Ferramentas de avaliação de desempenho para servidores web: análise, implementação de melhorias e testes

Hermes Pimenta de Moraes Júnior

Orientador: *Prof. Dr. Marcos José 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.

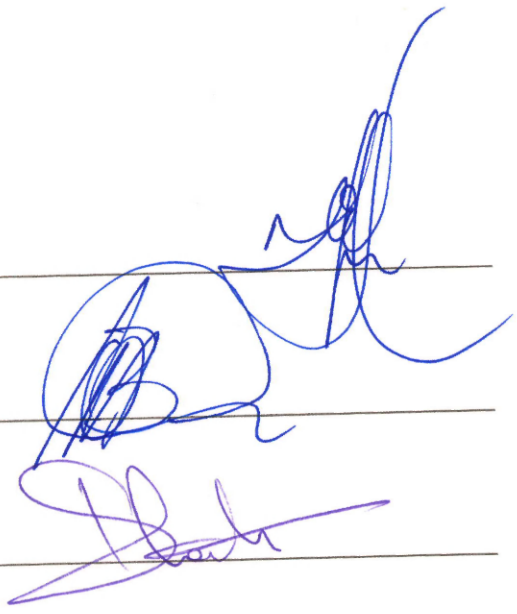
USP – São Carlos
Janeiro/2004

A Comissão Julgadora:

Prof. Dr. Marcos José Santana _____

Prof. Dr. Odemir Martinez Bruno _____

Prof. Dr. Jan Frans Willen Slaets _____



Dedicatória

Aos meus pais, Hermes e Dirce.

Agradecimentos

Em primeiro lugar gostaria de agradecer aos professores Marcos e Regina Santana. Ao professor por ter-me proporcionado muito além da orientação acadêmica e pela paciência demonstrada em diversas ocasiões. À professora, pelas idéias e sugestões apresentadas nas inúmeras conversas que tivemos.

À Luciana Garcia, minha namorada, pelo companheirismo e apoio ao longo deste último ano. Não esquecendo a dedicação demonstrada nas correções gramaticais do meu texto.

Ao amigo Mário Meireles, pela grande ajuda prestada por meio dos esclarecimentos de dúvidas que pareciam não terminar.

Ao amigo Márcio Augusto, pelos e-mails antológicos enviados para a lista do grupo e por sempre estar disposto a ajudar nos momentos mais difíceis.

Aos amigos Luciano (Lulu) e Douglas (porquinho), pelas brincadeiras e palhaçadas apresentadas no laboratório de tempos em tempos. E também por sempre perderem para o meu time no vôlei.

Às Professoras Sarita e Elisa pelas diversas vezes em que me ajudaram.

Aos grandes amigos que encontrei na USP: Álvaro, Caio, Edmilson, Juliano, Kalinka, Michel, Omar, Renato (JapaGay), Renato Bulcão, Simone e Thais.

À USP, por toda a infraestrutura e oportunidades que disponibiliza a seus alunos.

Ao CNPq, pelo apoio financeiro sem o qual este trabalho não poderia ser realizado.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta o projeto e implementação de melhorias na ferramenta httperf, que é um *benchmark* para servidores Web. A melhoria implementada constitui na interpretação de logs para a extração de parâmetros de carga de trabalho. Para tanto, foi necessária a criação de um ferramenta à parte, que trabalha o arquivo de log e retira os parâmetros a serem utilizados pelo httperf.

O desenvolvimento do trabalho foi baseado em uma revisão bibliográfica cobrindo avaliação de desempenho de sistemas computacionais, características da Web e ferramentas/*benchmarks* especializados em sistemas baseados na Web. Apresenta ainda uma discussão sobre arquiteturas de servidores Web e o uso dos *benchmarks* na avaliação desses servidores. Foi desenvolvido um estudo sobre as ferramentas, onde foram analisadas suas características, com o intuito de identificar características não abordadas, mas julgadas importantes nesse tipo de avaliação. A partir desse estudo selecionou-se algumas das características identificadas que foram incorporadas à ferramenta escolhida. Finalmente, a ferramenta que recebeu as alterações foi avaliada por meio de diversos experimentos, quando os resultados obtidos mostraram que a implementação desenvolvida era viável.

Abstract

This work presents the design and implementation of improvements for httperf - a benchmark tool for Web servers. The improvements are basically the interpretation of logs for extracting workload parameters. The creation of another tool was necessary to extract from the log the parameters to be used by httperf.

Work development was based in reviewing the literature on performance evaluation of computing systems, Web characteristics and benchmarks for Web based systems. Discussion about Web server architecture and benchmark using for server evaluation are also presented. A study about similar tools (such as httperf) was made, where characteristics were analyzed to identify the ones not approached yet, being relevant for performance evaluation. Some were identified, selected and inserted in the chosen tool. Finally, changes were incorporated to the tool, which was then evaluated through several experiments; results show the developed improvements are usable.

Capítulo 1

Introdução

1.1 Contextualização

Com a idéia inicial de criar uma rede que não pudesse ser destruída por bombardeios e fosse capaz de ligar pontos estratégicos, como centros de pesquisa e tecnologia, surgiu em 1969, a Arpanet. O que começou como um projeto de estratégia militar, com a finalidade de atender às demandas do DoD (Departamento de Defesa dos Estados Unidos), acabou se transformando no que hoje é conhecido como a Internet (Tanenbaum, 2002).

Para alcançar os objetivos almejados, foi necessário criar uma estrutura totalmente descentralizada, com todos os pontos (nós) tendo o mesmo status. Os dados deveriam caminhar de forma independente, em qualquer sentido. Assim, se um nó fosse perdido, destruído, o restante da rede continuaria operando normalmente.

Os pontos de comunicação da rede cresceram rapidamente, interligando cada vez mais instituições de pesquisa e órgãos do governo. Em 1989, quando a Internet já havia alcançado dimensões que extrapolavam os limites dos Estados Unidos, Tim Berners-Lee, físico ligado ao CERN (Centro Europeu de Pesquisa Nuclear), propôs uma nova estrutura arquitetural para acessar documentos interligados espalhados em milhares de máquinas. A idéia inicial era de uma teia de documentos ligados e surgiu da necessidade de cooperação entre grandes grupos internacionais de pesquisadores que utilizavam documentos de tipos variados e em constante mutação. Essa nova estrutura deu origem à World Wide Web, ou simplesmente Web (Tanenbaum, 2002).

Apesar de ter surgido apenas como um serviço dentro de toda a Internet, a Web se tornou um dos principais fatores que impulsionaram o crescimento da Rede. Pode-se dizer que a Web foi a responsável pela popularização do uso da Internet, de tal forma que algumas pessoas mais leigas a confundem com a própria Internet. Serviços disponíveis há muito mais tempo, como *e-mail* e FTP, passaram a ter, nos navegadores web, sua forma de interface preferencial com os usuários.

Desde a idéia inicial, as técnicas empregadas na Web evoluíram muito (Orfali *et al.*, 1999; Ivan Herman, 2003), sempre visando o aumento das aplicações que se pode executar sobre a estrutura oferecida, ou seja, serviços e novos recursos que os usuários podem obter. Com isso, a Web conseguiu transformar a Internet de um ambiente para fins de pesquisa, restrito a centros acadêmicos e instituições governamentais, para um sistema altamente explorado, no que tange à educação, ao entretenimento, e principalmente ao comércio.

Em relação a este último ponto e devido ao crescimento explosivo apresentado pela Web, empresas e instituições que ofereciam serviços baseados na Rede, se viram cada vez mais dependentes de servidores com grande capacidade para garantirem o sucesso de seus negócios. A grande demanda de requisições sempre saturava até mesmo sistemas com grande desempenho. Tornou-se então necessário o freqüente incremento na capacidade dos sistemas responsáveis pelo atendimento a essas requisições, podendo esse aumento ser obtido através de diversas maneiras como replicação (*mirroring*) (Stading *et al.*, 2002), melhor utilização de caches (Fan *et al.*, 2000) e melhores servidores http (J. Hu, 1998; Apache Software Foundation, 2003).

Atualmente, grande parte dos trabalhos feitos para se medir o desempenho de *software* na Web tem se concentrado na caracterização precisa de cargas geradas nos servidores, em termos de tipos de arquivos requisitados, tamanho das transferências e outras estatísticas relacionadas (Arlitt & Williamson, 1996; Abdelzaher & Bhatti, 1999). Existem exemplos que utilizam diretamente cargas reais (Maltzahn *et al.*, 1997), e outros que utilizam cargas geradas sinteticamente (Mosberger & Jin, 1998; Crovella *et al.*, 1999; Mindcraft, 2002), baseadas em invariantes observadas no tráfego real da Web. Esse último método apresenta falhas que se apresentam quando há a necessidade de definir cargas que se aproximem, o máximo possível, de cargas reais. Aquele primeiro método apresenta dificuldades relacionadas à reprodução de cargas reais.

A avaliação de servidores Web não é trivial. Existem diversas ferramentas, que implementam diferentes métodos, disponíveis para executar essa tarefa. Não há, no entanto, nenhuma forma ideal, deixando esse problema ainda em aberto.

1.2 Objetivos

O presente trabalho tem como objetivo o estudo de mecanismos utilizados para avaliação de servidores web, em particular as ferramentas que utilizam cargas sintéticas. Com isso, pretende-se chegar a uma comparação das ferramentas abordadas e identificar características chaves, que auxiliem essas ferramentas a alcançar cargas mais próximas do real. A partir desse estudo e das características identificadas, são selecionados uma característica e um mecanismo que irá recebê-la por meio de implementação. Assim, o objetivo principal deste trabalho é incluir melhorias em uma ferramenta existente para que ela seja capaz de gerar cargas mais próximas do real, alcançando, dessa forma, melhores resultados

nas avaliações de servidores.

Atualmente, podem ser encontradas diversas ferramentas (*benchmarks*) para a avaliação de servidores Web. Entre as mais citadas na literatura especializada estão WebStone (Mindcraft, 2002), Surge (Crovella *et al.*, 1999), SPECweb (SPEC, 1999) e Httperf (Mosberger & Jin, 1998).

Quanto às características, diversas podem ser acrescentadas a esses mecanismos ou ferramentas. Um exemplo importante é a capacidade de leitura de logs gerados por servidores Web. Esses logs possuem informações à respeito de tudo o que foi visto pelo servidor como: número de requisições atendidas; tipo e tempo entre chegadas dessas requisições. A extração de parâmetros de carga presentes nesses logs representa uma importante forma de se caracterizar a carga a que o servidor foi exposto.

De forma mais específica, o que se pretende é fazer com que um *benchmark* escolhido dê uma passo a mais em direção à obtenção de cargas de trabalho mais próximas do real por meio do tratamento e extração de parâmetros de logs.

1.3 Estrutura

No Capítulo 2, discute-se a avaliação de desempenho de sistemas. São apresentadas as técnicas de aferição (Protótipos, *Benchmarks* e Coleta de Dados) e de modelagem (Redes de Filas, Redes de Petri e *Statecharts*). Posteriormente é feita uma discussão sobre a resolução de modelos, com a apresentação dos tipos de solução, que podem ser por abordagem Analítica ou por Simulação.

O Capítulo 3 desta dissertação apresenta a infraestrutura da Internet, seus protocolos (TCP/IP, HTTP) e serviços principais, destacando-se a Web. É feita também uma revisão sobre estudos recentes a respeito da caracterização de carga na Web e sobre arquiteturas de servidores Web. Sobre os servidores é destacada a estratégia de concorrência utilizada por cada um deles. São discutidos servidores do tipo Iterativo, (que usam um processo por requisição), do tipo *Thread pool* e do tipo Única *thread*.

Os *benchmarks*, ou ferramentas, utilizados para a avaliação de servidores Web, são apresentadas no Capítulo 4. Em particular são discutidos os *benchmarks* WebStone, e httperf, escolhidos por serem os mais abordados na literatura. Ao final deste capítulo é feita uma breve comparação entre os dois *benchmarks*, escolhendo-se um para receber as implementações propostas.

O Capítulo 5 aborda o desenvolvimento do objetivo proposto pelo trabalho. Assim, algumas considerações são feitas sobre os *benchmarks* adotados, as decisões do projeto e os detalhes de implementação envolvendo a nova ferramenta construída e as características adicionadas ao httperf. São apresentados ainda nesse capítulo, os resultados obtidos através de experimentos realizados com a nova ferramenta alcançada.

Para finalizar, as conclusões, contribuições do trabalho, dificuldades encontradas e sugestões para trabalhos futuros são apresentadas no Capítulo 6.

Capítulo 2

Avaliação de Desempenho

2.1 Introdução

Uma figura de desempenho de um sistema computacional corresponde ao resultado de uma avaliação da quantidade de serviços prestados ao longo de um determinado período de tempo. O desempenho de um sistema computacional é, antes de tudo, observado por seus usuários a partir dos tempos de respostas observados. Embora não constitua um parâmetro confiável, a qualidade de um sistema é, de certa forma, avaliada de acordo com o grau de contentamento de seus usuários. Assim, quando há descontentamento entre um grupo de usuários de um determinado sistema, torna-se necessária um avaliação minuciosa a fim de se determinar possíveis gargalos.

A avaliação de desempenho em Sistemas Computacionais Distribuídos pode ser empregada, na *avaliação de sistemas existentes*, na *avaliação de sistemas em desenvolvimento*, ou na *seleção de um sistema determinado* (Francês, 1998). No primeiro caso, podem-se destacar como possíveis objetivos a maximização da eficiência e da utilização de um determinado recurso e a minimização do tempo de resposta e do custo de processamento de uma carga de trabalho. No caso de sistemas inexistentes, a avaliação deve ser capaz de fazer previsões sobre o comportamento do novo sistema. Dessa forma a avaliação de desempenho está presente desde a concepção do sistema até sua instalação e utilização diária. Para a seleção de novos sistemas são feitas comparações entre os desempenhos de diversos sistemas e a escolha é feita de acordo com um parâmetro específico.

Apesar da avaliação de desempenho ser de grande importância, muitas vezes ela é negligenciada, o que acarreta conseqüências graves para as atividades do sistema em questão. Esse descaso, na maioria das vezes, é atribuído a dois fatores: à falta de conhecimento de ferramentas para efetuar a avaliação, ou à dificuldade de realizá-la - apesar da utilização dessas ferramentas. De modo geral, dependendo do sistema em foco e do tipo desejado de avaliação, é possível utilizar-se técnicas de aferição (*benchmarks, prototipação, coleta de dados*), ou modelagem (com solução analítica ou por simulação).

2.2 Técnicas para Avaliação de Desempenho

Para a avaliação de desempenho de um sistema, o avaliador deve coletar informações referentes aos parâmetros significativos à avaliação. Técnicas de avaliação são, justamente, métodos para a obtenção dessas informações. Isso pode ser feito através do próprio sistema (técnicas de aferição - *benchmarks*, coleta de dados e prototipação) ou a partir de modelagem do sistema.

Os sistemas computacionais atuais atingiram uma grande complexidade tornando a utilização de técnicas de aferição em sua avaliação também bastante complexa. Dessa forma, as técnicas de modelagem têm sido amplamente utilizadas na avaliação de sistemas em geral.

Entretanto, o emprego de técnicas baseadas em modelos requer que estes sejam resolvidos, e que sua confiabilidade seja garantida. Isso pode ser obtido através do conhecimento prévio de que sua base conceitual esteja correta. Sem essas "garantias" não se pode dizer nada além de que os resultados obtidos apenas refletem o comportamento do sistema real.

2.2.1 Técnicas de Aferição

Quando o sistema a ser avaliado já existe ou está em fase final de desenvolvimento, o estudo de seu desempenho pode ser feito através da experimentação direta. Pode-se utilizar o sistema e coletar dados a seu respeito diretamente.

Em relação à modelagem, as técnicas de aferição apresentam vantagens - abservadas através de resultados mais precisos - e desvantagens - a necessidade de se ter um sistema disponível. Nos dois casos, é necessário garantir que a própria técnica não interfira no comportamento do sistema e comprometa os resultados obtidos.

Algumas das principais técnicas de aferição são (Orlandi, 1995): os protótipos, os benchmarks e a coleta de dados. A seguir, essas técnicas serão discutidas.

Protótipos

A construção de protótipos representa uma simplificação de um sistema computacional, mantendo, contudo, a mesma funcionalidade. Os protótipos consideram algumas características em detrimento de outras e podem ser considerados um meio termo entre o sistema final e as expectativas que se tem dele, quando ele ainda não existe. Essa técnica possui um custo menor do que a construção do sistema real, apesar de ainda relativamente elevado em relação às demais técnicas de aferição. Há outra dificuldade ao se construir um protótipo: determinar quais características são essenciais ao sistema.

Benchmarks

Benchmarks são programas usados para o teste de *software*, *hardware* ou sistemas computacionais completos (Collin, 1993). Na medição do desempenho de sistemas computacionais, os *benchmarks* podem utilizar tanto tarefas mais gerais (operações de I/O), quanto tarefas específicas (como representação de polígonos ou operações sobre matrizes). Em suma, qualquer aspecto de desempenho de um sistema computacional que importe aos usuários pode ser objeto de medição por meio de *benchmarks*. Entretanto, alguns cuidados devem ser observados em relação à utilização dos *benchmarks*. Primeiro, por se tratar de uma técnica de aferição, deve-se verificar se o próprio *benchmark* não influenciará no comportamento do sistema. Outra dificuldade reside na escolha da unidade usada como referência para realização da comparação. A utilização, por exemplo, de unidades como MIPS (*Millions of Instructions Per Second*) ou FLOPS (*FLoating Point Operations Per Second*) gera bastante controvérsia, pois elas fornecem valores perigosamente absolutos, mesmo diante de fatores distintos (como arquiteturas RISC e CISC), que podem influenciar sobremaneira os resultados.

Em virtude da importância desse assunto no contexto geral do trabalho, ele será discutido mais detalhadamente em um capítulo à parte.

Coleta de dados

Como mencionado, as técnicas de aferição são os métodos que oferecem os resultados mais precisos. Dentre essas técnicas, a coleta de dados é a mais precisa.

Uma utilização bastante interessante pode ser dada à coleta de dados: os resultados obtidos por meio dessa técnica podem ser usados para comparações com os resultados obtidos a partir de modelos do mesmo sistema. Esse procedimento pode ser empregado como parte da validação de um modelo.

A coleta de dados pode ser realizada através de dois recursos: os **monitores de hardware** e os **monitores de software**.

- Monitores de *hardware*: são *hardwares* específicos empregados para coletar e analisar alguns dados pertinentes ao objeto em estudo (Orlandi, 1995). Os Monitores de *hardware* devem ser também não intrusivos, isto é, devem limitar-se a obter os sinais sem alterá-los, mantendo fidelidade aos valores originais.

- Monitores de *software*: são usados nos casos em que se deseja observar características específicas de *software* como, por exemplo, a verificação da existência ou não de uma fila de espera, associada a algum recurso do sistema.

A Figura 2.1 mostra alguns exemplos de utilizações das técnicas mencionadas acima. As formas geométricas idênticas indicam uma relação(técnica, proposta de avaliação) mais

apropriada. Já as setas indicam a possibilidade de utilização da técnica, porém, sendo menos aconselhada. Dessa forma, a Coleta de Dados, por exemplo, pode ser utilizada na resolução de qualquer problema citado, mas se aplica melhor na avaliação de sistemas existentes.

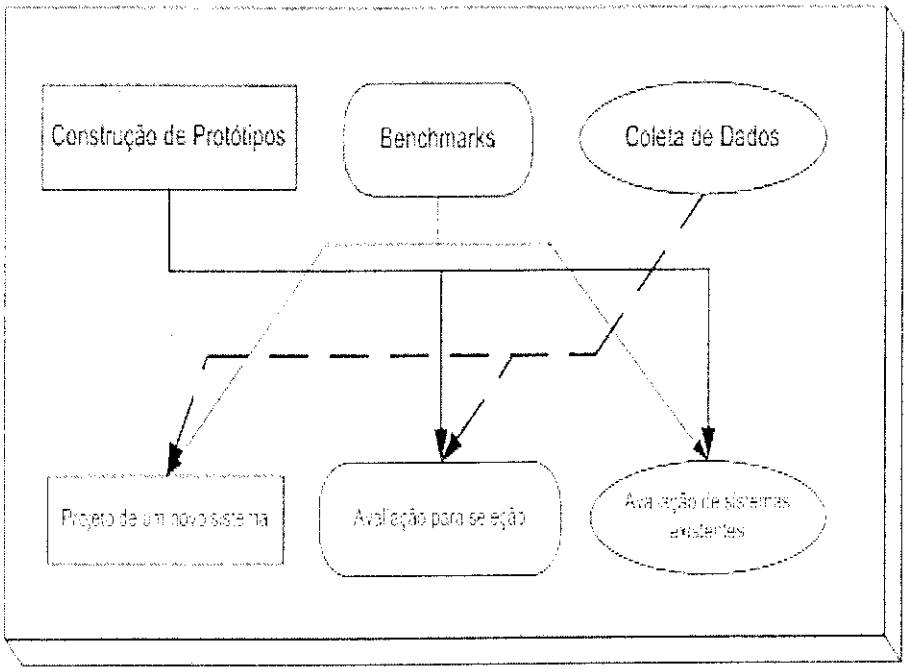


Figura 2.1: Relação entre propostas de avaliação e técnicas de aferição.

2.2.2 Modelagem

Não é interessante para as indústrias, centros de pesquisas, ou qualquer outra instituição, ter que construir um sistema para só então verificar seu desempenho. A verificação e o teste de um sistema, antes mesmo de este existir, representa uma redução enorme de gastos para o desenvolvedor. Para esse tipo de situação, em que se deseja testar um sistema sem que ele exista, ou que sua experimentação prática seja bastante complexa, é que se propõe a modelagem.

Um **modelo** nada mais é que a representação de um determinado sistema, com o intuito de evidenciar suas características mais importantes. Os modelos sofrem alterações com o tempo, pois o estado de um sistema também se altera dessa forma. Assim, os modelos podem ser classificados em relação ao tempo como: discretos ou contínuos. Uma vez que a alteração de estado em computadores ocorre a intervalos discretos de tempo, para a modelagem desse tipo de sistema são usados também modelos discretos.

Existem várias técnicas para modelar um sistema computacional. Três técnicas têm sido amplamente utilizadas e possuem vantagens e desvantagens conforme o domínio de aplicação considerado: redes de filas, redes de Petri e *statecharts*. A seguir cada uma

dessas técnicas é detalhada.

Redes de Filas

Na computação, há diversas ocasiões em que aparecem "disputas" por algum recurso, como por exemplo, acesso a disco, utilização da CPU, acesso à rede. Nessas ocasiões é inevitável o surgimento de filas. Para modelar esses sistemas, foi criada uma técnica baseada na teoria de filas (um ramo das probabilidades) denominada rede de filas.

Uma **rede de filas** consiste de duas entidades: **centros de serviços** e **usuários**. As primeiras são entidades que oferecem serviços dos quais os usuários usufruem (Soares, 1992). Um centro de serviço pode ser constituído de um ou mais servidores que correspondem aos recursos no sistema modelado. Há ainda uma área de espera (**fila**) para os usuários que estão requisitando serviços mas ainda não conseguiram acesso. Como exemplo, uma situação que ocorre constantemente em sistemas computacionais é a disputa pelo processador por diversos processos. A Figura 2.2 ilustra essa situação.

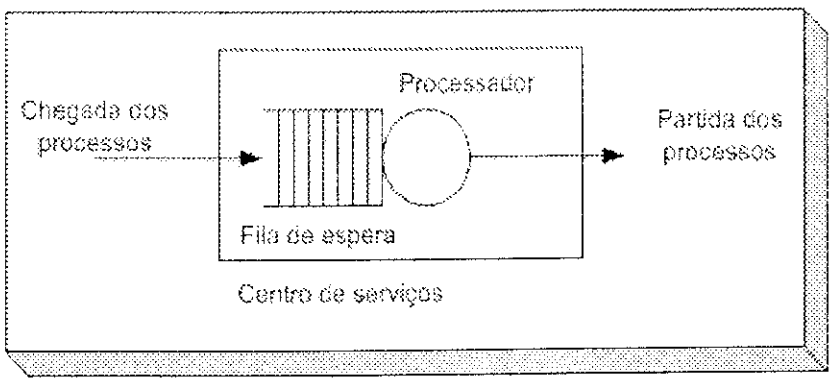


Figura 2.2: Rede de Filas Ilustrando Processos Competindo pelo Processador.

Redes de Petri

Redes de Petri é uma técnica que permite a representação de sistemas utilizando uma forte base matemática. Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos (Maciel *et al.*, 1996).

A representação gráfica de uma rede de Petri básica, ilustrando a mesma situação mostrada na Figura 2.2 através de redes de filas, é apresentada na Figura 2.3. O gráfico possui três componentes: um ativo chamado **transição** (barra), outro passivo denominado **lugar** (círculo) e os **arcos direcionados**, que são elementos de ligação entre os dois primeiros. Os lugares equivalem às variáveis de estado e as transições correspondem às ações realizadas pelo sistema. Os arcos podem ser únicos ou múltiplos.

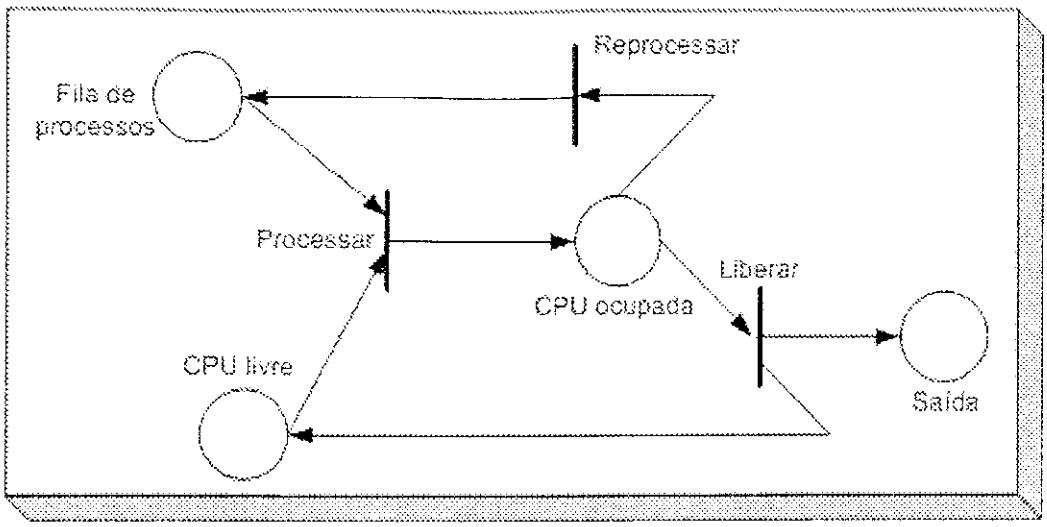


Figura 2.3: Uma Rede de Petri Ilustrando Processos Competindo pelo Processador.

Statecharts

Statecharts é uma extensão das máquinas de estado finito, que possibilita a representação de hierarquia, concorrência e comunicação entre os diversos estados de um determinado sistema (Harel, 1987). Essa técnica tem a finalidade de especificar sistemas reativos, ou seja, sistemas que devem reagir a estímulos externos e internos, normalmente sob condições críticas em relação ao tempo (Harel & Politi, 1998).

A definição de *statecharts* é fundamentada em conjuntos de estados, eventos e condições primitivas, transições e variáveis. A partir disso o modelador pode especificar os valores das variáveis do sistema em um certo instante. A idéia principal é seu uso na representação de sistemas complexos, onde os diagramas de estado apresentam deficiências. Sistemas complexos requerem uma estrutura de representação hierárquica (com agrupamento e refinamento de estados) e de concorrência, de maneira que seja facilmente visível o movimento através dos estados do sistema no decorrer do tempo. A Figura 2.4 apresenta o exemplo anteriormente expresso em redes de filas e de Petri, agora ilustrado em *statecharts*.

2.2.3 Soluções para o Modelo

Depois de definida a técnica de análise para a construção do modelo do sistema em estudo, chega-se ao momento de escolher a solução a ser dada. Existem, basicamente, duas técnicas para a resolução de modelos: analítica e por simulação. Ambas apresentam vantagens e desvantagens, dependentes do tipo de modelo a ser resolvido, o que é determinante para a escolha de uma delas.

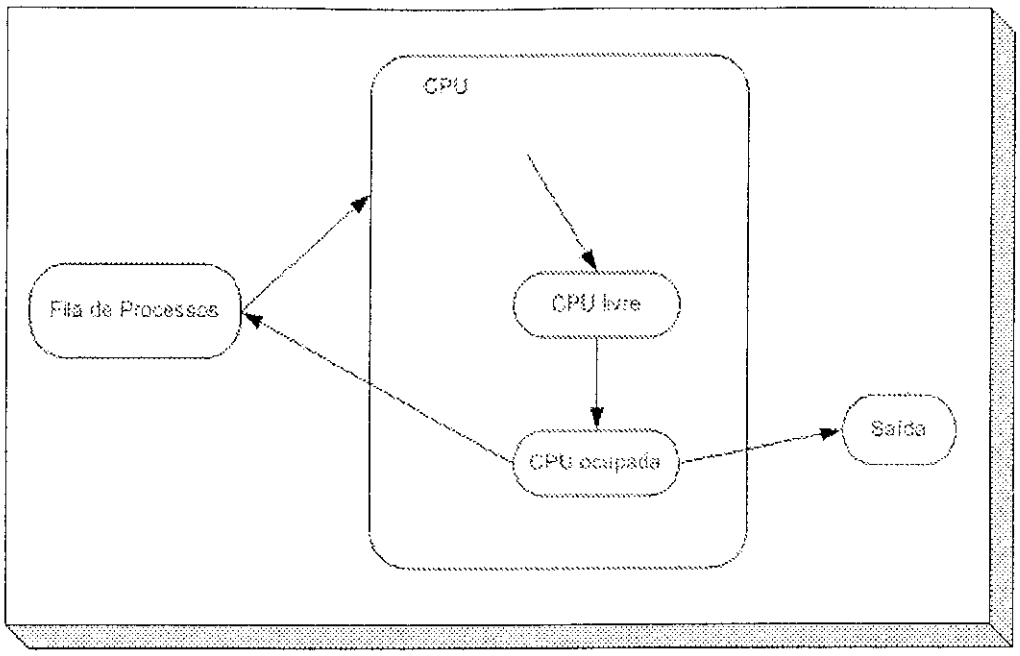


Figura 2.4: Processos Competindo por um Processador (*statecharts*).

Solução Analítica

Por ser o método mais rápido, a solução analítica é a preferida, mas nem sempre é aplicável (Kleinrock, 1976), porque são necessárias simplificações no modelo para que este possa ser resolvido. Simplificações nada mais são do que restrições impostas pela solução analítica, que geralmente não correspondem ao verificado em sistemas reais. Apesar de ser um método que propicia resultados expressivamente exatos, essas restrições podem levar a um modelo que não representa o sistema real de forma adequada, o que torna sem sentido a utilização desse modelo.

Para exemplificar a dificuldade imposta por essas simplificações, na solução analítica de modelos baseados em redes de filas, não há como estabelecer prioridades nas disciplinas das filas. Dessa forma, todos os usuários são tratados de forma idêntica, o que inviabiliza, por exemplo, a representação de esquemas de prioridades presentes em servidores Web que prestam serviços de forma diferenciada. Ainda existem várias outras restrições como a possibilidade de filas infinitas, impossível em sistemas reais.

Nessas situações em que a solução analítica apresenta dificuldades pode-se adotar a resolução por simulação.

Simulação

A possibilidade de simular o comportamento de um objeto (ou um sistema de objetos) constitui uma necessidade presente em diversas áreas do conhecimento. Se o modelo proposto para o sistema envolve uma gama de informações e/ou exige que se façam algumas

simplificações, o modelador pode optar por resolver o modelo através de simulação.

Em computação, a simulação refere-se ao emprego de um programa para implementar um modelo de algum fenômeno ou sistema dinâmico (sistemas cujos estados se alteram com o tempo) (Orlandi, 1995).

A simulação tem sido muito empregada devido a alguns fatores que lhe são amplamente favoráveis, por exemplo: flexibilidade, facilidade de utilização e custo relativamente baixo. As utilizações da simulação são as mais diversas possíveis, destacando-se: aplicações em áreas médicas, em engenharia aeronáutica, de transportes, nuclear, etc. Dentre as aplicações da simulação na área da computação, é interesse básico a avaliação de desempenho dos sistemas computacionais, em particular dos sistemas distribuídos.

A Figura 2.5 apresenta o inter-relacionamento entre as duas soluções de um modelo, assim como anteriormente, as formas determinam quando, preferencialmente, utilizar uma solução em detrimento da outra.

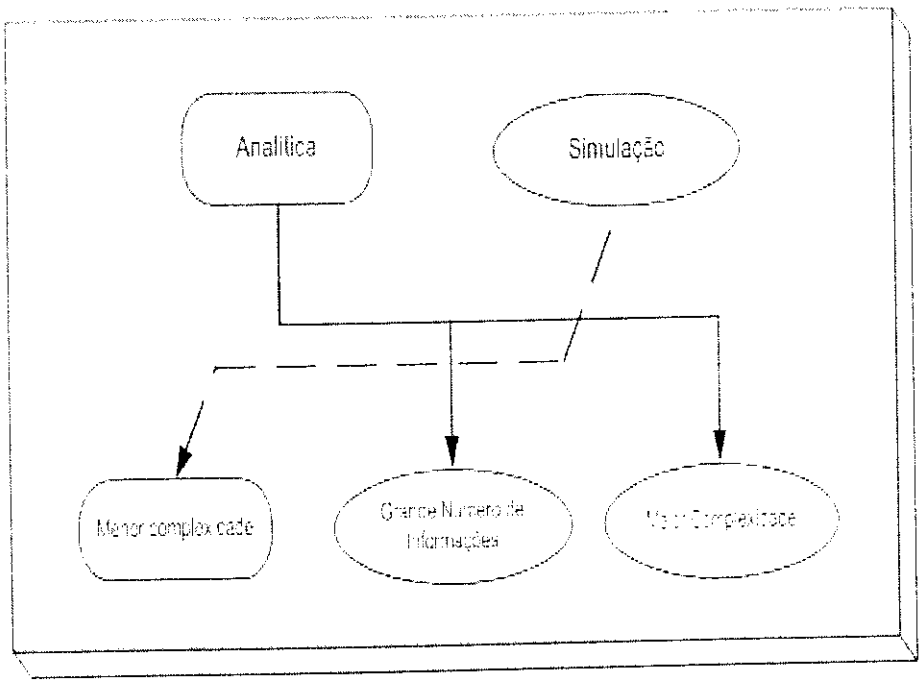


Figura 2.5: Solução Preferencial para um Modelo.

2.3 Considerações Finais

A escolha de uma técnica para a análise de um sistema computacional (distribuído ou centralizado) está diretamente relacionada ao estado de desenvolvimento do sistema que se deseja avaliar. As técnicas de aferição são bastante atrativas para sistemas já existentes ou em fase final de implementação. É pouco provável que, no atual estado da arte dos ambientes computacionais, alguém corra o risco de desenvolver um sistema para depois

analisá-lo: nesse caso, técnicas de modelagem devem ser adotadas.

Outro aspecto a ser considerado após a escolha da técnica de modelagem é a solução do modelo: analítica ou por simulação. Apesar da decisão ficar a critério do modelador, há pontos que devem ser considerados. O principal deles é relativo ao grau de complexidade do modelo a ser resolvido, isto é, para modelos menos complexos é mais recomendável a utilização de soluções analíticas, pois são situações que, normalmente, admitem alguns tipos de restrições. Para modelos mais complexos, é mais recomendável a utilização de simulação. Ainda pode-se levar em consideração a quantidade de informações que o modelo manipula. Para um número menos significativo de informações, a solução analítica é mais apropriada. Para modelos que manipulam uma gama maior de informações, é mais recomendável o uso de simulação.

Para o caso particular de instituições que precisam verificar o desempenho de seus servidores Web, a consideração a respeito da fase de desenvolvimento do sistema não é necessária - o sistema já existe e se encontra em funcionamento. Assim, as técnicas de aferição (Protótipos, *benchmarks*, Coleta de Dados) podem ser empregadas diretamente. Embora seja uma técnica de aferição, os protótipos se aplicam melhor no caso de sistemas inexistentes ou que se apresentam em fase de construção. Outra possibilidade para esse caso é a utilização da Coleta de Dados. Como já visto, essa técnica é a que oferece os resultados mais precisos. Entretanto, ela também apresenta problemas. Um ponto importante a se destacar é que, como os dados são coletados durante a execução do sistema, cada coleta apresenta um resultado diferente. Isso porque a utilização real de um sistema, vista por um período de tempo, nunca se repete exatamente da mesma forma. Com isso, a impossibilidade de se reproduzir as medições pode se tornar um problema.

Por fim, a utilização de *benchmarks* representa mais uma maneira de se avaliar sistemas existentes e, como as outras, apresenta suas vantagens e desvantagens. Os *benchmarks* podem alcançar resultados bastante precisos e, por utilizar cargas de trabalho definidas por meio de parâmetros fixos, a reprodução de seus resultados não é um problema. Mais um ponto a se destacar é que, para a Coleta de Dados, ainda é necessária a utilização de monitores, sejam eles de *hardware* ou *software*. Dependendo dos dados que se pretende coletar, obter esses monitores pode não ser trivial. Já os *benchmarks* se apresentam em diversas ferramentas, algumas delas disponíveis publicamente e de fácil aquisição.

Capítulo 3

Visão Geral da Web

3.1 Introdução

Em 1989, Tim Berners-Lee, físico ligado ao CERN (*European Organization for Nuclear Research*), apresentou uma proposta para o gerenciamento das informações geradas por aquela instituição. Além de um melhor gerenciamento, houve uma preocupação em facilitar a colaboração entre grandes grupos internacionais de pesquisadores que utilizavam documentos de tipos variados e em constante mutação (Tanenbaum, 2002).

A idéia inicial da Web (*World-Wide-Web*), consistia de uma teia de informações que poderiam conter ligações (referências) com outras informações relacionadas. Posteriormente, cada ponto dessa teia receberia o nome de página. As ligações, também conhecidas como *links*, poderiam ser criadas sem preocupação com qualquer tipo de organização pré-estabelecida, ou seja, uma página poderia referenciar outra que poderia estar em qualquer lugar, conter qualquer tipo de informação, etc. Cada informação poderia ser constituída de texto, imagem, ou até mesmo de programas executáveis.

Depois de algum tempo passou-se a utilizar o termo hipertexto, que nada mais é do que texto com ligações para outras informações. Com esses documentos, as informações poderiam ser "seguidas" apenas com cliques do mouse. O hipertexto é uma maneira de organizar documentos de forma em que fique mais simples encontrá-los.

A Web não possui fronteiras no sentido de que um usuário pode ver toda a informação contida nela, como um grande documento hipertexto. Não há a necessidade de saber onde a informação está armazenada, ou qualquer detalhe de sua formatação ou organização. Por trás dessa aparente simplicidade, a Web possui um conjunto de conceitos, protocolos e convenções (seção 3.3) que proporcionam, ao usuário, os recursos mencionadas anteriormente.

Finalmente, para a confecção das páginas, Berners-Lee criou uma linguagem denominada HTML (W3C, 1999). Essa linguagem possibilitou a escrita de documentos em um

modo padronizado, o que veio a facilitar a apresentação das páginas por *browsers*, que constituem programas utilizados para se ter acesso aos documentos na Web.

3.2 Infraestrutura

Desde o seu surgimento, a Web apresentou um crescimento espantoso, tornando-se rapidamente o serviço mais utilizado na Internet. A Web superou, em geração de tráfego na Internet, serviços já tradicionais como FTP, *Telnet* e *e-mail*. Para muitos usuários, principalmente os leigos, a Web é a própria Internet.

A Internet, por sua vez, foi concebida no final dos anos 60, como um sistema *peer-to-peer*. O objetivo da ARPANET original era compartilhar recursos de computação ao redor dos Estados Unidos. Os primeiros *hosts* conectados à ARPANET eram sistemas de computação independentes e foram conectados não em um esquema mestre/escravo ou cliente/servidor, mas como pares (*peers*) equivalentes.

Serviços como FTP, *Telnet* e a Web alteraram o conceito inicial da Internet, porque são basicamente aplicações que funcionam de acordo com o esquema cliente/servidor. Um cliente *Telnet* precisa se conectar a um computador servidor; e um cliente FTP recebe e envia arquivos a um servidor FTP.

A Web pode ser entendida como um gigantesco sistema de hipertexto em escala global. A infraestrutura dos dados na Web é basicamente um conjunto de documentos, páginas, escritos em HTML. As informações encontram-se em repositórios (servidores) espalhados ao redor do mundo e os usuários têm acesso a elas a partir dos *browsers* (clientes). Dessa forma, a Web utiliza o esquema cliente/servidor, assim como FTP e *Telnet*, sobre a infraestrutura da Internet.

O modelo de interação na Web baseia-se em um paradigma de requisição/resposta. Assim, quando da utilização da Web, o *browser* - ou navegador - inicia uma conexão com o servidor, faz uma requisição, recebe dados (resposta) e desconecta. O modelo é simples e permite ao usuário navegar livremente para assistir vídeos, ouvir música, ler notícias, participar de jogos interativos e várias outras possibilidades. A tecnologia da Web foi concebida para funcionar em ambientes heterogêneos, considerando-se *hardware* e *software*. Para um servidor Web, é indiferente a plataforma em que o navegador está sendo executado e vice-versa; ele apenas retorna a informação solicitada e o navegador se encarrega de apresentá-la ao usuário. Um cliente não precisa ter um endereço permanente; não precisa ter uma conexão contínua com a Internet; ele precisa apenas saber como fazer uma pergunta e ouvir a resposta.

Na Web, particularmente, a portabilidade é garantida através da utilização de dois padrões em particular:

- a linguagem HTML: define um método para a confecção de páginas. Permite ao editor do documento definir sua estrutura, formatação e estabelecer ligações com outros documentos na Web. Os navegadores são capazes de ler um documento HTML, interpretá-lo e então gerar uma apresentação deste documento ao usuário.

- o protocolo HTTP: a comunicação entre os clientes (navegadores) e os servidores é feita utilizando-se o HTTP (seção 3.3.3) que funciona sobre outro protocolo conhecido como TCP (seção 3.3.2).

3.3 Protocolos

Para que o processo de comunicação entre os diversos sistemas (*hardware* e *software*) existentes em uma rede de computadores seja possível, é necessária a existência de um conjunto de regras e convenções que permitam disciplinar a troca de informações. Essas regras comuns constituem os chamados protocolos de comunicação.

Diferentes empresas começaram a desenvolver protocolos proprietários para fazer a comunicação entre seus produtos. Dessa forma, surgiram muitos protocolos distintos que não se comunicavam um com o outro. Não tardou para que fosse percebida a necessidade de um protocolo padrão, através do qual todos poderiam se comunicar.

O conjunto de protocolos TCP/IP veio ao encontro desse propósito e é atualmente a maneira mais utilizada para a comunicação em redes de computadores. Utilizado já há um bom tempo nos EUA, o TCP/IP foi implementado pelo DoD (*Departament of Defense*) com o fim de integrar seus computadores em uma única rede, a Internet.

O TCP/IP foi desenvolvido para permitir a interoperabilidade de serviços entre computadores ligados a redes físicas baseadas em tecnologias distintas. Entre as tecnologias abrangidas incluem-se: redes locais de diferentes tipos, conexões seriais ponto a ponto e redes X.25. O modelo admite a interligação direta ou indireta de um número arbitrário de redes distintas, e exige de uma rede participante o suporte para um serviço de entrega de pacotes. A partir dessa base, cria-se uma rede virtual utilizando um esquema global de endereçamento (IP - seção 3.3.1) e oferecendo vários tipos de serviços: serviços orientados à conexão, utilizando TCP (seção 3.3.2), e serviços sem conexão, utilizando UDP.

3.3.1 IP

Os serviços Internet, orientados ou não à conexão, estão construídos sobre um sistema de entrega de pacotes cuja unidade básica de transferência é o datagrama IP (Tanenbaum, 2002), ou simplesmente datagrama. O datagrama é dividido em cabeçalho e área de dados. O cabeçalho (Figura 3.1) possui uma parte fixa de 20-bytes e outra parte opcional de tamanho variável.

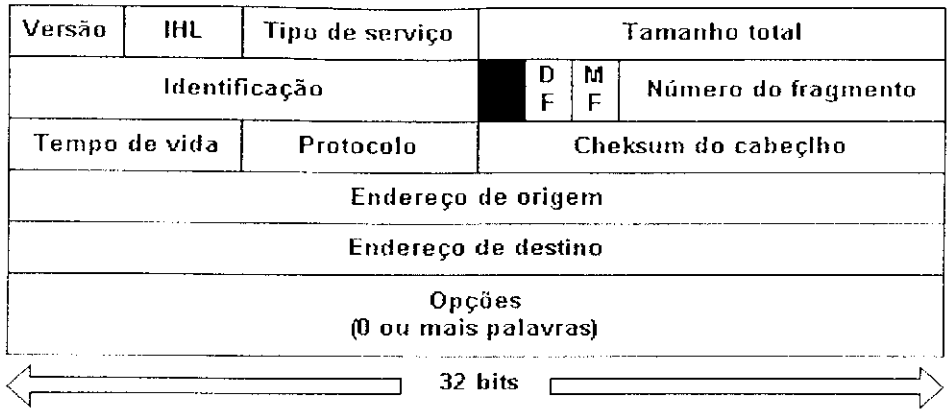


Figura 3.1: Cabeçalho IP

O campo **Versão** indica a qual versão do protocolo o datagrama pertence. Pela inclusão da versão em cada datagrama, é possível haver transições entre versões que durem meses, até anos, com algumas utilizando a versão antiga e outras, a nova.

Como o tamanho do cabeçalho não é constante, o campo **IHL** é usado para especificar esse tamanho em palavras de 32-bits.

O campo **Tipo de serviço** permite que o host indique à sub-rede o tipo de serviço desejado. Várias combinações de confiabilidade e velocidade são possíveis. Para a transmissão de arquivos, por exemplo, uma transmissão sem erros é mais importante do que uma transmissão rápida.

Tamanho total indica o tamanho total do datagrama - incluindo cabeçalho e dados. O tamanho máximo é de 65.535 bytes.

Identificação é necessário para permitir à máquina destino determinar a qual datagrama o fragmento recém chegado pertence.

Em seguida há um campo, não utilizado, de um bit e mais dois campos, **DF** e **MF**, também de um bit. **DF** significa "não fragmentar" (*Don't Fragment*). Ele funciona como uma ordem aos roteadores para que não fragmentem o datagrama pois a máquina destino não seria capaz de montá-lo novamente.

MF significa "mais fragmentos" (*More Fragments*). Todos os fragmentos, exceto o último, possuem esse bit setado (com valor 1). Isto é necessário para saber quando todos os fragmentos do datagrama chegaram.

O campo **Número do fragmento** (*Fragment offset*), diz em que lugar do datagrama o fragmento se encaixa.

Tempo de vida, (*Time to live*), é um contador que limita o tempo de vida dos pacotes. Ele deve ser decrementado a cada *hop* (roteador por onde passa). Quando ele chega a zero, o pacote é descartado e um pacote de aviso é enviado a máquina de origem.

Quando a camada de rede monta um datagrama completo, ela precisa saber o que fazer com ele. O campo **Protocolo** indica qual processo de transporte dar ao datagrama. TCP é uma possibilidade, mas UDP e vários outros também podem ser adotados.

O **Checksum do cabeçalho**, (*Header checksum*), é utilizado para a verificação dos dados do cabeçalho do pacote.

Os campos **Endereço de origem** e **Endereço de destino** indicam respectivamente o endereço IP de origem e destino do pacote.

O campo **Opções** é utilizado em testes para a experimentação de novas idéias. Seu tamanho é variável.

Endereço IP

Cada máquina (*host*) e roteador na Internet tem um endereço IP, que codifica seu número de rede e número de host. A combinação é única: não existem duas máquinas com o mesmo IP. Os endereços são de 32 bits e utilizados nos campos *Source address* e *Destination address* dos pacotes.

Alguns endereços possuem significado especial, como mostrado na Figura 3.2. O valor 0 significa "esta rede" ou "este host". O valor 1 é usado como endereço de *broadcast* para referenciar todos os hosts de uma determinada rede.

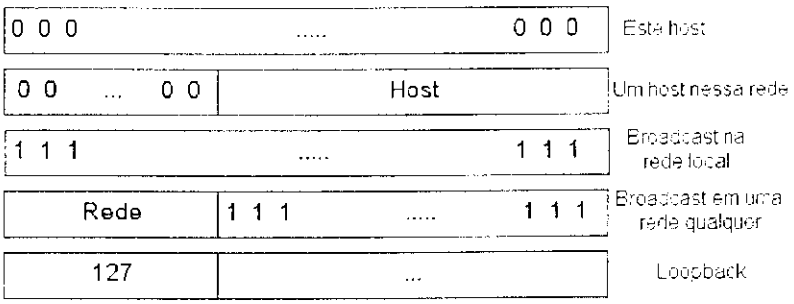


Figura 3.2: Endereços IP especiais

O endereço IP 0.0.0.0 é usado pelos hosts em sua iniciação e descartado depois disso. Um IP com o valor 0 para rede referencia a rede atual. Esses endereços permitem às máquinas referenciarem sua própria rede sem saber seu número. O IP consistindo apenas de 1s permite *broadcast* na rede local. Um endereço com um número de rede apropriado e apenas 1s no campo de host permite às máquinas enviarem pacotes em *broadcast* para redes em qualquer lugar da Internet. Finalmente, todos os endereços da forma 127.xx.yy.zz são reservados para testes de *loopback*. Pacotes enviados para esses endereços não são colocados na rede; eles são processados localmente e tratados como pacotes vindos da rede.

3.3.2 TCP

O TCP(*Transition Control Protocol*) (Comer, 2000; Tanenbaum, 2002) foi projetado especificamente para prover um serviço confiável de transmissão de dados sobre uma rede não confiável.

Ele é um protocolo orientado a conexão, o que implica no estabelecimento desta entre os pontos finais de comunicação antes que qualquer dado possa ser enviado.

As entidades TCP, receptora e emissora, trocam dados na forma de segmentos. Um segmento consiste de um cabeçalho fixo de 20 bytes (mais uma parte opcional) seguido de zero ou mais bytes de dados. Dois limites restringem o tamanho dos segmentos. Primeiro, cada segmento, incluindo o cabeçalho, deve caber em um pacote IP que possui 65.535 bytes. Segundo, cada rede possui uma "Unidade Máxima de transferência" (**MTU**) e cada segmento deve caber em um MTU.

O protocolo básico utilizado pelas entidades TCP é conhecido como "Protocolo da janela deslizante" (*Sliding Window Protocol*). Quando o emissor transmite um segmento, ele inicia um contador. Quando o segmento chega ao destino, a entidade receptora envia de volta outro segmento, com ou sem dados, contendo um número de confirmação (**ACK**) igual ao número do próximo pacote a ser recebido. Se o contador do emissor se esgotar antes da recepção do **ACK**, o emissor envia novamente o pacote.

Cabeçalho TCP

A Figura 3.3 mostra um segmento TCP. Cada segmento possui um cabeçalho que pode ser seguido por opções. Após essas opções, pode existir até 65.535 - 20 - 20 = 65.495 bytes de dados, onde os primeiros 20 referem-se ao cabeçalho IP e os 20 bytes seguintes ao cabeçalho TCP.

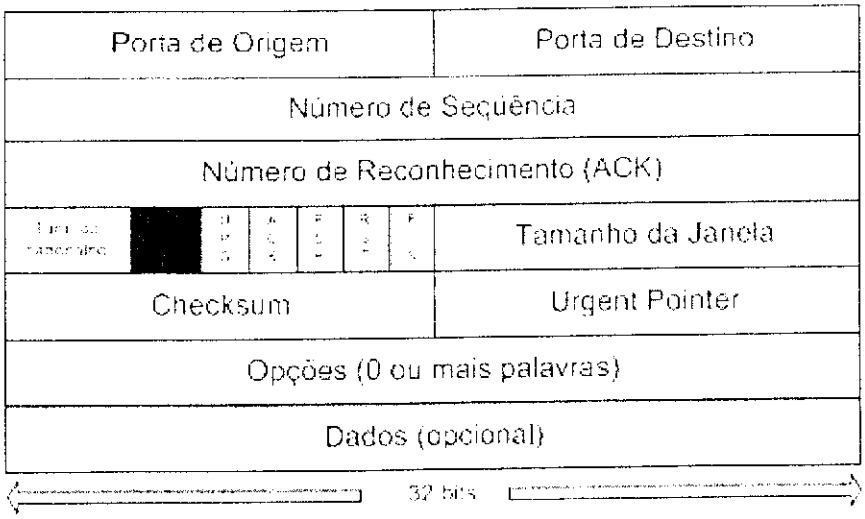


Figura 3.3: Cabeçalho TCP

Os campos **Porta de origem** e **Porta de destino** identificam os pontos finais da conexão.

Número de seqüência identifica a posição do segmento em toda a seqüência e o **Número de reconhecimento**, (*Acknowledgement number*), indica o próximo segmento esperado.

Tamanho do cabeçalho indica o tamanho, em palavras de 32 bits, do cabeçalho.

Na seqüência aparecem seis *flags* de um bit. *Urgent Pointer* (URG) é usado para indicar onde os dados urgentes estão. **ACK** indica quando o *Acknowledgement number* é válido. O campo **PSH** é utilizado para indicar ao receptor que entregue os dados à aplicação imediatamente, não utilizando buffers. O bit **RST** é usado para reiniciar a conexão que se tornou confusa devido a algum problema. **SYN** é usado para estabelecer conexões. Uma requisição de conexão possui **SYN** = 1 e **ACK** = 0. O bit *FIN* libera a conexão. Ele indica que o emissor não possui mais dados para enviar.

O controle de fluxo no TCP é feito através de uma "janela deslizante" de tamanho variável. O campo **Tamanho da janela** especifica quantos bytes podem ser enviados em um segmento.

Checksum é utilizado para prover alta confiabilidade. Ele verifica o cabeçalho e os dados.

Finalmente, o campo **Opções** é utilizado para prover maneiras de adicionar funcionalidades extras não existentes no cabeçalho regular.

3.3.3 HTTP

O protocolo HTTP (*Hypertext Transfer Protocol*) é o elemento que une as partes que compõem a Web. Toda e qualquer informação que trafega entre os *browsers* e os servidores web o faz dentro de uma mensagem HTTP.

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

É importante destacar que, embora - sob o ponto de vista do usuário - uma solicitação de página HTML se resuma muitas vezes a apenas um clique do mouse, na verdade, ela freqüentemente dá origem a várias solicitações HTTP que são enviadas do *browser* para o servidor web (seguidas de suas respectivas respostas no sentido contrário). Isso

ocorre porque, para cada objeto contido em uma página HTML, é gerada uma requisição independente ao servidor. No protocolo HTTP 1.0, isso implica em estabelecer uma nova conexão TCP para cada objeto solicitado, o que pode levar a uma sobrecarga desnecessária no servidor e na rede.

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

Mensagens de Requisição e Resposta

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

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

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

Figura 3.4: Exemplo de uma requisição HTTP

No exemplo da Figura 3.4, o cliente solicita (GET) o arquivo /docs/arq.html do servidor www.empresa.com usando o protocolo HTTP 1.1. O campo *Accept* no cabeçalho informa que o cliente sabe como tratar texto em formato HTML (text/html) e imagens em formato JPEG (image/jpeg). O campo *User-Agent* informa ao servidor qual o tipo do *browser*.

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

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

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

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

Figura 3.5: Exemplo de uma resposta HTTP

A Figura 3.5 mostra um exemplo de uma resposta HTTP 1.1 válida. O código 200 reporta que a requisição foi bem-sucedida. O campo *Server* informa o tipo do servidor web (CERN 3.0) e o campo *Content-Type* diz que o objeto retornado é um documento HTML, cujo tamanho é de 150 bytes (*Content-Length*). Finalmente, tem-se uma linha em branco e o documento propriamente dito.

HTTP 1.0 e 1.1

O protocolo HTTP 1.0, definido na RFC 1945 (Berners-Lee *et al.*, 1996), foi introduzido juntamente com a Web, em 1990. Nesta versão inicial, ele nada mais era que um modo simples de recuperar dados através da Internet. Com o crescimento da Web, surgiram novos requisitos e algumas de suas fraquezas foram aparecendo.

A primeira delas é que o HTTP 1.0 exige o estabelecimento de uma nova conexão TCP para cada objeto solicitado. No início, em que os documentos da Web eram constituídos basicamente de texto, isso não chegava a ser um problema, porém, atualmente, em que uma simples página HTML pode conter dezenas de pequenas imagens, isso tende a causar uma grande sobrecarga no tráfego da Internet, bem como nos servidores. O protocolo HTTP 1.1, definido na RFC 2616 (Fielding *et al.*, 1999) e padronizado em 1999 pelo W3C (*World Wide Web Consortium*), usa como default o esquema de conexões persistentes,

que permite que uma mesma conexão TCP seja usada em várias transações HTTP, o que é bem mais eficiente.

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

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

3.4 Caracterização de Carga da Web

O desempenho de um sistema distribuído com muitos clientes, servidores e redes, depende em muito das características de sua carga. Assim, o primeiro passo em qualquer estudo de avaliação de desempenho é o entendimento e a caracterização de sua carga (Menascé & Almeida, 1998; Fonseca *et al.*, 2003). A carga de um sistema pode ser definida como o conjunto de todas as entradas que recebe, durante um intervalo de tempo determinado, de seu ambiente. Por exemplo: considere um Banco de Dados (BD) observado durante 1 hora onde ocorreram 70.000 transações. A carga no BD durante aquela hora é o conjunto das 70.000 transações e as características dessa carga são representadas por um conjunto de informações (tempo de CPU, número de operações de I/O, etc.) para cada uma das 70.000 transações.

Fica clara a dificuldade de lidar com cargas reais que possuam grande número de elementos. Entretanto, para trabalhar com problemas práticos, é preciso reduzir a informação necessária para descrever a carga. Em outras palavras, é necessário construir um modelo de carga que contenha as características mais relevantes da carga real.

A escolha das características e parâmetros que irão descrever a carga depende do propósito do estudo. Se a intenção é o estudo da relação Custo/Benefício de se criar um "cache proxy" para um *site*, então as características podem ser a freqüência de referência a um documento, seu tamanho, etc. Entretanto, se o interesse está em determinar o impacto de uma CPU mais veloz no tempo de resposta de um servidor Web, as informações utilizadas devem ser outras.

Ainda que alguns sistemas exijam um método específico para a caracterização de sua carga, há uma metodologia (Menascé & Almeida, 1998) que pode ser utilizada, de maneira geral, na caracterização de qualquer tipo de carga (Calzarossa & Serazzi, 1993; Arlitt, 1996; Vallamsetty, 2003).

3.4.1 Metodologia para a caracterização de carga

Escolha de um ponto de análise

A carga global de um sistema distribuído é a combinação de diferentes cargas "vistas" por diferentes componentes do sistema. Para uma máquina cliente, em que um usuário manipula um *browser*, a carga apresentada são os cliques dados pelo usuário e as respostas vindas dos servidores Web. Já do ponto de vista do servidor, a carga constitui-se das requisições HTTP que chegam dos clientes. Na visão da rede, a carga são os pacotes que circulam por dela.

Identificação dos componentes básicos

Neste passo, são identificados os componentes básicos que compõem a carga. **Transações** e **Requisições** são os componentes mais usuais. A escolha dos componentes depende tanto da natureza do sistema como do objetivo da caracterização. Em um servidor Web, por exemplo, a carga é composta das requisições que chegam dos clientes.

Escolha dos parâmetros de caracterização

Uma vez que os componentes tenham sido identificados, o próximo passo é escolher quais parâmetros caracterizam cada tipo de componente básico. Os parâmetros podem ser separados em dois grupos: um diz respeito à intensidade da carga e o outro à demanda de serviço dos componentes. Em cada grupo pode-se obter as seguintes informações:

- Intensidade da carga:
  - taxa de chegada
  - número de clientes e tempo de "pensar" (*think time*)
  - número de processos ou *threads* em execução simultaneamente
- Demanda de serviço, especificada por k-tuplas $(D_{i1}, D_{i2}, ..., D_{ik})$, onde k é o número de recursos considerados, e D_{IJ} é a demanda de serviço do componente I no recurso J. Por exemplo, a 2-tupla (CPU, I/O) da forma (0,0095s, 0,04s).

Coleta de dados

Este passo especifica valores para os parâmetros de cada componente do modelo. São geradas tuplas de caracterização de acordo com o número de componentes da carga. A coleta de dados inclui as seguintes tarefas:

- Identificar os intervalos de tempo (*time windows*), que definem as sessões de medida. A observação contínua de um sistema durante dias ou semanas, permite ao analista captar os intervalos de tempo apropriados nos quais possa basear seus estudos.
- Monitorar e medir as atividades do sistema durante intervalos de tempo definidos.
- Dos dados coletados, definir valores para cada parâmetro de caracterização de cada componente da carga.

Partição da carga

Cargas reais podem ser vistas como uma coleção de componentes heterogêneos. Em relação ao nível de utilização de recursos, a requisição por um vídeo difere muito de uma requisição por um pequeno documento HTML. Por essa heterogeneidade, representar uma carga através de uma simples classe, significa perder tanto a representatividade da caracterização quanto o poder de previsão do modelo. Técnicas de partição são utilizadas para dividir a carga em classes, de forma que sejam formadas por componentes homogêneos. A seguir são detalhados alguns atributos para a partição de carga.

Utilização de recurso - O consumo de recursos por componente pode ser usado para quebrar a carga em classes. No caso da Web seria possível dividir as requisições HTTP segundo seu consumo esperado de CPU e I/O. A Tabela 3.1 mostra um exemplo de classes desse tipo.

Transação	Freqüência	Tempo máx. de CPU (ms)	Tempo máx. de I/O (ms)
Trivial	40%	8	120
Leve	30%	20	300
Média	20%	100	700
Pesada	10%	900	1200

Tabela 3.1: Partição de carga baseada em utilização de recursos

Aplicações - Um agrupamento de acordo com a aplicação também pode ser aplicado. Se pretende-se estudar o tráfego na Internet por exemplo, pode-se quebrá-lo em classes como Web, telnet, FTP e outras.

Objetos - Pode-se dividir a carga de acordo com o tipo de objeto tratado pela aplicação. Na Web, encontram-se requisições de objetos que vão desde simples texto ou páginas HTML a imagens, audio, vídeo, documentos formatados, páginas dinâmicas e outros. Uma requisição de uma página HTML de alguns poucos kilobytes impõe uma demanda sobre um servidor web que é muito diferente daquela originada, por exemplo, pelo *download* de um vídeo com alguns megabytes.

Origem geográfica - Devido aos atrasos presentes em redes de grandes dimensões (WANs), torna-se importante fazer uma distinção entre transações ou requisições que são

servidas local ou remotamente. Essa separação poderia ser determinada analisando-se os endereços IP dos clientes.

3.4.2 Cargas em rajadas

Novas características têm sido observadas em sistemas distribuídos de grandes escalas como a Internet e a Web (Menascé & Almeida, 1998). Diversos estudos (Arlitt & Williamson, 1996; Crovella & Bestavros, 1997; Scheuermann *et al.*, 1997) revelaram importantes propriedades do tráfego em redes, tal como auto-similaridade, novas políticas de cache e cargas em rajadas(*bursty workloads*). Em Leland et al. (1993), é mostrado que o tráfego total (medido em bytes/s ou pacotes/s) em LANs e WANs é auto-similar. A Figura 3.6 plota um típico comportamento auto-similar. De acordo com (Crovella & Bestavros, 1997), as rajadas de tráfego são visíveis mesmo quando o intervalo de tempo é alterado para 100, 10, ou 1 segundo. O mais importante é que a auto-similaridade do tráfego na rede (Internet, WWW) implica na presença de rajadas em diversas escalas de tempo. Além disso, essa característica tem um grande impacto no desempenho de sistemas que funcionam em rede.

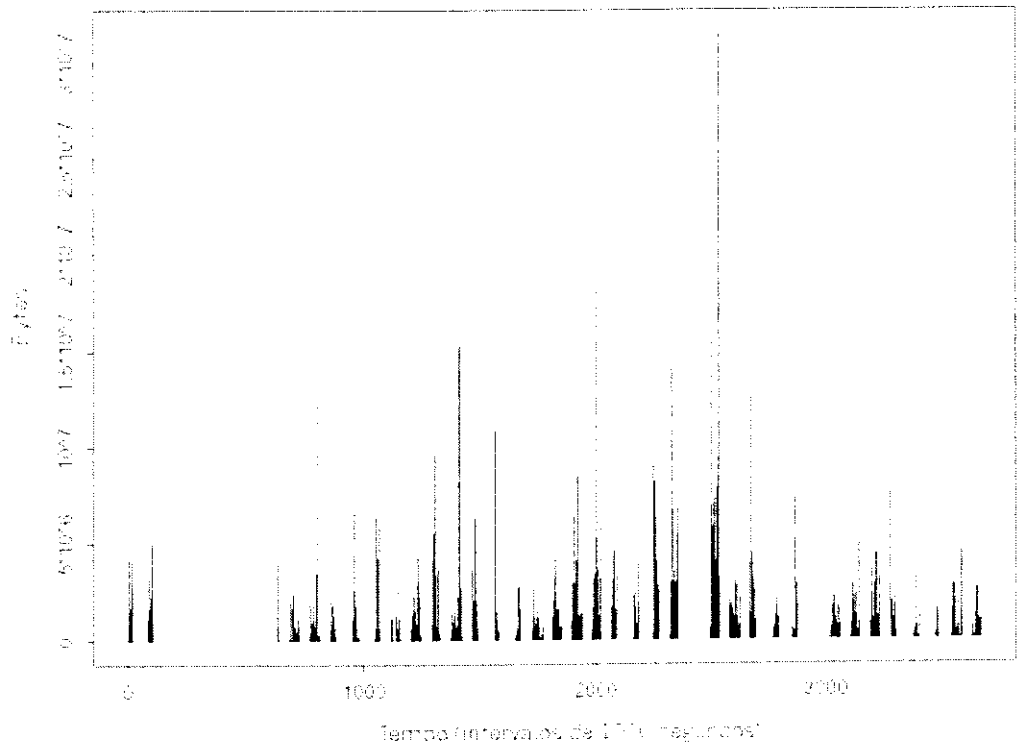


Figura 3.6: Tráfego da Web em rajadas. (Crovella & Bestavros, 1997)

Na Web, as requisições HTTP são geradas por um grande número de clientes. Cada

um desses clientes apresenta um "tempo de pensar" com grande variância. Além disso, o "tempo de pensar" de cada um não é independente; fatores como padrões comportamentais humanos e a publicação de conteúdos na Web de forma agendada causam uma grande correlação entre as requisições geradas pelos clientes. Como resultado, o tráfego de requisições chega ao servidor em rajadas, e essa característica pode ser verificada em diversas escalas de observação (Crovella & Bestavros, 1997) e com picos que excedem a taxa média em fatores de 8 a 10 vezes (Stevens, 1996). É importante ressaltar que esses picos de requisições podem exceder facilmente a capacidade do servidor, por isso a importância da inclusão dessa característica no estudo de servidores Web.

3.5 Arquitetura de Servidores Web

O *software* de um servidor Web, também conhecido como servidor HTTP, são programas que controlam o fluxo de entrada e saída de dados em um computador conectado a uma intranet ou à Internet. Basicamente, o *software* aguarda requisições HTTP vindas de clientes na rede. O programa servidor estabelece a conexão entre ele e o cliente, envia o arquivo requisitado e retorna ao estado de espera. Com o propósito de melhorar o serviço, os servidores HTTP incorporaram diversas inovações que aconteceram ao longo do tempo. Dentre essas inovações pode-se destacar as estratégias de concorrência, que estão sempre em foco quando o assunto é desempenho (Hu *et al.*, 1997). De acordo com essas estratégias, os servidores podem ser classificados como (Hu *et al.*, 1998):

Iterativo

A maneira mais simples de se implementar um servidor é não ter concorrência. Esse servidor simplesmente recebe cada requisição e atende uma de cada vez, respeitando a ordem: a primeira que chega é a primeira a ser atendida (FIFO). Esse tipo de servidor não funciona bem, exceto em casos de cargas muito baixas. Por isso ele é muito pouco utilizado atualmente.

Um processo por requisição

A utilização da chamada de sistema **fork()**, para criar um processo novo a cada requisição recebida, é uma estratégia comum de concorrência em servidores Web. No entanto, essa estratégia não funciona bem em servidores com cargas altas porque a sobrecarga da criação de novos processos prejudica o desempenho durante picos de requisições.

Existem várias técnicas para se minimizar essa sobrecarga. O servidor Apache (Apache Software Foundation, 2003), por exemplo, elimina o problema através de um conjunto de processos pré-inicializados (*pre-forked*). Para implementar esse esquema, o Apache

inicializa um certo número de processos antes de receber qualquer requisição. Quando uma requisição HTTP chega, um processo dedicado é escalonado para seu atendimento. Essa estratégia alcança bons resultados para cargas moderadas. Entretanto, durante cargas muito altas, a taxa de requisições ultrapassa a quantidade de processos presentes no conjunto, quando a estratégia volta a funcionar como no início, **um processo por requisição**.

Além disso, servidores com múltiplos processos podem sofrer de mais uma sobrecarga, devido à constante troca de contexto. Por isso, os servidores recentes utilizam apenas um processo de acordo com uma arquitetura dirigida a eventos. Um servidor desse tipo usa outra chamada de sistema, **select()**, para aguardar eventos simultaneamente em todas as conexões que estão sendo manipuladas pelo servidor. Quando **select()** entrega um ou mais eventos, o servidor invoca manipuladores para cada conexão pronta.

Uma *thread* por requisição

Essa técnica é similar à anterior, exceto pelo fato de que são usadas *threads* no lugar de processos. Para cada requisição recebida, uma nova *thread* é criada.

A estratégia que utiliza *threads* é mais eficiente do que a que utiliza processos, pois o custo para se criar uma nova *thread* é muito menor que o de se criar um novo processo. Isso ocorre porque um processo filho requer uma cópia do espaço de endereçamento de seu processo pai, enquanto as *threads* compartilham um espaço de endereçamento com outras *threads* no mesmo processo.

Thread pool

Pode-se alcançar desempenhos melhores podem ser alcançados por meio da mudança do modelo de conjunto de processos para conjunto de *threads*. Nesse modelo, *thread pool*, um conjunto de *threads* é criado na iniciação da máquina. Todas as *threads* ficam bloqueadas no estado **accept**, aguardando requisições dos clientes. Isso elimina a sobrecarga de criar uma nova *thread* para atender cada nova requisição. Essa estratégia é utilizada pelo JAWS (J. Hu, 1998; Silva & Kulesza, 2000).

Entretanto, nem sempre é possível um suporte eficiente para um grande *pool* de *threads*. Por isso alguns servidores utilizam um sistema híbrido em que um *pool* de *threads* de tamanho moderado é multiplexado entre as diversas conexões, utilizando eventos para associá-las às *threads*.

Única *thread*

Threads são uma boa escolha pois funcionam bem em plataformas multiprocessadas. Entretanto, para plataformas com uma única CPU, o uso de *threads* aumenta a sobrecarga

devido a trocas de contexto e sincronização. Assim, a estratégia de um única *thread* funciona melhor. Um servidor com uma única *thread* difere de um interativo no fato de que ele manipula múltiplas requisições ao mesmo tempo. Essa estratégia é mais complicada para implementação devido à utilização de I/O assíncrona. O potencial, entretanto, são servidores mais rápidos para plataformas uniprocessadas.

Cargas dinâmicas

Tudo o que foi discutido acima assume requisições a documentos estáticos. Entretanto, o protocolo HTTP dá suporte também a requisições a documentos dinâmicos cujo conteúdo depende de parâmetros particulares e do tempo de chegada da requisição. Documentos dinâmicos são criados por programas auxiliares que são executados como processos separados. Para tornar mais fácil a construção desses programas auxiliares, foram definidos diversos padrões de interface como o CGI (CGI, 2002) e o FastCGI (FastCGI, 2002). A interface CGI cria um novo processo para manipular cada requisição a documento dinâmico e o FastCGI permite processos persistentes para o tratamento desse tipo de documento.

Há uma tendência para a construção de servidores Web que trabalhem de acordo com uma arquitetura onde um pequeno conjunto de processos implementa a funcionalidade do servidor. Há um processo principal responsável pelo atendimento de todos os documentos estáticos e os documentos dinâmicos são criados por processos auxiliares. Dessa forma, a sobrecarga devido à troca de contexto atinge o ponto ideal, pois fica reduzida ao absolutamente necessário.

3.6 Considerações Finais

Neste capítulo foi apresentado um panorama geral da Web que abordou inicialmente a infraestrutura da Internet e posteriormente os protocolos TCP/IP e HTTP. Nessa primeira parte foram apresentados o surgimento e o método de funcionamento da Web. O protocolo HTTP foi estudado com maior detalhe por ter grande importância no funcionamento e no comportamento da Web.

A caracterização da carga presente na Web foi apresentada através de uma metodologia de caracterização. Cada etapa foi discutida e um exemplo de aplicação prática na Web foi apresentado. Posteriormente foi discutida a importância que cargas auto-similares, que se apresentam em rajadas, possuem no estudo de sistemas distribuídos de grande escala como a Internet e a Web.

Finalmente foi feita uma discussão de algumas das arquiteturas para servidores Web existentes atualmente. Foram apresentados servidores Iterativos, que funcionam de maneira simples e sem qualquer tipo de concorrência; servidores com concorrência em nível

de processo que, por isso, apresentam sobrecarga devido à excessiva troca de contexto; e servidores que utilizam *threads* no lugar de processos e conseguem uma certa vantagem por exigir menos recursos da máquina.

A Web é um sistema distribuído que está em constante mutação. Assim, paradigmas e protocolos utilizados atualmente podem ser descartados em pouco tempo. As arquiteturas de servidores também se enquadram nesse esquema. Uma arquitetura que apresenta bom desempenho hoje, pode perder sua qualidade diante de um novo tipo de carga que se apresente na Web.

Capítulo 4

Benchmarks para Servidores Web

4.1 Introdução

Como descrito no Capítulo 2, os *benchmarks* são programas utilizados para teste de *software*, *hardware* ou sistemas computacionais completos. Por se tratar de uma técnica de aferição para avaliação de desempenho, ela é aplicável apenas em sistemas já existentes ou que estão em sua fase final de desenvolvimento; há ainda a preocupação em garantir a não interferência do *benchmark* no comportamento do sistema.

Devido ao crescimento e à importância alcançados pela Web, os sistemas destinados a trabalhar nessa rede passaram a receber grande atenção. O desempenho de servidores Web se tornou uma preocupação constante em todo sistema que oferece serviços através da Web. Administradores e desenvolvedores desses sistemas procuram respostas para perguntas como: Qual a taxa de transferência de dados alcançada pelo servidor? Quantas conexões podem ser atendidas em um determinado intervalo de tempo? Qual a latência presente nesse servidor? A partir das respostas obtidas pode-se chegar a uma definição de desempenho e capacidade alcançados pelos servidores. Com o intuito de auxiliar na obtenção das respostas citadas, foram propostos diversos *benchmarks* especializados em avaliação na Web (Tittel, 1996; Banga & Druschel, 1999).

Quando se pretende avaliar o desempenho na Web, devem-se considerar três fatores principais: o usuário final que gera as requisições, a infraestrutura da rede que transporta os dados, e o servidor onde estão armazenadas as informações requisitadas.

Todos os *benchmarks* para Web disponíveis atualmente são, na verdade, *benchmarks* para servidores Web. Isso ocorre porque o administrador do servidor (*webmaster*) não é capaz de controlar as configurações e comportamentos do cliente, nem a latência das redes às quais o servidor está conectado. Em outras palavras, toda a informação a respeito de desempenho na Web é, realmente, apenas uma medida de quão bom é o atendimento que um servidor Web pode dar a um conjunto de requisições.

As características mais comuns desses servidores abordadas pelos *benchmarks* são:

- **Latência de resposta**: Tempo necessário para o reconhecimento e atendimento de uma requisição HTTP.
- **Capacidade de atendimento a conexões**: Número máximo de conexões por unidade de tempo que o servidor pode manipular.
- ***Throughput* do servidor**: Quantidade de dados que o servidor pode, por unidade de tempo, enviar e receber.

Existem atualmente diversos *benchmarks* para a avaliação de servidores Web disponíveis na Internet. A seguir são descritos aqueles selecionados para a utilização neste trabalho.

4.2 WebStone

O WebStone é um *benchmark* configurável para servidores HTTP. Ele utiliza parâmetros de caracterização de carga e processos clientes para gerar tráfego HTTP com o intuito de sobrecarregar um servidor de diferentes maneiras (Trent & Sake, 1995; Mindcraft, 2002). O WebStone foi desenvolvido para medir o *throughput* máximo e o tempo médio de resposta. Ele gera requisições GET para documentos específicos no servidor Web e coleta dados de desempenho.

Composto de processos clientes e mestre, o WebStone trabalha de maneira distribuída. O processo mestre gera um número predefinido de processos clientes que, por sua vez, começam a gerar requisições ao servidor. Quando todos os clientes terminam sua execução, o mestre recolhe os dados de desempenho coletados pelos clientes e faz um relatório geral. O usuário pode definir tanto o tempo de duração do teste quanto o número de iterações executadas.

4.2.1 Configuração

Instalação

O WebStone utiliza o mecanismo GNU padrão de instalação, ou seja, para instalá-lo é necessário a execução dos passos: `configure`; `make`; e `make install`. Nesse caso, entretanto, a instalação não ocorreu de forma tranqüila, pois houve problemas na compilação do código fonte. A definição de variáveis em alguns arquivos teve que ser alterada e tudo que fazia referência à API foi tirado do *Makefile*, pois essa parte da compilação necessita de arquivos adicionais. Como não havia o interesse em explorar essa característica nos testes, ela foi simplesmente extraída da ferramenta.

A instalação requer, além da compilação do código fonte, as seguintes configurações:

1. criação de uma conta de usuário específica para a utilização do WebStone.

   O WebStone utiliza uma conta de usuário para executar comandos remotamente nas máquinas clientes. A partir desses comandos, o sistema *webmaster* controla a execução de cada sistema cliente.

2. edição dos arquivos que controlam os comandos-r (.rhosts e /etc/hosts.equiv), para que o usuário criado possa ter acesso a todas as máquinas do teste.

   É necessário acertar a configuração de cada máquina cliente para que estas aceitem a conexão do *webmaster*. Conexões feitas por meio de comandos-r são normalmente bloqueadas por se tratar de uma potencial falha de segurança na rede. Nesse caso, as configurações ficaram restritas às máquinas que foram utilizadas nos testes e, retornava-se a configuração ao normal ao fim do processo.

3. geração do conjunto de arquivos acessados no servidor.

   O WebStone pressupõe que os arquivos requisitados durante o teste existam no servidor, caso contrário, é indicado um erro e o teste é abortado. Por isso, o próprio *benchmark* disponibiliza um modo de criar um conjunto de arquivos. O comando "./webstone -genfiles" cria vários arquivos de diversos tamanhos, como file50k.html, por exemplo. Depois de criados, esses arquivos são transferidos para o servidor para serem acessados durante o teste.

Com isso, finaliza-se o processo de instalação da ferramenta que, como mencionado, apresentou dificuldades; algumas superadas facilmente, outras com dificuldade. Como ponto bastante positivo, pode-se destacar o fato de não ser necessária a instalação em cada máquina que se pretendia utilizar no teste. Nessas máquinas, foi necessário apenas a execução do passo 2, com o objetivo de definir a configuração dos comandos-r.

Execução

Mesmo após a conclusão da instalação, o WebStone ainda não se apresenta em condições de execução. Antes disso, é necessário fazer a configuração dos parâmetros que serão utilizados. Isto é feito através de dois arquivos: testbed e filelist.

A Tabela 4.1 ilustra o arquivo testbed utilizado em testes feitos durante o trabalho.

O primeiro conjunto de parâmetros significa que a execução será iniciada com 1 cliente e, por meio de um incremento de 50, serão executadas 11 iterações, quando o número de clientes superará 550 finalizando o teste. Cada iteração é executada por 1 minuto e o teste como um todo será repetido 3 vezes.

```
ITERATIONS="3"
MINCLIENTS="1"
MAXCLIENTS="550"
CLIENTINCR="50"
TIMEPERRUN="1"

SERVER="lasdpc17"
PORTNO=80

RCP=rcp
RSH=rsh

WEBDOCDIR=/usr/local/apache2/htdocs

CLIENTS="lasdpc15"

CLIENTACCOUNT=projeto
CLIENTPASSWORD=*******

FIXED_RANDOM_SEED=true
```

Tabela 4.1: Parâmetros utilizados na execução dos testes

O conjunto de parâmetros seguinte significa que será testado um servidor chamado "lasdpc17" na porta 80 (a porta pode ser alterada para possibilitar a utilização de proxies na mesma máquina). A seguir são especificados os comandos utilizados pelo webmaster para acessar as máquinas clientes, rcp e rsh, bem como o nome da conta e a senha utilizadas nesse acesso. A entrada "WEBDOCDIR" define onde estão, no servidor, os arquivos requisitados pelos clientes.

Finalmente, são especificados os nomes - *hostnames* - das máquinas clientes. No caso ilustrado foi utilizada apenas 1 máquina. A quantidade de processos nessa máquina fica definida de forma implícita. Como dito, o teste será executado de 1 até 550 clientes. Empregando-se "duas" máquinas, cada uma executará de 1/2 até 550/2 processos. Como não é possível dividir 1 processo entre duas máquinas, no primeiro passo é utilizado apenas 1 cliente. O Webmaster controla todo o teste utilizando a conta e senha definidas para executar comandos de forma remota nas máquinas clientes.

Uma vez terminada a configuração do testbed, passa-se a editar o filelist. A Tabela 4.2 ilustra um arquivo utilizado em testes feitos durante o trabalho.

/file500.html	350	#500
/file5k.html	500	#5125
/file50k.html	140	#51250
/file500k.html	9	#512500
/file5m.html	1	#5248000

Tabela 4.2: Arquivo filelist usado nos testes.

Na tabela há a definição de cinco arquivos diferentes. O número que segue o nome do arquivo representa seu peso na distribuição. Todos os pesos são somados e a freqüência de cada arquivo é o seu peso dividido pelo peso total. No exemplo mostrado, a soma dos pesos é igual a 1000. Dessa forma, o arquivo *file500k.html* será requisitado 350 de 1000 vezes. A última coluna, considerada um comentário pelo programa, mostra o tamanho de cada arquivo em bytes.

Terminada a configuração e definição dos parâmetros do teste, ele pode então, ser iniciado. O procedimento de teste é controlado em parte por *shell scripts* e em parte por programas. O *script* webstone controla o sistema como um todo, com opções para ajuda ao usuário, geração dos arquivos do servidor (genfiles), configuração do sistema e inicialização do teste de diversas maneiras. Essas opções são definidas acrescentando-se ao comando "./webstone", argumentos que podem definir a execução do teste como "silenciosa", para depuração ou para coleta automática de resultados após o término do teste. A sinopse deste *script* é:

```
webstone [-help | -setup | -kill | -genfiles | -silent | -results]
```

Outro *script*, denominado runbench, trata especificamente da execução do teste com o *benchmark*. Ele é chamado pelo *script* webstone para executar o trabalho de preparar, de acordo com os parâmetros, uma seqüência de execuções do WebStone. Entende-se por seqüência um conjunto de iterações a ser executadas. Em todo teste realizado com o WebStone é necessário especificar números mínimo e máximo de processos clientes, além de um incremento que será aplicado ao valor mínimo tantas vezes quantas forem necessárias para se atingir o valor máximo. Assim, uma iteração é executada para cada diferente número de processos. Acima desse nível há iterações do teste como um todo, isto é, há o procedimento de se atingir o valor máximo a partir do mínimo por meio de um número finito de passos, que pode ser repetido um número determinado de vezes.

Com os parâmetros definidos, o processo webmaster é invocado para executar o teste de fato. Esse programa apresenta a seguinte sinopse:

```
   webmaster -f config_file [-adrsvRSTWX] [-C webclient_path] [-D]
[-M netmask] [-P proxy_server] [-U clientfilelist] [-p port_num]
[-t run_time | -l loop_count] [-u masterfilelist] [-w webserver] [URLs]
```

Cada opção caracteriza a execução do programa de uma forma particular. Nesse caso, as características são:

-C webclient_path: Localização do programa webclient nos sistemas clientes.

-D: Arquivo, nos sistemas clientes, para receber os dados de depuração. Padrão stderr.

-M netmask: Valor a ser usado como máscara de rede.

-P proxy_server: Nome do servidor proxy para o qual enviar as requisições.

-R: Diz ao webclient para registrar todas as transações.

-S: Utiliza um valor fixo para a semente de números aleatórios que especificam qual URL será requisitada.

-T: Habilita as instruções de trace.

-U URL_file: Lê as URLs a serem requisitadas no arquivo "URL_file".

-W: Indica que o nome do servidor está no arquivo de configuração no sistema webmaster.

-X: Não utilizar rexec() para inicializar os clientes. Em seu lugar, utilizar sleep() para permitir ao usuário inicializá-los manualmente.

-a: Imprime informações de tempo para todos os clientes.

-d: Habilita instruções de depuração e trace.

-f config_file: Arquivo para especificação de clientes. Deve haver uma linha para cada sistema cliente, com a seguinte sintaxe:

```
<client host name> <login> <password> <number of webclients>
[<Web server>]
```

É perfeitamente possível utilizar o sistema webmaster também como cliente.

-l loop_count: Faz com que o filelist seja percorrido "loop_count" vezes, requisitando cada URL contida nesse arquivo.

-p portnum: Número da porta em que o servidor espera por requisições.

-s: Diz ao webclient para salvar todos os dados recebidos do servidor.

-t run_time: Duração do teste em minutos.

-u URL_file: Arquivo contendo as URLs para o teste, no sistema webmaster.

-v: Habilita modo "*verbose*".

-w webserver: Nome do host onde está o servidor a ser testado. Se o nome começar com um dígito, webmaster o utiliza, juntamente com a máscara especificada, opção "-M", para calcular o endereço IP do servidor, na mesma rede do cliente. Por exemplo, se o IP do sistema cliente é 168.10.2.14, a rede tem o valor padrão (255.255.255.0), e o servidor é 25, então o endereço a ser usado é 168.10.2.25.

Dessa forma, cada parâmetro acrescenta uma determinada característica ao teste. Independentemente dessas características, o programa sempre analisa toda a informação de configuração do teste, abre um *socket* para comunicação com os sistemas clientes que irão gerar a carga no servidor e então, através do comando rexec(), inicializa os processos webclient em cada um dos sistemas clientes.

O webclient, por sua vez, tem a seguinte sinopse:

```
webclient -w webserver -n numclients [-t run_time | -l loop_count]
[-dsvRT] [-c webmasterhost:port] [-u URL_file | URL [...]] [-D]
[-P proxyserver] [-S seed] [-p port_num]
```

As opções de definição da execução do programa são as seguintes:

-D: Envia os dados de depuração para stderr.

-P proxy_server: Envia requisições à "proxy_server".

-R: Grava todas as transações em arquivo.

-S seed: Usa a semente fornecida para inicializar o gerador de números aleatórios.

-T: Habilita as instruções de trace.

-c webmasterhost:port: Conecta ao sistema webmaster através da porta fornecida para trocar mensagens de controle e enviar resultados.

-d: Habilita as instruções de depuração e trace.

-l loop_count: Faz com que o filelist seja percorrido "loop_count" vezes, requisitando cada URL contida nesse arquivo.

-p portnum: Número da porta em que o servidor espera por requisições.

-s: Diz ao webclient para salvar todos os dados recebidos do servidor.

-t runt_time: Duração do teste em minutos.

-u URL_file: Arquivo contendo as URLs para o teste, no sistema cliente.

-v: Habilita modo "*verbose*".

-w webserver: Nome do host onde se encontra o servidor a ser testado.

Depois de verificar cada parâmetro, o processo webclient cria um número de instâncias de si mesmo, especificado em testbed. Embora a parte cliente da execução do teste esteja sendo referenciada aqui como processo, essas instâncias podem ser tanto processos como *threads*. A tarefa básica desses processos ou *threads*, é enviar uma requisição HTTP e

medir o tempo tomado para que ela seja atendida. Logo após o atendimento e registro de tempo, outra requisição é feita.

Quando a execução termina, o cliente relata seus resultados - o que pode acontecer de duas formas diferentes: se o programa está sendo executado de forma independente, "*standalone*", os resultados são escritos na saída padrão(stdout); se a execução está acontecendo sob o controle do webmaster, então espera-se até que este último "entre em contato" para receber os resultados.

Embora seja possível a execução dos programas webmaster e webclient sem a utilização de controladores como os *scripts* runbench e webstone; não é aconselhada, pois a execução do benchmark através da linha de comando torna-se muito complexa.

Ao final do teste, o procedimento padrão - isto é, se não foi especificado algum parâmetro que interfira nesse ponto - estabelece que os resultados obtidos em cada iteração sejam armazenados no diretório runs. O WebStone oferece ainda um mecanismo, o *script* wscollect, que formata esses resultados em uma tabela unificada, tornando mais simples a sua visualização.

4.2.2 Arquitetura

O WebStone é um *benchmark* multi-processado e distribuído. A seção anterior já exibiu alguns pontos de sua arquitetura que volta a ser abordada com mais detalhes nesta seção.

O procedimento de execução do WebStone ocorre da seguinte forma: runbench é executado em uma máquina (*webmaster*) para controlar o teste no nível mais alto. Este *script* criará então, na máquina atual, uma instância do processo *webmaster* que lê os arquivos de configuração e, baseado neles, cria um comando a ser executado pelos clientes. Esses clientes, *webclients*, são então criados de forma remota em diversas máquinas e iniciam sua execução. Ao término da sua execução, cada cliente coleta os dados medidos e os passa ao processo central. Este último aguarda até que todos os clientes emitam seus resultados que são compilados em um relatório final. Nesse ponto, quando o controle volta a runbench, termina-se uma iteração. Se ficou especificado, através dos parâmetros, a execução de mais iterações, todo o processo é repetido até se alcançar o final do teste como um todo. A Figura 4.1 ilustra a relação entre as diversas partes de um teste com o WebStone.

Em relação ao código fonte, a ferramenta apresenta um funcionamento confuso, apesar de bem comentado. Isso porque em alguns pontos há implementações com o intuito de permitir o uso de requisições a páginas com vários arquivos - utilizando uma conexão para fazer várias requisições - mas, de forma geral, a ferramenta não permite esse tipo de funcionamento. São utilizadas ainda, diversas estruturas para a coleta de dados e estatísticas, o que acaba gerando confusão. Outro agravante desse ponto é o controle feito

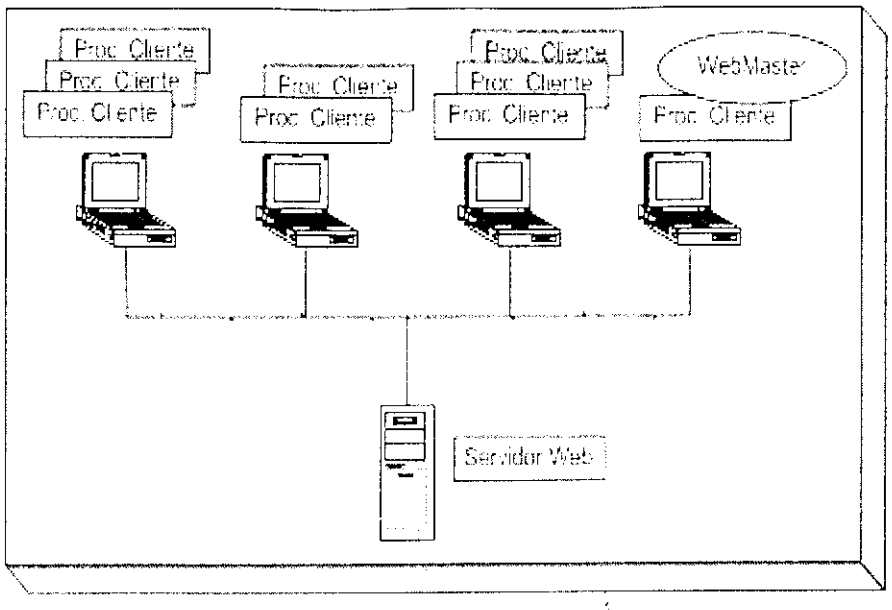


Figura 4.1: Arquitetura do benchmark WebStone.

em parte por programas em código C, em parte por *shell scripts*, o que acaba gerando mais confusões.

4.2.3 Carga de trabalho

O WebStone permite a modelagem de cargas reais por meio de cargas sintéticas criadas de acordo com parâmetros definidos pelo usuário. Os parâmetros de especificação são da seguinte forma:

- Número de clientes que requisitam páginas. Os clientes requisitam páginas tão rápido quanto o servidor pode responder (tempo de pensar igual a zero). O tempo de pensar (*think time*), não pode ser representado em um carga gerada pelo WebStone.
- Tipo de página, definido pelo tamanho e freqüência de acesso. Cada página na combinação possui um peso que indica a sua probabilidade de ser acessada.
- O número de páginas disponíveis no servidor.
- Número de máquinas clientes, onde os processos clientes são executados.

A definição desses parâmetros é feita de acordo com o que foi mostrado na seção 4.2.1, isto é, através de arquivos de configuração. Além dos parâmetros mostrados neste capítulo, a forma de definição da duração do teste também influencia na geração da carga. Se foi utilizada a opção "-l loop_count", cada webclient irá requisitar as URLs contidas na lista de forma seqüencial, passando por toda a lista "loop_count" vezes, quando o teste se encerrará. Mas se a definição for feita através da opção "-t run_time", os processos clientes irão requisitar as URLs de forma aleatória até que o tempo especificado se esgote.

4.2.4 Resultados apresentados pelo WebStone

Os principais resultados produzidos pelo WebStone são a latência e o *throughput*. Este último, medido em bytes por segundo, representa o número total de bytes recebidos do servidor, dividido pelo tempo de teste. A primeira mede, do ponto de vista do usuário, o tempo necessário para a conclusão de uma requisição. A latência consiste de três componentes: conexão, tempo de resposta, e latência da rede devido a roteadores, modems e outros. O primeiro componente reflete o tempo necessário para estabelecer uma conexão, enquanto o segundo reflete o tempo tomado para completar a transferência dos dados, uma vez que a conexão tenha sido estabelecida.

A Tabela 4.3 ilustra os resultados obtidos pelo WebStone em um teste típico, isto é, utilizando valores padrão para os parâmetros. Várias iterações são necessárias porque o WebStone é um processo estocástico e, portanto, haverá variações de execução para execução, especialmente se o conjunto de arquivos do teste contiver arquivos grandes, ou se o servidor chegar a ficar sobrecarregado.

Número total de Clientes	Taxa de Conexões (conexões/s)	LLF	Tempo médio de resposta (s)	Nível de Erros (%)	*Throughput* médio (Mbit/s)
1	936,7	0,99	0,001	0	40,39
51	2060,5	50,84	0,025	0	88,85
101	2060,15	98,72	0,048	0	88,83
151	2061,65	146,78	0,071	0	88,9
201	2065,37	196,96	0,095	0	89,06
251	2071,25	239,17	0,115	0	89,31
301	2063,48	290,13	0,141	0,3	88,98
351	2069,13	306,88	0,148	0	89,22
401	2063,33	343,52	0,166	0,57	88,97
451	2062,68	404,31	0,196	0,7	88,94
501	2059,12	431,86	0,21	0	88,79

Tabela 4.3: Exemplo de resultados obtidos pelo WebStone

A métrica LLF (*Little's Load Factor*) é derivada da lei de *Little* (Menascé *et al.*, 1994). Ela indica o grau de concorrência na execução das requisições, isto é, o número médio de conexões abertas no servidor Web em um momento qualquer do teste. É também uma indicação de quanto tempo é gasto no processamento de requisições - não em erros ou sobrecarga. Em condições ideais, LLF deveria ser igual ao número de clientes, pois um número menor indica que o servidor está sobrecarregado e, por isso, algumas requisições não estão sendo atendidas antes de seu tempo acabar (*time out*). Quando conexões começam a ser descartadas, o nível de erros cresce, reafirmando que o servidor não está conseguindo atender a todas as requisições recebidas. O número de erros pode ser um excelente indicador de como o servidor se comporta em caso de sobrecarga.

4.3 Httperf

O httperf (Mosberger & Jin, 1998; Provos *et al.*, 2000) é uma ferramenta construída para medir o desempenho de servidores Web. Ela pode utilizar o protocolo HTTP em suas versões 1.0 e 1.1, e oferecer ainda uma variedade de geradores de carga. Durante a execução de um teste, são coletadas diversas métricas de desempenho que são compiladas em um quadro de estatística no final da execução. A operação mais básica do httperf é gerar um número fixo de requisições GET e medir quantas respostas foram recebidas do servidor e a que taxa essas respostas aconteceram.

É importante ressaltar que para se obter resultados corretos, torna-se necessário executar, no máximo, um único processo do httperf por máquina cliente. Além disso, deve haver o menor número possível de processos executando no cliente e no servidor.

4.3.1 Configuração

Instalação

O httperf, assim como o WebStone, utiliza o mecanismo GNU padrão de instalação. Dessa forma, para instalá-lo é necessária a execução dos seguintes passos: configure; make; e make install. Caso haja interesse em alterar valores padrões como os diretórios de instalação dos executáveis ou da documentação, podem-se passar opções ao *script* configure. O httperf pode ainda ser compilado com suporte a SSL (*secure sockets layer*), mas para isso é necessário que o OpenSSL (www.openssl.org) esteja instalado no sistema.

A instalação do httperf feita no presente trabalho ocorreu de forma tranqüila, sem grandes dificuldades. Primeiro foi feito o *download* do código fonte da ferramenta. O arquivo foi então descompactado no local desejado - nesse caso /usr/local/httperf - e finalmente os três passos citados acima foram executados. Todo esse procedimento foi executado em cada máquina que se pretendia utilizar como cliente.

Execução

Uma execução simples do httperf pode ser alcançada da seguinte forma:

```
httperf --server hostname --port 80 --uri /test.html --rate 150
 --num-conn 27000 --num-call 1 --timeout 5
```

O comando faz com que o httperf utilize a máquina "*hostname*" como servidor Web, e é executado na porta 80. A página requisitada é "/test.html" e, nesse caso, a mesma página é requisitada repetidamente. A taxa de requisição é de 150 por segundo. O teste envolve a criação de 27.000 conexões TCP com cada uma gerando uma chamada HTTP

(uma chamada consiste em enviar uma requisição e receber a correspondente resposta). A opção "*timeout*" informa o tempo em segundos que o cliente permanece esperando uma resposta do servidor. Se o "*timeout*" expirar, o *benchmark* considera que a chamada falhou. Destaca-se que, com um total de 27.000 conexões a uma taxa de 150 conexões/s, o tempo total de teste será de 180 segundos, independentemente da carga que o servidor pode suportar.

No caso da utilização de diversas máquinas clientes, um comando desse tipo deve ser executado em cada máquina a ser utilizada. O httperf não possui um gerenciador central para o controle dos clientes. Terminada a execução do teste, diversas estatísticas sobre o desempenho do servidor são impressas por cada máquina cliente. Para se obter um relatório geral sobre a execução é necessário juntar, manualmente, todos os relatórios gerados por cada uma dessas máquinas.

A forma de execução descrita utiliza apenas algumas das opções presentes no httperf. Para a obtenção de um teste mais completo podem-se utilizar mais parâmetros de configuração, tais como:

--burst-length =*N*: Especifica o tamanho das rajadas. Cada rajada consiste de N chamadas ao servidor.

--hog: Esta opção requisita o uso de quantas portas TCP forem necessárias. Sem esta opção, o httperf fica limitado ao uso das portas "efêmeras" - ephemeral ports - (na faixa de 1024 a 5000). Esse limite pode se tornar rapidamente um gargalo, por isso é uma boa idéia a especificação dessa opção. Além disso, essa opção deve ser especificada quando da utilização de servidores NT, pois ela impede a incompatibilidade entre máquinas UNIX e NT.

--http-version =*S*: Especifica o valor do campo versão que deve ser incluído nas requisições enviadas ao servidor. O valor padrão utilizado é "1.1". Essa opção pode ser definida como "1.0" para forçar a geração de requisições HTTP/1.0. Definir essa opção para qualquer valor diferente de "1.0" ou "1.1" pode resultar em um comportamento imprevisível.

--period =*[D]T1[T2]*: Especifica o intervalo de tempo entre a criação de conexões ou sessões. O parâmetro D especifica a distribuição desse tempo. Se omitido ou definido como "**d**", um período determinístico é usado como especificado pelo parâmetro $T1$ em unidades de segundos. Se D for definido como "**e**", uma distribuição exponencial, é usada como um tempo médio de $T1$. Finalmente, se$D$ for definido como "**u**", uma distribuição uniforme sobre o intervalo $[T1, T2)$ é usada. Em todos os casos, um período igual a 0 resulta em conexões/sessões geradas seqüencialmente (uma nova conexão/sessão é criada imediatamente após o término da anterior).

--ssl: Indica que toda a comunicação feita entre o httperf e o servidor deve utilizar

o protocolo SSL. Essa opção é disponibilizada apenas se a ferramenta for compilada com suporte a SSL.

--wsess =*N1,N2,X*: Força a utilização de sessões no lugar de requisições individuais. Uma sessão consiste em uma seqüencia de rajadas espaçadas pelo tempo de pensar do usuário. Cada rajada é constituída por um número fixo L de chamadas ao servidor (L é especificado pela opção **--burst-length**). Essas chamadas são feitas da seguinte forma: inicialmente, apenas uma chamada é feita; assim que a resposta para essa primeira chamada for recebida, todas as chamadas restantes são feitas de forma concorrente. Essas últimas chamadas podem ser feitas em *pipeline* em uma só conexão persistente, ou de forma separada em conexões individuais. A decisão sobre qual método será utilizado depende da resposta à primeira chamada enviada pelo servidor. Se o cabeçalho da resposta contiver a linha *"Connection: close"*, serão utilizadas conexões separadas. A opção especifica os seguintes parâmetros: *N1* é o número total de sessões a ser geradas, *N2* é o número de chamadas por sessão e *X* é o tempo de pensar, em segundos, que separa rajadas consecutivas. Um teste envolvendo sessões termina tão logo o número *N1* de sessões tiver sido completado ou ter ocorrido alguma falha. Uma sessão é considerada falha se alguma operação contida nela tomar mais tempo do que o especificado pelas opções **--timeout** e **--think-timeout**.

Todas as opções citadas representam apenas uma parte do que é oferecido pela ferramenta. Elas foram mencionadas por terem sido consideradas mais importantes. O httperf possui ainda diversas opções para a obtenção de testes mais específicos e dirigidos aos objetivos do usuário.

4.3.2 Arquitetura

Como mostrado na seção anterior, o httperf trabalha com apenas um processo por máquina. Esse processo lê os parâmetros da linha de comando, estabelece as características do teste e inicia sua execução com a criação de uma *thread*. Não há nenhuma forma de controle centralizado. Na utilização de diversas máquinas clientes, cada uma trabalha de forma independente. Cabe ao usuário, ao final do teste, compilar os resultados de cada máquina em um documento único.

O *software* foi projetado tendo em vista os objetivos: ter bom desempenho e facilidade para futuras extensões. O bom desempenho é alcançado pela implementação da ferramenta em C, com bastante atenção aos trechos do código referentes ao desempenho. Outro fator é o gerenciamento de *timeout*: para não depender de mecanismos do SO, o httperf implementou um mecanismo próprio: um instrumento leve e especializado que descarta as pesadas chamadas ao sistema (*system calls*).

Para alcançar a facilidade de mudanças, o httperf foi dividido em três partes diferentes: módulo HTTP (*CORE*), geração de carga e coleta de estatísticas. O módulo HTTP manipula toda a comunicação com o servidor, assim, cuida do gerenciamento das conexões e da geração de requisições HTTP. A geração de carga é responsável por iniciar chamadas HTTP para gerar uma carga particular no servidor. A terceira parte é responsável por medir diversos dados e produzir estatísticas relevantes a respeito do desempenho.

O módulo HTTP é o que controla a execução de ferramenta como um todo. Ele faz a inicialização e o disparo de cada um dos outros módulos através das funções init() e start() (ver Figura 4.2). Com isso, os três módulos passam a estar aptos para execução. A interação entre eles ocorre através de um mecanismo dirigido a eventos. A idéia é que sempre que algo acontece no httperf, um evento é sinalizado. As partes interessadas em eventos registram um manipulador para tal evento e esses manipuladores são invocados sempre que o evento certo é sinalizado. Por exemplo, o coletor de estatísticas associa a função conn_created() ao evento que registra o estabelecimento de uma conexão. Dessa forma, sempre que uma conexão é estabelecida, o módulo de coleta de estatística recebe um sinal e a função citada é colocada em execução. Com esse mecanismo, a medida do tempo necessário para estabelecer uma conexão TCP pode ser obtida por meio do registro de manipuladores para os eventos que sinalizam o envio de um pedido e o estabelecimento de uma conexão. A Figura 4.2 ilustra os três módulos do httperf.

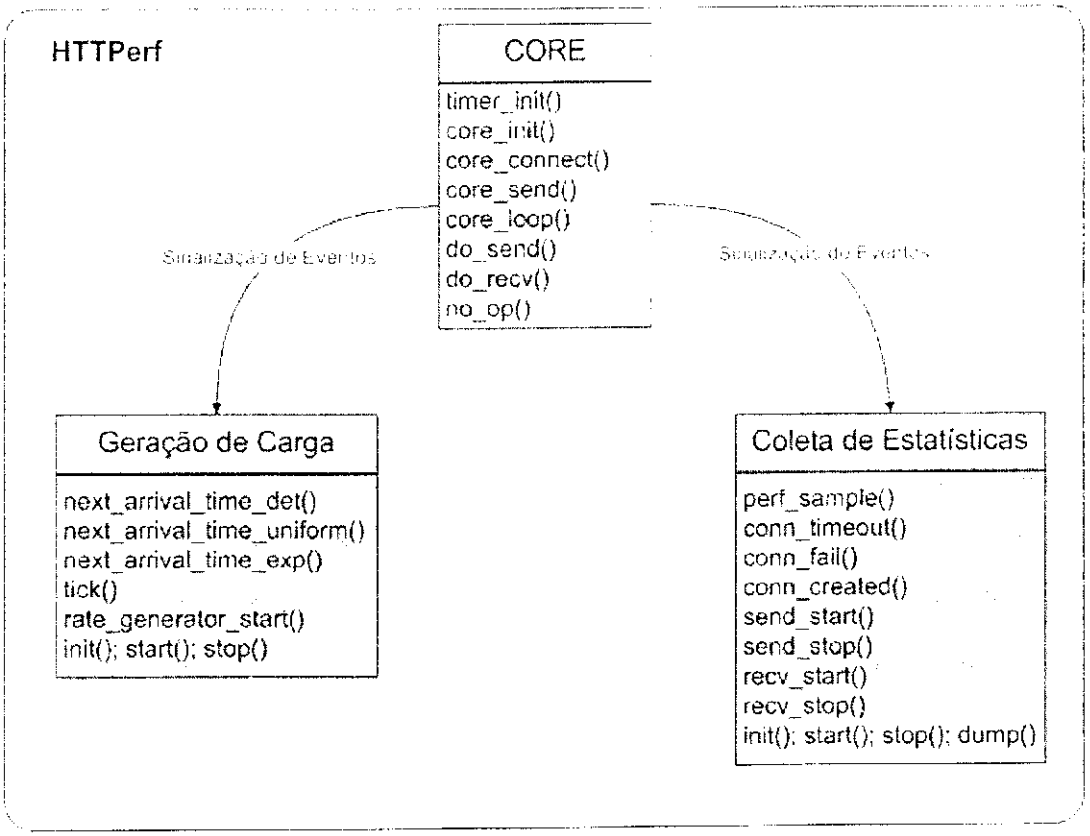


Figura 4.2: Relação entre os módulos do httperf.

O código fonte propriamente dito encontra-se bem organizado, com os arquivos pertencentes a cada módulo do sistema separados em diretórios distintos. No diretório principal residem os arquivos referentes ao módulo HTTP, ou *CORE*, responsável pelo controle geral do sistema. Há ainda dois subdiretórios, gen e stat, que correspondem respectivamente aos módulos de geração de carga e coleta de estatísticas. O modo de funcionamento dirigido a eventos também facilita o entendimento de todo o mecanismo da ferramenta. De forma geral, o código do httperf contém comentários em uma quantidade reduzida, mas ainda assim seu entendimento se dá de maneira direta.

4.3.3 Carga de trabalho

O módulo HTTP dá suporte ao HTTP 1.0 e 1.1. Entre as características mais interessantes desse módulo estão o suporte a conexões persistentes e a requisições em cascata (*pipelining*).

A versão oficial do httperf (Mosberger & Jin, 1998) dá suporte a dois tipos de geradores de carga: geradores de requisição e geradores de URL:

- **Geração de requisições**: Há dois tipos de geradores de carga: o primeiro gera requisições deterministicamente a uma taxa fixa. Cada requisição é usada para executar uma linha de comando especificando o número de chamadas em cascata a ser executadas. O número padrão de chamadas em cascata é 1, o que reproduz o comportamento do HTTP 1.0 onde cada conexão é usada para apenas uma chamada.

  O segundo tipo de gerador cria *sessões*, também deterministicamente, a uma taxa fixa. Cada sessão consiste em um número especificado de chamadas em rajadas, *call-bursts*, e há um intervalo de tempo, *tempo de pensar*, entre elas. Essas rajadas consistem em um número fixo de chamadas independentes. As chamadas em rajadas imitam o comportamento típico de um navegador. Quando o usuário seleciona uma página, o navegador primeiro requisita o arquivo HTML e depois os objetos pertencentes à página.

- **Geração de URLs**: Geradores desse tipo criam uma seqüência desejada de URLs que devem ser acessadas no servidor. Geradores mais primitivos apenas especificavam a mesma URL repetidamente.

  Outros geradores utilizam um conjunto fixo de URLs. Com esses geradores, as páginas são assumidas como organizadas em uma árvore tenária de diretórios (cada diretório possui até 10 arquivos ou subdiretórios) no servidor. Esses geradores são úteis, por exemplo, para induzir uma taxa de erros no cache (*cache miss*) do servidor em teste.

A definição desses parâmetros é feita de acordo com a forma mostrada na seção 4.3.1. Embora tenha sido citada apenas a geração de carga de forma determinística, o httperf

oferece ainda os métodos Uniforme e Exponencial especificados por meio do parâmetro "period =*[D]T1[T2]*".

4.3.4 Resultados apresentados pelo httperf

A Figura 4.3 ilustra um resultado final típico de um teste realizado com o httperf. Primeiramente aparece a linha de comando utilizada para dar início à execução do programa e, então, os resultados propriamente ditos. Na linha de comando pode-se perceber a entrada: "–wsesslog=10000,5.000,sessoes.05.txt". Exceto pela definição do nome de um arquivo, este parâmetro funciona como a opção "–wsess" mencionada acima. O arquivo é utilizado para especificar como devem ser feitas as requisições dentro de cada sessão. São definidos o número e a forma, seqüencialmente ou em *pipeline*, que as requisições devem ser feitas.

Os parâmetros utilizados para construir o arquivo "sessoes.05.txt" (número de sessões, quantidade de requisições por sessão, tamanho dos arquivos requisitados), foram tirados do *benchmark* Surge (Crovella *et al.*, 1999).

```
httperf --hog --timeout=5 --client=0/1 --server=143.107.232.253 --port=80
--uri=/ --period=e0.05 --send-buffer=4096 --recv-buffer=16384
--wsesslog=10000,5.000,sessoes.05.txt
Maximum connect burst length: 2

Total: connections 10000 requests 45270 replies 45270 test-duration 510.487 s

Connection rate: 19.6 conn/s (51.0 ms/conn, <=151 concurrent connections)
Connection time [ms]: min 0.4 avg 5753.9 max 20197.9 median 5008.5 stddev 4228.7
Connection time [ms]: connect 0.2
Connection length [replies/conn]: 4.527

Request rate: 88.7 req/s (11.3 ms/req)
Request size [B]: 77.0

Reply rate [replies/s]: min 1.8 avg 88.8 max 112.4 stddev 15.9 (102 samples)
Reply time [ms]: response 14.1 transfer 1.6
Reply size [B]: header 250.0 content 16145.0 footer 0.0 (total 16395.0)
Reply status: 1xx=0 2xx=45270 3xx=0 4xx=0 5xx=0

CPU time [s]: user 176.39 system 334.09 (user 34.6% system 65.4% total 100.0%)
Net I/O: 1426.6 KB/s (11.7*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 1.20 avg 19.59 max 27.00 stddev 4.01 (10000/10000)
Session: avg 1.00 connections/session
Session lifetime [s]: 5.8
Session failtime [s]: 0.0
Session length histogram: 0 550 860 1730 1930 1700 1750 940 400 110 10 20
```

Figura 4.3: Estatísticas de desempenho obtidas pelo httperf.

Pode-se perceber seis grupos de estatísticas: resultados gerais ("*Total*"), resultados pertinentes a conexões ("*Connection*"), referentes a requisições HTTP ("*Request*"), respostas recebidas do servidor("*Reply*"), utilização de CPU ("*CPU*") e rede ("*Net I/O*") e, finalmente, um sumário de erros ("*Errors*").

- ***Total Section***: Compila quantas conexões TCP foram iniciadas pelo httperf, quantas requisições ele enviou, quantas respostas foram recebidas e qual a duração total do teste. No exemplo acima, foram feitas 10000 conexões, que geraram 45270 requisições e obtiveram esse mesmo número de respostas. O teste teve uma duração total de 510,487 segundos.

- ***Connection Section***: Exibe informações relacionadas às conexões TCP geradas pela ferramenta. Especificamente, a linha "*Connection rate*" mostra que novas conexões foram iniciadas a uma taxa de 19,6 conexões por segundo. Essa taxa corresponde a um período de 51 milisegundos por conexão. O último valor na linha mostra que, no máximo, 151 conexões estavam abertas em um determinado instante.

  A primeira linha entitulada "*Connection time*" mostra estatísticas para o tempo de vida das conexões que ocorreram corretamente. O tempo de vida de uma conexão é o tempo entre a inicialização e o encerramento da conexão TCP. Uma conexão é considerada correta se teve pelo menos uma chamada completada corretamente. No exemplo ilustrado, a linha indica que o tempo de vida mínimo ("*min*") foi de 0,4 milisegundos, além de fornecer valores também para a média("*avg*"), o máximo, a mediana e o desvio padrão.

  A linha seguinte exibe o tempo médio tomado para estabelecer uma conexão. Apenas conexões que ocorreram com sucesso são levadas em conta. No exemplo, 0,2 milisegundos foram necessários, em média, para estabelecer uma conexão.

  Finalmente, "*Connection length*" dá o número médio de respostas recebidas em cada conexão que obteve pelo menos uma resposta - no caso, o valor alcançado foi 4,527 respostas/conexão. Esse número pode ser maior que 1,0 devido a conexões persistentes.

- ***Request Section***: "*Request rate*" mostra a taxa em que requisições HTTP foram emitidas e o período a que essa taxa corresponde. No exemplo acima, a taxa de requisição foi de 88,7 por segundo, o que corresponde a 11,3 milisegundos por requisição. Para o caso da não utilização de conexões persistentes, os resultados nessa seção podem ser bastante parecidos, ou até mesmo idênticos, aos encontrados na seção de conexões.

  "*Request size*" apresenta o tamanho médio, em bytes, das requisições HTTP. No exemplo, esse tamanho foi de 77 bytes.

- ***Reply Section***: "*Reply rate*" exibe diversas estatísticas sobre a taxa de resposta. No exemplo, a taxa mínima de resposta("*min*") foi de 1,8 respostas por segundo, a média("*avg*") de 88,8 respostas por segundo, e o máximo de 112.4. O número entre parênteses mostra que 102 amostras foram colhidas. O httperf coleta uma amostra de taxa, aproximadamente, a cada 5 segundos.

A linha entitulada "*Reply time*" apresenta informações sobre quanto tempo o servidor levou para responder e quanto tempo foi necessário para receber a resposta. No exemplo, foram gastos em média 14,1 milisegundos entre o envio do primeiro byte da requisição e o recebimento do primeiro byte da resposta. O tempo para transferir ou ler a resposta foi de 1,6 ms. Pode ocorrer de o tempo para transferência ser registrado como 0,0. Isso acontece, tipicamente, quando a resposta cabe em apenas um segmento TCP.

A linha seguinte, "*Reply size*", contém estatísticas sobre o tamanho médio das respostas. Especificamente, a linha lista o tamanho médio do cabeçalho, do conteúdo e de rodapé("*footers*"). Por conveniência, a média do número total de bytes nas respostas também é dado, número entre parênteses. No exemplo, o tamanho total das respostas foi de 16395 bytes, em média.

A última linha é um histograma dos principais códigos de status recebidos do servidor. Esses códigos são uma referência às classes de status utilizadas pelo protocolo HTTP. No exemplo, todas as respostas foram recebidas com o status "2xx", indicando que a operação ocorreu com sucesso.

- ***Miscellaneous Section***: Essa seção começa com um sumário da utilização da CPU na máquina cliente. No exemplo, a linha entitulada "*CPU Time*" mostra que 176,39 segundos foram gastos com execução em modo usuário ("*user*"), 334,09 segundos pelo sistema ("*system*") e que isso corresponde a uma execução de 34,6% em modo usuário e 65,4% pelo sistema. A utilização total foi de 100%, o que era esperado dado que o httperf exige muito processamento. Uma utilização total menor do que 100% é sinal de que há processos competindo pelo processador e interferindo no teste.

  "*Net I/O*" fornece o throughput médio da rede em kilobytes (1024 bytes) e em megabits (10^6 bits) por segundo. No exemplo, o uso médio da rede ficou em 1426,6 kilobytes por segundo, e o número entre parênteses mostra que isso corresponde a 11,7 megabits por segundo. Essa banda de rede é medida com base no número de bytes enviados e recebidos nas conexões TCP. Em outras palavras, não são levados em conta os cabeçalhos ou retransmissões TCP.

- ***Errors Section***: A próxima seção contém estatísticas sobre erros encontrados durante o teste. No exemplo, as duas linhas nomeadas "*Errors*" mostram que houve um total de 0 erros. Uma descrição de cada erro é apresentada a seguir:

  - ***client-timo***: Número de vezes que uma sessão, conexão ou chamada falhou devido a *timeout*.
  - ***socket-timo***: Número de vezes que uma conexão TCP falhou devido a um *timeout* em nível de socket (ETIMEDOUT).

- ***connrefused***: Vezes em que uma conexão TCP falhou devido à recusa do servidor (ECONNREFUSED).
- ***connreset***: Número de erros devido a um "RESET" vindo do servidor. Normalmente, um "RESET" é recebido quando um cliente tenta enviar dados ao servidor que já encerrou a conexão.
- ***fd-unavail***: Vezes em que o httperf esgotou os descritores de arquivo (*file descriptors*). Nesse caso o teste perde o significado porque o cliente foi sobrecarregado.
- ***addrunavail***: Número de vezes em que o cliente esgotou os números de portas TCP (EADDRNOTAVAIL). Este erro nunca deve ocorrer, caso contrário, os resultados devem ser descartados.
- ***ftab-full***: Vezes em que a tabela de descritores de arquivos do sistema esteve cheia. Novamente, este erro não deve ocorrer.
- ***other***: Registra a ocorrência de outros erros. Sempre que este contador for diferente de zero, deve-se rastrear a causa real do erro.

Além desse conjunto de estatísticas, quando são utilizadas sessões durante o teste, o httperf imprime dados a respeito dessas sessões. Isso é o que pode ser visto como o último conjunto de estatísticas apresentado na Figura 4.3.

A linha "*Session rate*" mostra valores de mínimo, máximo, média e desvio padrão atingidos durante a execução do teste. Os números entre parênteses indicam quantas sessões ocorreram com sucesso e quantas foram inicializadas. Na seqüência, "*Session*" indica o tamanho médio das sessões, medido em conexões. No caso ilustrado, foi utilizada apenas uma conexão por sessão.

"*Session lifetime*" mostra que foram gastos 5,8 segundos, em média, para completar uma sessão. O próximo valor, "*Session failtime*", fornece o tempo médio até que ocorra uma falha em alguma sessão criada no teste. Como o total de erros foi zero, já mostrado anteriormente, esse valor também é 0,0.

A última linha impressa como resultado do teste, "*Session length histogram*", é um histograma do número de respostas recebidas por cada sessão. No exemplo acima, nenhuma sessão terminou sem receber alguma resposta; 550 sessões terminaram após receber uma resposta; 860 sessões receberam 2 respostas e assim sucessivamente.

4.4 Considerações finais

A utilização de ferramentas para a avaliação de servidores Web deve ser feita com bastante cuidado, pois os resultados obtidos por meio desse tipo de *benchmak* podem tanto informar, como confundir os usuários a respeito da real capacidade do sistema. Isso

depende de como os resultados são interpretados. Antes de utilizar qualquer resultado, é necessário entender o sistema em estudo, os testes e, finalmente, os próprios resultados.

Apesar de serem citados vários *benchmarks* na introdução deste trabalho, no presente capítulo foram discutidos em detalhes apenas dois. Isto ocorreu porque uma das ferramentas, no caso o Surge, apresentou sérios problemas. Mesmo depois de se investir muito tempo em correções no código fonte e acertos na configuração de máquinas, o *benchmark* não funcionou como esperado. Assim, o que se utilizou da ferramenta foram apenas os parâmetros de caracterização de carga que se mostraram muito interessantes. Já o SPECWeb foi descartado por se tratar de um *software* comercial, não estando assim, disponível publicamente.

Como pôde ser visto, os *benchmarks* descritos apresentam características bastante distintas, cada um com suas vantagens e desvantagens. O WebStone apresentou problemas na instalação e na execução. Foi necessário grande esforço para que essa ferramenta funcionasse de forma adequada. Além dos erros, o código fonte se apresentou bastante confuso, e foi difícil a identificação do modo de implementação de cada parte do algoritmo. Quanto à carga, o WebStone não permite a utilização de sessões, ou requisições em cascata, nem a representação do tempo de pensar dos usuários.

Como ponto positivo do WebStone, pode-se destacar o fato de este ser distribuído, ou seja, para a utilização de várias máquinas não são necessárias diversas instalações. No entanto, há a necessidade de configuração de cada uma dessas máquinas para que elas possam receber conexões através da rede. Outros pontos positivos são a existência de um mecanismo automatizado para criar os arquivos que serão requisitados ao servidor e a presença de uma configuração (*filelist* e *testbed*) pré-determinada que pode ser utilizada em um teste tão logo a ferramenta esteja instalada.

Já o httperf se mostrou mais fácil de se instalar e executar. Apresentou também um código mais legível e fácil de se entender, conseqüentemente, mais propício a receber as alterações propostas. Em relação à geração de carga, há a possibilidade de modelar o tempo de pensar dos usuários e utilizar o protocolo HTTP em sua versão 1.1, possibilitando o uso de chamadas em cascata(pipeline). Nesse quesito, o httperf apresenta uma desvantagem por não oferecer um conjunto de arquivos pré-definidos para a utilização em testes.

Outro fato bastante importante é a quantidade e a qualidade da documentação disponível. Nesse ponto, o httperf se saiu muito melhor do que o WebStone, pois este disponibiliza um material que gera confusão, por misturar diferentes versões da ferramenta. Já o httperf mostrou, dentre outros documentos, um manual (*manpage*) bastante completo, contendo detalhes a respeito da implementação e do mecanismo de funcionamento da ferramenta.

Por fim, os dois *benchmarks* apresentam seus resultados de forma muito distinta, o

que pode tornar difícil a comparação direta entre eles. O httperf não utiliza o conceito de clientes, assim, não faz sentido comparar por exemplo, o throughput de 101 clientes no WebStone - 88,83 Mbits/s - com o throughput obtido pelo httperf - 11,7 Mbits/s.

A Tabela 4.4 apresenta, de forma sucinta, um comparativo entre o httperf e o WebStone.

	WebStone	**httperf**
Documentação	Apresenta-se em boa quantidade mas bastante confusa, com partes que fazem referência à versão 2.0 e outras à versão 2.5 da ferramenta	Apresenta-se em menor quantidade porém, de forma mais técnica e sem confusões.
Instalação	Bastante complicada, sendo necessárias correções no código fonte. Entretanto, a instalação é feita em apenas uma máquina.	Ocorreu sem problemas. Só foi necessária a instalação em todas as máquinas que se pretendia utilizar nos experimentos.
Execução	Ocorre de forma tranqüila. Há valores já definidos para serem aplicados em testes padrão. É necessário apenas executar o *script* que controla todo o teste. A utilização de diversas máquinas clientes é trivial.	Não há nenhuma definição padrão. É necessário, antes de tudo, buscar parâmetros para definir o teste a ser realizado. Se mais de uma máquina cliente for utilizada, o usuário deve inicializar o teste em cada uma dessas máquinas.
Carga de Trabalho	Menos flexível pois utiliza apenas o HTTP 1.0; não utiliza o conceito de sessões; não permite requisições em cascata. Como vantagem, possibilita a representação de usuários por meio de processos, ou seja, é possível definir testes com 100 500 ou 1000 clientes, por exemplo.	Mais Flexível. Pode-se utilizar o HTTP nas versões 1.0 e 1.1 permitindo assim a utilização de requisições em *pipeline* juntamente com sessões. É possível ainda representar o *think time*, e apresenta uma forma "primitiva" para se utilizar logs. Entretanto, não é possível definir um número específico de usuários.
Resultados	São apresentados de forma clara e simplificada, devido à existência de um tratamento feito aos dados coletados. Os resultados de diversos clientes são compilados em uma única tabela automaticamente. Pode-se verificar o resultado por número de processos clientes.	São apresentados por cada máquina cliente. Para se obter o resultado geral do teste é necessário compilá-los manualmente. Não é possível verificar o resultado por cliente mas apenas por teste.

Tabela 4.4: Quadro comparativo entre o WebStone e o httperf

Capítulo 5

Modificações no Httperf

5.1 Introdução

O capítulo 4 expôs os benchmarks WebStone e httperf, destacando características de cada um deles e terminando com uma breve comparação entre os dois *softwares*. Como pôde ser visto, o httperf se mostrou mais propício a receber as implementações planejadas neste trabalho.

A melhoria de cargas de trabalho geradas por um *benchmark* pode ser obtida de diversas formas, dentre elas, pode-se destacar a pesquisa por novos parâmetros em cargas reais; execução de testes distribuídos - com várias máquinas clientes para alcançar maiores cargas - e mudanças no sistema operacional das máquinas clientes. Esta última opção traz benefícios no que diz respeito à minimização de gargalos existentes no S.O como: troca de contexto em excesso e número máximo de descritores de arquivos abertos. Entretanto, essa opção se mostra bastante complexa pois não são todos os S.O.s que permitem alterações em seu modo de funcionamento e quando permitem, isso ocorre de forma bastante complexa. A segunda opção, sem dúvida, acrescenta funcionalidades à ferramenta e facilita o trabalho do usuário que pretende executar testes de forma distribuída. No entanto, em casos como o httperf, mesmo sem esta funcionalidade e com um pouco mais de trabalho, o usuário pode conseguir testes distribuídos. A primeira opção, de se utilizar novos parâmetros de caracterização traz o seguinte problema: Como conseguir estes parâmetros novos? Ao mesmo tempo, ela traz a vantagem de que, uma vez executada a implementação, novas cargas podem ser conseguidas por meio de novos parâmetros e não por meio de novas implementações. Uma boa resposta para a pergunta colocada são os logs de servidores web, pois eles possuem informações a respeito de tudo o que foi presenciado pelo servidor em um determinado período de tempo. Assim, a partir da interpretação do log pode-se reproduzir aquela carga registrada.

A inclusão da capacidade de leitura de log no httperf pode parecer, num primeiro momento, redundante, pois esse *benchmark* já possui uma maneira para fazer isto. En-

tretanto, a forma como isso é feito não é eficiente. Primeiro é necessário que o usuário transforme um arquivo de log, retirando praticamente todos os campos de cada registro, para que este possua apenas o nome das URLs presentes nesse arquivo. Com o arquivo construído, é necessário ainda criar e transferir para o servidor, todos os arquivos (URLs) que serão requisitados. Após todo esse processo, o teste poderá, finalmente, ser inicializado fazendo com que o httperf percorra o arquivo de URLs de forma seqüencial, requisitando objeto por objeto.

Com a utilização da ferramenta implementada neste trabalho, a tarefa se torna mais simples. É necessário apenas o log, no formato padrão CLF, que o resto é executado automaticamente. A ferramenta trabalha o log, gera um histograma, cria todos os arquivos (URLs) representativos, e provê uma forma de o httperf ter acesso ao histograma no momento do teste. Com isso, o *benchmark* não apenas percorre um arquivo de forma seqüencial, mas utiliza uma representação da distribuição dos tamanhos dos objetos presentes no log tratado.

O algoritmo implementado pode ser dividido em três fases que funcionam basicamente da seguinte forma:

1. **Análise do log**. A primeira fase consiste de uma análise de todo o log, com o objetivo de retirar os registros que se encontram com erro. Os registros considerados corretos são então convertidos para um formato mais simples.

2. **Geração do histograma**. O log convertido é todo colocado em uma estrutura, na memória. Com a estrutura gerada, monta-se o histograma que consiste de uma função de distribuição acumulada do tamanho dos arquivos presentes no log. Cada faixa desse histograma possui um tamanho de arquivo associado e um peso que indica a sua "participação" em relação ao total.

3. **Geração dos arquivos**. A geração do histograma engloba a associação de um arquivo para cada faixa existente. Esses arquivos ainda não existem, precisam ser gerados. Isto é feito nesta última fase, completando assim, todo o tratamento do log.

Ao final da utilização da ferramenta, o usuário possui um histograma representativo daquele log tratado, podendo inicializar o httperf com o conjunto de parâmetros corretos para executar o teste.

5.2 Logs de servidores web

Os logs constituem uma dentre várias maneiras de se medir o tráfego na Web (Arlitt & Williamson, 1996; Krishnamurthy & Rexford, 2001). Esses logs, ou rastros, podem

ser coletados nos servidores, em proxies ou nos próprios clientes. Neste trabalho serão utilizados apenas logs registrados por servidores.

Um servidor Web normalmente gera um log como parte do processamento dos pedidos do cliente. Cada entrada no log corresponde a um pedido HTTP tratado pelo servidor, incluindo informações sobre o cliente solicitante, o horário do pedido e as mensagens de pedido e resposta.

A maioria dos servidores Web realiza o *logging* (registro em log) por padrão. No entanto, na prática, os rastros do servidor não oferecem informações muito detalhadas. Isso ocorre porque quanto maior o número de informações registradas, maior será a sobrecarga no servidor. Por isso, normalmente são registradas apenas informações como o endereço do cliente, método do pedido, URL do pedido, código de resposta e tamanho do arquivo requisitado.

Embora nenhum padrão formal dite os formatos do log, a maioria das implementações segue normas informais para os dados que devem ser registrados. Um tipo de log bastante utilizado é o *Common Log Format* (CLF) descrito a seguir.

Common Log Format - CLF

Cada entrada em um log CLF consiste de sete campos, listados na Tabela 5.1 e descritos abaixo.

Campo	Significado
Remote Host	Nome de host ou endereço IP do cliente solicitante
Remote Identity	Conta associada à conexão na máquina do cliente
Authenticated User	Nome fornecido pelo usuário para autenticação
Time	Data e hora associada com o pedido
Request	Método de pedido, URL do pedido e versão do protocolo
Response Code	Código de resposta HTTP com três dígitos
Content Length	Número de bytes associados com a resposta

Tabela 5.1: Campos presentes em um log CLF

- **Remote Host**: Registra o nome, ou o endereço IP, da máquina cliente. O endereço IP pode ser registrado, diretamente, através do socket HTTP. O registro de nomes, no entanto, requer uma consulta a um servidor DNS para obter este nome através do endereço IP. A conversão de IP em nome fica normalmente desativada por aumentar significativamente o tempo de atendimento da requisição.

- **Remote Identity**: Indica o proprietário, na máquina cliente, associado à conexão TCP. Novamente, a obtenção desse dado implica em consultas a hosts remotos que demoram para responder ou nem respondem. Assim, a maioria dos servidores Web não utiliza esse campo.

- **Authenticated User**: Nome de usuário fornecido no cabeçalho HTTP para autenticação pelo servidor. Se o servidor não utiliza informações de autenticação, esse campo não é registrado.
- **Time**: Registra a hora do pedido na granularidade de um segundo.
- **Request**: Corresponde à primeira linha do cabeçalho HTTP que contém o método do pedido, URI do pedido e a versão do protocolo.
- **Response Code**: Código de resposta de três dígitos, com 200 para uma resposta correta e 404 para URI não encontrada, dentre outros.
- **Content Length**: Indica o número de bytes associados à resposta. Para respostas com código 404 ou 304, esse tamanho é igual a 0 pois nenhum conteúdo é retornado.

A Figura 5.1 ilustra parte de um log em formato CLF. Pode-se perceber os sete campos em cada um desses registros, sendo o segundo e o terceiro campos iguais a "-", pois o servidor não registrou essas informações.

```
... ... ... ...
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET / HTTP/1.1"200 736
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET /principal.htm HTTP/1.1"200 2422
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET /lateral.htm HTTP/1.1"200 401
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET /navemundo.gif HTTP/1.1"200 3329
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET /branco.gif HTTP/1.1"200 807
200.192.103.8 - - [02/Oct/2002:10:24:43 -0300] "GET /slogan.gif HTTP/1.1"200 1142
200.192.103.8 - - [02/Oct/2002:10:24:44 -0300] "GET /cubopreto.gif HTTP/1.1"200 818
... ... ... ...
```

Figura 5.1: Exemplo de um log em formato CLF

No presente trabalho foram utilizados três logs distintos. Cada um deles é descrito em detalhes nas próximas seções.

5.2.1 log do site da Copa do Mundo de 1998

A Copa do Mundo de 1998 foi realizada na França durante o período de 10 de junho a 12 de julho. O site construído possibilitava aos usuários acessar diversos serviços como resultados de jogos em tempo real, estatísticas sobre jogadores, história dos times e estádios, além de uma grande quantidade de imagens e sons. Para suportar o tráfego esperado durante o evento foi construída uma estrutura com 30 servidores, distribuídos em 4 localizações diferentes, uma na França e três nos Estados Unidos.

Os logs foram coletados inicialmente no formato CLF e, posteriormente, convertidos para um formato binário por razões de espaço de armazenamento. Seus registros representam a atividade de todos os servidores Web no período observado, entre 30 de abril e

26 de julho de 1998, alcançando um total de cerca de 1,3 bilhão de registros. A base de dados encontra-se dividida em arquivos de 7 milhões de registros cada, correspondendo a partes de um dia de coleta. Neste trabalho, foi utilizada uma parte dessas de cada vez.

Em relação à carga registrada pelo log, pode-se afirmar que a grande maioria das requisições usou o método GET (99,88%) para obtenção dos dados, seguido por um pequeno percentual de métodos HEAD (0,1%) e POST (0,02%). Analisando, em seguida, os códigos de resposta checou-se que a maioria das requisições resultou em sucesso (código 200). O segundo código de resposta mais comum foi o 304(*not-modified*). Pôde-se perceber, ainda, uma pequena quantidade de acessos que resultaram em erro, código de resposta igual a 404 (*file not found*).

A Tabela 5.2 mostra uma divisão das respostas por tipo de arquivo requisitado pelo usuário. Ela revela que quase todos os clientes (98.01%) fizeram requisições a arquivos em HTML ou imagens. Das requisições restantes, o tipo mais freqüente foi o de arquivos em Java. Tanto as requisições a audio e vídeo, como as de natureza dinâmica, representaram uma parcela ínfima do total. Em relação à quantidade de bytes transferidos, houve um predomínio de arquivos em HTML sobre as imagens. Isso se deveu, em parte, pelo fato de os arquivos HTML serem maiores do que as imagens, em média; além disso, muitas requisições de imagens resultaram em respostas sem dados, do tipo *Not Modified*, ou seja, o arquivo, por não ser modificado com freqüência, não precisou ser enviado novamente pelo servidor. Nota-se também que os arquivos comprimidos, embora respondam por uma parcela muito pequena dos acessos, compreenderam um valor significativo, 20%, do total de bytes transferidos. A discrepância entre a porcentagem de requisições a arquivos comprimidos e a porcentagem de dados transferidos por estas requisições é uma indicação dos efeitos que grandes arquivos podem causar na carga de um servidor Web e de uma rede.

Tipo de Arquivo	% de requisições	% de dados transferidos
HTML	9,85	38,60
Imagens	88,16	35,02
Audio	0,02	0,10
Vídeo	0,00	0,82
Compr.	0,08	20,33
Java	0,82	0,83
Dinâmicos	0,02	0,38
Outros	1,05	3,92
Total	100,00	100,00

Tabela 5.2: Tipos de arquivos requisitados - Copa do Mundo

5.2.2 log do site Navemundo

Navemundo é um portal que disponibiliza a seus usuários, informações diversas como Emprego, Saúde, Tecnologia, Negócios e Música. Esse portal concentra diversos canais de informação que respondem a seus usuários através de domínios como `www.redenegocios.com.br`, `www.aforja.com.br` e o próprio `www.navemundo.com.br`. Redenegócios está relacionado a economia e mercado, oferece um serviço de assessoria e provê informações como cotações, tarifas públicas, investimentos e empresas. Aforja concentra informações relacionadas a música, tais como notícias a respeito de bandas, entrevistas e críticas sobre discos e shows.

A estrutura montada para hospedar o portal e seus domínios constitui-se de dois servidores - um para atendimento aos usuários e outro para auditoria dos domínios existentes.

O log foi coletado no formato CLF, com seus registros representando a atividade do servidor Web no período observado entre 28 de setembro e 05 de outubro de 2003, alcançando um total de cerca de 885.640 registros.

Em relação à carga, pode-se dizer que a grande maioria das requisições usou o método GET (99,82%) para obtenção dos dados, seguido por um pequeno percentual de métodos HEAD (0,01%) e POST (0,15%). Analisando, em seguida, os códigos de resposta notou-se que a maioria das requisições resultou em sucesso (código 200). O segundo código de resposta mais comum foi o 304 (*not-modified*). Pôde-se perceber ainda uma pequena quantidade de acessos que resultaram em erro, código de resposta igual a 404 (*file not found*).

A Tabela 5.3 mostra uma divisão das respostas por tipo de arquivo requisitado pelo usuário. Ela mostra que a grande maioria dos clientes (89,2%) fizeram requisições a arquivos em HTML ou imagens. Das requisições restantes, o tipo mais freqüente foi o de requisições dinâmicas. Em relação à quantidade de bytes transferidos, houve um predomínio de arquivos em HTML sobre as imagens. Isso se deveu, em parte, pelo fato de os arquivos HTML serem maiores do que as imagens, em média; além disso, muitas requisições de imagens resultaram em respostas sem dados, do tipo *Not Modified*, ou seja, o arquivo, por não ser modificado com freqüência, não precisou ser enviado novamente pelo servidor. Nota-se também que as requisições dinâmicas compreenderam um valor significativo, 8,6%, do total de bytes transferidos.

5.2.3 log de testes com o WebStone

Durante o andamento do presente trabalho, foram executados diversos testes e experimentos com os *benchmarks* mencionados. O servidor Web Apache, estava presente em todas as execuções, e como é padrão neste servidor, foram registrados logs desses experimentos.

Tipo de Arquivo	% de requisições	% de dados transferidos
HTML	21,96	52,23
Imagens	67,24	34,18
Audio	0,0	0,0
Vídeo	0,0	0,0
Compr.	0,02	1,54
Dinâmicos	6,70	8,64
Outros	4,06	3,38
Total	100,00	100,00

Tabela 5.3: Tipos de arquivos requisitados - Navemundo

Como um dos objetivos deste trabalho é utilizar diferentes logs para testar servidores, optou-se por utilizar também esses logs registrados durante os experimentos. A coleta ocorreu após a execução de testes com o WebStone em que foram utilizadas quatro máquinas - três clientes e um servidor. O arquivo final apresentou mais de 230.000 registros. Em relação à carga, diferentemente dos logs já mencionados, este apresenta, em sua totalidade, requisições do tipo GET; nenhuma requisição errada - que tenha retornado com código 404; apenas requisições estáticas e nenhum ítem de tamanho 0.

Embora este log não represente uma situação real, ele tem sua validade no que diz respeito a comparações entre os resultados obtidos para esse caso especial e os casos citados acima.

5.3 Implementação

A implementação da nova característica do *benchmark* foi alcançada através da construção de uma ferramenta de tratamento de logs e da modificação do httperf para que este fosse capaz de utilizar o resultado gerado pela ferramenta.

De acordo com a geração de carga do httperf, abordada no Capítulo 4, a inclusão do novo parâmetro acarretou na criação de uma nova forma de definir a URL que deve ser requisitada. A nova forma foi intitulada "uri_histogram" e definida como mostra a Figura 5.2.

O algoritmo ilustrado na Figura 5.2 funciona da seguinte forma: Primeiro, fase de inicialização, o arquivo contendo o histograma é lido e mapeado na memória como um vetor. Esse vetor é montado de forma que cada posição dentro dele represente uma faixa do histograma, e ainda apresente o nome do arquivo associado a essa faixa.

Com o vetor na memória, a definição da URL é feita através de um número aleatório, gerado de acordo com uma distribuição uniforme no intervalo de 0 a 1. Verifica-se, então, em qual posição do vetor esse número se encaixa, retornando assim, o arquivo associado

```
HISTOGRAMA *vetor;
init_uri_histogram(){
   int i = 0;

   abra arquivo que contém histograma;
   Enquanto não é final de arquivo
      leia uma linha do arquivo;
      converta campos string para números;
      vetor[i].valor = valor_convertido;
      vetor[i].arquivo = nome do arquivo representativo;
      i++;
   fim-Enquanto;

   passe semente para gerador de números aleatórios;
   registre função 'set_uri' como manipulador para evento de criação de nova requisição;
}

set_uri(){
   double result;
   char *uri;
   int len, i = 0;

   verifique se evento et é mesmo um evento de nova requisição;

   result = número aleatório gerado;
   Enquanto(result > vetor[i].valor)
      i++;
   uri = vetor[i].arquivo;
   passe uri para ser requisitada ao servidor;
}
```

Figura 5.2: Algoritmo acrescentado ao httperf

àquela posição ou faixa. Esse arquivo é o que será requisitado ao servidor.

Além das modificações no httperf, o tratamento de logs ainda exigiu a criação de uma ferramenta que, como mencionado, pode ser considerada em três tempos, a saber: Análise do Log, Construção do Histograma e Geração dos Arquivos.

5.3.1 Análise do log

Neste momento o log é lido pela primeira vez, em seu formato inicial, contendo, possivelmente, diversos erros e registros defeituosos. Cabe a essa parte da ferramenta verificar se cada registro do log atende ao padrão especificado abaixo:

```
<host> <identidade> <usuário> <tempo> <requisição>
  <código de resposta> <tamanho>
```

Dessa forma, cada entrada do log possui um nome de host ou endereço IP do cliente (host); uma conta associada à conexão na máquina do cliente (identidade); o nome fornecido pelo usuário para autenticação (usuário); a data/hora associada com o pedido

(tempo); o método do pedido, URL do pedido e versão do protocolo (requisição); código de resposta HTTP com três dígitos (código de resposta); e, finalmente, o número de bytes associados à resposta (tamanho).

É considerado defeituoso todo registro que não possuir algum desses campos ou que possua um campo com formato diferente do esperado. Como por exemplo, um campo de tamanho com formato não numérico, ou um código de resposta fora do intervalo que vai de 100 a 599. Todos os registros defeituosos são ignorados.

Os registros restantes são então convertidos para um formato mais simples, descartando os campos "identidade" e "usuário". Além disso, o campo "tempo" é convertido para o formato de segundos desde as 00:00:00 de 1/1/1970.

Assim, o "novo" log criado através do tratamento e conversão iniciais, fica com o formato:

```
<host> <tempo> <requisição> <código de resposta> <tamanho>
```

Um último ponto a se considerar durante esta fase é o registro das requisições com **maior** e **menor** tamanhos. Esses valores servirão de base para a criação do histograma na próxima fase.

5.3.2 Construção do histograma

Cada registro do log, convertido e sem erros, é agora lido e colocado em uma estrutura, definida como VETOR, na memória. Esse VETOR (será discutido mais detalhadamente na próxima seção) constitui-se de uma lista de listas, onde os registros são separados por faixas, de acordo com o valor de seus campos "tamanho".

Com o VETOR todo montado, a construção do histograma é obtida, basicamente, com o mapeamento dessa estrutura de uma nova forma. Isso ocorre porque o histograma utiliza faixas de tamanhos de arquivos, assim como no VETOR. A diferença aqui está no fato de que na estrutura está presente cada registro do log, já no histograma, as faixas possuem "apenas" um valor representativo em relação ao total.

Essa nova forma de representação do VETOR é concretizada por uma nova estrutura, que é utilizada para representar o histograma. Essa estrutura constitui-se de um vetor de faixas, tendo cada faixa dois campos, arquivo e peso, que armazenam respectivamente o nome do arquivo representativo da faixa e a sua representação, em relação ao total, no log.

O peso de cada faixa é obtido através da divisão do número de registros que caíram naquela faixa pelo número total de registros em todo o log. Esses pesos são então somados de forma a se obter uma função de probabilidade acumulada, que é o próprio histograma. A Figura 5.3 mostra um exemplo.

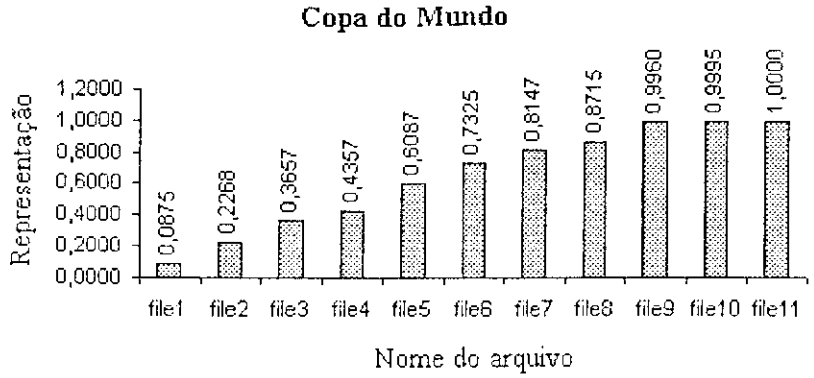


Figura 5.3: Histograma criado a partir do log da Copa do Mundo

Por fim, o histograma é gravado em um arquivo para que possa ser utilizado pelo httperf no momento do teste.

Estrutura Vetor

A necessidade de se trabalhar com os registros do log de forma ágil, tornou necessário manter esses registros em memória. Como a intenção principal é dividir todo o log em faixas, quanto mais próximo disso a estrutura se apresentar, melhor.

Assim, as estruturas ilustradas na Figura 5.4 foram criadas para o armazenamento das informações desejadas. Posteriormente foram definidas as regras de tratamento de cada uma delas.

REQ contém os dados de cada registro presente no log, incluindo todos os campos do registro mencionados acima. Cada NO possui uma estrutura REQ e mais algumas informações de controle. Os NOs são arranjados em LISTAs, que possuem ainda a quantidade de elementos presentes nela e funções de manipulação. Toda LISTA está associada a um INDICE. No INDICE há um valor superior que estabelece quais os registros que entram em sua LISTA. Todos os INDICEs, por sua vez, são organizados em uma lista controlada pela estrutura VETOR. A Figura 5.4 apresenta uma visão completa da estrutura.

A construção do VETOR inicia-se pelo estabelecimento de quantas faixas serão criadas no histograma. O número de faixas estabelece o número inicial de índices. Cada índice possui um valor agregado, chamado valor superior, que varia do menor ao maior tamanho de requisição presentes no log (registrados na primeira fase). Com isso, se os valores menor e maior fossem respectivamente 0 e 100, e fossem criadas 2 faixas, por exemplo, seriam criados dois índices com valores superiores iguais a 50 e 100. Assim, quando os registros fossem incluídos no VETOR, todos que possuíssem tamanho menor que 50 ficariam na lista associada ao primeiro INDICE, e os de tamanho entre 50 e 100, estariam no segundo INDICE.

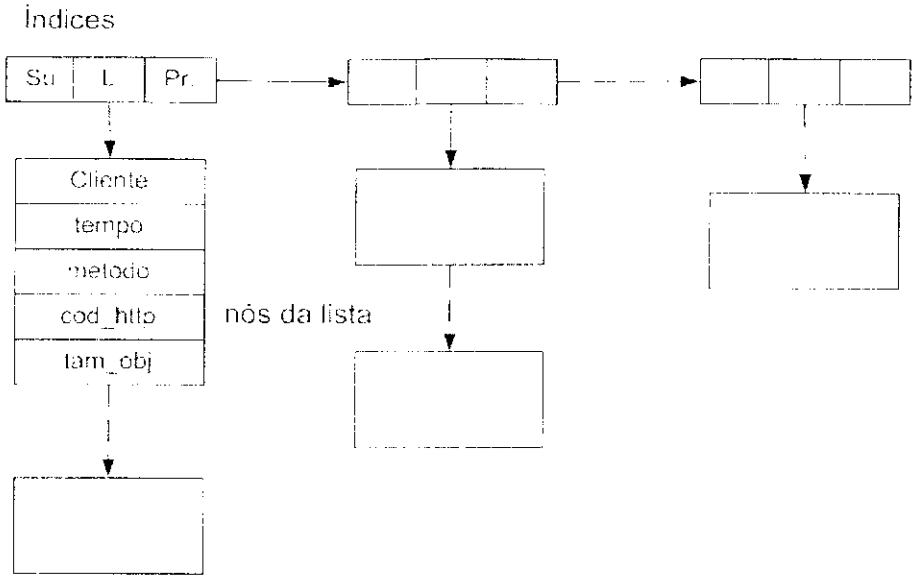


Figura 5.4: Estrutura VETOR.

A implementação executada neste trabalho apresenta duas formas de criação dessas estruturas INDICE. A mais simples delas é a divisão do intervalo [min, max], em um número, especificado pelo usuário, de faixas iguais. O exemplo citado acima trabalha dessa forma.

A outra forma se vale de valores pré-definidos para dividir os arquivos em diferentes faixas de tamanho. São utilizados os valores 500B, 5KB, 50KB, 500KB e 5MB (valores utilizados pelo WebStone). Dessa forma, a primeira faixa vai de 0 a quinhentos bytes, a segunda de quinhentos bytes a 5 kilobytes e assim sucessivamente até a última faixa que, para logs que apresentam arquivos maiores que 5MB, vai de cinco megabytes até o tamanho do maior arquivo.

De uma forma ou de outra, nesse momento tem-se o VETOR constituído de uma lista de índices vazios. É necessário ainda incluir os registros do log na estrutura. Os registros são então, um a um, inseridos de acordo com o seu tamanho. Utilizando a segunda forma de divisão, todos os registros com tamanho abaixo de 500 bytes são colocados na primeira faixa, ou seja, na lista associada ao primeiro índice. Da mesma forma, os arquivos com tamanho maior que 50KB e menores que 500KB entram na lista associada ao índice que possui valor superior igual a 500KB.

Após a inserção de todos os registros, é de se esperar que alguns índices possuam suas listas maiores que a lista de outros. Ou até mesmo índices completamente vazios. Para cada um desses casos há uma forma de tratamento específica. Os INDICEs que permaneceram vazios são eliminados do VETOR. Para os que apresentaram uma lista muito maior do que a dos outros índices, há uma redivisão. Este processo também é controlado pelo usuário, podendo este definir qual a representação máxima que uma única faixa pode

alcançar. Se o usuário definir, por exemplo, que não quer faixas com representação maior do que 20%, após a inserção dos registros, os nós INDICE que estiverem acima de 20% serão redivididos, até que se alcance o valor desejado.

5.3.3 Geração dos arquivos

Durante a construção do histograma, foi associado a cada faixa, um arquivo representativo. Como mencionado, esse arquivo é a URL que será requisitada ao servidor durante o teste. Como esses arquivos ainda não existem, eles precisam ser gerados.

Cada faixa possui dois valores delimitantes. Um valor inferior e um valor superior. Como a URL deve representar essa faixa, foi escolhido como tamanho do arquivo, o valor médio entre esses dois valores.

Com o tamanho definido, são criados arquivos e estes preenchidos com caracteres até que se alcance o tamanho desejado. Ou seja, são criados arquivos texto, com o tamanho determinado pela faixa a que pertence. O fato de serem criados arquivos texto, mesmo quando os logs apresentam imagens, vídeos e outros tipos de dados, não influencia nos resultados pois para o objetivo do trabalho, o mais importante é o tamanho e não o tipo de arquivo tratado. Apenas trata-se tudo como bits.

Posteriormente esses arquivos devem ser transferidos para o servidor, para que este possa responder às requisições dos clientes.

5.4 Testes e experimentos

Como discutido no decorrer desta dissertação, os benchmarks para servidores Web não apresentam alguns pontos importantes na geração de cargas utilizadas durante os testes. Com isso, a tentativa de se alcançar uma carga o mais próximo do real possível é sempre um problema. O presente Capítulo apresentou, com maiores detalhes, uma forma de alcançar cargas que, se ainda não são ideais, apresentam-se mais perto desse objetivo do que as cargas já oferecidas pelas ferramentas mencionadas.

Nos ítens seguintes são apresentados diversos experimentos onde procurou-se avaliar os resultados obtidos com o *benchmark* modificado. O fato mais importante não são "apenas" os resultados, mas a forma utilizada para se chegar até eles, o que garante uma maior validade dos dados.

5.4.1 Plataforma utilizada para realização dos experimentos

Tanto a implementação como os experimentos apresentados neste Capítulo foram realizados em uma rede de computadores pessoais em que todos utilizavam o sistema ope-

racional Linux. Mais precisamente, foi utilizada a distribuição RedHat (Red Hat, 2003), com o kernel 2.4.7. A Tabela 5.4 apresenta mais informações sobre a configuração das máquinas utilizadas. Em todos os experimentos realizados, as máquinas utilizadas foram isoladas, sendo assim o tráfego existente na rede devido apenas à execução do próprio experimento.

Máquina	Processador	Memória	Interface de Rede
Lasdpc14	AMD Duron 1,2GHz	256MB	100Mb
Lasdpc15	Pentium IV 2GHz	256MB	100Mb
Lasdpc17	AMD Athlon XP 2GHz	256MB	100Mb

Tabela 5.4: Máquinas utilizadas para execução dos experimentos

Em relação aos *softwares* utilizados, optou-se por utilizar o servidor Web Apache (versão 2.0), por ser a última versão do servidor http mais utilizado atualmente; e o httperf modificado, uma vez que este é o principal objeto de estudo do trabalho.

5.4.2 Descrição dos experimentos

Devido à criação de uma nova ferramenta, tornou-se necessário executar, primeiramente, uma série de testes para verificar seu funcionamento. Assim, foram tratados diversos logs com esta ferramenta, dentre eles, os já mencionados em maiores detalhes. Para cada log tratado foram obtidos, como resultado, um arquivo com características específicas do log, um histograma e um arquivo representativo para cada faixa do histograma.

Após o teste e validação da ferramenta, de posse dos histogramas e arquivos gerados, deu-se início aos testes envolvendo o httperf e o servidor http.

Em cada teste foram utilizadas inicialmente apenas duas máquinas, o servidor e um cliente. Posteriormente observou-se que a mudança de um para dois clientes causava mudanças significativas nos resultados, mas que, no entanto, a utilização de três clientes não acrescentava muito em relação a dois. A única alteração era a chegada mais rápida ao ponto de saturação do servidor, isto é, os erros de *timeout* reconhecidos pelos clientes aumentava rapidamente. Por isso a maioria dos testes foram realizados com a utilização de 3 máquinas - um servidor e dois clientes. Em relação ao número de processos, como explicado no Capítulo 4, execuções do httperf utilizam apenas 1 processo por máquina.

O objetivo principal a ser alcançado com os experimentos é a verificação de como será o comportamento do *benchmark* trabalhando de acordo com o novo método implementado, isto é, como a nova carga de trabalho influencia nos testes.

Valendo-se dos parâmetros disponibilizados pelo httperf, descritos na seção 4.3.1 os testes realizados foram definidos da seguinte forma: partiu-se de uma taxa de 100 requisições por segundo que era então incrementada para 200, 300, 400, e assim sucessivamente

até se atingir 1000 requisições por segundo, quando passou-se a incrementar o número de requisições com 200 unidades até se alcançar o máximo de 1400 requisições por segundo. Dessa forma, os testes realizados com cada log consistia de 12 passos (100 a 1400req/s).

As outras variantes nos experimentos são o histograma utilizado e os arquivos requisitados ao servidor. Estes pontos dependem do log em uso no momento, ou seja, quando utilizando-se o log da Copa do Mundo, por exemplo, o histograma referente a este log era passado ao httperf e os arquivos possíveis de serem requisitados eram copiados para o servidor. A seguir são apresentados os resultados obtidos.

5.4.3 Resultados

Os gráficos seguintes ilustram os resultados obtidos através do tratamento dos diferentes logs, pela ferramenta construída.

A diferença mais aparente entre os resultados, notada a primeira vista, é aquela que diz respeito ao número de classes obtidas. No caso do WebStone, como foram utilizados cinco arquivos diferentes durante os testes que originaram o log, era esperado que o resultado também apresentasse cinco classes. Para os outros dois casos, que representam o comportamento de servidores reais, o número de classes foi maior pois estes logs apresentam maior diversidade de arquivos, desde pequenas imagens até videos, além de respostas do tipo 404 e 304 que não aconteceram no caso do WebStone.

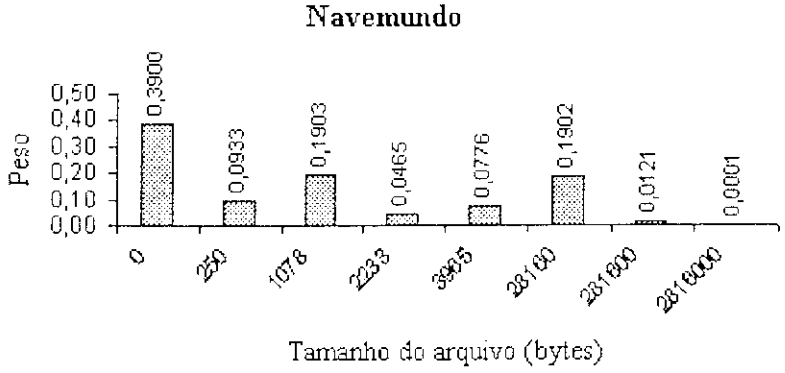


Figura 5.5: Distribuição de arquivos - log Navemundo

O gráfico da Figura 5.5 mostra que 39% das requisições registradas, retornaram com campo de tamanho igual a 0. Como explicado, isso se deve ao fato de existirem requisições que retornam com códigos 304 e 404. E a classe que representa os arquivos de maior tamanho, isto é, de aproximadamente 2MB, aparece apenas com um peso insignificante em relação ao total.

No caso da Figura 5.6 pode-se perceber a existência de 11 classes, sendo a mais "contemplada", aquela cujos arquivos possuem um tamanho entre 934 e 1367 bytes. É ilustrado

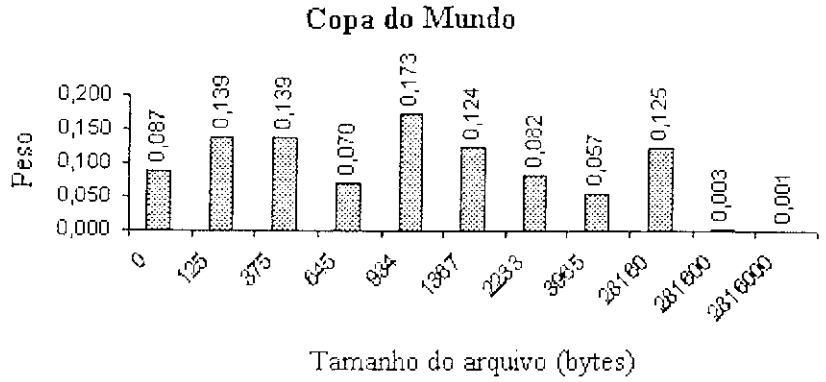


Figura 5.6: Distribuição de arquivos - log Copa do Mundo

ainda que quase 9% das requisições registradas, retornaram com campo de tamanho igual a 0, e que os arquivos de tamanho maior que 2MB representaram menos de 1%.

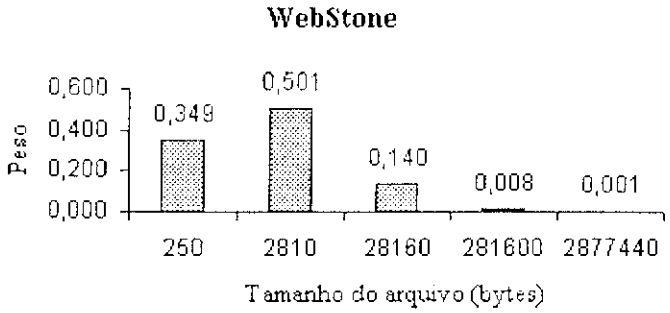


Figura 5.7: Distribuição de arquivos - log WebStone

Um fato importante a ser notado no gráfico da Figura 5.7 é a inexistência de uma classe para arquivos de tamanho 0. Isso ocorreu porque no teste não foram feitas requisições com "GET condicional", o que possibilitaria um código de retorno igual a 304, e todos os arquivos requisitados existiam no servidor, o que impossibilita o retorno de erros do tipo 404. Outro ponto a ser destacado é a maior concentração das faixas, em relação aos outros logs, presente neste caso. Além de possuir menos classes, o que justifica, em parte, a presença de classes de maior peso, cada faixa é representada por um único arquivo, tornando assim a variância igual 0. Dessa forma, fica impossível redividir as faixas para gerar mais classes.

Em relação aos três gráficos de distribuição de arquivos, vale ressaltar que em todos os casos, os arquivos de 2KB ou menos representaram algo em torno de 80%, o que reforça a observação de que a carga na Web é constituída de arquivos pequenos. Além disso, a presença de classes com grande peso em relação ao total, se deve ao algoritmo utilizado para redivisão. Este algoritmo, como explicado para o log WebStone, nada pode fazer no caso de classes que apresentam pequena, ou nenhuma, variância entre seus objetos.

A distribuição de arquivos, representada pelas figuras já citadas, constitui um passo intermediário em todo o algoritmo desenvolvido. Isto porque o que é realmente utilizado nos testes são as representações ilustradas nas Figuras 5.8, 5.9, e 5.10, onde cada uma mostra uma função de distribuição acumulada. Essa função estabelece a probabilidade de um arquivo ser requisitado durante a execução do experimento.

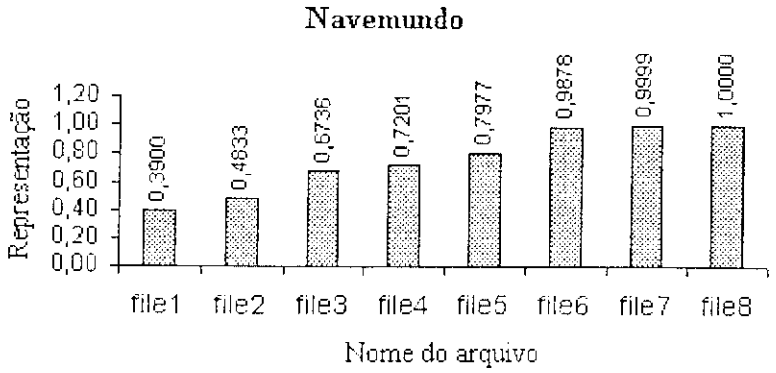


Figura 5.8: Histograma para o log Navemundo

Na Figura 5.8, pode-se perceber que o arquivo "file1", será requisitado em 39% das oportunidades. Relacionando as Figuras 5.5 e 5.8, verifica-se que o arquivo "file1" representa a classe de arquivos de tamanho 0. Já a segunda classe, de arquivos com tamanho entre 250 e 1078 bytes, identificada pelo arquivo "file2", será contemplada em, aproximadamente, 10% (0,4833 - 0,39) das oportunidades.

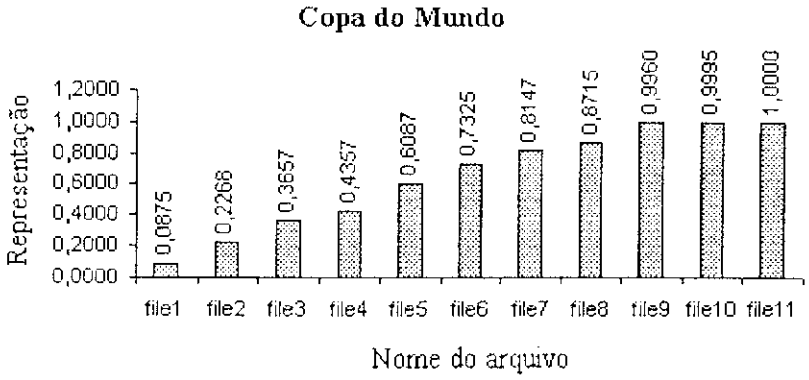


Figura 5.9: Histograma para o log Copa do Mundo

As Figuras 5.9 e 5.10, apresentam gráficos bastante distintos, tanto pelo número de classes existentes, quanto pela representação de cada uma dessas classes. O tratamento do log da Copa do Mundo gerou mais faixas, cada uma com pequena representação, enquanto que no caso do WebStone houve uma grande concentração, existindo uma faixa com representação acima de 50%, "file2". Como mencionado acima, isso ocorreu devido a

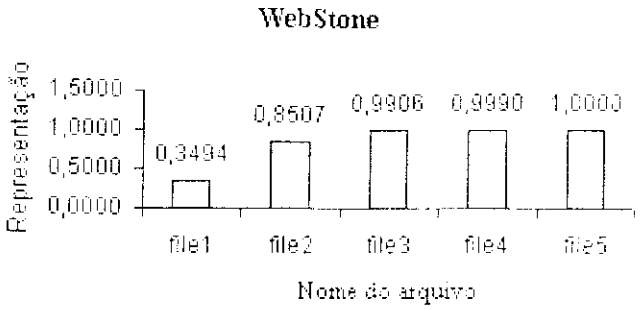


Figura 5.10: Histograma para o log WebStone

grande diferença, em termos de quantidade de objetos distintos, e conseqüente variância, apresentada pelos logs em questão.

Os histogramas, ou funções de distribuição acumulada, recém expostos, foram então utilizados nos experimentos de avaliação do servidor web Apache. Estes experimentos obtiveram como resultados, os dados expostos nas figuras 5.11 a 5.16. Os gráficos apresentam os valores médios - linha sólida - obtidos após 15 execuções de cada um dos experimentos. São ilustradas ainda duas linhas pontilhadas: uma para valores mínimos e outra para valores máximos constituindo o intervalo de confiança estipulado. O intervalo de confiança - linhas pontilhadas - foi calculado com um nível de confiança igual a 0.05.

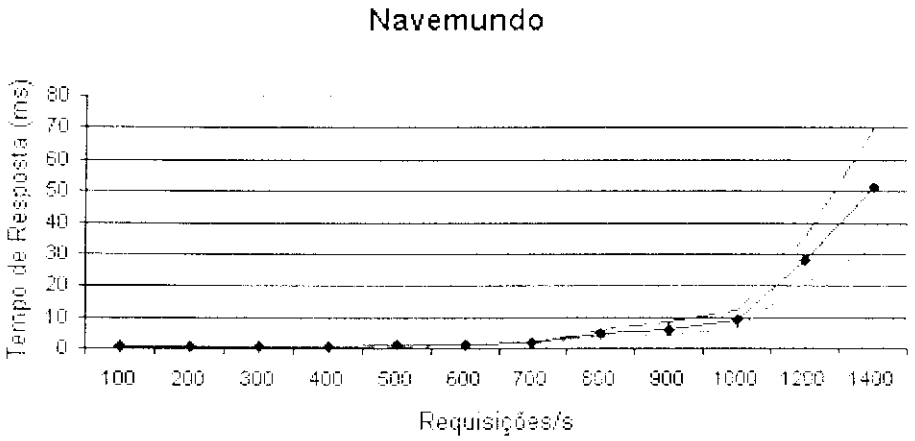


Figura 5.11: Tempo de resposta - log Navemundo

A Figura 5.11 expõe a progressão do tempo de resposta, para o log Navemundo, alcançado pelo servidor durante o teste. Os valores apresentados representam o intervalo de tempo, visto pelo cliente, entre a execução de uma requisição e o início do recebimento da resposta. O gráfico apresenta um crescimento uniforme, ou seja, aumentando-se o número de requisições por segundo, aumenta-se também o tempo de resposta.

O log da Copa do Mundo, Figura 5.12, apresentou um comportamento similar àquele

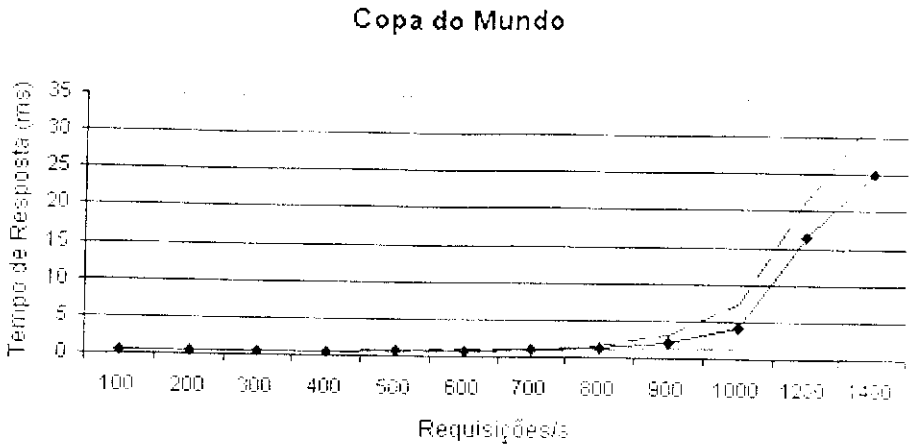


Figura 5.12: Tempo de resposta - log Copa do Mundo

apresentado pelo log Navemundo. Apesar de apresentar um crescimento também contínuo, os tempos de resposta encontrados foram significativamente menores do que naquele caso. Esta diferença pode ser explicada analisando-se novamente as Figuras 5.9 e 5.8. A verificação mais detalhada dessas figuras mostra que no primeiro caso, Copa do Mundo, os arquivos com tamanho acima de 2KB possuem uma representação de, aproximadamente, 26%. Já para o log Navemundo, essa representação sobe para 32%. Assim, com mais requisições a arquivos de maior tamanho, é normal que o tempo de resposta também seja maior.

É importante destacar que a variância do tempo de resposta é significativa apenas a partir do ponto em que o servidor começa a ficar sobrecarregado, isto é, quando começa a ocorrer erros. Isto pode ser observado por meio do intervalo de confiança que aumenta à medida que o tempo de resposta cresce.

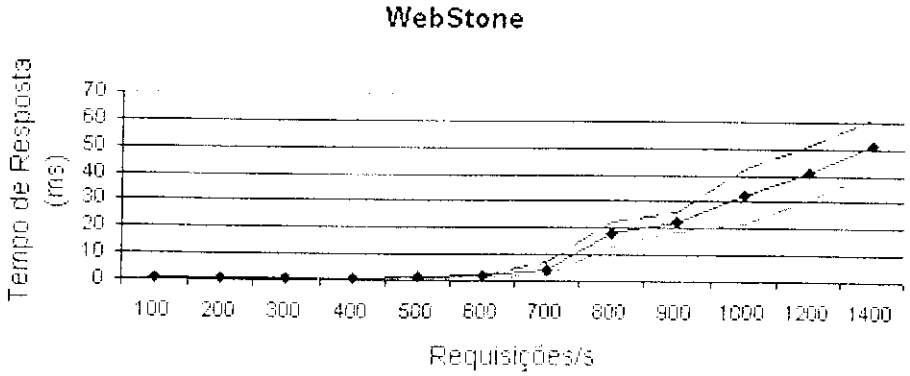


Figura 5.13: Tempo de resposta - log WebStone

O gráfico obtido para o log WebStone, Figura 5.13, também apresentou um comportamento análogo ao primeiro caso, ou seja, crescimento contínuo, mesmo que tenha sido

mais suave nesse caso.

As Figuras 5.14 a 5.16 mostram o número de erros obtido na execução do experimento para cada log utilizado. A verificação mais detalhada dessas figuras mostra que o comportamento dos dados permanece o mesmo daquele exposto nas figuras anteriores. Há um intervalo inicial onde não aparecem erros, onde também o tempo de resposta é pequeno, até se atingir um ponto em que o número de erros cresce rapidamente. Fica fácil inferir que nesse ponto o servidor alcançou seu ponto de saturação, isto é, a medida que se ultrapassa este ponto, a quantidade de erros e os tempos de resposta tendem a aumentar drasticamente.

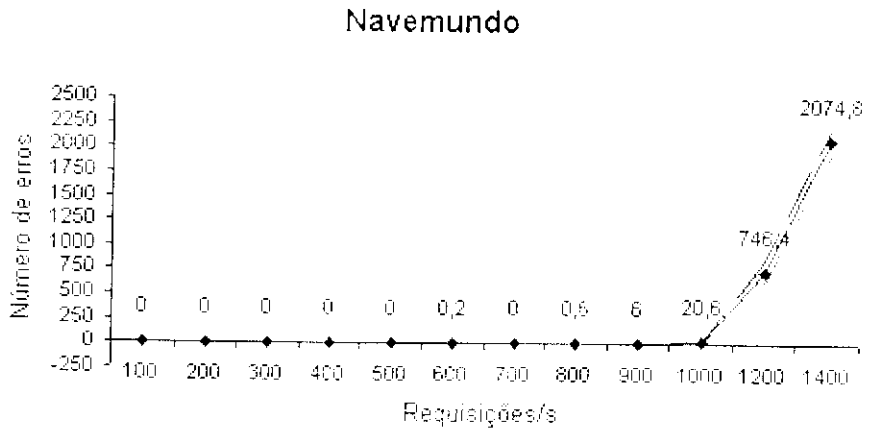


Figura 5.14: Número de erros - log Navemundo

Figura 5.15: Número de erros - log Copa do Mundo

Um ponto que merece destaque quando confrontando os gráficos das figuras citadas, é a "posição" que cada um ocupa em relação ao outro. Pode-se afirmar que o log da Copa do Mundo aparece na posição mais baixa pois é o caso que leva a menores tempos de resposta e menor número de erros. Com isso, pode-se afirmar que esse log "exige"

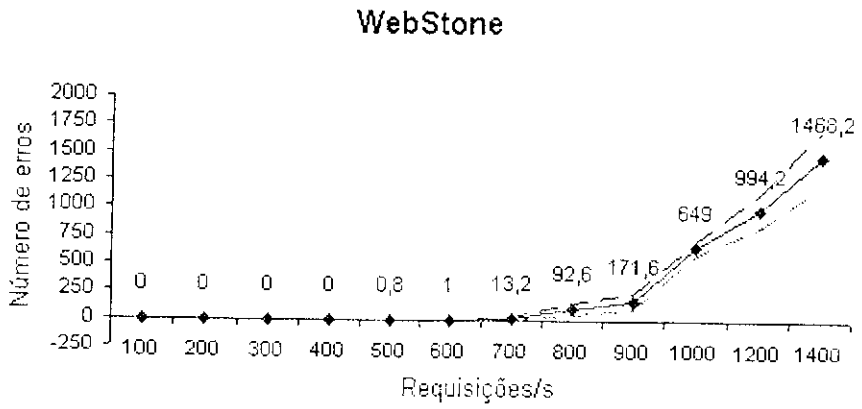


Figura 5.16: Número de erros - log WebStone

menos do servidor. Nas posições superiores aparecem, respectivamente, os logs WebStone e Navemundo. Sendo assim, o que gera mais carga no servidor, levando este a alcançar maiores tempos de resposta é o log Navemundo.

5.5 Considerações finais

A utilização de *benchmarks* representa um papel de considerável importância na avaliação de servidores Web. Estas ferramentas podem disponibilizar informações, de formas variadas, em relação ao desempenho do sistema em avaliação. No entanto, a forma como utilizar essas ferramentas - que parâmetros usar, por quanto tempo executar - pode ser vista como um obstáculo.

Na tentativa de facilitar o trabalho do usuário, alguns *benchmarks*, como o WebStone (seção 4.2), oferecem configurações prontas, para serem utilizadas em testes padrão. Assim, o usuário precisaria se preocupar com a configuração do *benchmark* apenas em testes específicos.

Com esse mesmo objetivo é que foram desenvolvidas a ferramenta e as alterações no httperf descritas neste Capítulo. Para conseguir parâmetros de configuração confiáveis, optou-se por utilizar logs de servidores como ponto de partida. Dessa forma, trabalhando com logs previamente coletados, foi possível a obtenção de valores próximos do real.

Tendo a ferramenta e o "novo" httperf implementados e prontos para execução, tornou-se necessário fazer suas validações. Isto foi feito através dos experimentos descritos na seção 5.4.

Além de servirem para validação da ferramenta e das implementações incorporadas ao *benchmark*, os resultados apresentados e discutidos confirmam que a utilização de logs em experimentos para avaliação de servidores Web é viável e bastante atrativa. Variando-se

o log utilizado, é possível obter, sem maiores complicações, cargas com as mais diferentes características.

Se as implementações descritas não existissem e fosse utilizado o httperf "original" para se obter os resultados mostrados na seção 5.4.3, o usuário precisaria, antes de tudo, definir uma carga de trabalho similar àquela utilizada para trabalhar. Isto poderia ser feito como no Capítulo 4, onde foram utilizados parâmetros do Surge para definir uma carga de trabalho para o httperf. No entanto, isso não elimina a necessidade de o usuário fazer, manualmente, todo o tratamento dos dados e extração desses parâmetros. Além disso, se a intenção for executar diversos testes, com diferentes cargas, o problema é maior pois todo o trabalho deve ser refeito para cada carga distinta.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Considerações finais

O trabalho discutido nesta dissertação abordou aspectos envolvendo o estudo de *benchmarks* para servidores Web com o intuito de identificar características importantes, presentes em sistemas reais, mas que não eram consideradas por essas ferramentas. Foi apresentada ainda a seleção de uma das ferramenta que recebeu, por meio de implementações, uma nova característica a ser utilizada em futuras avaliações. Para o desenvolvimento de todo esse processo, houve a necessidade de realizar estudos e avaliações visando escolher uma ferramenta que fosse mais adequada aos requisitos do trabalho. De forma mais específica, foram abordados os temas Avaliação de Desempenho (Capítulo 2), Web (Capítulo 3) e *Benchmarks* para servidores Web (Capítulo 4).

Após esses estudos, constatou-se a adequabilidade da ferramenta httperf para receber as novas características, e também a importância da interpretação de logs para extração de parâmetros de teste.

Como já visto, os logs são, normalmente, arquivos de grande tamanho. Por isso, a sua utilização direta na ferramenta, durante a execução de experimentos, é inviável. Este fato justificou a criação de uma ferramenta à parte, capaz de analisar o log, coletar as informações requeridas e gerar uma forma mais fácil, e rápida, de acessar essas informações durante os testes. Esta forma mais "fácil" foi obtida através de histogramas que representavam a distribuição, da característica escolhida, em todo o log. Assim, de forma geral, foi construída uma ferramenta que trabalhava o log e gerava histogramas. Estes eram então passados ao httperf para serem utilizados na geração de carga de trabalho em experimentos de avaliação de servidores. Os detalhes envolvendo a implementação da nova ferramenta e as mudanças feitas no httperf foram apresentados no Capítulo 5.

Ainda no Capítulo 5, podem ser vistos os resultados alcançados em experimentos feitos com o "novo" httperf. Esses resultados mostraram que a utilização de logs em experimentos para avaliação de servidores Web é bastante viável e atrativa. Com este procedimento,

fica fácil obter diferentes cargas para testes. É necessário, apenas, conseguir um log com as características desejadas que o restante do trabalho é executado pelos *softwares*.

6.2 Contribuições deste trabalho

O desenvolvimento de uma nova maneira para se alcançar cargas de trabalho mais próximas do real, com o intuito de avaliar servidores web, constitui uma das principais contribuições, uma vez que o método empregado neste trabalho não tem sido explorado pelas ferramentas disponíveis atualmente. De forma geral, outras contribuições relevantes são:

- Estabelecimento de conhecimento a respeito dos benchmarks utilizados, pois como mencionado quando da discussão dos mesmos, algumas dessas ferramentas possuem documentação incompleta e, às vezes, até incoerente. Além de ter havido a necessidade da correção do código fonte em alguns casos. Assim, este trabalho oferece, no mínimo, uma fonte de referência bastante ampla, servindo também como um manual de utilização para os *benchmarks* tratados.

- Construção de uma ferramenta independente utilizada para o tratamento de logs. Neste trabalho, a ferramenta foi usada para a construção de histogramas, mas nada impede que ela seja aproveitada para outros fins, como, por exemplo, a remoção de registros expúrios presentes em algum log.

- Alterações/adições executadas no httperf. A este *benchmark* foi acrescentada a capacidade de leitura (utilização em experimentos) dos histogramas gerados pela ferramenta já mencionada. Assim, de forma indireta, foi possível fazer com que o httperf se valesse de características extraídas de logs, para gerar a carga de trabalho utilizada em seus testes.

6.3 Dificuldades encontradas

O trabalho relacionado a Web, por si só, traz algumas dificuldades, pois esta área encontra-se em evidência, o que faz com que surjam novos aspectos em intervalos pequenos de tempo. Pode-se citar, como exemplo, a caracterização de carga que está sempre apresentando novos parâmetros.

A escolha da ferramenta a ser utilizada também não foi trivial, pois estas se apresentaram em diversas alternativas, cada uma contendo ainda algumas variações. Além disso, o aprendizado de cada ferramenta teve o seu custo. Como mostrado no Capítulo 4, cada *benchmark* apresentou pontos positivos e negativos, podendo qualquer um deles ser escolhido. O que acabou definindo a decisão a favor do httperf foi a estrutura em que o

código fonte se apresentou. Nesta ferramenta o código aparece mais simples, comentado e simples de entender.

Outro ponto importante foi a obtenção dos logs utilizados, pois o objetivo era conseguir logs que representassem cargas distintas. O que causou dificuldades foi o fato de que empresas responsáveis por site comerciais não disponibilizam facilmente seus logs, alegando questões de segurança. Aqui a exceção foi o log da copa do mundo de 98. Nesse caso, já havia um estudo publicado disponibilizando o próprio log e informações a respeito dele.

O *benchmark* Surge representou um problema considerável, pois foi dispendido muito tempo na tentativa de sua utilização. O que se mostrou em vão pois, mesmo após todo o esforço, a ferramenta não funcionou a contento.

De forma mais específica, durante a fase de implementação apresentaram-se fatores que trouxeram dificuldades para o andamento do trabalho. Alguns desses fatores foram:

- Estudo minucioso do código fonte das ferramentas, visando selecionar uma delas para receber as implementações objetivadas. Esse estudo foi de extrema utilidade para a compreensão do funcionamento dos *benchmarks*, principalmente do httperf que foi o escolhido para tal objetivo.

- Implementação da estrutura utilizada para o armazenamento dos registros presentes no log. As implementações iniciais não obtiveram bom resultado pois consumiam muita memória. Assim, quando eram tratados logs com mais 100.000 registros, o programa causava o travamento da máquina. A implementação atual executou sem problemas com logs que continham mais de 5.000.000 de registros.

- Definição de um mecanismo de redivisão das faixas/classes do histograma que apresentavam peso muito grande. Foi utilizado, inicialmente, um número grande e fixo de faixas, por exemplo 100. Mas mesmo assim algumas dessas faixas apresentavam peso acima de 90%. Finalmente optou-se por criar um número de classes pequeno e redividir as que apresentavam peso maior do que o desejado.

6.4 Trabalhos futuros

Como mencionado diversas vezes no decorrer desta dissertação, a avaliação de servidores web é uma tarefa bastante complexa, se apresentando em constante evolução. Com isso, sempre há pontos a serem abordados por novos trabalhos desenvolvidos neste escopo.

Algumas sugestões para a realização de novos trabalhos são:

- Acrescentar novas funcionalidades ao httperf como:

  - Controlar testes com a utilização de diversas máquinas clientes, de forma centralizada.
  - Novos métodos para a geração de diferentes cargas.
  - Mecanismos para controlar uma seqüência de testes, tratar os resultados e disponibilizar um resultado geral - como é feito no WebStone.
  - Construção de conjuntos de dados para a execução de testes específicos.

- Estudo e comparação de novas versões do WebStone e httperf, ou ferramentas totalmente novas que venham a ser construídas.

- Modificar a ferramenta aqui construída para trabalhar com arquivos de log em formato binário, exigindo assim, menos espaço em disco.

- Acrescentar à ferramenta a capacidade de extrair mais informações do log: tempo entre requisições; requisições dinâmicas: entre outras.

- Obtenção de novos parâmetros de caracterização de carga para utilização nos benchmarks.

- Construção de mecanismos que possibilitem a avaliação de servidores web com diferenciação de serviços (Teixeira *et al.*, 2003).

- Implementação de um mecanismo de redivisão baseado em variância. O problema de se redividir uma classe/faixa é saber até quando repetir o processo, isto é, até quando continuar dividindo. Uma proposta interessante é: verificar a variância do tamanho dos objetos contidos na classe; se estiver acima do valor desejado, dividir a classe; repetir o processo para as novas classes criadas.

Referências Bibliográficas

Abdelzaher, T. F.; Bhatti, N. (1999). Web content adaptation to improve server overload behavior. *Computer Networks (Amsterdam, Netherlands: 1999)*, v.31, n.11–16, p.1563–1577.

Apache Software Foundation (2003). Apache HTTP Server Project. Disponível em `http://httpd.apache.org`.

Arlitt, M. (1996). A performance study of internet web servers.

Arlitt, M. F.; Williamson, C. L. (1996). Web server workload characterization: The search for invariants. *Proceedings of the ACM SIGMETRICS'96 Conference*, p. 126–137.

Banga, G.; Druschel, P. (1999). Measuring the capacity of a web server under realistic loads. *World Wide Web*, v.2, n.1-2, p.69–83.

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

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

Calzarossa, M.; Serazzi, G. (1993). Workload characterization: a survey.

CGI (2002). The common gateway interface. Disponível em `http://hoohoo.ncsa.uiuc.edu/cgi/`.

Collin, S. M. H. (1993). *MICHAELIS: Dicionário Prático de Informática*. Melhoramentos.

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

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

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

Fan, L.; Cao, P.; Almeida, J.; Broder, A. Z. (2000). Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, v.8, n.3, p.281–293.

FastCGI (2002). Fastcgi specification. Disponível em `http://www.fastcgi.com/`.

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

Fonseca, R.; Almeida, V.; Crovella, M. (2003). Locality in a web of streams.

Francès, C. R. L. (1998). Stochastic Feature Charts -- Uma extensão estocástica para os statecharts. Dissertação (Mestrado), USP/ICMC, São Carlos, SP.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, , n.8, p.231–74.

Harel, D.; Politi, M. (1998). *Modeling Reactive Systems with Statecharts*. MCGRAWHILL TRADE.

Hu, J.; Mungee, S.; Schmidt, D. (1998). Principles for developing and measuring high-performance web servers over atm.

Hu, J.; Pyarali, I.; Schmidt, D. (1997). Measuring the impact of event dispatching and concurrency models on web server performance over high-speed networks.

Ivan Herman (2003). World wide web consortium. Disponível em `http://www.w3.org`.

J. Hu (1998). The jaws adaptive web server. Disponível em `http://www.cs.wustl.edu/~jxh/research/`.

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

Krishnamurthy, B.; Rexford, J. (2001). *Redes para a Web*. Campus.

Maciel, P. R.; Lins, R. D.; Cunha, P. R. (1996). *Introdução às Redes de Petri e Aplicações*. Unicamp.

Maltzahn, C.; Richardson, K. J.; Grunwald, D. (1997). Performance issues of enterprise level web proxies. *Proceedings of the ACM SIGMETRICS'97 Conference*, p. 13 – 23.

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

Menascé, D. A.; Almeida, V. A. F.; Dowdy, L. W. (1994). *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*. Prentice Hall.

Mindcraft (2002). Webstone - the benchmark for web servers. Disponível em `http://www.mindcraft.com/webstone/`.

Mosberger, D.; Jin, T. (1998). httperf - a tool for measuring web server performance.

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

Orlandi, R. C. G. S. (1995). *Ferramentas para Análise de Desempenho de Sistemas Computacionais*. ICMSC/USP. Dissertação de mestrado.

Provos, N.; Lever, C.; Tweedie, S. (2000). Analyzing the overload behavior of a simple web server. p. 1–12.

Red Hat (2003). Red hat linux. Disponível em `http://www.redhat.com`.

Scheuermann, P.; Shim, J.; Vingralek, R. (1997). A case for delay-conscious caching of Web documents. *Computer Networks and ISDN Systems*, v.29, n.8–13, p.997–1005.

Silva, D.: Kulesza, U. (2000). Reengineering of the jaws web server design using aspecto-riented programming.

Soares, L. F. G. (1992). *Modelagem e Simulação Discreta de Sistemas*. Campus Ltda.

SPEC (1999). Specweb99 benchmark. Disponível em `http://www.specbench.org/osg/web99/`.

Stading, T.; Maniatis, P.; Baker, M. (2002). Peer-to-peer caching schemes to address flash crowds. *1st International Peer To Peer Systems Workshop (IPTPS 2002)*, Cambridge, MA, USA.

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

Tanenbaum, A. S. (2002). *Computer Networks*. Prentice-Hall, 4. edição.

Teixeira, M. M.; Santana, M. J.; Santana, R. H. C. (2003). Avaliação de algoritmos de escalonamento de tarefas em servidores web distribuídos. *XXX Seminário Integrado de Hardware e Software (SEMISH)*. XXXIII Congresso da SBC, Campinas, SP.

Tittel, E. (1996). Benchmarking the web. Disponível em `http://sunsite.uakom.sk/sunworldonline/swol-09-1996/swol-09-webbench.ht%ml`.

Trent, G.: Sake, M. (1995). Webstone:the first generation in http server benchmarking. *MTS Silicon Graphics*.

Vallamsetty, U. (2003). Characterization of e-commerce traffic.

W3C (1999). HTML 4.01 Specification. Disponível em `http://www.w3.org/TR/html4`.