Um sistema servidor web distribuído com provisão
de QoS absoluta e relativa

Edwin Luis Choquehuanca Mamani

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Um sistema servidor web distribuído com provisão de QoS absoluta e relativa

Edwin Luis Choquehuanca Mamani

Orientador: *Prof. Dr. Francisco José Monaco*

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

USP - São Carlos
Dezembro/2010

A minha querida mãe, sem ela nada seria na vida.

Agradecimentos

Agradeço, primeiramente, a Deus. Ele, que nunca me deixa só.

A Universidade de São Paulo, USP - Campus de São Carlos, pela oportunidade de realizar o curso de Pós-Graduação.

Aos meus pais, Eduarda e José, em especial a minha mãe pelo apoio incondicional em todas as horas.

Ao meu irmão Orlando e minha irmã Alicia, que preocupam-se sempre com o meu bem estar e pelo apoio na minha formação.

Aos meus amigos, Juliano, Haline, Manuel, Yone, Eli, Rosangela, Elisangela, Oscar, Teresa, Daniel e todo o pessoal do Salão do Reino. Pela companhia e pelos conselhos nos momentos difíceis.

Ao meu orientador e amigo, Monaco, pela dedicação ao me orientar. A todos meus colegas e amigos do LaSDPC, que não vou citar os nomes porque são muitos, e todos igualmente especiais, pela dedicação e ajuda.

Aos professores Marcos, Regina, Kalinka, Sarita, Paulo e Edson pela promoção dos eventos do grupo de Sistemas Distribuídos e Programação Concorrente (SDPC).

Aos funcionários do ICMC-USP, em especial aos funcionários da secretária da Pós-Graduação, pelo convívio amigável e descontraído, e pela disposição em sempre atender bem.

A **CAPES** (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), pelo apoio financeiro.

RESUMO

Este trabalho de pesquisa apresenta um protótipo de servidor Web distribuído com diferenciação de serviços baseado em *QoS* relativa e absoluta. São implementados e comparados diferentes algoritmos de escalonamento. Um dos algoritmos avaliados é o *EBS*. O objetivo é transpor a teoria da política para o mundo real, e comparar o seu comportamento com os resultados das simulações, utilizando o tempo de resposta como medida de desempenho. Além do *EBS*, outros algoritmos são avaliados, tais como, *Round-Roubin* e *Weighted Round Robin*.

ABSTRACT

This work presents a prototype of a distributed Web server with service differentiation based on relative and absolute QoS. Are compared different schedule algorithms. One of the policies to be evaluated is the EBS. The goal is to implement the policy theory to the real world, comparing their behavior with the simulation results, using the response time as a performance measure. In addition to the EBS, others policies are evaluated, such as Round-Roubin and Weight Round Robin.

Conteúdo

1	Introdução	1
1.1	Considerações Iniciais	1
1.2	Motivação	1
1.3	Objetivos	2
1.4	Organização do Documento	2
2	A Internet e a Web	5
2.1	Considerações Iniciais	5
2.2	A Internet	6
2.2.1	Modelo de Referência <i>TCP/IP</i>	6
2.3	A <i>World Wide Web</i>	8
2.3.1	Descrição geral do <i>HTTP</i>	10
2.3.1.1	Conexão Persistente	10
2.4	Servidor <i>Web</i>	13
2.4.1	Modelos de Atendimento	14
2.4.2	Servidor Proxy	16
2.4.3	Lista de Servidores Web	16
2.5	Considerações Finais	17
3	Serviços Diferenciados e Tempo Real	19
3.1	Considerações Iniciais	19
3.2	Conceitos Básicos de QoS	20
3.2.1	QoS em Nível de Rede	20
3.3	QoS em Nível de Aplicação	22
3.3.1	QoS Relativa	22
3.3.2	QoS Absoluta	23
3.4	Sistemas de Tempo Real	25

3.4.1	Sistemas <i>Hard-RT</i> e <i>Soft-RT</i>	25
3.4.2	Restrições Temporais	26
3.5	Algoritmos de Escalonamento	27
3.5.1	Algoritmos de fila	27
3.5.2	Algoritmos de distribuição	30
3.6	Considerações Finais	31
4	Servidor Proxy com Diferenciação de Serviços (JProxyQoS)	33
4.1	Considerações Iniciais	33
4.2	Arquitetura do Protótipo	34
4.2.1	Classificador	36
4.2.2	Controle de Admissão	37
4.2.3	Distribuidor	38
4.3	Modelagem do protótipo	39
4.3.1	Diagramas de Caso de Uso	40
4.3.2	Diagramas de classes	40
4.4	Considerações Finais	45
5	Resultados Experimentais	47
5.1	Considerações Iniciais	47
5.2	Avaliação de desempenho do protótipo	48
5.2.1	Configuração do Ambiente	48
5.2.2	Planejamento dos Experimentos	49
5.2.3	Carga de trabalho sintética	51
5.3	Validação do modelo Web utilizado	51
5.4	Análise dos Resultados dos Algoritmos	53
5.4.1	Algoritmos do Módulo Distribuidor	54
5.4.1.1	RR e WRR	54
5.4.1.2	RR e WRR com carga externa	55
5.4.1.3	Tempo de resposta RR e WRR	56
5.4.2	Algoritmos do módulo Controle de Admissão	57
5.4.2.1	EBS 50% A e 50% B	57
5.4.2.2	EBS 10% A e 90% B	58
5.4.2.3	SJF e EBS 90% A e 10% B	59
5.5	Considerações Finais	60
6	Conclusão	63
6.1	Considerações Finais	63
6.2	Principais Resultados e Contribuições	64
6.3	Trabalhos Futuros	65

Referências Bibliográficas**67**

Lista de Figuras

2.1	Requisição HTTP (Kurose e Ross, 2006).	12
2.2	Resposta HTTP (Kurose e Ross, 2006).	13
2.3	Uma transação HTTP utilizando o método GET (Kurose e Ross, 2006).	14
2.4	Um processo por cada requisição (Callaway, 2001).	15
2.5	Um processo usa vários <i>threads</i> para resolver as requisições (Callaway, 2001).	15
2.6	O processo <i>dispatcher</i> repassa a requisição ao servidor <i>Web</i> (Callaway, 2001).	16
3.1	Ilustração do Funcionamento de Sinalização do <i>RSVP</i> (Zhao et al., 2000).	20
3.2	Ilustração das Restrições Temporais.	27
3.3	Exemplo da Utilização do Algoritmo de Escalonamento FIFO.	28
3.4	Exemplo da Utilização do Algoritmo de Escalonamento SJF.	28
4.1	Visão geral da arquitetura do protótipo.	34
4.2	Arquitetura do protótipo.	35
4.3	Módulo de classificação.	36
4.4	Módulo de controle de admissão.	37
4.5	Módulo de controle de admissão.	39
4.6	Diagrama de casos de uso <i>nível 1</i>	40
4.7	Diagrama do Classificador e Pool de threads.	41
4.8	Diagrama do Controle de Admissão.	42
4.9	Diagrama do Distribuidor.	43
4.10	Diagrama do monitor dos usuários.	44
5.1	Configuração do <i>hardware</i> das máquinas envolvidas no experimento.	48
5.2	Comportamento dos níveis de requisições na aplicação <i>Web</i>	52
5.3	Tempos de resposta dos diferentes níveis de requisições.	52
5.4	Tempo de resposta para o Tomcat com 60 requisições/s.	53

5.5	Porcentagem de utilização da CPU nas máquinas envolvidas.	53
5.6	Relação de % da CPU e % de reqs enviadas.	55
5.7	Relação de % da CPU e % de reqs enviadas, com carga externa.	55
5.8	Tempos de resposta para dos algoritmos RR e WRR com e sem carga externa.	56
5.9	Tempos de resposta – Cenário 50% classe A e 50% classe B.	57
5.10	Tempos de resposta – Cenário 10% classe A e 90% classe B.	59
5.11	Tempos de resposta – Cenário 90% classe A e 10% classe B.	60

Lista de Tabelas

2.1	Métodos do protocolo <i>HTTP/1.1</i>	11
2.2	Totais por servidores ativos em todos os domínios (NETCRAFT, 2010).	17
5.1	Elementos de <i>software</i> envolvidos.	49
5.2	Experimentos dos algoritmos do módulo Distribuidor.	50
5.3	Experimentos dos algoritmos do módulo Controle de Admissão.	50
5.4	Porcentagens de satisfação dos contratos, 50% classe A e 50% classe B. . .	58
5.5	Porcentagens de satisfação dos contratos, 10% classe A e 90% classe B. . .	59
5.6	Porcentagens de satisfação dos contratos, 90% classe A e 10% classe B. . .	60

Nomenclatura

ANR Algoritmo de Negociação Rigoroso.

ATM Asynchronous Transfer Mode.

CERN Organisation Européenne pour la Recherche Nucléaire.

DiffServ Differentiated Services.

DNS Domain Name System.

EBS Exigency-Based Scheduling.

EDF Earliest Deadline First.

FIFO First Input First Output.

FTP File Transfer Protocol.

GWS Google Web Server.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IETF Internet Engineering Task Force.

IETF Internet Engineering Task Force.

IIS Internet Information Services.

IntServ Integrated Services.

IP Internet Protocol.

ISO International Organization for Standardization.

ISP Internet Service Provider.

MIT Massachusetts Institute of Technology.

MVC Model View Controller.

OoS Quality of Service.

OSI Open System Interconnection.

PPP Point-to-Point Protocol.

RSVAdap Reserva Adaptativa de Recursos.

RSVP Resource Reservation Protocol.

SFD Short Flow Differentiating.

SJF Shortest Job Firs.

SJF Shortest Job First.

SLA Service Level Agreement.

SMTP Simple Mail Transport Protocol.

SWDS Servidor Web com diferenciação de serviços.

TCP Transmission Control Protocol.

TOS Type of Service.

UML Unified Modeling Language.

URL Uniform Resource Locator.

W3C World Wide Web Consortium.

WFQ Weighted Fair Queuing.

WWW World Wide Web.

XML Extensible Markup Language.

Introdução

1.1 Considerações Iniciais

Com o avanço das redes de comunicação (LAN, WAN, WLAN, Internet, etc.), o número de usuários desses meios têm se multiplicado a cada dia, o mesmo ocorrendo com os tipos de aplicativos. Junto a esse crescimento vem a necessidade de transmissões mais rápidas e confiáveis, que garantam a satisfação dos clientes, estes que estão cada vez mais exigentes e diferentes. Contudo, não basta apenas aumentar a banda e a velocidade das transmissões; é necessário a garantia de entrega dos pedidos que trafegam na rede, e ainda no tempo compatível com as exigências de cada usuário. Outro fator a ser considerado é o aumento da complexidade das operações que as aplicações executam. Uma solução desse problema é a divisão de tarefas entre várias unidades de processamento.

1.2 Motivação

A Web está se tornando cada vez mais uma plataforma orientado a negócios ([Lee et al., 2004](#)) e a interface preferida para os mais recentes sistemas de informação ([Andreolini et al., 2004](#)). Consequentemente torna-se mais importante projetar e implementar sistemas capazes de diferenciar o desempenho designado a usuários e serviços individuais, resolvendo cada tipo de pedido com preferências especiais. Tais sistemas devem ser capazes de lidar com diferentes exigências de diversos tipos de aplicações e também com as diferentes expectativas dos usuários ([Zhou et al., 2004](#)), permitindo tratamento diferenciado de acordo com suas prioridades. Por exemplo, transações econômicas são geralmente mais

importantes que uma simples visualização de um site de entretenimento e usuários pagantes de um determinado serviço podem esperar melhor qualidade em relação ao público não pagante.

No entanto, o modelo de melhor esforço usado pela Internet, tanto em nível de rede através do protocolo IP, quanto em nível de aplicação, onde as requisições são atendidas pelos servidores de acordo com a política FIFO (First Input First Output) (Nancy J. Yeager, 1996), não é adequado para prover a qualidade de serviço exigida por usuários e aplicações na atualidade. Uma quantidade significativa de pesquisas sobre Qualidade de Serviço (QoS) tem sido focada sobre a infra-estrutura da rede, tais como protocolos, roteadores, entre outros. Também foram desenvolvidos algoritmos que visam a solução do cenário de vários usuários com níveis de serviço diferentes, dentre eles estão o algoritmo EBS (Casagrande e Monaco, 2007). Além disso, é bem conhecido que a carga das requisições para um determinado serviço está em aumento devido ao número de usuários e a cada vez mais complexa algorítmica que os serviços utilizam (Gao et al., 2009).

Também já foram desenvolvidas sob a coordenação da IETF (Internet Engineering Task Force), várias especificações para a provisão de QoS sobre redes IP, dentre elas destacam-se as arquiteturas de Serviços Integrados (IntServ) (Braden et al., 1994) e de Serviços Diferenciados (DiffServ) (Blake et al., 1998). Mas, são desafiados pela grande diversidade de tipos de usuários e aplicações existentes, e como resultado existe um aumento no número de usuários mal atendidos e recursos mal gerenciados.

1.3 Objetivos

Com a idéia de tentar dar solução para alguns dos problemas citados anteriormente, os objetivos desse trabalho de mestrado são:

- Construir um protótipo de um servidor Web distribuído baseado na utilização de clusters de servidores *Web*;
- Implementar políticas de escalonamento de tempo real para garantia de *QoS* absoluta e relativa;
- Analisar o desempenho das políticas na prática por meio do protótipo.

1.4 Organização do Documento

No projeto foi decidido dar destaque à parte de fundamentação teórica assim como as ferramentas utilizadas e a apresentação dos resultados, isto é feito da seguinte maneira:

- No **Capítulo 2**, é apresentada uma revisão geral sobre a estrutura da Web, bem como os protocolos da arquitetura *TCP/IP* e uma visão geral do protocolo *HTTP*.

- No **Capítulo 3**, são introduzidos os conceitos de qualidade de serviço. É abordada a QoS em nível de rede, descrevendo as arquiteturas de serviços integrados e de serviços diferenciados, também é destacada a importância da QoS em nível de aplicação e apresentada a fundamentação teórica que embasa sistemas de tempo real, é feita uma apresentação dos algoritmos que vão ser implementados no protótipo;
- No **Capítulo 4**, descreve-se a arquitetura e a modelagem do protótipo a ser implementado, especificando detalhes das mesmas.
- No **Capítulo 5**, são apresentados os resultados obtidos por meio dos experimentos realizados com o protótipo utilizando carga sintética e também uma análise desses resultados, comparando-os com os resultados obtidos em trabalhos prévios.
- No **Capítulo 6**, são apresentadas as conclusões e principais contribuições desta dissertação, bem como, algumas sugestões para trabalhos futuros.

A Internet e a Web

2.1 Considerações Iniciais

Até os anos 60 a comunicação empregada pela computação se restringia apenas a mover *bits* de forma segura de um *mainframe* até seus terminais aplicativos. Os avanços tecnológicos alcançados ao longo das décadas de 70 e 80 permitiram a evolução dessas redes de terminais para redes de computadores, baseadas na tecnologia de comutação de pacotes, que por sua vez propiciou o surgimento de redes interconectadas, como a Internet. A partir desse momento, e principalmente a partir da década de 90, a Internet passou por uma evolução contínua de sua extensão, importância e tecnologias agregadas. Um dos fatores que contribuíram para seu forte crescimento foi o surgimento da *World Wide Web* (ou simplesmente Web), aplicação de rede que tornou mais simples e intuitivo o acesso do público em geral ao conteúdo hospedado por essa imensa rede pública. A Web projetou a Internet aos lares e empresas de milhões de pessoas no mundo inteiro, intensificando e enriquecendo o compartilhamento e a prestação de serviços no cotidiano dessas pessoas (Rose, 1996).

Este Capítulo está organizado por duas sessões principais. A seção 2.2, traz uma breve descrição histórica da Internet, bem como de sua estrutura geral, apresentando de forma sucinta dois importantes modelos de referência para protocolos: o *ISO/OSI*¹ e o *TCP/IP*². A seção 2.3 apresenta uma visão geral da arquitetura Web, listando seus tipos

¹*International Organization for Standardization / Open System Interconnection.*

²*Transmission Control Protocol / Internet Protocol.*

de documentos e descrevendo, de forma geral, o protocolo *HTTP*³, o qual embasa sua arquitetura. Também serão apresentados conceitos de servidor Web e servidor Proxy.

2.2 A Internet

A Internet é uma rede de computadores mundial, isto é, uma rede que conecta milhões de equipamentos de comutação em todo o mundo. Não faz muito tempo, esses equipamentos eram prioritariamente *PCs* tradicionais de mesa, estações de trabalho com sistemas Unix e os chamados servidores que armazenam e transmitem informações, como páginas Web e mensagens de *e-mail*. Porém, outros sistemas finais estão sendo utilizados, tais como: *TVs*, telefones celulares, câmeras Web, entre outros. Realmente, o termo *rede de computadores* está começando a soar um tanto desatualizado (Kurose e Ross, 2006).

A acelerada expansão da Internet em grande parte deve-se ao surgimento de uma aplicação que passou a proporcionar o compartilhamento de recursos e serviços de uma forma bastante similar à associação de idéias utilizadas pelo ser humano (Berners-Lee et al., 1992). A Web, como tornou-se conhecida, tem contribuído para uma interatividade cada vez maior entre as pessoas, servindo para habilitar e disponibilizar centenas de novas aplicações, como negociação de ações e serviços bancários *on-line*, serviços multimídia e serviços de recuperação de informações.

Quanto à sua estrutura, a *Internet* basicamente é composta por diversas redes interconectadas por meio de *gateways* e um protocolo de interconexão que tornam transparentes os detalhes fundamentais de cada rede interligada, a fim de prover um serviço uniforme ao longo destas redes.

Embora cada rede possa ser constituída por uma tecnologia diferente e com regras específicas de transmissão, todos os sistemas finais interligados por estas redes apresentam uma visão comum da “rede”. Este poder de interconexão abstrata da Internet é conseguido com o emprego de protocolos e algoritmos comuns (Rose, 1996).

2.2.1 Modelo de Referência *TCP/IP*

Dentre os vários protocolos de interconexão que têm sido utilizados na *Internet*, os Protocolos de *Internet TCP/IP* (ou simplesmente *TCP/IP*) são os mais amplamente utilizados. Desenvolvidos com o objetivo de oferecer um serviço universal, independente da tecnologia das redes interconectadas, eles surgiram junto com a Internet e formam o primeiro conjunto de protocolos desenvolvidos para ela (Comer, 2000).

Essa ampla utilização se deve ao fato de ser um padrão aberto de protocolo e amplamente suportado, independente da tecnologia do dispositivo de rede, e que provê um

³*Hypertext Transfer Protocol*.

esquema de endereçamento comum para identificação única de dispositivos em toda a rede.

O modelo *TCP/IP* está organizado em camadas, onde cada uma provê seu serviço à camada imediatamente acima, e utiliza os serviços oferecidos pela camada diretamente abaixo. Essa modularidade facilita a atualização de componentes de sistema. A pilha está composta de cinco camadas ([Kurose e Ross, 2006](#)):

- Camada de aplicação;
- Camada de transporte;
- Camada de rede;
- Camada de enlace;
- Camada física.

Camada de Aplicação

A camada de aplicação é onde residem aplicações de rede e seus protocolos. Ela inclui diversos protocolos, tais como: o protocolo *HTTP* (*Hypertext Transfer Protocol*) que provê requisição e transferência de documentos pela Web; o *SMTP* (*Simple Mail Transport Protocol*) que provê transferência de mensagens de correio eletrônico; o *FTP* (*File Transfer Protocol*) que provê transferência de arquivos entre dois sistemas finais e o *DNS* (*Domain Name System*), utilizado para resolver domínios de acordo com o *IP* (*Internet Protocol*) ([Kurose e Ross, 2006](#)).

Camada de Transporte

A finalidade dessa camada é permitir que as entidades pares dos *hosts* de origem e de destino mantenham uma conversação, exatamente como acontece na camada de transporte *OSI*. Dois protocolos fim a fim foram definidos aqui. O primeiro deles, o *TCP*, é um protocolo orientado a conexões confiável, que permite a entrega sem erros de um fluxo de bytes originário de uma determinada máquina em qualquer computador da inter-rede. O segundo é um protocolo não orientado a conexão, é basicamente o *IP* com um pequeno cabeçalho ([Tanenbaum, 2003](#)).

Camada de Rede

A camada de rede da Internet encaminha um datagrama por meio de uma série de comutadores de pacotes entre a origem e o destino. Para levar um pacote de um nó ao nó seguinte na rota, a camada de rede depende dos serviços da camada de enlace. Particularmente em cada nó, a camada de rede passa o datagrama para a camada de

enlace que o entrega, ao longo da rota, ao nó seguinte, no qual o datagrama é passado da camada de enlace para a de rede (Kurose e Ross, 2006).

Camada de Enlace

Os serviços prestados pela camada de enlace dependem do protocolo específico empregado nessa camada. Alguns protocolos provêm entrega garantida entre enlace, até o nó receptor. Note que esse serviço confiável de entrega é diferente do serviço de entrega garantida do *TCP*, que provê serviço de entrega garantida de um sistema final a outro. Exemplos de protocolos de camada de enlace são *Ethernet* e *PPP* (*point-to-point protocol*). Como datagramas normalmente precisam transitar por diversos enlaces para percorrer o caminho desde a origem até o destino. Esses podem ser manuseados por diferentes protocolos de camada de enlace em diferentes enlaces ao longo de sua rota, podendo passar por *Ethernet* em um enlace e por *PPP* no seguinte. Assim, a camada de rede receberá um serviço diferente de cada um dos variados protocolos de camada de enlace (Tanenbaum, 2003).

Camada Física

Enquanto a tarefa de camada de enlace é movimentar quadros inteiros de um elemento da rede até um elemento adjacente, a camada física é responsável por movimentar os *bits* individuais que estão dentro do quadro de um nó para o seguinte. Os protocolos nessa camada dependem do enlace e, além disso, dependem do próprio meio de transmissão do enlace (par de fios trançados ou fibra ótica monodal). A *Ethernet* tem muitos protocolos de camada física: um par de fios trançados, um para cabo coaxial, outro para fibra e assim por diante. Em cada caso, o *bit* é movimentado pelo enlace de um modo diferente (Kurose e Ross, 2006).

2.3 A World Wide Web

A Web teve início em 1989 no *CERN* (*Organisation Européenne pour la Recherche Nucléaire*), o centro europeu para pesquisa nuclear. O *CERN* tem vários aceleradores de partículas, nos quais grandes grupos de cientistas dos países europeus participantes desenvolvem pesquisas na área da física de partículas. Esses grupos são quase sempre compostos por membros de mais de meia dúzia de países diferentes. A maioria das experiências é altamente complexa e exige anos de planejamento para a construção dos equipamentos necessários. A Web nasceu da necessidade de fazer com que esses grupos de cientistas de diferentes nacionalidades pudessem colaborar uns com os outros através da troca de relatórios, plantas, desenhos, fotos e outros documentos (Tanenbaum, 2003).

Em 1994, o *CERN* e o *MIT* (*Massachusetts Institute of Technology*) assinaram um acordo criando o *World Wide Web Consortium* (às vezes abreviado como *W3C*), uma organização voltada para o desenvolvimento da Web, a padronização de protocolos e para o incentivo à interoperabilidade entre os sites.

Por meio da *Web* os usuários não só passaram a selecionar as informações do modo que queriam, e quando queriam, diferentemente do que até então era oferecido pelos grandes meios de comunicação em massa, como o rádio e a televisão, mas também passaram a ter a oportunidade de disponibilizar pela Internet suas próprias informações de forma muito mais fácil (Kurose e Ross, 2006).

A arquitetura da Web basicamente está suportada por quatro conceitos: *hipertexto*, sistema de referência entre documentos, e suas partes, que permite ao usuário seguir os *links* até a informação desejada, *identificadores de recurso*, que permitem que um dado recurso (máquina, documento ou outro recurso) seja localizado na rede por meio de um identificador único, modelo *cliente-servidor*, em que uma máquina cliente procura um certo recurso ou serviço à em máquina servidora, *linguagem de marcação*, que é um conjunto de códigos aplicados a um texto indicando ao computador como exibir suas informações.

Do ponto de vista do usuário, a Web é uma vasta coleção mundial de documentos, também chamados páginas Web, que são acessíveis pela *Internet*. As páginas podem estar interligadas por meio de vínculos (*links*), de acordo com o sistema de referências denominado hipertexto. Através dos *links*, o usuário pode “navegar” pelos documentos até encontrar a informação desejada. Estes documentos podem conter outros itens além de texto, como imagens, arquivos de áudio.

Dois são os elementos importantes para a implementação da Web: navegador (*browser*) e servidor. O primeiro é um programa aplicativo cliente responsável por buscar o documento solicitado, interpretar seu texto e seus comandos de formatação e exibir a página e seus possíveis objetos ao usuário. O segundo é responsável por atender requisições por páginas Web específicas. Como um dado servidor pode manter diversas páginas, é necessário identificá-las de forma única. Para isto utiliza-se como nomenclatura a *URL*⁴, basicamente específica: o protocolo utilizado, o nome do servidor acessado através do *DNS*, a porta pela qual o servidor recebe as conexões (padrão porta 80), o diretório onde o arquivo está armazenado e o nome do arquivo.

Os documentos Web podem ser classificados em:

- **Documentos estáticos:** documentos estáticos são simples arquivos que ficam armazenados em algum servidor esperando o momento de serem recuperados. Podem ser texto ou arquivos binários (vídeos, imagens, entre outros). Um dos tipos de arquivos de texto mais populares são os arquivos *HTML* (*HyperText Markup Lan-*

⁴ *Uniform Resource Locator*.

guage).

- **Documentos dinâmicos:** um documento Web dinâmico só passa a existir quando um *browser* o requisita ao servidor, que por sua vez utiliza um programa auxiliar que acessa as informações necessárias e cria o documento. O conteúdo da resposta pode variar de uma requisição para outra. Tais documentos são utilizados em aplicações com informações atualizadas frequentemente, como preços de mercadorias, condições climáticas, entre outras.

2.3.1 Descrição geral do *HTTP*

O *HTTP* é um protocolo da camada de aplicação amplamente utilizado para a transferência de dados na Internet. Ele está definido no *RFC* 1945 (Berners-Lee et al., 1996) e no *RFC* 2616 (Fielding et al., 1999). O *HTTP* está implementado em dois programas: um programa cliente e um programa servidor. Por meio de padrões de mensagens requisição/resposta *HTTP* bem definidos, ele permite que um cliente (um *browser*, por exemplo) interaja com um servidor Web, recebendo dele recursos e serviços. O *HTTP* assume receber das camadas inferiores um serviço confiável e orientado a conexão (Kurose e Ross, 2006). Dentre as características básicas oferecidas por ele, pode-se citar:

- Não mantém o estado das requisições recebidas;
- Possibilita transferência bidirecional de dados;
- Oferece capacidade de negociação entre o cliente e o servidor;
- Provê suporte a *caching* de páginas Web, a fim de melhorar o desempenho.

2.3.1.1 Conexão Persistente

O paradigma das primeiras versões do *HTTP* era o de estabelecer uma nova conexão a cada transferência de dados, sinalizada por uma requisição. No entanto, como um documento Web pode conter vários objetos, para a solicitação de um documento completo seriam necessárias múltiplas conexões *TCP*. Com o *HTTP*/1.1 este paradigma foi mudado, sendo possível a transferência de múltiplas requisições e respostas por uma mesma conexão *TCP*. Quando o cliente ou o servidor estiverem prontos para encerrar a conexão, um lado informa ao outro. A vantagem é uma menor latência de resposta (devido à minimização do tempo de partida lenta do *TCP*, gasto no início de cada nova conexão para o controle de congestionamento), menor *overhead* sobre a rede, economia de memória para *buffers*, além de diminuir tempo gasto com *CPU* (Fielding et al., 1999).

Requisição *HTTP*

Uma requisição *HTTP* especifica ao servidor o que fazer, sobre quem fazer e como fazer, consistindo basicamente de uma linha inicial (denominada *request-line*) composta por três campos, com um caractere de espaço separando dois campos consecutivos, e finalizada pelo conjunto de caracteres *Carriage Return e Linefeed* (CRLF). O primeiro campo indica o método a ser utilizado pelo servidor no tratamento da requisição (o que fazer), o segundo campo indica o recurso a ser manipulado (sobre quem fazer) e o terceiro campo indica a versão do protocolo *HTTP* a ser utilizado (como fazer). O conjunto de métodos disponibilizados pelo *HTTP/1.1*, e especificado pelo primeiro campo, encontra-se descrito na Tabela 2.1.

Tabela 2.1: Métodos do protocolo *HTTP/1.1*.

Método	Funcionalidade
GET	Solicita que seja retornado o recurso identificado pela <i>URL</i> .
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 dados para serem processados (por exemplo, dados de um formulário <i>HTML</i>) para o recurso especificado. Os dados são incluídos no corpo do comando.
OPTIONS	Recupera os métodos <i>HTTP</i> que o servidor aceita.
PUT	Envia certo recurso.
DELETE	Exclui o recurso.
TRACE	Ecoa o pedido, de maneira que o cliente possa saber o que os servidores intermediários estão mudando em seu pedido.
CONNECT	Serve para uso com um <i>proxy</i> que possa se tornar um túnel <i>SSL</i> (um túnel pode ser usado, por exemplo, para criar uma conexão segura).

Além da linha inicial, uma requisição pode conter linhas de cabeçalho (*header lines*) e o corpo de uma mensagem (*entity body* - transportando algum tipo de dado). As linhas de cabeçalho permitem ao cliente transmitir informações adicionais sobre a mensagem, e sobre si mesmo, ao servidor, como os tipos de documentos aceitáveis, a linguagem de preferência para os hipertextos, identificação do agente do cliente, etc. Ao término de cada linha de cabeçalho existe um finalizador idêntico ao da linha inicial (o CRLF). No caso de haver um corpo de mensagem a ser transmitido junto à requisição, depois das linhas de cabeçalho, e antes dos dados do corpo da mensagem existe uma linha separando-os (*blank line*), composta apenas pelos caracteres CRLF.

Figura 2.1: Requisição HTTP (Kurose e Ross, 2006).

Resposta *HTTP*

Uma mensagem de resposta *HTTP* possui uma linha inicial (*response-line*) num formato semelhante ao da *request-line*, com três campos separados por um caractere de espaço entre eles, e com o finalizador **CRLF**. O primeiro campo indica a versão do protocolo *HTTP*, o segundo campo indica o código de *status* da resposta (número de três dígitos) e o terceiro campo traz uma descrição curta sobre o código de *status*. O primeiro dígito do código de *status* indica a classe de resposta, enquanto que os outros 2 dígitos definem sua categoria. Existem 5 possíveis classes de respostas:

- **1xx:** *Informação* - Requisição recebida, continuando o processo;
- **2xx:** *Sucesso* - A ação foi recebida com sucesso, interpretada e aceita;
- **3xx:** *Redirecionamento* - Mais de uma ação deve ser realizada a fim de completar a requisição;
- **4xx:** *Erro do Cliente* - A requisição contém erro de sintaxe e não pode ser realizada;
- **5xx:** *Erro do Servidor* - O servidor falhou ao tentar realizar a requisição aparentemente válida.

A mensagem de resposta é composta ainda por linhas de cabeçalho (*header lines*) finalizadas por **CRLF**, por uma linha vazia (*blank line*) contendo apenas o finalizador **CRLF**, e após esta linha pelo corpo de dados da mensagem (*entity body*). Uma dessas linhas de cabeçalho especifica o tipo de dado transportado pelo corpo da mensagem (indicado por um *MIME type*), para que o cliente saiba como interpretá-lo. A Figura 2.2 ilustra a estrutura de uma resposta *HTTP*.

Figura 2.2: Resposta HTTP (Kurose e Ross, 2006).

Transação *HTTP*

Um servidor Web possui um conjunto de recursos organizados em uma hierarquia de documentos, disponibilizando-os por meio de regras bem definidas pelo protocolo *HTTP*. A seguir é descrito passo a passo o funcionamento geral de uma transação *HTTP* utilizando o método *GET*.

Figura 2.3: Uma transação HTTP utilizando o método GET (Kurose e Ross, 2006).

Ao receber uma requisição, o servidor poderá ter noção do que fazer, sobre quem realizar e de que forma. Caso o cumprimento da requisição seja possível, o servidor localiza o documento no disco e, caso o mesmo esteja disponível, o envia junto com informações de cabeçalho ao cliente. Uma dessas informações é o *MIME type*, que indica

ao cliente como interpretar o dado incluso no corpo de dados da mensagem. Ao término do tratamento da requisição, o servidor encerra a conexão *TCP* com o cliente.

2.4 Servidor Web

As pessoas utilizam a Web a partir de sistemas finais, tais como: TVs, telefones celulares, câmeras Web, entre outros através de um software chamado navegador, isto é o sistema cliente. O sistema cliente recupera as informações, com a forma de documentos de texto, imagens e outros tipos de informação, tudo isso a partir de outros computadores, que são chamados de “servidores” (Callaway, 2001).

Um servidor Web, também conhecido como servidor *HTTP*, está encarregado de várias tarefas, dentre elas estão: reforçar as políticas de segurança, armazenar arquivos frequentemente solicitados em *cache*, registrando pedidos (*log*), e muitas outras tarefas. Os servidores Web são os pilares da Web.

O servidor Web recebe requisições de navegadores Web pedindo documentos, por meio de uma conexão de rede. Ele decifra a requisição para determinar qual arquivo é solicitado, procura o arquivo e, se estiver disponível, o envia para o navegador Web através da conexão de rede que ficou aberta. Dependendo do protocolo (Fielding et al., 1999) o servidor pode reutilizar essa conexão ou criar uma nova conexão.

O Servidor Web pode ser entendido pela seguinte formula (Callaway, 2001):

Servidor Web = Plataforma + Software + Informações

- **Plataforma:** É a parte que fornece a base para o servidor funcionar e está composta pelo computador (*hardware*), o sistema operacional e o software de rede;
- **Software:** O servidor Web;
- **Informações:** A mais importante, as informações que o servidor contém. Sem elas o Servidor Web não teria sentido.

2.4.1 Modelos de Atendimento

Um processo por requisição

Uma das principais características deste modelo é estar baseado na criação de processos para cada atendimento. Um exemplo disso pode-se ver nos sistemas UNIX, onde o modelo multitarefa utiliza a função *fork()* para atender uma requisição do cliente no servidor Web. O novo processo é escalonado junto com os outros processos do sistema operacional (Bach, 1986).

A criação do novo processo tem um custo relativamente alto. Outro problema com esta forma de funcionamento é o limite de requisições que podem ser atendidas pelo servidor.

Isto está limitado à quantidade de processos que o sistema pode criar ou especificamente clonar. Se o servidor Web estiver compartilhando recursos com outros programas a criação de processos em grandes quantidades pode gerar problemas de desempenho para outros programas que estão sendo executados no mesmo computador.

Figura 2.4: Um processo por cada requisição (Callaway, 2001).

O *overhead* gerado no atendimento das requisições não deveria ser muito, além de sofrer *delays* quando o sistema está ocupado por outros processos.

Um processo para várias requisições

Uma alternativa para evitar o problema do uso excessivo de recursos, é utilizar um processo só. Cada requisição é tratada por um *thread* criado a partir do programa principal (Black, 1990).

Para um servidor Web multitarefa cada *thread* está encarregado de uma requisição que chega no sistema, fazendo uma cópia da conexão para depois resolvê-la. Todos os *threads* estão compartilhando as informações do programa principal, sendo esta uma das principais vantagens da sua utilização. Na Figura 2.5, é possível ver como este modelo resolve as requisições que vão chegando no sistema.

Figura 2.5: Um processo usa vários *threads* para resolver as requisições (Callaway, 2001).

Outra característica da utilização dos *threads* está diretamente relacionada com ganhos no desempenho do servidor, evitando o *overhead* na cópia do processo pai para o processo

filho por cada requisição que está chegando ao sistema. O ganho no desempenho já acontece em sistemas monoprocessados, agora, para sistemas com vários processadores o ganho ainda é maior (Narlikar e Blleloch, 1998).

Processo auxiliar

Este modelo está focado na criação de alguns processos do servidor Web e um *dispatcher*, este processo está encarregado de distribuir as requisições para cada processo do servidor Web que estiver disponível. Cada requisição é resolvida pela cópia separada do servidor Web, então as complexidades internas do servidor *multithread* são evitadas. Os processos são criados apenas uma vez, para depois ser reutilizados. Com isto a sobrecarga da criação de processos é quase eliminada melhorando o desempenho geral do sistema.

O *dispatcher* aceita inicialmente a requisição feita pelo cliente, depois repassa a requisição para um dos processos cópia que estão executando na máquina servidora. Se todos os processos cópias estão resolvendo uma requisição, então um deles faz uma cópia dele mesmo para resolver essa nova requisição. Na Figura 2.6 é possível visualizar o funcionamento deste modelo.

Figura 2.6: O processo *dispatcher* repassa a requisição ao servidor Web (Callaway, 2001).

2.4.2 Servidor Proxy

Um servidor *proxy* é um software que atende as requisições de clientes repassando os dados da rede dos clientes para outra rede (*Internet*). Os servidores *proxy* são usados frequentemente para fazer *cache* de arquivos ou para monitorar o uso da Internet dentro de uma corporação (Callaway, 2001).

Os servidores *proxy* também fornecem algum grau de segurança da rede interna. *Firewalls* e servidores *proxy* são comumente usados em conjunto para fornecer uma estratégia abrangente de segurança de rede.

2.4.3 Lista de Servidores Web

Existe uma grande quantidade de servidores *Web* na atualidade. Alguns dos mais populares segundo (NETCRAFT, 2010) são apresentados junto com suas porcentagens de utilização na Web na Tabela 2.2:

Tabela 2.2: Totais por servidores ativos em todos os domínios (NETCRAFT, 2010).

Desenvolvedor	Mai 2010	Porcentagem
Apache	46 608 654	55.36%
IIS (Microsoft)	14 977 560	17.79%
GWS (Google)	10 064 872	11.95%
nginx	7 387 460	8.77%
lighttpd	339 862	0.40%

No primeiro lugar encontra-se o *Apache*⁵ seguido do *IIS*⁶, no terceiro lugar está o servidor *Web* da *Google* (esta é uma versão especial do servidor Apache, contém algumas características particulares desenvolvidas pela *Google*). No quarto lugar encontra-se o *Nginx*⁷, a característica principal deste servidor é ser um *proxy* reverso *HTTP* de alta performance. No quinto lugar está o *lighttpd*⁸, uma das características principais desse servidor é o baixo nível de recursos que utiliza.

2.5 Considerações Finais

Este Capítulo abordou a infra-estrutura da Web, assunto de grande importância para o desenvolvimento do trabalho. Primeiramente foi introduzido o modelo de referência a pilha de protocolos *TCP/IP*. Também foi abordado o protocolo *HTTP*, bem como as mensagens de requisição e resposta, além de suas características.

Um breve conhecimento de como a estrutura da Web funciona é importante para se entender quais as características atuais da Internet.

Também foi feita uma introdução aos conceitos de servidor Web e os tipos de atendimento que utiliza no seu funcionamento, e uma pequena introdução ao conceito de servidor *Proxy*.

⁵<http://httpd.apache.org/>

⁶<http://www.iis.net>

⁷<http://www.nginx.org/>

⁸<http://www.lighttpd.net/>

Serviços Diferenciados e Tempo Real

3.1 Considerações Iniciais

No modelo atual de serviços da *Internet*, as requisições são tratadas, de forma geral, de modo equivalente, ou seja, a entrega dos dados é realizada igualmente a todos os usuários. Nesse serviço, cada usuário compartilha a capacidade de transmissão com todos os fluxos de dados, os quais percorrem uma rota possível para chegar ao seu destino, de acordo com as rotas definidas e a capacidade de banda que estiverem disponíveis no momento.

E, acrescentando a complexidade do ambiente, existe uma crescente utilização de sistemas computacionais interativos na realização de serviços, requisitos de desempenho e confiabilidade associados à responsividade de aplicações como ensino à distância, telemedicina, comércio eletrônico, entre outros (Xiangbin, 2008; Monaco et al., 2009).

Na ocorrência de congestionamento, não há garantia de desempenho bem como de sucesso na realização do serviço. Nesses casos, atrasos ou descartes de pacotes de dados podem ocorrer (Vasiliou, 2000). Seria, então, conveniente, a diferenciação desses serviços. Portanto, a utilização do paradigma de **Qualidade de Serviço** torna-se cada vez mais importante e uma necessidade para a evolução da *Internet*, principalmente no segmento da utilização comercial (Bhatti et al., 2000; Strnadl, 2002). A ISO (*International Organization for Standardization*) define QoS como efeito coletivo do desempenho de um serviço, o qual determina o grau de satisfação de um usuário (ISO/IEC, 1998), ou seja, a percepção do usuário quanto à eficiência de um serviço (Peixoto e Monaco, 2008).

3.2 Conceitos Básicos de QoS

Para a ISO, QoS é definida como o efeito coletivo do desempenho de um serviço, o qual determina o grau de satisfação de um usuário (ISO/IEC, 1998). Essa definição é genérica e deve ser especificada para o problema que se deseja tratar.

As principais formulações de QoS são: garantia de desempenho e diferenciação de serviço. O primeiro está diretamente relacionado à largura de banda, atraso e perda de pacotes, enquanto que, a diferenciação de serviço refere-se aos diferentes níveis de prioridade a diversas aplicações, cada qual com requisitos distintos.

3.2.1 QoS em Nível de Rede

Em nível de rede, QoS pode ser definida como a capacidade de um elemento em prover certo grau de garantia do cumprimento dos requisitos de seu tráfego e serviços oferecidos, tais como níveis mínimos de perdas e atrasos de dados requisitados pelo usuário.

Dentre as várias abordagens existentes para provisão de QoS na *Internet*, destacam-se duas delas desenvolvidas sob a coordenação da IETF (*Internet Engineering Task Force*): a de **Serviços Integrados** (*IntServ - Integrated Services*) (Braden et al., 1994) e a de **Serviços Diferenciados** (*DiffServ - Differentiated Services*) (Blake et al., 1998).

Serviços Integrados

A abordagem de Serviços Integrados é caracterizada pela reserva de recursos, que consiste em alocar previamente os recursos de rede necessários para que se obtenha uma boa qualidade na transmissão de dados das aplicações. Para tanto, tais aplicações utilizam o protocolo RSVP (*Resource Reservation Protocol*), um protocolo de controle e sinalização atuante na camada de rede da pilha de protocolos TCP/IP. O funcionamento do RSVP é ilustrado pela Figura 3.1.

Figura 3.1: Ilustração do Funcionamento de Sinalização do *RSVP* (Zhao et al., 2000).

Conforme pode ser observado pela Figura 3.1, para a requisição dos recursos, um sistema emissor (*sender*) envia uma mensagem *PATH* ao receptor especificando os recursos que necessita. Cada roteador intermediário (*router*) repassa a mensagem para o próximo

roteador, de forma que a mensagem chegue ao receptor (*receiver*). Após receber a mensagem *PATH*, o receptor envia uma mensagem *RESV* de volta ao emissor pelo caminho contrário da mensagem *PATH*, tentando, dessa forma, solicitar a reserva de recursos para o fluxo de dados. Cada roteador intermediário pode aceitar ou rejeitar a reserva de recursos solicitada, de acordo com a quantidade de recursos disponíveis. Caso a requisição seja rejeitada, uma mensagem de erro é enviada de volta ao receptor e a sinalização é encerrada. Caso contrário, os recursos necessários para o fluxo são alocados e as informações de estado do fluxo de dados são armazenadas no roteador (Zhao et al., 2000).

No entanto, com o aumento do fluxo, aumenta-se, consideravelmente, a quantidade de informações de estado, devido à reserva de recursos a ser realizada para cada fluxo individualmente. Sendo assim, são necessários uma grande capacidade de processamento e um elevado poder de processamento dos roteadores, tornando o núcleo da rede mais complexo e, conseqüentemente, prejudicando a escalabilidade da *Internet*. Além disso, todos os roteadores ao longo da rota devem oferecer suporte a serviços integrados. Tais problemas acabam dificultando a implementação e a implantação desse tipo de abordagem. De forma a solucionar os problemas mencionados, surge uma abordagem alternativa, denominada serviços diferenciados.

Serviços Diferenciados

A abordagem de Serviços Diferenciados é caracterizada pela definição de tipos de serviços (Magalhães e Cardozo, 1999). Sendo assim, o gerenciamento pelos roteadores torna-se mais simples, pois os fluxos são agregados e os roteadores não precisam guardar o estado de cada fluxo individualmente, como é o caso da abordagem de Serviços Integrados.

No cabeçalho de um pacote IP (*Internet Protocol*), os tipos de serviços podem ser representados por um campo denominado TOS (*Type of Service*). Define-se um novo *layout* (DSField – *Differentiated Service Field*) para esse campo, conforme descrito em Nichols et al. (1999). Com base nesse *layout*, um conjunto de procedimentos distintos de envio de pacotes é especificado, oferecendo, dessa forma, diferentes classes de serviços.

Em uma solicitação de serviço diferenciado, é importante o estabelecimento de um contrato de nível de serviço, firmado entre o usuário e o provedor de serviços de *Internet* (ISP - *Internet Service Provider*). Esse contrato, denominado SLA (*Service Level Agreement*) determina as classes de serviços suportadas e a quantidade de tráfego na banda entre os domínios.

Contratos SLA podem ser definidos como estáticos ou dinâmicos. Um SLA é considerado estático quando negociado de forma regular (mensalmente ou anualmente). Já em um SLA dinâmico, os serviços podem ser solicitados sob demanda. Sendo assim, faz-se necessária a utilização de algum protocolo de sinalização e controle (como o RSVP) de forma a indicar as necessidades do usuário em determinados momentos (Zhao et al., 2000).

Portanto, a abordagem empregada pelo *DiffServ* baseia-se em um esquema de priori-

dades relativas, ou seja, ele não garante que o tráfego gerado por uma aplicação alcançará o seu destino dentro de um limite máximo de tempo, ou com uma variação máxima de atraso, mas sim que um tráfego com um certo nível de prioridade receberá um atendimento melhor que qualquer outro que possua uma prioridade inferior (Xiao e Ni, 1999).

Tal abordagem foi utilizada em Chen e Heidemann (2003), no qual é proposto um algoritmo denominado SFD (*Short Flow Differentiating*) com o intuito de melhorar o desempenho de tráfego *Web* interativo. O algoritmo proposto prioriza tráfegos mais curtos, como os de rajada, visto que, estudos mostraram que tráfegos mais curtos correspondem a mais de 80% das respostas *Web*, além de fluírem mais rapidamente pela rede. Os resultados apresentados, por meio de simulação, mostraram a importância e o impacto na priorização de serviços mais curtos, com uma melhora de mais de 30% no tempo de resposta das transmissões mais curtas.

3.3 QoS em Nível de Aplicação

Nas seções anteriores, foram abordadas as clássicas técnicas de garantia de QoS em nível de rede, particularmente as abordagens de serviços integrados e serviços diferenciados. Entretanto, a provisão de QoS seria mais abrangente e propiciaria melhores resultados se houvesse a cooperação das outras camadas, especificamente da camada de aplicação, a qual permite processar e analisar informações mais complexas.

Alguns trabalhos têm sido desenvolvidos com o intuito de viabilizar e evoluir as formas de disponibilização de QoS em nível de aplicação, tanto em termos relativos quanto em termos absolutos (Almeida et al., 1998; Eggert e Heidemann, 1999). Diversos pesquisadores têm realizado esforços particularmente no contexto de servidores *Web* (Kanodia e Knightly, 2003; Pan et al., 2008; Lee et al., 2003; Abdelzaher et al., 2002; Pradhan et al., 2002; Wei et al., 2005; Monaco et al., 2009; Casagrande e Monaco, 2007). Nesse contexto as pesquisas mais preponderantes abordam controle de admissão e diferenciação de serviço (Teixeira et al., 2005; Estrella et al., 2006; Traldi et al., 2006; Messias et al., 2007).

3.3.1 QoS Relativa

Este tipo de qualidade de serviço está caracterizado pela definição de uma relação de prioridade com base a um contrato de serviço. Nessa abordagem a diferenciação entre classes se dá de maneira a garantir um melhor atendimento a uma determinado usuário mais prioritário, cujo valor estabelecido em contrato é menor em relação às demais classes, as quais apresentam um valor de contrato maior. No caso específico de tempo de resposta, um contrato de QoS relativa especifica que as requisições provenientes de determinada classe serão atendidas em geral antes daquelas provenientes de uma classe menos prioritária. Trabalhos abordando QoS relativa, mais frequentes na literatura, capturam a

noção intuitiva de atendimento preferencial comum aos modelos de prestação de serviço convencional de provedores comerciais, oferecendo classes de serviço com prioridades.

Inspirado no modelo *DiffServ*, [Teixeira et al. \(2005\)](#) realiza uma aplicação de seus princípios para a camada de aplicação, concebendo um modelo de servidor *Web* com diferenciação de serviços (SWDS). Esse tipo de abordagem está relacionada à QoS relativa, usualmente mais comum em servidores *Web* ([Henriksson et al., 2004](#); [Kang et al., 2003](#); [Lu et al., 2001](#)). A arquitetura de servidor *Web* proposta fornece serviços diferenciados a seus clientes de acordo com suas características de demanda. Nessa arquitetura são propostos mecanismos de diferenciação de serviços baseados em prioridades e um módulo de controle de admissão com o objetivo de garantir estabilidade na qualidade de serviço oferecida. A arquitetura basicamente é composta por um módulo classificador de requisições, por um controle de admissão e um *cluster* de servidores *Web*. Quando admitida, a requisição é atribuída a um dos nós do *cluster* segundo o algoritmo de escalonamento ou diferenciação de serviços vigente.

Em [Estrella et al. \(2006\)](#) estende-se o trabalho realizado em [Teixeira et al. \(2005\)](#), incluindo um mecanismo de negociação ao módulo de controle de admissão do modelo SWDS, com o objetivo de permitir que requisições de determinadas classes de serviços tenham prioridade no atendimento em relação às classes inferiores, mesmo em situações de sobrecarga nos servidores *Web*. Para tanto, são propostos dois algoritmos: Algoritmo de Negociação Rigoroso (ANR) e Algoritmo de Negociação com o Cliente (ANC).

Em [Traldi et al. \(2006\)](#) também é abordada a QoS em termos relativos, propondo e comparando dois novos algoritmos de diferenciação de serviços com o intuito de prover qualidade de serviço em servidores *Web*. Sendo eles o Reserva Adaptativa de Recursos (RSVAdap) e o *Weighted Fair Queuing* (WFQ). Esses algoritmos foram adicionados ao modelo proposto em [Teixeira et al. \(2005\)](#) e simulados sem a utilização de políticas para o controle de admissão e em sistemas homogêneos e heterogêneos. O algoritmo RSVAdap é baseado na Reserva de Recursos, realizando a alocação de recursos de forma dinâmica, ou seja, sob demanda, segundo a carga de trabalho vigente.

No trabalho ([Messias e Santana, 2007](#)) é apresentado um estudo e implementação de um modelo de servidor *Web* com diferenciação de serviços, esse protótipo está baseado no modelo proposto por ([Teixeira et al., 2005](#)). Com o intuito de controlar a carga no sistema foram implementados algoritmos de reserva de recursos, escalonamento baseado em prioridades e mecanismos de controle de admissão.

3.3.2 QoS Absoluta

Nessa modalidade, são utilizadas métricas e valores de desempenho para cada classe individualmente. Nessa abordagem, existe a garantia por usuário ou classe, independente do que é definido para os demais. Nesse caso, a prioridade é definida dinamicamente em

função do estado do sistema e das condições efetivas dos contratos. São estabelecidas taxas mínimas de serviço ou atrasos máximos de atendimento para as requisições. No caso específico de tempo de resposta, um contrato de QoS absoluta especifica limites para o tempo de resposta das requisições de cada usuário ou classe, independente daquele contratado para as outras classes.

Mais recentemente, têm sido desenvolvidas pesquisas sobre QoS em termos absolutos, com a qual se pode oferecer garantias mais estritas de qualidade, conforme pode ser observado em [Wei et al. \(2005\)](#); [Monaco et al. \(2009\)](#); [Casagrande e Monaco \(2007\)](#); [Peixoto e Monaco \(2008\)](#).

Em [Wei et al. \(2005\)](#) é proposta uma abordagem de controle *fuzzy* para garantir atrasos absolutos em servidores *Web*. A arquitetura do sistema proposto é composta pelos módulos: escalonador de conexão, monitor de sistema e controlador *fuzzy*. O escalonador de conexão é responsável pelo recebimento e aceitação de todas as requisições de conexão que chegam ao sistema. Essas requisições são classificadas em diferentes classes baseadas, por exemplo, em endereços IP dos usuários. Uma nova requisição é alocada a um processo somente se o número de processos atribuídos à determinada classe de requisições é menor que o contador de processos dessa classe. O monitor é responsável pela medição e informação ao sistema do atraso absoluto de cada classe de requisições. O controlador *fuzzy*, por sua vez, é responsável pelo ajuste do contador de cada classe (cálculo do número de processos que são alocadas para cada uma delas), de forma que o atraso das classes não ultrapasse o limite estipulado.

Um estudo realizado em [Casagrande e Monaco \(2007\)](#) introduz uma política de escalonamento de tempo real não-determinístico (*Soft-RT*) para provisão de garantias de tempo de resposta estocásticas em ambientes interativos *online*, mais especificamente em servidores *Web*. No modelo proposto, são definidas duas classes de usuários (A e B), as quais apresentam menores e maiores valores (tempos médios de resposta) estabelecidos em contrato, respectivamente. O parâmetro de QoS contratado por cada usuário é um limite superior para a média do tempo de resposta que é especificado previamente por um acordo entre o provedor de serviços e o usuário. A política elaborada tem como princípio comparar tal limite com o tempo de resposta médio efetivo, que reflete a média instantânea de tempo de resposta de fato praticada, o que possibilita a verificação dos contratos que estejam mais próximos de serem violados.

[Casagrande e Monaco \(2007\)](#) utilizam uma estratégia híbrida associada à heurísticas baseadas nos algoritmos clássicos EDF (*Earliest Deadline First*) e SJF (*Shortest Job First*) (ver Seção 3.5), levando em consideração, portanto, o tempo de espera em fila (urgência de *deadline*) e o custo de execução de uma requisição, impondo ao sistema a menor demanda de recursos possível e garantindo um compromisso entre desempenho e confiabilidade no atendimento de contratos individuais. Nesse trabalho foi demonstrado que a política EBS proporciona resultados superiores à heurísticas convencionais.

Em [Peixoto e Monaco \(2008\)](#), comparam-se um conjunto de combinações de políticas de fila (EDF e EBS) e políticas de recursos (*Multiple Queue* - MQ, *Single Queue* - SQ e *Dynamic Single Queue* - DSQ). No modelo MQ ou *Web switch*, as requisições são enviadas para um servidor qualquer do *array* por meio do balanceamento de carga, de acordo com algumas regras de escalonamento, tais como políticas que levam em consideração as características de carga do servidor. Os modelos SQ e DSQ apresentam apenas uma única fila sendo gerenciada de maneira centralizada. No modelo SQ, a fila de requisições é ordenada baseando-se nos resultados obtidos com uma heurística, sendo o primeiro *job* executado no primeiro servidor livre do *array*. No DSQ, foi implementado um modelo híbrido entre SQ (com fila única) e o *Web switch* (com balanceamento). Nesse modelo, ao invés de atribuir a requisição mais prioritária ao primeiro servidor livre, o DSQ segue o mesmo princípio da EBS, prevendo o impacto causado ao sistema. Sendo assim, para cada requisição da fila de requisições, o DSQ seleciona o servidor que apresenta o menor tempo estimado de término de processamento da requisição em questão. Caso o servidor esteja ocupado, a requisição é colocada em uma das filas virtuais, referente ao seu respectivo recurso. Esse trabalho é o primeiro que utilizou-se à política EBS num ambiente de *cluster* de servidores.

3.4 Sistemas de Tempo Real

Abordagens de tempo real são utilizadas em sistemas que necessitam sincronizar eventos internos com o ambiente (eventos externos ao sistema). Para tanto, é necessário satisfazer as requisições temporais individuais, garantindo que cada requisição seja atendida antes do seu *deadline* (restrição de tempo, correspondendo ao instante máximo em que uma requisição deve ser concluída) ([Cheng, 2002](#)), de forma a não violar a sincronização do sistema com o ambiente externo ([Nissanke, 1997](#)).

Tais sistemas não necessariamente implicam em desempenho computacional, mas sim em garantir o tempo de resposta do sistema aos eventos externos ([Nissanke, 1997](#)).

Portanto, o requisito-chave na formulação das especificações RT é o limite superior para atrasos no tempo de reação do sistema aos estímulos externos, ou seja, restrições sobre máximos tempos de resposta.

Para a descrição e caracterização dos diferentes tipos de sistemas RT, bem como os métodos para escalonamento e gerência de recursos, utilizam-se termos gerais para tratamento da carga de trabalho de sistemas computacionais e de comunicação ([Casagrande e Monaco, 2007](#)). Sendo assim, a unidade de trabalho agendada e executada por um sistema, denomina-se requisição (ou *job*), já o conjunto dessas unidades, o qual realiza determinadas funções no sistema será denominado, neste trabalho, tarefa (ou *task*) ([Liu, 2000](#)).

3.4.1 Sistemas *Hard-RT* e *Soft-RT*

Os sistemas de tempo real podem ser classificados em dois tipos: *Hard Real-Time* e *Soft Real-Time* (Mathes et al., 2008), os quais diferenciam-se segundo a forma de restrição de tempo estabelecida pelo sistema a uma determinada tarefa (Laplante, 2004).

Para **sistemas *Soft Real-Time***, a precisão de tempo é importante, porém não é crítica. Um atraso na conclusão de uma tarefa é indesejável, mas é aceitável, pois não trará sérios danos, apenas uma degradação do desempenho a cada não cumprimento de *deadline*. Nesses sistemas, as tarefas são realizadas tão rapidamente quanto possível, podendo haver certo relaxamento na precisão de alguns de seus tempos em situações de sobrecarga do sistema, conforme pode ser observado em (Calandrino et al., 2007).

Em **sistemas *Hard Real-Time***, é de extrema importância a exatidão do tempo de resposta. Nesses sistemas, o não cumprimento das restrições temporais (como tempo de liberação e *deadline*) pode ocasionar graves consequências (Zhang et al., 2008), como o comprometimento da confiabilidade do sistema ou até mesmo colocar vida de pessoas em risco (Liu, 2000; Tavares et al., 2008). Por serem mais desafiadores, sistemas *Hard-RT* apresentam um custo mais elevado em relação a sistemas *Soft-RT* (Mathes et al., 2008).

3.4.2 Restrições Temporais

Restrições temporais são relevantes, tanto para caracterizar o sistema, quanto para impor o comportamento temporal desejado ou necessário de uma requisição do sistema, durante a execução das tarefas, conforme pode ser observado em (Zhang et al., 2008; Mathes et al., 2008). Dentre elas (Figura 3.2), além do *deadline* (d), têm-se:

- **Tempo de chegada** (*arrival time* - at): correspondente ao tempo em que o escalonador tem conhecimento da ativação de uma requisição;
- **Tempo de liberação** (*release time* - rt): correspondente ao instante em que a requisição está disponível para processamento e, portanto, é incluída na fila de prontas para serem executadas;
- ***Release jitter*** (J): correspondente à variação do retardo na entrega de dados, ou seja, a medida de variação do atraso entre as sucessivas requisições;
- **Tempo de início** (*start time* - st): correspondente ao instante inicial do processamento da requisição;
- **Tempo de execução** (*computation time* - c): correspondente ao tempo gasto para completar a execução de uma determinada requisição. Útil para determinar se uma requisição cumprirá seu *deadline*;

- **Tempo de término** (*completion time* - ct): correspondente ao instante final do processamento da requisição;

Figura 3.2: Ilustração das Restrições Temporais.

- **Tempo de resposta** (*response time* - r): correspondente ao tempo desde o recebimento de um estímulo externo pelo sistema (um conjunto de entradas) até a disponibilização de todas as saídas associadas (resposta).

3.5 Algoritmos de Escalonamento

Em cenários específicos, diversos algoritmos de escalonamento têm sido desenvolvidos, como por exemplo em (Kato e Yamasaki, 2008; Shi-jun et al., 2008).

Alguns são muito importantes em função de sua larga aplicação e relevância científica, conforme pode ser observado em (Leontyev e Anderson, 2007; Devi e Anderson, 2006b,a), dentre eles, pode-se destacar os algoritmos FIFO e SJF, bem como a política EBS (Casagrande e Monaco, 2007), a qual motivou a presente proposta e possibilitou o prosseguimento de numerosas variações e adaptações (Monaco et al., 2009; Peixoto et al., 2008; Tott e Monaco, 2008; Nery e Monaco, 2008; Saito e Monaco, 2010). Também estão sendo visados os algoritmos: WRR (Zhang, 2000) e o clássico RR. Estes algoritmos estão orientados à distribuição de tarefas para um grupo de elementos de processamento (cluster de servidores).

3.5.1 Algoritmos de fila

FIFO

A política de escalonamento mais utilizada do modelo convencional de servidor *Web* é a *FIFO*. Nessa política, as requisições são atendidas na ordem em que chegam ao sistema (Ye et al., 2007, 2005).

Para ilustrar essa situação, considera-se quatro requisições (R_1 , R_2 , R_3 e R_4) em uma fila e seus respectivos tempos de execução, representados em unidades de tempo (ut), $9ut$, $7ut$, $5ut$ e $3ut$.

A Figura 3.3 ilustra a ordem em que as requisições são processadas e os tempos de espera em fila de cada uma dessas requisições, utilizando-se a política *FIFO*. Sendo assim, o tempo médio de resposta proporcionado por este escalonamento seria $(0+9+16+21)/4 = 11.5ut$.

Figura 3.3: Exemplo da Utilização do Algoritmo de Escalonamento FIFO.

Shortest Job Firs (SJF)

Nesta política as requisições são organizadas em uma fila de acordo com o tempo de processamento (T_p), sendo os menores *jobs* colocados à frente. Realocar as requisições mais curtas na frente das que exigem maior tempo de processamento permite que o tempo de espera das mais curtas diminua mais do que aumenta o das requisições mais longas, possibilitando uma diminuição na média de tempo de resposta oferecida pelo sistema.

A Figura 3.4 ilustra a ordem em que as requisições são processadas e seus respectivos tempos de espera, utilizando a política *SJF*. Dessa forma, o tempo médio de resposta é de $(0 + 3 + 8 + 15)/4 = 6.5ut$.

Figura 3.4: Exemplo da Utilização do Algoritmo de Escalonamento SJF.

Exigency-Based Scheduling (EBS)

A política *EBS* (*Exigency-Based Scheduling*) (Casagrande e Monaco, 2007), provê garantias de QoS absoluta em nível de aplicação. Para tanto, utiliza um limite superior

para a média dos tempos de resposta, a ser garantido às requisições de um determinado usuário, como parâmetro de qualidade de serviço. Esse parâmetro, denominado tempo médio de resposta contratado pela i -ésima classe (T_{c_i}), é especificado previamente por um acordo entre o provedor de serviços e o usuário, esse valor é utilizado pelo escalonador como base para atribuição de prioridades.

O seguinte exemplo ilustra como opera o algoritmo da *EBS*. Seja um servidor mono-processador com uma fila única de espera para processamento.

As requisições estão chegando ao sistema e não encontram servidor disponível são transferidos para a fila de espera, visto que a política não é preemptiva. Ao término da execução de uma requisição j do usuário u , o valor de T_u , média instantânea de tempo de resposta efetivamente oferecida ao usuário u , é recalculada. A Equação 3.1 mostra esse cálculo, o qual corresponde à média entre o antigo tempo de resposta efetivo de u (T'_u) e o tempo de residência da requisição j recém atendida. Os valores de $time()$, $timeStamp_j$ e R_u representam, respectivamente, o tempo atual, o tempo de chegada da requisição e o número de requisições anteriormente submetidas por u .

$$T_u = \frac{(T'_u \cdot R_u) + (time() - timeStamp_j)}{R_u + 1} \quad (3.1)$$

Para definir de uma maneira mais clara qual requisição é a mais prioritária, a Equação 3.2 mostra essa atribuição de prioridades, em que a prioridade de uma dada requisição j em fila, do usuário u , é dada por P_j e as requisições que apresentarem maior urgência (D_j) e menor valor esperado do tempo de processamento (T_{p_j}) serão classificadas como mais prioritárias.

$$P_j = D_j \cdot T_{p_j} = ((T_{cu} \cdot (R_u + 1)) - (T_u \cdot R_u) - T_{w_j}) \cdot T_{p_j} \quad (3.2)$$

Em alguns casos o *deadline* pode assumir valores negativos, representando que o tempo que a requisição pode aguardar na fila é menor que zero, ou seja, o contrato foi violado. Se existirem duas requisições (R_1 e R_2) mais urgentes, com seus respectivos *deadlines* (D_i) negativos e tempos de processamento (T_{p_i}), por exemplo, uma requisição R_1 com $D_1 = -1$ e $T_{p_1} = 7.5ut$ e uma requisição R_2 com $D_2 = -2$ e $T_{p_2} = 2.5ut$. Se o escalonamento proposto (multiplicar o *deadline* pelo tempo de processamento) for aplicado nesse exemplo, embora R_2 seja mais urgente, a requisição R_1 receberia maior prioridade e seria inicialmente escalonada, devido $P_1 < P_2$ ($P_1 = -7.5ut$ e $P_2 = -5.0ut$). Nesses casos, é realizada uma correção, de forma a manter o objetivo proposto.

A Equação 3.3 representa tal correção, em que a prioridade das requisições mais urgentes, que ainda não tiveram seus *deadlines* descumpridos, é diretamente proporcional ao seu *deadline* e ao seu custo esperado de processamento. Já para aquelas com descumprimento de *deadline*, a prioridade é inversamente proporcional ao seu custo esperado de processamento, visto que, quanto menor for esse custo, menor será o valor de P_j resultante

e, portanto maior será sua prioridade de escalonamento. Sendo assim, garante-se que as requisições mais urgentes, independente de terem descumprido ou não seus *deadlines*, e com menores custos esperados de processamento sejam escalonadas primeiro.

$$P_j = \begin{cases} D_j \cdot T_{p_j} & \text{se } D_j \geq 0 \\ D_j \cdot \frac{1}{T_{p_j}} & \text{se } D_j < 0 \end{cases} \quad (3.3)$$

3.5.2 Algoritmos de distribuição

Round-Robin (RR)

É um algoritmo muito simples e bem conhecido na literatura, ele envia as tarefas para cada unidade de processamento de forma sequencial. Este algoritmo gerencia as tarefas sem considerar prioridade alguma apenas pela ordem de chegada, fornecendo um modelo simples de execução de tarefas.

Weighted Round-Robin (WRR)

O algoritmo WRR é um algoritmo de agendamento de servidor mais justo do que o anterior. Este algoritmo já foi utilizado com outras tecnologias como ATM ([Shimonishi et al., 1997](#); [Wang et al., 1994](#)).

O algoritmo *Weighted Round-Robin* utiliza um conjunto de servidores, cada um deles pode ter diferentes capacidades de processamento. Este algoritmo aloca pesos a cada servidor. Quanto maior for o peso, maior é a disponibilidade. Baseado na disponibilidade de processamento de cada servidor, o algoritmo estabelece uma sequência de execução na qual serão alocadas as requisições para os diferentes servidores.

Algoritmo 3.5.1 *Algoritmo Weighted Round-Robin* ([Zhang, 2000](#)).

```

enquanto true faça
  i = (i + 1)%n
  se i = 0 então
    cw = w - gcd(S)
    se cw <= 0 então
      cw = max(S)
      se cw = 0 então
        return NULL
    fim-se
  fim-se
  fim-se
  se W(Si) >= cw então
    return Si
  fim-se
fim-enquanto

```

Suponha que S é um conjunto de servidores, $S = \{S_0, S_1, \dots, S_{n-1}\}$:

- $W(S_i)$ indica o peso associado ao servidor S_i ;
- i indica o último servidor selecionado, no início $i = -1$;
- cw é o peso atual no sistema, no início $cw = 0$;
- $\max(S)$ é o máximo peso no conjunto de servidores S ;
- $\gcd(S)$ é a função do máximo comum divisor dos pesos do conjunto de servidores S .

Vamos supor um exemplo para observar melhor como é o funcionamento do algoritmo. As requisições estão chegando e são alocadas para os servidores com maiores disponibilidades em ordem decrescente, Por exemplo, os servidores reais A, B, C tem o 4, 3, 2 como pesos, respectivamente, então a sequência de agendamento vai ser $AABABCABC$ em um período de programação para nove requisições.

O algoritmo usado no protótipo está baseado na implementação do projeto *Linux Virtual Server*, segundo (Zhang, 2000), o algoritmo é eficiente para agendar requisições para um cenário não muito variável (sem muitas mudanças), esse problema acontece quando os tempos de processamento são muito dinâmicos, o algoritmo não consegue cumprir o princípio de agendar requisições para o servidor mais disponível para que essa requisição seja atendida o mais rápido possível. Para corrigir este problema foram feitas atualizações dos estados de cada servidor usando para isso um tempo de 0,5 segundos de intervalo, com essa mudança de tempo em tempo o algoritmo tem informação atualizada para atender ao cenário dinâmico.

3.6 Considerações Finais

O presente capítulo introduziu os principais fundamentos da área de **qualidade de serviço** e **sistemas de tempo real**. Também foram abordados conceitos básicos de *Serviços Integrados* e *Serviços Diferenciados* e *Hard-RT* e *Soft-RT* respectivamente. Além disso, a Secção 3.3 tratou de **QoS em nível de aplicação** abordando QoS Absoluta e Relativa. O Capítulo também apresentou os algoritmos que foram implementados no protótipo, incluindo algumas formas de classificação para os mesmos.

Servidor Proxy com Diferenciação de Serviços (JProxyQoS)

4.1 Considerações Iniciais

O crescimento da Internet nas últimas décadas tem sido objeto de inúmeras tentativas para melhorar a qualidade do serviço. Muitas dessas tentativas se basearam na infraestrutura e dispositivos de rede obtendo resultados interessantes, porém não suficientes, o principal motivo é que simplesmente o processamento do cabeçalho IP não é suficiente para gerenciar QoS com requisições de vários tipos de aplicações e em vários níveis de serviço ([Subramanian e Dutta, 2010](#)).

Enquanto o número de usuários da *World Wide Web* aumenta dia a dia, as requisições dos usuários que trafegam na rede são tratados em iguais condições, sem considerar tempo de processamento ou prioridade para requisições mais urgentes, esses dois fatores são alguns dos inumeráveis critérios que poderiam ser tomados em conta na diferenciação de serviço dos usuários ([Choi et al., 2007](#)).

Neste Capítulo se apresenta a implementação do protótipo de Servidor Web distribuído. Esta implementação está baseada nos conceitos divulgados por [Cardellini et al. \(2002\)](#) e no modelo proposto por [Teixeira et al. \(2005\)](#). Também neste capítulo são apresentados os componentes do protótipo, usando para isso uma agrupação em módulos: Classificador, Controle de Admissão e Distribuidor. E como ponto final se apresenta a modelagem do WebSwitch usando UML (*Unified Modeling Language*).

4.2 Arquitetura do Protótipo

Foi considerada uma arquitetura distribuída de servidores Web, nessa arquitetura todos os servidores têm informações replicadas (as mesmas páginas Web), todo isso com o objetivo de encaminhar as requisições para qualquer um dos servidores. Esta arquitetura pode ser organizada de várias maneiras, por exemplo, em (Cardellini et al., 2002) são apresentadas várias arquiteturas de configuração e também são utilizados vários termos como por exemplo **WebSwitch** ou **Front-End**, que é o elemento intermediário entre o cluster de servidores e os clientes Web, ele opera na camada de aplicação da pilha de protocolos do modelo TCP, O termo **Back-End**, designa o conjunto de servidores encarregados de servir as páginas Web.

Figura 4.1: Visão geral da arquitetura do protótipo.

O processo de comunicação é mostrado na Figura 4.1, onde se pode diferenciar o *WebSwitch* dos *Back-Ends*. Os *Back-Ends* estão conectados por meio de uma rede local de alta velocidade. Após atender as requisições o *Back-End* envia a resposta de volta ao *Front-End* que a repassa ao cliente. Para saber qual é o estado das máquinas que constituem o *Back-End*, foram instalados monitores em todas as máquinas do *Back-End*. Por meio do monitor, o *WebSwitch* consegue saber qual é o estado de cada máquina servidora. Na Figura 4.2, é apresentada a arquitetura do sistema proposto com um pouco mais de detalhes e as interações entre os componentes que o constituem. O protótipo foi construído usando a linguagem de programação **Java**¹, além de outros softwares e bibliotecas, entre eles estão:

- **Ganglia:** É um software para monitoramento distribuído escalável de sistemas computacionais de alto desempenho, tais como *clusters* e *grids* (Massie et al., 2003). Ele se baseia em uma escuta *multicast* e usa uma árvore de ligações ponto-a-ponto entre os nós. Como meio de divulgação das informações utiliza o formato XML;

¹<http://www.oracle.com/technetwork/java/index.html>

- **Httpperf:** É uma ferramenta para medir a disponibilidade de um servidores Web. Ele fornece um mecanismo flexível para a geração de várias cargas de trabalho para medir o desempenho do servidor ([Mosberger e Jin, 1998](#));
- **XPP:** É um *parser* XML ([Slominski, 2005](#)), é utilizado para o processamento das mensagens XML que vem do monitor. Esta biblioteca foi escolhida pela sua simplicidade e seu alto desempenho no processamento no processamento de arquivos XML, e seu trabalho é extrair do arquivo XML gerado pelo monitor *Ganglia* as informações que servem como base dos algoritmos implementados no protótipo, por exemplo: CPU, memória, entre outros;
- **HttpClient:** É uma biblioteca que permite flexível utilização do protocolo HTTP, ela é parte do conjunto de bibliotecas *Apache Jakarta Commons* ([Albing et al., 2005](#)), entre suas características estão:
 - Suporta os protocolos HTTP 1.0 e 1.1;
 - Gerenciamento automático de *cookies*;
 - Permite fazer *upload* de arquivos por partes usando o método POST;
 - Boas implementações dos métodos *resquest/response*;
 - Suporte para manipulação de redirecionamento de forma transparente.

Figura 4.2: Arquitetura do protótipo.

O *WebSwitch* foi dividido em três módulos: o Classificador, o Controle de Admissão, e o Distribuidor. Esses módulos vão ser explicados com mais detalhadamente nas próximas seções.

4.2.1 Classificador

O classificador é o primeiro elemento do sistema que trabalha com as requisições, ele é responsável por fazer um primeiro processamento das requisições, classificá-las, e com essas informações criar objetos contendo atributos como: prioridade, usuário dono ou, sessão, carga, entre outros. Na Figura 4.3 é apresentado um diagrama de blocos com as partes do classificador de requisições.

Figura 4.3: Módulo de classificação.

Para definir melhor o funcionamento do módulo Classificador, vamos ver uma descrição mais detalhada dos submódulos dos quais ele é composto:

- **Pool de conexões:** Este submódulo está encarregado da recepção das requisições. Ele fornece uma interface com um número definido de *threads*. Uma característica ressaltante da utilização do Pool de *threads* é a existência de uma quantidade limitada de recursos disponíveis, isto para não afetar às outras aplicações que estão executando ao mesmo tempo na máquina. Outra característica importante do Pool de *threads* é que se não tiver *threads* disponíveis, as requisições são armazenadas num *buffer* na espera do recurso se liberar;
- **Parser HTTP:** Está encarregado de tratar as requisições e analisá-las, com o objetivo de identificar os elementos característicos do protocolo HTTP. Na Seção 2.3.1 foram detalhadas amplamente essas características;
- **Algoritmos de classificação:** Nesse módulo encontra-se todos os filtros encarregados de caracterizar as requisições. Este submódulo do classificador foi desenvolvido com o objetivo de servir de base para a criação de filtros novos. No protótipo foram implementados alguns filtros especializados na detecção de:

- Detecção de MIME RFC 2045: dependendo do tipo de arquivo solicitado é atribuída uma prioridade para que depois outros algoritmos possam utilizar essa informação;
- Detecção de endereço IP: O principal objetivo é saber o endereço do cliente que fez a requisição, assim com essa informação o classificador consegue associar o endereço de origem com um perfil de usuário previamente estabelecido no sistema;
- Detecção da seção do usuário e o nível de carga: Para conseguir a identificação do usuário e a carga de cada requisição, é usado o mesmo critério que o filtro anterior acrescentando o valor da carga ou tempo de processamento da requisição, todas essas informações são extraídas da mensagem HTTP.

4.2.2 Controle de Admissão

Este módulo está encarregado de receber as requisições já classificadas e gerenciar a sua aceitação pelo servidor, levando em conta os algoritmos de fila que esteja executando e as informações da carga do sistema. Caso o sistema esteja sobrecarregado, uma requisição poderá ser rejeitada ou ter suas exigências de qualidade de serviço relaxadas ficando num *buffer* ou fila auxiliar.

O Controle de Admissão é essencial para evitar a sobrecarga do sistema, o que comprometeria alcançar os níveis de *QoS* pretendidos. A arquitetura do módulo está caracterizada pela ligação com um módulo que provê informações do estado do sistema, outro elemento importante é a *fila auxiliar* ou *buffer*, essa fila serve para armazenar as requisições que não podem ser atendidas no momento que chegaram ao módulo. A Figura 4.4 ilustra a arquitetura do módulo de controle de admissão com as subdivisões dentro dele.

Figura 4.4: Módulo de controle de admissão.

Seguidamente são descritas as principais partes do módulo:

- **Algoritmos de descarte:** Estão orientados a manter um nível aceitável de estabilidade do protótipo, mantendo um número máximo de requisições esperando

para serem atendidas, como consequência vai manter uma taxa de tempo de resposta razoável. A principal característica desses algoritmos é a monitoração da *fila auxiliar*, se ela estiver no limite da capacidade, segundo o algoritmo utilizado algumas requisições serão descartadas. Para os experimentos foi utilizado o algoritmo *droptail*;

- **Algoritmos de escalonamento:** Este grupo de algoritmos está encarregado de tirar as requisições da *fila auxiliar* segundo algum critério de seleção, estes algoritmos percorrem a fila auxiliar na procura de requisições mais urgentes, exemplos desses algoritmos são SJF e EBS que foram descritos na Seção 3.5.1, depois da seleção da requisição ela é encaminhada para o próximo módulo chamado **Distribuidor**;
- **Módulo de descarte:** Esse módulo está encarregado de gerar uma resposta para o cliente com códigos **5xx**;
- **Fila auxiliar:** Esse elemento está encarregado do armazenamento temporário das requisições, também é utilizado pelos algoritmos de escalonamento.

4.2.3 Distribuidor

Em trabalhos prévios como (Teixeira et al., 2005) ele não é muito estudado, focando mais nos estudos para os módulos de classificação e controle de admissão. No trabalho (Peixoto e Monaco, 2008) foi mencionado com mais notoriedade um módulo encarregado da distribuição de requisições para um cluster de servidores.

Este módulo é de grande importância para a interação com o *cluster*, ele é o responsável pela escolha do servidor que vai resolver o pedido para o cliente. Internamente o módulo executa uma *thread* encarregada de coletar informações do *cluster*. Para fazer as conexões *HTTP*, está sendo utilizada no protótipo a biblioteca do projeto Apache chamada *HttpClient*.

O distribuidor também está encarregado de definir os limites de atendimento do *cluster* com um número fixo de conexões simultâneas. Esse limite é atualizado em tempo real, assim com alguma conexão liberada, esse valor é atualizado novamente. Uma característica importante sobre o limite de conexões simultâneas é que se o limite for atingido, o módulo Controle de Admissão fica suspenso, simplesmente ele fica armazenando requisições na *fila auxiliar*, apenas uma conexão for liberada o módulo Controle de Admissão é ativado novamente.

Na Figura 4.5 são apresentadas as partes do módulo Distribuidor, e a seguir é feita uma explicação desses submódulos:

- **Algoritmos de distribuição:** Contém a lógica da distribuição das requisições nos servidores reais, para este protótipo foram implementados dois algoritmos: WRR

Figura 4.5: Módulo de controle de admissão.

(Weighted Round-Robin) e RR (Round-Robin), para mais informações destes algoritmos consulte a Seção 3.5;

- **Servidores:** Esta é uma abstração dos servidores ou *Back-Ends*, contém informações como: endereço IP, porta, nível de disponibilidade, entre outros parâmetros;
- **Implementação de conexões:** Esta parte foi criada para fazer um enlace entre o protótipo e alguma biblioteca de gerenciamento de *Sockets*. No protótipo está sendo usada a biblioteca *HttpClient*²;
- **Coletor de informações:** Este submódulo é executado para coletar informações do grupo de servidores ou *Back-End*, estas informações são atualizadas no submódulo “Servidores”, especificamente no nível de disponibilidade.

4.3 Modelagem do protótipo

Um modelo é uma representação em um determinado meio de algo no mesmo ou outro meio. O modelo capta os aspectos importantes da única coisa que está sendo modelado a partir de certo ponto de vista e simplifica ou omite o resto. Engenharia, arquitetura, e muitos outros campos criativos podem utilizar modelos no seu desenvolvimento (Rumbaugh et al., 1999).

Com o intuito de definir melhor a estrutura do protótipo e também facilitar a continuação do seu desenvolvimento optou-se por usar a UML (*Unified Modeling Language*). Algumas das vantagens da UML são as disseminações de padrões entre os vários desenvolvedores e a facilidade de comunicação. O objetivo é esclarecer como foi implementado o protótipo e auxiliar futuros desenvolvedores.

²<http://hc.apache.org/httpclient-3.x/>

4.3.1 Diagramas de Caso de Uso

Um diagrama de caso de uso modela a funcionalidade do sistema como percebido por usuários externos, chamados atores. Um caso de uso é uma unidade coerente de funcionalidade expressada como uma transação entre atores e o sistema. O propósito do diagrama de caso de uso é listar os atores e os casos de usos, mostrando quais atores participam em cada caso de uso (Rumbaugh et al., 1999). Nestes diagramas não são considerados comportamentos internos do sistema, simplesmente interações de atores com o sistema (Bezerra, 2002). Na Figura 4.6 se mostra o ator e os casos de uso mais representativos no sistema.

Figura 4.6: Diagrama de casos de uso *nível 1*.

O único ator identificado no sistema é o “*Cliente*”, que pode ser exemplificado por um navegador Web. Existe um caso de uso principal chamado “*Resolver pedido*”, esse caso de uso utiliza outros casos de uso para sua execução, entre eles: “*Classificar requisição*”, “*Procurar a requisição mais urgente*”, “*Parsear mensagem HTTP*”, “*Distribuir requisições no cluster*”, e “*Coletar informações dos servidores*”. Eles encontram-se relacionados com o caso de uso principal (“*Resolver pedido*”) mediante os estereótipos de relação “**include**” e “**extend**”, esses estereótipos definem uma relação de execução conjunta ou individual com respeito ao caso de uso principal respectivamente (Rumbaugh et al., 1999).

4.3.2 Diagramas de classes

Também chamada *vista estática*, pois não descreve um comportamento dependente do tempo no sistema (Rumbaugh et al., 1999). Os diagramas de classes são parte fundamental da maior parte das metodologias orientadas a objetos e são amplamente utilizados em projetos de software (Bezerra, 2002). Serão ressaltadas as principais características das classes mais importantes e das relações que existem entre as mesmas. Os diagramas de classes do protótipo foram organizados segundo os módulos estudados em seções anteriores.

Diagrama do Classificador

A principal classe da Figura 4.7 é *MainInterface*, que está encarregada de receber as requisições do cliente Web. Para isso é utiliza um objeto chamado **pool**, que representa uma agrupação *threads*. O objeto *pool* é uma instância da classe *ExecutorService* (Deitel e Deitel, 2006), esse objeto é inicializado com um método da classe estática *Executors*, ali também é definida a quantidade de *threads* com *Executors.newFixedThreadPool(numThreads)*.

Uma vez aceita a requisição o próximo passo é identificar as características da mesma, nesse processo é utilizada uma classe chamada *ThreadClassifier*, a principal tarefa desta classe é extrair a mensagem HTTP e criar um objeto que vai ser a representação da requisição no sistema, esse objeto é uma instância da classe *DRequisition* que implementa a interface *IElement*.

Figura 4.7: Diagrama do Classificador e Pool de threads.

Também na Figura 4.7 existe uma hierarquia de classes contendo os diferentes tipos de filtros que foram implementados no protótipo, esta arquitetura deixa as bases para a criação de diferentes classificadores para futuras pesquisas usando o protótipo.

Diagrama do Controle de Admissão

O diagrama do Controle de Admissão está caracterizado pela existência de dois blocos: um para as implementações de algoritmos de controles de admissão e outro para implementações de tipos de filas. Na figura 4.8 é apresentado o diagrama de classes do Controle de Admissão.

A maior parte do algoritmo de escalonamento em fila está escrito no método *getElement()* das classes que implementam a interface *IControlAdm*. Analisando, por exemplo a

Figura 4.8: Diagrama do Controle de Admissão.

classe *EBSControlAdmission*, encontra-se o algoritmo que procura a requisição com o menor deadline (Casagrande e Monaco, 2007). Outra característica ressaltante desse módulo são as implementações de tipos de filas auxiliares:

- **SimpleQueue:** Está baseada no objeto da classe *java.util.ArrayList*. As políticas FIFO e EBS, usam esse tipo de fila para o armazenamento de requisições;
- **MyPriorityQueue:** Basicamente é uma fila de prioridades, está baseada na classe *java.util.PriorityQueue* com a correspondente implementação da interface *java.util.Comparator*, a comparação de valores de cada requisição está baseada no valor que devolve o método *getPriority()* das classes que implementam a interface *IElement*³. A política que está usando este tipo de fila é a SJF.

Outro tipo de fila com características diferentes, pode ser implementada a partir da interface *IQueue*. Essa metodologia também é aplicada caso sejam necessários novos algoritmos de Controle de Admissão.

³As classes que implementam esta interface são as abstrações de uma requisição no sistema.

Diagrama do Distribuidor de Requisições

O objetivo deste modelo é permitir a criação de vários tipos de algoritmos de distribuição de requisições sobre várias unidades de processamento. Para isso existe a classe abstrata *AbstractScheduler*, nela encontram-se as estruturas básicas para a construção de mais algoritmos. Esta classe contém uma abstração dos servidores reais representados pela classe *RealServer*, que internamente tem a tarefa de fazer a requisição e enviar de volta a resposta para o cliente, para isso utiliza a classe *ResolutionRequest*, também tem uma referência à classe *RemoteServerState* para receber pela rede os estados dos *Back-Ends*.

Figura 4.9: Diagrama do Distribuidor.

- A classe *RemoteServerState*: É uma classe que herda da classe *Thread*. Está encarregada de trazer as informações do estado do *cluster* para o *WebSwitch*, extraindo por exemplo, a porcentagem de utilização da CPU da mensagem vinda de cada *Back-End*, especificamente do monitor Ganglia⁴, para isso utiliza a biblioteca XPP;
- A classe *ResolutionRequest*: É uma parte fundamental do protótipo, ela está encarregada da execução da requisição e do reenvio da resposta para o cliente. Esta classe também herda da classe *Thread* e também está associada a uma biblioteca de gerenciamento de *Sockets*, especificamente o *HttpClient*⁵.

⁴<http://ganglia.sourceforge.net/>

⁵<http://hc.apache.org/httpclient-3.x/>

Na Figura 4.9, mostra-se os detalhes dos relacionamentos de multiplicidade entre as classes. Conceitualmente a multiplicidade segundo Rumbaugh et al. (1999), indica quantos objetos de uma classe foram instanciados em outra classe, por exemplo, uma instância da classe *WeightedRR* pode ter de uma a várias instâncias da classe *RealServer*.

Diagrama do Monitor dos Usuários

As classes apresentadas neste diagrama estão diretamente relacionadas à política de escalonamento EBS da Seção 3.5. A principal tarefa deste conjunto de classes é monitorar as requisições que estão chegando ao servidor e manter um registro dos tempos de resposta das requisições, com esses valores a política EBS tenta gerenciar qualidade de serviço para os usuários.

Figura 4.10: Diagrama do monitor dos usuários.

Uma das principais características da modelagem deste grupo de classes foi a utilização do padrão MVC (Reenskaug, 2003). Em (Gamma et al., 1993) existe uma descrição mais aprofundada dos padrões que internamente o MVC utiliza para o seu funcionamento, por exemplo: o *Iterator* e o *Observer*.

Segundo Gamma et al. (1993), em geral o MVC está composto de: um modelo (*Model*), uma vista (*View*) e um controlador (*Controller*), na continuação é feita uma breve descrição dos submódulos do MVC projetado ao funcionamento do protótipo:

- **Modelo:** Guarda um conjunto de objetos da classe *EBSUser*. A classe *EBSUser* é uma abstração dos usuários contendo informações tais como:

- Código do usuário;
 - Classe do usuário;
 - Número de requisições;
 - Tempo médio de resposta;
 - Contrato (tempo máximo que as requisições podem demorar).
- **Controlador:** O controlador está encarregado da atualização das informações dos usuários (tempos de espera no atendimento). Pelas características do MVC, apenas as informações do modelo são mudadas pelo controlador, as vistas são automaticamente notificadas pelo modelo para que elas procurem as novas informações do mesmo modelo. Desta forma as vistas ficam com informações temporais dos estados dos usuários até uma requisição chegar do *Back-End* para o *WebSwitch*. É importante esse efeito porque se o limite da quantidade de conexões simultâneas é alcançado não deveriam ser mudadas as informações dos usuários nas vistas até algumas conexões forem liberadas;
 - **Vista:** Está encarregada de disponibilizar as informações dos estados dos usuários, no caso do protótipo, a vista está encarregada de prover os tempos médios e o valor do tempo de processamento o carga de que cada requisição que está esperando na fila.

4.4 Considerações Finais

O capítulo apresentou a modelagem e a arquitetura do protótipo, fazendo uma análise das principais partes, entre elas: o módulo **Classificador** encarregado da filtragem das requisições, o módulo de **Controle de Admissão** que também esta encarregado do escalonamento de requisições da fila auxiliar e o módulo **Distribuidor** que faz o gerenciamento dos recursos do *cluster*. O capítulo contém uma descrição dos principais componentes que foram utilizados na construção do protótipo.

No próximo capítulo serão apresentados os resultados obtidos com os experimentos realizados e uma discussão sobre os mesmos, com uma comparação entre os algoritmos implementados.

Resultados Experimentais

5.1 Considerações Iniciais

No Capítulo 4 foi apresentada a arquitetura proposta e a implementação do protótipo do servidor Web com diferenciação de serviços, abordando os seus principais componentes, tais como o Classificador, o Controle de Admissão e o Distribuidor de requisições.

Este Capítulo apresenta o planejamento do ambiente de experimentos e os resultados obtidos através dos experimentos realizados com o sistema. Os resultados dos experimentos visam verificar os algoritmos implementados no protótipo, para verificar se são capazes de fornecer QoS relativa e se a política EBS consegue fornecer QoS absoluta entre as classes de usuários.

Na avaliação dos algoritmos, serão apresentados os resultados referentes aos algoritmos do módulo Distribuidor, entre eles: WRR e RR, considerando-se os tempos de resposta fim a fim medidos nos clientes, isto é, a soma do tempo de resposta da rede e do *WebSwitch* e a quantidade de tarefas encaminhadas para cada *Back-End*. Em seguida serão apresentados os resultados dos algoritmos do módulo de Controle de Admissão, dando especial ênfase nos resultados da política EBS.

Para a política EBS considera-se a média dos tempos de respostas fim a fim, da mesma forma para os algoritmos do módulo distribuidor. Na parte da política EBS considera-se mais uma métrica chamada “satisfação” dos clientes. Essa métrica foi utilizada no trabalho de [Casagrande e Monaco \(2007\)](#), ela considera contratos ou limites de tempo de resposta no atendimento.

5.2 Avaliação de desempenho do protótipo

5.2.1 Configuração do Ambiente

Para uma correta avaliação de desempenho de um sistema computacional é imprescindível detalhar todos os elementos de *hardware* e *software* utilizados e verificar como tais elementos podem influenciar de forma positiva ou negativa no desempenho do sistema. A Figura 5.1 exibe o conjunto de elementos computacional utilizados. O ambiente de testes é composto de sete unidades de processamento com funcionalidades diferentes, sendo que: duas máquinas atuam como clientes, quatro máquinas atuam como servidores ou *Back-Ends* e uma máquina está funcionando como o *WebSwitch* da arquitetura.

Figura 5.1: Configuração do *hardware* das máquinas envolvidas no experimento.

- **Clientes:** As características dos clientes são iguais no *software* e no *hardware*, essas máquinas têm a missão de executar o Httperf com uma lista de requisições já estabelecida;
- **WebSwitch:** Possui melhores características de *hardware* comparadas às outras máquinas do sistema de testes. Esta máquina possui duas interfaces de rede, a idéia é interagir com a rede dos *Back-Ends* e a redes dos clientes;
- **Back-Ends:** A característica do grupo é que está composto de máquinas heterogêneas em *hardware*. O intuito é criar um *cluster* heterogêneo que possa ser utilizado nos experimentos. Na Figura 5.1, é possível observar a diferença que existe entre as máquinas S1 e a máquina S2, na quantidade de memória e na capacidade das CPUs. Na máquina S3, foi feita uma mudança na frequência do *Clock* da placa mãe para baixar a capacidade da CPU.

Além dos elementos de *hardware* descritos na Figura 5.1, a Tabela 5.1 descreve os elementos de *software* usados no ambiente de testes. Os elementos assinalados com um (*) foram desenvolvidos pelo autor e o restante foram adaptados ou simplesmente usados sem nenhuma alteração. O sistema operacional utilizado nas máquinas foi Ubuntu 9.10¹ de 32 bits para os clientes e 64 bits para os *Back-Ends* e para o *WebSwitch*.

Tabela 5.1: Elementos de *software* envolvidos.

Nome	Versão	Descrição	Utilização
JProxyQoS *	1.0	Protótipo de servidor Web distribuído.	<i>WebSwitch</i> .
Java 2 Standard Edition	1.6.0_20	Ferramenta de desenvolvimento para a plataforma Java.	<i>WebSwitch</i> e nos servidores do <i>cluster</i> .
Ganglia	3.1.7	Sistema de monitoração distribuído e escalável.	Nas sete máquinas utilizadas nos experimentos.
Httpperf	0.9.0	Ferramenta para mensurar o desempenho de servidores Web.	Nos clientes <i>C1</i> e <i>C2</i> .
Apache Tomcat	7.0.0	<i>Container</i> de <i>Servlets</i> .	Nos servidores do <i>cluster</i> <i>S1</i> , <i>S2</i> , <i>S3</i> , e <i>S4</i> .
CPUBoundApp*	0.5	Aplicação orientada ao consumo de CPU.	Nos servidores do <i>cluster</i> <i>S1</i> , <i>S2</i> , <i>S3</i> , e <i>S4</i> .
Perl	5.10.1	Linguagem de programação interpretada.	Nas sete máquinas utilizadas nos experimentos.
ExpManager *	0.5	Aplicação encarregada da execução dos experimentos	Nas sete máquinas utilizadas nos experimentos.
PostgreSQL	8.4.4	Sistema Gerenciador de Bancos de Dados.	Utilizado para guardar os resultados dos experimentos.

5.2.2 Planejamento dos Experimentos

Esta etapa é muito importante na avaliação de sistemas computacionais, a tarefa fundamental é definir quais partes analisar e como que serão avaliadas. Os *fatores* são as características do sistema que afetam as variáveis de resposta que se pretende avaliar. Os valores que um determinado fator pode assumir são denominados *níveis*.

Em complemento aos termos presentes em uma avaliação de desempenho, há vários tipos de técnicas utilizadas para determinar a etapa de planejamento de experimentos (Jain,

¹<http://www.ubuntu.com>

1991). As mais utilizadas são o planejamento *simples*, planejamento *fatorial completo* e o planejamento *fatorial parcial*. No planejamento simples é considerada uma configuração inicial, depois fixar todos os valores e variar um fator por vez, e seguidamente verificar se esse fator está afetando o desempenho. É fácil de ser implementado, além de ser muito utilizado. No entanto, não permite verificar a interação entre os fatores. No planejamento fatorial completo são utilizadas todas as combinações considerando todos os fatores e níveis. Com isso, todos os fatores são avaliados, o efeito de cada fator é determinado, e as interações entre fatores podem ser verificadas. O principal problema dessa abordagem é o custo da avaliação de desempenho, no qual é gerada uma grande quantidade de experimentos. Para reduzir a quantidade de experimentos gerados, pode-se reduzir o número de níveis para cada fator, reduzir o número de fatores, ou então utilizar a abordagem de fatorial parcial. A abordagem de fatorial parcial consiste em utilizar apenas uma fração do planejamento fatorial completo. Nessa abordagem considera-se apenas parte de todos os experimentos.

O planejamento de experimentos usado neste projeto foi o **fatorial completo**. Para todos os experimentos foram realizadas 30 execuções para determinar uma média, o desvio padrão e o intervalo de confiança de 95% de acordo com a tabela *T-Student*. O número de execuções foi definido considerando-se a necessidade de obter diferença significativa dentre os intervalos de confiança dos experimentos que implica em comparação dos resultados. Nas Tabelas 5.2 e 5.3, são apresentados os fatores e seus respectivos níveis. Os experimentos foram agrupados segundo os módulos onde funcionam: Controle de Admissão e Distribuidor.

Tabela 5.2: Experimentos dos algoritmos do módulo Distribuidor.

i,j	Algoritmos de distribuição	
Carga externa	RR	WRR
Sim	$X_{0,0}$	$X_{0,1}$
Não	$X_{1,0}$	$X_{1,1}$

Tabela 5.3: Experimentos dos algoritmos do módulo Controle de Admissão.

i,j	Algoritmos de controle de admissão	
Proporções	SJF	EBS
50% A - 50% B	$Y_{1,0}$	$Y_{1,1}$
10% A - 90% B	$Y_{0,0}$	$Y_{0,1}$
90% A - 10% B	$Y_{2,0}$	$Y_{2,1}$

5.2.3 Carga de trabalho sintética

Para validar o protótipo desenvolvido neste projeto, foi criada uma lista de requisições com vários níveis de carga (tempo de processamento), esses níveis de carga estão baseados em uma distribuição exponencial (especificamente uma distribuição de Poisson (Paxson e Floyd, 1995)).

Como parte complementar, para os *Back-Ends* foi desenvolvida uma aplicação Web dinâmica do tipo *servlet* (Davidson e Ahmed, 1998), essa aplicação está encarregada de receber as requisições e assim gerar os níveis de CPU esperados.

Para conseguir gerar uma carga de trabalho na máquina servidora usando a aplicação Web mencionada no parágrafo anterior, utilizou-se uma característica da linguagem de programação Java: Um objeto do tipo *String* não pode ser alterado após sua criação. O que pode ser feito com esse objeto é ser excluído pelo coletor de lixo da máquina virtual Java, se não houver outras variáveis fazendo referências a ele, mas não pode ser alterado. Por este motivo, os objetos do tipo *String* são chamados **imutáveis** (Kak, 2003).

É por esta razão que dada uma concatenação sobre um objeto do tipo *String* o que acontece é a criação de um objeto novo, depois o conteúdo do objeto antigo é copiado para o objeto novo mais os caracteres novos, esse processo é repetido várias vezes na geração de carga na CPU.

Para fazer a concatenação de texto de uma forma eficiente; Java disponibiliza a classe *StringBuffer*, os objetos instanciados a partir dessa classe permitem a concatenação sem problema de sobrecarga no sistema.

5.3 Validação do modelo Web utilizado

Segundo Pezzè e Young (2008), avaliar o grau em que um sistema de *software* realmente satisfaz seus requisitos, no sentido de atender às necessidades reais do usuário é chamado de **validação**. Assim esse vai ser o objetivo desta seção, saber como o protótipo satisfaz ao cliente e em que medida ele responde às requisições num determinado ambiente de *software* e *hardware*.

O primeiro aspecto a ser considerado é a validação do modelo de experimentos. Essa validação é realizada por meio de testes que seguem as diretrizes sugeridas por Jain (1991). No segundo aspecto verificou-se quanto de gargalo gera o *JProxyQoS* durante a troca de informações entre os *Back-Ends* e os clientes. Também serão apresentados resultados associados à aplicação Web e as taxas de requisições que vão ser utilizadas nos experimentos.

O primeiro passo é saber como a aplicação Web comporta-se no ambiente dos experimentos. Para isso foram submetidas várias taxas de requisições na máquina com maiores recursos de *hardware* (S2). Assim, na Figura 5.2 mostra-se como foi o comportamento da

máquina para várias taxas de requisições. Para uma taxa de 1/s forma submetidas 100 requisições, como resultado, o impacto na máquina não é muito significativo, Resultado deste primeiro experimento foi um tempo de resposta médio de **62,92ms**.

Figura 5.2: Comportamento dos níveis de requisições na aplicação Web.

A taxa das requisições é incrementada até conseguir estressar o servidor, o que ocorre com uma taxa de 16/s. Os testes foram executados até uma taxa de 100/s, nesse nível o servidor começou a ter problemas pelo excesso de carga, como nota-se na Figura 5.2, com quedas na porcentagem de utilização da CPU para taxas de 80/s e 100/s, basicamente pode se observar um comportamento instável do sistema. A taxa de requisições que conseguiu sobrecarregar ao servidor sem produzir paradas nele foi de 60/s. Além disso, esta configuração não tem intervalos de confiança muito altos nem quedas como acontece com 80/s e 100/s. Tendo em conta todos esses fatores ela foi elegida para os futuros experimentos. Para a taxa de 1/s foram utilizadas 100 requisições, a partir de 6/s são usadas 1000 requisições. Estes comportamentos se repetiram por 5 execuções, mas simplesmente foi apresentada uma delas.

O experimento com taxa de 60/s foi repetido novamente para 1000 requisições geradas em uma taxa de 60/s. Obtendo-se um tempo médio de resposta de **29624ms**. O resultado desse experimento é apresentado na Figura 5.4.

Por último são apresentadas informações da porcentagem de utilização da CPU das máquinas envolvidas no primeiro experimento, a máquina *S2* ficou com uma porcentagem de 84%, nota-se também que as máquinas *C1* e *C2* têm uma baixa utilização da CPU, isso acontece porque essas máquinas simplesmente estão executando os cliente Web (*Httpperf*).

Figura 5.3: Tempos de resposta dos diferentes níveis de requisições.

Figura 5.4: Tempo de resposta para o Tomcat com 60 requisições/s.

5.4 Análise dos Resultados dos Algoritmos

Nesta seção são apresentados alguns resultados obtidos dos experimentos usando o protótipo *JProxyQoS*. Foram considerados algoritmos de diferenciação de serviços para efetuar o escalonamento/encaminhamento de requisições.

Os experimentos foram divididos em dois grupos, o critério da agrupação é feito segundo o módulo onde estão funcionando no protótipo. Assim, temos algoritmos do módulo Distribuidor e os algoritmos do módulo de Controle de Admissão.

5.4.1 Algoritmos do Módulo Distribuidor

Como já foi dito em seções prévias, este módulo está encarregado da distribuição de requisições nos *Back-Ends*. Nesta avaliação busca-se saber quanto influenciam os

Figura 5.5: Porcentagem de utilização da CPU nas máquinas envolvidas.

algoritmos no tempo de resposta para os clientes e como eles aproveitam os recursos (*Back-Ends*), fazendo contas de quantas requisições são encaminhadas a cada *Back-End*.

Foram mensurados:

- Tempos de resposta nos clientes;
- Quantidade de requisições que são encaminhadas para cada servidor e;
- Quantidade de utilização da CPU nas máquinas do *Back-End*.

A avaliação do comportamento dos algoritmos de distribuição de requisições tenta descobrir como as quantidades de requisições enviadas para cada *Back-End* afeta no desempenho do sistema inteiro.

5.4.1.1 RR e WRR

Nesta seção são apresentados os resultados obtidos da execução dos experimentos, monitorando quantidade de requisições e porcentagem de utilização da CPU.

No caso do WRR, há uma forte dependência da métrica utilizada na monitoração de cada máquina do *Back-End*, o algoritmo leva em conta esse valor para mudar a configuração interna do *WebSwitch*, assim consegue obter um melhor aproveitamento dos recursos, mudando as quantidades de requisições enviadas para cada *Back-End* segundo as informações da métrica. A Figura 5.6 mostra os resultados para o clássico RR e para o WRR, este gráfico apresenta informações sobre: a porcentagem de utilização da CPU e também a porcentagem de requisições que foram enviadas a cada servidor. Nota-se que as porcentagens de requisições enviadas usando o algoritmo RR é de 25% em todos os servidores e as porcentagens do WRR ficaram com níveis diferentes.

Figura 5.6: Relação de % da CPU e % de reqs enviadas.

Analisando o comportamento RR, pode-se ver como os níveis de carga variáveis agem no sistema gerando níveis diferentes de porcentagem de utilização da CPU, isso em máquinas com características similares de *hardware* (*S2* e *S4*), para as outras máquinas este fator é influenciado pelas diferenças nas configurações de *hardware* de cada servidor, principalmente entre as máquinas *S1* (a mais fraca) e *S2* (com maiores recursos). As máquinas *S3* e *S4* são quase idênticas no *hardware*, porém houve uma mudança no relógio (*Clock*) da placa mãe da máquina *S3* para 133MHz quando o valor padrão é 266MHz . Isto com a intenção de ter heterogeneidade de recursos nos servidores.

O algoritmo WRR foi melhor no gerenciamento dos recursos, ele baixou o número de requisições para o servidor *S3*, pois dada uma quantidade inicial de requisições a porcentagem de utilização sobe muito e as requisições que não foram mais encaminhadas para o servidor *S3* são enviadas para os outros servidores.

5.4.1.2 RR e WRR com carga externa

Este experimento foi executado para identificar como é o comportamento do sistema se nos *Back-Ends* existir uma carga externa além da gerada pela aplicação Web. Para tais fins foi utilizada a compilação do Kernel Linux² na máquina *S2*, é bem conhecido que este tipo de processo não é simplesmente do tipo CPU *Bound*, para diminuir o impacto essa compilação foi feita na máquina com maiores recursos de *hardware* (*S2*).

Os experimentos com a aplicação Web e a compilação do *Kernel Linux* foram executados ao mesmo tempo na máquina *S2*. Dada a perspectiva da quantidade de requisições enviadas e o impacto na utilização da CPU o sistema usando o algoritmo RR não sofreu muitas mudanças significativas nos servidores do *Back-End*, com a exceção do servidor *S2*, na Figura 5.7 a porcentagem de utilização da CPU usando RR subiu consideravelmente.

Nos experimentos onde foi utilizado o algoritmo WRR, a máquina *S2*, a quantidade de requisições enviadas para essa máquina foi diminuída com relação ao caso anterior

²<http://kernel.org/>

Figura 5.7: Relação de % da CPU e % de reqs enviadas, com carga externa.

onde não tinha carga adicional. No caso anterior foi registrada uma porcentagem média de requisições enviadas de 27%, e quando foi agregada a carga externa ficou em 17%, dando uma diferença aproximadamente de 10% de diminuição. Esse 10% de requisições foram encaminhadas para os outros servidores.

5.4.1.3 Tempo de resposta RR e WRR

Nas duas seções anteriores foi analisado o desempenho do sistema do lado do *WebSwitch* e os *Back-Ends* com a porcentagem de requisições que foram enviadas para cada servidor e qual foi a porcentagem de utilização da CPU em cada máquina. Nesta seção é feita a avaliação dos resultados da perspectiva dos clientes usando os tempos de resposta medidos nas máquinas clientes *C1* e *C2*.

Esses resultados são apresentados na Figura 5.8, foi utilizada uma notação, (*RR+Ext* e *WRR+Ext*) que significa RR e WRR com carga externa respectivamente. Uma análise preliminar deixa claro como a utilização do algoritmo WRR proporciona tempos de resposta baixos em comparação com os resultados do algoritmo RR.

Figura 5.8: Tempos de resposta para dos algoritmos RR e WRR com e sem carga externa.

Uma análise mais discriminada mostra o quão baixos são os tempos médios de resposta para o WRR comparando-o com o RR, isto se repete para os dois ambientes: com e sem carga externa. Quando não existe sobrecarga os tempos de resposta do WRR ficaram em uma meia de $3214ms$, enquanto que o RR ficou em $3772ms$, a diferença foi de aproximadamente $558ms$. Para o caso com a utilização de carga externa a diferença dos tempos médios é de $2131ms$. Outra grande diferença notada é como o WRR trata de forma igualitária aos dois clientes. Com esses resultados pode se afirmar que o WRR ficou mais estável do que o RR quando as requisições têm níveis de carga variável.

5.4.2 Algoritmos do módulo Controle de Admissão

Nessa seção são apresentados os algoritmos implementados no módulo de Controle de Admissão, eles estão fortemente relacionados com a utilização de uma **fila auxiliar** para armazenar e escolher a requisição mais urgente.

Casagrande e Monaco (2007), avaliaram a política de escalonamento EBS em diferentes cenários. Para explicar os resultados dos experimentos que realizaram, eles usaram os termos: *satisfação* e *variação contratual*. Esses termos serão levados em conta para falar dos resultados obtidos. Isto com a intenção de tentar constatar os resultados obtidos neste projeto com os resultados do trabalho original.

No trabalho de (Casagrande e Monaco, 2007), foram utilizados vários níveis de variações contratuais, mas neste trabalho serão utilizados somente três níveis: 20%, 30% e 40%.

5.4.2.1 EBS 50% A e 50% B

Neste cenário a quantidades de requisições de ambas as classes são as mesmas, 50% para cada uma. Assim como foi feito em (Casagrande e Monaco, 2007), é preciso definir um valor de referência para estabelecer os contratos. Para isso foi executado primeiro o experimento utilizando o algoritmo FIFO, o valor referencial obtido foi $3883ms$. Outro ponto de comparação além do algoritmo FIFO foi o algoritmo SJF.

Na Figura 5.9, a política SJF deveria ficar com um tempo médio de resposta menor do que o tempo médio da FIFO, isto não está aconteceu porque o *cluster* está subutilizado, a única mudança foram os intervalos de confiança menores, resultado da melhora na latência usando SJF.

Assim foram definidos os contratos para as requisições de classe A e classe B para três cenários de variação contratual de 20%, 30% e 40% como se observa na Figura 5.9. Pode-se perceber que a satisfação dos clientes foi conseguida numa porcentagem alta, somente no caso da variação contratual de 40% teve uma pequena queda no $C1$ (cliente de classe A). Na Tabela 5.4, são apresentadas as porcentagens de satisfação dos tipos de usuários com as diferentes variações contratuais. Para fazer uma comparação com o trabalho que

Figura 5.9: Tempos de resposta – Cenário 50% classe A e 50% classe B.

originou a política EBS, foi feita a mesma mudança, com 3% a mais acima do contrato estabelecido.

Tabela 5.4: Porcentagens de satisfação dos contratos, 50% classe A e 50% classe B.

VC	Clientes	Satisfação
20%	C1	96.6%
	C2	53.3%
30%	C1	86.6%
	C2	76.6%
40%	C1	3.3%
	C2	93.3%

O único caso que o nível de satisfação do usuário foi baixo no ambiente de teste foi com 40% de variação contratual, ficando em 3.3%. No trabalho original feito usando um simulador, os níveis de satisfação são de 100% para as duas classes em condições similares de proporção de requisições. As variações contratuais em relação ao valor médio gerado ao executar o algoritmo FIFO. A diferença com o modelo de testes feito no simulador é a quantidade de requisições, para o caso deste projeto foi uma limitante longo período de tempo que seria necessário para executar cada experimento em condições reais.

5.4.2.2 EBS 10% A e 90% B

Este cenário está caracterizado pela baixa quantidade de requisições de classe A e uma grande quantidade de requisições da classe B. Analisando o primeiro caso de variação contratual pode-se ver que os tempos de resposta do cliente C1 ficou abaixo do contrato e como um intervalo de confiança também baixo. Fazendo uma comparação com o caso anterior, a quantidade de requisições da classe A é menor causando um menor impacto no *WebSwitch* no momento de dar preferência às requisições da classe A. As requisições de classe A tem uma maior variação de tempos de resposta, devido à baixa proporção de

requisições com relação à classe B, e como evidencia estão os intervalos de confiança de cada classe. Esse fenômeno não é mais visto com a utilização do algoritmo EBS porque ele não faz caso do tempo de chegada, se não pela urgência de cada classe no sistema.

Figura 5.10: Tempos de resposta – Cenário 10% classe A e 90% classe B.

Da Figura 5.10, nota-se que existe uma relação direta com a variação contratual. Se o contrato da classe A for muito exigente isso faz com que o sistema não consiga cumprir o contrato. O pior dos casos foi com uma variação contratual de 40% na classe A, onde a média de tempo de resposta ficou acima do contrato estabelecido.

Tabela 5.5: Porcentagens de satisfação dos contratos, 10% classe A e 90% classe B.

VC	Clientes	Satisfação
20%	C1	96.6%
	C2	100.0%
30%	C1	80.0%
	C2	100.0%
40%	C1	40.0%
	C2	100.0%

Fazendo uma análise mais específica dos níveis de satisfação do usuário de cada classe, deixa-se ver que esses níveis são relativamente altos, fazendo uma comparação com o anterior ambiente (50%–50%). Na Tabela 5.5, são apresentados esses valores. No primeiro cenário com variação contratual de 20% pode-se dizer que a satisfação dos usuários foi quase do 100% e assim essa satisfação foi baixando com variações contratuais maiores. Segundo os resultados feitos no simulador, este cenário deveria ter níveis de satisfação de 100%, mas nesta implementação não se obteve esses resultados. Algumas razões poderiam ser: os níveis de carga, quantidade de requisições e o modo pre-entivo no atendimento de requisições do protótipo. No simulador a o modo de atendimento foi não pre-entivo.

5.4.2.3 SJF e EBS 90% A e 10% B

Neste experimento apresenta-se o caso inverso do cenário anterior, agora o escalonador está num ambiente com uma maior quantidade de requisições de classes A (90%) e conjuntamente aplicando as variações contratuais de 20%, 30% e 40%, criarão um ambiente hostil para o escalonador conseguir não ultrapassar os contratos estabelecidos a partir da política FIFO.

Figura 5.11: Tempos de resposta – Cenário 90% classe A e 10% classe B.

Analisando o primeiro cenário com variação contratual de 20%, o escalonador não conseguiu satisfazer o contrato da classe A ficando com um 0%. Na Figura 5.11, as requisições da classe A ficaram com tempos acima do tempo estabelecido no contrato, somente as requisições da classe B são satisfeita quase ao 100% com variação contratual de 20% e 30%, com 40% o escalonador teve uma queda geral, essas informações são apresentadas na Tabela 5.6.

Tabela 5.6: Porcentagens de satisfação dos contratos, 90% classe A e 10% classe B.

VC	Clientes	Satisfação
20%	C1	0.0%
	C2	100.0%
30%	C1	0.0%
	C2	100.0%
40%	C1	0.0%
	C2	53.3%

Como foi dito anteriormente, este é um ambiente hostil para o escalonador pela grande quantidade de requisições que têm que ser servidas com tempos de resposta baixos. Na Tabela 5.6 nota-se que com uma variação contratual de 40% sendo esta mais exigente

do que 20% ou 30% o algoritmo EBS já não consegue cumprir pelo menos o contrato da classe B ficando em 53.3% de satisfação.

5.5 Considerações Finais

Este capítulo apresentou os resultados da validação e demonstração de como avaliar o desempenho do protótipo JProxyQoS. Foram discutidos inicialmente a configuração do ambiente de validação e testes envolvendo configurações de *software* e *hardware*, um planejamento de experimentos e os algoritmos de diferenciação de serviços.

Foram apresentados os resultados dos experimentos usando os algoritmos estudados na Seção 3.5, foi feito também uma análise desses resultados.

O próximo capítulo tem como objetivo apresentar de forma detalhada as conclusões obtidas com o desenvolvimento deste projeto de mestrado, no que diz respeito aos resultados alcançados e de novos projetos futuros.

Conclusão

6.1 Considerações Finais

A finalidade deste trabalho foi implementar um protótipo de servidor Web com diferenciação de serviços tendo em vista a avaliação de seu desempenho. A motivação surgiu a partir do crescimento da Web nos últimos anos, tendo que gerenciar aplicações cada vez mais complexas as quais exigem mais processamento para os servidores.

Esses servidores, quase em sua totalidade, atendem às requisições de acordo com o algoritmo FIFO, ou seja, a primeira a chegar é a primeira a ser atendida. Esse tipo de comportamento por parte dos servidores nem sempre é o desejado, pois cada vez mais surgem aplicações e clientes com expectativas diferentes, necessitando então de atendimentos diferenciados.

Durante os experimentos, a carga de trabalho estava composta por requisições dinâmicas que exigem uso intenso da CPU. Os níveis estão baseados em uma função exponencial.

Várias políticas foram implementadas e divididas em dois grupos: distribuição no *cluster* e escalonamento na fila auxiliar. No primeiro grupo estão o *Round-Robin* e o *Weighted Round-Robin*. A monitoração feita no *cluster* foi um fator importante no momento de realizar o balanceamento da carga em cada nó do *cluster*. O *Weighted Round-Robin* apresentou resultados melhores do que o *Round-Robin* simples. Além disso, o *Weighted Round-Robin* obteve tempos de respostas mais estáveis para os dois tipos de usuários, é possível notar isso analisando os resultados depois de submeter uma carga externa no *cluster*.

No segundo grupo foram implementados os algoritmos FIFO, SJF e EBS. O último

algoritmo gerou vários outros trabalhos relacionados desenvolvidos pelo grupo de Sistemas Distribuídos e Programação Concorrente. Segundo [Casagrande e Monaco \(2007\)](#), eles propuseram a política EBS que fornecer QoS absoluta usando classes de usuários, com o mesmo simulador forma implementados vários outros trabalhos de pesquisa como: ([Nery e Monaco, 2008](#)), ([Tott e Monaco, 2008](#)), ([Saito e Monaco, 2010](#)) entre outros.

Da análise feita por [Casagrande e Monaco \(2007\)](#) da política EBS, eles concluíram que a política EBS estava fortemente ligada às pequenas variações contratuais para conseguir um melhor atendimento, um exemplo disso é o experimento onde as requisições foram distribuídas em 10% para a classe A e 90% para a classe B. Na mudança da variação contratual de 20% para 40% originou o não cumprimento dos contratos para a classe A, mas no estudo feito no simulador em condições similares o cumprimento foi de 100%.

Devido a que não foi possível reproduzir o ambiente da simulação onde foi desenvolvida a política EBS, não se pode afirmar que ela não fornece QoS absoluta, nos ambientes onde não conseguiu 100% de satisfação dos usuários.

6.2 Principais Resultados e Contribuições

Os principais resultados deste trabalho estão relacionados a seguir, destacando-se as contribuições feitas ao modelo de Servidor Web com Diferenciação de Serviços e, consequentemente, relevantes para a área:

1. **Revisão Bibliográfica:** Foi feita uma revisão crítica da bibliografia existente, afim de contextualizar o presente trabalho. A revisão foi feita em torno de informações sobre o fornecimento de serviços diferenciados nos servidores web. Revisou-se também o modelo de Servidor Web com Diferenciação de Serviços (SWDS), no qual o presente trabalho teve sua origem. Além de maneiras de organização dos servidores e do cluster de servidores web. Esta revisão contribui no sentido de facilitar futuros trabalhos e a iniciação de futuros pesquisadores.
2. **Protótipo do Servidor Web com Diferenciação de Serviços (ProxyQoS):** Foi implementado um protótipo utilizando-se uma arquitetura de cluster, todo o protótipo foi desenvolvido utilizando a linguagem de programação Java, também o protótipo foi construído com a idéia de ser facilmente mudado e utilizado.
3. **Avaliação do protótipo através de submissão de carga sintética:** Foi utilizado o *Httpperf* como cliente, ele estava encarregado de fazer as requisições com níveis de carga variável.
4. **Análise dos resultados e comparação dos mesmos com e sem a utilização dos algoritmos propostos:** Vários experimentos foram realizados afim de analisar a eficiência dos algoritmos propostos e a influência dos componentes do sistema

(rede, Web Switch, servidor, etc.) nos resultados. Além disso, uma comparação entre os resultados obtidos com a utilização dos *Weighted Round-Robin* e do algoritmo *Round-Robin* simples. Essa análise permitiu apontar vantagens e desvantagens dos algoritmos avaliados em uma arquitetura de cluster de servidores Web localmente distribuídos usando cargas de trabalho variáveis.

6.3 Trabalhos Futuros

Uma das principais contribuições deste trabalho é disponibilizar uma plataforma para futuras implementações de algoritmos que poderão ser desenvolvidos e outros já existentes. Baseados neste modelo poderão ser implementados e avaliados no protótipo, novos mecanismos de diferenciação de serviços que poderão até mesmo ser incorporados ao mesmo. A seguir, são comentadas algumas sugestões para trabalhos futuros:

- Fazer novos experimentos considerando-se uma carga de trabalho dinâmica usando, por exemplo, modelos Markovianos (Mi et al., 2010) para a geração da carga com a idéia de gerar carga o mais perto da realidade do que usando funções exponenciais clássicas. Para tanto, teria que haver um estudo aprofundado sobre as características deste tipo de carga;
- Fazer novos experimentos com a utilização dos algoritmos de descarte e renegociação de requisições;
- Implementar no Web Switch um mecanismo de *cache* a fim de quantificar quanto é a melhora;
- Implementar mecanismos no *WebSwitch* capazes de prever a carga de trabalho e auxiliar na tomada de decisões sobre escalonamento e descarte de requisições antes que os servidores estejam sobrecarregados;
- Implementar e realizar novos experimentos considerando um *WebSwitch One-Way* para observar se há ou não ganho de desempenho, para isso faz necessário mudanças no Kernel do sistema operacional tanto da máquina que fará o papel de *WebSwitch*, quanto das máquinas *Back-Ends*.

Bibliografia

- ABDELZAHER, T. F.; SHIN, K. G.; BHATTI, N. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, v. 13, n. 1, p. 80–96, 2002.
- ALBING, C.; SCHWARZ, M.; BLANCHETTE, J.; SUMMERFIELD, M.; CAMERON, J.; GORMAN, M.; GUTMANS, A.; BAKKEN, S.; RETHANS, D.; HERTEL, C.; HOWLETT, T.; MASSA, A.; MCFARLANE, N.; REHMAN, R. U.; PAUL, C.; TERPSTRA, J. H.; VERNOOIJ, J. R.; TERPSTRA, J. H.; IVERSON, W. Apache jakarta commons reusable java components. 2005.
- ALMEIDA, J.; DABU, M.; CAO, P. Providing differentiated levels of service in web content hosting. In: *First Workshop on Internet Server Performance - WISP (in conjunction with ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems)*, p. 91–102, 1998.
- ANDREOLINI, M.; CASALICCHIO, E.; COLAJANNI, M.; MAMBELLI, M. A cluster-based web system providing differentiated and guaranteed services. *Cluster Computing*, v. 7, n. 1, p. 7–19, 2004.
- BACH, M. J. *Design of the unix operating system*. 1 ed. Upper Saddle River, New Jersey 07458: Prentice Hall PTR, 1986.
- BERNERS-LEE, T.; CAILLIAU, R.; GROFF, J.-F.; POLLERMANN, B. World-wide web. *Electronic Networking: Research, Applications and Policy*, v. 1, n. 2, p. 74–82, 1992.
- BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0. Status: INFORMATIONAL., 1996.
- BEZERRA, E. *Princípios de análise e projeto de sistemas com uml*. 1 ed. Rio de Janeiro: Campus, 2002.

- BHATTI, N.; BOUCH, A.; KUCHINSKY, A. Integrating user-perceived quality into web server design. In: *Proceedings of the 9th international World Wide Web conference on Computer networks: the international journal of computer and telecommunications networking*, Amsterdam, The Netherlands: North-Holland Publishing Co., p. 1–16, 2000 (1-6, v.33).
- BLACK, D. L. Scheduling support for concurrency and parallelism in the mach operating system. *IEEE Computer*, v. 23, p. 35–43, 1990.
- BLAKE, S.; BLACK, D.; CARLSON, M.; DAVIES, E.; WANG, Z.; WEISS, W. RFC 2475: An architecture for differentiated services. *Internet RFC, Internet Engineering Task Force - IETF*, 1998.
- BRADEN, R.; CLARK, D.; SHENKER, S. RFC 1633: Integrated services in the internet architecture: an overview. *Internet RFC, Internet Engineering Task Force - IETF*, 1994.
- CALANDRINO, J. M.; BAUMBERGER, D.; LI, T.; HAHN, S.; ANDERSON, J. H. Soft real-time scheduling on performance asymmetric multicore platforms. *13th IEEE Real Time and Embedded Technology and Applications Symposium - RTAS*, v. 0, p. 101–112, 2007.
- CALLAWAY, D. R. *Inside servlets: Server-side programming for the java(tm) platform*. 2 ed. Reading, Massachusetts 01867: Addison-Wesley Professional, 2001.
- CARDELLINI, V.; CASALICCHIO, E.; COLAJANNI, M.; YU, P. S. The state of the art in locally distributed web-server systems. *ACM Comput. Surv.*, v. 34, n. 2, p. 263–311, 2002.
- CASAGRANDE, L. S.; MONACO, F. J. *Política de escalonamento de tempo real baseada em exigência para provisão de qos absoluto em serviços web*. Dissertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação., São Carlos, Brasil, 2007.
- CHEN, X.; HEIDEMANN, J. Preferential treatment for short flows to reduce web latency. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, v. 41, n. 6, p. 779–794, 2003.
- CHENG, A. M. K. *Real-time systems: Scheduling, analysis, and verification*. Hoboken, New Jersey: Wiley - Interscience, 2002.
- CHOI, S. W.; HER, J. S.; KIM, S. D. Qos metrics for evaluating services from the perspective of service providers. In: *ICEBE '07: Proceedings of the IEEE Internati-*

- onal Conference on e-Business Engineering*, Washington, DC, USA: IEEE Computer Society, p. 622–625, 2007.
- COMER, D. E. *Internetworking with tcp/ip*, v. 1. 4 ed. Upper Saddle River, New Jersey 07458: Prentice Hall, 750 p., 2000.
- DAVIDSON, J. D.; AHMED, S. Java servlet api specification: Version 2.1a. In: *Technical Document of Sun Microsystems*, 1998.
- DEITEL, P. J.; DEITEL, H. M. *Java how to program*. 7 ed. Upper Saddle River, New Jersey 07458: Prentice Hall, 2006.
- DEVI, U.; ANDERSON, J. Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. *20th International Parallel and Distributed Processing Symposium - IPDPS*, v. 0, p. 1–10, 2006a.
- DEVI, U.; ANDERSON, J. *Soft real-time scheduling on multiprocessors*. Tese de Doutorado, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2006b.
- EGGERT, L.; HEIDEMANN, J. Application-level differentiated services for web servers. *World Wide Web*, v. 2, n. 3, p. 133–142, 1999.
- ESTRELLA, J. C.; TEIXEIRA, M. M.; SANTANA, M. J. Negotiation mechanisms on application level: a new approach to improve quality of service in web servers. *The 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and 2nd International Workshop on Collaborative Computing, Integration, and Assurance. SEUS/WCCIA*, v. 0, p. 255–260, 2006.
- FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. Hypertext transfer protocol – HTTP/1.1. 1999.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: Abstraction and reuse of object-oriented design. In: *ECOOOP '93*, Springer-Verlag, p. 406–431, 1993.
- GAO, S.; MIOC, D.; YI, X. The measurement of geospatial web service quality in sdis. In: *Proc. 17th Int Geoinformatics Conf*, p. 1–6, 2009.
- HENRIKSSON, D.; LU, Y.; ABDELZAHER, T. Improved prediction for web server delay control. *Proceedings of 16th Euromicro Conference on Real-Time Systems - ECRTS*, v. 0, p. 61–68, 2004.
- ISO/IEC *Information technology - quality of service: Framework*. Relatório Técnico, ISO/IEC International Organization for Standardization/International Electrotechnical Commission 13236, 1998.

- JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York, NY, USA, Wiley, 1991.
- KAK, A. C. *Programming with objects: A comparative presentation of object oriented programming with c++ and java*. 1 ed. Hoboken, New Jersey: John Wiley & Sons, 2003.
- KANG, K.-D.; SON, S. H.; STANKOVIC, J. A. Differentiated real-time data services for e-commerce applications. *Electronic Commerce Research, Special Issue on Business Process Integration and E-Commerce Infrastructure*, v. 3, n. 1-2, p. 113–142, 2003.
- KANODIA, V.; KNIGHTLY, E. W. Ensuring latency targets in multiclass web servers. *IEEE Transactions on Parallel and Distributed Systems*, v. 14, n. 1, p. 84–93, 2003.
- KATO, S.; YAMASAKI, N. Portioned EDF-based scheduling on multiprocessors. In: *Proceedings of the 7th ACM International Conference on Embedded Software - EM-SOFT*, New York, NY, USA, p. 139–148, 2008.
- KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a internet*. 3 ed. São Paulo: Pearson Addison Wesley, 634 p., 2006.
- LAPLANTE, P. A. *Real-time systems design and analysis*. 3 ed. Hoboken, New Jersey: Wiley-IEEE Press, 2004.
- LEE, S. C. M.; LUI, J. C. S.; MEMBER, S.; YAU, D. K. Y. A proportional-delay diffserv-enabled web server: Admission control and dynamic adaptation. *IEEE Transactions on Parallel and Distributed Systems*, v. 15, n. 5, p. 385–400, 2003.
- LEE, S. C. M.; LUI, J. C. S.; YAU, D. K. Y. A proportional-delay diffserv-enabled web server: admission control and dynamic adaptation. *IEEE_J_PDS*, v. 15, n. 5, p. 385–400, 2004.
- LEONTYEV, H.; ANDERSON, J. H. Tardiness bounds for FIFO scheduling on multiprocessors. In: *Proceedings of the 19th Euromicro Conference on Real-Time Systems - ECRTS*, Washington, DC, USA: IEEE Computer Society, p. 71–81, 2007.
- LIU, J. W. S. *Real-time systems*. 1 ed. Upper Saddle River, New Jersey 07458: Prentice Hall, 2000.
- LU, C.; ABDELZAHER, T. F.; STANKOVIC, J. A.; SON, S. H. A feedback control approach for guaranteeing relative delays in web servers. In: *Proceedings of the 7th IEEE Real-Time and Embedded Technology and Applications Symposium - RTAS*, Washington, DC, USA: IEEE Computer Society, p. 51–63, 2001.

- MAGALHÃES, M. F.; CARDOZO, E. *Qualidade de serviço na internet*. Relatório Técnico, Faculdade de Engenharia Elétrica e de Computação, Departamento de Engenharia de Computação e Automação Industrial - UNICAMP., Campinas, SP, 1999.
- MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, v. 30, p. 2004, 2003.
- MATHES, M.; HEINZL, S.; FREISLEBEN, B. WS-temporal policy: A WS-policy extension for describing service properties with time constraints. *32nd Annual IEEE International Computer Software and Applications Conference - COMPSAC*, v. 0, p. 1180–1186, 2008.
- MESSIAS, V. R.; ESTRELLA, J. C.; SANTANA, M. J.; SANTANA, R. H. C.; BRUSCHI, S. M.; TEIXEIRA, M. M. Implementação de um protótipo de servidor web distribuído com diferenciação de serviços. In: *XIII Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia.*, p. 103–110, 2007.
- MESSIAS, V. R.; SANTANA, M. J. *Servidor web distribuído com diferenciação de serviços - implementação e avaliação de um protótipo*. Dissertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação., São Carlos, Brasil, 2007.
- MI, N.; CASALE, G.; CHERKASOVA, L.; SMIRNI, E. Sizing multi-tier systems with temporal dependence: benchmarks and analytic models. *Journal of Internet Services and Applications*, v. 1, p. 117–134, 2010.
- MONACO, F. J.; NERY, M.; PEIXOTO, M. M. L. An orthogonal real-time scheduling architecture for responsiveness QoS requirements in SOA environments. In: *Proceedings of the 2009 ACM symposium on Applied computing*, New York, NY, USA: ACM, to be published, p. 1–6, 2009.
- MOSBERGER, D.; JIN, T. httpperf - a tool for measuring web server performance. In: *In First Workshop on Internet Server Performance*, ACM, p. 59–67, 1998.
- NANCY J. YEAGER, R. E. M. *Web server technology*, v. 1. 1 ed. San Francisco: Morgan Kaufmann Publishers, 407 p., 1996.
- NARLIKAR, G. J.; BLELLOCH, G. E. Pthreads for dynamic and irregular parallelism. In: *In Proc. of Supercomputing 98*, IEEE, p. 4–1, 1998.
- NERY, M.; MONACO, F. J. *Uma política de escalonamento de tempo-real para garantias de qos na web baseada em parâmetros média e dispersão do tempo de resposta*. Dis-

- sertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil, 2008.
- NETCRAFT July 2010 web server survey. WVisitado em 18 de Julho de 2010 - 7:55, 2010.
Disponível em http://news.netcraft.com/archives/2010/05/14/may_2010_web_server_survey.html
- NICHOLS, K.; BLAKE, S.; BAKER, F.; BLACK, D. RFC 2474: Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 headers. *Internet RFC, Internet Engineering Task Force - IETF*, 1999.
- NISSANKE, N. *Realtime systems*. 1 ed. Upper Saddle River, NJ, USA: Prentice-Hall Inc., 1997.
- PAN, W.; MU, D.; WU, H.; YAO, L. Feedback control-based QoS guarantees in web application servers. In: *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications - HPCC*, Los Alamitos, CA, USA: IEEE Computer Society, p. 328–334, 2008.
- PAXSON, V.; FLOYD, S. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, p. 226–244, 1995.
- PEIXOTO, M. L. M.; MONACO, F. J. *Políticas de escalonamento de tempo-real para garantia de qos absoluta em array de servidores web heterogêneos*. Dissertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil, 2008.
- PEIXOTO, M. L. M.; TOTT, R.; NERY, M.; MONACO, F. J. Arquitetura de escalonamento ortogonal de tempo-real para garantias de QoS em servidores web. In: *Workshop em Desempenho de Sistemas Computacionais e de Computação - WPerformance*, Anais do XXVIII Congresso da SBC, p. 18–37, 2008.
- PEZZÈ, M.; YOUNG, M. *Teste e análise de software: processos, princípios e técnicas*. 1 ed. Porto Alegre: Bookman, 2008.
- PRADHAN, P.; TEWARI, R.; SAHU, S.; CH, A.; SHENOY, P. An observation-based approach towards self-managing web servers. In: *Proceedings of the 10th International Workshop on Quality of Service - IWQoS*, Miami Beach, Florida, USA, p. 13–22, 2002.
- REENSKAUG, T. The model-view-controller (mvc). In: *Its Past and Present*, 2003.
- ROSE, M. T. *The simple book*. 2 ed. Prentice Hall, 336 p., 1996.

- RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. *The unified modeling language reference manual*. 1 ed. Massachusetts 01867: Addison Wesley Longman, Inc, 1999.
- SAITO, P.; MONACO, F. J. *Provisão integrada de qos relativa e absoluta em serviços computacionais interativos com requisitos de responsividade de tempo-real*. Dissertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil, 2010.
- SHI-JUN, Z.; XIN, Y.; SHAO-HUA, Y.; BEN-XIONG, H. A DMR fair algorithm for real-time scheduler. *2nd International Conference on Future Generation Communication and Networking - FGNC*, v. 1, p. 362–366, 2008.
- SHIMONISHI, H.; YOSHIDA, M.; FAN, R.; SUZUKI, H. An improvement of weighted round robin cell scheduling in atm networks. In: *Proc. IEEE Global Telecommunications Conference GLOBECOM '97*, p. 1119–1123, 1997.
- SLOMINSKI, A. A. Xml pull parser (xpp). World Wide Web page. <http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html>. Acessado em 9 de Agosto de 2010 - 9:20, 2005.
- Disponível em <http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html>
- STRNADL, C. At your service: QoS for the internet. *IEEE Multimedia - Media Reviews*, v. 9, n. 1, p. 93–95, 2002.
- SUBRAMANIAN, S. V.; DUTTA, R. Comparative study of secure vs. non-secure transport protocols on the sip proxy server performance: An experimental approach. In: *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, p. 301–305, 2010.
- TANENBAUM, A. S. *Redes de computadores*. 4 ed. Prentice Hall, 945 p., 2003.
- TAVARES, E.; SILVA, B.; MACIEL, P.; DALLEGRAVE, P. Software synthesis for hard real-time embedded systems with energy constraints. In: *Proceedings of the 2008 20th International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, Washington, DC, USA: IEEE Computer Society, p. 115–122, 2008.
- TEIXEIRA, M. M.; SANTANA, M. J.; SANTANA, R. H. C. Servidor web com diferenciação de serviços: Fornecendo qos para os serviços da internet. In: *XXIII Simpósio Brasileiro de Redes de Computadores (SBRC)*., Fortaleza, CE, p. 1–14, 2005.
- TOTT, R. F.; MONACO, F. J. *Extensões na política ebs - controle de admissão e redução da ordem de complexidade temporal*. Dissertação de mestrado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil, 2008.

- TRALDI, O. A.; BARBATO, A. K.; SANTANA, R. H. C. Service differentiating algorithms for QoS-enabled web servers. In: *Proceedings of the 12th Brazilian Symposium on Multimedia and the Web - WebMedia*, New York, NY, USA: ACM, p. 263–272, 2006.
- VASILIOU, N. Overview of internet QoS and web server QoS, reading course report. Department of Computer Science, The University of Western Ontario, London, Ontario, Canada., 2000.
Disponível em www.cs.uwo.ca/Research/DiGS/Papers/nikoread.ps
- WANG, Y.-T.; LIN, T.-P.; GAN, K.-C. An improved scheduling algorithm for weighted round-robin cell multiplexing in an atm switch. 1994.
- WEI, Y.; REN, F.; LIN, C.; VOIGT, T. Fuzzy control for guaranteeing absolute delays in web servers. In: *Proceedings of the 2nd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks - QSHINE*, Washington, DC, USA: IEEE Computer Society, p. 48–51, 2005.
- XIANGBIN, Z. Support QoS in open real-time systems. *International Conference on Computer Science and Software Engineering - CSSE*, v. 2, p. 190–193, 2008.
- XIAO, X.; NI, L., M. Internet QoS: A big picture. *IEEE Network*, v. 13, n. 2, p. 8–18, 1999.
- YE, N.; GEL, E. S.; LI, X.; FARLEY, T.; LAI, Y.-C. Web server QoS models: applying scheduling rules from production planning. *Computer and Operations Research*, v. 32, n. 5, p. 1147–1164, 2005.
- YE, N.; LI, X.; FARLEY, T.; XU, X. Job scheduling methods for reducing waiting time variance. *Computers and Operations Research*, v. 34, n. 10, p. 3069–3083, 2007.
- ZHANG, W. Abstract linux virtual server for scalable network services. "Linux Virtual Server for Scalable Network Services", was published at Ottawa Linux Symposium 2000., 2000.
- ZHANG, Y.; KRECKER, D. K.; GILL, C.; LU, C.; THAKER, G. H. Practical schedulability analysis for generalized sporadic tasks in distributed real-time systems. In: *Proceedings of the 20th Euromicro Conference on Real-Time Systems - ECRTS*, IEEE Computer Society, p. 223–232, 2008.
- ZHAO, W.; OLSHEFSKI, D.; SCHULZRINNE, H. *Internet quality of service: an overview*. Relatório Técnico, Technical Report CUCS-003-00, Columbia University, Computer Science Department, 2000.
Disponível em <http://www.cs.columbia.edu/techreports/cucs-003-00.pdf>

ZHOU, X.; WEI, J.; XU, C.-Z. Processing rate allocation for proportional slowdown differentiation on internet servers. In: *IEEE Transactions on Computers*, p. 88–97, 2004.

Este documento foi preparado com o formatador de textos L^AT_EX .