Modelo de Servidor Web com Quatro Módulos de
Atendimento de Requisições (SWMAR)

Geraldo Guiesi Junior

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 28.02.2008

Assinatura: _____

Modelo de Servidor Web com Quatro Módulos de Atendimento de Requisições (SWMAR)

Geraldo Guiesi Junior

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

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

**USP - São Carlos
Fevereiro/2008**

*Dedico este trabalho ao meu filho
Pedro, que me motiva a sempre
tentar alcançar maiores objetivos
para conseguir o mínimo que ele
merece.*

Agradecimentos

Agradeço aos meus pais, Geraldo e Telma, pela educação, por estarem presentes em todos os momentos difíceis e felizes da minha vida, pela confiança que depositam sempre em mim, pela preocupação, pelo apoio financeiro, enfim, por tudo.

À professora Regina Santana, pela orientação nesses últimos anos e pelos conhecimentos transmitidos durante o mestrado.

À professora Sarita por sempre estar disposta a me ajudar.

Ao professor Marcos Santana, pelas críticas e opiniões seguras.

Aos amigos do LaSDPC, Júlio, Luis, Lucas, Fábio, Valter, Augusto, Marcelo e Ottone, pela ajuda e por proporcionarem um ambiente de trabalho alegre.

A todos, que mesmo indiretamente, contribuíram para a conclusão deste mestrado.

Ao CNPq, pelo apoio financeiro dado a este trabalho.

RESUMO

Esta dissertação de mestrado apresenta a implementação e validação de um modelo de servidor *web* que divide o funcionamento de o servidor *web* em quatro módulos onde cada um desses módulos é responsável por uma etapa que a requisição percorre ao longo de seu processamento. Esses módulos são: atendimento da requisição (módulo 1), transferência do arquivo para a memória principal (módulo 2), processamento de requisições dinâmicas (módulo 3) e envio do arquivo ao cliente (módulo 4). Esses quatro módulos são interligados e são alimentados primeiramente por uma carga inicial gerada pelo gerador de cargas W4Gen e passa obrigatoriamente, nessa ordem, pelo módulo 1, módulo 2 e módulo 4. O módulo 3 só é utilizado quando se trata de uma requisição dinâmica. Ao ser atendido por um dos módulos, é atribuído um tempo de execução (leia-se tempo que a requisição toma para ser processada por esse módulo). Esses tempos foram baseados em trabalhos que fizeram *benchmarks* em servidores *web* reais. Os resultados alcançados com o desenvolvimento deste trabalho visam principalmente integrar-se aos trabalhos de simulação envolvendo servidores *web* do grupo de Sistemas Distribuídos e Programação Concorrente (LaSDPC) e com isso alcançar resultados próximos a resultados aplicados em servidores *web* reais.

Abstract

This work presents the implementation and validation of a web server model that divides the web server functions into four modules and each one of these is responsible for an execution step, in which a request goes through during processing. These modules are: request serving (module 1); file transferring to the main memory (module 2); dynamic request processing (module 3); and client file sending (module 4). These four modules are linked and the W4Gen generates the initial loads. These loads run in the order as follows: modules 1, 2 and 4. The module 3 is used only when the request is dynamic. When a request is served by a module, an execution time is defined (i.e. the time used by the request in the module), which is based on real world web servers benchmark's. The results obtained in the work aims to be integrated to other projects conducted in the Distributed System and Concurrent Programming group (LaSDPC) in order to reach results close to real world web servers.

Sumário

Lista de Figuras	vi
Lista de Tabelas	vii
Lista de Abreviaturas.....	viii
1. INTRODUÇÃO.....	1
1.1. CONTEXTUALIZAÇÃO.....	1
1.2. MOTIVAÇÃO	2
1.3. OBJETIVOS	3
1.4. ESTRUTURA DA DISSERTAÇÃO.....	3
2. A INTERNET.....	5
2.1. CONSIDERAÇÕES INICIAIS	5
2.2. MODELO DE REFERÊNCIA ISO/OSI E PROTOCOLO TCP/IP	6
2.2.1 Camada de Aplicação.....	7
2.2.2 Camada de Transporte	8
2.2.3 Camada de Rede	9
2.2.4 Camada de Enlace	10
2.2.5 Camada Física.....	11
2.3. A WORLD WIDE WEB	11
2.4. PROTOCOLO HTTP	12
2.5. CONSIDERAÇÕES FINAIS	16
3. ANÁLISE DE DESEMPENHO E SIMULAÇÃO	17
3.1. CONSIDERAÇÕES INICIAIS	17
3.2. CARGA DE TRABALHO.....	18
3.3. MÉTRICAS.....	18
3.4. MÉTODOS DE AVALIAÇÃO DE DESEMPENHO – TÉCNICAS DE AFERIÇÃO.....	19
3.4.1 Benchmarks.....	19
3.4.2 Monitoramento.....	20
3.5. MÉTODOS DE AVALIAÇÃO DE DESEMPENHO – TÉCNICAS DE MODELAGEM.....	21
3.5.1 Simulação.....	21
3.5.2 Solução Analítica (Teoria de Filas).....	22
3.6. CONSIDERAÇÕES FINAIS	23
4. SERVIDORES WEB	25
4.1. CONSIDERAÇÕES INICIAIS	25
4.2. FUNÇÕES DO SERVIDOR WEB.....	25
4.3. ARQUITETURA DOS SERVIDORES WEB	26
4.3.1 Requisições Estáticas.....	27
4.3.2 Requisições Dinâmicas	28
4.4. SESSÕES HTTP	30
4.5. ESTRUTURA INTERNA DOS SERVIDORES WEB	31

4.5.1	<i>Tratamento de Requisições TCP</i>	31
4.5.2	<i>Pool de Threads HTTP</i>	31
4.5.3	<i>Envio de Arquivos Requisitados</i>	32
4.5.4	<i>Descarga de Buffers de Saída</i>	33
4.6.	CONSIDERAÇÕES FINAIS	33
5.	MODELO DE SERVIDOR WEB COM QUATRO MÓDULOS DE ATENDIMENTO DE REQUISIÇÕES (SWMAR)	35
5.1.	CONSIDERAÇÕES INICIAIS	35
5.2.	TRABALHOS RELACIONADOS	35
5.2.1	<i>Modelos de Servidores Web</i>	36
5.2.2	<i>Análise de Servidores Web</i>	37
5.3.	W4GEN.....	39
5.3.1	<i>Código de Resposta</i>	40
5.3.2	<i>Classe de Objeto</i>	41
5.3.3	<i>Intervalo de Chegada</i>	42
5.3.4	<i>Tamanho de Objeto</i>	43
5.3.5	<i>Arquivo Resultante</i>	43
5.4.	SIMPACK.....	45
5.5.	MODELO DE SERVIDOR WEB COM QUATRO MÓDULOS DE ATENDIMENTO DE REQUISIÇÕES (SWMAR).....	46
5.5.1	<i>Módulo 1: Atendimento de Requisições HTTP</i>	48
5.5.2	<i>Módulo 2: Transferência de Arquivos à Memória Principal</i>	48
5.5.3	<i>Módulo 3: Script Engine – Execução de Requisições Dinâmicas</i>	50
5.5.4	<i>Módulo 4: Envio de Dados</i>	51
5.5.5	<i>Estimativa e Tempos de Serviço</i>	53
5.5.6	<i>Entrada de Dados</i>	57
5.5.7	<i>Saída de Dados</i>	59
5.6.	CONSIDERAÇÕES FINAIS	60
6.	ANÁLISE DE RESULTADOS	61
6.1.	CONSIDERAÇÕES INICIAIS	61
6.2.	CARACTERÍSTICAS GERAIS DE SIMULAÇÃO	61
6.3.	CONSIDERAÇÕES FINAIS	72
7.	CONCLUSÃO	73
7.1.	CONSIDERAÇÕES FINAIS	73
7.2.	CONTRIBUIÇÕES.....	74
7.3.	SUGESTÕES PARA TRABALHOS FUTUROS	75
8.	REFERÊNCIAS BIBLIOGRÁFICAS	77

LISTA DE FIGURAS

FIGURA 2.1 - CAMADAS, PROTOCOLOS E INTERFACES	6
FIGURA 2.2- CAMADA DE APLICAÇÃO (KUROSE; ROSS, 2003)	8
FIGURA 2.3 - CAMADA DE TRANSPORTE (KUROSE; ROSS, 2003).....	9
FIGURA 2.4 - CAMADA DE REDE (KUROSE; ROSS, 2003).....	10
FIGURA 2.5 - CAMADA DE ENLACE (KUROSE; ROSS, 2003)	11
FIGURA 2.6 - ESQUEMA DO PROTOCOLO URL.....	12
FIGURA 2.7 - COMPARAÇÃO ENTRE A VERSÃO 1.0 (A) E 1.1 (B) DO PROTOCOLO HTTP.....	14
FIGURA 2.8 - REQUISIÇÃO HTTP.....	14
FIGURA 2.9 - RESPOSTA HTTP.....	15
FIGURA 4.1 - TIPOS DE ARQUITETURA DE SERVIDORES WEB	26
FIGURA 4.2 - ESQUEMA CLIENTE-SERVIDOR NAS REQUISIÇÕES ESTÁTICAS	28
FIGURA 4.3 ESQUEMA CLIENTE-SERVIDOR DAS REQUISIÇÕES DINÂMICAS	30
FIGURA 4.4 - UM SERVIDOR HTTP DE UM PROCESSO POR CONEXÃO COM UM PROCESSO MESTRE.	32
FIGURA 4.5 - UM SERVIDOR DE UM ÚNICO PROCESSO COM MÚLTIPLAS THREADS.....	32
FIGURA 5.1- GERAÇÃO DE REQUISIÇÕES COM CÓDIGO 200 – W4Gen.....	40
FIGURA 5.2 - GERAÇÃO DE CÓDIGO DE STATUS BASEADO EM DISTRIBUIÇÃO – W4Gen.....	41
FIGURA 5.3 – GERAÇÃO DE CLASSES OBJETOS SEGUNDO DISTRIBUIÇÃO GEOMÉTRICA.....	41
FIGURA 5.4 – GERAÇÃO DE CLASSES DE OBJETOS SEGUNDO VALORES DE INTERFACE.....	42
FIGURA 5.5 – INTERVALO DE CHEGADA DAS CLASSES.....	43
FIGURA 5.6 – TAMANHO DE OBJETO DAS CLASSES.....	43
FIGURA 5.7 – GERAÇÃO DE CARGA DE TRABALHO.....	44
FIGURA 5.8 – EXEMPLO DE CARGA SINTÉTICA GERADA PELO W4Gen.....	44
FIGURA 5.9 – FLUXOGRAMA DO SWMAR.....	47
FIGURA 5.10 – ESQUEMA DE ATENDIMENTO NO MÓDULO 1.....	48
FIGURA 5.11 – FLUXO DE REQUISIÇÕES NO MÓDULO 2.....	50
FIGURA 5.12 – FLUXO DE REQUISIÇÕES NO MÓDULO 3.....	51
FIGURA 5.13 – FLUXO DE REQUISIÇÕES DO MÓDULO 4.....	52
FIGURA 5.14 – FUNCIONAMENTO DO ESVAZIAMENTO DE BUFFERS.....	52
FIGURA 5.15 – EXEMPLO DE DISTRIBUIÇÃO EXPONENCIAL.....	56
FIGURA 6.1 – CARACTERÍSTICAS DAS DIFERENTES CARGAS INICIAIS.....	62
FIGURA 6.2 – CARACTERÍSTICAS DOS TRÊS TIPOS DE LOGS.....	63
FIGURA 6.3 – UTILIZAÇÃO DOS MÓDULOS.....	64
FIGURA 6.4 – RESULTADO DE ALTERAÇÃO DE PARÂMETROS DO MÓDULO 1 - TRADICIONAL.....	64
FIGURA 6.5 - RESULTADO DE ALTERAÇÃO DE PARÂMETROS DO MÓDULO 1 – ACADÊMICO.....	65
FIGURA 6.6 - RESULTADO DE ALTERAÇÃO DE PARÂMETROS DO MÓDULO 1 – INFORMATIVO/NOTÍCIAS.....	65
FIGURA 6.7 – THROUGHPUT E TEMPOS DE SIMULAÇÃO – TRADICIONAL.....	66
FIGURA 6.8 – THROUGHPUT E TEMPOS DE SIMULAÇÃO – ACADÊMICO.....	66
FIGURA 6.9 – THROUGHPUT E TEMPOS DE SIMULAÇÃO – INFORMATIVO/NOTÍCIA.....	67
FIGURA 6.10 – RESULTADO DE ALTERAÇÃO DOS PARÂMETROS DE HARDWARE PARA LOG TRADICIONAL.....	68
FIGURA 6.11 – RESULTADO DE ALTERAÇÃO DOS PARÂMETROS DE HARDWARE PARA LOG ACADÊMICO.....	68
FIGURA 6.12 - RESULTADO DE ALTERAÇÃO DOS PARÂMETROS DE HARDWARE PARA LOG INFORMATIVO/NOTÍCIAS.....	68
FIGURA 6.13 – PORCENTAGEM DE UTILIZAÇÃO DOS MÓDULOS – LOG TRADICIONAL.....	69
FIGURA 6.14 – PORCENTAGEM DE UTILIZAÇÃO DOS MÓDULOS – LOG ACADÊMICO.....	69

FIGURA 6.15 – PORCENTAGEM DE UTILIZAÇÃO DOS MÓDULOS – LOG INFORMATIVO/NOTÍCIA. ...	69
FIGURA 6.16 – THROUGHPUT – LOG TRADICIONAL.	70
FIGURA 6.17 – THROUGHPUT – LOG ACADÊMICO.	71
FIGURA 6.18 – THROUGHPUT – LOG INFORMATIVO/NOTÍCIA.	71

LISTA DE TABELAS

Tabela 1 – Camadas e funções do modelo OSI	7
Tabela 2 – Métodos e funções existentes nas mensagens de requisição do protocolo http 1.1	15
Tabela 3 – Códigos de resposta do protocolo HTTP 1.1	16
Tabela 4 – Classificação de tempos de acordo com as classes de objetos.....	54
Tabela 5 – Tempos de utilização do CPU no processamento de requisições.....	56
Tabela 6 – Cenários de simulação com alteração de parâmetros do Módulo 1.....	66
Tabela 7 – Cenários de testes com alterações de parâmetros de hardware.....	69

LISTA DE ABREVIATURAS

ARPA	Advanced Research Projects Agency
CGI	Common Gateway Interface
CISC	Complex Instruction Set Computing
CPU	Unidade central de processamento
DNS	Domain Name System
DoD	Departamento de Defesa
FCFS	First Come, First Served
FTP	File Transfer Protocol
HTML	Hyper Text Markup Language
HTTP	Protocolo de Transferência de Hipertexto
Httpd	daemon http
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
JSP	Java Server Pages
LCFS-PR	First Come, First Served with Preempt and Resume
MFLOPS	Million Floating-point Operations Per Second
MIPS	Millions of instructions per second
MSS	Maximal Segment Size
OSI	Open System Interconnection
PPP	Point-to-Point Protocol
QoS	Qualidade de Serviço
RFC	Request for Comments
RISC	Reduced (or regular) instruction set computer
RR	Round-Robin
RTT	Round Trip Time
SE	Script Engine
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WWW	World Wide Web

CAPÍTULO 1

Introdução

1.1. Contextualização

A popularidade da *World Wide Web*, também chamada WWW ou simplesmente *Web*, tem aumentado significativamente nos últimos anos. Em dezembro de 1992, o tráfego da WWW era praticamente inexistente, sendo no total de apenas 74MB por mês no *backbone* da rede NSFNET (ARLITT; WILLIAMSON, 1997). Hoje, o tráfego WWW é um dos componentes dominantes da Internet (GILDER, 1997).

Algumas explicações para o crescimento exponencial da Internet são: o fácil uso da *Web* mediante a disponibilidade de interfaces gráficas para a navegação e para a visualização de documentos; a existência de editores e ferramentas de suporte para criação e publicação de documentos *Web*; crescente tendência entre pesquisadores, educadores, instituições, e organizações comerciais em usar a Internet para disseminar informações *timely fashion*¹; a independência de localidade dos documentos *Web* que podem ser referenciados livremente; e um contínuo crescimento exponencial no número de *hosts* e usuários da Internet (ARLITT; WILLIAMSON, 1997).

Acompanhado com o crescimento da *Web* veio o surgimento de novas utilizações que geraram problemas não enfrentados com as aplicações originais. Esses problemas aconteceram pelo fato da Internet passar, de uma rede para fins de pesquisa, restrita a órgãos governamentais e instituições acadêmicas onde o tráfego era baseado em documentos textuais, para atualmente utilizar um dos mais importantes meios de comunicação, para fins informativos, educacionais, comerciais e entretenimento.

¹ Informações publicadas em “tempo real”.

Hoje na Internet trafegam, além de documentos textuais, documentos multimídia (vídeos e áudios). Segundo (COMER, 2000) há a tendência de convergência da Internet com outras redes, como as de telefonia, rádio e TV, o que tende a piorar ainda mais o problema.

Outro fato que afeta o desempenho da *Web* está relacionado à arquitetura da Internet. Sem administração centralizada e escalonável, a Internet permite a inserção de dispositivos clientes heterogêneos, o que aumenta a necessidade de autenticação e sistemas de segurança para o funcionamento de tais dispositivos clientes. O aumento de autenticação e segurança está atrelado ao aumento da complexidade do *middleware*² e das aplicações *software* (CARDELLINI et al., 2002).

Como consequência de tais problemas surge um aumento do tráfego, das cargas nos servidores *web* e, devido a esses problemas, no número de recusas de atendimento e falha de envio de resposta. Esses problemas podem ser atribuídos, em parte, ao aumento de carga nos servidores *web* graças à mudança de conteúdos suportados, passando de páginas estáticas para páginas dinâmicas e também ao crescimento desproporcional da capacidade das redes com relação aos servidores *web* (GRAY; SHENOY, 2000).

Essa defasagem sofrida pelos servidores *web* são visíveis aos usuários finais, que sofrem com uma queda na qualidade do serviço oferecido, ou QoS³ na *Web*.

1.2. Motivação

Em um ambiente *Web*, um usuário não se importa com gargalos no tráfego, servidores sobrecarregados, capacidade total da rede⁴, entre outros efeitos. Além do conteúdo, os usuários exigem desempenho, disponibilidade e segurança (MENASCÉ; ALMEIDA, 2003).

Como solução para a queda na qualidade dos serviços oferecidos na *Web*, no início eram feitas atualizações tanto nos meios de rede quanto nos próprios servidores *web* (DEVLIN et al., 1999). Mas essa técnica simplesmente expande a capacidade do sistema pela adição de mais recursos, resolvendo apenas temporariamente o problema. Essa técnica não é muito apropriada porque, além de inutilizar os servidores antigos demandavam muitos recursos financeiros para a aquisição de novos equipamentos (CPU, disco, interface de disco) (CARDELLINI et al., 2002).

² Software de interface que permite interação de diferentes aplicações de softwares, geralmente sobre diferentes plataformas de hardware e infra-estrutura, para troca de dados.

³ *Quality of Service*.

⁴ Bandwidth

Com a capacidade das redes crescendo mais rapidamente que o previsto pela lei de Moore⁵, os servidores *web* tornam-se o gargalo da *Web* sendo que recentes medições mostram que 40% do atraso na *Web* são de responsabilidade das transações que acontecem nos servidores *web* (HIUTEMA, 2000).

Uma forma de avaliar os servidores *web*, visando encontrar a melhor proposta para atender à crescente demanda pelos servidores *web*, é a utilização de modelos de servidores *web*. Esses modelos podem responder perguntas especulativas tais como: qual será o comportamento do servidor *web* se um determinado componente interno foi acrescido, ou se for feita alguma alteração na política de esvaziamento dos *buffers* de saída de dados.

Este projeto foi motivado justamente pela capacidade das técnicas de modelagem em responder às perguntas “e se” e também pela capacidade de integrar outros projetos em desenvolvimento no grupo de sistemas distribuídos do ICMC da USP como (TEIXEIRA, 2004) e (ESTRELLA, 2005) podendo substituir os servidores *web* utilizados nesses projetos, modelados como “caixas-pretas”, por modelos de servidores mais detalhados.

1.3. Objetivos

O objetivo do projeto apresentado nesta monografia é desenvolver um modelo de servidor *web*, baseado em SANT’ ANNA (2004), onde sejam modelados componentes internos responsáveis pelo tratamento das conexões desde o estabelecimento da conexão até o envio da resposta ao cliente. E como foi dito anteriormente, integrar outros projetos do grupo de Sistemas Distribuídos e Programação Concorrente (LaSDPC).

Esse modelo deve considerar tanto requisições estáticas quanto requisições dinâmicas, protocolo HTTP 1.1.

1.4. Estrutura da Dissertação

Esta dissertação é dividida em 7 capítulos. No capítulo 1 foi introduzido o tema de trabalho a ser desenvolvido, identificando a contextualização, a motivação e os objetivos a serem alcançados durante o mestrado.

⁵ A capacidade dos computadores dobra a cada 18 meses.

No capítulo 2 serão apresentadas a Internet e a *Web*, conceitos importantes para situar o leitor no contexto onde os servidores *web* trabalham. Nesse capítulo serão explicados os protocolos TCP/IP e suas camadas e HTTP, além de apresentar os conceitos relacionados a *Web*.

No capítulo 3 serão apresentados conceitos envolvidos na avaliação de desempenho e simulação. Conceitos muito importantes para o desenvolvimento do projeto.

No capítulo 4 serão apresentadas características dos servidores *web* e técnicas de simulação desses elementos da *Web*. Nesse capítulo, são mostradas as características das arquiteturas disponíveis de servidores *web* e as aplicações relativas a *Web*. Ainda nesse capítulo, são apresentados alguns modelos da literatura que simulam servidores *web*. Dentre esses modelos são destacados os descritos em (SANT' ANNA, 2004), (MEI, 2001) e (DILLEY *et al.*, 2000).

No capítulo 5 é apresentado o modelo desenvolvido e trabalhos relacionados que serviram de base, inspiração e que contribuíram de alguma forma para a construção do modelo.

No capítulo 6 são apresentados testes em nove cenários utilizando três tipos de *logs* como carga inicial gerados pelo W4Gen (SILVA, 2006) e são apresentados resultados desses testes.

E, finalmente, são apresentadas as referências bibliográficas utilizadas para a elaboração desta dissertação.

CAPÍTULO 2

A Internet

2.1. Considerações Iniciais

No alto da Guerra Fria o Departamento de Defesa (DoD) necessitava de uma rede de comunicação que, mesmo em caso de um ataque nuclear continuasse funcionando. Até então, a comunicação era feita inteiramente através da rede pública de telefone. Para resolver esse problema foi criada a ARPA (*Advanced Research Projects Agency*), baseada em pesquisas desenvolvidas por diversos especialistas, dentre eles Larry Roberts que propôs a ARPANET. A ARPANET surgiu em 1967 tornou-se o *backbone* da Internet (COMER, 2000).

A ARPANET usava comunicação ponto-a-ponto, além de fazer troca de pacotes entre diferentes redes como: a de telefonia, a rede de rádio e canais de comunicação de satélites. No início a ARPANET era utilizada apenas para fins de pesquisa pelo fato que os nós da ARPANET eram localizados em algumas instituições acadêmicas e por isso eram muito utilizados por pesquisadores do meio acadêmico. Apesar da integração da ARPANET havia o problema de não haver um protocolo padrão. Em 1983, para solucionar esse problema foi adotado o protocolo TCP/IP (SOCOLOFSKY; KATE, 1997), aceito principalmente por causa da versão *Berkley* UNIX que, além de ter incluso o protocolo TCP/IP o colocou em situação de domínio público possibilitando a alteração e desenvolvimento de aplicações.

Com a abertura do protocolo TCP/IP surgiram comitês e órgãos que participavam da criação e divulgação de tecnologias e novos protocolos, destacando o IETF, que até hoje é responsável pela manutenção e apoio aos padrões de Internet e TCP/IP (principalmente através dos documentos conhecidos como RFC). Tais documentos descrevem diversos protocolos e servem de base para novas tecnologias. Tais fatores foram de vital importância para o crescimento e adesão de novos nós para a Internet.

2.2. Modelo de referência ISO/OSI e Protocolo TCP/IP

Para reduzir a complexidade, muitas redes de comunicação são organizadas como uma pilha de camadas ou níveis, cada uma construída sobre a anterior. O número de camadas, o nome e o conteúdo de cada camada diferem de rede para rede. O propósito de cada camada é oferecer certos serviços para camadas superiores, acobertando para uma camada como foi provido um determinado serviço. Outro aspecto importante das camadas é o fato que uma camada n de uma máquina só tem a possibilidade de se comunicar com a camada n de outra máquina como visto na figura 2.1. As regras e contenções usadas nessa comunicação são conhecidas como protocolo da camada n (TANENBAUM, 2003). Um protocolo é um acordo entre as partes em comunicação de como a informação será processada. A quebra de protocolo pode tornar a comunicação difícil quando não impossível.

Figura 2.1 - Camadas, protocolos e interfaces

A comunicação entre as camadas de mesmo nível são demonstradas nas figuras 2.2, 2.3, 2.4, 2.5. Essas figuras representam a comunicação entre os níveis equivalentes.

Em 1997 a ISO (*International Organization for Standardization*) tomou o primeiro passo para a padronização internacional dos protocolos de diferentes camadas (Tanenbaum, 2003). O modelo foi chamado ISO OSI (*Open System Interconnection*) *Reference Model* ou simplesmente OSI.

O modelo OSI tem sete camadas, que são apresentadas na tabela 3

Camada	Função
Aplicação	Contém uma variedade de protocolos comumente usados pelo usuário.
Apresentação	Camada responsável com a sintaxe e a semântica da informação transmitida.
Sessão	Permite que usuários de diferentes máquinas estabeleçam sessões de comunicação.
Transporte	Sequenciamento, confirmação e recebimento de pacotes.
Rede	Roteamento de dados através da rede.
Enlace	Formatação de informação em quadros.
Física	Conexão física entre o sistema operacional e a rede.

Tabela 1 – Camadas e funções do modelo OSI

Uma camada de protocolo pode ser implementada em software, em hardware, ou em uma combinação dos dois. Protocolos de camada de aplicação como HTTP (POSTEL, 1981) e SMTP (*Simple Mail Transfer Protocol*) (POSTEL, 1982) – quase sempre são implementados em software. O mesmo acontece com protocolos de camada de transporte. Como a camada física e a camada de enlace de dados são responsáveis pelo manuseio da comunicação por um enlace específico, normalmente são implementadas em placas de interface de rede. Já a camada de rede geralmente é implementada de forma mista, hardware e software.

O primeiro modelo de referência foi o Modelo de Referência TCP/IP, definido em (CERF; KAHN, 1974). O protocolo TCP/IP é o que está atualmente em uso. As camadas e as características desse modelo são apresentadas a seguir.

2.2.1 Camada de Aplicação

Nessa camada residem aplicações de rede e seus protocolos. Ela inclui diversos protocolos, tais como: o protocolo HTTP, provê requisição e transferência de documentos pela *Web*; o SMTP provê transferência de mensagens de correio eletrônico; o FTP (*File Transfer Protocol*) (POSTEL, 1985), provê transferência de arquivos entre dois sistemas finais e o DNS (*Domain Name System*) (POSTEL, 1984), utilizado para resolver domínios de acordo com o IP (*Internet Protocol*) (POSTEL, 1981) (KUROSE; ROSS, 2003).

Na figura 2.2 é exemplificada a comunicação entre as camadas de aplicação.

Figura 2.2- Camada de Aplicação (KUROSE; ROSS, 2003)

2.2.2 Camada de Transporte

Onde são transportadas mensagens da camada de aplicação entre os lados do cliente e servidor de uma aplicação. Os protocolos de transporte na Internet: TCP e UDP. O TCP oferece serviços orientados à conexão para suas aplicações, além de possuir garantia de mensagens da camada de aplicação ao destino e controle de fluxo, fragmenta mensagens longas em fragmentos mais curtos e provê controle de congestionamento. Já o protocolo UDP não provê serviço não orientado à conexão.

Na figura 2.2 é exemplificada a comunicação entre as camadas de transporte.

Figura 2.3 - Camada de transporte (KUROSE; ROSS, 2003)

2.2.3 Camada de Rede

A camada de rede é a responsável pela movimentação de pacotes entre as máquinas, também chamados de datagramas. Essa camada possui dois componentes principais: o protocolo que define os campos no datagrama e o protocolo de roteamento. No protocolo que define os campos do datagrama, tanto os sistemas finais quanto os roteadores agem nesses campos, o protocolo IP. E o protocolo de roteamento determina as rotas que os datagramas seguem entre origem e destino. Embora a camada de rede contenha o protocolo IP (POSTEL, 1981) também numerosos protocolos de roteamento, ela quase sempre é denominada simplesmente camada IP.

O protocolo da camada de transporte da Internet (TCP ou UDP) em uma máquina de origem passa um segmento de camada de transporte e um endereço de destino à camada de rede. Essa camada provê o serviço de entrega do segmento à camada de transporte na máquina destinatária.

A figura 2.4 esquematiza o roteamento sofrido por um datagrama na camada de transporte.

Figura 2.4 - Camada de rede (KUROSE; ROSS, 2003)

2.2.4 Camada de Enlace

A camada de rede da Internet roteia 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 passa o datagrama para a camada de enlace que o entrega, ao longo da rota, ao nó seguinte. No nó destino o datagrama é passado da camada de enlace para a de rede.

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* – protocolo ponto-a-ponto) (SIMPSON, 1994). Como datagramas normalmente precisam transitar por diversos enlaces para percorrer o caminho desde a origem até o destino estes podem ser manuseados por diferentes protocolos de camada de enlace em diferentes enlaces ao longo de sua rota, como visto na figura 2.5. Em alguns casos, podem, por exemplo, ser manuseados 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. Os pacotes de camada de enlace são denominados quadros (TANENBAUM, 2003).

Figura 2.5 - Camada de enlace (KUROSE; ROSS, 2003)

2.2.5 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 para par de fios trançados, um para cabo coaxial, um outro para fibra e assim por diante. Em cada caso, o bit é movimentado pelo enlace de um modo diferente.

2.3. A World Wide Web

Até a década de 1990, a Internet era usada por pesquisadores, acadêmicos e estudantes universitários para se interligar com hospedeiros remotos, transferir arquivos de hospedeiros locais para hospedeiros remotos e vice-versa, enviar e recebe notícias e enviar e receber correio

eletrônico. Embora essas aplicações fossem (e continuem sendo) extremamente úteis, a Internet não era conhecida fora das comunidades acadêmicas e de pesquisa. No início da década de 1990, entrou em cena uma nova aplicação – A *World Wide Web* (BERNERS-LEE, 1994). A *Web* é a aplicação da Internet que chamou a atenção do público em geral. Ela transformou significativamente a maneira como pessoas interagem dentro e fora de seus ambientes de trabalho. Alçou a Internet de apenas mais uma entre muitas redes de dados (X.25 e redes *frame relay*) para, essencialmente, a única rede de dados. (KUROSE; ROSS, 2003).

O funcionamento da *Web* é baseado no paradigma cliente-servidor (TANENBAUM, 1995) onde há uma estrutura de interligação entre os documentos. Essa estrutura permite que um documento referencie outro que pode estar distribuído geograficamente (em outras máquinas na Internet). Do ponto de vista do usuário, a *Web* consiste de uma vasta coleção de documentos, ou páginas, que podem conter *links* para outras páginas relacionadas. Para visualizar as páginas são utilizados programas chamados *browsers*, que merecem crédito pelo crescimento no número de usuários devido à interface gráfica disponível, que em um clique, pode levar o usuário a um novo documento (ARLITT; WILLIAMSON, 1997).

Figura 2.6 - Esquema do protocolo URL.

As páginas são referenciadas usando o URL (*Uniform Resource Locators*). O URL, como é mostrado na Figura 2.6, é dividido em 5 partes: o protocolo utilizado, o nome do servidor acessado através do DNS (MOCKAPETRIS, 1987), a porta pela qual o servidor recebe as conexões (*default* porta 80), o diretório onde o arquivo está armazenado e o nome do arquivo.

2.4. Protocolo HTTP

O protocolo de transferência usado sobre a *World Wide Web* é o HTTP (*HyperText Transfer Protocol*). Ele especifica quais mensagens podem ser enviadas para o servidor e quais respostas serão retornadas ao cliente. O protocolo HTTP estabelece uma conexão cliente-servidor, transmite e recebe parâmetros incluindo um arquivo retornado e encerra a conexão

cliente-servidor. Os servidores produzem conteúdo (páginas HTML) independente da plataforma do cliente. Os *browsers* são responsáveis pelos detalhes específicos para o sistema. O protocolo HTTP utiliza o TCP como protocolo de transporte.

Quando um cliente solicita uma página *Web*, o *browser* envia uma requisição HTTP de objetos ao servidor que, por sua vez responde com uma mensagem de resposta HTTP que contém os objetos. Existem várias versões do protocolo HTTP. Até 1997 a versão que dominava era a HTTP/1.0 definida na RFC 1945 (BERNERS-LEE, 2000). A partir de 1998 alguns servidores *Web* e *browsers* começaram a implementar a versão HTTP/1.1, definida na RFC 2616 (FIELDING ET AL., 1999). Para manter a compatibilidade, os servidores *Web* que rodam a versão 1.1 “conversam” com os *browsers* que rodam versão 1.0 e um *browser* que roda a versão 1.1 pode “conversar” com um servidor que roda a versão 1.0.

O protocolo HTTP é conhecido como protocolo sem estado (*stateless*) por não incluir o conceito de sessão ou iteração além da entrega do documento solicitado. No protocolo HTTP original, uma conversação é restrita à transferência de documentos e imagens, não mantendo qualquer informação de requisições passadas. As duas versões do protocolo HTTP possuem diferenças quanto ao tratamento das conexões após o envio do conteúdo, como mostrado na figura 2.7. Em ambas versões, a ordem seguida é a seguinte: 1) é estabelecida a conexão entre cliente e servidor através do *three hand shake* (COMER, 2000); 2) o cliente envia a mensagem HTTP de requisição HTML para o servidor; 3) o servidor envia uma mensagem de confirmação de recebimento da requisição (ACK) 4) o servidor envia o conteúdo requisitado. A partir desse ponto há divergência entre as versões 1.0 e 1.1. Na 1.0, após o envio do conteúdo, o servidor envia uma mensagem para encerrar a conexão entre cliente e servidor (FIN). Com isso, se a mesma página for requerida logo em seguida, será efetuado todo o processo, desde o estabelecimento da conexão, até a chegada do conteúdo para o cliente. Já na versão 1.1 do protocolo HTTP há a diferença de que a conexão pode ser mantida aberta (conexões persistentes) por um tempo (*timeout*) na esperança que novas requisições sejam feitas para o mesmo servidor, evitando o *overhead* de criar novamente a requisição.

Figura 2.7 - Comparação entre a versão 1.0 (a) e 1.1 (b) do protocolo HTTP.

O protocolo HTTP possui dois tipos de mensagens, as de requisição e as de resposta. Nas requisições, (figura 2.8) a primeira linha está o método HTTP a ser aplicado, a localização do objeto no servidor e a versão do protocolo. Além dessas informações, é possível saber o nome do

```
GET /dir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
(extra carriage return, line feed)
```

Figura 2.8 - Requisição HTTP.

servidor *Web* (linha 2), o tipo de *browser* utilizado pelo cliente (linha 4) e a linguagem padrão do cliente (linha 5). O campo *connection* é responsável pelas conexões persistentes, se o campo estiver com valor *close*, mesmo sendo um cliente versão 1.1, a conexão será fechada.

A tabela a seguir mostra alguns métodos de requisição do protocolo HTTP.

Métodos	Função
GET	Solicita que seja retornado o recurso identificado pela URL
HEAD	Obtém informações sobre o recurso sem que o mesmo seja retornado ao cliente. Testa a validade de links, acessibilidade e a data da última modificação.
POST	Estabelece e encerra enlaces de comunicação.
OPTIONS	Sequenciamento, confirmação, recebimento de pacotes de dados.
PUT	Roteamento de pacotes através da rede.
DELETE	Formatação da informação em quadros
TRACE	Conexão física entre o sistema operacional e a rede.
CONNECT	Conexão física entre o sistema operacional e a rede.

Tabela 2 – Métodos e funções existentes nas mensagens de requisição do protocolo http 1.1.

As mensagens de resposta seguem o padrão mostrado na figura 2.9.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 2003 12:00:15 GMT
Server: Apache/1.3.31 (Linux)
Last-Modified: Mon, 22 Oct 2003 09:23:24 GMT
Content-Length: 6831
Content-Type: text/html

(data data data data data ...)
```

Figura 2.9 - Resposta HTTP.

Na primeira linha tem-se a versão do protocolo e um código de *status*, no exemplo, a solicitação foi completa com sucesso (código 200). A linha *connection*, com o valor *close*, diz que a conexão TCP será fechada após o término do envio da mensagem. Os três campos seguintes são informativos quanto a data e horário da resposta, o tipo de servidor *web*, a última atualização do arquivo, o tamanho do arquivo e o tipo do documento (text/html).

O protocolo HTTP oferece uma lista com códigos de status para os servidores *web* informarem aos clientes o grau de sucesso obtido na tentativa de atender uma requisição. As classes de códigos de respostas podem ser observados na tabela 3.

Classes	Descrição
1xx	Requisição recebida e em fase de processamento.
2xx	Requisição recebida, entendida e processada com sucesso.
3xx	Redireciona o cliente para uma outra URL.
4xx	Requisição inválida.
5xx	Incapacidade do servidor em processar a requisição.

Tabela 3 – Códigos de resposta do protocolo HTTP 1.1.

2.5. Considerações finais

Neste capítulo foram abordados os conceitos básicos da *Web*, um assunto importante para este trabalho. Foi feito um breve histórico da criação da internet seguido pela explicação dos modelos de referência ISO/OSI, do protocolo TCP/IP e do funcionamento das camadas desse protocolo. Também foi abordado o protocolo HTTP e as mensagens de requisições e respostas possíveis.

O conhecimento da estrutura pela qual uma requisição a um servidor *web* necessita passar é de vital importância para que a modelagem e posteriormente a simulação sejam bem fundamentadas e elaboradas. Um reforço positivo para tal estudo pode ser relacionado ao crescimento de aplicações que necessitam de QoS, que pode vir a não existir se houver atraso nas requisições e nas entregas de dados.

No próximo capítulo será abordada a avaliação de desempenho e alguns métodos de simulação, dois assuntos que foram importantes para o desenvolvimento deste projeto.

Análise de Desempenho e Simulação

3.1. Considerações Iniciais

Avaliar o desempenho de um sistema computacional é essencial para obter conclusões sobre o desempenho e o comportamento em determinadas situações. Em todo o ciclo de vida de um sistema de computador, desde o projeto até sua atualização, o estudo de avaliação permite aos responsáveis do projeto abstrair informações importantes, tais como, o melhor sistema para uma dada aplicação e a quantidade de recursos a ser utilizado. Apesar de sua importância, muitas vezes esta atividade é desprezada com consequências drásticas para o sistema final. Essa falta de interesse está relacionada, principalmente, a dois fatores: os responsáveis não dominam o emprego de ferramentas de análise; ou a avaliação torna-se uma tarefa difícil, apesar do analista possuir conhecimento sobre as ferramentas (SANTANA et al., 1994).

A avaliação de desempenho é uma arte, pois é algo que não pode ser feito mecanicamente (JAIN, 1991). A avaliação de um sistema envolve conhecimento do sistema modelado e uma seleção cuidadosa de metodologia, carga de trabalho e ferramentas. O problema deve ser definido e convertido para uma forma em que as ferramentas e técnicas existentes possam ser usadas, considerando uma série de requisições (entre elas o tempo disponível). Cada analista tem seu estilo: dado o mesmo problema, podem-se escolher métricas e métodos de avaliação diferentes. E, com os mesmos dados obtidos, pode-se chegar a conclusões diferentes, e até opostas (JAIN, 1991). As principais técnicas para avaliar desempenho são técnicas de aferição e modelagem. As técnicas de aferição podem utilizar *benchmarks*, monitoramento ou coleta de dados diretamente no sistema. Já as técnicas de modelagem podem usar Redes de Petri (FRANCÊS, 2000), Redes de filas e *Statecharts* (HAREL, 1987). Os modelos podem ser

resolvidos por simulação ou analiticamente (FRANCÊS, 1998). Muitas vezes é usada uma combinação dessas técnicas.

A seguir serão explicados alguns fundamentos utilizados na avaliação desempenho.

3.2. Carga de trabalho

A carga de trabalho de um sistema se refere às solicitações feitas pelos usuários, entidades que fazem pedidos de serviços no sistema. Exemplos de carga são instruções enviadas a uma CPU, consulta feita a uma base de dados ou transações num sistema bancário. Uma carga real é aquela observada em operações normais, não pode ser repetida de modo controlado e, em geral, não é adequada para uso como teste. Assim, freqüentemente é desenvolvida uma carga sintética (cujas características devem ser similares àquelas da carga real) que pode ser aplicada repetidamente e de forma controlada para estudos. Essa carga sintética é chamada *benchmark* e se constitui de programas ou partes de programas relativos às aplicações mais utilizadas (SILVA, 2005).

3.3. Métricas

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

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

Do ponto de vista do sistema, a principal métrica é o *throughput*⁶: o número médio de pedidos processados por unidade de tempo.

Para CPU⁷s, o desempenho é medido em MIPS (Milhões de Instruções por Segundo) ou MFLOPS (Milhões de Operações de Ponto Flutuante por Segundo). É importante ressaltar que

⁶ Velocidade com que um computador processa dados. É uma combinação da velocidade de processamento interno, velocidade dos periféricos (I/O) e eficiência do sistema operacional e outros Software do sistema, todos funcionando juntos.

⁷ Unidade central de Processamento.

não faz sentido comparar MIPS em diferentes arquiteturas de CPU, por exemplo, RISC⁸ e CISC⁹ devido à diferença entre as instruções utilizadas.

Para redes, o *throughput* é medido em pps (pacotes por segundo) ou bps (bits por segundo). Observa-se que o tempo de resposta obtido com o *throughput* máximo pode ser alto demais para ser aceitável. O ponto ótimo de uma aplicação é encontrado com o cálculo *throughput* versus tempo de resposta. É o ponto além do qual o tempo de resposta aumenta rapidamente com a carga, mas o ganho em *throughput* é pequeno. O *throughput* máximo alcançável sob condições ideais de carga de trabalho é chamado de capacidade nominal do sistema; para redes, isso é denominado *bandwidth*¹⁰. A razão da capacidade utilizável (*throughput* máximo alcançável sem exceder um limite de tempo de resposta pré-especificado) para a capacidade nominal é chamada de eficiência.

A taxa de utilização de um recurso é determinada pela porcentagem de tempo que o recurso está ocupado, para determinada carga (SANT'ANNA, 2004). O recurso do sistema que primeiramente atingir 100% de utilização, o que ficar saturado em primeiro lugar, é chamado de gargalo. A otimização no desempenho desse recurso é a que oferece o maior retorno em termo de investimento. Por isso, achar a utilização de recursos no sistema é uma parte importante na análise global.

Outras medidas de desempenho que devem ser citadas são:

- Confiabilidade: medida pela probabilidade de erros ou tempo médio entre erros;
- Disponibilidade: fração de tempo que o sistema está disponível para utilização;

3.4. Métodos de Avaliação de Desempenho – Técnicas de Aferição

3.4.1 Benchmarks

Benchmarks são programas ou partes agrupadas de programas, de diferentes tipos. Esses programas podem ser relativos tanto para aplicações comerciais quanto para aplicações científicas por exemplo, e que devem representar a carga real a ser submetida ao sistema; são usados em testes para comparações ou em estudos de projeto ou planejamento.

⁸ Conjunto Reduzido de Instruções de Computador.

⁹ Conjunto de Instruções Complexas de Computador.

¹⁰ Largura de banda.

Os *benchmarks* possuem baixo custo, porém apresentam limitações como influência do compilador, da biblioteca em tempo de execução e tamanho do cache. Além disso, não representam em geral sistemas interativos. Alguns *benchmarks* podem abordar basicamente a CPU, a Unidade de Ponto Flutuante e um subsistema de memória; outros podem ser direcionados para comparação de I/O e outros subsistemas.

3.4.2 Monitoramento

Um monitor é uma ferramenta usada para observar as atividades em um sistema (JAIN, 1991). Em geral, os monitores observam o desempenho de sistemas, coletam estatísticas de desempenho, avaliam os dados, e mostram resultados. Alguns também identificam áreas de problemas e sugerem soluções. Alguns exemplos de utilização de monitores são:

- Um gerente de sistemas pode usar um monitor para medir as utilizações de recursos e achar o gargalo, ou também para ajustar os parâmetros do sistema visando melhoria no desempenho.
- Um analista pode fazer um monitoramento para caracterizar a carga de trabalho, sendo os resultados usados para planejamento de capacidade ou para criar cargas de trabalho para testes.
- Um programador pode usar o monitor para achar os segmentos do software freqüentemente usados e otimizar seu desempenho.

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

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

Os monitores de *hardware* são dispositivos de *hardware* específicos para coletar e avaliar dados do objeto em estudo. Não devem ser intrusos em relação ao sistema. Os monitores de hoje são inteligentes e programáveis contendo seu próprio processador, memória e dispositivos de I/O. Consistem de vários elementos, como por exemplo: *probes* – para observar

sinais em pontos desejados do *hardware* do sistema; contadores – incrementados quando um certo evento ocorre e temporizadores – para marcar o tempo ou disparar uma operação de montagem.

3.5. Métodos de Avaliação de Desempenho – Técnicas de Modelagem

3.5.1 Simulação

Uma simulação é um programa que representa o comportamento de um sistema real. No caso de sistemas computacionais, um *hardware* ou *software* pode ser simulado. Para se construir um programa de simulação relativo a um sistema em estudo, devem-se inicialmente especificar os componentes e sub-componentes: *hardware*, *software*, aplicativos e todos os mecanismos e dispositivos de interesse no caso. Em seguida se constrói um modelo da estrutura do sistema. Após, transforma-se o modelo num programa, representando-se a ocorrência de eventos consecutivos, tais como: atualizações, transações, consultas, interrupções, retardos e esperas. Nele se colocam as características pertinentes: taxas, prioridades, comprimento de mensagens, origens e destinos entre outras. A verificação é a etapa em que se analisa se o programa implementa (representa) corretamente o modelo. E na validação se analisa se as suposições feitas no modelo correspondem à realidade (SANT'ANNA, 2004).

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

Uma simulação pode ser sequencial ou distribuída. Numa simulação sequencial, os eventos não podem ser executados eficientemente em sistemas paralelos ou multiprocessadores,

porque existem duas estruturas globais que são compartilhadas por todos processos: o relógio da simulação e a lista de eventos.

Quanto à escolha da linguagem de simulação, pode ser usada uma linguagem de propósito geral, uma extensão dessa linguagem de propósito geral adaptada para simulação, uma linguagem específica para simulação ou ainda um pacote de simulação. Outras considerações comumente feitas em simulações são verificar se a simulação atingiu um estado permanente, o tempo total da implementação da simulação, geração de números aleatórios e escolha de sementes.

3.5.2 Solução Analítica (Teoria de Filas)

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

O uso de modelos matemáticos baseados em teoria de filas pode-se tornar difícil em sistemas complexos, mas apresenta vantagens como baixo custo e tempo para a avaliação. Algumas características que devem ser especificadas para avaliação com teoria de filas são:

- processo de chegada (muitas vezes se considera como um processo de Poisson, onde os tempos entre chegadas têm distribuição exponencial).
- Tempo de serviço: muitas vezes considerado como tendo distribuição exponencial;
- número de servidores;
- capacidade do sistema: máximo número de clientes que podem permanecer dentro do sistema; tamanho da população (de clientes);
- disciplina de serviço, sendo alguns exemplos: FCFS (*First Come, First Served*); LCFS-PR (*First Come, First Served with Preempt and Resume*); RR (*Round-Robin*).

3.6. Considerações Finais

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

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

Combinações de diversas técnicas em avaliação de desempenho são normalmente utilizados. Por exemplo, pode-se considerar um modelo analítico para determinar a faixa de parâmetros juntamente com um simulador para analisar o desempenho na faixa considerada ou validação de técnica analítica com uso de simulação ou medições reais (monitoramento).

Com esse capítulo foram introduzidos conceitos básicos envolvidos na análise de desempenho e na simulação.

No próximo capítulo serão apresentados os conceitos relativos aos servidores *web* como características de funcionamento, arquitetura dos servidores e tipos de requisições atendidas.

Servidores Web

4.1. Considerações Iniciais

Com o passar do tempo a *World Wide Web* tem sofrido um grande crescimento. Não são apenas milhões de novos navegando na *Web* a cada dia, também há centenas de novos *web sites* que são adicionados diariamente (WELLS, 2001). Logo, há o crescimento no número de servidores (i.e., HTTP) *Web*. Com isso, os vendedores de servidores *web* tem orgulho em enaltecer os atributos de seus produtos, e características de fabricação baseadas em teorias em como tornar o servidor *web* rápido. Entretanto, essas virtudes e teorias são geralmente baseadas em evidências erradas, instinto, ou limitadas evidências empíricas que são pouco úteis (SLOTHOUBER, 1995).

Por esse fato que é necessário conhecer “todos os tipos de aplicações possíveis” que um servidor *web* pode receber e características importantes. A partir daí será possível anteceder as possíveis requisições e obter o melhor rendimento possível do *hardware*.

Ao longo deste capítulo serão mostrados: as características técnicas do *hardware*, em específico para suprir necessidades de requisições HTTP, tipos de arquiteturas de servidores *web*; e características de aplicações próprias para os servidores *web*.

4.2. Funções do Servidor Web

Com a *Web* se tornando a interface padrão para acessar serviços remotos de sistemas de informações, centro de hospedagem de dados e provedor de serviços de aplicação há uma crescente demanda no lado dos servidores *web* que estão sofrendo cada vez mais com este aumento de requisições (CARDELLINI, 2002).

Graças à complexidade da infra-estrutura da *Web*, problemas com a performance podem surgir em diversos pontos durante a interação com os servidores *web*. Por exemplo, eles podem

ocorrer na rede devido ao congestionamento dos roteadores da Internet, tanto como na *Web* e nos servidores de banco de dados, ou pior ainda, por causa da capacidade limitada de atendimento ou surgimento de uma requisição inesperada. A solução do problema das redes tem sido mais facilmente resolvido, tendo em vista a velocidade com que as capacidades de banda crescem, já os problemas relacionados aos servidores *web* tornam-se mais difíceis pela menor velocidade de desenvolvimento de seus componentes (COFFMAN, 2001). O aumento da capacidade das redes por si só não resolve o problema de QoS, mas comumente transfere o problema para dentro dos servidores *web*, uma vez que eles se tornam o “gargalo”.

A seguir serão apresentados alguns tipos de documentos suportados pelos servidores *web* e sua particularidades.

4.3. Arquitetura dos servidores web

A WWW pode ser considerada, em uma visão de alto nível, sendo composta por três elementos principais: os *browsers*, a rede e os servidores (TEIXEIRA, 2004). Tais elementos se relacionam através do modelo cliente-servidor (TANENBAUM, 2003). Sendo que, esse modelo pode ser ordenado em diferentes arquiteturas. Na figura 4.1 são mostradas as ramificações possíveis que um sistema distribuído pode utilizar.

Figura 4.1 - Tipos de arquitetura de servidores web

As diferentes arquiteturas dos servidores *web* são motivadas especialmente pelo *upgrade* necessário pelos servidores *web* devido o aumento de demanda. No início, os servidores *web* eram constituídos de apenas uma máquina que, ao não suportar a demanda era substituída por outra máquina mais veloz e com maior armazenamento, técnica conhecida como *hardware scale-up* (DEVLIN, 1999). Outra opção utilizada era melhorar o software, técnica conhecida como *software scale-up* (BANGA et al., 1998). Essas técnicas, apesar de melhorar o desempenho por um certo tempo, tornaram-se impraticáveis com o aumento sofrido pela Internet. A saída encontrada foi a utilização de mais de um nó em cada servidor *web*, ou servidores *web* distribuídos para fazer o compartilhamento da carga recebida. Os sistemas *web* distribuídos, como pode ser visto na figura 4.1, podem tanto ser subdivididos em *global scale-out* e *local scale-out*. Em *global scale-out* os nós são separados em distintas localidades geográficas, já em *local scale-out* esses nós servidores estão residentes em uma única rede.

Apesar das diferenças estruturais, os componentes internos e o funcionamento dos servidores *web* não diferem, ou seja, as requisições são atendidas da mesma maneira em todas as arquiteturas. A diferença é observada na variação da capacidade total entre diferentes arquiteturas e servidores *web*.

Apesar das requisições não serem tratadas de maneira diferente por diferentes arquiteturas, há diferenças entre requisições como as requisições dinâmicas e estáticas.

4.3.1 Requisições Estáticas

As páginas estáticas têm a característica de apresentar textos planos e acompanhados de imagens e no máximo algum recurso multimídia como vídeos e áudios. Essas páginas são desenvolvidas através da linguagem HTML (*HyperText Markup Language*). O maior recurso de tais páginas fica por conta dos *links* para outras páginas *Web*.

Até meados da década de 90, a *Web* era constituída apenas por documentos hipertexto interligados. Devido à baixa atualização no conteúdo dos documentos, as páginas estáticas eram uma boa solução (SILVA, 2005).

Figura 4.2 - Esquema cliente-servidor nas requisições estáticas

As requisições estáticas seguem o esquema mostrado na figura 4.2, 1) cliente faz requisição de um documento utilizando um *browser*; 2) a requisição é enviada através do *middleware* (Internet); 3) o local onde o documento se encontra (servidor *web*) responde a requisição enviando o documento. Para o envio da requisição é utilizado o protocolo URL. Através de um *browser* o cliente requisita um documento. O *browser*, por sua vez, produz uma requisição HTTP ao servidor, que analisa o pedido e se possível envia as informações no padrão HTML ao *browser* cliente, que apresenta as informações ao usuário.

Nesse tipo de requisição a interação fica por conta do pedido e do próprio documento retornado. Apesar da baixa interatividade, esse tipo de documento ainda é muito popular (SILVA, 2005).

4.3.2 Requisições Dinâmicas

A característica das páginas dinâmicas é que elas são geradas em tempo de processamento da requisição. Essas páginas possuem efeitos especiais como: controle de formulários e janelas entre outros recursos. Entretanto, para a geração de tais páginas é necessário utilizar outras linguagens de programação, além do HTML.

Páginas dinâmicas podem ser obtidas através de aplicativos rodando no cliente ou no servidor. Algumas tecnologias utilizadas no lado cliente são: Java, ActiveX e Dynamic HTML, esses recursos são executados diretamente pelo navegador. Nessa abordagem, há a vantagem de reduzir o tempo de resposta e evitar o uso excessivo de recursos de rede (MENASCÉ;

ALMEIDA, 2003). Outra abordagem possível é gerar as páginas dinâmicas diretamente no servidor, que pode ocasionar um consumo do recurso do processador do servidor *web*, gerando uma sobrecarga no mesmo caso as aplicações forem muito complexas. ASP (*Active Server Pages*), PHP (*Hypertext Preprocessor*) e JSP (*Java Server Pages*) são alguns exemplos de tecnologias de criação de páginas dinâmicas no servidor.

As requisições dinâmicas ocorridas nos servidores seguem o esquema mostrado na figura 4.3. A sequência é: 1) o cliente faz uma requisição de um documento através de uma requisição HTTP; 2) a requisição é enviada através da Internet até o servidor *web*, 3) o servidor *web* verifica que se trata de uma requisição dinâmica e a envia para ser processada; 4) a requisição é processada por um sistema de processamento de transação e faz o caminho de volta até o cliente. No passo 4 é possível haver a integração com um sistema de gerência de banco de dados.

O protocolo HTTP e a linguagem HTML fornecem recursos dinâmicos como o protocolo CGI (*Common Gateway Interface*) que permite a execução de programas e marcadores para definição de formulários (SANT'ANNA, 2004). Dessa forma, os servidores são configurados para reconhecer nomes de arquivos como chamadas para aplicativos que deverão ser executados. Toda a comunicação continuará sendo realizada em formato HTML, garantindo nenhum vínculo a plataforma. O que o CGI faz é transferir o pedido de execução de uma aplicação para o programa apropriado, agindo como um tradutor entre o código HTML e os requisitos de aplicações.

O CGI nasceu em 1995 oferecendo uma interação maior sobre o modelo cliente-servidor utilizando 3 níveis (ou camadas), esses permitem ao servidor *web* “rotear” o conteúdo de formulários HTML para aplicações de servidores *back-end*.

Figura 4.3 Esquema cliente-servidor das requisições dinâmicas

A partir do CGI foi possível ampliar as aplicações na Internet, realizando atualizações, inserções e deleções em bases de dados, o que tornou o comércio eletrônico possível. Pelo fato do protocolo CGI não manter informações de um formulário para o seguinte (*stateless*) foram criados os *cookies*, pequenas porções de informações que ficam nos clientes. Essas informações ficam a cargo da aceitação do cliente mediante o *browser*.

4.4. Sessões HTTP

Com o incentivo dado pelas requisições dinâmicas e segurança no atendimento das aplicações, houve o crescimento da utilização da Internet como meio de compra e venda de produtos. Os clientes que fazem compras na Internet são atraídos pela velocidade e conforto em realizar compras sem sair de casa fatos que contribuem com o crescimento do número de usuários que fazem uso desse recurso. Entretanto, o aumento de demanda nos servidores *web* causados pelo comércio eletrônico pode ser extremo e comprometer a qualidade dos serviços (garantia de atendimento) disponíveis no servidor web (WEI et al, 2003).

Tipicamente, um acesso ao servidor web ocorre na forma de uma sessão, sendo essa, constituída de várias requisições individuais de um mesmo cliente (Arlitt, 2000). Por exemplo, um pedido de um cliente a um site comercial, pode envolver uma requisição para selecionar o produto, outra para prover informações necessárias, uma para estabelecer um acordo de pagamento e finalmente uma última requisição para receber uma confirmação (MOURÃO, 2005). Para garantir o atendimento de todas as requisições pertencentes a uma transação

comercial é necessária uma qualidade de serviço em nível de aplicação. Ou seja, uma transação de sucesso é aquela que atende toda a sequência de requisições.

4.5. Estrutura interna dos servidores web

Além da importância do meio de comunicação da requisição (Internet), como já foi dito anteriormente, os componentes internos de um servidor *web* são diretos responsáveis pelo desempenho final do servidor *web*.

A seguir serão explicados o funcionamento e a importância de alguns componentes que serão simulados neste projeto.

4.5.1 Tratamento de Requisições TCP

Após ser estabelecida a conexão TCP entre cliente e servidor, as requisições HTTP são redirecionadas pelo *daemon*¹¹TCP para uma fila onde essas requisições ficam aguardando serem atendidas.

4.5.2 Pool de Threads HTTP

Os servidores *web* primitivos utilizavam cópias de um novo processo para atender cada uma das requisições HTTP (figura 4.5). Entretanto, a sobrecarga gerada tornou-se um problema. Como solução, foi criado um conjunto de processos que eram previamente gerados, um desses processos era o processo mestre que gerenciava os pedidos e os demais processos.

¹¹ Programa em execução num computador servidor e está sempre pronto para receber solicitações de outros programas, executar determinada ação e retornar a resposta adequada.

Figura 4.4 - Um servidor HTTP de um processo por conexão com um processo mestre.

Figura 4.5 - Um servidor de um único processo com múltiplas threads.

Como tempo, foi utilizado apenas um processo por servidor *web*, mas esse processo era composto por múltiplas *threads*, como pode ser visto na figura 4.5 (TANENBAUM, 1995). Nessa abordagem, como visto, as *threads* ao estarem prontas utilizam o método *accept* para tratar uma requisição que fica em uma fila de espera de requisições TCP. Essa abordagem é a mais usada atualmente.

4.5.3 Envio de Arquivos Requisitados

Após a requisição ser interpretada por uma *thread*, o arquivo é procurado tanto no sistema de arquivos quanto em memória. Caso encontrado esse documento é armazenado em um

buffer de rede, se disponível, para ser enviado ao cliente. É importante dizer que se o arquivo for maior que o *buffer* a ser utilizado, deve ser fracionado com o tamanho do *buffer* o número de vezes necessárias e todas as partes devem ser enviadas pelo mesmo *buffer*, dessa forma alocando um *buffer* até o término do envio.

É nessa fase que as requisições dinâmicas são redirecionadas ao *script engine* para execução dos scripts. Os *scripts engines* são servidos por *threads* e tem o desempenho atrelado à implementação do servidor *web* e do sistema operacional.

Após o processamento no *script engine*, as requisições sofrem o mesmo procedimento de carregamento de *buffers* que as requisições estáticas.

4.5.4 Descarga de Buffers de Saída

Após estar com os dados de saída nos *buffers* é necessário fazer o esvaziamento dos mesmos. Esse esvaziamento é feito através de um controlador de I/O que envia o conteúdo dos buffers pela conexão TCP aberta como cliente. Para esvaziar os *buffers* o controlador de I/O pode utilizar, por exemplo, a política de *round-robin*.

A unidade de transmissão em redes baseadas no TCP/IP é o MSS (*Maximal Segment Size*), ou seja, o tamanho máximo de dados que o TCP consegue enviar para o cliente em um segmento. Em outras palavras, o conteúdo do *buffer* é particionado em porções de tamanho MSS e são enviados seguindo o mecanismo de janelas (SANT'ANNA, 2004).

4.6. Considerações Finais

Neste capítulo foram mostradas as interações e um pouco do funcionamento de elementos internos de um servidor *web*, além de mostrar as estruturas externas que os servidores *web* podem assumir.

Também foram mostrados alguns modelos de servidores *web* encontrados na literatura, sendo que alguns desses modelos, que tentam mapear e modelar os passos que as requisições sofrem até retornar aos clientes servirão de base para esse projeto.

No próximo capítulo será apresentado o projeto a ser desenvolvido, o plano de trabalho e o cronograma a ser seguido.

Modelo de Servidor Web com Quatro Módulos de Atendimento de Requisições (SWMAR)

5.1. Considerações Iniciais

Neste capítulo serão apresentados trabalhos relacionados a modelagem de servidores *web*, e fundamentos utilizados para a construção do SWMAR. Também serão apresentados: a biblioteca utilizada para controlar as redes de fila (SimPack) e o gerador de *logs* (W4Gen) utilizado como carga inicial para o modelo desenvolvido. Características de implementação do SWMAR também são apresentadas. Esse capítulo está organizado da seguinte maneira, na seção 5.2 são apresentados os trabalhos relacionados a análise e modelagem de servidores *web*. A geração de *logs* (W4Gen) e a biblioteca de manipulação de redes de fila (SimPack) são apresentadas nas seções 5.3 e 5.4, nessa ordem. E na seção 5.5 são apresentadas características e particularidades de implementação do SWMAR, modelo e simulação de um servidor *web* desenvolvido nesta dissertação de mestrado.

5.2. Trabalhos Relacionados

Apesar da grande necessidade de se avaliar os servidores *web*, visando oferecer aos usuários um melhor serviço ou qualidade de serviço (*QoS*), poucos trabalhos são encontrados na literatura descrevendo modelagens de servidores *web*. Dentre esses trabalhos poucos apresentam abstrações de componentes internas, muitos são baseados em *benchmarks* de servidores *web* e/ou análise dos “gargalos” e nem todos utilizam cargas sintéticas para carga inicial, utilizam distribuições estatísticas para esse papel. A seguir são apresentados alguns destes trabalhos separados em dois grupos: os que modelam servidores *web* e os que analisam o desempenho de um servidor *web*.

5.2.1 Modelos de Servidores Web

Existem diversos modelos de servidores *web* na literatura (ABDELZAHER; LU, 2000), (ALMEIDA, 2000), (ROBERTSOON, 2003), (WELLS, 2001), (GANDHI, 2002), (REESER, 2000) e (DILLEY, 2000). Entretanto, nem todos possuem características de modelar todas as etapas e estruturas de um servidor *web*, ou seja, que modele desde uma requisição HTTP, passando por todo o processo de análise e tratamento dessa requisição até o envio da resposta ao cliente. O mais comum é tratar os servidores *web* como “caixas-pretas”, utilizando uma visão em alto nível e tirando o peso que o servidor *web* possui no desempenho total da Internet.

As modelagens de servidores *web* sempre focam na análise do potencial de uma ou mais máquinas, considerando a mudança de configurações tanto em *hardware* quanto *software*. Mas, a forma como os modelos são desenhados é que mostra o quanto esses modelos são aplicáveis no ambiente real.

Geralmente esses modelos são testados com o auxílio de uma carga que pode vir tanto de *logs* gerados em servidores reais quanto através de programas de *benchmark* como: WebStone, SPECWeb, S-clients. Esses programas de *benchmark* geram o que é conhecido como carga sintética.

Os vários trabalhos citados anteriormente estão mais relacionados à análise comparativa de vários servidores *web* ou até mesmo aos resultados de alterações em características de *hardware* e/ou *software* de um servidor existente. Essas abordagens, apesar de importantes para o conhecimento aprofundado das características dos servidores *web*, não são de grande utilidade ao se simular um ambiente do qual os servidores *web* fazem parte. Para essa finalidade, existem alguns modelos que se aproximam em maior ou menor grau de um servidor *web* real.

Em (DILLEY et al., 2000) é apresentado um modelo em alto nível de um servidor HTTP. Esse modelo recebe várias requisições de *browser* clientes. Essas requisições eram atendidas pelo *daemon* HTTP (*httpd*) e são despachadas a um processo do *pool* de *server*, o número de processos no *pool* pode variar para análise dos resultados. Esse modelo também prevê a requisição de páginas dinâmicas. Nesse processo, é criado um processo filho que executará o *script* CGI e retorna a informação requerida ao *server pool*. No final da requisição tanto de páginas dinâmicas quanto estáticas são retornadas ao cliente.

As deficiências analisadas nesse trabalho são: a falta de detalhes ao modelar o módulo `httpd` (considerar os componentes internos) e a maneira como foi tratado o meio de comunicação (não foram considerados eventuais problemas com a rede).

Em van der MEI et al, (2001) e também em SANT'ANNA et al, (2004), devido o trabalho ser baseado no trabalho de van der MEI et al, (2001), há a preocupação em modelar algumas abstrações de componentes internas e o meio de comunicação. Em van der MEI, foi feita a divisão dos processos que ocorrem dentro dos servidores web em três fases:

- Estabelecimentos de conexão TCP – são modeladas desde a requisição entre cliente e servidor até o envio da requisição HTTP.
- Processamentos do HTTP – nessa fase são modeladas da aceitação das requisições até o carregamento dos *buffers* de saída.
- Processamento de I/O – a modelagem aqui é relacionada ao esvaziamento dos *buffers* e a entrega do resultado requerido ao cliente.

Apesar de todo o cuidado tomado para que o modelo se pareça com um servidor real, o protocolo utilizado foi o HTTP 1.0 e não há tratamento de requisições dinâmicas.

SANT'ANNA (2004) baseou-se no trabalho de van der MEI (2001), mantendo a mesma estrutura das fases anteriormente mostradas, fazendo a atualização do protocolo HTTP para a versão 1.1 e inserindo o tratamento de requisições dinâmicas através de um *script engine*.

5.2.2 Análise de Servidores Web

Em (Hu et al., 1999) é estudado o comportamento o servidor APACHE executando sobre o sistema operacional AIX da IBM. Nesse estudo é feita a identificação dos “gargalos” e são fornecidas informações detalhadas a respeito dos eventos ocorrendo no núcleo do sistema operacional. As cargas sintéticas são geradas pelas ferramentas SPECWeb e Webstone. O trabalho fornece dados sobre as porcentagens de tempo gastas executando em nível de usuário, em chamadas de sistema no núcleo e em manipulação de interrupções. Com relação aos gargalos, a conclusão no trabalho é que para sistemas com memória principal (RAM) de pequeno tamanho, o desempenho é limitado pela largura de banda do disco. Para sistemas com memória principal de tamanho razoavelmente grande, os gargalos são os *softwares* que processam os

protocolos TCP/IP e o manipulador de interrupções da rede. O comportamento é semelhante para os sistemas uniprocessador e SMP (*Symetric Multiprocessor*). Após identificar “gargalos”, são propostas e implementadas diversas técnicas, com as quais se consegue um aumento de cerca de 60% no desempenho do servidor, conforme relatado.

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

(ALMEIDA et al. 2000) descrevem a concepção e implementação de uma ferramenta para gerar uma carga sintética. A carga de trabalho utiliza uma distribuição de tamanho de arquivos de cauda pesada para representar a característica da carga de um servidor *web*, que deve ser capaz de manipular concorrentemente algumas requisições para grandes arquivos (por exemplo, áudio e vídeo) e um grande número de requisições para documentos pequenos contendo texto ou imagens. Nesse estudo é verificado que em um servidor *web* saturado por requisições, 90% do tempo para manipular essas requisições HTTP é gasto no *kernel*. Manter as conexões abertas, conforme requerido pelo protocolo TCP, provoca um aumento de fator dois no tempo decorrido para atender uma requisição HTTP. Os resultados do estudo mostram o importante papel da implementação do sistema operacional e do protocolo de rede para determinação do desempenho.

(WALLACE & JR. 1997) apresentam um estudo que mostra algumas formas simples para instrumentar um servidor *web*, usando recursos e/ou mecanismos existentes, já implementados em sistemas de gerência da rede ou no sistema operacional, para obter

estatísticas. Esses dados coletados e analisados, são usados em alguns modelos simples e genéricos de dimensionamento para componentes do servidor *web*, descritos pelos autores. Os modelos são genéricos de dimensionamento para componentes do servidor *web* descritos pelos autores. Os modelos são centrados em quatro “gargalos” comumente considerados: rede, memória, disco e CPU. Segundo os autores, o *throughput* de rede pode ser descartado já que isso não é, em geral, uma preocupação do servidor *web* em si, pois a largura de banda de rede pode ser aumentada, configurando o servidor *web* com mais ,ou mais rápidas, placas de rede.

5.3. W4Gen

O W4Gen é resultado do estudo de (SILVA, 2006) do comportamento de vários *logs* de servidores *web*. No trabalho foram utilizados *logs* disponibilizados na internet e *logs* que já foram objeto de estudo como: Cdmil (2005), Tum (2005), Copi (2005), Stanford (2005), Veenet (2005) e Connectmed (2005). Além desses, foram utilizados os logs do CISC (Centro de Informática de São Carlos) e Prefeitura de Londrina e do Br10, estes foram obtidos através de intermediações com pessoas responsáveis pela administração de servidores *web* da instituição e da empresa.

Esses *logs* foram classificados em três categorias: acadêmico, notícia/informativo e tradicional. Um site acadêmico possui registros de usuários, serviços de biblioteca por meio de geração de páginas dinâmicas, áreas estatísticas com informações da instituição, além das páginas pessoais de professores e alunos. Os sites do tipo notícia/informativo incluem informações periodicamente atualizadas, como temperatura do dia, dados financeiros, notícias da região ou do mundo, uma área restrita aos usuários cadastrados, propagandas por meio de *banners*, *download* de arquivos, entre outras. Já um site tradicional possui poucas atualizações e utiliza muitos recursos estáticos. Essa última categoria inclui páginas pessoais e de empresas de pequeno porte cujo objetivo é simplesmente apresentar o seu perfil e os produtos aos clientes.

Através dos *logs* foi possível fazer análises do intervalo de chegada dos diferentes tipos de requisições¹² e foram selecionadas distribuições que mais se aproximassem do intervalo de chegada para esses tipos de requisições. Essas distribuições são: Weibull, Gamma e Lognormal.

¹² Arquivos que foram requisitados: html, imagem, arquivos dinâmicos, textos, documentos, scripts, áudio, binário e vídeo.

Como resultado desse estudo o W4Gen é capaz de gerar *logs* baseados nos três tipos definidos com intervalos de chegadas e nos tipos de requisições e tamanho dos arquivos requeridos baseados nos *logs* estudados. A seguir são apresentadas as possíveis configurações para a geração da carga de trabalho das seções 5.3.1 até 5.3.5.

5.3.1 Código de Resposta

O W4Gen possui uma interface gráfica que permite ao usuário configurar o código de resposta (código de *status*) retornado pelos servidores web. Há três técnicas diferentes para gerar o código de resposta. São elas:

Usar somente requisição do tipo 200¹³. Ativando a caixa de checagem, o W4Gen não utiliza nenhuma distribuição ou qualquer informação para gerar os códigos de respostas. Ele, simplesmente, acrescentará o código 200 a todas as requisições (Figura 5.1).

Figura 5.1– Geração de requisições com código 200 – W4Gen.

Outra forma de gerar código de resposta é segundo uma distribuição. No caso do primeiro item não estar selecionado, o W4Gen permite ao usuário gerar os códigos de resposta segundo a distribuição geométrica. Neste caso, é preciso definir o valor do parâmetro que configura a função de distribuição. A barra de exibida na figura 5.2 permite essa modificação.

¹³ Requisições com sucesso.

Figura 5.2 - Geração de código de status baseado em distribuição – W4Gen.

Como resultado da geração de código de resposta por distribuição são gerados quatro tipos diferentes de códigos: 2xx, 3xx, 4xx e 5xx, que são distribuídos no arquivo de *log* gerado pelo W4Gen.

5.3.2 Classe de Objeto

Há a opção de customizar os objetos de acordo com as necessidades de cada usuário. No W4Gen são disponibilizadas duas técnicas para gerar os tipos de objetos. São elas:

Segundo Distribuição de Probabilidade

Ao usar a distribuição de probabilidade, o W4Gen permite gerar as classes de objetos segundo a distribuição geométrica. Para isso, é preciso definir o valor do parâmetro que configura a função de distribuição. A barra de rolagem que aparece na figura 5.3 permite modificar esse parâmetro.

Figura 5.3 – Geração de classes objetos segundo distribuição geométrica.

Segundo Valores de Porcentagem

Outra forma de gerar os objetos é segundo valores de porcentagem. O W4Gen permite gerar as classes de objetos segundo os valores de porcentagem. Para isso, o usuário deve configurar a porcentagem de cada um dos nove tipos diferentes de objeto: Imagem, HTML, Dinâmico, Texto, Documento, Script Cliente/Animação, Áudio, Binário/Compactado e Vídeo. Esses valores são modificados através da barra de rolagem, como mostra a figura 5.4. A sequência de requisições segue o comportamento de uma função de números aleatórios.

A interface de configuração do W4Gen, intitulada 'Objeto', apresenta duas opções de geração de carga: 'Gerar segundo uma Distribuição' (desselecionada) e 'Gerar segundo valores de Porcentagem' (selecionada). Abaixo, a seção 'Porcentagem dos Códigos de Status' contém nove controles deslizantes para ajustar a porcentagem de cada tipo de objeto. Os valores atuais são: Imagem: 53,62%, Script/Animação: 0,03%, Html: 39,77%, Áudio: 0,02%, Dinâmico: 0,80%, Binário/Compactado: 5,14%, Texto: 0,04%, Vídeo: 0,08%, Documento: 0,50%. O total acumulado é exibido como 100,00%.

Objeto	Porcentagem
Imagem	53,62%
Script/Animação	0,03%
Html	39,77%
Áudio	0,02%
Dinâmico	0,80%
Binário/Compactado	5,14%
Texto	0,04%
Vídeo	0,08%
Documento	0,50%
Total	100,00%

Figura 5.4 – Geração de classes de objetos segundo valores de interface.

5.3.3 Intervalo de Chegada

O usuário pode ainda escolher os intervalos de chegada. Esta funcionalidade é atingida através de funções de distribuições para cada tipo de objeto. Com isso o usuário pode escolher as características que mais se adaptam as suas necessidades. Além disso é possível fazer uma sobrecarga no servidor *web*, que aumenta o número de requisições por tempo.

É possível escolher doze diferentes distribuições para a geração das classes do objetos. Conforme ilustrado na figura 5.5.

Intervalo de Chegada			
Imagem		Html	
Weibull	Forma: 0,30 Escala: 7,99	Weibull	Forma: 0,57 Escala: 37,50
Dinâmico		Texto	
Gamma	Forma: 0,14 Escala: 74.988,00	Weibull	Forma: 0,55 Escala: 600,00
Documento		Script/Animação	
Gamma	Forma: 0,22 Escala: 43.377,00	Gamma	Forma: 0,14 Escala: 428.528,00
Áudio		Binário/Compactado	
Lognormal	Média: 2,03 Desvio: 3,25	Gamma	Forma: 0,07 Escala: 268.705,00
Vídeo		Sobrecarga	
Gamma	Forma: 0,23 Escala: 154.645,00	Valor: 1	

Figura 5.5 – Intervalo de chegada das classes.

5.3.4 Tamanho de Objeto

O tamanho do objeto (figura 5.6) segue o mesmo princípio do intervalo de chegada, onde o usuário utiliza distribuições para gerar o tamanho dos arquivos. Além de escolher as funções é preciso definir os valores de seus parâmetros. A mudança desses valores pode distorcer a realidade da *Web* e reproduzir um comportamento diferente, gerando conclusões falsas na análise, já que a carga de trabalho influencia diretamente no desempenho global do sistema (MENASCÉ; ALMEIDA, 2003).

Tamanho do Objeto			
Imagem		Html	
Weibull	Forma: 0,78 Escala: 14.570,00	Lognormal	Média: 8,30 Desvio: 1,53
Dinâmico		Texto	
Weibull	Forma: 0,89 Escala: 1.543,00	Weibull	Forma: 0,60 Escala: 8.498,00
Documento		Script/Animação	
Weibull	Forma: 0,49 Escala: 32.395,00	Weibull	Forma: 0,64 Escala: 5.326,00
Áudio		Binário/Compactado	
Weibull	Forma: 0,56 Escala: 26.920,00	Pareto Geral	Escala: 14.500,00 Forma: 0,60
Vídeo			
Lognormal	Média: 11,40 Desvio: 1,52		

Figura 5.6 – Tamanho de objeto das classes.

5.3.5 Arquivo Resultante

Com todas as opções de configuração apresentadas é possível gerar a carga sintética de duas maneiras diferentes (figura 5.7). Por intervalo de tempo em minutos ou por número de requisições da carga.

Figura 5.7 – Geração de carga de trabalho.

A figura 5.8 mostra um exemplo de um arquivo reproduzido pelo W4Gen. O arquivo contém quatro campos: *timestamp* da requisição, contendo o instante que a requisição chegou ao servidor *web*; tipo de objeto que foi requisitado; código de *status* retornado pelo servidor *web* e o tamanho do objeto requerido. Na figura 5.8, todas as requisições chegaram no mesmo instante, ou seja, no tempo zero. Observa-se também a presença de diversos tipos de objetos, tais como, imagem (*image*), texto (*text*), entre outros e dois grupos de código de resposta, a 2xx (*Successful*) e a 3xx (redirecionamento). Também se podem observar os tamanhos dos arquivos das requisições que em alguns casos é zero, o que significa que os arquivos são menores que 1KB.

```
0 text 3xx 0
0 script 3xx 0
0 html 2xx 23483
0 image 3xx 0
0 text 2xx 56
0 image 4xx 0
0 image 3xx 0
0 image 2xx 1056
0 video 4xx 0
0 image 2xx 560
0 image 2xx 2723
0 image 2xx 2620
0 image 2xx 1645
0 image 2xx 240
0 script 2xx 6271
0 html 2xx 15635
0 dynamic 2xx 14352
```

Figura 5.8 – Exemplo de carga sintética gerada pelo W4Gen.

5.4. Simpack

SimPack originalmente é uma coleção de ferramentas C++ (rotinas e programas) para simulação de computadores. O propósito do SimPack é prover a pessoas com uma “idéia inicial” simular um sistema. O propósito principal é que através de uma visão ou uma idéia se consiga gerar um programa de simulação. SimPack foi utilizado em diversos estudos e trabalhos como: BARBATO et al. (2005) e TEIXEIRA et al. (2004).

SimPack é um conjunto de ferramentas que suportam eventos discretos de simulação. SimPack era originalmente escrito na linguagem C, e atualmente existe a versão na linguagem, Java. Atualmente a versão C foi depreciada e só ha atualizações na versão Java. SimPack é disponibilizada via a licença Gnu Public License (GPL).

A biblioteca oferece 3 conjuntos de métodos:

- Escalonamento de eventos: Núcleo do SimPack lidam com a lista de eventos futuros
 - Sim.init – inicia uma simulação discreta;
 - Sim.schedule – escalona um evento para ocorrer no futuro;
 - Sim.next_event – requisita o próximo evento da lista de eventos futuros;
 - Sim.next_event_dup – requisita todos os próximos eventos, com o mesmo tempo, da lista de eventos futuros;
 - Sim.cancel_event – remove o primeiro evento com um item especificado pelo evento da lista de eventos futuros;
 - Sim.cancel_token – remove o primeiro evento com um item especificado pelo identificador *token*;
- Alocação de recursos – métodos que tratam com as filas e os servidores que a atendem
 - Sim.create_facility – cria uma fila e os servidores que a atendem
 - Sim.request – requisita um serviço a uma fila;
 - Sim.preempt – causa preempção em uma fila;
 - Sim.release – libera a fila para um evento que está no topo da fila;

- Estado da simulação e rastreo – retorna os dados relacionados a lista de eventos futuros, às filas
 - Sim.state – retorna os dados da simulação a qualquer ponto da simulação;
- Geradores aleatório de números – gera números aleatórios em diferentes funções de densidade
 - Sim.random – gera distribuição aleatório;
 - Sim.uniform – gera distribuição uniforme;
 - Sim.normal – gera distribuição de normal;
 - Sim.expntl – gera distribuição exponencial;
 - Sim.erlang – gera distribuição erlang;

Algumas peculiaridades da biblioteca são: o tipo de estrutura das requisições que pode ser LINKED ou HEAP (diferença relacionada a estrutura de dados utilizada pela biblioteca) e tipo de simulação síncrona (SYNC) e assíncrona (ASYNC). A simulação síncrona é feita utilizando os tempos de simulação através do relógio físico do computador, ou seja, o relógio do computador dispara o atendimento das requisições. Já o modo assíncrono as requisições são atendidas o tempo atual sempre é 0 (zero) e os demais tempos são calculados pelo decremento baseado no antigo tempo atual, ou seja, se a fila de requisições são 10, 20, 20 e 32 o SimPackj atualiza os tempos para 0, 10, 10 e 22.

O SimPack apresenta algumas limitações, dentre elas cabe ressaltar que. Alguns valores são fixos no arquivo *Const.java*. Por exemplo, o número de eventos suportados pela biblioteca é fixado em 100 eventos. Esse valor pode ser alterado para mais eventos mas, o novo valor escolhido pode ser limitado por: (1) pela quantidade de memória disponível na máquina onde a simulação é executada e (2) limitações da máquina virtual Java (JVM) que interpreta a simulação.

5.5. Modelo de Servidor Web com Quatro Módulos de Atendimento de Requisições (SWMAR)

Esta seção apresenta um modelo para servidor *web* baseado em redes de filas, representado uma abstração dos componentes internos de um servidor *web* dividida em quatro

módulos diferentes, cada módulo é responsável por uma etapa do atendimento da requisição. Neste modelo são levados em consideração: impactos de diferentes características de carga inicial (gerada pelo W4Gen) e configuração de *hardware* e *software* do servidor. O modelo baseia-se principalmente no trabalho de van der MEI et al., (2001). Para auxílio na simulação foi utilizado do SimPack para manipular as filas e as requisições. Como parâmetros do SimPack foram utilizados como estrutura de dados *LINKED* e tipo de simulação *ASYN*C, assíncrona.

No SWMAR são analisadas requisições dinâmicas e estáticas. O fluxos de transações no servidor *web* são descritos por um modelo de filas de séries, que consiste de quatro sub-modelos que serão aqui denominados módulos de 1 a 4. A forma como esses módulos se integram é apresentada na figura 5.9.

A seguir os módulos são apresentados individualmente de maneira detalhada.

Figura 5.9 – Fluxograma do SWMAR.

5.5.1 Módulo 1: Atendimento de Requisições HTTP

Este módulo é responsável pela manipulação da requisição HTTP. Após o estabelecimento de conexão o pedido de transação está apto a ser filtrado e interpretado pelo subsistema HTTP, que consiste de uma fila de escuta HTTP servida por um ou mais *daemons* HTTP. O fluxo normal de atendimento por esse módulo é descrito na figura 5.10.

Cada *daemon* HTTP possui um *pool* de *threads* que é responsável pelo atendimento das requisições. Se uma das *threads* do *pool* de *threads* HTTP está disponível então a *thread* busca o arquivo requisitado e encaminha o pedido para o módulo seguinte, módulo 2.

Figura 5.10 – Esquema de atendimento no módulo 1.

Nesse módulo há apenas uma maneira de haver recusa de requisição, através do preenchimento total da fila de escuta HTTP. No caso de que todas as *threads* do *pool* de *threads* estejam sendo utilizadas a requisição fica esperando na fila até que se desocupe um das *threads*. Como valor padrão para o tamanho dessa fila é utilizado o valor padrão do APACHE de no máximo 511 requisições na fila e de 5 a 10 *threads* no *pool* de *threads*.

5.5.2 Módulo 2: Transferência de Arquivos à Memória Principal

Nesse módulo é feita a transferência do arquivo requisitado para a memória através do DMA. Ao utilizar o DMA o gerenciamento da transferência fica a cargo da controladora da placa-mãe e dessa forma não é utilizado o processador. Sendo assim, para calcular o tempo utilizado para executar a tarefa de transferência são utilizados o tamanho do arquivo requisitado e a velocidade do disco. É utilizada a velocidade do disco por se tratar do recurso computacional mais lento nessa operação e, portanto, ser o fator que limita a velocidade da transferência.

Nesse módulo podem ser inseridos parâmetros relativos a *software* e *hardware*. Os parâmetros de *software* são o tamanho de fila de requisição (*modulo2.length*) e o número de *threads* (*modulo2.threads*) que atendem as requisições em fila. Já os parâmetros de *hardware* são o tamanho da memória principal disponível e o tamanho de *swap* que é utilizado. Nesse módulo há a possibilidade de recusa de requisições por memória cheia. O seguinte cálculo é efetuado para analisar se a memória está cheia.

$$Memória_{Total} = Memória_{Principal} + Memória_{Swap}$$

$$Memória_{Disponível} = Memória_{Total} - Tamanho_{Arquivo}$$

Nesse cálculo são utilizados parâmetros de *hardware*. São eles *machine.memo.size* (tamanho de memória principal disponível para uso na transferência de arquivos) e *machine.disk.size* (tamanho de memória *swap* disponível para uso na transferência de arquivos). Se o valor da memória disponível for positivo então é efetuada a alocação da memória para a requisição. Caso contrário a requisição é recusada e retorna ao cliente.

Uma particularidade do SWMAR é que se pode utilizar o tamanho de fila infinito. Desta forma as requisições que chegam ao módulo 2 são atendidas desde que haja memória disponível. Nesse módulo o parâmetro de número de *threads* tem a função de simular o número de recursos que atende a operação de transferência de arquivos à memória. Por exemplo, se o sistemas contar com mais de um disco é possível que simule esse cenário utilizando mais de uma *thread*.

O fluxo da requisição no módulo 2 é apresentado na figura 5.11.

Figura 5.11 – Fluxo de requisições no módulo 2.

Após executar a transferência do arquivo para memória é possível 2 diferentes fluxos:

- 1) a requisição não é dinâmica e segue ao módulo 4 que faz o envio do arquivo até a interface de rede para ser entregue ao requisitante.
- 2) a requisição é do tipo dinâmica e necessita de um processamento que é executado pelo módulo 3 ou *Script Engine*.

5.5.3 Módulo 3: Script Engine – Execução de Requisições Dinâmicas

Esse módulo é responsável pela execução de requisições dinâmicas. Para simular o tempo de execução de cada requisição foi utilizada uma distribuição exponencial, que segundo (JAIN, 1991) é a distribuição mais indicada para simular o funcionamento do processador (recurso computacional utilizado nesse módulo) uma vez que, não possui interdependência entre as tarefas executadas (*Memoryless*), ou seja, não existe relação entre as tarefas executadas e, portanto, não há memória entre as tarefas. Na figura 5.12 é mostrado o fluxo das requisições do módulo 3.

Neste módulo também existe os parâmetros de tamanho de fila (*modulo3.length*) e de número de *threads* (*modulo3.threads*) que atendem esse módulo e seguem o mesmo princípio

dos parâmetros do módulo 2, onde o tamanho de fila pode ser adotado como infinito e o número de *threads* referente ao números de processos que atendem as requisições dinâmicas. Há variações de cenários para este módulo. Pode-se simular o *script engine* na mesma máquina onde o servidor *web* está operante ou é possível que o processamento de requisições dinâmicas ocorram em outra máquina.

Figura 5.12 – Fluxo de requisições no módulo 3.

5.5.4 Módulo 4: Envio de Dados

O módulo 4 é responsável pelo envio do arquivo requisitado para a placa de rede. Os componentes de *hardware* envolvidos são a memória principal, o barramento principal e o barramento PCI. Nesse módulo ocorre o mesmo problema de limitação de velocidade que ocorre no módulo 2. Aqui a velocidade máxima de transferência não é superior a menor velocidade dos componentes envolvidos, nesse caso a menor velocidade é a do barramento PCI.

Na figura 5.13 é exibido o fluxograma do módulo 4. A diferença neste módulo é a existência, além dos parâmetros de tamanho de fila e do número de *threads* que atendem essa fila, do parâmetro de tamanho de *buffer*, que impacta na operação de transferência de arquivos para a interface de rede.

Figura 5.13 – Fluxo de requisições do módulo 4.

Neste estágio o arquivo requisitado já está na memória e deve ser enviado para o *buffer* de dados e depois, ser enviado para a interface de rede, como mostrado na figura 5.14. Nesta operação é importante notar que o tamanho do arquivo pode ultrapassar o tamanho do *buffer* de dados. Sendo assim, deve haver um trabalho extra em dividir esse arquivo em tamanhos iguais ao tamanho do *buffer* e calcular o *checksum* destas partes para que o arquivo enviado seja exatamente o arquivo que foi requisitado. Essa operação de divisão é descrita a seguir.

Figura 5.14 – Funcionamento do esvaziamento de buffers.

Um fato importante é que ao eleger um *buffer* para enviar o arquivo, o mesmo deve ser enviado todo utilizando o mesmo *buffer*. Dessa forma, o *buffer* fica dedicado a um mesmo arquivo enquanto o mesmo está no sistema. O número de vezes que o arquivo precisa ser particionado é obtida por meio da equação apresentada na figura 5.14. Isto é, o número de envios necessários é igual ao topo da divisão do tamanho do arquivo pelo tamanho do *buffer*.

5.5.5 Estimativa e Tempos de Serviço

Talvez a maior dificuldade do projeto foi encontrar dados para serem aplicados no cálculo de tempo gasto em cada um dos módulos. Para que a simulação se baseia em dados reais foram pesquisados na literatura *benchmarks* de servidores *web* que analisassem separadamente o tempo em diferentes passos que as requisições atravessam. O que mais se aproximou desse resultado foi o trabalho de (ALMEIDA, 2000). Nesse trabalho é feita uma análise em uma máquina com um processador Intel Pentium 75MHz, 16 MB de memória RAM, uma interface de rede Ethernet de 10 Mbps e 36MB de memória swap. Uma instrumentação no servidor APACHE é realizada para que se obtenha tempos de diferentes passos que a requisição percorre. De acordo com esse trabalho há três etapas que, quando tem os tempos somados, são responsáveis pelo processamento da requisição. São essas etapas:

Manipulação da requisição

Nessa etapa as requisições são manipuladas e encaminhadas para o processamento. Esse tempo é iniciado após o estabelecimento com o usuário e termina quando a requisição foi tratada e está pronta para ser processada.

Processamento da requisição

É o tempo gasto com o processamento da requisição. Esse tempo envolve o tempo gasto com a leitura da URL¹⁴ e o tempo gasto para mover o arquivo requerido da memória principal ou do disco para a interface de rede.

Escrita em log

Para toda operação realizada é gerado um *log* com os dados relativos ao cliente e a requisição efetuada. Essa etapa é a que captura o tempo gasto para tal tarefa. O *log* é escrito logo após a transferência do arquivo para a interface de rede.

Toda a vez que uma requisição passa por um módulo é preciso calcular qual é o tempo gasto no mesmo. Para isso foi implementado um método que calcula esse tempo baseado nos parâmetros de *hardware*, do *software* (servidor *web*) e tamanho do arquivo que foi requisitado.

¹⁴ Uniform Resource Locator

Baseado nesses dados foi possível estimar os tempos para os quatro módulos propostos nesta dissertação. A seguir são apresentados os cálculos de tempo de serviço em cada módulo.

Módulo 1

Dos parâmetros iniciais utilizados para gerar uma simulação no SWMAR o CPU é o único recurso computacional utilizado, para gerar o tempo de utilização. Por se assemelhar muito com a etapa de manipulação de requisição de Almeida et al, (2000) foram utilizados os resultados do trabalho para gerar os tempos para a simulação.

Almeida et al, (2000) executa requisições divididas em três classes de requisições estáticas feitas por 30 clientes. A diferença entre essas classe é baseada no tamanho dos arquivos requisitados. Em seu trabalho foram obtidos os seguintes tempos:

	<i>Classe 1</i>	<i>Classe 2</i>	<i>Classe 3</i>
<i>Tamanho de arquivos</i>	Até 12,1KB	Entre 12,1KB e 2,3MB	Maiores 2.3MB
<i>Tempo (ms)</i>	6,20	21,80	23,45

Tabela 4– Classificação de tempos de acordo com as classes de objetos.

Entretanto, esses tempos têm fundamentos apenas se aplicados no mesmo processador utilizado no trabalho. O que limitaria muito a gama de testes a serem executados no SWMAR para resolver esse problema foi feito o cálculo do $Fator_{CPU}$ que é calculado utilizando uma medida da capacidade do processador realizado pelo *benchmark Dhrystone*. Nesse *benchmark* o processados utilizado no trabalho de (Almeida et al. 2000) pontuou 102Mips¹⁵ através dessa pontuação é feito o cálculo:

$$Fator_{CPU} = \frac{Mips_{Base}}{Mips_{Atual}}$$

Nesse cálculo o $Mips_{Base}$ possui o valor de 102 Mips e o valor de $Mips_{Atual}$ é um valor inserido pelo parâmetro *machine.cpu* com esse valor de $Fator_{CPU}$ e os tempos obtidos no trabalho de Almeida et al, (2000) é possível obter os tempos de utilização no módulo 1. Esses tempos são obtidos com os cálculos abaixo.

¹⁵ Milhões de instruções por segundo.

$$Tempo_{classe1} = 0,02345 \times Fator_{CPU}$$

$$Tempo_{classe2} = 0,02180 \times Fator_{CPU}$$

$$Tempo_{classe3} = 0,00620 \times Fator_{CPU}$$

Com esse cálculo é possível encontrar os tempos gastos no módulo 1 de acordo com o tamanho dos arquivos requeridos.

Módulo 2

Neste módulo é utilizado apenas o tempo de transferência do arquivo do disco para a memória principal. Como dito anteriormente, não foram considerados outros recursos de *hardware* porque essa operação utiliza o DMA. Para o cálculo desse tempo é utilizado o tempo de transferência de disco pois, apesar de ser mais lento que a memória é um limitante da transferência (BINKERT, 2003). O cálculo do tempo é dado por:

$$Tempo_{Módulo\ 2} = \frac{Tamanho_{Arquivo}}{Velocidade_{Disco}}$$

No cálculo, o tamanho do arquivo é o tamanho do arquivo requerido na requisição e a velocidade de disco é um parâmetro inserido pelo usuário em *machine.disk.speed*. O tempo de utilização do módulo 2 é dado em segundos.

Módulo 3

Em nenhum dos trabalhos analisados há uma medida baseada nas requisições dinâmicas. No entanto, em (Sant'Anna et al. 2004) é dito que em média esse tempo é de 2ms. Mas esse tempo pode variar muito dependendo do processador utilizado no sistema. Por se tratar de uma operação que não possui interdependência com outras operações foi utilizada uma distribuição exponencial que segundo (JAIN, 1991) modela bem a utilização de um CPU. Esse tempo foi encontrado segundo a fórmula abaixo:

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Um exemplo de distribuição exponencial é apresentado na figura 5.15. Nessa figura é possível verificar o comportamento da distribuição com diferentes valores de lamda.

Figura 5.15 – Exemplo de distribuição exponencial.

Módulo 4

Devido a característica de separação do arquivo em tamanhos de *buffers* para envio para a placa de rede é utilizado a CPU para fazer o particionamento do arquivo quanto o barramento de PCI para o envio das partes para a interface de rede. Esse módulo descreve a operação de processamento de requisição de Almeida et al, (2000). Em Almeida o tempo gasto para essa etapa é dado pela tabela.

	Classe 1	Classe 2	Classe 3
Tempo de CPU no Processamento de Requisição	0.01875	0.15056	2.23135

Tabela 5 – Tempos de utilização do CPU no processamento de requisições.

No trabalho de (Almeida et al. 2000) é dito que as requisições de classe 1 são menores que o tamanho do *buffer*. Logo, não há cálculo de *checksum*. Apenas gasto de processamento para envio do arquivo para o barramento PCI e o tempo gasto pelo barramento para levar o arquivo para o interface de rede. Com essa informação se obteve o valor do tempo gasto pelo CPU apenas para enviar um arquivo para a interface de rede e calcular o checksum. A partir

disso, foi possível calcular o tempo gasto para enviar múltiplas partes de um mesmo arquivo até a interface de rede segundo a fórmula abaixo:

$$Número_{partes} = \frac{Tamanho_{arquivo}}{Tamanho_{Buffer}}$$

$$Tempo_{Módulo4} = (Número_{partes} \times Tempo_{checksum} \times Fator_{CPU}) + \left(\frac{Número_{partes} \times Tamanho_{Buffer}}{Velocidade_{PCI}} \right)$$

Na fórmula, o topo da divisão entre o tamanho do arquivo e o tamanho do *buffer* encontra o número de partes que o arquivo será dividido, o tempo de *checksum* é obtido no trabalho de (Almeida et al. 2000) e tem o valor de 0.01875 segundos e o fator CPU compensará a diferença da simulação de diferentes CPUs, com esses dados é possível encontrar o tempo que o CPU gasta para dividir os arquivos. Além disso, é necessário obter o tempo gasto com a transferência das partes do arquivo para a interface de rede, para isso é utilizado o número de partes que o arquivo será dividido multiplicado pelo tamanho do *buffer* para encontrar o total de *bytes* que serão transferido dividido pela capacidade de transferência do barramento PCI.

5.5.6 Entrada de Dados

Para gerar a simulação é necessário introduzir alguns dados relativos a configurações de *software* (servidor *web*) e valores que são do *hardware*. Alguns destes parâmetros já foram apresentados durante o decorrer da dissertação. Esses valores são todos inseridos em um mesmo arquivo de propriedades chamado *Configuration.properties*. Nesse arquivo existem 15 parâmetros de entrada sendo nove para *software*:

```
modulo1.lenght=511
modulo1.threads=10
modulo2.lenght=infinity
modulo2.threads=1
modulo3.lenght=infinity
modulo3.threads=1
modulo4.lenght=infinity
modulo4.threads=128
modulo4.buffer-size=40960
```

Os parâmetros de *software* são na maioria de dois tipos: *length* que insere o valor da fila de requisições de cada módulo e o parâmetro *threads* que insere o número de *threads* que

atendem cada módulo. No módulo 1 esses valores são utilizados com os valores padrão do APACHE que são 511 espaços na fila de requisição e 10 *threads* no pool de *threads*. Nos módulos 2 e 3 há uma diferença no parâmetro de *length* que é “infinito”. O motivo de ser infinito é que garante que, uma vez que a requisição foi atendida no módulo 1, essa requisição não será rejeitada por falta de espaço na fila de requisições. Além do parâmetro *length* há o parâmetro *threads* que nos módulos 2 e 3 também é diferenciado porque há apenas um recurso computacional que atende esse módulos, o DMA, e porque apenas um operador que executa as requisições dinâmicas.

O módulo 4 possui, além dos parâmetros *length* e *threads* o parâmetro *buffer-size* que controla o tamanho do *buffer* que é utilizado para transferir os arquivos para a placa de rede.

Os parâmetros de *hardware* são aqueles que influenciam diretamente no quesito máquina a ser simulada. Esses parâmetros influenciam diretamente nos tempos gastos na execução de tarefas em cada um dos módulos. São eles:

```
machine.mips=102  
machine.pci.speed=104857600  
machine.disk.speed=16777216  
machine.memo.size=16777216  
machine.disk.size=37748736
```

O parâmetro *machine.mips* é relativo à CPU do computador e para representá-lo, é utilizada uma métrica MIPS que é comumente utilizada em *benchmark* para processadores. Neste projeto foi utilizado os resultado baseados no *benchmark Dhystone*.

Os parâmetros que envolvem a velocidade dos dispositivos são *machine.disk.speed*, *machine.memo.size* e *machine.pci.speed*. Essas velocidades foram obtidas baseadas nos dados disponibilizados pelos fabricantes dos respectivos dispositivos.

O último tipo de parâmetros que representa os tamanhos da memória principal e a memória *swap*. Esses valores influenciam no tamanho disponível para transferência de arquivos no módulo 2. Se esse recurso for totalmente utilizado gerará recusa de requisições. Toda vez que um arquivo passa pelo módulo 2 ele, necessariamente, precisa ser carregado para a memória que só é liberada após o envio do arquivo (efetuado pelo módulo 4). Caso o arquivo requisitado não possa ser carregado para a memória por falta de espaço então a requisição é recusada (retorna ao cliente com erro).

5.5.7 Saída de Dados

Como resultado da simulação é gerado um arquivo com os dados resultantes. Para o arquivo de relatório foi escolhido o padrão CSV¹⁶ por facilitar a importação e manipulação dos dados. Nesse arquivo há o resultado da simulação gerado pelo SimPackj e estatísticas relativas a requisições que foram atendidas e recusadas. A estrutura do arquivo de resposta será exibida a seguir.

No início do arquivo há um relatório de simulação que é gerado pela biblioteca Simpackj. Nessa parte do arquivo de resposta encontra-se:

- O tempo total de simulação;
- Total de requisições utilizadas no sistema;
- Total de requisições atendidas;
- Utilização do sistema;
- Taxa de chegada;
- *throughput*;
- Número médio de requisições no sistema;
- Tempo médio de permanência para cada requisição;
- Porcentagem de utilização de cada módulo;

Outras informações que são relevantes para a compreensão do funcionamento do ciclo de simulação foram adicionadas à esse relatório. Dentre elas pode-se citar a porcentagem de requisições que foram recusadas e que tiveram sucesso, e uma relação com todas as requisições que foram atendidas e dados dessas requisições. Tais dados são:

- Número que representa a ordem que a requisição foi atendida;
- dados da requisição como: tempo, tipo de requisição e tamanho do arquivo;
- tempo de utilização;

Também existem dados quanto as requisições que foram recusadas, por exemplo o número de recusas e a porcentagem de recusas nas seguinte ocasiões:

¹⁶ Comma Separed Values

- Fila cheia do módulo 1;
- Fila cheia do módulo 2;
- memória cheia;
- Fila cheia do módulo 3;
- Falta de *buffers*;

Além dos arquivos no formato apresentado são gerados arquivos com as médias da simulação. Nesses arquivos são apresentados: o tempo de utilização, motivo de recusa de requisição e *throughput* do cenário.

5.6. Considerações Finais

Neste capítulo foram apresentados os trabalhos relacionados a modelagem de servidores web além da biblioteca utilizada para a geração das simulações que foram importantes na implementação e funcionamento do SWMAR um modelo de servidor web com 4 módulo de abstrações de atendimento das requisições.

A seção 5.5 discutiu detalhes do simulado, a seção 5.3 apresentou o gerador de cargas utilizado como carga inicial e na seção 5.4 foram fornecidas informações sobre a biblioteca utilizada na implementação do simulador.

O próximo capítulo desta dissertação apresenta os testes realizados com cargas de trabalho fornecidas pelo W4Gen e faz comparações entre os resultados ao se alterar os parâmetros de simulação, sejam eles de *hardware* ou de *software*.

Análise de Resultados

6.1. Considerações Iniciais

No capítulo anterior foi discutido o SWMAR, um simulador de servidor *web* que possui quatro módulos de abstrações de componentes internos. Foram mostradas características de implementação e outros trabalhos que auxiliaram no desenvolvimento (W4Gen e SimPack). O objetivo desse capítulo é validar o simulador proposto e verificar a sua adequação para a avaliação de desempenho. Além disso, pretende-se com este capítulo mostrar quais as possíveis formas de se utilizar o simulador desenvolvido. Diante disso, neste capítulo serão apresentados os cenários e os testes realizados.

6.2. Características Gerais de Simulação

As simulações que serão apresentadas neste capítulo consideram diferentes cenários onde são alteradas algumas características tanto do hardware a ser utilizado quanto do número de threads e tamanho de fila. O primeiro cenário utilizado na simulação considera 10 *threads* no *pool* de *threads* e tamanho de fila de requisições de 511 posições para o servidor web. Os parâmetros de *hardware* foram obtidos no trabalho de (ALMEIDA et al, 2000) que considera os seguintes valores: um processador Intel Pentium de 75MHz, 16MB de memória RAM, 36 MB de memória *swap*. Entretanto, há certas características de *hardware* que não foram descritas no trabalho de Almeida, para esses valores foram utilizados valores obtidos em pesquisas onde foram selecionadas placas-mãe que eram compatíveis com o processador. Desta forma, foi possível obter a velocidade do barramento PCI (100MBps) e a velocidade do disco (16MBps).

Além desses dados foram utilizadas características dos módulos (abstrações de componentes internos). No módulo 2 (transferência de arquivos para memória principal) foi

simulado um disco rígido logo, nesse caso foi utilizado apenas uma *thread* no parâmetro *machine.threads* e o tamanho da fila (*machine.length*) é infinito porque uma vez que a requisição é atendida por esse módulo não há recusa. Os mesmos valores foram utilizados no módulo 3. No módulo 4 foram utilizados, além do valor do tamanho da fila, valores relacionados a *buffer*, o tamanho do *buffer* e a quantidade de *buffers* que atendem esse módulo, respectivamente infinita, 128 *buffers* de tamanho de 40KB (SANT'ANNA, 2004).

Para testar o simulador com os parâmetros descritos acima foram utilizados *logs* com 2000 requisições com sobrecarga de 20%. A opção de parada por número de requisições foi escolhida porque, se escolhida a geração de *logs* por tempo haveria uma diferença no número de requisições consideradas em cada cenário simulado e a comparação do throughput obtido para os diferentes tipos de *logs* (tradicional, acadêmico e informativo/notícias) ficaria prejudicada. Na figura 6.1 são apresentadas as características das cargas iniciais baseadas nos números de objetos existentes em cada *log*.

Figura 6.1 – Características das diferentes cargas iniciais.

Na figura 6.1 os objetos foram ordenados de acordo com o maior número de objetos do tipo tradicional, foi feita essa ordenação para que se observe a característica de cauda pesada nesse tipo de *log* e também para que se possa analisar as diferenças entre os demais tipos de *logs*. Por exemplo, é possível observar que há mais imagens no tipo tradicional seguindo a lógica explicada anteriormente (possui muitas páginas estáticas com informações e figuras). No tipo acadêmico o número de documentos requisitados é muito superior aos outros tipos de *logs*, já no

tipo informativo/notícias as requisições dinâmicas e de *scripts* são maiores devido a agilidade em que as informações são atualizadas.

Para uma primeira análise dos resultados gerados pelo SWMAR foi feita uma primeira bateria de testes utilizando os parâmetros do trabalho de Almeida e se obteve os resultados apresentados na figura 6.2. Nessa figura, é possível verificar o número de requisições atendidas e o motivos de recusa de requisições. Como se pode observar, apesar dos três *logs* possuírem o mesmo número total de requisições o número de atendimentos, de recusas e motivos de recusas diferem graças as diferenças entre tipos, tamanhos e distribuições dos objetos.

Figura 6.2 – Características dos três tipos de logs.

Também é possível analisar a porcentagem de utilização de cada um dos módulos através da figura 6.3. Nesse gráfico é possível verificar que no módulo 1 a utilização é de menos de 4% na maior utilização.

Com os dados de porcentagem de utilização (figura 6.3) e os dados de recusas (figura 6.2) é possível gerar duas baterias de testes: (1) fazendo alterações visando aumentar o número de requisições atendidas ou (2) tentar aumentar a capacidade computacional aumentando parâmetros de hardware, dessa forma tenta-se aumentar a velocidade em cada um dos módulos para aumentar a capacidade de atendimento do simulador simulado. Partindo dessas duas opções foram gerados cenários que ataquem o ponto que está gerando mais recusas, no caso dos testes, o módulo 1.

Sendo assim foram feitas provas com a alteração dos parâmetros do módulo 1 para analisar quais são os resultados obtidos. Para essas novas provas foram criados mais três cenários.

Figura 6.3 – Utilização dos módulos.

Os cenários são descritos na tabela 6.1. Os 3 cenários adicionais que extrapolam os valores reais, como no caso do tamanho da fila de requisições é alterado para infinito e o número de *threads* do pool é aumentado para 500 *threads*.

	Cenário 1	Cenário 2	Cenário 3	Cenário 4
modulo1.length	511	Infinito	511	Infinito
modulo1.threads	10	10	500	500

Tabela 6 – Cenários de simulação com alteração de parâmetros do Módulo 1.

Como resultado dos cenários foram obtidos os seguintes resultados:

Figura 6.4 – Resultado de alteração de parâmetros do módulo 1 - Tradicional.

Figura 6.5 - Resultado de alteração de parâmetros do módulo 1 – Acadêmico.

Figura 6.6 - Resultado de alteração de parâmetros do módulo 1 – Informativo/Notícias.

Nas figuras 6.4, 6.5 e 6.6 são apresentados resultados que, a princípio, provam que aumentar o tamanho da fila de requisições é a melhor saída para aumentar o número de requisições atendidas por esse servidor *web*. Entretanto, mesmo aumentando o tamanho de fila não é possível garantir que todas as requisições sejam atendidas, como pode ser observado, começa a surgir recusas por falta de memória.

Outra característica é que, devido os valores de *hardware* não serem alterados, os valores de tempo total (medido em segundos) de atividade das simulações não sofrem grandes alterações e o *throughput* (medido em KBps) só recebe um ganho porque estão sendo atendidas mais requisições até o fim da simulação, como pode ser observado nas figuras 6.7, 6.8. Na figura 6.9 é

observado um comportamento diferente. Por possuir muitas requisições dinâmicas, o servidor carregado com o *log* informativo/notícias já apresentava uma alta utilização (veja figura 6.3) do módulo 3.. Sendo assim, o aumento de requisições nesse módulo já saturado gera impacto no desempenho do servidor simulado (*throughput*). Pode-se observar que ao aumentar o número de requisições no sistema com o aumento de fila a utilização do módulo 1 aumenta o número de requisições com sucesso, mas o tempo de simulação continua o mesmo uma vez que o sistema não estava sobrecarregado. Esse caso é válido tanto para os logs do tipo tradicional quanto o acadêmico. Entretanto, ao analisar o *log* informativo/notícias, que já tinha o módulo 3 muito carregado, verifica-se que com o aumento do número de requisições no sistema é o número de atendimentos mas com uma degradação da qualidade de serviço. O módulo 3 que já estava muito carregado sofre uma super utilização que gera diminuição do *throughput*.

Figura 6.7 – Throughput e tempos de simulação – Tradicional.

Figura 6.8 – Throughput e tempos de simulação – Acadêmico.

Figura 6.9 – Throughput e tempos de simulação – Informativo/notícia.

Outra maneira de se obter um aumento no número de requisições atendidas é através de alteração dos valores de hardware, dessa forma, aumentando o poder computacional dos módulos. Para escolher quais valores devem ser alterados, mais uma vez foram utilizadas as informações de porcentagem de utilização dos módulos obtidas no cenário 1.

Com as informações obtidas através dos testes anteriores, foram criados 4 cenários apenas com alteração de alguns parâmetros de hardware. Como pode ser visto na tabela 7:

	Cenário 4	Cenário 5	Cenário 6	Cenário 7
<i>machine.cpu</i>	2472Mips	102Mips	102Mips	2472Mips
<i>machine.disk.speed</i>	16MBps	300MBps	16MBps	300MBps
<i>machine.pci.speed</i>	100MBps	100MBps	533MBps	533MBps

Tabela 7 – Cenários de testes com alterações de parâmetros de hardware.

Nesses novos cenários os parâmetros foram escolhidos através de dados de recursos computacionais reais. Por exemplo, o valor de *machine.cpu* é baseado em resultados apresentados em testes sobre um processador Intel Pentium 4 de 1,3GHz. O valor de *machine.disk.speed* foi baseado em dados de fabricantes de discos rígidos SATA¹⁷ 2 ou SATA 300 que obtém 300 MBps de taxa de transferência. E por fim, o parâmetro *machine.pci.speed* também foi extraído de dados de fabricantes sobre o barramento PCI que alcança 533 MBps. Aplicando esses cenários foram obtidos os resultados apresentados nas figuras 6.10, 6.11 e 6.12. Nas figuras é possível analisar que não houve grandes ganhos em número de requisições

¹⁷ Serial ATA.

atendidas. Foi inserido o cenário 1 para que fosse feita uma comparação com os resultados obtidos nos outros cenários.

Figura 6.10 – Resultado de alteração dos parâmetros de hardware para log tradicional.

Figura 6.11 – Resultado de alteração dos parâmetros de hardware para log acadêmico.

Figura 6.12 - Resultado de alteração dos parâmetros de hardware para log informativo/notícias.

Apesar disso é possível verificar que sim houve uma diminuição dos módulos com o aumento da capacidade dos *hardwares* como pode ser observado na figura 6.13, 6.14 e 6.15. A melhora mais expressiva é no módulo 3, nos cenários que a capacidade da CPU é aumentada.

Figura 6.13 – Porcentagem de utilização dos módulos – log tradicional.

Figura 6.14 – Porcentagem de utilização dos módulos – log acadêmico.

Figura 6.15 – Porcentagem de utilização dos módulos – log informativo/notícia.

É importante ressaltar que os *logs* de entradas foram sobrecarregados. Logo, apresentam um cenário onde uma grande parte de requisições possuem o mesmo tempo de chegada. Sendo assim, essa é uma simulação de um curto espaço de tempo de um servidor *web* muito requisitado. Dessa forma, uma grande parte das requisições é rejeitada devido o tamanho da fila de requisições e ao número de *threads* no *pool* que atendem o servidor *web*. Com isso foi criado mais um cenário que possui as mesmas características de *hardware* que o cenário 8 mas com alterações no tamanho da fila de requisições e no número de *threads* do *pool* de *threads*, ou seja, um melhor cenário de *hardware* unido a melhora obtida com o aumento dos parâmetros do módulo 1. Portanto, no cenário 9 a fila possui tamanho de 1022 (100% a mais que o padrão) e o *pool* possui 20 *threads* (100% a mais que o padrão).

Mesmo com as melhoras agregadas, o cenário 9 não obteve 100% de requisições atendidas (ficou entre 60% no pior caso e 75% no melhor caso).

Para verificar o resultado da alteração no tamanho da fila e número de *threads* no *pool* foram feitas comparações com os cenários anteriores e levado em conta o *throughput* de cada cenário analisado nesse capítulo em todos os cenários testados.

Figura 6.16 – Throughput – Log Tradicional.

Figura 6.17 – Throughput – Log Acadêmico.

Figura 6.18 – Throughput – Log Informativo/Notícia.

Como resultado da análise dos *throughputs* é possível observar as características que realmente afetam o *throughput*. No casos dos *logs* testados o maior gargalo era mesmo a fila de requisições e o número de *threads* no *pool*. Tendo em vista que a sobrecarga aplicada nos *logs* aumentou o número de requisições com tempo de chegada igual a um ponto que o servidor *web* não poder atender as requisições mesmo possuindo pouca utilização dos módulos.

Também é possível aumentar o *throughput* do sistema apenas alterando o hardware como observado nos cenários de 5 a 8.

6.3. Considerações Finais

O objetivo do presente capítulo foi apresentar os resultados de uma avaliação de desempenho utilizando o SWMAR com cargas de trabalho geradas pelo W4Gen. Para gerar essas avaliações foram considerados nove cenários diferentes onde o cenário 1 é extraído do trabalho de (Almeida et al. 2000), três cenários são alterações de parâmetros relacionados a características do servidor *web*, quatro cenários são alterações visando melhoria de atendimento modificando o *hardware* e um cenário faz melhoria tanto de *software* quanto de *hardware*. Houve casos que os resultados foram melhores devido a característica da carga inicial.

No próximo capítulo serão apresentadas as conclusões obtidas com o desenvolvimento deste trabalho, as contribuições e as propostas de trabalhos futuros relacionados à avaliação de desempenho.

7.1. Considerações Finais

A Web é um sistema em crescimento que passou de um simples meio de publicação para disseminar informações, por meio do uso de páginas estáticas, para uma infra-estrutura com suporte a processamento de transações, com uso de páginas dinâmicas com acesso regular a banco de dados e páginas personalizadas. Com essa evolução, muitos esforços já foram e serão dispendidos para melhorar o tempo de resposta das requisições dos usuários.

Dentre os diversos estudos, pode-se destacar o modelo de servidor *web* com diferenciação de serviços (SWDS), cujo objetivo é oferecer uma arquitetura para o fornecimento de serviços diferenciados, integrando os princípios de QoS aplicados na camada de rede.

No entanto, a validação do modelo SWDS, executada por meio de simulações usando o *trace* da copa do mundo de 1998, não demonstrou sua viabilidade com base nas características atuais, constituindo um comportamento antigo de uma época onde o protocolo HTTP 1.1 e requisições dinâmicas eram pouco usadas. Ao longo desses últimos anos, muitas tecnologias mudaram, provocando uma mudança na demanda dos servidores *web*.

Com base nessa idéia, a finalidade deste trabalho é oferecer um modelo de servidor *web* que possa integrar ao modelo do SWDS um modelo de servidor *web* que possui características mais próxima de um servidor *web* real (SWMAR) substituindo o *cluster* de servidores *web*, agregado as cargas de trabalho geradas pelo W4Gen. Assim, este trabalho proporciona a integração de vários trabalhos já realizados e permite obter novos resultados mais confiáveis e mais realistas. Dentre os trabalhos desenvolvidos no grupo de Sistemas Distribuídos e Programação Concorrente do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo que podem ser beneficiados, por meio de simulações mais realistas, utilizando-se o simulador desenvolvido neste trabalho vale ressaltar:

- O trabalho de iniciação científica de Traldi et al. (2004) consistiu em transportar o algoritmo Weighted Fair Queuing (WFQ), projetado para o nível de rede, para o nível de aplicação, sendo incluído, assim, no modelo de Servidor *Web* com Diferenciação de Serviços (SWDS).
- O trabalho de iniciação científica de Barbato et al. (2005) consistiu em implementar o algoritmo RSVAdap, cujo objetivo é particionar dinamicamente o cluster de servidores *Web*, de acordo com a carga de trabalho vigente no sistema, a fim de melhorar o atendimento dos clientes de mais alta prioridade.
- O trabalho de mestrado de Estrella (2006) estudou a validação de mecanismos de negociação no módulo de controle de admissão do modelo SWDS, verificando uma melhoria no atendimento aos clientes pertencentes a uma determinada classe de serviço para que esses tivessem uma qualidade de serviço garantida.
- O trabalho de mestrado de Mourão (2005) tem como objetivo introduzir a capacidade de reconhecimento de sessões HTTP no modelo do servidor SWDS, característica atualmente não contemplada.
- O trabalho de mestrado de Messias (2006) teve como objetivo implementar um protótipo de servidor *web* com diferenciação de serviços e compará-lo com um servidor *web* tradicional para avaliar os possíveis ganhos em ambiente real.

7.2. Contribuições

A maior contribuição deste projeto de mestrado foi desenvolver um modelo e o respectivo simulador de servidor *web* com abstrações de componentes internos que tratam as requisições (SWMAR) cujo objetivo é facilitar diversos estudos em andamento, visando à avaliação do desempenho da *Web*, aumentando a capacidade de análise de diferentes cenários com diferentes configurações de servidores *web*. O servidor desenvolvido é flexível e apresenta a possibilidade de alteração no comportamento de um servidor *web*, por meio da alteração de diferentes parâmetros, tanto de software quanto de hardware do servidor. Por exemplo, pode-se alterar a política de atendimento da fila de requisições de FIFO para uma política de prioridades de maneira rápida graças aos métodos disponíveis pela biblioteca SimPack.

Além do desenvolvimento do protótipo, têm-se a realizar diversas simulações que ilustram formas de utilização de simulador desenvolvido. Por meio dos exemplos apresentados pode-se verificar como detectar pontos de gargalo do sistema, quantificar como alterações no hardware e no software do sistema inferem em seu desempenho e analisar a influência de cada fator avaliado e a interação existente entre os fatores.

Como todos os parâmetros utilizados no exemplo desenvolvido são realistas e foram propostos fundamentando-se em dispositivos existentes, os resultados obtidos na simulação podem ser utilizados para avaliar o comportamento de um servidor *web*.

7.3. Sugestões para Trabalhos Futuros

O projeto desenvolvido apresentou como motivação a integração de diversos trabalhos desenvolvidos no grupo de Sistemas Distribuídos e Programação Concorrente. A continuidade deste trabalho deve prever o desenvolvimento de novas funcionalidades para o simulados, visando à possibilidade de comportar todas as simulações contidas nos trabalhos citados na seção 7.1. Desta forma, algumas sugestões de trabalhos futuros:

- Atualizar as medidas de tempo de utilização de cada módulo do SWMAR através de uma análise de desempenho feita pela monitoração de um servidor *web* real.
- Gerar um *cluster* com vários servidores *web* utilizando o SWMAR. Dessa forma, pode-se gerar *cluster* heterogêneos ou homogêneos e verificar o comportamento nesses diferentes cenários.
- Agregar ao SWMAR a abstração do meio físico de comunicação entre o cliente e o servidor *web*, a rede.
- Integrar ao SWDS as cargas geradas pelo W4Gen e como modelo de servidor *web* SWMAR.

Referências Bibliográficas

- ABDELZAHER, T. F.; LU, C. Modeling and Performance Control of Internet Servers. In: *Proceeding of the 39th IEEE*, Dez. 2000.
- ALMEIDA, J., M.; ALMEIDA, V.; YATES, D. *Measuring the Behavior of a World-Wide Web Server*. 2000.
- ARLITT, M. F.; WILLIAMSON, C. L. Web Server Workload Characterization: the search for invariants. In: *Proceedings of ACM SIGMETRICS '96*. [S.I. : s.n.], 1996. p.126-37.
- BANGA, G.; DRUSCHEL, P.; MOGUI, J.C Better operating system features for faster network servers. *ACM Performance Evaluation Review*, 1998, p. 23-30.
- BERNERS-LEE, T.; MASINTER, L.; McCAHILL, M. *Uniform Resource Locators*, [S.I.], Dez. 1994. RFC 1738, IETF.
- BINKERT, N. L., HALNOR, E. G., REINHARDT, S. K. *Network-Oriented Full-System Simulation using M5*. In Workshop on Computer Evaluation using Commercial Workloads, Fev 2003.
- CARDELLINI, V.; CASALICCHIO, E.; COLAJANNI, M.; YU, P. S. The State of the Art in Locally Distributed Web-Server Systems. In: *ACM Computing Surveys*, 2002, v. 34, n. 2, p. 263-311.
- CERF, V. C.; KAHN, R. E. "A Protocol for Packet Network Interconnections", IEEE Transaction on Communications, Vol. COM-22, No. 5, Mai 1974.
- COFFMAN, K. G; ODLYZKO, A. M. Internet growth: Is there a "Moore's Law" for data traffic? In *Handbook of Massive Data Set*, 2001.
- COMER, D. E. *Internetworking with TCP/IP: Principles, Protocols and Architecture*. 4.ed. [S.I.]: Prentice Hall, 2000.
- DEVLIN, B.; GRAY, J.; LAING, B.; SPIX, G. *Scalability terminology: Farms, clones, partitions, and pack: RACS and RAPS*. Tech Rep. MS_TR-99-85, Microsoft Research, 1999.
- DILLEY, J.; FRIEDRICH, R.; JIN, T.; ROLIA, J. *Web server performance measurements and modeling techniques*, Performance Evaluation, 2000.
- ESTRELLA, J, C,. *Mecanismos de negociação no módulo de controle de admissão da arquitetura de servidor web com diferenciação de serviços (swds)*. Qualificação (Mestrado) – ICMC-USP, São Carlos, 2005.
- FIELDING, R. *Hypertext Transfer Protocol -- HTTP/1.1*, [S.I.], Jun. 1999. RFC 2616, IETF.
- FRANCÉS, C. R. L. et al. *Modelagem e Especificação Utilizando Redes de Petri e Statecharts. Monografia* (Notas Didáticas) — ICMC-USP, São Carlos, 2000.
- FRANCÉS, C. R. L. *Uma extensão estocástica para Statecharts*. Dissertação (Mestrado), ICMC-USP, São Carlos, 1998.
- GANDHI, N.; TILBURY, D. M.; DIAO, Y.; HELERSTEIN, J.; PAREKH, S. MIMO Control of a Apache Web Server: Modeling and Controller Design. In *Proceeding of the American Control*

Conference, Mai. 8-10, 2002.

GILDER, G. *Fiber keeps its promise: Get ready. Bandwidth will triple each year for the next 25.* Forbes, 7 abr, 1997.

GRAY, j.; SHENOY, P. Rules of thumb in data engineering. In *Proceedings of the 16th IEEE International Conference on Data Engineering.*, IEEE Computer Soc. Press, 2000, p. 3-10.

HAREL, D. *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming. [S.I.]: John Wiley, 1987.

HEIDERMAN, J.; OBRACZKA, K.; TOUCH, J. Modeling the Performance of HTTP Over Several Transport Protocols. *IEEE/ACM Transactions on Networking*, v. 5, n.5. Out. 1997.

HUITEMA, C. *Network vs. server issues in end-to-end performance. Keynote speech at Performance and Architecture of Web Servers Workload.* Disponível em : http://kkant.csswebhost.com/PAW2000/huitma_keynote.ppt. Acesso em: 13 Jan. 2006

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* [S.I.]: John Wiley, 1991.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores*, 2 ed [S.I.]. Addison Wesley, 2003.

MENASCÉ, D. A.; ALMEIDA, V. A. F. *Planejamento de capacidade para Serviços na Web*. 1. ed [S.I.]: Campus, 2003.

MOCKAPETRIS, P. Domain Names - Concepts and Facilities. Req. For Com. 1034, *USC Information Sci.* Institute, Nov. 1987.

MOURÃO, h, C,. Reconhecimento de Sessões HTTP em um Modelo para Servidor Web com Diferenciação de Serviços. Dissertação (Mestrado) – ICMC-USP, São Carlos, 2005.

POSTEL, J. *Domain Name System*, [S.I.], Fev. 1984. RFC 897, IETF.

POSTEL, J. *Internet Protocol*, [S.I.], Set. 1981. RFC 791, IETF.

POSTEL, J. *Simple Mail Transfer Protocol*, [S.I.], Aug. 1982. RFC 821, IETF.

POSTEL, J. *Transmission Control Protocol*, [S.I.], Set. 1981. RFC 793, IETF.

POSTEL, J; REYNOLDS, J. *File Transfer Protocol*, [S.I.], Out. 1985. RFC 959, IETF.

REESER, P.; HARIHARAN, R. Analytic Model of Web Servers in Distributed Environment

ROBERTSON, A.; WITTENMARK, B.; KIHIL, M. Analysis and design of a admission control in web-server systems. In *Proceeding of the American Control Conference*, Jun. 2003.

SANT' ANNA FILHO, A. *Modelo de servidor Web com conexões persistentes e carga orientada a sessão.* Tese (Doutorado) – ICMC-USP, São Carlos, 2004.

SANTANA, R. H. C. et al. *Técnicas para Avaliação de Desempenho de Sistemas Computacionais.* Monografia (Notas Didáticas) — ICMC-USP, São Carlos, 1994. *SPEC. SPECweb99 Benchmark. 2001.* Disponível em: <<http://www.spec.org/osg/web99/>>. Acesso em: 04 Nov. 2004.

SILVA, L. H. C. *Um Gerador de Cargas de Trabalho Sintéticas - Aplicação ao Modelo de Servidor Web com Diferenciação de Serviços*, São Carlos, 2005. (Monografia de Qualificação de Mestrado) - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

SILVA, L. H. *Caracterização de carga de trabalho para testes de modelos de servidores web*, São Carlos, 2006 (Dissertação de Mestrado) – Instituto de Ciências Matemáticas e de Computação –

ICMC-USP.

SIMPSON, W. *The Point-to-Point Protocol (PPP)*, [S.I.], JuL. 1994. RFC 1661, IETF.

SLOTHOUBER, P. *A Model OF Web Server Performance*, 1995.

SOCOLOFSKY, T.; KATE, C. *TCP/IP Protocol*, [S.I.], Jan. 1991. RFC 1180, IETF.

TANENBAUM, A. *Distributed Operating Systems*. 3 ed. [S.I.] Prentice Hall, 1995.

TANENBAUM, A. S. *Computer Networks*. 4 ed. [S.I.]. Prentice Hall, 2002.

TEIXEIRA, M. A. M. *Suporte a Serviços Diferenciados em Servidores Web: Modelos e Algoritmos*. Tese (Doutorado) — ICMC-USP, São Carlos, 2004.

van der MEI, R.; HARIHARAN, R.; RESSER, P. K. Web Server Performance Modeling. *Telecommunications Systems*, v. 16, n. 3, p. 361-378, 2001.

WELLS, L.; CHRISTENSEN, S.; KRISTENSEN, L. M.; MORTENSEN, K. H. *Simulation Based Performance Analysis of Web Servers* In: *Proceedings of 9th International Workshop on Petri Nets and Performance Models, PNPM'01 Aachen, Sept. 11-14 , 2001, Reinhard German and Boudewijn Haverkort (eds.), IEEE*, pages 59-68. 2001.

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.