**Avaliação de desempenho com algoritmos
de escalonamento em clusters de servidores
Web**

Caio Peres Sabo

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 28.03.2006

Assinatura: _____

Avaliação de desempenho com algoritmos de escalonamento em clusters de servidores Web

Caio Peres Sabo

Orientadora: *Prof.^a. Dr.^a. Regina Helena Carlucci Santana*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC–USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área Ciências de Computação e Matemática Computacional

VERSÃO REVISADA

USP – São Carlos
Março de 2006

Dedico este trabalho a minha família e a minha esposa Rita que sempre estiveram do meu lado, dando-me a força e o incentivo necessários para concluí-lo com sucesso.

Agradecimentos

Agradeço primeiramente a Deus.

À Profa. Regina Santana, pela orientação prestada e confiança demonstrada em meu trabalho.

À minha família e à minha esposa Rita, por me apoiarem incondicionalmente e estarem sempre ao meu lado, sendo um alicerce nos momentos difíceis.

Aos amigos Japa, Juliano, Michel, Robertox, Márcio, Nadja, Luciano, Liliam e Rodrigo e Cláudia pelos conselhos, incentivos e por compartilharem conhecimentos e experiências tanto na vida acadêmica quanto na vida pessoal.

Aos amigos e colegas da USP: Valéria, Wiliam, Bárbara, Gustavo, Rogério, Marinho, Menotti, Zorzan, Celsinho e Baiano. E também ao Brunão, Marião, Júlio, Luís, Lucas, Valter e Geraldo.

A todas as pessoas que direta ou indiretamente contribuíram de alguma maneira para que este trabalho pudesse ser realizado.

Resumo

SABO, C.P. *Avaliação de desempenho com algoritmos de escalonamento em clusters de servidores Web*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, São Carlos, 2006. 108 p.

O surgimento de novos serviços e aplicações baseados na *Web* tem provocado um aumento desenfreado na quantidade de usuários da *World Wide Web* que, por sua vez, se torna cada vez mais popular no mundo dos negócios. *Sites* de *e-commerce*, que demandam grande tráfego de requisições, têm adotado sistemas de servidores *Web* distribuídos, como a arquitetura *Web Cluster*. Isso se deve ao fato de enfrentarem frequentemente situações de sobrecarga, durante as quais podem deixar de atender requisições de transação (com grande probabilidade de gerar renda) por conta do aumento na demanda de requisições de navegação (geram renda apenas de forma indireta). A utilização ineficiente de recursos pode comprometer o desempenho do sistema e é nesse contexto que este trabalho se insere. Neste trabalho foi desenvolvido um modelo de Servidor *Web* para *E-Commerce* (SWE-C), validado por meio de um modelo de simulação e uma carga sintética gerada a partir de um modelo desenvolvido com os principais tipos de requisições que caracterizam um *site* de *e-commerce*. Foram realizadas simulações no sistema com diversas combinações de algoritmos de escalonamento e disciplinas de atendimentos para filas, dentre as quais se destaca uma nova disciplina que utiliza um mecanismo de prioridades orientado ao consumo de CPU proposto neste trabalho. O objetivo é aumentar o *throughput* de requisições de transação e melhorar os tempos de resposta em situações de sobrecarga. Uma avaliação de desempenho foi realizada e constatou-se que o mecanismo de prioridades proposto é adequado às necessidades de um *site* de *e-commerce*.

Palavras-chave: servidores Web, Web clusters, E-commerce

Abstract

SABO, C.P. *Performance Evaluation of scheduling algorithms in Web clusters*. Master Thesis — Institute of Science Mathematics and Computing – University of Sao Paulo, Sao Carlos, 2006. 108 p.

The appearance of new services and applications based on the web has causing a wild increase in the amount of users of the World Wide Web that becomes more popular in the business world. E-commerce sites that demand great request traffic has adopted a distributed web servers system, such as the Web Cluster. This occurs because this sites frequently are on overloaded situations, leaving to serve request transactions (with large probability of generate income) due to increasing demand of navigation requests (generates income only indirectly). The low resource utilization can compromise the system performance. This work is inserted in this context. In this master thesis has been developed a Web server model for E-Commerce (WSE-C). It is validated with a simulation model using a synthetic workload characterized with the main types of request identified from e-commerce sites. Several simulations were accomplished in the system, combining the scheduling algorithms, the queue attendances disciplines and the use of a new priority mechanism oriented to the CPU utilization. The aim of this work is increase the request throughput and to obtain a better response time on overload situation. A performance evaluation was conducted and shown that priority mechanisms is adequate to a e-commerce site.

Keywords: Web servers, Web clusters, E-commerce

Lista de Figuras

2.1	Camadas do modelo OSI e TCP/IP	19
2.2	Camadas, protocolos e ligações físicas do modelo TCP/IP	20
2.3	Arquitetura client-servidor da <i>Web</i> (TANENBAUM, 1997)	21
2.4	Formato genérico de uma mensagem HTTP	23
2.5	Exemplo de uma requisição HTTP	24
2.6	Exemplo de uma resposta HTTP	25
3.1	Arquitetura <i>Web Cluster</i> (CARDELLINI et al., 2002)	33
3.2	Arquitetura <i>Web cluster virtual</i> (CARDELLINI et al., 2002)	34
3.3	Arquitetura de um sistema <i>Web</i> distribuído (CARDELLINI et al., 2002)	34
3.4	Método baseado no cliente (CARDELLINI et al., 1999)	37
3.5	Método baseado no DNS (CARDELLINI et al., 1999)	38
3.6	Simple reescrita de pacotes pelo direcionador (CARDELLINI et al., 1999)	40
3.7	Dupla reescrita de pacotes pelo direcionador (CARDELLINI et al., 1999)	40
3.8	Redirecionamento HTTP pelo servidor (CARDELLINI et al., 1999)	41
3.9	Redirecionamento de pacotes pelo servidor (CARDELLINI et al., 2002)	42
4.1	Taxonomia de arquiteturas <i>Web Cluster</i> (CARDELLINI et al., 2002)	45
4.2	Arquitetura two-way em direcionadores de camada 4 (CARDELLINI et al., 2002)	47
4.3	Arquitetura one-way em direcionadores de camada 4 (CARDELLINI et al., 2002)	48
4.4	Arquitetura two-way em direcionadores de camada 7 (CARDELLINI et al., 2002)	51
4.5	Arquitetura one-way em direcionadores de camada 7 (CARDELLINI et al., 2002)	52
4.6	Taxonomia detalhada dos mecanismos de roteamento para <i>Web Clusters</i> (CARDELLINI et al., 2002)	54
5.1	Taxonomia de políticas de escalonamento para <i>Web clusters</i> (CARDELLINI et al., 2002)	60
5.2	Políticas de escalonamento não baseadas em conteúdo (CARDELLINI et al., 2002)	60
5.3	Políticas de escalonamento baseadas em conteúdo (CARDELLINI et al., 2002)	65
6.1	Máquina de estados finitos (SINGHMAR et al., 2004)	72
6.2	Máquina de estados proposta para modelar a carga de trabalho	73
6.3	Cenário utilizado para a realização de experimentos	76
6.4	Exemplo de arquivo de registros do servidor <i>Web</i>	78
7.1	Modelo do Servidor <i>Web</i> para <i>E-Commerce</i> (SWE-C).	82
7.2	Utilização média de CPU (intensidade de carga baixa - 70 req/seg).	88
7.3	Utilização média de CPU (intensidade de carga alta - 100 req/seg).	88
7.4	Utilização média de CPU (intensidade de carga total - 105 req/seg).	89
7.5	Tempo médio de resposta do sistema (intensidade de carga baixa - 70 req/seg).	90

7.6	Requisições atendidas por segundo (intensidade de carga baixa - 70 req/seg).	91
7.7	Tempos de resposta por tipo de requisição (intensidade de carga alta - 100 req/seg).	92
	(a) Combinação RR/FIFO	92
	(b) Combinação RR/LIFO	92
	(c) Combinação RR/LIFO-Pri	92
7.8	Tempos de resposta por tipo de requisição (intensidade de carga alta - 100 req/seg).	93
	(a) Combinação SQF/FIFO	93
	(b) Combinação SQF/LIFO	93
	(c) Combinação SQF/LIFO-Pri	93
7.9	Tempo médio de resposta do sistema (intensidade de carga alta - 100 req/seg).	94
7.10	Throughput e fila de CPU do sistema (intensidade de carga alta - 100 req/seg).	96
	(a) Requisições atendidas por segundo	96
	(b) Tamanho médio das filas de CPU dos hosts servidores	96
7.11	Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).	97
	(a) Combinação RR/FIFO	97
	(b) Combinação SQF/FIFO	97
7.12	Tempo médio de resposta do sistema (intensidade de carga total - 105 req/seg).	98
7.13	Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).	98
	(a) Combinação SQF/LIFO	98
	(b) Combinação SQF/LIFO-Pri	98
7.14	Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).	99
	(a) Combinação RR/LIFO	99
	(b) Combinação RR/LIFO-Pri	99
7.15	Throughput e fila de CPU do sistema (intensidade de carga total - 105 req/seg).	100
	(a) Requisições atendidas por segundo	100
	(b) Tamanho médio das filas de CPU dos hosts servidores	100

Lista de Tabelas

2.1	Métodos do protocolo HTTP/1.1	24
2.2	Classes de códigos de <i>status</i> HTTP	25
6.1	Porcentagens de acesso para interações do tipo <i>Navegar</i>	73
6.2	Porcentagens de acesso para interações do tipo <i>Pedir</i>	74
6.3	Probabilidades de acesso para interações de <i>Navegar</i> e <i>Pedir</i>	74
6.4	Distribuição dos tipos de requisições da carga de trabalho.	74
6.5	Valores obtidos COM o recurso de <i>cache</i> no cliente.	78
6.6	Valores obtidos SEM o recurso de <i>cache</i> no cliente.	79
7.1	Taxas de acerto em <i>cache</i> no cliente.	83
7.2	Prioridades para cada tipo de requisição.	86
7.3	Utilização média de CPU do sistema sob diferentes intensidades de carga.	87

Lista de Siglas e Abreviaturas

CPU	<i>Central Processing Unit</i>
DNS	<i>Domain Name Server</i>
FIFO	<i>First In First Out</i>
GPL	<i>General Public License</i>
HTTP	<i>Hypertext Transfer Protocol</i>
I/O	<i>Input / Output</i>
ISO	<i>International Organization Standardization</i>
IP	<i>Internet Protocol</i>
LAN	<i>Local Address Network</i>
LaSDPC	Laboratório de Sistemas Distribuídos e Programação Concorrente
LIFO	<i>Last In First Out</i>
LIFO-Pri	<i>Last In First Out with Priority</i>
PHP	<i>Hypertext Preprocessor</i>
OSI	<i>Open Systems Interconnection</i>
RAM	<i>Random Access Memory</i>
RR	<i>Round Robin</i>
SQF	<i>Shortest Queue First</i>
SSL	<i>Security Socket Layer</i>
SWE-C	Servidor Web para <i>E-Commerce</i>
TCP	<i>Transmission Control Protocol</i>
WAN	<i>Wide Area Network</i>

Sumário

1	Introdução	13
1.1	Contextualização	13
1.2	Motivação e objetivos	14
1.3	Estrutura	15
2	A Estrutura da Web	17
2.1	Considerações Iniciais	17
2.2	A Internet	18
2.2.1	A Pilha de Protocolos TCP/IP	18
2.2.2	Arquitetura TCP/IP	18
2.3	A World Wide Web	21
2.4	Protocolo HTTP	22
2.4.1	Mensagens HTTP	23
2.5	Considerações Finais	26
3	Balanceamento de Carga na Web	27
3.1	Considerações Iniciais	27
3.2	Balanceamento de Carga em Sistemas Distribuídos	28
3.3	Taxonomia de Arquiteturas de Servidores Web Localmente Distribuídos	30
3.3.1	Web Clusters	32
3.3.2	Web Cluster Virtual	33
3.3.3	Sistemas Web Distribuídos	33
3.4	Técnicas para Balanceamento de Carga em Servidores Web Distribuídos	35
3.4.1	Balanceamento de Carga Baseado no DNS	38
3.4.2	Balanceamento de Carga Baseado em um Direcionador	39
3.4.3	Balanceamento de Carga Baseado no Servidor	41
3.5	Considerações Finais	42
4	Mecanismos de Roteamento para Clusters Web	44
4.1	Considerações Iniciais	44
4.2	Soluções Baseadas em Direcionadores de Camada 4	46
4.2.1	Arquiteturas <i>Two-Way</i>	47
4.2.2	Arquiteturas <i>One-Way</i>	47
4.3	Soluções Baseadas em Direcionadores de Camada 7	50
4.3.1	Arquiteturas <i>Two-Way</i>	50
4.3.2	Arquiteturas <i>One-Way</i>	52
4.4	Comparação de Mecanismos de Roteamento para Clusters Web	53
4.4.1	Mecanismos de Roteamento na Camada 4	53
4.4.2	Mecanismos de Roteamento na Camada 7	55

4.4.3	Roteamento na Camada 4 x Roteamento na Camada 7	55
4.5	Considerações Finais	56
5	Algoritmos e Políticas de Escalonamento para Clusters Web	57
5.1	Considerações Iniciais	57
5.2	Uma Taxonomia de Algoritmos de Escalonamento	58
5.3	Políticas de Escalonamento não Baseadas em Conteúdo	60
5.3.1	Algoritmos Estáticos	60
5.3.2	Políticas Baseadas em Informações do Cliente	61
5.3.3	Políticas Baseadas em Informações do Servidor	62
5.3.4	Políticas Baseadas em Informações do Cliente e do Servidor	63
5.3.5	Considerações Sobre o Escalonamento não Baseado em Conteúdo	63
5.4	Políticas de Escalonamento Baseadas em Conteúdo	64
5.4.1	Políticas Baseadas em Informações do Cliente	66
5.4.2	Políticas Baseadas em Informações do Cliente e do Servidor	67
5.4.3	Considerações Sobre o Escalonamento Baseado em Conteúdo	68
5.5	Considerações Finais	69
6	Carga de Trabalho	70
6.1	Considerações Iniciais	70
6.2	Comércio Eletrônico	71
6.3	Modelo da Carga de Trabalho	71
6.4	Cenário Adotado	74
6.5	Instrumentação para Experimentos	76
6.6	Resultados dos experimentos	77
6.7	Considerações Finais	80
7	Avaliação de Desempenho	81
7.1	Considerações Iniciais	81
7.2	O Modelo Proposto	82
7.3	Metodologia para Experimentos	83
7.3.1	Algoritmos de Balanceamento de Carga	84
7.3.2	Disciplinas para Fila	85
7.3.3	O Mecanismo de Prioridades	85
7.4	Resultados Experimentais	87
7.4.1	Intensidades de Carga	87
7.5	Considerações finais	101
8	Conclusões	103
8.1	Visão Geral	103
8.2	Resultados e Contribuições	103
8.3	Trabalhos Futuros	105
8.3.1	Geração de Carga de Trabalho	105
8.3.2	Modelo de Servidor Web para E-Commerce (SWE-C)	105
	Referências Bibliográficas	106

Introdução

1.1 Contextualização

Atualmente, o crescimento da quantidade de serviços de acesso remoto oferecidos na *Web* vem contribuindo para o aumento da popularidade da *World Wide Web*. De acordo com uma análise recente a população *online* mundial aumentou de 513 milhões em agosto de 2001 para 605 milhões em setembro de 2002 (CYBERATLAS, 2002).

Com a alta demanda de tráfego dos usuários, servidores *Web* podem se tornar um ponto de saturação, ocasionando aumento nos tempos de respostas. Surge então a necessidade da existência de servidores *Web* com capacidade para suportar um grande número de acessos com desempenho aceitável.

Uma primeira solução utilizada para obtenção de escalabilidade em servidores *Web* é realizar uma atualização de hardware. Essa estratégia, denominada *ampliação de recursos em hardware*, simplesmente consiste em adicionar recursos de hardware à máquina, tais como: processador, disco rígido, placas de rede e memória. Considerando a curva que caracteriza o aumento da demanda na *Web*, essa estratégia não resolve o problema de forma efetiva e não tem uma relação custo-benefício satisfatória. Outra opção utilizada para alcançar a escalabilidade em servidores *Web* está relacionada a software, denominada *ampliação de recursos em software* que inclui melhoramentos nos sistemas operacionais dos servidores, construção de servidores *Web* mais eficientes e implementação de algoritmos de escalonamento de requisições.

Entre os inúmeros tipos de aplicações baseados na *Web*, destaca-se as aplicações volta-

das para o comércio de produtos. Aplicações desse gênero, denominadas *sites de e-commerce* são conhecidas pela alta carga de tráfego, onde um único *host* servidor *Web* pode não ser suficiente para atender os requisitos de escalabilidade e desempenho requeridos pelos consumidores, haja visto que o atendimento de múltiplas conexões simultâneas consomem muitos recursos e podem facilmente atingir o limite do hardware (HIRATA, 2002).

Arquiteturas baseadas em servidores *Web* distribuídos se apresentam como alternativas efetivas e vem sendo adotada para suportar altas demandas, principalmente por ser uma solução economicamente mais acessível. Um servidor *Web* distribuído consiste de múltiplos *hosts* conectados por meio de uma LAN (*Local Address Network*, fisicamente próximos) ou WAN (*Wide Area Network*, dispersos geograficamente), que se beneficiam do poder computacional agregado para tratar requisições de clientes (ZHUO et al., 2002).

Arquiteturas de servidores *Web* localmente distribuídos são discutidas no capítulo 3 desta dissertação, dentre as quais se destaca a arquitetura *Web cluster*, composta por um componente escalonador e inúmeros *hosts* servidores *Web*. Essa arquitetura apresenta um desempenho satisfatório quando adotada na infra-estrutura de lojas virtuais de *e-commerce*, sistemas de informação voltados para *Web*, centros de hospedagem *Web* e provedores de aplicações.

1.2 Motivação e objetivos

Algoritmos de escalonamento e disciplinas de atendimento em filas de espera são aplicados em diferentes áreas da computação, possuindo uma infinidade de classificações distintas, das quais muitas se aplicam à *Web*. Na arquitetura *Web cluster*, os algoritmos de escalonamento são aplicados ao componente escalonador com a função de definir entre os *hosts* servidores *Web* qual irá atender uma requisição solicitada. Já as disciplinas de atendimento são aplicadas nos *hosts* servidores *web* do cluster e têm a função de estabelecer a ordem no atendimento de requisições repassadas pelo componente escalonador.

Em *Web clusters*, o algoritmo de escalonamento adotado no componente escalonador e a disciplina de atendimento das filas de cada *host* servidor do cluster têm um grande impacto tanto no desempenho quanto na escalabilidade do sistema em situações de sobrecarga. Porém, é comum encontrar *Web clusters* que utilizem algoritmos e disciplinas ineficientes, principalmente por apresentarem maior facilidade na implementação.

Servidores *Web* para sites de *e-commerce* freqüentemente se deparam com situações de sobrecarga durante as quais podem deixar de atender requisições de transação (que geram renda diretamente) ocasionando prejuízos econômicos. Isso pode ocorrer por conta de um aumento na demanda de requisições de navegação (que geram renda apenas de forma indireta) ou pela

utilização ineficiente de algoritmos de escalonamento e disciplinas de atendimento para filas.

Observa-se a ausência de trabalhos que avaliem o desempenho de sistema baseados na arquitetura *Web Cluster* combinando diferentes algoritmos e disciplinas. Além disso, constata-se a necessidade de implementar melhorias nos algoritmos e disciplinas existentes com a intenção de oferecer tratamento diferenciado para os diversos tipos de requisições de *sites* de *e-commerce* conforme critérios pré-estabelecidos.

O presente trabalho tem como objetivo principal avaliar o desempenho de um *Web cluster* sob diferentes combinações de algoritmos de escalonamento e disciplinas de atendimento para filas implantados em componentes específicos da arquitetura (escalonador e *hosts* servidores do *cluster* respectivamente).

Dentre os objetivos específicos, destacam-se:

- Criação de um modelo de carga de trabalho para *sites* de *e-commerce*;
- Elaboração de um mecanismo de atribuição de prioridades para requisições de *sites* de *e-commerce* baseado no consumo de CPU;
- Desenvolvimento de uma disciplina de atendimento para filas que ofereça tratamento diferenciado para requisições de transação em relação a requisições de navegação em *sites* de *e-commerce*;
- Avaliar o desempenho de um *Web cluster* para *sites* de *e-commerce* com diferentes combinações de algoritmos de escalonamento e disciplinas de atendimento para filas.

O comportamento de um sistema *Web*, baseado na arquitetura *Web Cluster* foi analisado sob diferentes intensidades de carga para comparar a eficiência das combinações de algoritmos/disciplinas em situações de não sobrecarga, comum na maioria do tempo em *sites* de *e-commerce*.

1.3 Estrutura

No Capítulo 2 desta monografia, a estrutura da Internet é apresentada, juntamente com seus protocolos e principais serviços, destacando-se a *Web*, com especial atenção às aplicações que podem ser executadas sobre a mesma e o protocolo HTTP.

Uma visão geral sobre a área de balanceamento de carga no contexto de sistemas distribuídos aplicados à *Web* é discutida no Capítulo 3. Uma taxonomia de arquiteturas de servidores *Web* localmente distribuídos é discutida, apresentando detalhes das três classes de arquiteturas:

Web Cluster, *Web Cluster Virtual* e *Sistema Web Distribuídos*. Algumas técnicas de balanceamento de carga em servidores *Web* escaláveis distribuídos também são discutidas.

No Capítulo 4 são abordadas diversas soluções existentes de mecanismos de roteamento aplicados em *Web clusters*. Detalhes de implementação desses mecanismos tanto em nível de rede (camada 4) quanto em nível de aplicação (camada 7) são apresentados, comparando as vantagens e limitações de cada um.

No Capítulo 5 são apresentadas políticas e algoritmos de escalonamento que podem ser utilizados por *Web clusters* para distribuir a carga entre inúmeros *hosts* servidores de forma compatível ao poder de processamento de cada um. Políticas baseadas em informações do cliente e dos *hosts* servidores são discutidas, bem como algoritmos implementados na camada de rede e de aplicação.

O Capítulo 6 apresenta um modelo de carga de trabalho para *sites de e-commerce* utilizado na geração de uma carga sintética. Detalhes sobre como foram realizados os experimentos também são apresentados. Os resultados obtidos nos experimentos realizados em servidores *Web* reais e as alterações necessárias realizadas no código fonte do servidor Web Apache são descritos em detalhes.

O Capítulo 7 aborda os resultados obtidos após a realização de experimentos. Inicialmente é apresentado um modelo de servidor *Web* para *sites de e-commerce*, validado por meio de simulação. Um mecanismo para estabelecer prioridades entre os diversos tipos de requisições de *sites e-commerce* foi proposto, utilizando como critério o consumo de recursos de CPU. Por fim, são apresentados os principais resultados obtidos e algumas considerações sobre o desempenho de algumas combinações de algoritmos/disciplinas.

Por fim, o Capítulo 8 apresenta as conclusões e principais contribuições desta dissertação, indicando caminhos para futuros trabalhos com temas relacionados.

A Estrutura da Web

2.1 Considerações Iniciais

Uma das grandes aspirações da humanidade é a capacidade de comunicação com seus semelhantes. Com o passar dos anos e a tecnologia então disponível surgiram inúmeras formas de troca de informações. No final do século passado, surgiu a Internet, apontada como uma revolução na história da comunicação entre os seres humanos.

Essa rede de caráter mundial vem evoluindo e transformando-se em um hábito diário na vida das pessoas físicas, instituições privadas e governamentais. A informação passou a ficar disponível de uma maneira simples e intuitiva e a transição entre um computador e outro passou a se dar de forma totalmente transparente para o usuário. Atualmente, a Internet é utilizada para fins não somente acadêmicos, mas também comerciais, fator esse que influencia de forma direta no crescimento explosivo da quantidade de usuários da Internet.

Para atender as necessidades desses usuários, inúmeros serviços foram criados sobre a Internet, dentre os quais se destaca a *World Wide Web*, um serviço que combina hipertexto e multimídia.

Uma visão geral da Internet será apresentada neste capítulo, destacando um de seus principais serviços: a *World Wide Web*. Os protocolos da arquitetura TCP/IP criados como solução para permitir aos computadores compartilharem recursos numa rede também serão abordados, assim como as possíveis aplicações que podem ser executadas sobre o protocolo HTTP.

2.2 A Internet

2.2.1 A Pilha de Protocolos TCP/IP

TCP/IP é o nome dado à pilha de protocolos utilizados pela Internet, desenvolvida pela DARPA (*Defense Advanced Research Project Agency*). Normalmente referenciada como TCP/IP (devido a seus dois protocolos mais importantes: TCP – *Transport Control Protocol* e IP – *Internet Protocol*), essa pilha de protocolos inclui um conjunto de padrões que especifica em detalhes como deve ser realizada a comunicação entre computadores interligados em rede.

Segundo Comer (2000), os principais recursos que diferem os protocolos TCP/IP de outros conjuntos de protocolos são:

- *Independência da tecnologia de redes.* O TCP/IP utiliza a comutação de pacotes para transmitir informações entre os *hosts* de uma rede e é capaz de funcionar sobre inúmeros protocolos das camadas de enlace e física de forma transparente para as aplicações. O TCP/IP define uma unidade para transmissão de dados chamada datagrama e especifica como transmiti-los em um rede de computadores.
- *Interconexão universal.* Em uma rede TCP/IP, um endereço unívoco é associado para cada *host*, permitindo a transmissão de informações entre qualquer ponto da mesma. Cada datagrama transmitido entre dois *hosts* possui o endereço de origem e de destino da informação.
- *Uso de padrões abertos.* Todas as especificações dos protocolos TCP/IP estão disponíveis na Internet gratuitamente através das RFCs (*Request for Comments*), promulgadas pela ITF (*Internet Engineering Task Force*).
- *Confirmações fim-a-fim.* O TCP/IP fornece um mecanismo de controle de fluxos entre os *hosts* de origem e destino final envolvidos na comunicação. Desta forma é possível garantir a ordem correta de pacotes que chegam ao *host* de destino e a identificar erros na transmissão.
- *Protocolos padronizados.* Os protocolos TCP/IP não são padronizados apenas em nível de rede e de transporte, mas também em nível de aplicação (por exemplo, padrões para e-mail e transferência de arquivos), o que facilita o desenvolvimento de novas aplicações.

2.2.2 Arquitetura TCP/IP

A arquitetura TCP/IP, divide as funções do sistema de comunicação em estruturas de camadas, nas quais são definidos um conjunto de protocolos específicos. Nenhuma camada

têm conhecimento de como as outras operam, porém precisam apenas conhecer o formato para entrega e recepção dos dados de cada uma.

O modelo de referência OSI (*Reference Model of Open Systems Interconnection*) da ISO (*International Organization for Standardization*) é a proposta mais conhecida para organização de software de comunicação e propõe uma arquitetura dividida em sete camadas: Aplicação, Apresentação, Sessão, Transporte, Rede, Enlace de dados e Física (TANENBAUM, 1997).

A arquitetura TCP/IP, quando comparada as camadas do modelo OSI, apresenta um total de quatro camadas, dentre as quais duas são formadas a partir da fusão de algumas camadas OSI. A camada de Aplicação (que inclui as camadas de Aplicação, Apresentação e Sessão do modelo OSI); a camada de Transporte (igual a do modelo OSI); a camada Internet (igual a camada de Rede do modelo OSI) e a camada de Acesso à Rede (que inclui as camadas de Enlace de Dados e Física do modelo OSI).

A Figura 2.1 apresenta as camadas da arquitetura TCP/IP ao lado das camadas do modelo de referência OSI. A Figura 2.2 ilustra o modelo TCP/IP com suas camadas, protocolos e ligações físicas. Em seguida cada camada é explicada.

Figura 2.1: Camadas do modelo OSI e TCP/IP

Camada de Acesso à Rede

Camada de abstração de hardware, tem como principal função a interface do modelo TCP/IP com os diversos tipos de redes (X.25, ATM, FDDI, Ethernet, Token Ring, Frame Relay, PPP e SLIP). Pela grande variedade de tecnologias de rede, essa não é normatizada pelo modelo, o que provê a possibilidade de interconexão e interoperação de redes heterogêneas. Essa camada é responsável pelo encapsulamento dos datagramas IP nos *frames* utilizados na rede, assim como a conversão de endereços IPs para o formato de rede.

Figura 2.2: Camadas, protocolos e ligações físicas do modelo TCP/IP

Camada de Internet

Primeira camada do modelo a ser normatizada, a camada de Internet (IP) é responsável pelo endereçamento, roteamento e controle de envio e recepção de datagramas. Nela está definido o protocolo IP (*Internet Protocol*), principal elemento da pilha de protocolos TCP/IP.

Esta camada possui também um protocolo para envio de mensagens de controle e de erro, que é enviado para a camada de Internet de outros computadores, chamado ICMP (*Internet Control Message Protocol*), RFC 792 (POSTEL, 1981). O ICMP tem a função de, por exemplo, checar a existência do endereço de um *host* numa rede.

Camada de Transporte

Essa camada fornece um serviço de comunicação fim-a-fim entre dois *hosts*, ou seja, uma entidade desta camada se comunica apenas com a entidade-par do *host* destinatário. Dois protocolos são definidos nessa camada: o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*). O TCP é orientado à conexão e o UDP não. O acesso das aplicações à camada de transporte é feito através de portas que recebem um número inteiro para cada tipo de aplicação.

Camada de Aplicação

Nessa camada encontram-se os processos que se utilizam dos serviços da camada de Transporte para a entrega dos dados. Nela são compreendidas as aplicações dos usuários e dos diferentes protocolos da camada de aplicação, tais como: TELNET, FTP, HTTP, DNS e outros.

2.3 A World Wide Web

A *World Wide Web* ou simplesmente *Web*, desenvolvida em 1991 no CERN (Conseil European pour la Recherche Nucleaire ou European Center for Nuclear Research), compreende a área hipermídia da Internet, conciliando texto, imagens, sons, animações, etc. e é considerada como um gigantesco sistema hipertexto. Para usuários leigos, a Internet é a *Web*.

Basicamente ela é composta de documentos chamados páginas *Web*, desenvolvidos em sua maioria em HTML (*HyperText Markup Language*) (W3C, 1999), uma linguagem que descreve a estrutura e formatação das páginas; estabelece ligações (*links*) entre documentos e recursos da *Web* e proporciona um padrão que garante a sua portabilidade. Segundo Comer (2000), em 1995 a *Web* se tornou responsável pela maior parte do tráfego da Internet, superando todos os serviços, inclusive aplicações tradicionais como o FTP e o *e-mail*.

Com conteúdo distribuído entre inúmeras máquinas com diferentes arquiteturas, softwares básicos e aplicativos, a *Web* deve permitir que as informações sejam acessadas como se estivessem em um único repositório. Assim a *Web* pode ser considerada um sistema distribuído cuja rede de comunicação é a Internet (Figura 2.3).

Figura 2.3: Arquitetura client-servidor da *Web* (TANENBAUM, 1997)

Os documentos e objetos que circulam na *Web* são fornecidos pelos chamados Servidores *Web* e são acessados pelo usuário através de programas denominados navegadores (ORFALI et al., 1999). Assim, o paradigma cliente-servidor fica caracterizado como a forma de interação empregada pela *Web*, na qual fica evidente a separação entre os softwares servidores *Web* e os softwares utilizados para visualização de documentos.

Páginas *Web* podem ser classificadas como estáticas ou dinâmicas. As páginas estáticas são documentos HTML fisicamente armazenados no sistema de arquivos de um servidor *Web*, disponíveis para leitura do usuário. Páginas dinâmicas são documentos HTML gerados dinamicamente em tempo de execução por um programa ou *script* executado no servidor *Web*.

no momento em que a página é requisitada. Para a comunicação entre navegadores e servidores *Web* é empregado o protocolo HTTP que funciona sobre o protocolo TCP e será discutido posteriormente.

Um terceiro elemento denominado servidor *Proxy* é encontrado na arquitetura cliente-servidor. Esses foram projetados como aplicações *firewalls* para disponibilizar aos administradores de rede um controle mais refinado no acesso de usuários a uma rede privada. Seu objetivo é receber as requisições de um conjunto de clientes, repassá-las ao servidor *Web* apropriado e enviar o objeto requisitado ao cliente (ORFALI et al., 1999).

Dessa forma, assumindo o papel de intermediário, uma outra máquina se faz passar por cliente. Em um servidor *proxy* existem efetivamente dois canais TCP: um entre o cliente e o *proxy* e outro entre o *proxy* e servidor. Ao abrir uma conexão com o servidor *Web* todo o tráfego proveniente do mesmo será redirecionado para o cliente e todo o tráfego proveniente do cliente será redirecionado para o servidor *Web*.

Muitas pessoas confundem o uso de mascaramento IP com o de servidores *proxies*, pois ambos possuem funcionalidades semelhantes. Para se compreender a diferença entre os dois, faz-se necessário ter conhecimento de como é a implementação do protocolo TCP/IP e as inúmeras funções desempenhadas pela camada de transporte e aplicação.

O mascaramento IP comporta-se apenas como um roteador de pacotes, trocando os endereços e portas dos frames dos remetentes ou destinatários. Um servidor *proxy*, remonta a *stream* que o cliente envia antes de retransmiti-la ao servidor, e vice-versa, tendo portanto uma sobrecarga maior. Porém, essa sobrecarga pode ser compensada por outros benefícios.

2.4 Protocolo HTTP

Na arquitetura TCP/IP, o protocolo HTTP (*Hypertext Transfer Protocol*) é um protocolo em nível de aplicação, que trabalha sobre o protocolo TCP e oferece objetividade e rapidez para suportar sistemas de informação distribuídos, cooperativos e de hipermídia. Toda informação que trafega entre cliente e servidor na Internet se faz por meio de uma mensagem HTTP.

Transações HTTP ocorrem por meio de canal de comunicação confiável bidirecional, segundo um mecanismo do tipo requisição/resposta (*request/response*). Por não guardar nenhuma informação em relação ao estado dos clientes, em caso de problemas durante o período de comunicação entre dois *hosts*, é responsabilidade do cliente submeter novamente a transação.

Considerando o mecanismo do tipo requisição/resposta, o protocolo HTTP em suas primeiras versões utiliza o mesmo paradigma do protocolo FTP, por utilizar uma nova conexão TCP para cada transferência de dados. Isto é, um cliente estabelece uma conexão TCP com um servidor *Web*, envia uma mensagem de requisição e após receber resposta encerra a conexão

(COMER, 2000).

A versão 1.1, padronizada em 1999 pela W3C (*World Wide Web Consortium*) utiliza como padrão o esquema de conexões persistentes, na qual, por meio de uma mesma conexão TCP estabelecida entre cliente e servidor é possível realizar inúmeras transações. Outra característica essencial é o *pipelining* de requisições, que permite ao usuário enviar inúmeras requisições sem ter de esperar pela resposta. Isso é bastante útil, por exemplo, na recuperação de objetos de uma página *Web* em ambientes de alta latência no estabelecimento de conexões TCP.

O protocolo HTTP oferece ainda recursos de suporte à cache para melhorar o tempo de resposta das requisições. O navegador *Web* armazena temporariamente os objetos das páginas em disco para que, quando o usuário solicitar um mesma página já visitada, o HTTP permitir ao navegador interrogar o servidor se o conteúdo da página armazenada sofreu algum tipo de alteração desde sua cópia.

Os dados transferidos entre servidor e cliente por meio do protocolo HTTP são representados de acordo com o padrão MIME (*Multi Purpose Internet Mail Extensions*), definido na RFC 1521 (BORENSTEIN et al.,).

2.4.1 Mensagens HTTP

O formato genérico das mensagens de texto trocadas entre cliente e servidor é padronizado pela RFC 822, atualizada pelas RFCs 1327 e 0987.

Figura 2.4: Formato genérico de uma mensagem HTTP

A Figura 2.4 ilustra a composição dessas mensagens, dividida em três partes: linha inicial, cabeçalhos e corpo da mensagem precedido de uma linha em branco (CTLF – Carriage Return Line Feed) para delimitar o término dos cabeçalhos.

Uma mensagem HTTP, quando possui cabeçalhos, pode conter três tipos distintos dos mesmos no formato *nome_do_campo:valor_do_campo*:

- **Cabeçalho geral.** São dados complementares não relacionados com as partes que estão se comunicando nem com o conteúdo sendo transferido. Exemplo: data e versão do MIME;

- **Cabeçalho da requisição/resposta.** Informações adicionais tanto da requisição ou resposta quanto do cliente ou servidor (dependendo de quem estiver enviando);
- **Cabeçalho da entidade.** Informações adicionais sobre a entidade, tais como: navegador utilizado pelo cliente, tipo de servidor *Web* e etc.

Mensagem de Requisição

Uma mensagem de requisição de um cliente a um servidor inclui em sua linha inicial o método HTTP a ser aplicado ao recurso, a localização do objeto no servidor e a versão do protocolo em uso.

```
GET /docs/arq.html HTTP/1.1
Host: www.empresa.com.br
Accept: text/html, image/jpeg
User-Agent: Netscape/4.7
```

Figura 2.5: Exemplo de uma requisição HTTP

Método	Funcionalidade
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 como, 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 loo-back, ao servidor.
CONNECT	Reservado para comunicação com servidores proxy.

Tabela 2.1: Métodos do protocolo HTTP/1.1

A Figura 2.5 apresenta o formato de uma requisição. Nesse exemplo, o cliente solicita através do método GET o arquivo `/docs/arq.html` do servidor *Web* de nome `www.empresa.com.br`

utilizando o protocolo HTTP 1.1. O campo *Accept* do cabeçalho informa que o cliente sabe como tratar o texto em formato HTML (text/html) e imagens em formato JPEG (image/jpeg). O campo *User-Agent* informa ao servidor qual o tipo de navegador *Web* do cliente. Nesse exemplo de requisição o corpo da mensagem não se faz presente.

O protocolo HTTP disponibiliza um conjunto de métodos para indicar o propósito das requisições. Esses métodos funcionam como comandos enviados ao servidor *Web*. A Tabela 2.1 apresenta um resumo dos métodos implementados pelo protocolo HTTP.

Mensagem de Resposta

```
HTTP/1.1 200 OK
Date: Sun, 22 Feb 2004 00:20:26 GMT
Server: NCSA/1.4
Content-Type: text/html
Content-Length: 1194
Last-Modified: Mon, 16 Feb 2004 07:12:47 GMT

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

Figura 2.6: Exemplo de uma resposta HTTP

Classe	Descrição
1xx	É uma resposta temporal; indica que a requisição foi recebida e está atualmente sendo processada.
2xx	Significa que a requisição foi recebida com sucesso, entendida e processada pelo servidor.
3xx	Redireciona o cliente a outra URL. O Cliente deve executar outras ações para que a requisição seja completada (por exemplo, requisitar uma outra URL).
4xx	Indica que uma requisição inválida foi feita pelo cliente. A requisição contém erro de sintaxe ou não pode ser atendida.
5xx	Indica que, embora a requisição do cliente tenha sido válida, o servidor foi incapaz de processá-la. Este erro ocorre enquanto a requisição está sendo processada.

Tabela 2.2: Classes de códigos de *status* HTTP

Uma mensagem de resposta possui como conteúdo em sua linha inicial a versão do protocolo HTTP e um código de *status*. No exemplo da Figura 2.6, a linha inicial informa que o servidor utiliza o protocolo HTTP versão 1.1 e que a solicitação foi completada com

sucesso (código de *status* 200). O campo *Date* (cabeçalho geral) especifica a data e o horário da resposta e o campo *Server* (cabeçalho da resposta) informa o tipo de servidor *Web* (NCSA/1.4). Os campos seguintes pertencem ao cabeçalho da entidade que denotam o tipo de documento (text/html), o seu tamanho em bytes (1194) e quando foi atualizado pela última vez. Por fim, tem-se uma linha em branco (para encerrar o cabeçalho) e o conteúdo do documento.

De acordo com grau de sucesso obtido tentativa de atender uma requisição, o protocolo HTTP oferece uma lista de códigos de *status* para os servidores *Web* informarem aos clientes. Os códigos compostos por três dígitos são agrupados em cinco classes de respostas diferenciadas pelo primeiro dígito. A Tabela 2.2 apresenta as classes de códigos de respostas.

2.5 Considerações Finais

Uma visão geral da *Web* foi apresentada neste capítulo, abordando inicialmente a estrutura da *Web* e a aplicação da pilha de protocolos TCP/IP. Por fim foi estudado o protocolo HTTP e seu impacto no comportamento da *Web*. Apesar das habilidades do HTTP serem responsáveis pelo sucesso da *Web* na Internet, esse protocolo não foi originalmente planejado para ser utilizado como um protocolo de informações comerciais.

O crescente enfoque comercial da Internet chamou a atenção para várias limitações básicas do HTTP. As principais limitações tratam da incapacidade de oferecer dados convencionais de marketing que apresentem por exemplo, o perfil dos visitantes que navegam pela página *Web* ou como os recursos oferecidos estão sendo utilizados.

Com o crescimento da *Web*, nota-se que a rede está sobrecarregada, apresentando sérios problemas de desempenho em relação ao provimento de conteúdo dos servidores *Web*. O próximo capítulo aborda possíveis soluções para atingir escalabilidade em servidores *Web* distribuídos bem como técnicas para balanceamento de carga.

Balanceamento de Carga na Web

3.1 Considerações Iniciais

A popularização da *World Wide Web* e aumento de vendas online tem causado um aumento na taxa de requisições em *sites* de venda de produtos. Para evitar rigorosos congestionamentos e possíveis prejuízos, administradores desses *sites de e-commerce* constantemente se deparam com a necessidade de melhorar a capacidade de seus servidores *Web* para atingir a demanda dos usuários.

Nos últimos anos, a grande quantidade de trabalhos publicados indica a preocupação da comunidade com possíveis soluções de aspectos relacionados ao aumento do desempenho global da *Web*. Em *sites* com altos índices de acesso, um único *host* servidor *Web* pode não ser suficiente para atender os requisitos de escalabilidade e desempenho requeridos, uma vez que o atendimento de múltiplas conexões simultâneas consomem muitos recursos e podem facilmente atingir o limite de hardware. Uma solução para tal problema é organizar um sistema *Web* distribuído composto por diversos *hosts*, no qual a carga deve ser distribuída para melhorar o desempenho do sistema. Diferentes propostas para atingir escalabilidade com servidores *Web*, tanto geograficamente quanto localmente distribuídos, são discutidas em Cardellini et al. (2002), por exemplo, o *Web cluster*.

A arquitetura *Web cluster* consiste de um conjunto de *hosts* servidores *Web* fisicamente próximos, interconectados através de uma rede de alta velocidade e que se beneficiam do poder computacional agregado para tratar requisições HTTP de clientes (ZHUO et al., 2002). Em ge-

ral, cada *host* servidor possui seu próprio disco rígido e um sistema operacional completo. Uma análise mais detalhada das arquiteturas de *Web clusters* é discutida em Schroeder e Harchol-Balter (2002).

No caso específico do *Web cluster*, como qualquer sistema distribuído, um componente centralizado deve distribuir as requisições HTTP dos clientes entre os *hosts* servidores. Esse componente, referenciado neste trabalho como direcionador, pode ser implementado como um dispositivo de *hardware* dedicado conectado à rede (menos flexível e de alto custo) ou como um *software* escalonador implementado em um dos *hosts*. Um *Web cluster* deve exportar para a Internet uma única imagem, ou seja, um único endereço IP. Assim, o DNS principal do *cluster* executa o mapeamento de um para um, traduzindo o nome da página Web no endereço IP do direcionador, fazendo com que a distribuição seja totalmente transparente aos usuários (CARDELLINI et al., 2002).

As informações que o direcionador utiliza para decisões de roteamento se baseiam nas camadas de rede e de aplicação (camadas 4 e 7 respectivamente) conforme o modelo de referência ISO/OSI e servem para classificar o tipo de direcionador. A escolha do mecanismo de roteamento a ser utilizado possui um grande impacto nos algoritmos de escalonamento, uma vez que as informações disponíveis para decisão são bastante distintas. Um *Web cluster* integra seus *hosts* servidores para que todas as requisições HTTP dos clientes sejam distribuídas de forma equilibrada. Um balanceamento de carga entre servidores faz parte de uma solução abrangente em uma explosiva e crescente utilização da *World Wide Web*, provendo um aumento na capacidade do sistema e melhorando o desempenho.

Este capítulo discute o conceito de balanceamento de carga no contexto de sistemas distribuídos aplicados à Web. Existem diversos trabalhos concluídos sobre escalonamento e balanceamento de carga em muitas outras áreas em que muitas das técnicas se aplicam na Web. Algumas técnicas de balanceamento de carga em servidores Web distribuídos também são discutidas.

3.2 Balanceamento de Carga em Sistemas Distribuídos

O objetivo principal dos sistemas computacionais distribuídos tem convergido para a busca de alto desempenho. Para que esse objetivo seja alcançado, não basta apenas existirem sistemas com grande número de entidades de processamento. Surge então a necessidade de garantir que todas as entidades de processamento pertencentes ao sistema sejam bem aproveitadas. O bom aproveitamento do sistema pode ser obtido pela distribuição das tarefas entre as entidades existentes. Entretanto, para obter o melhor aproveitamento desse sistema, a dis-

tribuição deve garantir que todas as unidades de processamento tenham uma carga compatível ao seu respectivo poder de processamento, de modo que seja possível evitar que algumas se tornem ociosas, enquanto outras estão sobrecarregadas. A distribuição adequada de tarefas nas diversas unidades de processamento é fundamental para o melhor desempenho do sistema. A identificação dessa distribuição ideal de tarefas caracteriza o problema de balanceamento de carga.

As nomenclaturas e terminologias são confusas, de modo que balanceamento de carga e escalonamento podem ser confundidos. Para Souza (2000), mesmo que implícita, a associação direta desses termos não é correta porque o balanceamento das cargas de uma plataforma é um dos possíveis objetivos de um algoritmo de escalonamento, e não o único.

Mullender (1993) exemplifica esse problema afirmando: “é possível, através de uma alocação prudente, otimizar o desempenho do sistema como um todo. Esse processo é conhecido como balanceamento de carga...”. Uma alocação prudente pode ser feita com vários objetivos, não apenas o de balancear as cargas de uma plataforma.

Ao contrário de Mullender (1993), Bestavros (1997) defende que o balanceamento de cargas não deve nem ser sinônimo e nem ser o único objetivo do escalonamento. Ele propõe uma nova política de escalonamento para um sistema distribuído de tempo real, cujo objetivo é garantir que os processos forneçam as respostas esperadas dentro dos prazos máximos preestabelecidos. Nesse caso, o balanceamento das cargas em seu trabalho é irrelevante. Para Branco (2000), escalonar de modo a obter um balanceamento de cargas implica, não somente em distribuir a carga através dos recursos disponíveis no sistema, mas também garantir que a carga esteja equilibrada em todos esses recursos.

A principal função do escalonamento, que visa ao balanceamento de carga, é sugerir decisões que possibilitam um aumento de desempenho. A escolha do objetivo do escalonamento depende basicamente do usuário e do tipo de aplicação. Uma estratégia de balanceamento de carga é composta basicamente de: medidas de carga (que caracterizam o nível de carregamento de cada elemento do sistema) e políticas de decisões (que determinam tanto a condição atual quanto o destino de cada tarefa). Políticas de balanceamento de carga fornecem regras que permitem utilizar medidas de carga para a tomada de decisão no escalonamento global (BRANCO, 2000).

Considerando o balanceamento de cargas, um algoritmo de escalonamento deve ser capaz de controlar o efeito de diferentes fatores de desequilíbrio que ocorrem em diversas aplicações e arquiteturas. Dentre esses fatores podem ser destacados: a heterogeneidade tanto da arquitetura quanto do sistema operacional presente em um sistema computacional distribuído; o

desconhecimento da quantidade de processamento envolvida em cada tarefa; a criação dinâmica de tarefas; além da heterogeneidade e da variação da carga externa à aplicação nos diversos processadores de um ambiente não dedicado, adaptativo e não uniforme (instável) (CASAVANT; KUHL, 1988).

De uma forma geral, existem dois tipos de balanceamento de carga: estático e dinâmico. No balanceamento estático, a distribuição de carga é feita de acordo com a demanda de carga das entidades de processamento envolvidas no problema, mas somente na inicialização do problema. Assim, esse tipo de balanceamento ajuda a distribuir a carga uniformemente, mas apenas nos primeiros instantes da aplicação. Se o domínio da aplicação modificar, esse balanceamento pode se tornar impreciso e até mesmo prejudicar o desempenho da aplicação. Por outro lado, no balanceamento dinâmico isso não acontece, pois sua principal característica é a distribuição de carga de acordo com a demanda, mas em determinados instantes da aplicação. Dessa forma, entidades de processamento que poderiam estar ociosas com uma eventual mudança de domínio podem ser reaproveitadas e entidades que estariam sobrecarregadas teriam seu trabalho reduzido.

Atualmente, servidores *Web* distribuídos com altas taxas de acesso, pertencem a uma classe de problemas que demandam o uso de um mecanismo de balanceamento de carga, cujo objetivo é propiciar escalabilidade, no qual todas as requisições HTTP de clientes são distribuídas entre um conjunto de *hosts* servidores chamados *Web cluster*.

Embora não exista uma relação direta entre os conceitos de escalabilidade e alta disponibilidade, na prática ambos usualmente estão ligados, uma vez que o investimento destinado a sistemas redundantes para alta disponibilidade não pode ser justificado se o equipamento adicional estiver ocioso ou apenas duplicando o trabalho executado nos *hosts* servidores primários. Um conjunto de *hosts* servidores não pode ser otimizado para as duas situações (escalabilidade e alta disponibilidade), por exemplo, se um sistema é configurado para uma determinada carga com N *hosts* servidores, e sofre uma falha, a carga precisa ser agora dividida entre $N - 1$ *hosts* servidores. Isto claramente afeta o desempenho total e, portanto, não pode ocultar devidamente a falha, pois a degradação do desempenho pode ser observada por um cliente externo.

3.3 Taxonomia de Arquiteturas de Servidores Web Localmente Distribuídos

Cardellini et al. (2002) propõem uma taxonomia de arquiteturas de servidores *Web* localmente distribuídos e destacam que a premissa básica desta é manter a compatibilidade de todas as soluções existentes com os protocolos padrões da *Web*, de forma que qualquer arquitetura

tura, algoritmo ou mecanismo possa ser adaptado sem qualquer limitação. Dado um conjunto de *hosts* servidores que hospedam uma página *Web* em um mesmo local, pode-se identificar três principais classes de arquiteturas:

- *Web cluster*, no qual os *hosts* servidores omitem seus endereços IPs para os clientes, ou seja, o endereço visível aos clientes é correspondente a um IP virtual (VIP) atribuído a um dispositivo localizado defronte a camada de servidores (direcionador);
- *Web cluster Virtual*, no qual o endereço visível para o cliente é apenas o endereço IP virtual, assim como em um *Web cluster*. A diferença para a arquitetura anteriormente mencionada é que o endereço IP virtual não é atribuído para um único dispositivo que se localize defronte aos *hosts* servidores e sim compartilhado entre eles;
- *Sistemas Web distribuídos*, no qual cada *host* servidor possui um endereço IP visível para as aplicações clientes.

Permitir ou não que os endereços dos *hosts* servidores sejam visíveis aos clientes é uma característica importante, pois implica em soluções diferentes para algoritmos e mecanismos de distribuição de carga entre os *hosts* servidores. Em particular, a arquitetura de um sistema *Web* distribuído é uma solução mais antiga, na qual a decisão de roteamento das requisições é realizada pelo DNS (*Domain Name Server*). Segundo Cardellini et al. (2002), a arquitetura *Web cluster* é uma solução mais recente, onde o roteamento das requisições é realizado por uma entidade interna ao *Web cluster*.

Para páginas *Web* onde os *hosts* servidores estão localizados em um mesmo ambiente, Cardellini et al. (2002) recomendam adotar uma arquitetura *Web cluster* ao invés de um sistema *Web* distribuído, por fornecer um controle mais refinado sobre as requisições HTTP dos clientes, além de uma melhor disponibilidade e segurança, considerados recursos fundamentais para serviços baseados na *Web*. Por outro lado, a arquitetura de um sistema *Web* distribuído com seus endereços IPs visíveis são adequados para implementar páginas *Web* onde os *hosts* servidores estão geograficamente dispersos. Os principais componentes de um sistema *Web* de múltiplos *hosts* incluem: um mecanismo de roteamento de requisições para distribuir as requisições HTTP das aplicações clientes para um único *host* servidor do cluster, um algoritmo de escalonamento para selecionar o *host* servidor que melhor atenderá uma requisição HTTP e um direcionador para executar o algoritmo de escalonamento e suportar o mecanismo de roteamento.

3.3.1 Web Clusters

Um *Web cluster* é um conjunto de *hosts* servidores *Web* distribuídos fisicamente próximos, interconectados por uma rede de alta velocidade e que apresentam um único endereço IP visível para as aplicações clientes. Cada *host* servidor do *cluster* apresenta em geral, seu próprio disco rígido e um sistema operacional completo. Os *hosts* do cluster trabalham coletivamente como um simples recurso computacional.

Sistemas de processamento paralelo em massa, no qual cada *host* satisfaz as características anteriores podem se assemelhar a um *Web cluster*. Na literatura, algumas terminologias são usadas para referenciar a arquitetura *Web cluster*. Um termo comum é *Web farm*, cujo significado abrange os *hosts* servidores, as aplicações e dados de uma única página *Web* (DEVLIN et al., 1999). No decorrer deste trabalho é utilizado o termo *Web cluster*.

Embora um *Web cluster* possa ser composto por dezenas de *hosts*, somente um nome e um endereço IP é publicado na Internet. Assim, o DNS primário do *Web cluster* executa o mapeamento de um para um, traduzindo o nome do domínio para o endereço IP virtual de uma entidade, defronte ao *cluster*, dedicada a distribuir as requisições HTTP entre os servidores. Essa entidade, referenciada neste trabalho como **direcionador**, que interliga os *hosts* servidores do cluster com a Internet, tem a responsabilidade de omitir dos usuários toda a arquitetura distribuída. Esse direcionador recebe todos os pacotes de dados que os clientes enviam para o endereço IP virtual do *Web cluster* e os distribuem para um dos *hosts* servidores, proporcionando um minucioso controle das requisições solicitadas pelos clientes.

Uma visão geral de um *Web cluster* básico, do direcionador e dos N *hosts* servidores é ilustrada na Figura 3.1. Nota-se que as linhas de respostas dos *hosts* servidores para os clientes não aparecem na figura, pois duas principais alternativas são discutidas detalhadamente no próximo capítulo. Para Cardellini et al. (2002), o DNS primário do *cluster* não está incluído na área que restringe o *Web cluster*, pois ele não possui nenhuma função no roteamento das requisições HTTP dos clientes.

O direcionador também pode ser implementado em um dispositivo de hardware de propósito específico conectado à rede (também chamado de circuito integrado para aplicações específicas) ou um módulo de software a ser executado em um sistema operacional tanto de propósito geral quanto de propósito específico. Essa definição não implica que o direcionador seja um dispositivo de hardware que simplesmente repasse quadros Ethernet.

Figura 3.1: Arquitetura Web Cluster (CARDELLINI et al., 2002)

3.3.2 Web Cluster Virtual

Segundo Cardellini et al. (2002), a arquitetura de um *Web cluster virtual* é totalmente baseada no projeto de um sistema distribuído que não usa uma entidade de direcionamento. Similar a arquitetura do *Web cluster*, um *Web cluster virtual* apresenta um simples sistema de imagens de servidores Web para a Internet através do uso de um único endereço IP virtual.

A principal diferença é que esse endereço não está associado a uma entidade centralizada que recebe todas as requisições HTTP e as distribui para um dos *hosts* servidores do cluster. Em um *Web cluster virtual* o endereço IP virtual é compartilhado por todos os *hosts* servidores de um cluster no qual cada um recebe todos os pacotes de requisições solicitadas pelos clientes, as filtram e decidem entre aceitar ou descartar. Apenas um *host* aceita a requisição solicitada e a responde. Esse sistema elimina o simples ponto de saturação representado pelo direcionador. Uma visão geral da arquitetura de um *Web cluster virtual* é mostrada na Figura 3.2.

3.3.3 Sistemas Web Distribuídos

Um sistema Web distribuído consiste de *hosts* servidores localmente distribuídos, no qual múltiplos endereços IPs podem estar visíveis para as aplicações clientes da Internet. Uma visão geral da arquitetura de um sistema Web distribuído é mostrada na Figura 3.3 (nessa arquitetura o DNS primário é incluído na área do sistema Web distribuído, pois esse componente exerce função no roteamento das requisições solicitadas pelos clientes).

Figura 3.2: Arquitetura *Web cluster virtual* (CARDELLINI et al., 2002) Figura 3.3: Arquitetura de um sistema *Web* distribuído (CARDELLINI et al., 2002)

Diferentemente da arquitetura *Web cluster* e similar a *Web cluster virtual*, um sistema *Web* distribuído não possui a presença de um direcionador. A tarefa de direcionar as requisições dos clientes é atribuída para outra entidade durante a resolução do endereço URL da página *Web* através de um mecanismo de DNS. Em alguns sistemas existe ainda a presença de um segundo nível de roteamento que é tipicamente executado por meio de mecanismos de roteamento ativados pelo *host* servidor que não pode executar uma requisição recebida.

3.4 Técnicas para Balanceamento de Carga em Servidores Web Distribuídos

A replicação de informações por meio da arquitetura de espelhamento de servidores é utilizada para aumentar a escalabilidade de servidores *Web*. Essa arquitetura oferece uma lista de URL independentes que deve ser selecionada manualmente pelo usuário, sem restrição alguma. Tal solução apresenta uma série de desvantagens, dentre as quais se destacam a falta de controle da distribuição das requisições dos clientes entre os *hosts* servidores *Web* e a não transparência da arquitetura para o usuário.

Uma solução mais promissora é confiar em uma arquitetura distribuída que faça o roteamento de requisições entre vários *hosts* servidores de forma transparente ao usuário. Esse método potencialmente pode melhorar o desempenho, além de fornecer escalabilidade e disponibilidade para o servidor *Web*. Entretanto, uma série de exigências deve ser tratada para fazer um cluster de servidores funcionar eficientemente como um simples servidor compatível com os padrões existentes na *Web*, como o protocolo HTTP e os navegadores *Web* (CARDELLINI et al., 1999).

Nesta seção são discutidas diversas soluções para roteamento de requisições entre inúmeros *hosts* servidores *Web*, ressaltando a efetividade e as principais características de cada uma.

Servidores Web Distribuídos

Para melhor compreensão são esclarecidos alguns nomes da terminologia apresentada na literatura. O termo **usuário** será referenciado como qualquer pessoa que estiver acessando informações na *World Wide Web*. Um **cliente** é considerado um programa, tipicamente um navegador *Web* que estabelece conexões na Internet para satisfazer as requisições dos **usuários**. Clientes estão conectados na Internet através de *gateways*, desta forma são referenciados como domínios todos os subdomínios de rede posteriores aos *gateways* locais (CARDELLINI et al., 1999). Um servidor *Web* tem o propósito de armazenar informações e servir as requisições dos

clientes. Para requisitar um documento para um servidor *Web*, cada cliente precisa mapear o URL da página *Web* em um endereço IP. O cliente obtém o endereço IP de um servidor *Web* por meio de envio de uma requisição para um servidor de nomes (DNS), responsável pelo processo de resolução de endereço. Entretanto, para reduzir o tráfego obrigatório da resolução de endereços, várias entidades (por exemplo, servidores de domínios intermediários, *gateways* locais e navegadores) podem armazenar em *cache* por um determinado período de tempo algumas URLs já resolvidas.

Um servidor *Web* distribuído é qualquer arquitetura que consista de múltiplos *hosts* servidores com algum mecanismo para distribuir as requisições HTTP dos clientes entre vários *hosts*. Cada *host* servidor do cluster pode responder qualquer requisição de clientes. Segundo Cardellini et al. (1999), essencialmente existem dois mecanismos para implementar a distribuição de informação entre vários *hosts* servidores: replicando a estrutura de diretórios e arquivos no disco local de cada *host* ou usando um sistema de arquivos distribuído.

Toda a estrutura de um servidor *Web* distribuído deve ser compreendida na *Web* como um simples *host* para os usuários, sem o consentimento dos nomes e localidades dos servidores replicados. Ao contrário de um servidor *Web* distribuído, a arquitetura de espelhamento de servidores é visível pelos usuários, violando a condição essencial de transparência.

As arquiteturas de servidores *Web* distribuídos são classificadas conforme a entidade que distribui as requisições que chegam aos *hosts* servidores. Cardellini et al. (1999) apresenta quatro classes de métodos, o primeiro dos quais requer modificações de software no lado do cliente e os três restantes afetam um ou mais componentes do *Web cluster*.

- Método baseado no cliente;
- Método baseado no DNS;
- Método baseado em um direcionador (nível de rede);
- Método baseado no servidor.

Todos os métodos citados anteriormente satisfazem a condição essencial de transparência. Outros fatores considerados incluem escalabilidade, balanceamento de carga, disponibilidade e aplicabilidade para os padrões atuais da *Web*.

Balanceamento de Carga Baseado no Cliente

Existem duas abordagens de como colocar o mecanismo de seleção de servidores no lado do cliente satisfazendo o requisito de transparência: por meio dos próprios clientes (na-

veadores) ou de servidores *proxies* (também visto como um cliente). A Figura 3.4 mostra os passos pelo qual o cliente pode selecionar um *host* servidor para requisições. A seguir são apresentadas três abordagens de balanceamento de carga no cliente.

Figura 3.4: Método baseado no cliente (CARDELLINI et al., 1999)

Método da Netscape: quando um cliente, por meio do navegador Netscape acessa a página Web da Netscape Communication (localizado em <http://www.netscape.com>), o navegador seleciona aleatoriamente um número i entre um e o número de servidores do *cluster* da Netscape. O usuário acessa então o *host* `wwwi.netscape.com`. O método da Netscape se restringe a páginas Web que têm o privilégio de possuir o seu próprio software navegador, o que o torna muito limitado. Além disso, o esquema de seleção Aleatório, utilizado para distribuir as requisições, não garante disponibilidade e nem balanceamento de carga entre os *hosts* servidores.

Clientes Inteligentes: Essa solução, proposta por Yoshikawa et al. (1997), visa a migração de algumas funcionalidades do servidor para a máquina do cliente, ao contrário dos tradicionais métodos nos quais o cliente não é envolvido. O roteamento de uma requisição do cliente é realizado por meio de um *applet* Java, que é executado no cliente cada vez que o usuário faz uma requisição a um servidor Web distribuído. Como o *applet* tem conhecimento dos endereços IPs de todos os *hosts* servidores do Web *cluster* ele pode enviar mensagens para cada um deles coletando informações de estados dos *hosts* e tempos de resposta da rede. Essas informações são analisadas pelo *applet* para selecionar o *host* adequado. Requisições dos clientes que passam pelo *applet* são, então, repassadas para o servidor selecionado. Para Cardellini et al. (1999), a maior desvantagem desse método é o aumento no tráfego da rede devido à troca de mensagens para monitorar a carga de cada *host* servidor e dos tempos de resposta da rede. Além disso, apesar desse método oferecer escalabilidade e disponibilidade, ele necessita que alterações sejam realizadas no cliente.

Proxies: Do ponto de vista do Web *cluster*, um servidor *proxy* é similar a um cliente. A implementação por meio de servidores *proxies* como clientes pode conservar a rota de endereços URL replicados e rotear requisições de clientes para o servidor mais adequado. A aplicabilidade

dessa técnica requer algumas modificações em componentes da Internet que normalmente não são controlados pelas mesmas instituições que gerenciam sistemas de servidores *Web*, portanto essa abordagem não será detalhada.

3.4.1 Balanceamento de Carga Baseado no DNS

Segundo Cardellini et al. (1999), em uma primeira solução desenvolvida no lado do *Web cluster*, a responsabilidade de distribuir as requisições entre os servidores é atribuída ao DNS do *cluster*, ou seja, ao servidor de DNS responsável pelo domínio do *cluster*. Por meio de um processo de tradução entre os nomes simbólicos (URL) e endereços IP, o DNS do *cluster* pode selecionar qualquer *host* que compõe o *cluster*.

Figura 3.5: Método baseado no DNS (CARDELLINI et al., 1999)

Além de fornecer o endereço IP de um *host* na Internet, o DNS também especifica o período de validade, conhecido como **tempo de vida** (*Time To Live* – TTL), para manter armazenado em memória *cache* endereços simbólicos mapeados para endereço IP. Após o término do TTL, as requisições de mapeamento de endereços dos clientes enviadas ao DNS local são repassadas para o DNS do *cluster* para a atribuição do *host* servidor que atenderá a requisição do cliente. Caso contrário, o mapeamento de endereços requisitados são tratados por servidores DNS intermediários. Essas duas formas alternativas de resolução de endereços URL são mostradas na Figura 3.5. Se um DNS intermediário possuir um mapeamento válido para o URL do *Web cluster*, ele pode resolver o mapeamento do endereço requisitado sem repassá-lo para outro DNS intermediário. Caso contrário, o endereço requisitado chega ao DNS do *cluster*, que seleciona o endereço IP de um *host* servidor *Web* e um período de validade para mantê-lo em *cache*. O mapeamento do URL para um endereço IP e o período de validade são repassados a todos os DNS intermediários ao longo da rota até que chegue no cliente.

Segundo Cardellini et al. (1999), deve-se observar a limitação de controle que essa so-

lução dispõe devido aos muitos servidores de nome intermediários entre o cliente e o DNS do *cluster*, sem contar o próprio *cache* do navegador cliente. Há também um problema de disponibilidade quando um dos *hosts* do cluster falha e o DNS do *cluster* continua direcionando requisições para o mesmo, sem perceber a falha. O algoritmo normalmente utilizado nessa seleção é o *Round Robin*. Algoritmos de escalonamento usados no DNS do *cluster* distinguem as arquiteturas baseadas em DNS. Algumas informações do sistema podem ser utilizadas na tomada de decisão entre os *hosts* servidores do *cluster*. Inúmeras políticas de escalonamento em DNS foram propostas em Colajanni et al. (1998) as quais utilizam informações do cliente para associar as requisições para o *host* servidor mais adequado. Decisões para o escalonamento propostas em Schemers (1995) são baseadas na carga corrente de cada *host* do *cluster*.

Valores iguais de TTL (*TTL constante*) podem ser atribuídos às requisições de mapeamento de endereços, assim como valores que se adaptam de acordo com o estado do sistema também podem ser utilizados (*TTL dinâmico*). Uma nova classe de políticas de escalonamento com TTL dinâmico foi proposta em Colajanni et al. (1998).

3.4.2 Balanceamento de Carga Baseado em um Direcionador

Uma abordagem alternativa à arquitetura baseada em DNS visa ter total controle sobre as requisições de clientes e ocultar o roteamento entre múltiplos servidores. Para esse propósito, a omissão de endereços dos *hosts* servidores realizada na solução baseada em DNS é estendida do nível da URL para o nível do IP. Nessa abordagem um único endereço IP virtual é fornecido ao *Web cluster*. Esse é o endereço do chamado direcionador que atua como escalonador central do *cluster*.

Tipicamente, arquiteturas baseadas em um direcionador utilizam algoritmos para seleção de um *host* servidor. O direcionador é responsável por gerenciar o fluxo de requisições HTTP do *Web cluster* com um mínimo de processamento para cada requisição. Algoritmos mais sofisticados podem interagir com essa arquitetura para obter um controle mais refinado das requisições.

Para distribuir a carga, o direcionador reconhece cada *host* do *cluster* por meio de um endereço privado que pode estar em um nível de protocolo diferente, dependendo da arquitetura proposta. Os dois principais métodos utilizados para roteamento são: por meio de um mecanismo de **reescrita de pacotes** e por **redirecionamento HTTP**.

A reescrita de pacotes pode ser feita de forma **simples** ou **dupla** pelo direcionador do *cluster*. Na forma simples, o direcionador recebe todas as requisições HTTP dos clientes, substitui o endereço IP de destino de cada uma, recalcula o *checksum* TCP/IP e reescreve o pacote

Figura 3.6: Simples reescrita de pacotes pelo direcionador (CARDELLINI et al., 1999)

para o *host* servidor selecionado.

Cada *host* servidor, antes de responder uma requisição de um cliente, precisa substituir o endereço de origem pelo endereço IP do direcionador. Desta forma, a arquitetura distribuída fica transparente para o cliente e a tarefa de reescrever os pacotes e recalculer os *checksums* fica dividida entre direcionador e os *hosts* servidores. A Figura 3.6 ilustra a simples reescrita de pacotes.

Figura 3.7: Dupla reescrita de pacotes pelo direcionador (CARDELLINI et al., 1999)

Na dupla reescrita de pacote, o direcionador também recebe todas requisições e as reescreve para o *host* servidor selecionado igualmente na reescrita simples. A diferença é que quando o *host* servidor selecionado responde a requisição do cliente, a mesma não é destinada diretamente ao cliente, ou seja, passa pelo direcionador. Assim, o direcionador precisa reescrever cada pacote que o *host* selecionado enviar para o cliente, substituindo o endereço IP de origem pelo endereço IP do *cluster*. Esse mecanismo de dupla reescrita de pacotes é baseado na tradução de endereços de rede (*Network Address Translation* – NAT) e se limita à capacidade do direcionador de reescrever pacotes e recalculer *checksums*. A Figura 3.7 ilustra a dupla reescrita

de pacotes.

No mecanismo de reescrita de pacotes por redirecionamento HTTP, o direcionador recebe todas as requisições e através de um mecanismo fornecido pelo protocolo HTTP, distribui as requisições entre os *hosts* servidores. O protocolo HTTP permite ao direcionador encaminhar uma requisição para clientes, especificando o código apropriado de estado na resposta, adicionando no cabeçalho HTTP o endereço IP do *host* servidor do qual o cliente deve solicitar o documento.

O capítulo 4 discute em maiores detalhes a reescrita simples e dupla de pacotes que podem ser implementadas através de diferentes mecanismos de roteamento.

3.4.3 Balanceamento de Carga Baseado no Servidor

As técnicas baseadas no servidor utilizam dois níveis de mecanismos de escalonamento: inicialmente as requisições clientes associadas ao *hosts* servidores pelo DNS primário do *cluster*; então, cada *host* pode re-atribuir uma requisição recebida para qualquer outro servidor do *cluster*. Segundo Cardellini et al. (1999) o redirecionamento pode ser implementado de duas formas diferentes: **baseado no mecanismo de reescrita de pacotes** e baseado nas vantagens e facilidades de **redirecionamento do protocolo HTTP**.

Cada *host* servidor decide sobre o redirecionamento baseado em critérios que visam minimizar o tempo de resposta da requisição HTTP do cliente. A estimativa desse tempo é calculada tomando como base o poder de processamento de cada *host* servidor, banda e latência da rede. Esse redirecionamento pode ser feito utilizando recursos do protocolo HTTP.

Figura 3.8: Redirecionamento HTTP pelo servidor (CARDELLINI et al., 1999)

A Figura 3.8 ilustra o redirecionamento HTTP baseado no servidor. Primeiramente o cliente solicita ao DNS a resolução do endereço URL do *cluster* para endereço IP. Em seguida, o DNS realiza o primeiro nível de escalonamento e envia ao cliente o endereço IP do *host* servidor selecionado. Com essa informação, o cliente estabelece conexão diretamente com um dos servidores que, com base em seu respectivo estado de carga, pode optar por responder ao cliente o conteúdo requisitado ou realizar um segundo nível de escalonamento, reenviando ao cliente o novo *host* servidor que deve ser contatado.

Todos os mecanismos de redirecionamento propostos implicam em uma sobrecarga nas comunicações internas ao *cluster*, pois cada *host* precisa transmitir periodicamente informações de seu estado para os outros *hosts* servidores e para o DNS do *cluster*. Esse custo porém, tem um efeito insignificante no tráfego de rede gerado pelas requisições de clientes.

Figura 3.9: Redirecionamento de pacotes pelo servidor (CARDELLINI et al., 2002)

Conforme ilustra a Figura 3.9, o redirecionamento de pacotes pelo servidor utiliza a política *Round Robin* no DNS para realizar o primeiro escalonamento das requisições entre os *hosts* do *cluster*. O *host* servidor que recebe uma requisição está habilitado a realizar novo roteamento para outros *hosts* por meio do mecanismo de reescrita de pacotes que, diferente do redirecionamento por HTTP, é transparente ao cliente.

3.5 Considerações Finais

Uma visão geral sobre a área de balanceamento de carga aplicada à arquiteturas de servidores *Web* distribuídos é apresentada neste capítulo. Também foram apresentadas algumas soluções de arquiteturas existentes baseadas em entidades que escalonam requisições de clientes entre múltiplos *hosts* servidores *Web* distribuídos: **baseadas no cliente, baseadas no**

DNS, baseadas em um direcionador e baseadas no servidor. Diante do conteúdo apresentado conclui-se que o balanceamento de carga é um fator crítico em operações realizadas em sistemas de servidores *Web* distribuídos de alto desempenho. Por fim, se faz necessário ressaltar a importância da compatibilidade das soluções apresentadas com os padrões e protocolos da Internet, permitindo adaptações sem nenhuma limitação.

O próximo capítulo aborda inúmeras soluções existentes de mecanismos de roteamento para *Web clusters*, apresentando suas vantagens, limitações e viabilidade nos atuais padrões da Internet.

Mecanismos de Roteamento para Clusters Web

4.1 Considerações Iniciais

Detalhes da arquitetura *Web clusters*, ressaltando a existência do direcionador defronte aos *hosts* servidores, responsável por receber e distribuir todo o fluxo de requisições HTTP do cluster, foram apresentados no capítulo anterior.

O direcionador identifica univocamente cada *host* do *Web cluster* por meio de um endereço IP privado que pode estar em um diferente nível de protocolo, dependendo da arquitetura. Mais especificamente, o endereço privado de um *host* servidor pode corresponder a um endereço IP ou um endereço de baixo nível. Existem várias técnicas para dispor *Web clusters*, entretanto, a principal função é sempre exercida pelo direcionador.

Na arquitetura *Web cluster*, alternativas para classificar os mecanismos de roteamento são propostas por Cardellini et al. (2002), de acordo com a pilha de protocolos do modelo de referência OSI, na qual o direcionador encaminha pacotes que chegam dos clientes tanto no nível de rede (camada 4) quanto no nível de aplicação (camada 7). A escolha do mecanismo de roteamento também tem um grande impacto na política de escalonamento, devido à grande diferença de tipos de informações disponíveis no direcionador. Existem dois tipos de direcionadores:

- Direcionadores de camada 4: executam um roteamento imediato, pois eles definem para qual *host* servidor é atribuída uma requisição assim que o cliente solicita conexão TCP/IP;

ou seja, logo em seguida ao primeiro pacote TCP SYN que chega no direcionador. Como o pacote do cliente não chega até a camada de aplicação o mecanismo de roteamento é eficiente, mas a política de escalonamento não usa o conteúdo da requisição do cliente como recurso para tomada de decisão.

- Direcionadores de camada 7: executam um roteamento mais refinado, baseando-se no conteúdo da requisição para tomada de decisão. Primeiro o direcionador estabelece uma conexão TCP completa com o cliente, examina a requisição HTTP na camada de aplicação e então a direciona para um *host* servidor. Esse mecanismo de roteamento é muito menos eficiente que o anterior, mas suporta políticas de escalonamento mais sofisticadas. (O direcionador de camada 7 é referenciado neste trabalho de acordo com o modelo de referência ISO/OSI, onde a camada de aplicação é a sétima).

Figura 4.1: Taxonomia de arquiteturas *Web Cluster* (CARDELLINI et al., 2002)

A arquitetura *Web cluster* baseada em direcionadores de camada 4 e de camada 7 pode ser detalhadamente classificada baseada no fluxo de dados entre o cliente e um dos *hosts* servidores, na qual a principal diferença está na forma como os dados são retornados do servidor para o cliente. Na verdade todas as requisições dos clientes passam pelo direcionador. Por outro lado, o *host* servidor responsável por processar a requisição pode responder diretamente para o cliente (denominada arquitetura *one-way*) ou retornar a resposta para o direcionador que torna a enviar a resposta de volta para o cliente (denominada arquitetura *two-way*).

A Figura 4.6 apresenta resumidamente a taxonomia proposta por Cardellini et al. (2002), que é discutida em maiores detalhes nas próximas seções. Tipicamente as arquiteturas *one-way* são mais complexas e mais eficientes, pois o direcionador processa apenas os pacotes que chegam enquanto o oposto é verdadeiro para arquiteturas *two-way*, devido ao fato de que

o direcionador tem de processar tanto pacotes que chegam quanto pacotes que saem do *Web cluster*.

4.2 Soluções Baseadas em Direcionadores de Camada 4

Considerando que os pacotes pertencentes a uma mesma conexão devem ser associados a um mesmo *host* servidor, a conexão do cliente é gerenciada na camada de sessão TCP. O direcionador mantém uma tabela de conexões associando cada sessão TCP dos clientes com seu respectivo *host* servidor.

O direcionador, ao receber um pacote, examina o cabeçalho TCP do mesmo e determina, baseado nos bits dos campos de *flags*, se o pacote pertence a uma nova conexão, a uma conexão correntemente estabelecida ou a nenhuma delas. Se o pacote é para uma nova conexão (isto é, o bit do *flag* SYN está ativado) então, por meio de uma política de escalonamento o direcionador seleciona um *host* servidor para processar a requisição e em seguida registra a conexão do cliente com o *host* servidor na tabela de conexões.

Por fim, encaminha o pacote para o *host* servidor selecionado. Se o pacote não pertencer a uma nova conexão, o direcionador verifica na tabela de conexões se o pacote pertence ou não a uma conexão existente. Se pertencer, o direcionador encaminha o pacote para o *host* servidor que está associado para aquela conexão. Caso contrário o direcionador descarta o pacote.

Para melhorar o desempenho dos direcionadores, a tabela de conexões tipicamente é armazenada em memória e acessada por meio de uma função *hash*. Cada entrada da tabela contem quatro tuplas (endereço IP de origem, porta de origem, endereço IP de destino, porta de destino) e outras informações (por exemplo, horário de início) que podem ser relevantes para alguns algoritmos de escalonamento.

A informação do cliente nas quatro tuplas é dinâmica, enquanto endereço/porta do servidor é uma informação estática, conhecida pelo direcionador na inicialização do cluster. *Web clusters* baseados na camada de rede podem ser classificados com base no mecanismo adotado para rotear tanto pacotes que chegam para um *host* servidor quanto para rotear pacotes que saem do *cluster* para o cliente.

A principal diferença está na forma como os pacotes saem de um *host* servidor do *cluster* para o cliente. Nas arquiteturas *two-way*, pacotes que chegam e que saem do *cluster* passam pelo direcionador, enquanto nas arquiteturas *one-way* apenas os pacotes que chegam passam pelo direcionador.

4.2.1 Arquiteturas Two-Way

Nas arquiteturas *two-way*, cada *host* servidor do *cluster* é configurado com um único endereço IP, que é o endereço privado na camada IP. Ambos os pacotes, tanto os que chegam quanto os que saem são reescritos na camada TCP/IP pelo direcionador. A Figura 4.2 mostra o fluxo dos pacotes. A reescrita de pacotes é baseada no método IP NAT (*Network Address Translation*) (SRISURESH et al., 2001).

Figura 4.2: Arquitetura two-way em direcionadores de camada 4 (CARDELLINI et al., 2002)

Quando o direcionador reescreve os pacotes que chegam, ele realiza uma alteração no campo do endereço de destino do cabeçalho do pacote, substituindo o endereço IP virtual (que representa o *cluster* como um todo) pelo endereço privado do *host* servidor selecionado.

Os pacotes que saem de um *host* servidor para o cliente também passam de volta pelo direcionador. No campo que especifica o endereço IP de origem (no cabeçalho dos pacotes servidor-cliente) está o endereço privado do *host* servidor. O direcionador substitui então esse endereço privado no cabeçalho do pacote pelo endereço IP virtual do *cluster*, para não confundir o cliente. Além disso, o direcionador precisa recalcular o valor do *checksum* no cabeçalho TCP e IP de cada pacote (tanto os pacotes de chegada quanto os de saída).

4.2.2 Arquiteturas One-Way

Segundo Cardellini et al. (2002), nas arquiteturas *one-way*, os pacotes que chegam para o *Web cluster* passam pelo direcionador e os que saem vão diretamente para o cliente. Isto

requer uma banda de rede de alta velocidade exclusiva para pacotes de saída. A Figura 4.3 mostra um exemplo do fluxo de pacotes de saída do *host* servidor 1 para o cliente.

Figura 4.3: Arquitetura one-way em direcionadores de camada 4 (CARDELLINI et al., 2002)

A camada de endereços privados dos servidores não é especificada intencionalmente na Figura 4.3, pois ela também pode estar na camada IP (camada 3) ou na camada de enlace (camada 2 ou camada MAC). O roteamento de pacotes realizado pelo direcionador pode ser implementado através de diferentes mecanismos, tais como reescrita do endereço IP de destino e recálculo do *checksum* TCP/IP dos pacotes que chegam (reescrita de pacotes), encapsulamento de cada pacote dentro de outro pacote (encapsulamento de pacotes) e repasse de pacotes na camada MAC (repasso de pacotes) (CARDELLINI et al., 2002). Cada mecanismo será descrito nas próximas seções.

Reescrita de Pacotes

O roteamento de pacotes do direcionador para um *host* servidor do *cluster* é realizado reescrevendo o endereço de destino dos pacotes que chegam: o direcionador substitui o endereço virtual do *cluster* pelo endereço privado do *host* servidor selecionado e recalcula o *checksum* do cabeçalho TCP e IP do pacote. Na verdade os endereços privados dos *hosts* servidores estão na camada IP (CARDELLINI et al., 2002).

A diferença para as arquiteturas *two-way* está na modificação do endereço de origem dos pacotes que saem para o cliente. O *host* servidor, antes de responder a requisição para o cliente, substitui o endereço IP de origem do pacote pelo endereço IP virtual do cluster e recalcula o

checksum do cabeçalho TCP e IP.

Encapsulamento de Pacotes

Encapsulamento IP (*tunneling IP*) é uma técnica para encapsular datagramas IPs dentro de outros datagramas IP, permitindo assim que datagramas destinados para um endereço IP sejam encapsulados e redirecionados para outro endereço IP (PERKINS, 1996). O objetivo do encapsulamento IP é transferir o cabeçalho e os dados do pacote que chega no direcionador para dentro do campo *payload* do novo pacote que será encaminhado para um dos *hosts* servidores do *cluster*.

Quando um pacote chega ao direcionador, ele é encapsulado em um novo pacote, mas agora contendo como endereço de origem o IP virtual do *Web cluster* e como endereço de destino o endereço privado do *host* servidor selecionado. Os endereços IPs privados dos *hosts* servidores continuam na camada IP, como na reescrita de pacotes. Para que este mecanismo seja implementado, existe a necessidade de que todos os *hosts* servidores do *Web cluster* suportem o encapsulamento de pacotes e que cada um tenha em um de seus dispositivos de rede um túnel configurado com o endereço IP virtual.

Assim que um dos *hosts* servidores recebe um pacote encapsulado, ele extrai os dados do cabeçalho IP e identifica que o pacote interno é destinado para o dispositivo de túnel configurado com o endereço IP virtual. Então o *host* servidor processa a requisição e retorna a resposta diretamente para o cliente (CARDELLINI et al., 2002).

Repasse de pacotes

Esse método assume que o direcionador e os *hosts* servidores tenham um de seus dispositivos de rede fisicamente interconectados por um segmento ininterrupto de uma LAN (CARDELLINI et al., 2002). Diferentemente dos mecanismos de reescrita e encapsulamento de pacotes, os endereços privados dos *hosts* servidores estão na camada de enlace (camada 2 ou camada MAC). O mesmo endereço IP virtual é compartilhado entre o direcionador e os *hosts* servidores do *Web cluster*, por meio do uso de endereços IP primários e secundários. Em particular, o endereço secundário de cada *host* servidor é configurado com um endereço IP virtual por meio do mascaramento da interface de *loopback*. Apesar de todos os *hosts* servidores compartilharem o mesmo endereço IP virtual, os pacotes chegam primeiro no direcionador e para evitar colisões, os *hosts* servidores têm desativado o mecanismo ARP (*Address Resolution Protocol*). O direcionador repassa um pacote para um dos *hosts* servidores, escreve o endereço MAC de destino na camada de enlace e retransmite o quadro no seguimento comum de rede LAN. Essa operação não requer qualquer modificação do cabeçalho TCP/IP do pacote repassado. Embora todos os

hosts compartilhem o mesmo endereço IP virtual do *Web cluster*, o *host* servidor que receber o pacote pode se identificar como um destino e assim responder diretamente para o cliente. Não se faz necessário executar operações na camada TCP/IP, tipicamente utilizadas em mecanismos de reescrita e encapsulamento de pacote.

4.3 Soluções Baseadas em Direcionadores de Camada 7

Direcionadores de camada 7 trabalham na camada de aplicação, disponibilizando uma distribuição baseada no conteúdo das requisições HTTP. Segundo Cardellini et al. (2002), os mecanismos de roteamento na camada 7 são muito mais complexos do que os mecanismos de roteamento na camada 4, pois o conteúdo das requisições HTTP é analisado antes de qualquer tomada de decisão. Por esse motivo o direcionador precisa primeiro estabelecer uma conexão TCP com o cliente (ou seja, as três formas de *handshake* na configuração da conexão TCP precisam ser completadas entre o cliente e o direcionador) e então receber a requisição HTTP na camada de aplicação.

Por outro lado, um direcionador de camada 4 seleciona um *host* servidor do *Web cluster* assim que recebe o pacote TCP SYN inicial, antes mesmo do cliente enviar a requisição HTTP. Para Cardellini et al. (2002), como as soluções baseadas na camada 4, a arquitetura *Web cluster* baseada na camada 7 também pode ser classificada tomando por base os mecanismos usados para responder pacotes de um *host* servidor do *Web cluster* para o cliente. As arquiteturas *one-way* e *two-way* se distinguem de acordo com fluxo de dados através do direcionador.

4.3.1 Arquiteturas Two-Way

Nas arquiteturas *two-way* o tráfego de saída dos *hosts* servidores do *Web cluster* precisam passar pelo direcionador, como mostra a Figura 4.4. Os métodos propostos por Cardellini et al. (2002), basicamente se diferem de acordo com o mecanismo utilizado pelo direcionador para rotear requisições para um *host* servidor.

TCP Gateway

Nesse método, um servidor *proxy* na camada de aplicação é executado no direcionador, intermediando a comunicação entre o cliente e o *hosts* servidores do *Web cluster*. Esse *proxy* mantém aberta uma conexão TCP persistente com cada *host*. Quando uma requisição chega, o *proxy* aceita a conexão do cliente e repassa a requisição do cliente para o *host* servidor selecionado por meio da conexão TCP persistente. Quando as respostas chegam dos *hosts* servidores, o *proxy* repassa as mesmas para o cliente pela sua respectiva conexão (CARDELLINI et al.,

Figura 4.4: Arquitetura two-way em direcionadores de camada 7 (CARDELLINI et al., 2002)

2002).

TCP Splicing

Esse mecanismo visa a melhorar o método *TCP gateway* que possui um custo computacional elevado, pois cada pacote de requisição/resposta necessita ser analisado pela camada de aplicação. Similar ao *TCP gateway*, o direcionador mantém uma conexão TCP persistente com cada *host* servidor.

Segundo Cardellini et al. (2002), diferentemente da técnica anterior, *TCP splicing* repassa pacotes entre a camada de rede e a camada de interface com a rede, em relação à pilha de protocolos TCP/IP. Isso é realizado diretamente pelo sistema operacional que deve ser modificado em nível de *kernel*.

Uma vez estabelecida uma conexão TCP entre cliente e direcionador e estabelecida entre o direcionador e o *host* servidor escolhido, as duas conexões estão interligadas. Da mesma forma, pacotes IP são repassados do direcionador para os *hosts* servidores sem ter de passar da camada de transporte até a camada de aplicação (CARDELLINI et al., 2002). Após o cliente e o *host* servidor estarem conectados, o direcionador trata os pacotes subsequentes modificando o cabeçalho TCP e IP dos pacotes (isto é, modifica o endereço IP e recalcula o *checksum*) fazendo com que cliente e servidor identifiquem os pacotes como destinados para si próprios. O mecanismo *TCP splicing* também pode ser implementado em hardware ou por bibliotecas de funções definidas em nível de *socket*.

4.3.2 Arquiteturas One-Way

Nas arquiteturas *one-way*, os *hosts* servidores retornam os pacotes requisitados diretamente para o cliente, sem passar pelo direcionador, como mostrado na Figura 4.5.

Figura 4.5: Arquitetura one-way em direcionadores de camada 7 (CARDELLINI et al., 2002)

TCP Hand-Off

Uma vez que o direcionador estabeleceu conexão TCP com o cliente e selecionou um *host* servidor do *Web cluster*, ele encerra a conexão TCP com o *host* servidor que poderá se comunicar diretamente com o cliente Pai et al. (1998). O mecanismo *TCP hand-off* é transparente para o cliente, pois os pacotes enviados do *host* servidor parecem ter sido enviados pelo direcionador.

O tráfego que chega ao direcionador em uma conexão já estabelecida (isto é, qualquer pacote de *acknowledgment* enviado pelo cliente para o direcionador) é repassado para o *host* servidor selecionado através de um módulo eficiente executado sob a pilha de protocolos do direcionador. O mecanismo *TCP hand-off* requer modificações no sistema operacional tanto do direcionador quanto dos *hosts* servidores do *Web cluster*. Entretanto, essa não é uma limitação grave, haja visto que os *Web clusters* são soluções especiais. Além disso, *TCP hand-off* permite ao direcionador tratar conexões HTTP/1.1 persistentes e associar requisições HTTP de uma mesma conexão para diferentes *hosts* servidores (Aron et al. 1999).

Existem dois mecanismos para suportar conexões persistentes: o protocolo *hand-off*

pode ser entendido para permitir ao direcionador migrar uma conexão de um *host* servidor para outro (múltiplo *hand-off*); o primeiro *host* servidor não consegue responder uma requisição e a repassa para um segundo *host* que envia a resposta para o cliente (repasse entre *hosts* servidores).

TCP Connection Hop

Segundo Cardellini et al. (2002), esta é uma solução proprietária baseada em *software* implementada pela Resonate como um protocolo de encapsulamento baseado em TCP. Uma vez que o direcionador estabeleceu conexão TCP com o cliente e selecionou o *host* servidor, ele transfere a conexão TCP para o *host* servidor.

Isso é possível através do encapsulamento de um pacote IP em um pacote RPX (*Resonate Exchange Protocol*), que é enviado para o *host* servidor (Resonate 2002). A operação de transferência de conexão na camada de rede, na verdade minimiza a latência na chegada de pacotes. O *host* servidor pode responder diretamente para o cliente por compartilhar o mesmo endereço IP virtual do direcionador. Pacotes ACK e de informações de sessões persistentes de cliente são gerenciados pelo direcionador.

4.4 Comparação de Mecanismos de Roteamento para Clusters Web

Nesta seção, primeiramente serão discutidas as diferenças entre as soluções de roteamento baseadas em camada de rede(4) e camada de aplicação(7) e em seguida realizadas comparações entre as duas classes. A Figura 4.6 resume a classificação dos mecanismos de roteamento para *Web clusters*.

4.4.1 Mecanismos de Roteamento na Camada 4

Para Cardellini et al. (2002), uma vantagem das soluções *two-way* é que os *hosts* servidores do cluster podem estar em diferentes LANs. A principal restrição é que tanto os pacotes que chegam quanto os que saem do *Web cluster* precisam passar pelo direcionador. A consequência é que o direcionador precisa reescrever os pacotes que chegam e que saem e tipicamente, o número de pacotes que saem do *Web cluster* é maior que o número de pacotes que chegam.

Na verdade, a escalabilidade de um *Web cluster* que usa a arquitetura *two-way* é limitado pela capacidade do direcionador de reescrever pacotes e recalcular *checksums*, mesmo que seja fornecido o suporte de um *hardware* dedicado para operações de *checksum*. Segundo Rijssinghani (1994), é importante notar que a reescrita de cabeçalhos TCP não representa ne-

Figura 4.6: Taxonomia detalhada dos mecanismos de roteamento para *Web Clusters* (CARDELLINI et al., 2002)

cessariamente a recomputação de *checksum* dos pacotes que chegam, pois os novos *checksums* podem ser obtidos pela subtração do *checksum* antigo.

Um direcionador de camada 4 que usa uma solução *one-way* pode sustentar maior tráfego (antes do sistema se tornar um ponto de saturação do sistema) do que um sistema que usa uma solução *two-way* (CARDELLINI et al., 2002). Na verdade o desempenho do sistema é limitado pela capacidade do direcionador ativar, localizar e remover entradas na tabela de conexões.

Se considerados os diferentes mecanismos para arquiteturas *one-way*, observa-se que a simples reescrita de pacotes causa a mesma sobrecarga que a dupla reescrita de pacotes, mas ele reduz as operações de direcionamento, pois o maior número de pacotes que saem do *Web cluster* para o cliente são reescritos pelos *hosts* servidores e não pelo direcionador. Isto requer modificações no *kernel* do sistema operacional dos *hosts* servidores.

Mecanismos de repasse de pacotes visam limitar a sobrecarga de reescrita de pacotes graças a dois mecanismos. Segundo Cardellini et al. (2002), o direcionador processa apenas pacotes que chegam e trabalha na camada MAC que evita o custo de recalculer *checksums*. Uma limitação do repasse de pacotes é a necessidade do direcionador e dos *hosts* servidores do *Web cluster* precisarem estar no mesmo segmento de rede. Entretanto, essa restrição tem um impacto bastante limitado, desde que o direcionador e os *hosts* servidores estejam conectados através de uma mesma LAN.

Soluções que utilizam o encapsulamento de pacotes têm boa escalabilidade, porém menor do que o repasse de pacotes. Elas requerem que os *hosts* servidores suportem o encapsulamento IP, o que ainda não é padrão nos atuais sistemas operacionais. Porém, tanto na literatura

quanto nas outras soluções baseadas em sistemas de propósitos especiais, isso não é considerado uma limitação grave.

Devido ao aprimoramento dos *hardwares* e mecanismo *one-way*, a escalabilidade de *Web clusters* baseados em soluções na camada 4 é primeiramente limitada pela capacidade de tráfego do canal de comunicação entre os *hosts* servidores e a Internet.

4.4.2 Mecanismos de Roteamento na Camada 7

Direcionadores de camada 7 que utilizam soluções *one-way* ativam os *hosts* servidores para responderem requisições HTTP diretamente para os clientes, oferecendo maior escalabilidade que as soluções baseadas na arquitetura *two-way*. Para Aron et al. (2000), em particular, o método *TCP hand-off* é mais escalável que o *TCP splicing*. Entretanto, soluções *one-way*, operando na camada 7 necessitam de alterações nos sistemas operacionais tanto do direcionador quanto dos *hosts* servidores.

Tipicamente, arquiteturas *two-way* são combinadas com operações na camada 4, embora elas possam ou não requerer sistemas operacionais de propósito específico quando combinados com soluções na camada 7. Entretanto, as comunidades acadêmica e comercial aceitam usar soluções de propósito específico baseadas em *Web clusters*, especialmente se elas garantirem melhor desempenho.

Arquiteturas *two-way* combinadas com direcionadores de camada 7 apresentam a vantagem de permitirem que soluções em cache sejam implementadas no direcionador. Isso permite ao direcionador responder diretamente por uma requisição ao cliente (caso contenha o conteúdo em cache), diminuindo a carga nos *hosts* servidores.

A principal vantagem do método *TCP gateway* é a simplicidade de sua implementação que pode ser feita em qualquer sistema operacional. O principal problema é a sobrecarga que essa solução adiciona aos pacotes que chegam e saem pelo direcionador, pois precisam chegar até a camada de aplicação. *TCP splicing* reduz a sobrecarga em relação ao *TCP gateway*, pois ele elimina o custo da cópia e troca de contexto, resultantes do uso da camada de aplicação pelo *proxy*. Assim mesmo, o direcionador pode se tornar facilmente um ponto de saturação do *cluster*, por precisar modificar o cabeçalho TCP e IP dos pacotes.

4.4.3 Roteamento na Camada 4 x Roteamento na Camada 7

Para Cardellini et al. (2002), a principal vantagem dos mecanismos de roteamento na camada 7 sobre as soluções na camada 4 é a possibilidade de usar no direcionador algoritmos de escalonamento que se baseiam no conteúdo das requisições para tomada de decisão.

No capítulo 5, observa-se que através destas políticas é possível alcançar altas taxas de acerto em cache, dividir o conteúdo *Web* entre os *hosts* servidores do *Web cluster*, utilizar *hosts* servidores especializados, associar sessões SSL para um mesmo *host* servidor e alcançar uma distribuição mais refinada de requisições mesmo com conexões persistente HTTP/1.1.

Por outro lado, mecanismos de roteamento na camada 7 introduzem altas doses de sobrecarga no direcionador, o que pode causar-lhe severos limites de escalabilidade. Aron et al. (2000) mostram que o tráfego máximo de soluções em software baseadas na camada 7 que utilizam *TCP hand-off* está limitado em 3500 conexões/segundo, enquanto as mesmas soluções em software, baseadas na camada 4 e implementadas em uma mesma configuração de hardware, suportam até 20000 conexões/segundo.

Para melhorar a escalabilidade em arquiteturas de camada 7, soluções alternativas para servidores *Web* escaláveis têm sido propostas, combinando técnicas de distribuição de carga que se baseiam no conteúdo das requisições para tomada de decisão juntamente com técnicas que não se baseiam no conteúdo para a tomada de decisão. Estas soluções serão descritas nas próximas seções.

4.5 Considerações Finais

Diversos mecanismos de roteamento de requisições que podem ser implementados em *Web clusters* foram apresentados neste capítulo, destacando detalhes de implementação tanto em nível de rede como em nível de aplicação e comparando as vantagens e limitações de cada mecanismo.

O próximo capítulo aborda diversas políticas e algoritmos de escalonamento que visam atingir o balanceamento de carga em *Web clusters*. Discute também diversas possibilidades para utilizar algoritmos de escalonamento que se baseiam em informações do cliente, do servidor e de ambos para a tomada de decisão.

Algoritmos e Políticas de Escalonamento para Clusters Web

5.1 Considerações Iniciais

Neste capítulo são descritas as políticas de escalonamento que podem ser utilizadas para selecionar um *host* servidor do *Web cluster*. A política de escalonamento influencia na escalabilidade do sistema e no desempenho para o usuário final. Em um *Web cluster*, a política de escalonamento é executada pelo direcionador, que age como um escalonador global para o sistema.

Políticas de escalonamento global foram classificadas de diferentes formas, permitindo diferentes critérios (CASAVANT; KUHL, 1988). Existem inúmeras alternativas, assim como: balanceamento de carga *vs.* compartilhamento de carga, centralizado *vs.* distribuído e algoritmos estáticos *vs.* algoritmos dinâmicos, mas apenas parte desse conjunto é adequado para ser aplicado no balanceamento de carga em *Web clusters*. Essa arquitetura com um simples direcionador que recebe todas as requisições HTTP do cliente, induz a escolha de uma política de balanceamento centralizada.

Considerando que o principal objetivo do compartilhamento de carga é amenizar os períodos de picos de sobrecarga e que o do balanceamento de carga é igualar a carga entre os *host* servidores, um algoritmo de escalonamento implementado no direcionador visa compartilhar a carga do *Web cluster* e não somente balancear (SHIVARATRI et al., 1992).

Absoluta estabilidade muitas vezes é impossível de ser alcançada em sistemas dinâmicos como *Web clusters* para hospedagem de páginas *Web* “populares” e nem sempre se faz necessário (CARDELLINI et al., 2002).

Uma alternativa real está entre os algoritmos estáticos e dinâmicos, um vez que o direcionador não pode usar algoritmos sofisticados de escalonamento, pois ele precisa tomar decisões imediatas para centenas ou milhares de requisições por segundo. Algoritmos estáticos são as soluções mais rápidas e evitam que o direcionador se torne o primeiro ponto de saturação do *Web cluster*, pois eles não consideram o estado atual do sistema na tomada de decisão. Entretanto, esses algoritmos podem tomar decisões potencialmente inúteis.

Algoritmos dinâmicos caracterizam o estado atual do sistema melhor do que algoritmos estáticos por utilizarem no momento da tomada de decisão, algumas informações relacionadas ao estado atual do sistema. Por outro lado, exigem mecanismos que coletam e analisam informações, desse modo implicando numa sobrecarga de custo potencialmente elevada. Para Cardellini et al. (2002), os requisitos apresentados a seguir resumem as restrições para algoritmos de escalonamento:

- Baixa complexidade computacional, pois as decisões de escalonamento são exigidas em tempo real;
- Total compatibilidade com os atuais padrões e protocolos existentes na *Web*.
- Todas as informações de estados necessárias pela política de escalonamento precisam ser acessíveis pelo direcionador.

5.2 Uma Taxonomia de Algoritmos de Escalonamento

Algoritmos de escalonamento de tarefas possuem um grande número de classificações diferentes, pois são aplicados em diferentes áreas da computação. Na prática, para a distribuição de carga *Web*, a principal classificação se restringe entre algoritmos estáticos ou dinâmicos (CARDELLINI et al., 2002). Uma terceira classe de políticas de escalonamento tem sido investigada na literatura de sistemas distribuídos, chamada de algoritmos adaptativos, onde a política de escalonamento pode ser modificada com base nas condições da carga do sistema (SHIVARATRI et al., 1992). Porém, está fora do conhecimento a existência de algoritmos dessa natureza aplicados em arquiteturas de servidores *Web* distribuídos.

Algoritmos de escalonamento estáticos não consideram qualquer informação de estado do sistema para a tomada de decisão. Por outro lado, algoritmos dinâmicos podem levar em con-

sideração uma variedade de informações de estado do sistema, entretanto dependem de qual camada na pilha de protocolos o direcionador opera. Se o direcionador operar na camada TCP/IP (camada 4), o algoritmo de escalonamento é classificado como *content-blind* (não baseado no conteúdo da requisição HTTP) e se o direcionador trabalhar na camada de aplicação (camada 7), o algoritmo é classificado como *content-aware* (baseado no conteúdo da requisição HTTP).

Cardellini et al. (2002) assumem que algoritmos estáticos são desenvolvidos apenas para direcionadores que trabalham na camada TCP/IP. Na verdade, o uso de uma arquitetura sofisticada como a de camada 7 é motivada apenas se os benefícios da mesma forem completamente explorados pelos algoritmos de escalonamento. Algoritmos dinâmicos podem ser classificados de acordo com o nível de informações de estados do sistema usadas pelo direcionador. Cardellini et al. (2002), consideram três classes:

- Políticas baseadas em informações do cliente: O algoritmo de escalonamento implementado no direcionador toma decisões com base em algumas informações do cliente. Direcionadores de camada 4 podem usar apenas informações da camada de rede relacionadas ao cliente, como endereço IP e porta TCP. Por outro lado, direcionadores de camada 7 podem examinar o conteúdo das requisições HTTP e tomar decisões com base em informações mais detalhadas sobre o cliente.
- Políticas baseadas em informações do servidor: O algoritmo de escalonamento implementado no direcionador toma decisões com base em algumas informações dos servidores do *Web cluster*, como: carga corrente e anterior, tempo de latência e disponibilidade. Além disso, em algoritmos de escalonamento baseados no conteúdo das requisições HTTP (*content-aware*) também podem ser consideradas informações de conteúdo em caches nos discos dos *hosts* servidores.
- Políticas baseadas em informações do cliente e do servidor: O algoritmo de escalonamento implementado no direcionador escalona requisições HTTP com base numa combinação de informações de estados tanto dos *hosts* servidores do *Web cluster* quanto do cliente. Atualmente, existem muitas políticas baseadas em informações dos clientes que pertencem a esta classe, pois sempre usam informações mais ou menos precisas sobre a carga dos *hosts* servidores.

A Figura 5.1 resume a taxonomia dos algoritmos de escalonamento e que são descritos posteriormente. Faz-se necessário salientar que algoritmos estáticos e políticas baseadas em informações do servidor são significativos apenas para direcionadores de camada 4 (que operam na camada TCP/IP), ou seja, que não levam em consideração o conteúdo das requisições HTTP.

Figura 5.1: Taxonomia de políticas de escalonamento para *Web clusters* (CARDELLINI et al., 2002)

5.3 Políticas de Escalonamento não Baseadas em Conteúdo

Nesta seção são descritas as principais políticas de escalonamento que não levam em consideração o conteúdo das requisições HTTP para a tomada de decisão, conforme a taxonomia mostrada na Figura 5.1, e detalhada na Figura 5.2 com alguns algoritmos representativos para cada categoria no nível inferior.

Figura 5.2: Políticas de escalonamento não baseadas em conteúdo (CARDELLINI et al., 2002)

5.3.1 Algoritmos Estáticos

Algoritmos estáticos não consideram qualquer informação de estado do sistema para a tomada de decisão. Exemplos típicos são algoritmos *Aleatórios* e *Round Robin*. Um algoritmo *Aleatório* distribui as requisições HTTP que chegam ao direcionador uniformemente, com igual probabilidade de entrega em qualquer *host* servidor.

Por outro lado, o algoritmo *Round Robin* utiliza uma lista circular e um apontador para o último *host* servidor selecionado, isto é, caso o servidor S_i seja o último escolhido, a nova

requisição que chegar ao direcionador será atribuída ao servidor S_{i+1} , onde $i+1 = (i+1) \bmod N$ e N é o número de *hosts* servidores que compõem o *Web cluster*. Porém, o algoritmo *Round Robin* utiliza apenas informações de decisões tomadas anteriormente.

Algoritmos que implementam as políticas *Aleatórios* e *Round Robin* podem facilmente ser estendidos para um cenário de servidores com capacidades diferentes de processamento fazendo-se as atribuições probabilísticas baseadas na capacidade dos servidores (COLAJANNI et al., 1998). Por isso, se C_i indica a capacidade de um dos *hosts* servidores, a capacidade relativa do mesmo $\xi_i (0 \leq \xi_i \leq 1)$ é definida como $\xi_i = C_i / \max(C)$, na qual $\max(C)$ é a capacidade máxima de um entre todos os outros *hosts* servidores do *Web cluster*. Faz-se necessário notar que a capacidade de um *host* servidor é um parâmetro configurado, ou seja, uma informação estática.

Para algoritmos *Aleatórios*, capacidades heterogêneas podem ser consideradas através da atribuição de diferentes probabilidades de acordo com a capacidade dos servidores. O algoritmo *Round Robin* pode tratar *hosts* servidores heterogêneos da seguinte forma: um número aleatório $\varrho (0 \leq \varrho \leq 1)$ é gerado e, assumindo que S_i seja o último *host* servidor escolhido, a requisição HTTP é atribuída para o servidor S_{i+1} apenas se $\varrho \leq \xi_i$. Caso contrário, o servidor S_{i+2} se torna o próximo candidato e assim sucessivamente.

Diferentes capacidades de processamento também podem ser tratadas utilizando um algoritmo chamado *Round Robin* com atribuição estática de pesos (*Static Weighted Round Robin* – SWRR), que vem a ser uma variação do algoritmo *Round Robin*. Para cada *host* servidor é associado um peso de valor inteiro W_i que indica a capacidade do mesmo. Especificamente, $W_i = C_i / \min(C)$, onde o $\min(C)$ é a capacidade mínima de um entre todos os outros *hosts* servidores do *Web cluster*. A sequência no escalonamento é gerada de acordo com os valores dos pesos dos servidores (Linux Virtual Server, 2002). Como exemplo, assumindo-se que os servidores S_1, S_2 e S_3 têm valores de pesos 3, 2 e 1 respectivamente. Então, a sequência de escalonamento pode ser $S_1 S_1 S_2 S_1 S_2 S_3$.

5.3.2 Políticas Baseadas em Informações do Cliente

Assumindo que os direcionadores de camada 4 não analisam o conteúdo das requisições HTTP, o tipo de informação referente ao cliente se restringe aos dados disponíveis nos pacotes TCP/IP, os quais são o endereço IP de origem e o número da porta TCP. Para Cardellini et al. (2002), essas informações podem ser utilizadas por algoritmos de escalonamento que dividem os *hosts* servidores estaticamente e associam requisições HTTP de mesmos clientes para mesmos *hosts* servidores, tipicamente por meio de uma função *hash* aplicada no endereço IP do

cliente.

5.3.3 Políticas Baseadas em Informações do Servidor

Informações do cliente são imediatamente disponibilizadas no direcionador pois ele recebe todas as requisições por conexões. Por outro lado, quando considerados alguns algoritmos que utilizam informações de estado dos *hosts* servidores, são encontradas algumas dúvidas conhecidas em sistemas interligados em rede: quais os índices de carga dos servidores? como e quando computá-los? como e quando transmiti-los para o direcionador?

Uma vez selecionados os índices de carga dos servidores, o direcionador pode aplicar diferentes algoritmos de escalonamento. O algoritmo chamado Carga Mínima (*Least Loaded* – LL) indica uma classe de algoritmos que dependem dos índices de carga dos *hosts* servidores para a tomada de decisão, ou seja, atribui uma nova conexão para o *host* servidor com índice de carga mais baixo. Por exemplo, no algoritmo de Conexões Mínimas (*Least Connections* – LC), usualmente utilizado em produtos comerciais, o direcionador atribui as novas requisições HTTP dos clientes para o *host* servidor com o menor número de conexões ativas.

Uma simples extensão, a qual atribui pesos estáticos para os *hosts* servidores de acordo com suas capacidade, permite considerar heterogeneidade na capacidade dos servidores (LINUX..., 2002). A idéia fundamental é que servidores com grande capacidade possam suportar um grande número de conexões ativas: isso simplesmente pode ser obtido dividindo o número de conexões ativas pelo peso do servidor. O algoritmo *Tempo de Resposta Rápido* (*Fastest Response Time* – FRT), também utilizado em produtos comerciais, é um exemplo de algoritmo LL, onde o direcionador atribui as novas conexões ao *host* servidor que apresenta o menor tempo de latência observado no intervalo de tempo anterior.

Para Cardellini et al. (2002), o algoritmo *Round Robin* com Atribuição Dinâmica de Pesos (*Dynamic Weighted Round Robin* – DWRR) é uma variação da versão com atribuição estática. Esse algoritmo associa para cada *host* servidor um peso dinamicamente estimado, proporcional ao estado de cada *host*. Periodicamente, o direcionador coleta informações dos servidores e computa os pesos, que são dinamicamente incrementados para cada nova conexão atribuída. O algoritmo *Round Robin* com Atribuição de Pesos (*Weighted Round Robin* – WRR) é freqüentemente referenciado na literatura sem ser especificado se é uma versão estática ou dinâmica.

5.3.4 Políticas Baseadas em Informações do Cliente e do Servidor

Informações de estado do servidor podem ser combinadas com algumas informações disponíveis em um direcionador de camada 4 (como, o endereço IP de origem e o número da porta de serviço do cliente no cabeçalho TCP). Razões de desempenho e funcionais motivam a escolha de atribuições de múltiplas conexões consecutivas de um mesmo cliente para um mesmo *host* servidor, para conceder a chamada *Relação com o Cliente* (*client affinity*) (LINUX..., 2002).

Por exemplo, para evitar tempos de resposta e consumo de recursos em operações de negociação e geração de chaves SSL, pode ser conveniente atribuir conexões SSL consecutivas de um mesmo cliente para um mesmo *host* servidor durante o tempo de vida da chave SSL. Uma sessão FTP é um exemplo de que a relação com o cliente não é atribuída para ganhar desempenho e sim funcionalidade, pois esse protocolo utiliza duas conexões (uma para controle outra para dados) que devem ser estabelecidas com o mesmo *host* servidor (CARDELLINI et al., 2002).

Em relação ao reuso de chaves SSL armazenadas em memória cache, deve-se notar que um direcionador de camada 4 pode identificar apenas um cliente emissor de requisições em um canal seguro por meio do endereço IP de origem. Como diferentes clientes compartilham o mesmo endereço IP de *proxy* ou *firewall*, pode acontecer de requisições HTTP de clientes que não fazem parte da sessão serem atribuídas para o mesmo *host* servidor.

Segundo Cardellini et al. (2002), em políticas que propõem uma relação com o cliente, informações dos clientes usualmente omitem informações dos *hosts* servidores na tomada de decisão. Isto significa que as tarefas anteriores dos clientes têm mais importância do que as condições de estados dos servidores.

5.3.5 Considerações Sobre o Escalonamento não Baseado em Conteúdo

Para um direcionador de camada 4, algoritmos estáticos são as soluções mais rápidas, pois não estão vinculados a qualquer informação do sistema na tomada de decisão. Além disso, são muito mais fáceis de serem implementados. Entretanto, esses algoritmos podem tomar decisões ineficientes devido aos tempos de serviços serem muito variáveis e o consumo de recursos depender da característica da carga de trabalho.

Para limitar esse risco, alguns autores propuseram a integração de um algoritmo estático no direcionador com um mecanismo de roteamento de segundo nível executado nos *hosts* ser-

vidores do *Web cluster*. Os algoritmos de escalonamento dos servidores tipicamente levam em consideração o conteúdo das requisições HTTP e visa melhorar o compartilhamento de carga e armazenamento em cache.

Algoritmos dinâmicos têm potencial para executar algoritmos estáticos por utilizarem algumas informações de estado do sistema na tomada de decisão. Entretanto, eles requerem mecanismos que coletam e analisam as informações de estados, se expondo a altos índices de sobrecarga. Além disso, ajustar corretamente os parâmetros de políticas dinâmicas pode ser uma tarefa difícil em sistemas variáveis como os *Web clusters*.

Para Cardellini et al. (2002), algoritmos baseados no conteúdo das requisições HTTP dos clientes parecem ser a melhor opção, apesar de nem todos trabalharem bem. Por exemplo, o algoritmo LL tende a saturar o *Web cluster* quando todas as requisições HTTP provenientes de clientes são enviadas para o mesmo *host* até a nova informação ser propagada. Esse efeito colateral é bem conhecido em sistemas distribuídos (Dahlin 2000; Mitzenmacher 2000) e ainda assim a política LL é comumente utilizada em produtos comerciais. Por outro lado, muitos experimentos e resultados de simulações demonstraram que a política *Round Robin* com pesos concilia simplicidade com melhor eficácia (CASALICHIO; COLAJANNI, 2001).

5.4 Políticas de Escalonamento Baseadas em Conteúdo

A complexidade dos direcionadores de camada 7 que podem examinar o conteúdo de requisições HTTP, motiva o uso de políticas sofisticadas de distribuição de carga baseadas em conteúdo. Cardellini et al. (2002) apresentam uma taxonomia de políticas de escalonamento baseados em conteúdo mostrada na Figura 5.1 com um nível adicional que considera o principal objetivo das políticas de escalonamento. A Figura 5.3 resume a taxonomia de políticas baseadas em conteúdo e mostra em seu nível inferior os algoritmos propostos que utilizam informações sobre a URL requisitada para diferentes propósitos, tais como:

- aprimorar a localização do conteúdo em memória cache nos *hosts* servidores para reduzir acessos a disco (*cache affinity*);
- utilizar *hosts* servidores especializados para diferentes serviços baseados na *Web* (servidores especializados), assim como *streaming* de dados, conteúdo dinâmico e particionamento de conteúdo *Web* entre vários *hosts* para aumentar a escalabilidade no armazenamento secundário;
- aumentar o compartilhamento de carga entre os *hosts* servidores de um cluster;

Figura 5.3: Políticas de escalonamento baseadas em conteúdo (CARDELLINI et al., 2002)

Informações adicionais relacionadas às requisições HTTP, assim como *cookies* e identificadores SSL podem ser exploradas por políticas *client affinity* ((CARDELLINI et al., 2002)). Na verdade, o protocolo SSL envolve um procedimento de *handshaking* (troca de certificados, negociação de compressão e encriptação, configuração de identificador de sessão) com custo computacionalmente alto, enquanto sessões subsequentes podem evitá-los (por usarem o mesmo identificador de sessão) por um determinado período de tempo. Entretanto, faz-se necessário notar que o suporte a serviços baseados em informações de estado do sistema também pode ser oferecido por direcionadores que operam na camada 4 (camada TCP/IP), porém com um grau inferior de acurácia. Segundo Hunt et al. (1998), esses serviços baseados em informações de estado do sistema podem ser identificados através da porta de serviço (por exemplo, 443 para SSL) e uma conexão pode ser reutilizada através da configuração de um novo *time-out*.

Segundo Cardellini et al. (2002), quando as conexões persistentes HTTP/1.1 são utilizadas, direcionadores de camada 7 podem associar requisições de uma mesma conexão TCP para diferentes *hosts* servidores, atingindo desta forma um controle mais refinado da granulosidade do sistema para requisições HTTP individuais.

Por outro lado, um direcionador de camada 4 deve associar as requisições HTTP de uma mesma conexão TCP para um mesmo servidor. Isto implica que múltiplas requisições de uma simples conexão persistente HTTP/1.1 são atribuídas a um mesmo *host* servidor, ou seja, o controle da granulosidade é realizado de acordo com as conexões TCP estabelecidas entre o cliente e o direcionador. Com conexões HTTP/1.0 não existe diferença para o controle de granulosidade entre direcionador de camada 4 e camada 7, pois existe correspondência de um para um entre uma requisição HTTP e uma conexão TCP.

5.4.1 Políticas Baseadas em Informações do Cliente

Primeiramente é considerada a parte esquerda da Figura 5.3. Em políticas *cache affinity* o espaço dos arquivos tipicamente é particionado entre os *hosts* servidores do *Web cluster*. Segundo Cardellini et al. (2002), para realizar um particionamento estático dos arquivos, uma função *hash* é aplicada a URL (ou uma substring da URL). O algoritmo de escalonamento executado no direcionador utiliza a mesma função. Esse esquema explora a localização dos arquivos nos *hosts* servidores e alcança uma melhor taxa de acertos em memória cache. Entretanto, a combinação de particionamento de objetos *Web* com função *hash* é significativo apenas para conteúdo estático. Além disso, essa combinação ignora completamente o compartilhamento de carga, dificultando o particionamento do espaço entre os arquivos de forma que as requisições fiquem balanceadas.

Na verdade, se um pequeno conjunto de arquivos é requerido por uma grande quantidade de requisições (uma característica bem conhecida de cargas de trabalho da *Web*), o *host* servidor responsável por armazená-los fica com carga superior aos demais. Esses picos críticos de carga podem ser minimizados por arquiteturas que integram soluções de servidores *proxies*, porém outros problemas ainda persistem.

Para páginas *Web* que fornecem serviços heterogêneos, a URL requisitada pode ser utilizada para particionar estaticamente os *hosts* servidores de acordo com o tipo de serviço que cada um fornecer. A política denominada *Servidores Especializados* (*Specialized Servers* –SS) tem como objetivo atribuir a um *host* servidor específico, determinados tipos de requisições, por exemplo: conteúdo dinâmico, arquivos multimídia ou streams de vídeo (Yang e Luo 2000). Um algoritmo de escalonamento pertencente a essa classe de políticas é o Particionamento de Serviços (*Service Partitioning*). Produtos comerciais como BIG-IP (F5 Networks 2002) e Central Dispatch (Resonate 2002)) utilizam essa política.

Outra política de escalonamento, chamada *Compartilhamento de Carga* (*Load Sharing* – LS) tem como principal objetivo melhorar o compartilhamento de carga entre os *hosts* servidores. Segundo Cardellini et al. (2002), essa estratégia não requer particionamento estático de espaço para arquivos e nem para serviços baseados na *Web*. Dois algoritmos pertencem a essa classe:

- Atribuição de Tarefas por Faixas de Tamanho (*Size Intervals Task Assignment with Equal Load* – SITA-E) (HARCHOL-BALTER et al., 2003): A política SITA-E particiona dinamicamente o conteúdo *Web* entre os *hosts* servidores do *Web cluster* de acordo com o tamanho dos arquivos. O direcionador seleciona o *host* servidor de acordo com o ta-

manho do arquivo requisitado. Para (HARCHOL-BALTER et al., 2003), o objetivo é separar os serviços que exigem menos recursos dos que exigem mais. Segundo Cardellini et al. (2002), a política SITA-E é baseada em demonstrações teóricas, mas assume que o tempo de serviço de uma requisição é proporcional ao seu tamanho. Isso é válido apenas para conteúdo estático (na verdade, o problema de predeterminar o tempo de serviço de uma requisição dinâmica continua em aberto). Além disso, em relação aos arquivos pequenos, a política SITA-E desconsidera a influência de possíveis otimizações no sistema operacional e o uso de cache.

- Algoritmos Baseados no Cliente (*Client-Aware Policy – CAP*) (CASALICHIO; COLAJANNI, 2001): O princípio básico desse algoritmo é que quando a página *Web* oferece serviços heterogêneos, cada requisição pode desgastar um recurso diferente do *Web cluster*, por exemplo: CPU, disco ou rede. Segundo Cardellini et al. (2002), apesar do direcionador não poder estimar precisamente o tempo de serviço de uma requisição estática ou dinâmica ele pode distinguir a classe da requisição da URL e estimar o impacto dela nos recursos do sistema. Uma possível classificação para a política CAP é considerar serviços I/O-bound, CPU-bound e network-bound, mas outras opções são possíveis dependendo da disposição do conteúdo *Web*. Segundo Cardellini et al. (2002), o objetivo da política CAP é compartilhar inúmeros tipos de carga entre todos os *hosts* servidores sem sobrecarregar nenhum de seus recursos individuais. A política CAP em sua proposta inicial é baseada em informações de estado do cliente, entretanto ela pode facilmente ser combinada com algumas informações de estados dos *hosts* servidores.

Para Cardellini et al. (2002), políticas que estabelecem relação de *host* servidores do *Web cluster* com clientes podem usar informações mais detalhadas quando aplicadas sobre direcionadores de camada 7. Esse é o caso da política *client affinity* que associa todas as transações *Web* de um mesmo cliente para um mesmo *host* servidor, podendo utilizar informações mais refinadas para tomada de decisão quando aplicadas à camada 7. Identificadores de sessão, assim como *cookies* e identificadores SSL são comumente explorados pelos direcionadores atuais.

5.4.2 Políticas Baseadas em Informações do Cliente e do Servidor

Algoritmos de escalonamento implementados na camada 7 (camada de aplicação) também podem combinar informações de estado do cliente e do servidor. Nesta seção serão descritas duas políticas que foram projetadas para considerar as informações de estado de ambas entidades. Para Cardellini et al. (2002), outras políticas baseadas em informações do cliente (a política CAP por exemplo) permitem facilmente ser integradas com algumas informações do

servidor.

Na taxonomia apresentada na Figura 5.3, as políticas baseadas em informações do cliente e do servidor pertencem a ambas as classes de objetivos, pois elas utilizam informações do cliente para propósitos de *cache affinity* e informações do servidor para compartilhamento de carga. O algoritmo *Distribuição de Requisição Ciente de Localização* (*Locality-Aware Request Distribution* – LARD) é baseado no conteúdo das requisições e considera a localidade e o balanceamento de carga (PAI et al., 1998). O princípio básico do LARD é direcionar todos os objetos *Web* de uma mesma requisição HTTP para o mesmo *host* servidor, visando aumentar temporariamente a taxa de acertos em cache. Essa política oferece melhores resultados para páginas *Web* que oferecem informações estáticas e algumas simples informações armazenadas em banco de dados.

Faz-se necessário monitorar a utilização dos servidores para evitar sobrecarga e indiretamente, melhorar o compartilhamento de carga. Quando a utilização de um *host* servidor atinge seu limite, o algoritmo de escalonamento do direcionador associa as requisições para o próximo *host* com menor carga. Um esquema similar à política LARD é implementado no *HACC cluster* (ZHANG et al., 1999).

Segundo Bunt et al. (1999), no algoritmo LARD o direcionador é quem mantém o mapeamento de quais *hosts* contêm determinado arquivo, enquanto o gerenciador de cache é responsável pelo conteúdo em cache de todos os *hosts* servidores. Cada servidor fornece esta informação para o gerenciador de cache.

5.4.3 Considerações Sobre o Escalonamento Baseado em Conteúdo

Segundo Cardellini et al. (2002), políticas estritamente baseadas em informações de estado do cliente têm uma grande vantagem sobre as políticas baseadas em informações de estado do servidor, pois não necessitam de nenhum mecanismo para monitorar e medir a carga de cada *host* servidor, reunindo os resultados e combinando-os para a tomada de decisão. Entretanto, políticas baseadas em informações do cliente devem levar em consideração ao menos uma informação binária sobre o estado dos *hosts* para evitar que o roteamento de requisições fique temporariamente indisponível ou sobrecarregue os servidores.

Para Cardellini et al. (2002), algoritmos de particionamento estático tendem a ignorar o compartilhamento de carga, dessa forma tornando quase impossível particionar serviços baseados na *Web* de forma que as requisições fiquem balanceadas. Na verdade, uma característica bem conhecida das cargas de trabalho da *Web* é que um pequeno conjunto de arquivos são requisitados em grande parte das requisições.

Segundo Cardellini et al. (2002), *Web clusters* que oferecem serviços extremamente heterogêneos e utilizam políticas como a CAP (que visa a compartilhar a carga entre todos *hosts* servidores) podem oferecer melhor desempenho.

5.5 Considerações Finais

Uma taxonomia de algoritmos de escalonamento para direcionadores que operam na camada de rede (camada 4) e na camada de aplicação (camada 7) foi apresentada neste capítulo. Políticas baseadas em informações tanto do cliente quanto do servidor também foram apresentadas com diversos algoritmos.

Com o crescente aumento do surgimento de novas aplicações e serviços de alta complexidade voltados para *Web*, em conjunto com a integração entre os mesmos, faz-se necessário o estudo e avaliação de algoritmos de escalonamento aplicados à *Web clusters*.

Carga de Trabalho

6.1 Considerações Iniciais

Neste capítulo será descrito o processo realizado para geração de uma carga de trabalho sintética para um site de comércio eletrônico. A demanda de tráfego para aplicações *Web* do gênero tem aumentado a cada dia, uma vez que a utilização da Internet para comprar e vender produtos se torna cada vez mais presente na economia mundial.

Esse aumento na demanda justifica a utilização de uma estrutura com diversos servidores *Web* para atender o tráfego de requisições com desempenho aceitável pelos consumidores. Para a realização de uma avaliação de desempenho sem a participação de uma empresa do mercado, faz-se necessário reproduzir um cenário que retrate a realidade atual de sites de comércio eletrônico e gerar uma carga de trabalho sintética que retrate o mais próximo possível o atual comportamento dos consumidores de sites de comércio eletrônico.

Com base em trabalhos relacionados ao assunto um modelo de carga de trabalho para sites de comércio eletrônico foi desenvolvido. Esse modelo considera algumas das principais funcionalidades que caracterizam um site do gênero, destacando funcionalidades e seus respectivos impactos no balanceamento de carga em diversos servidores *Web*.

Utilizando-se a carga de trabalho definida, foram realizados experimentos em servidores *Web* reais. Os resultados e suas respectivas análises são apresentadas com valores médios de tempos de resposta com e sem o uso do recurso de memória *cache* no lado do cliente.

6.2 Comércio Eletrônico

Lojas virtuais de varejo no Brasil, como o Submarino, também assumidas neste trabalho como sites de comércio eletrônico, comercializam mais de 700 mil itens diferentes distribuídos em 24 categorias, segundo IDG-Now (2006), o que gera uma demanda de pelo menos 5 mil pedidos de produtos por dia.

Segundo E-Bit (2006), o comércio *online* brasileiro deve crescer 56% em 2006, movimento estimado em 3,9 bilhões de reais. Ao todo, foram ouvidas 500 lojas virtuais para compor dos dados. Desde que a entidade começou a medir o volume de vendas pela internet no Brasil, em 2001, o comércio *online* já cresceu 355% até o final de 2005. Somente em 2005, as vendas pela *Web* aumentaram 45% em relação a 2004, totalizando 2,5 bilhões de reais. O número de consumidores *online* cresceu na mesma proporção - 45% - atingindo o total de 4,7 milhões. Mas ainda há espaço para crescer. Nos Estados Unidos, por exemplo, dos 170 milhões de internautas, 100 milhões compram na *Web* (IDG-Now, 2006).

Cada seção de usuário que acessa uma loja virtual para realizar ou não um pedido é definida com uma sequência de interações da *Web*, que começam com um acesso a página principal do site. Neste trabalho, essas interações serão classificadas em duas categorias gerais:

- Categoria *Navegar* – envolvem navegação e pesquisa, mas nenhuma atividade de pedido de produtos. As interações típicas que abrangem essa categoria são: *Página Principal*, *Listar Categorias*, *Visualizar Detalhes* de produto e *Buscar produto*;
- Categoria *Pedir* – envolvem somente atividades de pedido de produtos, e incluem as seguintes interações: *Login*, *Local de Entrega*, *Forma de Pagamento* e *Confirmação* de pedido.

Interações da categoria *Pedir* têm maior probabilidade de gerar renda do que as de categoria *Navegar* e devem ter maior prioridade. Servidores *Web* para sites de comércio eletrônico em situações de sobrecarga, podem deixar de atender requisições que geram renda por conta de aumento excessivo na quantidade de requisições geradas por interações da categoria *Navegar* (com probabilidade mínima de gerar renda).

6.3 Modelo da Carga de Trabalho

Um dos problemas encontrados na simulação de sistemas relacionados à *Web* consiste na geração de uma carga de trabalho que reproduza um cenário o mais próximo da realidade atual.

Tendo em vista a dificuldade de encontrar um arquivo de registros com acessos reais de uma loja varejista *online*, decidiu-se gerar uma carga de trabalho sintética para a experimentação do modelo.

Em uma típica loja virtual de varejo, muitos usuários navegam e realizam buscas pelas páginas do site a procura de um produto específico (interações do tipo *Navegar*). Se encontrarem o produto desejado e tiverem intenção de comprá-lo, podem iniciar um processo de transação para a realização do pedido (interações do tipo *Pedir*). Existe ainda possibilidade de o usuário abandonar o site durante qualquer uma das interações.

Uma interação da categoria *Pedir* é constituída de um primeiro passo: a página de *Login*. Deste ponto em diante o usuário é orientado a seguir algumas etapas em seqüência (escolha do local de entrega da mercadoria e seleção da forma de pagamento) até culminar em uma requisição do tipo *Confirmação*, que finaliza o pedido.

A Figura 6.1 ilustra essas transições entre as diferentes categorias de interações (*Navegar* e *Pedir*) dos usuários pelo site, representadas por uma máquina de estados finitos, onde as probabilidades são estacionárias e as transições de estados sem memória (SINGHMAR et al., 2004). Os estados correspondem às páginas que caracterizam as interações e os arcos representam transições. Para Singhmar et al. (2004), as linhas pontilhadas são assumidas como probabilidade zero.

Figura 6.1: Máquina de estados finitos (SINGHMAR et al., 2004)

Sabe-se que a probabilidade máxima de transição entre um estado e outro é 1. Para representar maior riqueza de detalhes, os valores das probabilidades nas linhas pontilhadas foram calculados baseados na subtração do valor máximo de transição dos valores das linhas contínuas pela quantidade de possíveis transições para outros estados.

Na Figura 6.1, observa-se que a soma das probabilidades de transição de um estado para outros não atingem a totalidade. Por exemplo, a soma da probabilidade de transição do estado

Principal para os estados *Categorias*(0, 4) e/ou *Buscas*(0, 5) é de 0, 9. Existe ainda uma probabilidade de 0, 1 chances de transição do estado *Principal* para qualquer outro estado. Nesse caso, o único estado que pode ser assumido é o de *Detalhes*.

Outro exemplo é do estado *Detalhes*. A diferença da probabilidade máxima de transição entre estados da soma de probabilidades possíveis representada na máquina de estados da figura 6.1 é de 0, 9. Desta forma, assume-se que a probabilidade de transição do estado *Detalhes* é de 0, 3 para cada um dos três possíveis outros estados que podem assumir.

A Figura 6.2 ilustra uma nova abordagem para a máquina de estados finitos apresentada por Singhmar et al. (2004), porém, assumindo (empiricamente) probabilidades de transição de estados das linhas pontilhadas com valores diferentes de zero.

Figura 6.2: Máquina de estados proposta para modelar a carga de trabalho

Assumindo valores para as linhas pontilhadas, pode-se calcular as probabilidades de acesso de cada estado na máquina. Somar todos os valores de probabilidade de assumir um determinado estado e calcular qual porcentagem esse valor representa na soma total de probabilidade de assumir todos os estados. Isto é, representa a porcentagem total de acesso que cada estado pode assumir ou que cada tipo de requisição pode receber. As Tabelas 6.1 e 6.2 apresentam as porcentagens de acesso para os diferentes tipos de interações e páginas.

INTERAÇÕES DO TIPO NAVEGAR (90%)		
	Probilidades de Acesso	Porcentagem de Acesso
<i>Principal</i>	0,9	23%
<i>Detalhes</i>	0,9	23%
<i>Buscas</i>	1,15	30%
<i>Categorias</i>	0,95	24%
Total	3,9	100%

Tabela 6.1: Porcentagens de acesso para interações do tipo *Navegar*

INTERAÇÕES DO TIPO PEDIR (10%)		
	Probabilidades de Acesso	Porcentagem de Acesso
<i>Login</i>	0,9	25%
<i>Entrega</i>	0,9	25%
<i>Pagamento</i>	0,9	25%
<i>Confirmação</i>	0,9	25%
Total	3,6	100%

Tabela 6.2: Porcentagens de acesso para interações do tipo *Pedir*

Como a probabilidade de acesso para as interações do tipo *Pedir* têm peso 0,1 e as interações do tipo *Navegar* têm peso 0,9, as porcentagens para a carga de trabalho total foram recalculadas.

CATEGORIAS DE INTERAÇÃO		
	Probabilidade de Acesso	Porcentagem de Acesso
<i>Navegar</i>	0,9	90%
<i>Pedir</i>	0,1	10%
Total	1	100%

Tabela 6.3: Probabilidades de acesso para interações de *Navegar* e *Pedir*

DISTRIBUIÇÃO DA CARGA DE TRABALHO	
<i>Principal</i>	21,00%
<i>Detalhes</i>	20,00%
<i>Buscas</i>	27,00%
<i>Categorias</i>	22,00%
<i>Login</i>	2,50%
<i>Entrega</i>	2,50%
<i>Pagamento</i>	2,50%
<i>Confirmação</i>	2,50%
Total	100,00%

Tabela 6.4: Distribuição dos tipos de requisições da carga de trabalho.

Com o modelo de distribuição de tipos de requisições apresentado neste trabalho, permite-se gerar um carga de trabalho sintética que represente o comportamento atual de usuários e clientes de uma loja virtual varejista. Esse modelo de carga de trabalho foi utilizado para a experimentação do modelo de simulação descrito na próxima seção.

6.4 Cenário Adotado

Com o modelo de carga de trabalho definido na seção anterior, decidiu-se por realizar experimentos reais e coletar tempos médios de resposta para cada categoria de página que recebe requisições Web. Com esses experimentos pode-se obter o comportamento de um sistema

Web real e encontrar valores para parametrizar as simulações de políticas de atendimento de requisições.

Para obtenção de valores médios de tempos de resposta para cada tipo de requisição que compõe o modelo de carga de trabalho definido, seria necessário analisar registros de servidores *Web* reais que disponibilizassem esse tipo de informação. Tendo em vista a dificuldade de encontrar arquivos de registros do gênero, optou-se por reproduzir um cenário real o mais próximo da realidade atual, gerar as informações necessárias e realizar uma posterior análise.

- um software servidor *Web*;
- um software servidor de banco de dados;
- uma aplicação que implemente as mesmas funcionalidades de uma loja varejista de produtos na Internet;
- computadores interligados em rede.

Uma análise da viabilidade de desenvolver uma aplicação do gênero que implemente as funcionalidades de uma loja virtual de varejo foi realizada. Tendo em vista a necessidade de implementar um modelo de dados não trivial para armazenar informações de produtos em uma base de dados, dominar uma linguagem de programação para Web com o uso de sessão, *cache*, entre outras funcionalidades e popular uma base de dados com informações coerentes e equivalentes aos de um catálogo de uma loja virtual real, concluiu-se que o tempo necessário para o desenvolvimento de tal aplicação é inviável e optou-se pela busca de uma solução já existente na Internet.

A *osCommerce* foi a solução Web encontrada e pode ser encontrada no site do desenvolvedor (LEON et al., 2006). Desenvolvida em PHP (<http://www.php.net>) e utilizando o banco de dados MySQL (site do fabricante: <http://www.mysql.com>), essa solução de código aberto e com licença GNU/GPL (*General Public License*), implementa um modelo de loja varejista que vende produtos na Internet. Oferece funções de comércio eletrônico que permite aos consumidores navegarem por categorias de produtos, procurarem informações sobre produtos existentes, verem detalhes de produtos, realizarem pedidos e escolherem local de entrega e forma de pagamento.

As interações relacionadas ao pedido de produtos são criptografadas por meio de conexões SSL (*Secure Socket Layer*). Os clientes precisam se registrar no site antes de terem permissão para comprar. O site mantém um catálogo de itens que podem ser pesquisados por

um cliente. Cada item possui uma descrição textual e uma imagem miniatura associada com tamanho inferior a 5 KBytes.

Figura 6.3: Cenário utilizado para a realização de experimentos

Para suprir as necessidades relacionadas a software para reprodução do cenário, utilizou-se o software Apache (servidor Web) e MySQL (servidor de banco de dados). A solução *Web osCommerce* completa a lista de recursos necessários.

A Figura 6.3 ilustra os três computadores do Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) que foram utilizados para a realização dos experimentos. Um computador (identificado como *Cliente*) para gerar requisições de clientes Web, outro configurado apenas como um *Servidor Web* (para registrar os tempos de atendimento) e um terceiro configurado como um servidor de aplicações, responsável pelas operações de acesso ao *Banco de Dados*. Todos os computadores estão interligados por uma rede padrão *Gigabit Ethernet* e possuem processadores AMD Athlon XP com 3.2 GHz de frequência, 1 GByte de memória RAM e disco rígido de 7200 rotações por minuto com capacidade de armazenamento de 80 GBytes.

6.5 Instrumentação para Experimentos

Servidores Web registram em arquivos texto (*logs*) eventos de sucesso e erros associados às requisições de clientes. Cada registro traz informações relacionadas à requisição atendida, tais como: data e hora de atendimento, nome do arquivo, método pelo qual realizou a solicitação, quantidades de bytes enviados, tempo total de atendimento etc. Com base nessas informações, administradores de sistema utilizam ferramentas para avaliar, por exemplo, o desempenho do sistema.

Uma informação importante para uma melhor análise de componentes específicos do sistema é o tempo total de consumo de CPU de cada requisição Web atendida. O servidor Web Apache disponibiliza apenas o tempo total de atendimento de cada requisição. Assim,

novas funcionalidades foram disponibilizadas no software servidor Web Apache por meio de instrumentação do código fonte. Uma nova função que coleta o tempo atual de consumo de CPU de uma requisição Web em um processo do servidor em execução foi adicionada neste trabalho, utilizando a chamada *clock()* do sistema operacional Linux.

Na estrutura de uma requisição do Apache existe um campo denominado *"request_time"*, designado para armazenar o tempo inicial de atendimento de uma requisição. Desta forma, utilizando a mesma analogia, adicionou-se um novo campo denominado *"request_time_cpu"* para armazenar o tempo inicial de CPU da requisição.

Dentre as diversas etapas que uma requisição passa para ser atendida pelo servidor Web Apache, a primeira é a atribuição dos dois tempos supra citados e a última é o registro de informações relacionados ao atendimento em arquivo texto. A estrutura do código fonte do servidor Web Apache é particionado em módulos que implementam partes diferentes das funcionalidades do software. Alguns módulos são responsáveis por disponibilizar recursos de acesso a disco, outros por controlar *threads* e processos servidores e outros por registrar informações de atendimento de requisições em arquivos texto, cuja tarefa é de responsabilidade do módulo *mod_log_config*.

Nesse módulo foi implementada uma função para coletar o tempo atual de consumo de CPU, que em seguida é subtraído do tempo inicial de CPU do campo *"request_time_cpu"*. Esse cálculo resulta no tempo total de consumo de CPU de cada requisição em *ticks* do relógio do sistema. Dividindo esse valor pela macro *CLOCKS_PER_SEC* obtém-se o número de segundos consumidos para processar uma requisição.

6.6 Resultados dos experimentos

Dados de atendimentos de requisições Web foram coletados durante intervalos de tempo e, posteriormente, utilizados para modelar o comportamento de um sistema Web composto por vários servidores.

Uma vez extraído do servidor Web o tempo total de atendimento de uma requisição e o tempo total de consumo de CPU, pode-se calcular o tempo consumido em entrada e saída (I/O). Esse tempo ($T_{r,io}$) é obtido através da equação 6.1 onde: r representa a requisição, $T_{r,total}$ o tempo total de atendimento de uma requisição r , e $T_{r,cpu}$ o tempo total de consumo de CPU.

$$T_{r,io} = T_{r,total} - T_{r,cpu} \quad (6.1)$$

O tempo de entrada e saída inclui o tempo de acesso a disco local e acesso a banco

de dados. No caso, como o banco de dados está em outro computador, o tempo de entrada e saída inclui, também, os atrasos de rede. A Figura 6.4 ilustra as informações de cada requisição atendida pelo software servidor Web, disponibilizadas para análise posterior.

```
143.107.232.210 15-01-2006 00:25:06 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29828 101687 60000
143.107.232.210 15-01-2006 00:25:12 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29570 90620 80000
143.107.232.210 15-01-2006 00:25:21 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29811 97758 60000
143.107.232.210 15-01-2006 00:25:26 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29777 89052 70000
143.107.232.210 15-01-2006 00:25:31 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29764 93592 60000
143.107.232.210 15-01-2006 00:25:36 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29811 90494 70000
143.107.232.210 15-01-2006 00:25:42 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29813 97452 90000
143.107.232.210 15-01-2006 00:25:47 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29713 90698 70000
143.107.232.210 15-01-2006 00:25:52 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29781 98282 100000
143.107.232.210 15-01-2006 00:25:58 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29727 89244 80000
143.107.232.210 15-01-2006 00:26:03 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29801 90005 80000
143.107.232.210 15-01-2006 00:26:08 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29765 97648 100000
143.107.232.210 15-01-2006 00:26:13 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29774 89951 80000
143.107.232.210 15-01-2006 00:26:18 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29826 90128 80000
143.107.232.210 15-01-2006 00:26:23 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29765 97894 100000
143.107.232.210 15-01-2006 00:26:28 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29816 90101 80000
143.107.232.210 15-01-2006 00:26:34 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29801 89870 70000
143.107.232.210 15-01-2006 00:27:03 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29786 92049 70000
143.107.232.210 15-01-2006 00:27:08 GET /tmp/caio/www/develop/apache/htdocs/index.php HTTP/1.1 200 29798 89668 70000
```

Figura 6.4: Exemplo de arquivo de registros do servidor Web

Páginas *Web* dinâmicas, em relação a páginas *Web* estáticas, têm como diferencial um maior consumo de recursos de CPU e o fato de não terem seu conteúdo armazenados em memória *cache*. Operações de acesso a base de dados são caracterizadas por exigirem maior demanda de dispositivos de I/O.

Como observado nas Tabelas 6.5 e 6.6, o tempo de consumo de CPU das requisições é maior que o tempo de consumo de dispositivos de I/O. Isso se deve não somente pelo fato da base de dados estar presente em um servidor de aplicações (separado do servidor *Web*), mas também pela presença do recurso de *cache* do sistema operacional no *host* servidor *Web*.

	Valores COM CACHE no cliente				
	Tempos de Resposta			Porcentagens	
	Total	CPU	I/O	CPU	I/O
Principal	0,088253	0,070430	0,017823	79,8046%	20,1954%
Detalhes	0,086097	0,063750	0,022347	74,0442%	25,9558%
Buscas	0,115783	0,094333	0,021450	81,4741%	18,5259%
Categorias	0,096115	0,076333	0,019782	79,4186%	20,5814%
Login	0,046445	0,041331	0,005114	88,9887%	11,0113%
Entrega	0,056348	0,048927	0,007421	86,8292%	13,1708%
Pagamento	0,105471	0,083768	0,021704	79,4220%	20,5780%
Confirmação	0,112506	0,063000	0,049506	55,9971%	44,0029%

Tabela 6.5: Valores obtidos COM o recurso de *cache* no cliente.

Arquivos estáticos, acessados com maior frequência (imagens, folhas de estilos, etc) no servidor *Web*, tendem a permanecer armazenados em memória *cache* gerenciada pelo sistema operacional onde executa o servidor *Web*. Dessa forma, ao contrário do que se espera, são resgatados da memória e não de um dispositivo de armazenamento. A ferramenta utilizada para medir o tempo de consumo de CPU, classifica acessos à memória como operações de CPU e

	Valores SEM CACHE no cliente				
	Tempos de Resposta			Porcentagens	
	Total	CPU	I/O	CPU	I/O
Principal	0,091791	0,072928	0,018863	79,4504%	20,5496%
Detalhes	0,087716	0,073754	0,013962	84,0830%	15,9170%
Buscas	0,121153	0,096474	0,024679	79,6301%	20,3699%
Categorias	0,098332	0,082080	0,016252	83,4725%	16,5275%
Login	0,076757	0,041000	0,035757	53,4153%	46,5847%
Entrega	0,125837	0,083333	0,042504	66,2232%	33,7768%
Pagamento	0,140452	0,072333	0,068118	51,5005%	48,4995%
Confirmação	0,144596	0,058667	0,085929	40,5729%	59,4271%

Tabela 6.6: Valores obtidos SEM o recurso de *cache* no cliente.

mede apenas o tempo de atendimento no servidor *Web*. O tempo consumido por uma requisição atendida no servidor *Web* para realizar operações de acesso a base de dados é considerado nos experimentos como tempo de I/O.

Para cada arquivo (página *Web* do servidor), foram geradas requisições seguidas com intervalos de 1 segundo até convergir em torno da média de uma distribuição Normal segundo o teorema do limite central ??, permitindo o uso de medidas estatísticas de resumo para representar a ocupação de recurso, posteriormente utilizada na modelagem do problema.

Cada requisição atendida no servidor *Web* Apache gera uma linha no arquivo de registros com as informações: nome do arquivo, tempo total de atendimento, tempo total de consumo de CPU e tempo total de consumo de recursos de I/O.

Caso o cliente *Web* utilize *cache* local (opcional e configurável no navegador), objetos que compõem as páginas *Web* podem ou não ser requisitados para o servidor, o que gera economia de recursos. Como as páginas atendidas pelo servidor são dinâmicas e compostas por pequenos arquivos (com tamanho inferior a 5 KBytes) estáticos embutidos, essa economia se torna desprezível.

Em requisições da categoria *Navegar*, o ganho de desempenho COM ou SEM o uso do recurso de *cache* no cliente se torna desprezível. Por outro lado, quando se trata de interações do tipo *Pedir*, os efeitos de usar ou não o recurso de *cache* no cliente apresenta diferença significativa no tempo total de atendimento da requisição no servidor, ocasionada pelo aumento no tempo de I/O da requisição. Isso se deve ao fato de requisições da categoria *Pedir* serem criptografadas por meio de conexões SSL (*Secure Socket Layer*).

Sem o recurso de *cache* no cliente, a chave de segurança da respectiva seção do cliente deverá ser reenviada ao cliente a cada requisição. Desta forma, o servidor executará novamente

o procedimento de *handshaking* (troca de certificados, negociação de compressão e encriptação, configuração de identificador de sessão). Esse procedimento é realizado durante o atendimento de uma requisição e pode aumentar em escalas muito maiores o tempo de I/O, conforme a largura de banda da rede que interliga o cliente e o servidor (GOLDBERG et al., 1998).

De acordo com os tipos de requisições que caracterizam um *site* de comércio eletrônico descritas na seção anterior, foram realizados experimentos em ambiente real para obtenção de informações relacionadas ao tempo total de atendimento de uma requisição (no software servidor Web), tempo de consumo de CPU e tempo de I/O. Os resultados obtidos são valores médios alcançados por meio da geração de quantidades diferentes de requisições seguidas, com intervalo de 1 segundo entre cada uma. Esses valores se encontram em um intervalo de confiança de 95%.

6.7 Considerações Finais

Neste capítulo foi proposto um modelo de carga de trabalho Web para sites de comércio eletrônico, especificamente de varejo. Experimentos foram realizados reunindo as principais funcionalidades que caracterizam um site do gênero e alguns resultados foram extraídos.

Conclui-se que, devido ao fato das páginas Web que compõem os sites de comércio eletrônico serem geradas dinamicamente, apenas os objetos embutidos podem ser armazenados em *cache* no lado do cliente. Esses objetos por sua vez, são arquivos de tamanho inferior a 5KB e implicam em baixa economia de utilização de recursos nos servidores Web.

Nota-se ainda que as requisições apresentadas nas Tabelas 6.5 e 6.6 apresentam um perfil com maior consumo de recursos de CPU. Fato esse justificado pelo recurso de *cache* do sistema operacional do servidor Web. Como a variedade de arquivos freqüentemente acessados e o tamanho em bytes são relativamente pequenos, a taxa de acerto em *cache*, no lado do servidor é alta.

Uma análise da viabilidade de realizar os experimentos com e sem o recurso de *cache* no lado do servidor foi realizada. Tendo em vista que esse recurso é implementado no kernel do sistema operacional, no caso o Linux e que sempre estará presente tanto em ambientes de produção quanto em ambientes em execução, decidiu-se por considerar o recurso de *cache* no servidor como padrão.

No próximo capítulo serão descritos os experimentos realizados para a validação de um modelo de servidor de Web com simulação que utiliza o modelo de carga de trabalho proposto neste capítulo.

Avaliação de Desempenho

7.1 Considerações Iniciais

Neste capítulo são apresentados os resultados obtidos com a aplicação de algoritmos de balanceamento de carga e disciplinas de atendimento para filas na arquitetura do Servidor *Web* para *E-Commerce* (SWE-C). Utilizou-se uma carga de trabalho sintética gerada a partir de resultados obtidos e discutidos no capítulo anterior.

Analizou-se o emprego de dois algoritmos de balanceamento de carga, *Round Robin* (RR) e *Shortest Queue First* (SQF), no componente Escalonador da arquitetura, cuja função é descrita na Seção 3.3.1. Em conjunto com tais algoritmos, empregou-se ainda três disciplinas de atendimento de requisições nas filas de CPU de cada *host* servidor do *cluster*: *First In First Out* (FIFO), *Last In First Out* (LIFO) e uma terceira disciplina, chamada neste trabalho de LIFO-Pri, que utiliza prioridades baseada em LIFO.

Nos experimentos realizados, foi simulado um *cluster* com oito servidores web e um escalonador. O objetivo é verificar o comportamento do sistema com os algoritmos RR e SQF no balanceamento de carga e as disciplinas FIFO, LIFO e LIFO-Pri nas filas de CPU. Os resultados apresentados foram obtidos a partir de média de trinta simulações com um intervalo de confiança de 95%.

Nas próximas seções serão discutidos os algoritmos e disciplinas em supra, bem como o mecanismo de prioridades proposto para um site de *e-commerce*.

7.2 O Modelo Proposto

Para Texeira (2004), os servidores Web atuais tratam as requisições que recebem segundo uma disciplina FIFO, 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 para *E-Commerce* (SWE-C). Esse modelo, uma contribuição original desta dissertação, representa uma arquitetura de servidor Web cujo objetivo é atender requisições da categoria *Pedir* com prioridade sobre as requisições da categoria *Navegar*, conforme descrito na Seção 6.2, uma vez que, a cada etapa que o visitante avança no processo de compra, maior a probabilidade de confirmação de pedido.

O modelo proposto apresenta as características mais relevantes de um sistema real. A arquitetura do servidor SWE-C possui dois componentes: um Escalonador e um *Cluster* de Servidores Web. O Escalonador é o componente responsável por admitir as requisições que chegam e atribuí-las a um dos *hosts* servidores Web do *cluster*, segundo um algoritmo de balanceamento de carga (RR ou SQF). Cada *host* servidor é modelado com um recurso que represente o consumo de CPU e um outro recurso que represente o consumo de I/O, conforme descritos na Seção 6.6. A figura 7.1 ilustra o modelo proposto.

Figura 7.1: Modelo do Servidor Web para *E-Commerce* (SWE-C).

As requisições, após serem admitida no sistema, são inserida na fila de CPU do *host* selecionado e atendidas conforme a disciplina de fila vigente (FIFO, LIFO ou LIFO-Pri). O tempo de atendimento de CPU é estabelecido segundo uma distribuição exponencial com valores mé-

dios baseados nas Tabelas 6.5 e 6.6, conforme o uso ou não do recurso de *cache* no cliente. Em seguida, a requisição passa por uma fila de I/O, que segue uma disciplina FIFO e também assume tempos de atendimentos baseados nas Tabelas 6.5 e 6.6. Finalizado o atendimento, os resultados são retornados ao cliente que originou a solicitação.

Com base no modelo apresentado, utilizou-se a técnica de simulação para realizar experimentos com uma carga de trabalho sintética, gerada a partir de resultados obtidos no capítulo anterior.

7.3 Metodologia para Experimentos

A abordagem utilizada para a validação do modelo foi a simulação, por permitir verificar diferentes configurações do modelo e devido à larga experiência do Grupo de Sistemas Distribuídos e Programação Concorrente (LaSDPC). É implementado um modelo do servidor SWE-C, conforme descrito na Seção 7.2, com oito servidores Web e um Escalonador. Cada *host* do *cluster* é composto por CPU e dispositivo de I/O, cada qual com uma fila de atendimento. Assume-se que todos os *hosts* servidores do *cluster* têm acesso aos mesmos arquivos.

O tempo de CPU gasto inclui o trabalho realizado pelo sistema de arquivos, o sistema de rede e outras atividades, como tratamento de interrupção, programação da execução e gerenciamento de memória. Neste trabalho, assume-se que o tempo de I/O gasto inclui o tempo de acesso a informações armazenadas no servidor de banco de dados (incluindo latência da rede) e o acesso a disco local.

Para a geração de uma carga sintética foi utilizada a Tabela 6.4 para distribuir o tráfego entre os tipos de requisições discutidas na Seção 6.2. Antes de chegar ao sistema, cada requisição gerada recebe um tipo, um tempo de consumo de CPU e um de consumo de I/O, além de uma taxa de acerto em *cache* no cliente.

Acertos em Cache (Cliente)	
Principal	30%
Detalhes	30%
Buscas	30%
Categorias	30%
Login	–
Entrega	100%
Pagamento	100%
Confirmação	100%

Tabela 7.1: Taxas de acerto em *cache* no cliente.

No programa de simulação, antes de processar cada requisição, um número gerado ale-

atoriamente é comparado a probabilidade de acerto em *cache* da requisição em questão. Desta forma, o tempo de consumo de CPU e I/O assume valores da da Tabela 6.5 ou da Tabela 6.6. A probabilidade de acerto em *cache* para cada tipo de requisição é apresentada na Tabela 7.1.

A taxa de acerto em *cache* para requisições da categoria *Navegar* é fixada em 30%, segundo SPECweb2005 (2005). Já as requisições da categoria *Pedir* possuem taxas de acerto diferenciadas. Isso se deve pelo fato de requisições desta categoria serem criptografadas por meio de conexões SSL (*Secure Socket Layer*) e apresentarem algumas particularidades em relação ao uso do recurso de memória *cache* no cliente.

Neste trabalho considera-se que o procedimento de *handshaking* (troca de certificados, negociação de compressão e encriptação, configuração de identificador de sessão) é executado apenas em requisições do tipo *Login* e que a chave gerada permanece armazenada no *cache* local do cliente. Isso significa que, para todas as requisições do tipo *Login* serão utilizados os tempo de consumo de CPU e I/O da Tabela 6.6, uma vez que não existe probabilidade do cliente possuir uma chave de sessão válida que o permite utilizar objetos embutidos armazenados em *cache* local.

Tendo em vista que as requisições dos tipos: *Entrega*, *Pagamento* e *Confirmação* não repetirão o procedimento de *handshaking*, atribui-se a elas os tempos de consumo de CPU e I/O da Tabela 6.5. Mesmo que, por motivos adversos, após a etapa de *Login* o conteúdo da memória *cache* do cliente possa ser apagado, assume-se uma taxa de acerto em *cache* de 100% das requisições de tipos supra citadas.

7.3.1 Algoritmos de Balanceamento de Carga

Dois algoritmos de escalonamento com função de balanceamento de carga foram utilizados nas simulações: *Round Robin* (RR) e *Shortest Queue First* (SQF), onde ambos são classificados em duas categorias distintas respectivamente: *estático* e *dinâmico*. O Capítulo 5 desta dissertação apresenta maiores detalhes a respeito da aplicação e características específicas de algoritmos de escalonamento.

O algoritmo estático RR aloca requisições aos *hosts* servidores do *cluster* segundo um esquema de rodízio, um após o outro. Já o algoritmo dinâmico SQF, por sua vez, aloca a requisição para o *host* com o menor número de processos em fila, procurando, assim, atingir um melhor balanceamento de carga.

Neste trabalho, o algoritmo SQF considera o número de processos em fila de CPU. Para tanto, exige informações precisas sobre o tamanho da fila de CPU de cada *host* servidor do *cluster* a cada momento, o que segundo Texeira (2004), ocasiona maior sobrecarga no escalo-

namento.

7.3.2 Disciplinas para Fila

Segundo Allen (1990), sistemas de filas têm sido utilizados em diversas áreas da Computação, dentre as quais se destaca a aplicação em sistemas operacionais e redes de computadores. Softwares servidores Web também utilizam tal sistema, uma vez que apenas uma requisição pode ser atendida por vez, formando-se filas de requisições a serem atendidas.

Para estabelecer a ordem de atendimento de processos em filas, utilizam-se disciplinas com características específicas, conforme o caso em que se aplicam. O software servidor Web Apache, atualmente o mais utilizado na Internet (Netcraft, 2006), aplica a disciplina FIFO em sua fila de atendimento, para estabelecer a ordem em que as requisições Web são processadas.

Tendo em vista tais argumentos, a seguir são descritas duas disciplinas de fila bastante conhecidas na literatura e uma nova disciplina, desenvolvida como consequência da realização de experimentos:

- *FIFO* (First In First Out): requisições são atendidas na ordem de chegada;
- *LIFO* (Last In First Out): a última requisição a chegar é a primeira a ser atendida;
- *LIFO-Pri* (Last In First Out with Priority): as últimas requisições a chegarem são avaliadas e atendidas segundo suas respectivas prioridades.

7.3.3 O Mecanismo de Prioridades

Para Singhmar et al. (2004), o objetivo de um site de *e-commerce* é gerar renda, que pode ser alcançada pela navegação de visitantes entre os produtos oferecidos no site até a efetivação da compra.

Embora grande parte dos visitantes não tenha intenção de comprar, é importante ressaltar que quando exercerem tal intenção e iniciarem o processo de compra devem ser orientados a completar a transação sem estourar um tempo limite (normalmente estabelecido por medidas de segurança). Isto é especialmente importante durante condições de sobrecarga, quando os servidores Web apresentam aumento no tempo de atendimento, ou seja, caso o tempo de resposta do sistema ultrapasse o tempo limite para completar uma transação, nenhuma transação poderá ser concluída com sucesso até que o tráfego do sistema diminua.

No processo de compra, as requisições geradas pelos clientes pertencem a categoria *Pedir*. Desta forma, optou-se por atribuir maiores prioridades para os tipos de requisições desta

categoria (*Login*, *Entrega*, *Pagamento* e *Confirmação*) em relação ao tipos de requisições da categoria *Navegar* (*Principal*, *Detalhes*, *Buscas* e *Categorias*). A introdução de prioridades foi aplicada nas filas de CPU de cada *host* servidor *Web* do *cluster*.

Os Critérios Adotados

Para definir quais tipos de requisições devem ter maior prioridade sobre as demais, faz-se necessário adotar critérios para justificar tais prioridades. Surgiram duas opções para adotar como critério na diferenciação do atendimento de requisições por tipos: um critério orientado ao negócio e um critério orientado ao consumo de recursos do sistema.

Em Singhmar et al. (2004), um mecanismo de prioridades orientado ao negócio foi utilizado. As prioridades foram estabelecidas segundo a relevância que cada tipo de requisição representa na lógica do negócio, no caso, uma loja virtual de *e-commerce*.

Conforme descrito na Seção 6.6, percebe-se que o comportamento de todos os tipos de requisições exigem maior consumo de CPU. Assim, conclui-se que o ponto de saturação do sistema em situações de sobrecarga tende a ser a CPU dos *hosts* servidores *Web* do *cluster*.

Tipo	Prioridade
<i>Principal</i>	3
<i>Detalhes</i>	4
<i>Buscas</i>	1
<i>Categorias</i>	2
<i>Login</i>	5
<i>Entrega</i>	6
<i>Pagamento</i>	7
<i>Confirmação</i>	8

Tabela 7.2: Prioridades para cada tipo de requisição.

Neste trabalho, para atribuir prioridades entre os tipo de requisições da categoria *Navegar*, decidiu-se utilizar o critério orientado ao consumo de recursos, especificamente o recurso CPU. Para calcular os valores da prioridades, os tempos de consumo de CPU apresentados nas Tabelas 6.5 e 6.6 (que são equivalentes) foram ordenados de forma decrescente e as prioridades atribuídas em ordem crescente. Assumiu-se ainda que qualquer tipo de requisição da categoria *Pedir* tem maior prioridade do que todos os tipos de requisições da categoria *Navegar*, pois tendem a gerar lucro para o negócio.

Conforme descrito no capítulo anterior, dois tempos médios de resposta foram obtidos nos experimentos: tempo com *cache* e tempos sem *cache* no cliente. Em ambos os casos, o tempo de consumo de CPU é equivalente. A Tabela 7.2 apresenta as prioridades para cada tipo

de requisição, conforme o consumo de CPU.

7.4 Resultados Experimentais

Simulações foram realizadas com o objetivo de avaliar o desempenho do sistema quando submetido a diferentes intensidades de carga com diversas combinações de algoritmos de escalonamento no componente escalonador e disciplinas de atendimento nas fila de CPU de cada *host* servidor *Web*, conforme descrito nas seções anteriores.

Os experimentos implementam o modelo SWE-C para simular o atendimento de requisições *Web* por um período de uma hora (3.600 segundos). Todos os gráficos possuem o tempo de simulação no eixo X. Os resultados apresentados foram obtidos a partir de média de cinco simulações com um intervalo de confiança de 95%.

7.4.1 Intensidades de Carga

Para Vallamsetty et al. (2003) a baixa utilização de recursos é típica em sites de *e-commerce*. Sistemas são projetados para sustentarem uma carga baixa durante a maior parte do tempo e resistirem a períodos de sobrecarga com desempenho aceitável.

Utilização Média de CPU		
Taxa de Chegada	Intensidade	Utilização
70 req/seg	Baixa	70%
100 req/seg	Alta	98%
105 req/seg	Total	100%

Tabela 7.3: Utilização média de CPU do sistema sob diferentes intensidades de carga.

Optou-se por submeter o sistema a três diferentes intensidades de carga: baixa, alta e total, afim de representar as situações que acontecem em sites reais de *e-commerce*. A Tabela 7.3 e as Figura 7.2, 7.3 e 7.4 apresentam a taxa de utilização de CPU alcançada nos *hosts* servidores *Web* do *cluster* ao longo de uma hora de tráfego.

A principal medida de desempenho utilizada é o tempo de resposta necessário para o sistema atender cada requisição, ou seja, desde que o instante em que ela é aceita no sistema até o momento em que a mesma é enviada de volta ao cliente. O tempo de resposta é composto pelo tempo de serviço e o tempo de espera em fila. Em situações de sobrecarga do sistema, a tendência é que o tempo de fila aumente rapidamente, constituindo-se na parcela dominante para o cálculo do tempo de resposta total para o cliente. Para analisar com maior riqueza de detalhes o comportamento do sistema, utilizou-se também as medida de desempenho a seguir:

- *Throughput*: quantidade de requisições *Web* atendidas por segundo pelo sistema;

Figura 7.2: Utilização média de CPU (intensidade de carga baixa - 70 req/seg).

Figura 7.3: Utilização média de CPU (intensidade de carga alta - 100 req/seg).

Figura 7.4: Utilização média de CPU (intensidade de carga total - 105 req/seg).

- *Taxa de Utilização de CPU*: representa a porcentagem de tempo que o recurso de CPU foi utilizado em relação ao tempo total de simulação;
- *Tamanho da Fila de CPU*: quantidade de requisições que aguardam ser atendidas na fila de CPU.

Os gráficos resultantes dos experimentos, suas respectivas análises, vantagens e desvantagens serão discutidos a seguir em seções separadas conforme a intensidade de carga a qual o sistema fora submetido.

Intensidade de Carga Baixa

Inicialmente, o sistema foi submetido a uma carga de baixa intensidade para analisar se o comportamento da combinação LIFO-Pri é ineficiente em períodos de não sobrecarga, comparado a outras combinações que não utilizam mecanismos de prioridades.

A Figura 7.5 ilustra o tempo médio de resposta do sistema quando submetido a uma carga de baixa intensidade. A taxa de chegada de requisições, que segue uma distribuição exponencial é de 70 requisições por segundo. Com isso, o sistema atinge uma taxa de utilização de recursos de CPU de aproximadamente 70% do total.

Nota-se que a combinação RR/LIFO-Pri apresenta o melhor tempo médio de resposta com ganho aproximado de 4,1% e 2,5% de desempenho em relação as combinações RR/FIFO e RR/LIFO. Já a combinação SQF/LIFO-Pri apresenta um desempenho equivalente às combi-

Figura 7.5: Tempo médio de resposta do sistema (intensidade de carga baixa - 70 req/seg).

nações SQF/FIFO e SQF/LIFO. Isso mostra que mesmo em casos de não sobrecarga, o uso da disciplina LIFO-Pri apresenta pequena melhoria no desempenho.

O *throughput* de requisições atendidas com sucesso pelo sistema é apresentado na Figura 7.6. Após 1.000 segundos de tráfego, nota-se que os valores se estabilizam e mantêm-se constantes ao longo dos 3.600 segundos de simulação. Observa-se também que a porcentagem de requisições atendidas em relação a quantidade de requisições que chegam é igual ou superior a 99,9%. Isso faz com que a diferença de *throughput* entre as diversas combinações se torne desprezível.

Durante os experimentos, foram monitoradas também as filas de CPU dos *host* servidores do *cluster*. Constatou-se que o tamanho médio de requisições em fila de CPU com intensidade de carga baixa é nula em todas as combinações, ou seja, o sistema consegue atender a demanda de tráfego em sua totalidade.

Figura 7.6: Requisições atendidas por segundo (intensidade de carga baixa - 70 req/seg).

Intensidade de Carga Alta

Com o objetivo de analisar o comportamento do sistema em situações de alto tráfego, porém sem ultrapassar a total capacidade de atendimento do mesmo, submeteu-se o sistema a uma intensidade de carga alta. A taxa de utilização média de CPU do sistema fica em torno de 98%.

Com uma taxa de chegada de aproximadamente 100 requisições por segundo, a porcentagem do aproveitamento no atendimento de requisições em relação a quantidade de requisições que chegam é igual ou superior a 99%.

Para analisar o comportamento do sistema com maior riqueza de detalhes, analisou-se os tempos médios de resposta geral e por tipo de requisição (*Principal, Detalhes, Buscas, Categorias, Login, Entrega, Pagamento e Confirmação*).

A Figura 7.7 apresenta os tempos médios de resposta geral e por tipo com o uso do algoritmo RR e as disciplinas FIFO, LIFO e LIFO-Pri. Observa-se que, para as duas primeiras disciplinas, os tempos médios de cada tipo de requisição apresentam valores próximos de seus

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75557 (21.02%)	75483 (99.90%)	2.168078	2
Detalhes	71563 (19.91%)	71500 (99.91%)	2.164388	4
Buscas	97322 (27.07%)	97238 (99.91%)	2.190449	3
Categorias	78811 (21.92%)	78733 (99.90%)	2.163400	1
Login	7310 (2.03%)	7302 (99.89%)	2.126525	5
Entrega	10812 (3.01%)	10795 (99.84%)	2.152444	6
Pagamento	7177 (2.00%)	7173 (99.94%)	2.139942	7
Confirmacao	10918 (3.04%)	10907 (99.90%)	2.169390	8
MÉDIA			2.170538	
TOTAL	359470	359131		

(a) Combinação RR/FIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75562 (21.03%)	75531 (99.96%)	2.111671	2
Detalhes	71746 (19.97%)	71716 (99.96%)	2.123289	4
Buscas	96871 (26.96%)	96827 (99.95%)	2.038433	3
Categorias	79143 (22.02%)	79104 (99.95%)	2.175950	1
Login	7153 (1.99%)	7151 (99.97%)	2.176056	5
Entrega	10870 (3.02%)	10869 (99.99%)	2.005643	6
Pagamento	7156 (1.99%)	7153 (99.96%)	2.056706	7
Confirmacao	10839 (3.02%)	10829 (99.91%)	1.992452	8
MÉDIA			2.101788	
TOTAL	359340	359180		

(b) Combinação RR/LIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75881 (21.08%)	75878 (100.00%)	0.240898	3
Detalhes	72173 (20.05%)	72169 (99.99%)	0.195759	4
Buscas	96776 (26.88%)	96614 (99.83%)	5.985908	1
Categorias	79240 (22.01%)	79234 (99.99%)	0.374117	2
Login	7307 (2.03%)	7307 (100.00%)	0.174341	5
Entrega	10708 (2.97%)	10708 (100.00%)	0.164026	6
Pagamento	7163 (1.99%)	7162 (99.99%)	0.183537	7
Confirmacao	10731 (2.98%)	10731 (100.00%)	0.203064	8
MÉDIA			1.797917	
TOTAL	359979	359803		

(c) Combinação RR/LIFO-Pri

Figura 7.7: Tempos de resposta por tipo de requisição (intensidade de carga alta - 100 req/seg).

respectivos tempos médios gerais, uma vez que não utilizam nenhum mecanismo de prioridade. Para a disciplina LIFO-Pri, a média geral dos tempos de resposta não retrata as médias individuais para cada tipo de requisição. Isso se deve pelo fato da disciplina LIFO-Pri utilizar um mecanismo de prioridade orientado ao consumo de CPU, conforme descrito na Seção 7.3.3.

Nota-se que o tempo médio de resposta para as requisições de menor prioridade (*Buscas*) está muito acima dos tempos médios de resposta dos demais tipos, influenciando diretamente na média geral. Embora a média geral do tempo de resposta da combinação RR/LIFO-Pri não retrate tão bem as médias individuais para cada tipo de requisição quanto das combinações RR/FIFO e RR/LIFO, observa-se ainda um ganho de desempenho de aproximadamente 17%.

Exceto para requisições do tipo *Busca*, a combinação RR/LIFO-Pri apresenta um ga-

nhos de desempenho que varia entre 890% e 920% no tempo médio de resposta em relação as combinações RR/FIFO e RR/LIFO. Entretanto, as requisições do tipo *Buscas* são penalizadas e perdem entre 177% e 186% para as combinações anteriores. Percebe-se ainda uma tendência de diminuição da porcentagem de atendimento de requisições de menor prioridade, no caso, requisições do tipo *Buscas*.

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75676 (20.97%)	75609 (99.91%)	0.665070	2
Detalhes	72091 (19.98%)	72046 (99.94%)	0.663689	4
Buscas	97536 (27.03%)	97471 (99.93%)	0.696099	3
Categorias	79086 (21.92%)	79041 (99.94%)	0.673262	1
Login	7348 (2.04%)	7344 (99.95%)	0.655702	5
Entrega	10926 (3.03%)	10920 (99.95%)	0.646925	6
Pagamento	7277 (2.02%)	7273 (99.95%)	0.661835	7
Confirmacao	10893 (3.02%)	10891 (99.98%)	0.689992	8
MÉDIA			0.674924	
TOTAL	360833	360595		

(a) Combinação SQF/FIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75741 (21.05%)	75706 (99.95%)	0.701456	2
Detalhes	71938 (20.00%)	71912 (99.96%)	0.651822	4
Buscas	96789 (26.90%)	96752 (99.96%)	0.702509	3
Categorias	79147 (22.00%)	79101 (99.94%)	0.699040	1
Login	7317 (2.03%)	7314 (99.96%)	0.712728	5
Entrega	10793 (3.00%)	10785 (99.93%)	0.584359	6
Pagamento	7236 (2.01%)	7235 (99.99%)	0.690709	7
Confirmacao	10813 (3.01%)	10811 (99.98%)	0.609529	8
MÉDIA			0.685020	
TOTAL	359774	359616		

(b) Combinação SQF/LIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	75625 (21.05%)	75621 (99.99%)	0.208950	3
Detalhes	71315 (19.85%)	71313 (100.00%)	0.185533	4
Buscas	97493 (27.14%)	97486 (99.99%)	1.206057	1
Categorias	78983 (21.98%)	78979 (99.99%)	0.262442	2
Login	7158 (1.99%)	7157 (99.99%)	0.168177	5
Entrega	10771 (3.00%)	10770 (99.99%)	0.165851	6
Pagamento	7142 (1.99%)	7142 (100.00%)	0.181941	7
Confirmacao	10786 (3.00%)	10785 (99.99%)	0.202324	8
MÉDIA			0.483794	
TOTAL	359273	359253		

(c) Combinação SQF/LIFO-Pri

Figura 7.8: Tempos de resposta por tipo de requisição (intensidade de carga alta - 100 req/seg).

As combinações baseadas em SQF apresentam pouca variação na média geral dos tempos de resposta. A combinação SQF/LIFO-Pri apresenta uma melhoria no tempo médio geral de resposta de aproximadamente 30% em relação as combinações SQF/FIFO e SQF/LIFO. Além disso, observa-se que durante os 3.600 segundos de tráfego submetidos ao sistema, o aproveitamento na quantidade de requisições atendidas em relação a quantidade de requisições que chegaram para a combinação SQF/LIFO-Pri é de 99,99%, superior a combinação SQF/FIFO

com 99,93% e a combinação SQF/LIFO com 99,95%.

A Figura 7.8 apresenta os tempos médios de resposta por tipos de requisições para as combinações baseadas em SQF. Novamente, constata-se que a média geral da combinação SQF/LIFO-Pri é comprometida pelo tempo médio de resposta das requisições de menor prioridade (*Buscas*) e não retrata o comportamento dos tempos médios individuais para os demais tipos de requisições.

Figura 7.9: Tempo médio de resposta do sistema (intensidade de carga alta - 100 req/seg).

Na Figura 7.9, observa-se uma melhoria expressiva na média geral dos tempos de resposta do sistema com a utilização das combinações baseadas em SQF em relação as combinações baseadas em RR. Constata-se que o sistema também apresentou comportamento semelhante quando submetido a uma intensidade de carga baixa.

A Figura 7.10(a) apresenta o quantidade de requisições atendidas com sucesso pelo sistema. Nota-se que os valores se estabilizam nos primeiros 300 segundos e permanecem até o final da simulação praticamente constantes. A porcentagem média de requisições atendidas em relação a quantidade de requisições que chegam é igual ou superior a 99,93%. Desta forma, a quantidade de requisições nas filas de CPU dos *hosts* servidores permanece razoavelmente

pequena, conforme apresenta a Figura 7.10(b).

As filas de CPU dos *hosts* servidores do *cluster* novamente foram monitoradas. Observa-se que o tamanho médio de fila de CPU em combinações baseadas em SQF se mantém num patamar de 10 requisições com picos de até 25 requisições. Já nas combinações baseadas em RR o tamanho médio de fila de CPU se mantém num patamar próximo a 25 requisições com picos que superam as 50 requisições em fila. Constatou-se que o tamanho médio de fila de CPU se mantém estável ao longo dos 3.600 segundos de tráfego ao qual o sistema fora submetido, sem contribuição significativa no aumento do tempo médio de resposta das requisições.

Intensidade de Carga Total

Finalmente o sistema fora submetido a uma intensidade de carga total, cuja taxa de utilização média de CPU atingiu os 100%. Desta forma, fez-se possível avaliar a eficiência do mecanismo de prioridades proposto no atendimento de requisições *Web* em sites de *e-commerce* em situações de sobrecarga.

Sob situações de sobrecarga, nota-se que a disciplina FIFO combinada tanto com o algoritmo SQF quanto com o algoritmo RR se mostra extremamente ineficiente. As Figuras 7.11(a) e 7.11(b) apresentam os tempos médios geral e por tipo de requisição para as combinações RR/FIFO e SQF/FIFO.

Observa-se que os tempos médios geral e individuais apresentam valores extremamente altos, portanto, fora do limite aceitável por clientes *Web* cada vez menos pacientes. Tendo em vista a ineficiência apresentada pelas combinações compostas pela disciplina FIFO em situações de sobrecarga, a seguir serão analisadas apenas as combinações das disciplinas LIFO e LIFO-Pri com os algoritmos RR e SQF.

Analisando os tempos médios por tipo de requisição para a combinação RR/LIFO (Figura 7.14(a)), nota-se que os mesmos estão próximos da média. Porém, observa-se que a porcentagem de requisições atendidas apresenta valores próximos a 97% em todos os tipos de requisições. Parte das requisições que chegaram ao sistema não foram atendidas no intervalo de uma hora e entre estas se encontram requisições da categoria *Pedir*. Sem dúvida, essa combinação de algoritmo/disciplina traria prejuízos para uma loja virtual, uma vez que esta deixa de atender requisições com alta probabilidade de geração de renda.

Os resultados da combinação RR/LIFO-Pri são apresentados na Figura 7.14(b). Como observado anteriormente, a disciplina LIFO-Pri possui a característica de aumentar o tempo médio de resposta das requisições de menor prioridade (*Buscas*). Desta vez, nota-se que a porcentagem de requisições atendidas do tipo *Busca* é inferior a porcentagem das demais cate-

(a) Requisições atendidas por segundo

(b) Tamanho médio das filas de CPU dos hosts servidores

Figura 7.10: Throughput e fila de CPU do sistema (intensidade de carga alta - 100 req/seg).

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79205 (20.98%)	77075 (97.31%)	52.845047	2
Detalhes	75425 (19.98%)	73430 (97.35%)	52.891626	4
Buscas	102321 (27.10%)	99632 (97.37%)	52.880444	3
Categorias	82691 (21.90%)	80515 (97.37%)	52.529428	1
Login	7674 (2.03%)	7460 (97.21%)	52.420732	5
Entrega	11284 (2.99%)	10985 (97.35%)	52.430976	6
Pagamento	7543 (2.00%)	7347 (97.40%)	52.560134	7
Confirmacao	11381 (3.01%)	11070 (97.27%)	53.395472	8
MÉDIA			52.784698	
TOTAL	377524	367514		

(a) Combinação RR/FIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79121 (20.96%)	76842 (97.12%)	46.169515	2
Detalhes	75238 (19.93%)	73145 (97.22%)	46.050654	4
Buscas	102125 (27.05%)	99338 (97.27%)	46.133939	3
Categorias	83117 (22.02%)	80839 (97.26%)	46.111482	1
Login	7680 (2.03%)	7452 (97.03%)	45.399973	5
Entrega	11296 (2.99%)	10973 (97.14%)	45.815935	6
Pagamento	7592 (2.01%)	7372 (97.10%)	45.321347	7
Confirmacao	11304 (2.99%)	10978 (97.12%)	46.092066	8
MÉDIA			46.077846	
TOTAL	377473	366939		

(b) Combinação SQF/FIFO

Figura 7.11: Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).

gorias, o que pode ocasionar uma situação de *starvation*, ou seja, em algum momento de carga extremamente alta a porcentagem de atendimento de requisições de menor prioridade será nula. Faz-se importante ressaltar que a porcentagem de atendimento de requisições da categoria *Pedir* alcança sua totalidade, indicando fortes indícios de que a combinação RR/LIFO-Pri se aplica em *sites de e-commerce*.

Comparando o tempo médio geral de resposta da combinação RR/LIFO-Pri com a RR/LIFO, observa-se um ganho de desempenho de 15,3%. Se excluídos os tempos médios das requisições de *Buscas*, observa-se um ganho de desempenho aproximado de 650% no tempo médio de resposta. Porém, constata-se uma perda considerável no desempenho do tempo médio de atendimento de requisições de *Buscas*, cuja proporção chega a 190% do tempo médio da combinação RR/LIFO.

A Figura 7.13 apresenta os relatórios de tempo de resposta médio geral e individual para as combinações baseadas no algoritmo SQF. Observa-se que, para ambas as disciplinas LIFO e LIFO-Pri, o comportamento do sistema é equivalente ao das combinações baseadas em RR, porém em diferentes escalas.

Para a combinação SQF/LIFO, novamente observa-se que a porcentagem de requisição atendidas apresenta valores em um patamar próximo a 97% para todos os tipos de requisições, incluindo requisições da categoria *Pedir*. Para a combinação SQF/LIFO-Pri observa-se que

Figura 7.12: Tempo médio de resposta do sistema (intensidade de carga total - 105 req/seg).

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79382 (21.01%)	77274 (97.34%)	0.576885	2
Detalhes	75650 (20.02%)	73614 (97.31%)	0.561649	4
Buscas	102148 (27.04%)	99362 (97.27%)	0.623308	3
Categorias	82862 (21.93%)	80568 (97.23%)	0.579414	1
Login	7708 (2.04%)	7488 (97.15%)	0.552144	5
Entrega	11243 (2.98%)	10951 (97.40%)	0.673110	6
Pagamento	7528 (1.99%)	7334 (97.42%)	0.659160	7
Confirmacao	11270 (2.98%)	10928 (96.97%)	0.648673	8
MÉDIA			0.593078	
TOTAL	377791	367519		

(a) Combinação SQF/LIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79136 (21.01%)	79128 (99.99%)	0.219307	3
Detalhes	75700 (20.10%)	75695 (99.99%)	0.192014	4
Buscas	101564 (26.97%)	94298 (92.85%)	1.402088	1
Categorias	82730 (21.97%)	82725 (99.99%)	0.291891	2
Login	7590 (2.02%)	7589 (99.99%)	0.173854	5
Entrega	11212 (2.98%)	11212 (100.00%)	0.163702	6
Pagamento	7534 (2.00%)	7534 (100.00%)	0.187744	7
Confirmacao	11137 (2.96%)	11136 (99.99%)	0.205541	8
MÉDIA			0.528291	
TOTAL	376603	369317		

(b) Combinação SQF/LIFO-Pri

Figura 7.13: Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).

todas as requisições para páginas da categoria *Pedir* foram atendidas com sucesso e que as requisições de *Buscas* tiveram não só o tempo médio de resposta comprometido, mas também a porcentagem no aproveitamento, em relação a quantidade de requisições que chegaram.

Se comparados o tempo médio geral de resposta da combinação SQF/LIFO-Pri com o da SQF/LIFO, observa-se um ganho de desempenho de 12,2%. Se excluídos os tempos médios das requisições de *Buscas*, observa-se um ganho de desempenho aproximado de 196% no tempo médio de resposta. Porém, constata-se uma perda considerável no desempenho do tempo médio de atendimento de requisições de *Buscas*, cuja proporção chega a 120% do tempo médio da combinação RR/LIFO.

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79453 (21.06%)	77268 (97.25%)	1.703470	2
Detalhes	75433 (19.99%)	73327 (97.21%)	1.727435	4
Buscas	101791 (26.98%)	98912 (97.17%)	1.721141	3
Categorias	82825 (21.95%)	80497 (97.19%)	1.654375	1
Login	7717 (2.05%)	7504 (97.24%)	1.823479	5
Entrega	11301 (3.00%)	10979 (97.15%)	1.630650	6
Pagamento	7458 (1.98%)	7243 (97.12%)	1.922430	7
Confirmacao	11321 (3.00%)	10996 (97.13%)	1.588014	8
MÉDIA			1.703390	
TOTAL	377299	366726		

(a) Combinação RR/LIFO

Tipo	Cheg	Term	Tempo	Prioridade
Principal	79348 (21.04%)	79343 (99.99%)	0.253797	3
Detalhes	75332 (19.98%)	75330 (100.00%)	0.200039	4
Buscas	101421 (26.90%)	93957 (92.64%)	4.996055	1
Categorias	82931 (21.99%)	82909 (99.97%)	0.413162	2
Login	7631 (2.02%)	7631 (100.00%)	0.176853	5
Entrega	11393 (3.02%)	11392 (99.99%)	0.167629	6
Pagamento	7529 (2.00%)	7529 (100.00%)	0.186327	7
Confirmacao	11500 (3.05%)	11499 (99.99%)	0.204384	8
MÉDIA			1.477008	
TOTAL	377085	369590		

(b) Combinação RR/LIFO-Pri

Figura 7.14: Tempos de resposta por tipo de requisição (intensidade de carga total - 105 req/seg).

Utilizando o tempo médio de resposta do sistema como medida de desempenho, observa-se na Figura 7.12 que as combinações baseadas em SQF apresentam tempos médios gerais de resposta menores em relação as combinações baseadas em RR. Considerando que a média dos tempos de resposta das combinações RR/LIFO e RR/LIFO-Pri é 1,590199 segundos e que a média dos tempos das combinações SQF/LIFO e SQF/LIFO-Pri é 0,560684 segundos, constata-se um ganho de desempenho em combinações baseadas em SQF de 183% em relação as combinações baseadas em RR.

A Figura 7.15(a) apresenta o quantidade de requisições atendidas com sucesso pelo sistema. Nota-se que, exceto para a combinação SQF/FIFO, os valores se estabilizam após 1.000

(a) Requisições atendidas por segundo

(b) Tamanho médio das filas de CPU dos hosts servidores

Figura 7.15: Throughput e fila de CPU do sistema (intensidade de carga total - 105 req/seg).

segundos e se mantém praticamente constantes até o final da simulação. Os valores da combinação SQF/FIFO diminuem consideravelmente entre 800 e 2.000 segundos, se estabilizando na sequência até o término do experimento. Ao final de 3.600 segundos, observa-se que a quantidade média de requisições atendidas por segundos na combinação RR/LIFO-Pri é a maior, com 102,663 req/seg, seguida da combinação SQF/LIFO-Pri com 102,587 req/seg. Em um patamar um pouco abaixo se encontram as combinações SQF/LIFO e RR/FIFO praticamente empata-das com 102,088 req/seg e em seguida as combinações SQF/FIFO e RR/LIFO com 101,927 e 101,868 respectivamente.

Com o sistema submetido a uma carga de intensidade total, o aumento na quantidade de requisições em filas de CPU se confirma e desta vez. Na Figura 7.15(b), nota-se um ganho de desempenho considerável no tamanho das filas de CPU das combinações baseadas na disciplina LIFO-Pri, tanto com o algoritmo RR quanto com o algoritmo SQF. Enquanto o tamanho das filas das combinações RR/LIFO-Pri e SQF/LIFO-Pri se encontra em um patamar de 922 requisições em espera, o das demais combinações se encontra em um patamar de 1.300 requisições.

7.5 Considerações finais

Neste capítulo foi apresentado um modelo para *clusters* de servidores *Web* projetados para atender a demanda de tráfego de *sites* de comércio eletrônico, especificamente lojas varejistas *online*.

Realizou-se a validação do modelo proposto e uma descrição detalhada dos parâmetros utilizados, com o intuito de provar que o comportamento do sistema implementado no modelo se aproximam de um sistema real.

Foram apresentados os gráficos referentes as respectivas medidas de desempenho adotadas bem como realizada a avaliação de desempenho do sistema sob intensidades de carga baixa, alta e total. Os algoritmos de escalonamento RR e SQF foram aplicados no componente escalonador do modelo e combinados com as disciplinas FIFO, LIFO e LIFO-Pri de atendimento nas filas de CPU nos *hosts* servidores do *cluster*.

Pode-se observar que o sistema implementado no modelo só atinge sua taxa de utilização máxima de CPU após ser submetido a uma intensidade de carga com taxa de chegada igual ou superior a 105 requisições por segundo.

Combinações baseadas no algoritmo SQF apresentam menores tempos de resposta, quando comparadas as combinações baseadas em RR. Isso se deve ao fato do algoritmo SQF

utilizar informações atualizadas de carga do sistema para a tomada de decisão no escalonamento de requisições.

Constata-se que a intensidade de carga influencia no tempo médio de resposta do sistema apenas em situações em que o sistema atinge sua utilização máxima e aplica a disciplina de atendimento FIFO nas filas dos *hosts* servidores *Web* do *cluster*.

Por fim, conclui-se que a utilização do mecanismo de prioridades proposto nas filas de CPU dos *hosts* servidores *Web* do *cluster* se adequa melhor as necessidades de um *site* de *e-commerce*, pois apresenta aproveitamento máximo no atendimento de requisições da categoria *Pedir*, tempos médios de resposta menores do que os apresentados sem a utilização do mecanismo de prioridades e menor tamanho médio de fila de CPU do sistema.

O capítulo seguinte apresenta as conclusões desta dissertação e algumas sugestões para futuros trabalhos.

Conclusões

8.1 Visão Geral

Neste trabalho realizou-se uma avaliação de desempenho com algoritmos de escalonamentos e disciplinas de atendimento para filas em *clusters* de servidores *Web* projetados para *sites de e-commerce*. Tendo em vista o importante papel que sites do gênero têm representado na economia mundial, a motivação surgiu devido ao fato de servidores *Web* de *sites de e-commerce* frequentemente se depararem com situações de sobrecarga, as quais podem acarretar falhas durante a efetivação de transações financeiras.

Atualmente, em sites de e-commerce, requisições destinadas a seções de navegação e busca de produtos e requisições destinadas a seções de compra de produtos são tratadas de forma equivalente. Com isso, constata-se a importância de classificar requisições por categorias e priorizá-las, respeitando suas peculiaridades.

8.2 Resultados e Contribuições

Organizados por assuntos, apresenta-se a seguir os principais resultados deste trabalho:

1. **Revisão Bibliográfica** – Um levantamento foi feito sobre arquiteturas de servidores *Web* distribuídos, juntamente com soluções para distribuir e balancear carga baseadas em informações da camada de rede e de aplicação. Um análise das principais políticas e algoritmos de escalonamento recentemente publicadas foi realizada.

2. **Modelo do Servidor Web para E-Commerce (SWE-C)** – Um modelo de um servidor *Web* projetado para *sites* de *e-commerce* foi proposto. O mesmo apresenta as características mais relevantes de um sistema real. Foram modelados o componente escalonador e um cluster de servidores *Web*, onde cada *host* possui uma fila de CPU e uma fila I/O, típicas de uma arquitetura de *Web Clusters*.
3. **Modelo e Geração de Carga de Trabalho** – Com base em trabalhos relacionados ao assunto, um modelo de carga de trabalho para *sites* de *e-commerce* foi desenvolvido com base nos principais tipos de requisições que caracterizam uma loja virtual varejista. Utilizando o modelo desenvolvido neste trabalho, gerou-se um carga de trabalho sintética para validar o modelo de servidor SWE-C.
4. **Validação do modelo SWE-C** – Foram realizadas simulações para validar o modelo SWE-C, juntamente com a implementação de de uma carga de trabalho sintética baseado no modelo proposto. Alterações foram realizadas no simulador Simpack para implementar o mecanismo de prioridades na disciplina LIFO-Pri. O tempo médio de resposta foi utilizado como principal medida de desempenho, embora o *throughput*, o tamanho das filas de CPU e a taxa de utilização de CPU do sistema também tenham sido analisados e contribuído para atingir os objetivos deste trabalho. Os resultados apresentados foram obtidos a partir de média de trinta execuções com um intervalo de confiança de 95%.
5. **Mecanismo de Prioridades Orientado ao Consumo de CPU** – Com base em tempos médios de resposta de experimentos realizados em um ambiente real, desenvolveu-se um mecanismo de prioridade para tipos de requisições orientado ao consumo de CPU. Para categorias de requisições com altos índices de consumo de CPU, foram atribuídas baixas prioridades, com o objetivo de não sobrecarregar o sistema. Esse mecanismo de prioridades foi implementado no modelo de simulação e apresentou resultados que o adequa as necessidades de *sites* de *e-commerce*.
6. **Modificações no Servidor Web Apache** – Foram realizadas modificações no código fonte do software servidor *Web* Apache. Um estudo das funcionalidades oferecidas pelo software foi realizado e identificou-se a necessidade de disponibilizar novos recursos. Funções foram desenvolvidas diretamente no código fonte do software com o objetivo de disponibilizar no arquivo de *logs* o tempo de consumo de CPU de cada requisição *Web* atendida.

Com os resultados, espera-se contribuir para a melhor utilização de recursos de sistemas implementados para atender o tráfego de requisições de sites de comércio eletrônico e aumentar

o *throughput* de requisições da categoria *Pedir* com desempenho aceitável.

8.3 Trabalhos Futuros

A seguir, são discutidas algumas sugestões para trabalhos futuros:

8.3.1 Geração de Carga de Trabalho

- Com base no modelo desenvolvido, implementar um gerador de carga de trabalho sintética para sites de e-commerce. Seria interessante implementar o recurso de cache no cliente e suporte a conexões SSL (*Security Socket Layer*);
- Implementar um analisador de *log* para analisar o conteúdo do logs do Apache após as modificações realizadas. A partir disso, seria possível modelar estatisticamente o consumo de recursos (CPU e I/O) de requisições Web;
- Alterar o modelo de carga de trabalho desenvolvido com o objetivo de caracterizar maiores detalhes dos tipos de requisições de um *site* de *e-commerce*, particionando as requisições da categoria *Pedir* em duas: *Carregar Formulário* e *Enviar Formulário*. O aumento na variedade de tipos de requisições disponibiliza mais recursos para uma avaliação de desempenho mais profunda.

8.3.2 Modelo de Servidor Web para E-Commerce (SWE-C)

- Experimentar o modelo SWE-C com uma carga de trabalho orientada a *traces*, utilizando *logs* de servidores reais com as modificações realizadas neste trabalho. A dificuldade, nesse caso, está em encontrar uma empresa que esteja disposta a utilizar a versão modificada do software servidor Web Apache e a ceder os arquivos de *logs*;

Referências

- ALLEN, A. O. Probability, statistics, and queueing theory. *Academic Press*, London, 1990.
- ARON, M.; DRUSCHEL, P.; ZWAENEPOEL, W. Cluster reserves: A mechanism for resource management in cluster-based network servers. In: *In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. Santa Clara: [s.n.], 2000.
- BESTAVROS, A. Load profiling in distributed real-time systems. *Journal of Information Sciences*, v. 101, p. 1–27, June 1997.
- BORENSTEIN, N.; BELLCORE; FREED, N.; INNOSOFT. *RFC1521 - MIME (Multipurpose Internet Mail Extensions)*: Part one: Mechanisms for specifying and describing the format of internet message bodies.
- BRANCO, K. R. L. J. C. *Índice de Carga e Desempenho em Ambientes Paralelos/Distribuídos: Modelagem e métricas*. Monografia (Exame de Qualificação do Mestrado) — ICMC-USP, São Carlos, 2000.
- BUNT, R. B.; EAGER, D. L.; OSTER, G. M.; WILLIAMSON, C. L. Achieving load balance and effective caching in clustered web servers. In: *in Proceedings of the Fourth International Web Caching Workshop*. San Diego: [s.n.], 1999. p. 159–169.
- CARDELLINI, V.; CASALICCHIO, E.; COLAJANNI, M.; YU, P. S. The state of the art in locally distributed web-server systems. In: *ACM Computing Surveys*. [S.l.: s.n.], 2002. v. 34, p. 263–311.
- CARDELLINI, V.; COLAJANNI, M.; YU, P. S. Dynamic load balancing on web-server systems. In: *IEEE Internet Computing*. [S.l.: s.n.], 1999. v. 3, n. 3, p. 28–39.
- CASALICCHIO, E.; COLAJANNI, M. A client-aware dispatching algorithm for web clusters providing multiple services. In: *Proceeding of 10th International World Wide Web Conference*. Hong Kong: [s.n.], 2001. p. 535–544.
- CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, v. 14, n. 2, p. 141–154, Fev. 1988.
- COLAJANNI, M.; YU, P. S.; DIAS, D. M. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, v. 9, n. 6, June 1998.

- COMER, D. E. *Internetworking with TCP/IP: Principles, protocols, and architecture*. 4. ed. [S.l.]: Prentice Hall, 2000. 750 p.
- CYBERATLAS. *CyberAtlas: Internet Statistics and Market Research for Web Marketers*. September 2002. Disponível em: <http://www.nua.net/surveys/how_many_online/index.htm>.
- DEVLIN, B.; GRAY, J.; LAING, B.; SPIX, G. *Scalability terminology: Farms, clones, partitions and pack: RACS and RAPS*. [S.l.], 1999.
- E-Bit. *Web Shoppers, Raio-X do Comércio Eletrônico em 2005*. Jan. 2006. 34 p.
- GOLDBERG, A.; BUFF, R.; SCHMITT, A. Secure web server performance dramatically improved by caching ssl session keys. *Workshop on Internet Server Performance*, Jun. 1998.
- HARCHOL-BALTER, M.; SCHROEDER, B.; BANSAL, N.; AGRAWAL, M. Size-based scheduling to improve web performance. In: 2 (Ed.). [S.l.]: ACM Transactions on Computer Systems (TOCS), 2003. v. 21, p. 207–233.
- HIRATA, R. *Otimizando Servidores Web de Alta Demanda*. Dissertação (Mestrado) — Instituto de Computação—Universidade Estadual de Campinas, Abril 2002.
- IDG-Now. *IDG Now - Tecnologia em Primeiro Lugar*. 2006. Disponível em: <<http://idgnow.uol.com.br>>. Acesso em: 01 fev. 2006.
- LEON, H. P. de; BISSETT, S.; ROLLIN, P.; RESSLER, M. *osCommerce - Open Source E-Commerce Solutions*. 2006. Disponível em: <<http://www.oscommerce.org>>. Acesso em: 1 mar. 2006.
- LINUX Virtual Server. 2002. Disponível em: <<http://www.linuxvirtualserver.org>>.
- MULLENDER, S. (Ed.). *Distributed Systems*. 2. ed. New York: Addison-Wesley Publishing Company, 1993. 601 p. (ACM Press Frontier Series).
- Netcraft. *Netcraft - Web Server Survey Archives*. 2006. Disponível em: <<http://www.netcraft.co>>. Acesso em: 1 mar. 2006.
- ORFALI, R.; HARKEY, D.; EDWARDS, J. *Client/Server Survival Guide*. 3. ed. [S.l.]: John Wiley & Sons, 1999.
- PAI, V. S.; ARON, M.; BANGA, G.; SVENDSEN, M.; DRUSCHEL, P.; ZWAENEPOEL, W.; NAHUM, E. Locality-aware request distribution in cluster-based network servers. In: *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*. [S.l.: s.n.], 1998.
- PERKINS, C. *RFC2003 - IP Encapsulation within IP*. October 1996. IBM.
- POSTEL, J. *RFC792 - Internet Control Message Protocol*. September 1981. IETF.
- RIJSINGHANI, A. *RFC1624 - Computation of the Internet Checksum via Incremental Update*. May 1994. Digital Equipment Corporation.
- SCHROEDER, B.; HARCHOL-BALTER, M. Web server under overload: How scheduling can help. *18th International Teletraffic Congress (ITC 2003)*, p. 18, May 2002.

- SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. Load distributing for locally distributed systems. *IEEE Computer*, v. 25, n. 12, p. 33–44, 1992.
- SINGHMAR, N.; MATHUR, V.; APTE, V.; D.MANJUNATH. A combined lifo-priority scheme for overload control of e-commerce web servers. In: *International Infrastructure Survivability Workshop (IISW'2004)*. [S.l.]: IEEE Computer Society, 2004.
- SOUZA, P. S. L. de. *AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos*. Tese (Doutorado) — Instituto de Física de São Carlos—Universidade de São Paulo, São Carlos, 2000.
- SPECweb2005. *SPECweb2005 E-Commerce Workload Design Document*. 2005. Disponível em: <<http://www.spec.org/web2005>>. Acesso em: 14 jun. 2005.
- SRISURESH, P.; NETWORKS, J.; EGEVANG, K. *RFC3022 - Traditional IP Network Address Translator (Traditional NAT)*. January 2001. Intel Corporation.
- TANENBAUM, A. S. *Redes de Computadores*. 3. ed. Rio de Janeiro: Campus, 1997. 923 p. Tradução da terceira edição.
- TEXEIRA, M. A. M. *Suporte a Serviços Diferenciados em Servidores Web: Modelos e Algoritmos*. Tese (Doutorado) — ICMC-USP, São Carlos, maio 2004.
- VALLAMSETTY, U.; KANT, K.; MOHAPATRA, P. Characterization of e-commerce traffic. *Electronic Commerce Research*, p. 167–192, Mar. 2003.
- ZHANG, X.; BARRIENTOS, M.; CHEN, J. B.; SELTZER, M. Hacc: An architecture for cluster-based web servers. In: *In Proceedings of the 3rd USENIX Windows NT Symposium*. Seattle: [s.n.], 1999.
- ZHUO, L.; WANG, C.-L.; LAU, F. C. *Document Replication and Distribution in Extensible Geographically Distributed Web Server*. 2002.