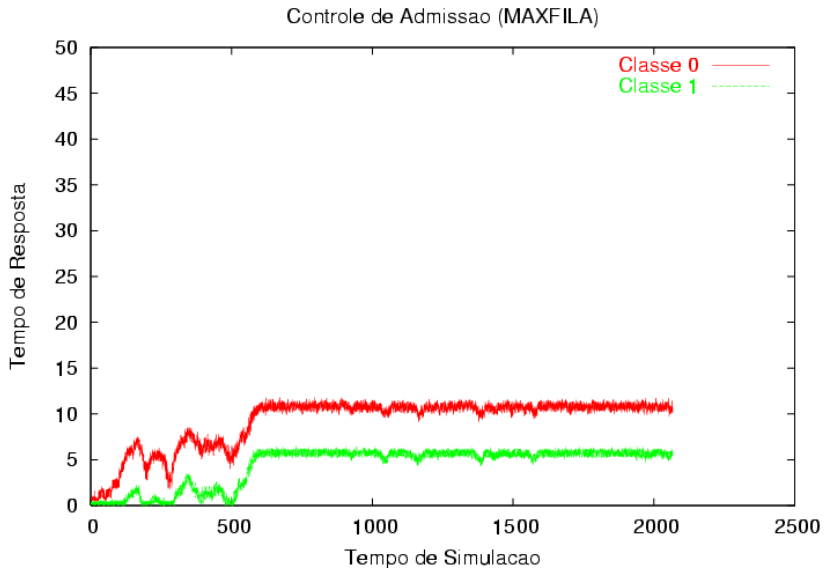


Figura 7.4: Tempo de resposta com Controle de Admissão (`MAXFILA`)

A oscilação inicial da curva do tempo de resposta não é devida a aspectos como *warm-up* da simulação ou similares. Esta deve-se única e exclusivamente ao fato de que, nesta faixa do *log*, a carga de trabalho não é suficiente para saturar o servidor, fato que só ocorre a partir de 500 segundos, quando, então, é ativado o controle de admissão. A escolha deste trecho do *log* para a geração de carga foi intencional.

A análise dos resultados da simulação (Figura 7.5) comprova que o tamanho máximo das filas não ultrapassa 1.024 clientes, como pretendido inicialmente. Entretanto, os nós do *cluster* continuam apresentando quase 100% de utilização, o que novamente evidencia a sobrecarga do sistema, embora menor que no caso da Seção 7.3.2. O tempo de resposta médio da classe de alta prioridade cai de 31,37 segundos, no caso sem controle de admissão,

para 4,47 segundos, uma diminuição de 86%. Tal fato se dá porque o número de clientes em fila também é cerca de 90% menor, portanto reduz-se o seu tempo médio de espera. Isso mostra que, mesmo um mecanismo simples como o aqui relatado, pode produzir bons resultados. Observa-se que as tarefas de classificação e controle de admissão não prejudicam o desempenho do sistema, dada a baixa utilização desses recursos (`CLASSIF` e `CTRLADM`).

```
Estatisticas por Recurso
------------------------
F 1 (CLASSIF): Idle: 94.0%, Util: 6.0%, Preemptions: 0, LongestQ: 624
F 2 (CTRLADM): Idle: 87.9%, Util: 12.1%, Preemptions: 0, LongestQ: 339
F 3 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 1024
F 4 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 1024
F 5 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 1024
F 6 (SWEB): Idle:  0.1%, Util: 99.9%, Preemptions: 0, LongestQ: 1024

Estatisticas por Classe
-----------------------
Chegadas  (0/1): 503801 495128
Admissoes (0/1): 486188 477815
Terminos  (0/1): 483720 476597
Tempo Resid Medio (0/1): 9.33 4.47
```

Figura 7.5: Resultados da simulação com Controle de Admissão (`MAXFILA`)

Também se constata uma melhoria no percentual de requisições não completadas no tempo da simulação. Como o número de admissões é menor que no caso sem controle de admissão (cerca de 96,5% do total de chegadas são aceitas, para a classe de alta prioridade), tem-se que apenas 0,25% das requisições de alta prioridade aceitas não conseguem ser completadas, uma melhoria em relação ao caso anterior. Para cargas maiores, ainda assim esse percentual seria mantido, uma vez que o parâmetro `MAXFILA` limita o número máximo de clientes no sistema. Raciocínio semelhante se aplica às requisições de baixa prioridade.

Em resumo, o mecanismo proposto consegue diminuir o tamanho das filas e, conseqüentemente, o tempo médio de espera das requisições. Entretanto, ele pode não ser capaz de impedir a sobrecarga do sistema e não fornece garantias de QoS aos clientes, qualquer que seja sua classe de serviço. Sendo assim, conclui-se que este mecanismo de controle não deve ser usado isoladamente, mas sim em conjunto com outros, como um último recurso do qual se pode lançar mão para impedir que ocorram situações extremas de sobrecarga. De fato, nos experimentos relatados nas seções seguintes, o parâmetro `MAXFILA` esteve sempre ajustado para 1.024, embora este limite não tenha sido atingido.

7.3.4 Admissão segundo o Tempo de Resposta

Descrição do Mecanismo

Este mecanismo procura controlar a carga no sistema usando como referência o tempo de resposta das requisições da classe de alta prioridade (`THRESHOLD`), o qual é colhido a cada término de serviço. Seu funcionamento consiste em aceitar as requisições de alta prioridade sempre que possível e somente admitir aquelas de baixa prioridade quando a carga estiver abaixo de um limiar pré-determinado. Usa-se aqui o princípio de limiares rígidos (*hard thresholds*) para orientar as decisões de descarte.

Adicionalmente, são alocados dois *buffers* de tamanho fixo para as requisições de cada classe (`BUFLOW` e `BUFHIGH`), a fim de controlar o número de máximo de clientes de cada categoria presentes no sistema. O objetivo da utilização dos *buffers* é proporcionar uma maior estabilidade ao tempo de resposta, evitando que o mesmo apresente picos quando ocorrerem súbitos aumentos na carga de trabalho.

O algoritmo de controle de admissão, para duas classes de serviço, é mostrado na Figura 7.6. Se a carga (o tempo de resposta) for menor que `THRESHOLD`, o sistema aceitará qualquer cliente da classe alta e os clientes da classe baixa só serão admitidos até o limite estabelecido por `BUFLOW`. Caso a carga exceda `THRESHOLD`, os clientes da classe alta só serão aceitos até o limite imposto por `BUFHIGH` e nenhum cliente da classe baixa será admitido no sistema. Observe-se que o algoritmo não requer, necessariamente, que seja utilizado o tempo de resposta como métrica de controle.

```
bool adm = true;

Se (carga < THRESHOLD)
    Se (classe == 0 && maxcli[0] == 0)    // maxcli[0] varia de BUFLOW até 0
        adm = false;
    fim-se
senão
    Se (classe == 1)
        Se (maxcli[1] == 0)               // maxcli[1] varia de BUFHIGH até 0
            adm = false;
        fim-se
    senão
        adm = false;
    fim-se
fim-se

Se (adm)
    maxcli[classe]−−;
fim-se
return adm;
```

Figura 7.6: Algoritmo de Controle de Admissão (*Hard Thresholds*)

Neste mecanismo, pode ser complicado determinar com exatidão o tamanho dos *buffers* e o valor do `THRESHOLD` necessários para alcançar determinados requisitos de serviço. Em geral, deve-se pensar em termos da qualidade que se espera oferecer às requisições de alta prioridade e, então, alocar um tamanho de *buffer* que pareça apropriado. Quanto às requisições de baixa prioridade, estas são coadjuvantes e sua admissão no sistema não deverá prejudicar o desempenho das classes mais privilegiadas. A limitação do número de clientes de baixa prioridade também evita que a sua população cresça sem controle, o que contribui para manter o serviço que lhes é oferecido dentro de certos padrões mínimos de qualidade.

Resultados Experimentais

Nos experimentos, seguiu-se a metodologia da Seção 7.3.1. Para a geração da carga de trabalho, foi utilizada a faixa do *log* entre 3,6 e 4,6 milhões de registros, como no caso anterior. Para a diferenciação de serviços, foi empregado o algoritmo PRIAdap com um valor de *look-ahead* igual a 300. O limiar do tempo de resposta para a classe de alta prioridade foi fixado em 1,0 segundo e os tamanhos de *buffer* para as classes de alta e baixa prioridade, em 100 e 1000 posições, respectivamente. O tamanho máximo das filas foi ajustado em 1.024 clientes (Figura 7.7).

```
const int   MAXFILA   = 1024;
const long  BUFLOW    = 1000;
const long  BUFHIGH   = 100;
const float THRESHOLD = 1.0;
```

Figura 7.7: Parâmetros de controle (`THRESHOLD`)

A Figura 7.8 mostra o comportamento do tempo de resposta ao longo da simulação. Inicialmente, ocorre uma certa instabilidade, devido à própria característica da carga de trabalho. A partir de 500 segundos, quando há um súbito aumento na carga oferecida ao sistema (Figura 7.2), o tempo de resposta tende a estabilizar-se para ambas as classes de serviço. Os valores são bem menores que os observados na Figura 7.4 (`MAXFILA`) e mantêm-se assim durante todo o experimento, alheios ao aumento contínuo da carga de trabalho. Isso ocorre porque os *buffers* `BUFLOW` e `BUFHIGH` funcionam como limitantes do número de clientes no sistema. Em particular, o tempo de resposta da classe de maior prioridade mantém-se sempre abaixo do limiar estabelecido por `THRESHOLD`, além de permanecer estável. A diferenciação de serviços é respeitada e o afastamento entre as duas curvas é função do valor escolhido para o *look-ahead*.

A observação do tamanho máximo das filas, na Figura 7.9, mostra que elas estão cerca de 67% menores que no caso da utilização de `MAXFILA` (Figura 7.5), devido à presença dos *buffers*. O tempo de resposta médio da classe de alta prioridade caiu de 4,47 segundos para 0,27 segundos, uma diminuição de 94%. As requisições de baixa prioridade também

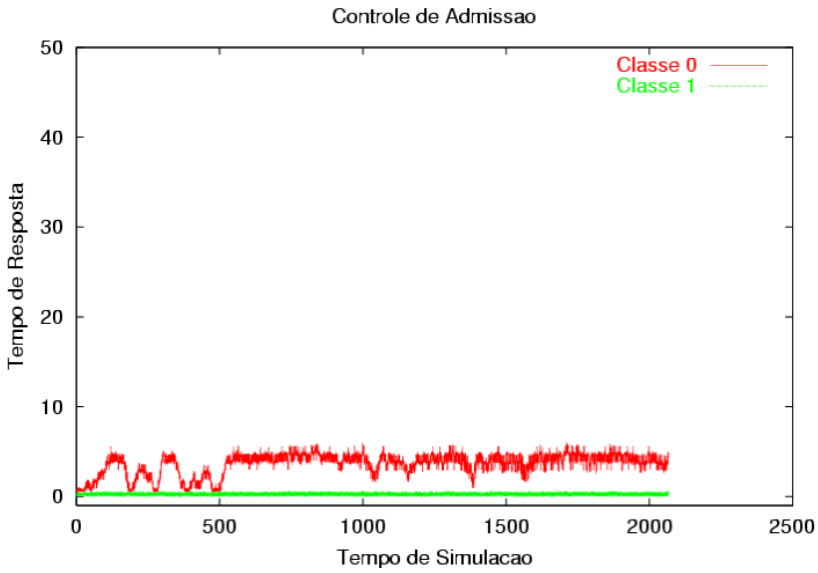


Figura 7.8: Tempo de resposta com Controle de Admissão (THRESHOLD)

são favorecidas, apresentando uma queda de 61,3% no seu tempo de resposta médio.

```
Estatisticas por Recurso
------------------------
F 1 (CLASSIF): Idle: 94.0%, Util: 6.0%, Preemptions: 0, LongestQ: 624
F 2 (CTRLADM): Idle: 87.9%, Util: 12.1%, Preemptions: 0, LongestQ: 339
F 3 (SWEB): Idle:  0.3%, Util: 99.7%, Preemptions: 0, LongestQ: 339
F 4 (SWEB): Idle:  0.2%, Util: 99.8%, Preemptions: 0, LongestQ: 331
F 5 (SWEB): Idle:  0.3%, Util: 99.7%, Preemptions: 0, LongestQ: 341
F 6 (SWEB): Idle:  0.3%, Util: 99.7%, Preemptions: 0, LongestQ: 320

Estatisticas por Classe
-----------------------
Chegadas  (0/1): 503801 495128
Admissoes (0/1): 464778 495127
Terminos  (0/1): 464014 495127
Tempo Resid Medio (0/1): 3.61 0.27
```

Figura 7.9: Resultados da simulação com Controle de Admissão (THRESHOLD)

Quanto ao número de admissões, observa-se que há uma redução de 4,40% no percentual de requisições de baixa prioridade admitidas pelo servidor, devido ao limite imposto por BUFLOW. Por outro lado, há um acréscimo de 3,49% na admissão de requisições de alta prioridade, as quais acabam sendo favorecidas. Observa-se também que todas as requisições de alta prioridade aceitas pelo sistema são concluídas com sucesso, prova de que a admissão controlada dos clientes permite oferecer um serviço ainda melhor.

Considerações sobre o Mecanismo

As conclusões acima, embora refiram-se ao experimento descrito, podem ser generalizadas para outras situações. Foram feitos vários experimentos, com diferentes valores

de BUFLOW, BUFHIGH e THRESHOLD. Na maioria deles, o tempo de resposta manteve-se abaixo do limiar especificado e os *buffers* contribuíram para evitar sua oscilação demasiada. Entretanto, em alguns casos, para limiares muito exigentes (abaixo de 0,5 segundo), o controle de admissão não se comportou de forma satisfatória, não sendo capaz de manter o serviço fornecido dentro dos requisitos especificados. Experimentos com outros tamanhos de *buffer* mostraram que *buffers* progressivamente menores melhoram os tempos de resposta de ambas as classes, se bem que à custa de um menor *throughput*.

Os *buffers* BUFLOW e BUFHIGH desempenham função semelhante à do parâmetro MAX-FILA, limitando o número de clientes no sistema. Adicionalmente, permitem também discriminar entre uma classe e outra. Nas simulações realizadas, a diferenciação de serviços não foi diretamente afetada pelo tamanho dos *buffers*.

Pode-se concluir que o mecanismo proposto funciona adequadamente, desde que se consiga dimensionar os parâmetros de controle (BUFLOW, BUFHIGH e THRESHOLD) de forma correta, o que não é uma tarefa trivial, devido ao número de parâmetros envolvidos e à variabilidade da carga. Como proposta de trabalho, sugere-se fazer o ajuste do tamanho dos *buffers* dinamicamente, de acordo com os requisitos de QoS previamente acertados.

7.3.5 Admissão segundo a Utilização do Sistema

Descrição do Mecanismo

Os mecanismos apresentados anteriormente fazem o controle de admissão através de medições instantâneas, ora usando o tamanho das filas, ora o tempo de resposta das requisições. Se por um lado são capazes de reagir rapidamente a mudanças no perfil da carga de trabalho, por outro podem vir a rejeitar desnecessariamente algumas requisições, como resultado de rajadas momentâneas que chegam ao servidor.

O mecanismo de controle de admissão aqui proposto emprega a *utilização média do cluster* de servidores web como métrica, a qual é calculada através de uma média exponencialmente ponderada. A escolha da utilização como parâmetro de controle é bastante conveniente, pois ela é capaz de indicar, numa visão global, se um sistema está de fato sobrecarregado. O uso de uma média, em vez do valor real da utilização, tem o objetivo de minimizar o efeito de um congestionamento momentâneo no servidor. O peso escolhido determina a sensibilidade da média a mudanças na utilização do *cluster*. Abordagem semelhante a essa é encontrada no algoritmo RED (*Random Early Detection*), o qual descarta os pacotes das filas dos roteadores antes que ocorra uma situação de sobrecarga. Para tanto, emprega uma média exponencialmente ponderada do tamanho das filas, além de limiares probabilísticos (Stallings, 2002).

O mecanismo aqui proposto funciona da seguinte forma: a utilização atual do *cluster* é medida a cada nova requisição que chega ao servidor. Esse valor é combinado com dados

históricos a fim de obter a média exponencialmente ponderada, U_{med}. Caso a mesma esteja acima de um limiar pré-estabelecido (`MAXUTIL`), então o servidor SWDS recusará quaisquer novas requisições, independentemente de sua classe, até que U_{med} caia para níveis aceitáveis, isto é, abaixo do limiar.

Seja assim definido o cálculo da média exponencialmente ponderada:

$$U_{med} = (1 - p)U_{ant} + pU_{atual} \tag{7.1}$$

onde:

- U_{ant} – valor anterior da média;
- U_{atual} – valor atual observado para a utilização;
- p – peso, onde $0 \leq p \leq 1$.

O peso p funciona como um coeficiente de sensibilidade a mudanças na carga do sistema, podendo gerar duas situações:

a) $p \rightarrow 1$: o mecanismo de controle torna-se bastante sensível a mudanças na utilização do sistema, reagindo prontamente. Indicado para situações em que o servidor esteja continuamente exposto a uma carga elevada;

b) $p \rightarrow 0$: o mecanismo de controle reage mais lentamente a mudanças na utilização do sistema, apresentando um comportamento mais estável. Indicado para situações em que a carga é geralmente baixa, apresentando algumas rajadas ocasionais.

Entre esses dois valores extremos do peso p, têm-se diferentes perfis de comportamento do módulo de Controle de Admissão, desde o mais complacente até o mais rigoroso. Na configuração (a), o servidor apresenta uma política de admissão mais restritiva, devido ao peso da utilização medida (U_{atual}) no cálculo da média. Neste caso, o tempo de resposta e as filas tendem a ser menores, pois as recusas de serviço precoces evitam a sobrecarga do sistema. Na configuração (b), a política de admissão leva em conta o histórico da carga do sistema (U_{ant}), minimizando o impacto da utilização atual na tomada de decisões. Neste caso, para cargas altas, o tempo de resposta e o número de requisições não completadas tendem a ser maiores, já que o servidor é menos criterioso na seleção das requisições.

Resultados Experimentais

Os experimentos foram feitos com a mesma configuração dos casos anteriores, para efeito de comparação. Foi escolhida a faixa do *log* entre 3,6 e 4,6 milhões de registros,

usando-se o algoritmo PRIAdap para a diferenciação de serviços, com o *look-ahead* ajustado para 300. Inicialmente, o limiar de utilização `MAXUTIL` foi fixado em 90% e o peso p em 0,9.

A Figura 7.10 mostra o comportamento do tempo de resposta no decorrer da simulação. O controle de admissão é ativado desde o início, pois a carga oferecida ao sistema supera o limiar `MAXUTIL`. Observa-se que o tempo de resposta cai drasticamente em relação ao caso sem controle de admissão (Figura 7.2) e mantém-se assim durante toda a simulação, independentemente do aumento da carga. A diferenciação de serviços entre as classes é respeitada. O comportamento da classe de alta prioridade é similar ao obtido com o mecanismo anterior (Seção 7.3.4). Já as requisições de baixa prioridade apresentam um tempo de resposta bem menor que o observado naquele caso, o qual atinge agora níveis aceitáveis para uma utilização prática do servidor SWDS.

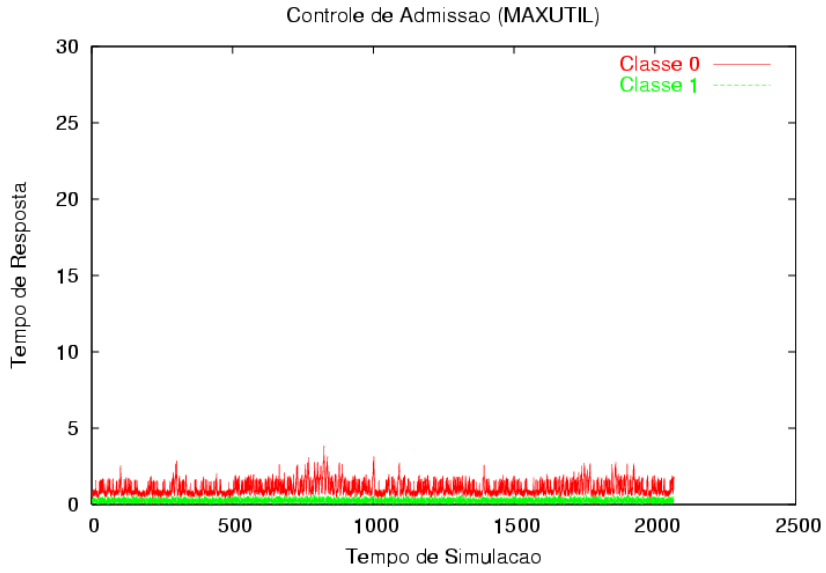


Figura 7.10: Tempo de resposta com Controle de Admissão (`MAXUTIL=90%`)

Essa situação pode ser melhor exemplificada analisando-se os resultados da Figura 7.11. Pode-se constatar que o tempo de resposta médio da classe de alta prioridade permanece em 0,27 segundos, evidência do bom tratamento recebido por esta classe já no mecanismo anterior. O tempo de resposta médio da classe de baixa prioridade, por sua vez, cai de 3,61 para 1,04 segundos, uma redução de 71%. Na verdade, como o limiar de utilização foi fixado em 90%, o mecanismo atual torna-se mais restritivo, admitindo menos requisições que no caso anterior; além disso, observa-se que 100% das requisições de baixa prioridade aceitas pelo servidor são completadas com sucesso. Essa é uma das características de um bom mecanismo de controle de admissão: somente admitir no sistema requisições que efetivamente possam ser atendidas, a fim de evitar o desperdício de recursos com tarefas que não chegarão ao final. Nos resultados acima, observa-se também um crescimento do número de recusas de requisições de alta prioridade, uma vez que o mecanismo de controle utilizado não faz diferença entre as classes de serviço ao tomar

as decisões de descarte. Contudo, não há uma queda perceptível no desempenho das requisições da classe mais privilegiada.

```
Estatisticas por Recurso
------------------------
F 1 (CLASSIF): Idle: 94.0%, Util: 6.0%, Preemptions: 0, LongestQ: 624
F 2 (CTRLADM): Idle: 87.9%, Util: 12.1%, Preemptions: 0, LongestQ: 339
F 3 (SWEB): Idle: 10.0%, Util: 90.0%, Preemptions: 0, LongestQ: 264
F 4 (SWEB): Idle: 10.0%, Util: 90.0%, Preemptions: 0, LongestQ: 264
F 5 (SWEB): Idle: 10.0%, Util: 90.0%, Preemptions: 0, LongestQ: 266
F 6 (SWEB): Idle: 10.0%, Util: 90.0%, Preemptions: 0, LongestQ: 264

Estatisticas por Classe
-----------------------
Chegadas  (0/1): 503801 495128
Admissoes (0/1): 436369 428966
Terminos  (0/1): 436369 428966
Tempo Resid Medio (0/1): 1.04 0.27
```

Figura 7.11: Resultados da simulação com Controle de Admissão (`MAXUTIL=90%`)

A observação do gráfico da utilização no decorrer do tempo (Figura 7.12) mostra que a mesma cresce rapidamente até o limite de 90% imposto pelo controle de admissão e mantém-se nesse patamar até o final, prova da eficácia do mecanismo proposto.

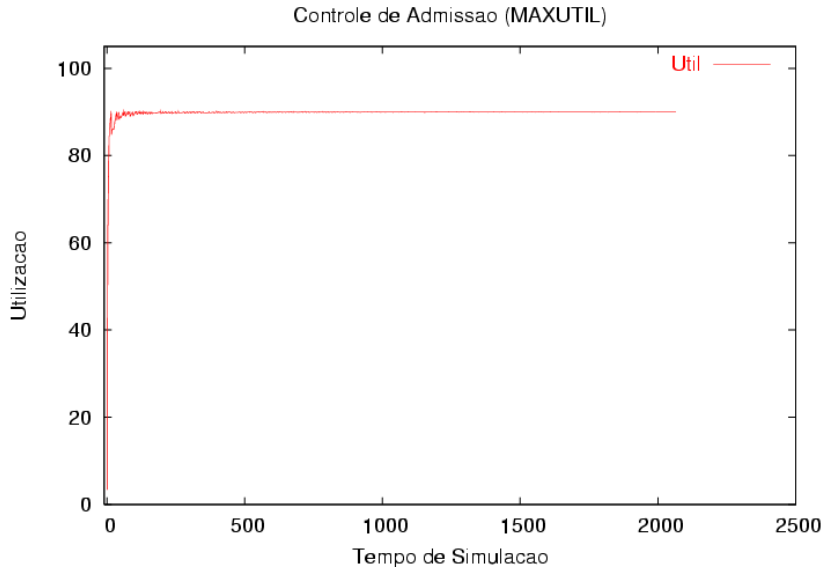


Figura 7.12: Comportamento da utilização com Controle de Admissão (`MAXUTIL=90%`)

Em seguida, o experimento anterior foi repetido com um limiar de utilização ainda mais restritivo, fixado em 60% (Figura 7.13). Novamente, a utilização permanece sempre abaixo do limiar estipulado. O tempo de resposta médio das requisições de alta prioridade continua o mesmo, porém o tempo de atendimento das tarefas de baixa prioridade cai de 1,04 para 0,78 segundo, cerca de 25% (Figura 7.14). O tamanho máximo das filas sofre uma queda de aproximadamente 47%, em conseqüência do maior número de requisições recusadas.

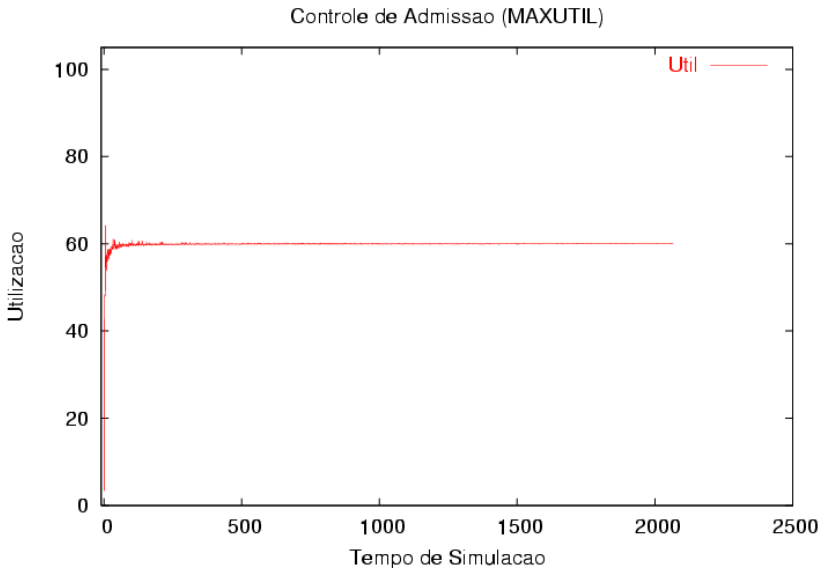


Figura 7.13: Comportamento da utilização com Controle de Admissão (`MAXUTIL=60%`)

```
Estatisticas por Recurso
------------------------
F 1 (CLASSIF): Idle: 94.0%, Util: 6.0%, Preemptions: 0, LongestQ: 624
F 2 (CTRLADM): Idle: 87.9%, Util: 12.1%, Preemptions: 0, LongestQ: 339
F 3 (SWEB): Idle: 40.0%, Util: 60.0%, Preemptions: 0, LongestQ: 143
F 4 (SWEB): Idle: 40.0%, Util: 60.0%, Preemptions: 0, LongestQ: 138
F 5 (SWEB): Idle: 40.0%, Util: 60.0%, Preemptions: 0, LongestQ: 141
F 6 (SWEB): Idle: 40.0%, Util: 60.0%, Preemptions: 0, LongestQ: 137

Estatisticas por Classe
-----------------------
Chegadas  (0/1): 503801 495128
Admissoes (0/1): 290988 285908
Terminos  (0/1): 290988 285908
Tempo Resid Medio (0/1): 0.78 0.27
```

Figura 7.14: Resultados da simulação com Controle de Admissão (`MAXUTIL=60%`)

Os resultados obtidos comprovam que o mecanismo proposto consegue efetivamente controlar a sobrecarga do servidor SWDS, mantendo a utilização sempre abaixo do limiar desejado. A classe de alta prioridade continua sendo tão bem atendida quanto antes. Adicionalmente, a classe de baixa prioridade passa a receber um tratamento mais justo, que se reflete na melhoria das médias do seu tempo de resposta e do percentual de requisições completadas. A escolha da utilização do *cluster* como referência para o controle de admissão tem conseqüências imediatas sobre o tamanho das filas, portanto sobre o tempo de resposta médio das requisições, o que contribui para a melhoria de desempenho do sistema como um todo.

Variação do Peso da Média Ponderada

O peso p da média exponencialmente ponderada funciona como um coeficiente de sensibilidade para o controle de admissão. Ele determina a rapidez com que o servidor SWDS deverá reagir a mudanças na carga de trabalho, com as implicações já comentadas no desempenho do sistema. Nos experimentos relatados previamente, utilizou-se um valor de p próximo de 1, pois a carga oferecida ao sistema era bastante elevada. Dessa forma, o controle de admissão precisava ter um perfil ágil.

O experimento anterior foi repetido com um valor menor de p, a fim de se observar o desempenho do servidor em uma situação na qual sua resposta a variações na carga de trabalho fosse mais lenta. O limiar `MAXUTIL` foi fixado em 90% e o *look-ahead* em 300. Na Figura 7.15, pode-se acompanhar o comportamento do tempo de resposta médio da classe de baixa prioridade, de 3,6 até 4,6 milhões de registros, para p igual a 0,9 e 0,001. Os resultados apresentados são a média de 10 simulações com um intervalo de confiança de 98%.

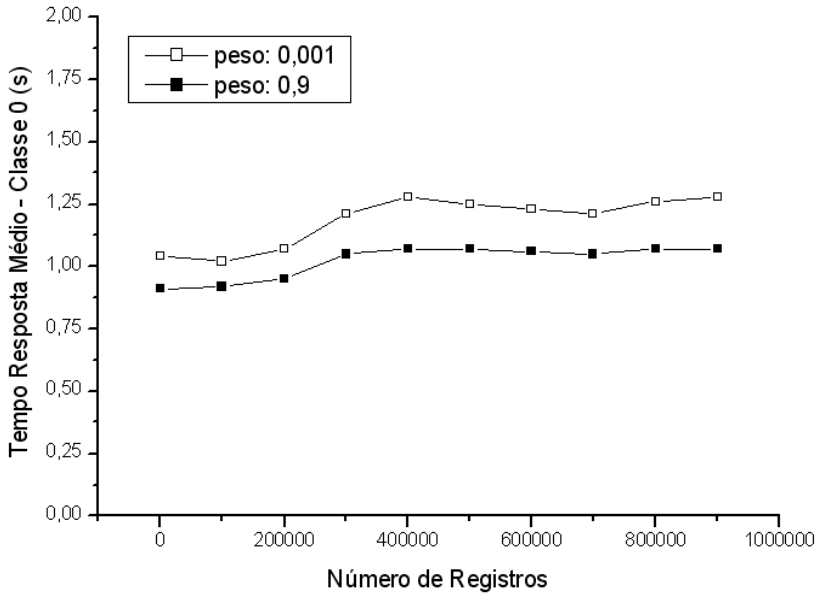


Figura 7.15: Comparação dos pesos da média ponderada

Como se pode constatar, a curva do tempo de resposta médio, para p igual a 0,001, situa-se sempre acima da curva para p igual a 0,9. Isso era esperado, uma vez que aquela configuração não é adequada para casos em que o servidor esteja continuamente submetido a uma carga elevada. O emprego do histórico da utilização do sistema ($p = 0,001$) é mais indicado quando a carga de trabalho é geralmente baixa, apresentando rajadas ocasionais, porém, no *log* utilizado, não foi possível detectar uma faixa de registros com essa característica.

No mecanismo de controle implementado, o valor de p é definido estaticamente, antes de se iniciar a simulação. Seria interessante também variá-lo dinamicamente, no decorrer

do experimento, procurando-se otimizar o comportamento do controle de admissão para cada situação.

Requisições Não Completadas

Nesta seção, faz-se uma avaliação da eficiência do mecanismo proposto sob o ponto de vista do número de requisições não completadas durante a simulação. O experimento utilizado como referência foi descrito na Seção 6.3.3, onde são consideradas duas classes de serviço e o percentual de clientes de alta prioridade varia de 10 até 90%, com incremento de 10 pontos. Na Tabela 6.2, observa-se que o aumento do percentual de requisições na classe de alta prioridade prejudica consideravelmente o tratamento dispensado à classe menos privilegiada, até o ponto em que pouco mais de um terço de suas requisições são concluídas com sucesso. As requisições de alta prioridade, por sua vez, recebem um tratamento adequado, pois o mecanismo de prioridades rigoroso, empregado na diferenciação de serviços, garante-lhes preferência de atendimento em qualquer situação.

O experimento acima foi repetido com as mesmas configurações anteriores, porém desta vez o controle de admissão foi habilitado, a fim de avaliar seu efeito no atendimento dispensado aos clientes, particularmente aos menos privilegiados. O limiar de utilização foi ajustado para 95% e o peso da média ponderada para 0,9. Os resultados a seguir correspondem à média de 10 simulações com um intervalo de confiança de 98%.

Os resultados da Tabela 7.1 mostram que as requisições de baixa prioridade recebem um tratamento bem melhor que no caso anterior, atingindo praticamente 100% de requisições completadas com sucesso. Assim, elimina-se o problema de negação de serviço detectado anteriormente. O tempo de resposta médio dessas requisições menos privilegiadas melhora muito, reduzindo-se em mais de 50%, em média, o que comprova que as requisições adicionais, admitidas no experimento sem controle de admissão, apenas serviam para agravar a situação de sobrecarga do servidor, pois não chegavam, de fato, a ser concluídas. O percentual de requisições não admitidas no sistema situa-se em aproximadamente 10% para ambas as classes de serviço. Portanto, como é desejável, as requisições de alta prioridade continuam recebendo um tratamento diferenciado, mas sem causar uma degradação exagerada do desempenho das tarefas de baixa prioridade, o que demonstra a eficiência do mecanismo de controle de admissão implementado no servidor SWDS.

Considerações sobre o Mecanismo

O mecanismo de controle de admissão discutido, o qual usa como referência a utilização do *cluster*, exponencialmente ponderada, permite o fornecimento de serviços diferenciados de uma maneira mais justa, contemplando todas as classes de serviço e não apenas as de alta prioridade. Essa visão global é fundamental, pois, caso contrário, pode-se afugentar os usuários menos favorecidos: isto seria indesejável não apenas pela perda de clientes,

Baixa/Alta	Baixa Prioridade			Alta Prioridade		
(%)	Adm (%)	Térm (%)	Resp (s)	Adm (%)	Térm (%)	Resp (s)
90/10	90,46	99,98	2,23	90,64	100,00	0,07
80/20	90,42	99,98	2,48	90,69	100,00	0,12
70/30	90,40	99,97	2,79	90,65	100,00	0,17
60/40	90,38	99,97	3,20	90,62	100,00	0,23
50/50	90,21	99,96	3,75	90,74	100,00	0,28
40/60	90,14	99,96	4,51	90,70	100,00	0,33
30/70	90,25	99,94	5,70	90,58	100,00	0,38
20/80	90,29	99,92	8,00	90,52	100,00	0,44
10/90	90,35	99,84	12,99	90,49	100,00	0,65

Tabela 7.1: Percentual de admissões e requisições completadas (`MAXUTIL`)

mas também porque, se a maioria dos clientes presentes no sistema tiverem uma credencial de alta prioridade, fica difícil garantir um tratamento diferenciado entre eles.

As médias obtidas para os tempos de resposta e o tamanho das filas foram sempre menores que nos dois mecanismos de controle apresentados nas Seções 7.3.3 e 7.3.4. Cabe destacar também que o tempo de resposta, de uma maneira geral, mostrou-se mais estável, imune a surtos momentâneos da carga de trabalho, principalmente para valores do peso p próximos de 1. Em todos os experimentos realizados, o limiar de utilização imposto foi rigorosamente respeitado. Os algoritmos de diferenciação de serviços descritos anteriormente mostraram-se ainda mais eficientes, pois puderam operar em um sistema aquém de seu nível de utilização crítica.

Como proposta de trabalho, sugere-se realizar mais testes com outros perfis de carga de trabalho, a fim de verificar a real influência do valor do parâmetro p na diferenciação de serviços. Seria interessante ajustar esse parâmetro dinamicamente, de acordo com o nível de carga do sistema, a fim de realizar o controle de admissão da forma mais adequada a cada situação.

7.4 Considerações Finais

Este capítulo descreveu a especificação e implementação do módulo de Controle de Admissão no modelo do Servidor Web com Diferenciação de Serviços. Foram discutidos sua arquitetura, componentes e métricas utilizadas, bem como os mecanismos de controle implementados.

O primeiro mecanismo procura fazer o controle de admissão limitando as filas dos nós do *cluster*. Embora consiga diminuir o tempo de resposta médio das requisições e o tamanho observado das filas, não é capaz de fornecer garantias de QoS aos clientes, sendo mais indicado para ser usado em conjunto com outros mecanismos.

O segundo mecanismo utiliza *buffers* de tamanho fixo para controlar o número de

clientes de cada classe de serviço presentes no sistema. Os resultados obtidos mostram que, na maioria dos casos, consegue-se manter o tempo de resposta abaixo do limiar especificado, porém pode ser difícil dimensionar os tamanhos dos *buffers* para atingir os requisitos de QoS previamente acertados.

Finalmente, foi implementado um mecanismo de controle de admissão que emprega uma média exponencialmente ponderada da utilização do *cluster*. O mesmo consegue manter a carga do servidor SWDS sempre abaixo do limiar especificado, evitando aceitar no sistema requisições que não poderão ser atendidas. Além disso, as requisições de baixa prioridade recebem um tratamento bem melhor que nos dois mecanismos anteriores, fato evidenciado pela melhoria das médias do tempo de resposta e do percentual de requisições completadas. A variação do peso da média ponderada permite, ainda, ajustar a sensibilidade do servidor SWDS a mudanças no perfil da carga de trabalho.

Em resumo, os experimentos realizados permitem concluir que a utilização de um Controle de Admissão é fundamental para o fornecimento de serviços diferenciados na Web com uma melhor qualidade. Os algoritmos de diferenciação de serviços propostos tornam-se ainda mais eficientes, pois é possível fornecer um serviço com uma característica mais estável e, principalmente, mais justa, evitando-se penalizar ou favorecer demasiadamente uma ou outra classe de clientes.

O capítulo seguinte apresenta as conclusões e principais contribuições desta tese.

Capítulo

8

Conclusão

8.1 Visão Geral

A finalidade deste trabalho foi estudar o fornecimento de serviços diferenciados na Web, em nível de aplicação, particularmente em servidores web. A motivação para o tema surgiu da constatação de que o modelo atual de serviços na Internet e na Web, denominado de *melhor esforço*, trata todas as transações de forma equivalente, sem nenhum tipo de diferenciação ou priorização. Entretanto, diferentes usuários e aplicações possuem necessidades distintas, portanto é interessante atendê-los de forma diferenciada, respeitando suas peculiaridades.

Inicialmente, fez-se uma revisão sobre alguns tópicos pertinentes à área. A infra-estrutura da Web foi discutida no Capítulo 2, destacando-se os principais aspectos que lhe dão suporte, especialmente o protocolo HTTP. Abordou-se também a arquitetura dos servidores web atuais e os resultados mais recentes relativos à caracterização de carga de trabalho, real ou sintética. Concluiu-se com uma discussão da aplicação de técnicas de modelagem e simulação à Web, abordagem empregada neste trabalho. O Capítulo 3 discutiu as limitações do modelo atual de serviços da Internet, motivando o presente tema. Foram descritos em detalhes os conceitos de qualidade de serviço, com destaque para as arquiteturas de serviços integrados e diferenciados. Para finalizar, foram comentadas as pesquisas mais recentes para a incorporação de características de QoS a servidores web.

Esta tese propôs, no Capítulo 4, um modelo para um Servidor Web com Diferenciação de Serviços (SWDS), o qual consiste em uma arquitetura para o fornecimento de serviços diferenciados em servidores web. O modelo integra princípios de QoS, encontra-

dos em nível de rede, às arquiteturas convencionais de servidores. No modelo do SWDS, destacam-se os seguintes módulos: o Classificador, que subdivide as requisições em classes de serviço segundo critérios pré-estabelecidos; o módulo de Controle de Admissão, que visa impedir a sobrecarga do sistema; e o *cluster* de processos ou servidores web, ao qual cabe o atendimento das requisições. O modelo proposto é flexível, podendo ser usado tanto como ponto de partida para a construção de infra-estruturas de servidores web, voltadas ao atendimento diferenciado de clientes, quanto como referência para o desenvolvimento de um programa de servidor web que incorpore funcionalidades de diferenciação de serviços. O modelo não pressupõe nenhuma organização específica, adaptando-se tanto a um sistema distribuído quanto a uma arquitetura mono ou multiprocessada.

Os algoritmos de diferenciação de serviços compreendem outro domínio de estudo do presente trabalho. Esses algoritmos especificam a forma de compartilhamento do *cluster* de servidores web entre as classes de serviço. Eles determinam qual categoria de clientes terá preferência no atendimento, realizando a diferenciação de serviços propriamente dita. Os algoritmos propostos empregam duas abordagens: a primeira delas subdivide o *cluster* em partições de tamanho fixo, alocando-as às diferentes classes de serviço, como descrito no Capítulo 5. Essa abordagem procura isolar requisições de classes diferentes em regiões distintas do *cluster*, tratando as questões de desempenho relativas a cada classe separadamente. A segunda abordagem, detalhada no Capítulo 6, consiste na utilização de prioridades nas filas do *cluster*, as quais irão determinar a ordem de atendimento das requisições. Nesse caso, não há uma separação física entre os clientes presentes no sistema e faz-se a diferenciação de serviços de forma distribuída, em cada nó do *cluster*, o que aumenta a confiabilidade e a escalabilidade do sistema. São definidos dois mecanismos baseados em prioridades, rigoroso e adaptativo. Este último permite dosar quão tendenciosa será a diferenciação de serviços, evitando que as requisições de maior prioridade monopolizem o uso dos recursos do sistema, como pode potencialmente ocorrer no mecanismo rigoroso.

A diferenciação de serviços pode ser seriamente comprometida caso o servidor fique sobrecarregado, pois, nesse caso, não será possível honrar os requisitos de QoS contratados pelos clientes, qualquer que seja sua classe de serviço. Surge, então, a necessidade de controle de sobrecarga, também prevista no modelo proposto. O módulo de Controle de Admissão, discutido no Capítulo 7, é responsável por coletar as informações da carga de trabalho presente no sistema e gerenciar a aceitação de novas requisições pelo servidor. Os mecanismos de controle implementados baseiam-se em diferentes parâmetros para a tomada de decisão: o primeiro deles simplesmente limita o tamanho máximo das filas do *cluster*; o segundo aloca *buffers* de tamanho fixo às classes de serviço, a fim de controlar o número de clientes de cada categoria presentes no servidor; o terceiro toma como referência a utilização do *cluster*, estimada por uma média exponencialmente ponderada, a qual pode ser ajustada para refletir o estado atual ou o histórico da carga do sistema. Os resultados obtidos mostram que o controle da sobrecarga do servidor é fundamental para

a estabilidade do sistema e para a garantia dos requisitos de QoS acertados com os clientes, não se podendo relegar esta tarefa a um segundo plano.

8.2 Principais Resultados e Contribuições

Os principais resultados deste trabalho são relacionados a seguir, destacando-se as contribuições relevantes para a área.

1. **Revisão Bibliográfica** — Foi feita uma revisão crítica da bibliografia existente, a fim de contextualizar o presente trabalho e relatar os mais recentes avanços obtidos na área. Destaca-se especialmente o levantamento feito sobre o fornecimento de serviços diferenciados, em nível de rede e nos servidores web, o qual será publicado como Notas Didáticas do ICMC–USP, uma contribuição que visa sistematizar o conhecimento sobre o assunto.

2. **Geração de Carga de Trabalho** — Foram estudadas diferentes abordagens para a geração de carga de trabalho para a simulação, tanto de forma sintética quanto usando *traces* de acesso a servidores web reais. A alternativa escolhida foi realizar a geração de carga usando *logs* de acesso ao *site* da Copa do Mundo 98. Nesse item, contou-se com a ajuda de outros trabalhos em andamento no LaSDPC.

3. **Modelo do Servidor Web com Diferenciação de Serviços (SWDS)** — Desenvolvimento de um modelo para uma arquitetura de servidor web capaz de fornecer serviços diferenciados aos seus clientes, com garantias de QoS. O modelo do SWDS compreende os seguintes aspectos:

   Classificação: no estágio atual, utiliza-se uma distribuição de probabilidade acumulada para a classificação das requisições que chegam ao servidor, porém o modelo é flexível o bastante para permitir a inclusão de mecanismos mais sofisticados de categorização.

   Controle de Admissão: compreende o módulo desenvolvido e os algoritmos de controle de admissão.

   Mecanismos de Diferenciação de Serviços: foram implementados algoritmos tradicionais de balanceamento de carga, bem como algumas soluções inovadoras para a diferenciação de serviços.

   O modelo do servidor SWDS constitui uma proposta inovadora para o fornecimento de serviços diferenciados em servidores web, sendo uma das principais contribuições desta tese. Ele integra características de QoS às arquiteturas de servidores web tradicionalmente encontradas, destacando-se por sua flexibilidade, a qual é conferida pelos variados mecanismos de controle de admissão e diferenciação de serviços.

Dessa forma, o servidor SWDS consegue adaptar-se a diferentes perfis de carga de trabalho e distribuições dos clientes pelas classes de serviço. O modelo não pressupõe nenhuma plataforma de hardware ou sistema operacional em particular, podendo ser usado tanto para a implantação de infra-estruturas de servidores web distribuídos quanto para o desenvolvimento de um programa de servidor web com funcionalidades de diferenciação de serviços.

4. **Controle de Admissão** — Foram estudadas diversas alternativas para evitar a sobrecarga em servidores web, convencionais ou diferenciados. O controle de admissão de requisições tem uma importância fundamental na diferenciação de serviços, pois evita que o servidor SWDS fique sobrecarregado, na medida em que são recusadas novas solicitações ao serem ultrapassados certos limiares pré-estabelecidos. Este módulo, uma das contribuições deste trabalho, foi projetado de forma flexível, podendo ser ampliado conforme a necessidade, através da introdução de novas métricas e principalmente de novos algoritmos de controle de admissão. Também está prevista a definição de políticas de atendimento, as quais servirão como diretrizes para a aplicação dos mecanismos de controle.

5. **Mecanismos de Controle de Admissão** — Foram implementados três mecanismos de controle de sobrecarga no modelo, os quais determinam o comportamento adotado pelo servidor SWDS frente a novas solicitações de serviço.

   Tamanho das Filas: este mecanismo estabelece um limite superior para o tamanho das filas dos nós do *cluster*. Se o mesmo for atingido, quaisquer novas requisições serão sumariamente recusadas, independentemente de sua classe de serviço. Embora consiga diminuir a sobrecarga no servidor e, por conseqüência, o tempo de atendimento das requisições, não são fornecidas garantias rígidas de QoS aos clientes. Recomenda-se, portanto, sua utilização em conjunto com outros mecanismos de controle.

   Tempo de Resposta: é fixado um valor máximo para o tempo de resposta das requisições, que serve como limiar para o controle de sobrecarga. As requisições de alta prioridade são recusadas somente em situações críticas e as de baixa prioridade são aceitas apenas quando a carga estiver abaixo do limiar especificado. Este mecanismo, uma proposta inovadora, também aloca *buffers* de tamanho fixo para cada classe, a fim de controlar o número de clientes presentes no sistema. Dessa forma, confere uma maior estabilidade ao tempo de resposta, evitando que o mesmo apresente picos devido a súbitos aumentos na carga de trabalho. Entretanto, pode ser difícil dimensionar com exatidão o tamanho dos *buffers* para atingir requisitos de QoS específicos.

   Utilização do Sistema: este mecanismo emprega, de forma inovadora, uma média exponencialmente ponderada da utilização do *cluster* para o controle de admissão. A escolha da utilização como parâmetro de controle permite uma visão

global da carga presente no servidor. O uso de uma média ponderada permite minimizar os efeitos de um congestionamento momentâneo, além de ser possível ajustar sua sensibilidade a mudanças no perfil da carga de trabalho. Este mecanismo consegue manter a utilização do servidor SWDS rigorosamente dentro do limiar especificado, dando um tratamento mais justo a todas as classes de requisições, pois não são aceitas no sistema solicitações que não poderão ser atendidas. A diminuição nos tempos de resposta obtidos e no tamanho das filas reforçam os benefícios do mecanismo.

Conclui-se, portanto, que o controle de admissão de requisições auxilia sobremaneira na diferenciação de serviços, sendo indispensável para que se possa fornecer garantias de QoS aos clientes. Os mecanismos de controle de admissão resumidos acima constituem importantes contribuições desta tese, pois, com a sua aplicação, os algoritmos de diferenciação de serviços mostraram-se ainda mais eficientes e tornou-se possível atender os clientes de uma forma mais estável e justa. Adicionalmente, evita-se penalizar ou favorecer demasiadamente determinadas classes de serviço.

6. **Mecanismos de Diferenciação de Serviços** — Foram implementados três algoritmos de diferenciação de serviços no modelo, os quais correspondem a duas abordagens distintas para o fornecimento de serviços diferenciados: enfileiramento baseado em classes e escalonamento baseado em prioridades.

   Reserva de Recursos: subdivide o *cluster* de servidores web em partições, as quais são associadas às classes de serviço de forma estática. Dessa forma, consegue-se reservar efetivamente uma parcela da capacidade de processamento do servidor SWDS para cada classe de requisições, além de promover o isolamento de desempenho entre as classes, a fim de evitar que uma interfira no comportamento da outra. Contudo, o particionamento estático do *cluster* revelou-se pouco flexível em um ambiente altamente dinâmico como a Web, pois pode sobrecarregar os recursos destinados a uma classe de serviço enquanto os de outra classe estão subutilizados.

   Mecanismo de Prioridades Rigoroso: introduz prioridades nas filas do *cluster*, de modo que o atendimento das requisições siga uma disciplina de prioridades rígida, ou seja, as requisições de alta prioridade serão sempre atendidas em primeiro lugar. Este mecanismo consegue efetivamente fornecer serviços diferenciados aos clientes, em diferentes configurações de carga de trabalho e do servidor e permite alcançar plena utilização dos recursos disponíveis, ao contrário da abordagem anterior. Entretanto, deve-se ter cuidado para que as requisições de alta prioridade não monopolizem o uso dos recursos do sistema, o que pode causar negação de serviço às requisições de prioridade inferior.

   Mecanismo de Prioridades Adaptativo: permite fazer uma sintonia fina do emprego das prioridades, através do uso de um parâmetro de *look-ahead* nas filas

do *cluster*, o qual determina quão rigoroso será o esquema de prioridades empregado. Este algoritmo evita os problemas de negação de serviço evidenciados no mecanismo de prioridades rigoroso e confere capacidade de adaptação ao servidor SWDS. Assim, pode-se ajustá-lo para melhor responder a variações na carga de trabalho oferecida ao sistema, característica bastante atrativa em um ambiente altamente mutável como a Web. Outra vantagem deste mecanismo é que a diferenciação de serviços é realizada de forma distribuída, ao contrário de outras abordagens que dependem de um componente central para esse fim. Dessa forma, transfere-se a carga associada à diferenciação de serviços para cada nó do *cluster*, o que aumenta a confiabilidade e a escalabilidade do sistema.

Os mecanismos de diferenciação de serviços relacionados acima, juntamente com o modelo do servidor SWDS, constituem uma das contribuições centrais desta tese. Dentre eles, destaca-se o Mecanismo de Prioridades Adaptativo, uma proposta inovadora deste trabalho. Esses mecanismos são apenas alguns exemplos, dentre vários possíveis, já que o modelo do servidor SWDS é bastante flexível e pode ser incrementado com muitos outros algoritmos.

Assim, conclui-se que esta tese atingiu plenamente os objetivos a que se propôs, isto é, o estudo do fornecimento de serviços diferenciados na Web, em nível de aplicação, particularmente no contexto de servidores web. O modelo proposto engloba os principais aspectos de qualidade de serviço e os transporta para o ambiente de servidor web, de forma flexível e independente de arquitetura. Os resultados obtidos comprovam que o servidor SWDS é efetivamente capaz de fornecer diferentes níveis de serviço a diferentes classes de clientes. Foram empregados diversos algoritmos de diferenciação de serviços, os quais alcançaram sucesso em variadas configurações do servidor e perfis de carga de trabalho. Finalmente, o Controle de Admissão revelou-se uma ferramenta poderosa para a garantia dos requisitos de QoS previamente acertados com os clientes, tornando a diferenciação de serviços ainda mais eficiente e justa.

Até a presente data, foram geradas as seguintes publicações a partir dos resultados obtidos durante o desenvolvimento desta tese:

- Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. Analysis of Task Scheduling Algorithms in Distributed Web-server Systems. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003)*, p.655–63. Montreal, Canadá, jul., 2003.

- Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. Avaliação de Algoritmos de Escalonamento de Tarefas em Servidores Web Distribuídos. In *XXX Seminário*

Integrado de Hardware e Software (SEMISH 2003), v.1. XXXIII Congresso da SBC, Campinas, SP, ago., 2003.

- Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. Using Adaptive Priority Scheduling for Service Differentiation in QoS-aware Web Servers. In *IEEE International Performance, Computing and Communications Conference, Workshop on End-to-End Service Differentiation (IEEE IPCCC 2004)*. Fênix, Arizona, EUA, abr., 2004.

- Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. Using Adaptive Priority Controls for Service Differentiation in QoS-enabled Web Servers. In *International Conference on Computational Science (ICCS 2004). Lecture Notes on Computer Science*, v.3039, parte IV, p.537–40 (Eds. M. Bubak; G.D.v. Albada; P.M.A. Sloot; J. Dongarra). Cracóvia, Polônia, jun., 2004. (resumo estendido)

Como trabalho adicional durante o período do doutorado, também foram publicadas as seguintes notas didáticas:

- Dos Santos, R. R.; Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. Desenvolvimento de aplicações distribuídas usando a arquitetura CORBA. Notas Didáticas do ICMC–USP, n.59, nov., 2002. 53p.

8.3 Trabalhos Futuros

A proposta para o Servidor Web com Diferenciação de Serviços consiste em uma definição inicial, a qual pode ser ampliada de diversas formas, tanto no que se refere ao modelo quanto aos mecanismos de classificação, controle de admissão e diferenciação de serviços. A seguir, são comentadas algumas sugestões para trabalhos futuros, organizadas por assunto, as quais servem também para demonstrar que o desenvolvimento desta tese abre uma linha de pesquisa ampla, permitindo seu prosseguimento através de trabalhos de iniciação científica, dissertações de mestrado e, possivelmente, outras teses de doutorado.

Modelo do SWDS e Geração de Carga de Trabalho

- Verificar a adequação do modelo do servidor SWDS para o atendimento de transações comerciais na Web, as quais envolvem um alto número de requisições dinâmicas e sessões seguras. A dificuldade, nesse caso, está em encontrar *logs* livremente disponíveis que apresentem essas características. Contudo, acredita-se que o modelo seja genérico o suficiente para lidar com tais questões sem que seja preciso modificá-lo;

- Estudar outras alternativas para a geração de carga de trabalho, possivelmente de forma sintética, a fim que se possa produzir diferentes perfis de carga para a

verificação do modelo, o que nem sempre é possível com o uso de *logs* de acesso a servidores web;

- Realizar um estudo mais aprofundado sobre o impacto da introdução de QoS no desempenho do servidor, analisando-se detalhadamente os atrasos devidos às tarefas de classificação, admissão de requisições e aos próprios algoritmos de diferenciação de serviços;

- Experimentar novas abordagens para a categorização das requisições no módulo Classificador, tomando por base aspectos como urgência de atendimento, tipo de arquivo solicitado, forma de geração de conteúdo, cliente de origem, dentre outros.

Mecanismos de Diferenciação de Serviços

- Desenvolver um algoritmo de Reserva de Recursos Adaptativo, o qual deverá realizar o particionamento do *cluster* de servidores web de forma dinâmica, segundo o perfil da carga de trabalho corrente. Este algoritmo é objeto de estudo de um projeto de Iniciação Científica em andamento no LaSDPC;

- No Mecanismo de Prioridades Adaptativo, atualizar dinamicamente o valor do *look-ahead* a partir de informações obtidas do módulo de Controle de Admissão, a fim de atingir os requisitos de QoS previamente acertados. Uma possível abordagem seria descrever a situação como um problema de otimização com restrições;

- Implementar novos algoritmos de diferenciação de serviços no modelo do SWDS, por exemplo, *Weighted Fair Queueing* e *Earliest Deadline First*. O desafio consiste em transportar esses algoritmos, concebidos segundo uma visão de pacotes de rede, para o nível de aplicação, em que a unidade de processamento são as requisições HTTP. Ambos os algoritmos são objeto de estudo de projetos de Iniciação Científica em andamento no LaSDPC.

Controle de Admissão

- Definir e implementar a funcionalidade de Negociação no módulo de Controle de Admissão. Para tanto, deve-se também simular alguma forma de interação com os clientes para a troca de informações relativas aos requisitos de QoS;

- Definir algumas Políticas de Atendimento a clientes, as quais servirão como diretrizes para a aplicação do controle de admissão;

- Desenvolver novos mecanismos de controle de admissão para o servidor SWDS, em especial mecanismos que façam diferença entre as classes de serviço ao decidir pela aceitação de novas requisições no sistema;

- Introduzir a capacidade de reconhecimento de sessões HTTP no servidor SWDS, característica não contemplada atualmente. Para sua validação, deve-se ser capaz de gerar uma carga de trabalho que utilize sessões.

Interface e Protótipo do SWDS

- Projeto e implementação de uma interface para o servidor SWDS, através da qual o administrador do sistema poderá especificar a parametrização do modelo, os mecanismos de classificação, controle de admissão e diferenciação de serviços utilizados, os requisitos de QoS a serem alcançados, o perfil da carga de trabalho, dentre outros parâmetros. Uma interface desse tipo permitirá o estudo de vários cenários de serviços diferenciados, de forma simples e intuitiva;

- Desenvolvimento de uma linguagem de marcação, provavelmente baseada em XML, para o registro dos eventos da simulação, o qual servirá como base para a criação de um módulo de acompanhamento visual para o servidor SWDS;

- Implementação de um protótipo do servidor SWDS em um sistema distribuído. Para tanto, será preciso modificar algum servidor web de código aberto (por exemplo, o Apache), a fim de dar suporte aos algoritmos de diferenciação de serviços propostos. A função de controle de admissão pode ficar a cargo de *switches* cientes de conteúdo, operando na camada de aplicação e a carga de trabalho pode ser gerada por meio de *benchmarks*, tais como WebStone ou HTTPerf;

- Realizar o mapeamento das funcionalidades do servidor SWDS sobre uma rede que utilize serviços diferenciados, a fim de verificar-se a melhoria alcançada no atendimento das requisições. Esse objetivo implica na geração de uma carga de trabalho com características de *DiffServ*, tornando-se necessária a simulação de detalhes da rede de comunicação.

Referências Bibliográficas

Abdelzaher, T.; Bhatti, N. (1999). Web content adaptation to improve server overload behavior. *Proceedings of the International World Wide Web Conference*, p. 92–103.

Allen, A. O. (1990). *Probability, Statistics and Queueing Theory with Computer Science Applications*. Academic Press, 2 edição.

Almeida, J.; Dabu, M.; Manikutty, A.; Cao, P. (1998). Providing differentiated levels of service in web content hosting. *Proceedings of the 1998 SIGMETRICS Workshop on Internet Server Performance*.

Almeida, V.; Bestavros, A.; Crovella, M.; de Oliveira, A. (1996). Characterizing reference locality in the WWW. *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems*, p. 92–103.

Apache Software Foundation (2002). Apache HTTP Server Project. Disponível em `http://httpd.apache.org`.

Arlitt, M.; Jin, T. (1999). Workload characterization of the 1998 World Cup web site. Relatório Técnico HPL-1999-35, HP Laboratories.

Arlitt, M. F.; Williamson, C. L. (1996). Web server workload characterization: the search for invariants. *Proceedings of ACM SIGMETRICS '96*, p. 126–37.

Aron, M.; Druschel, P.; Zwaenepoel, W. (2000). Cluster reserves: A mechanism for resource management in cluster-based network servers. *Proceedings of ACM SIGMETRICS 2000*, p. 90–101.

Aurrecoechea, C.; Campbell, A. T.; Hauw, L. (1997). A survey of QoS architectures. *Multimedia Systems*, v.6, n.3, p.138–51.

Banga, G.; Druschel, P. (1998). Measuring the capacity of a web server. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.

Banga, G.; Druschel, P.; Mogul, J. C. (1999). Resource containers: A new facility for resource management in server systems. *Operating Systems Design and Implementation*, p. 45–58.

Barford, P.; Bestavros, A.; Bradley, A.; Crovella, M. (1998). Changes in web client access patterns: Characteristics and caching implications. Relatório Técnico 1998-023, Boston University.

Barford, P.; Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. *Proceedings of ACM SIGMETRICS '98*, p. 151–60.

Barford, P.; Crovella, M. E. (1997). An architecture for a WWW workload generator. *Proceedings of the World Wide Web Consortium Workshop on Workload Characterization.*

Baumgartner, F.; Braun, T.; Habegger, P. (1998). Differentiated services: A new approach for quality of service in the Internet. *Proceedings of the IFIP TC-6 Eighth Conference on High Performance Networking (HPN '98).*

Belloum, A.; Kaletas, E.; Afsarmanash, H.; Hertzberger, L. O. (2002). A scalable web server architecture. *World Wide Web Journal: Internet and Web Information Systems*, v.5, n.1, p.5–23.

Berners-Lee, T.; Fielding, R.; Frystyk, H. (1996). *Hypertext Transfer Protocol — HTTP/1.0.* RFC 1945, IETF.

Bhatti, N.; Friedrich, R. (1999). Web server support for tiered services. *IEEE Network*, v.13, n.5, p.64–71.

Blake, S.; Black, D.; Carlson, M.; Davies, E.; Wang, Z.; Weiss, W. (1998). *An Architecture for Differentiated Services.* RFC 2475, IETF.

Borenstein, N. (1993). *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.* RFC 1521, IETF.

Bouch, A.; Kuchinsky, A.; Bhatti, N. (2000). Quality is in the eye of the beholder: Meeting users' requirements for internet quality of service. *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, p. 297–304.

Braden, R.; Clark, D.; Shenker, S. (1994). *Integrated Services in the Internet Architecture.* RFC 1633, IETF.

Braden, R.; Zhang, L.; Berson, S.; Herzog, S.; Jamin, S. (1997). *Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification.* RFC 2205, IETF.

Breslau, L.; Estrin, D.; Fall, K.; Floyd, S.; Heidemann, J.; Kelmy, A.; Huang, P.; McCanne, S.; Varadhan, K.; Xu, Y.; Yu, H. (2000). Advances in network simulation. *IEEE Computer*, v.33, n.5, p.59–67.

Bush, V. (1945). As we may think. *The Atlantic Monthly*, p. 101–08.

Calzarossa, M.; Serazzi, G. (1993). Workload characterization. *Proceedings of the IEEE*, v.81, n.8.

Cardellini, V.; Casalicchio, E.; Colajanni, M. (2001a). A performance study of distributed architectures for the quality of web services. *Proceedings of the IEEE International Conference on System Sciences (HICSS '01)*, p. 3551–60.

Cardellini, V.; Casalicchio, E.; Colajanni, M.; Mambelli, M. (2001b). Web switch support for differentiated services. *ACM Performance Evaluation Review*, v.29, n.2, p.14–19.

Casalicchio, E.; Colajanni, M. (2000). Scalable web cluster with static and dynamic contents. *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '00)*.

Chen, X.; Mohapatra, P. (1999). Providing differentiated services from an Internet server. *Proceedings of the IEEE International Conference on Computer Communications and Networks*, p. 214–217.

Chen, X.; Mohapatra, P.; Chen, H. (2001). An admission control scheme for predictable server response time for web accesses. *Proceedings of the 10th World Wide Web Conference*, p. 545–554.

Comer, D. E. (2000). *Internetworking with TCP/IP: Principles, Protocols and Architecture*. Prentice Hall, 4. edição.

Coulouris, G.; Dollimore, J.; Kindberg, T. (2000). *Distributed Systems: Concepts and Design*. Addison-Wesley, 3. edição.

Crovella, M. E.; Bestavros, A. (1997). Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, v.5, n.6, p.835–46.

Crovella, M. E.; Lipsky, L. (1997). Long-lasting transient conditions in simulations with heavy-tailed workloads. *Proceedings of the 1997 Winter Simulation Conference*, p. 1005–12.

Cubert, R. M.; Fishwick, P. (1995). *Sim++: Version 1.0*. University of Florida. Disponível em `http://www.cise.ufl.edu/~fishwick/simpack/simpack.html` .

Cunha, C. A.; Bestavros, A.; Crovella, M. E. (1995). Characteristics of WWW client-based traces. Relatório Técnico TR-95-010, Boston University.

Demers, A.; Keshav, S.; Shenker, S. (1990). Analysis and simulation of a fair queuing algorithm. *Internetworking Research and Experience*, v.1, n.1, p.3–26.

Dos Santos, R. R.; Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2002). Desenvolvimento de aplicações distribuídas usando a arquitetura CORBA. Notas Didáticas do ICMC–USP, n.59. 53p.

Dovrolis, C.; Ramanathan, P. (1999). A case for relative differentiated services and the proportional differentiation model. *IEEE Network*, v.13, n.5.

Dovrolis, C.; Stiliadis, D. (1999). Relative differentiated services in the Internet: Issues and mechanisms. *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, p. 204–5.

Dowd, K. (1993). *High Performance Computing*. O'Reilly.

Eggert, L.; Heidemann, J. (1999). Application-level differentiated services for web servers. *World Wide Web Journal*, v.3, n.2, p.133–42.

Feldmann, A.; Gilbert, A.; Willinger, W. (1998). Data network as cascades: Investigating the multifractal nature of Internet WAN traffic. *Proceedings of SIGCOMM '98*, p. 42–55.

Ferguson, P.; Huston, G. (1998). *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley.

Ferrari, D.; Serazzi, G.; Zeigner, A. (1983). *Measurement and Tuning of Computer Systems*. Prentice Hall.

Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. (1999). *Hypertext Transfer Protocol — HTTP/1.1*. RFC 2616, IETF.

Floyd, S.; Paxson, V. (2001). Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, v.9, n.4, p.392–403.

Heinanen, J.; Baker, F.; Weiss, W.; Wroclawski, J. (1999). *Assured Forwarding PHB Group*. RFC 2597, IETF.

Hornig, C. (1984). *A Standard for the Transmission of IP Datagrams over Ethernet Networks*. RFC 894, IETF.

Hu, J. C.; Mungee, S.; Schmidt, D. C. (1998). Principles for developing and measuring high-performance web servers over ATM. *Proceedings of INFOCOM '98*.

Hu, J. C.; Pyarali, I.; Schmidt, D. (1997). Measuring the impact of event dispatching and concurrency models on web server performance over high-speed networks. *Proceedings of the II Global Internet Mini-conference*. IEEE GLOBECOM '97.

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*.

Huberman, B. A.; Pirolli, P. L. T.; Pikow, J. E.; Lukose, R. M. (1998). Strong regularities in World Wide Web surfing. *Science*, v.280, n.5360, p.95–97.

Hunt, C. (1997). *TCP/IP Network Administration*. O'Reilly, 2. edição.

IBM (2003). IBM hard disk drives. Disponível em `http://www.storage.ibm.com`.

Internet Traffic Archive (1998). 1998 World Cup web site access logs. Disponível em `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

Iyengar, A. K.; MacNair, E. A.; Squillante, M. S.; Zhang, L. (1998). A general methodology for characterizing access patterns and analyzing web server performance. *Proceedings of MASCOTS '98*.

Iyer, R.; Tewari, V.; Kant, K. (2000). Overload control mechanisms for web servers. *Proceedings of the International Conference on the Performance and QoS of Next Generation Networking*.

Jacobson, V.; Nichols, K.; Poduri, K. (1999). *An Expedited Forwarding PHB*. RFC 2598, IETF.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley.

Kanodia, V.; Knightly, E. W. (2000). Multi-class latency-bounded web services. *Proceedings of the 8th IEEE International Workshop on Quality of Service (IWQoS '00)*, p. 231–239.

Kant, K.; Mohapatra, P. (2000). Scalable Internet servers: Issues and challenges. *ACM SIGMETRICS Performance Evaluation Review*, v.28, n.2, p.5–8.

Kant, K.; Mohapatra, P. (2001). Current research trends in Internet servers. *ACM SIGMETRICS Performance Evaluation Review*, v.29, n.2, p.5–7.

Kant, K.; Won, Y. (1999). Server capacity planning for web traffic workload. *IEEE Transactions on Data and Knowledge Engineering*, v.11, n.5, p.731–47.

kHTTPd (2002). kHTTPd Linux HTTP Accelerator. Disponível em `http://www.fenrus.demon.nl`.

Kilkki, K. (1999). *Differentiated Services for the Internet*. Macmillan Publishing.

Kleinrock, L. (1976). *Queueing Systems: Computer Applications*, v. 2. Wiley-Interscience.

Krishnamurthy, B.; Rexford, J. (1998). Software issues in characterizing web server logs. *Proceedings of the World Wide Web Consortium Workshop on Web Characterization*.

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

Levitt, J. (1997). Measuring web-server capacity. *InformationWeek*.

Li, K.; Jamin, S. (2000). A measurement-based admission-controlled server. *Proceedings of IEEE INFOCOM 2000*.

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

MacDougall, M. H. (1987). *Simulating Computer Systems: Techniques and Tools*. MIT Press.

Magalhães, M. F.; Cardozo, E. (1999). Qualidade de serviço na Internet. Relatório técnico, UNICAMP/FEEC/DCA, Campinas, SP.

Mah, B. (1997). An empirical model of HTTP network traffic. *Proceedings of INFOCOM '97*, v. 2, p. 592–600.

Manley, S.; Seltzer, M. (1997). Web facts and fantasy. *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*.

Menascé, D. A.; Almeida, V. A. F. (1998). *Capacity Planning for Web Performance: Metrics, Models and Methods*. Prentice Hall.

Menascé, D. A.; Almeida, V. A. F. (2003). *Planejamento de Capacidade para Serviços na Web: Métricas, modelos e métodos*. Campus.

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

Mosberger, D.; Jin, T. (1998). httperf: A tool for measuring web server performance. *Proceedings of the First Workshop on Internet Server Performance*, p. 59–67. ACM.

Nichols, K.; Blake, S.; Baker, F.; Black, D. (1998). *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474, IETF.

Odlyzko, A. M. (1999). Paris metro pricing for the internet. *ACM Conference on Electronic Commerce*, p. 140–147.

Orfali, R.; Harkey, D.; Edwards, J. (1996). *The Essential Distributed Objects Survival Guide*. John Wiley.

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

Pandey, R.; Barnes, J. F.; Olsson, R. (1998). Supporting quality of service in HTTP servers. *17th SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, p. 247–56.

Parekh, A. K.; Gallager, R. G. (1993). A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, v.1, n.3, p.344–357.

Paxson, V.; Floyd, S. (1995). Wide-area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, v.3, n.3, p.226–44.

Pinheiro, J. C. (2001). Avaliação de políticas de substituição de objetos em caches na web. Dissertação (Mestrado), USP/ICMC, São Carlos, SP.

Plummer, D. C. (1982). *An Ethernet Address Resolution Protocol*. RFC 826, IETF.

Postel, J. (1980). *User Datagram Protocol*. RFC 768, IETF.

Postel, J. (1981a). *Internet Control Message Protocol*. RFC 792, IETF.

Postel, J. (1981b). *Internet Protocol*. RFC 791, IETF.

Postel, J. (1981c). *Transmission Control Protocol*. RFC 793, IETF.

Rao, G.; Ramamurthy, B. (2001). DiffServer: Application level differentiated services for web servers. *Proceedings of the IEEE International Conference on Communications*, p. 1633–7.

Rosen, E.; Viswanathan, A.; Callon, R. (2001). *Multiprotocol Label Switching Architecture*. RFC 3031, IETF.

Shenker, S.; Partridge, C.; Guerin, R. (1997). *Specification of Guaranteed Quality of Service*. RFC 2212, IETF.

Slothouber, L. P. (1995). A model of web server performance. Disponível em: `http://www.geocities.com/webserverperformance/modelpaper.html`.

SPEC (2001). SPECweb99 benchmark. Disponível em `http://www.spec.org/osg/web99/`.

Stallings, W. (1998). *High-Speed Networks: TCP/IP and ATM Design Principles*. Prentice Hall.

Stallings, W. (2002). *High-Speed Networks and Internets: Performance and Quality of Service*. Prentice Hall, 2. edição.

Stardust (1999a). White Paper — QoS protocols & architectures. Disponível em `http://www.qosforum.com`.

Stardust (1999b). White Paper — The need for QoS. Disponível em `http://www.qosforum.com`.

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

Stevens, W. R.; Fenner, B.; Rudoff, A. (2003). *UNIX Network Programming*, v. 1. Prentice Hall, 3. edição.

Stoika, I.; Zhang, H. (1998). LIRA: An approach for service differentiation in the Internet. *Proceedings of NOSSDAV*.

Tanenbaum, A. S. (2002). *Computer Networks*. Prentice Hall, 4. edição.

Teitelbaum, B.; Hares, S.; Dunn, L.; Neilson, R.; Narayan, R. V.; Reichmeyer, F. (1999). Internet2 qbone: Building a testbed for differentiated services. *IEEE Network*, v.13, n.5, p.8–16.

Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2003a). Analysis of task scheduling algorithms in distributed web-server systems. *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003)*, p. 655–63.

Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2003b). Avaliação de algoritmos de escalonamento de tarefas em servidores web distribuídos. *XXX Seminário Integrado de Hardware e Software (SEMISH 2003)*, v. 1. XXXIII Congresso da SBC, Campinas, SP.

Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2004a). Using adaptive priority controls for service differentiation in QoS-enabled web servers. *Proceedings of the International Conference on Computational Science (ICCS 2004). Lecture Notes on Computer Science*, v. 3039, parte IV, p. 537–40. (resumo estendido).

Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2004b). Using adaptive priority scheduling for service differentiation in QoS-aware web servers. *Proceedings of the IEEE International Performance, Computing and Communications Conference. Workshop on End-to-End Service Differentiation (IEEE IPCCC 2004)*.

Thompson, K.; Miller, G.; Wilder, R. (1997). Wide area Internet traffic patterns and characteristics. *IEEE Network*, v.11, n.6, p.10–23.

Trent, G.; Sake, M. (1995). WebStone: the first generation in HTTP benchmarking. *MTS Silicon Graphics*.

Vasiliou, N. (2000). Overview of Internet QoS and web server QoS. Relatório técnico, Univ. of Western Ontario, Canadá.

Vasiliou, N.; Lutfiyya, H. (2000). Providing a differentiated quality of service in a World Wide Web server. *ACM SIGMETRICS Performance Evaluation Review*, v.28, n.2, p.22–28.

W3C (1999). HTML 4.01 specification. Disponível em `http://www.w3.org/TR/html4`.

W3C (2003). Extensible markup language (XML). Disponível em `http://www.w3.org/XML`.

Weicker, R. (1990). An overview of common benchmarks. *IEEE Computer*, v.23, n.12, p.65–75.

Wroclawski, J. (1997). *Specification of the Controlled-Load Network Element Service.* RFC 2211, IETF.

Xiao, X.; Ni, L. M. (1999). Internet QoS: a big picture. *IEEE Network*, v.13, n.2, p.8–18.

Yeager, N. J.; McGrath, R. E. (1996). *Web Server Technology: the Advanced Guide for World Wide Web Information Providers.* Morgan Kaufmann.

Zhao, W.; Olshefski, D.; Schulzrinne, H. (2000). Internet quality of service: an overview. Relatório Técnico CUCS-003-00, Columbia University.

Zhu, H.; Tang, H.; Yang, T. (2001). Demand-driven service differentiation in cluster-based network servers. *Proceedings of IEEE INFOCOM*, p. 679–688.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least-Effort.* Addison-Wesley.