FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

SÉRGIO CESAR MOTTI

AVALIAÇÃO DE DESEMPENHO EM SERVIDORES WEB

MARÍLIA

AGOSTO 2006

SERGIO CESAR MOTTI

AVALIAÇÃO DE DESEMPENHO DE SERVIDORES WEB TRANSACIONAIS

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do título de Mestre em Ciência da Computação. (Área de Concentração: Arquitetura de Computadores).

Orientador:

Prof. Dr. Marcos Luiz Mucheroni

MARÍLIA

AGOSTO - 2006

SÉRGIO CESAR MOTTI

AVALIAÇÃO DE DESEMPENHO DE SERVIDORES WEB TRANSACIONAIS

Banca examinadora da qualificação apresentada ao Programa de Mestrado da UNIVEM, /F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação. Área de Concentração: Arquitetura de Computadores.

ORIENTADOR: _____

Prof. Dr. Marcos Luiz Mucheroni

1º EXAMINADOR: _____

Prof. Dr. Jorge Luiz e Silva

2º EXAMINADOR: _____

Prof. Dra. Kalinka Regina L. Jaquie Castelo Branco

Marília, _____de _____de 2006

AGRADECIMENTOS

A todos que direta e indiretamente contribuíram para a produção deste trabalho. A minha família pela dedicação e compreensão nesta etapa. Ao Prof. Dr. Marcos Luiz Mucheroni pela paciência, dedicação e incentivo demonstrado a todo o momento.

MOTTI, Sergio César. **Avaliação de Desempenho de Servidores Web Transacionais**. Dissertação de Mestrado, na area de Arquitetura de Sistemas Computacionais. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Um dos maiores desafios para o uso amplo da Internet é a escalabilidade dos servidores, ou seja, a sua capacidade em suportar uma demanda crescente sem que a qualidade dos serviços providos seja afetada. Escalabilidade em servidores transacionais, que compreendem servidores de comércio eletrônico e governo eletrônico, é ainda mais difícil, uma vez que é necessário manter dados das transações de um mesmo usuário durante a sua interação. Uma estratégia comum para conseguir escalabilidade é utilizar servidores agrupados, onde a carga é distribuída entre os servidores. Entretanto, como conseqüência das características da carga de trabalho e da necessidade da manutenção de dados coerentes entre os servidores agrupados, pode surgir desbalanceamento de carga entre os servidores, reduzindo a sua eficiência. A proposta deste trabalho é a avaliação de uma estratégia de escalabilidade com o objetivo de expandir e distribuir a capacidade de processamento de Servidores Web Transacionais, usando como métrica testes de benchmark.

Palavras-chave: Servidor Web, Escalabilidade, Servidores Agrupados, Benchmark.

MOTTI, Sergio César. **Avaliação de Desempenho de Servidores Web Transacionais**. Dissertação de Tese de Mestrado, na area de Arquitetura de Sistemas Computacionais. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

One of the main challenges to the wide use of the Internet is the scalability of the servers, that is, their ability to handle the increasing demand without affecting the quality of the services provided. Scalability in stateful servers, which comprise e-Commerce and other transaction-oriented servers, is even more difficult, since it is necessary to keep transaction data across requests from the same user. One common strategy for achieving scalability is to employ clustered servers, where the load is distributed among the several servers. However, as a consequence of the workload characteristics and the need of maintaining data coherent among the servers that compose the cluster, load imbalance arise among servers, reducing the efficiency of the server as a whole. In this work propose the evaluation of an scalability strategy aiming to expand and to distribute the capacity of processing of Clustered Web Servers, using as benchmark test metric.

Keywords: Web Server, Scalability, Web Cluster, Benchmark

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização e Motivação

O crescimento recente da Internet está relacionado à expansão da *World Wide Web (* WWW). Uma parte significativa deste crescimento pode ser creditado ao aumento do número de serviços e usuários de comércio eletrônico e outras aplicações transacionais que se utilizam da WWW. Uma medida freqüente de sucesso de aplicações WWW e aplicações de comércio eletrônico em particular é a capacidade dos Servidores Web de responder requisições prontamente. No caso de servidores de comércio eletrônico, o sucesso do serviço é conseqüência da capacidade do servidor de obter a atenção de um grande número de usuários e mantê-los satisfeitos com a qualidade do serviço provido. Por outro lado, um grande número de usuários significa sobrecarga nos servidores responsáveis pelos serviços, a qual não deve afetar a expectativa dos clientes.

Em conseqüência, escalabilidade se tornou uma questão fundamental dos servidores de comércio eletrônico. Mas, aumentar a capacidade de um servidor nem sempre é possível, pois há limites para a velocidade do processador e da memória, entre outros fatores (TAVARES, 2003). Uma estratégia comum nesses casos é o uso de servidores agrupados, onde as capacidades de máquinas individuais não afetam diretamente a qualidade do serviço provido. Essa estratégia tem sido aplicada com sucesso em servidores de conteúdo estático, motivando o desenvolvimento de novas técnicas de distribuição e identificando gargalos nesses sistemas.

Por outro lado, grande parte dos trabalhos na área de análise de desempenho de servidores de comércio eletrônico tem sido feita no contexto de servidores individuais (TAVARES, 2003). A distribuição de serviços traz uma nova dimensão para o problema, que não tem sido muito discutida na literatura.

1.2 Objetivos

O trabalho tem como objetivos principais; avaliar e determinar a capacidade de arquiteturas para servidores e serviços Web; identificar o comportamento da infra-estrutura (computadores, sistema operacional, aplicações, conectividade de rede, protocolos de rede) sob condições de carga de trabalho; identificar o comportamento da rede durante o balanceamento de carga.

1.3 Descrição do Trabalho

O capitulo 2 apresenta o levantamento bibliográfico sobre modelos de arquiteturas para Servidores *Web*, como escalabilidade, distribuição de processamento, confiabilidade, disponibilidade. Uma análise de trabalhos sobre balanceamento de carga e Servidores *Web* baseados em *Clusters*.

O capitulo 3 apresenta um modelo de arquitetura de Servidores Web, baseados em Cluster Beowulf, com Sistema Operacional FreeBSD, Servidor de HTTP Apache, sistema de análise estatística do tráfego de rede com o *software ntop*, um gerador e interpretador de carga de trabalho Siege-2.65, registrando o desempenho dos Servidores Web, onde foram avaliados vários aspectos de comportamento da arquitetura do sistema, como capacidade de transações, maior e menor tempo de resposta à requisições, transações por minuto, taxa de transferência de dados, transações concorrentes, vazão (*troughtput*) em *Megabytes* por segundo.

O capitulo 4 descreve a implementação da arquitetura para avaliação, o *hardware*, a infra-estrutura utilizada, os procedimentos utilizados para compilação do *Kernel*, o algoritmo de balanceamento de carga, metodologia para o teste de desempenho e os resultados das simulações de carga de trabalho realizadas com o *benchmark*.

O capitulo 5 apresenta as conclusões e o resultado geral do trabalho.

2. MODELOS DE ARQUITETURAS PARA SERVIDORES WEB

As relações entre as pessoas estão em grande parte sendo estabelecidas em *padrões virtuais*. Neste aspecto, a Internet é significativamente responsável pela consolidação dos computadores como meios de *conectar* as pessoas entre si, seja qual for a distância que as separe. Se por um lado a *conectividade* traz inúmeros benefícios, reduzindo custos e agregando agilidade aos mais variados tipos de negócios, também traz consigo problemas adicionais que desafiam a capacidade técnica e científica de pesquisadores e profissionais da área. Um destes problemas seguramente é atender ao número crescente de usuários da grande rede, fazendo com que principalmente os servidores *web*, sediados em empresas ou instituições de um modo geral, possam expandir a sua capacidade. A possibilidade de expandir a capacidade de atendimento às demandas existentes da Internet e estar preparado para as demandas futuras, seja pelo acréscimo de novos servidores ou processadores, ou ainda pelo incremento de aplicações, protocolos de comunicação e algoritmos, denomina-se *escalabilidade* (BAKER, 1998).

Um *site web* típico compõe-se de uma estrutura de comunicações e várias classes de servidores, como servidores HTTP, servidores FTP, servidores de cache, e servidores de email.

A arquitetura para atender as transações entre clientes e servidores, pode ser vista a partir de uma estrutura de serviço como da figura 1:

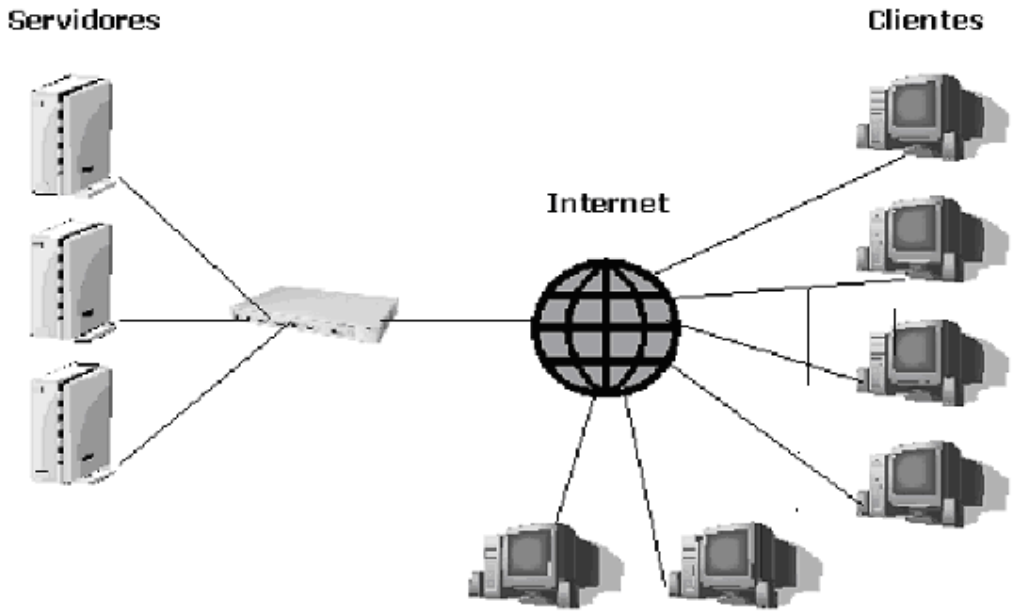

Figura 1: Estrutura de um site web

As características de um *site web* pautam-se pelas regras da arquitetura *cliente/servidor*, onde o servidor ou servidores recebem e atendem as requisições dos clientes dispersos pela Internet. Como a quantidade de clientes tende a ser muito maior do que a quantidade de servidores de um *site web* em particular, fica estabelecido aí o ponto de partida para análise das questões envolvidas em escalabilidade.

2.1 Ampliação de Capacidade e Escalabilidade

Escalabilidade pode ser definida como a capacidade de se fazerem acréscimos nas quantidades de determinadas grandezas, sem necessariamente haver um limite pré-estabelecido (MILLER, 1981). O termo é utilizado nas mais diversas áreas do conhecimento, passando pela economia, física e estatística. Na tecnologia da informação, designa acréscimo de equipamentos em sistemas potencialmente capazes de um aumento gradual do seu poder de atender à demanda, sem prejuízo que invalide o ganho de desempenho.

A ampliação da capacidade de servidores pode ser feita em inúmeras alternativas, porém estará restrita a dois sentidos de escala: horizontal e vertical.

A expansão vertical é uma forma simples de ampliar a capacidade de servidores, mediante o acréscimo de memória, *upgrade* de processador ou instalação de um disco com maior capacidade, ou mesmo substituição do equipamento por outro mais poderoso. Talvez nem mesmo seja apropriado denominar a expansão vertical de escalabilidade, pois existe um limite para o *upgrade*, a partir do qual a solução deve ser descartada e adotada uma nova, substituindo-se totalmente o equipamento ou partindo para a expansão horizontal.

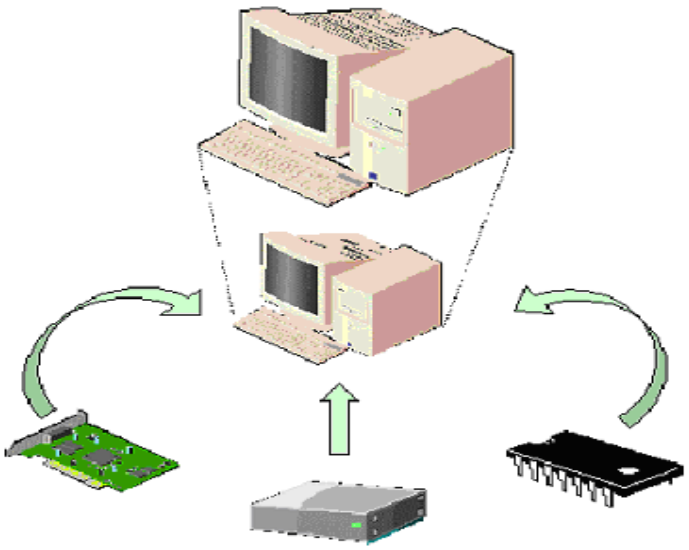

Figura 2: Expansão vertical / mesmo equipamento

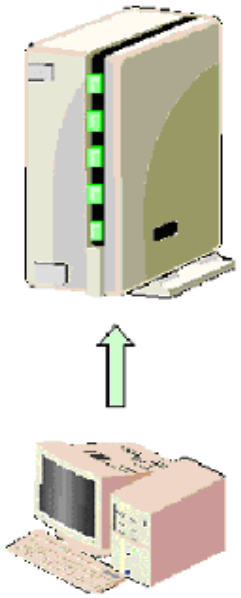

Figura 3: Expansão vertical / servidor mais poderoso

A agregação de maior poder de processamento de forma horizontal envolve dividir tarefas. Para conseguir este objetivo, os recursos utilizados são o processamento distribuído e paralelo, o que pode ser obtido de duas formas:

- Adição de um ou mais servidores, conectando-os em rede;
- Adição de mais processadores (desde que o projeto suporte esta expansão).

Sendo adicionados novos servidores, é importante que se dê a devida atenção aos algoritmos e protocolos para um efetivo gerenciamento do processamento que se pretende distribuir.

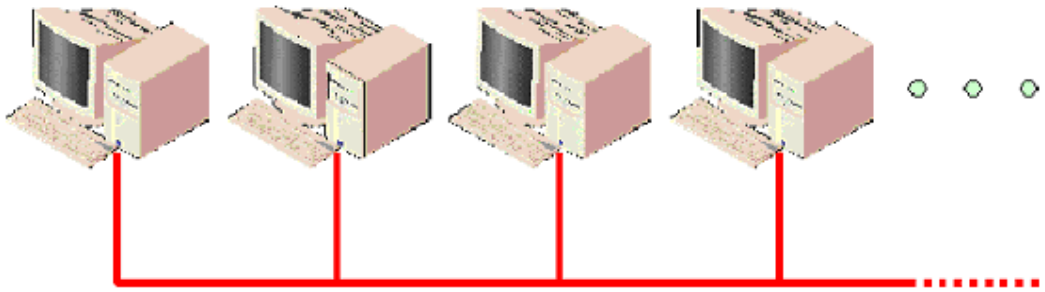

Figura 4: Expansão horizontal ; escalabilidade de servidores

2.1.1 A importância de servidores escaláveis

Servidores que possam ter sua capacidade ampliada, ou mesmo conectarem-se em rede com outros para distribuir a carga de processamento, atendem diretamente os objetivos de negócios e serviços na Web. Muitas vezes, ao ser formulada uma estratégia de negócios na Web, que implique na necessidade da montagem e operação de um *site web*, é possível que por questões de disponibilidade se prefira iniciar com uma estrutura básica, ajustada a um momento inicial de operações, e depois ir gradativamente expandindo a capacidade na medida das exigências futuras.

2.1.2 Problemas envolvidos

O problema básico da implementação de sistemas ou servidores escaláveis é permitir o aumento da capacidade de atendimento à demanda, sem reflexos negativos no desempenho, e mantendo a consistência no processamento. A confiabilidade de sistemas deste tipo deve ser mantida, mesmo perante cargas elevadas de requisições ou picos momentâneos de exigência de processamento(OLSTON et al., 2005).

Com relação aos picos, também descritos como rajadas (MENASCÉ, 2003), de utilização dos serviços *Web*, muitas vezes é preferível negar o serviço do que procurar

oferecê-lo correndo o risco de um colapso total, que comprometa o atendimento de todos os clientes envolvidos.

2.1.3 Funcionamento de um servidor *web*

O entendimento do problema da escalabilidade de servidores *web* passa pela compreensão do seu funcionamento. Em resumo, um servidor *web* dedica o seu tempo em duas tarefas principais: primeiramente o *Atendimento a conexões* e em seguida o *Processamento de requisições*.

2.1.3.1 Atendimento às conexões dos clientes

Analisando do ponto de vista da máquina cliente, um *browser* necessita fazer uma série de conexões ao servidor *web* para montar localmente uma página. Deve ser feita uma conexão para cada elemento da tela, como texto, figuras ou elementos de multimídia.

A seqüência de estabelecimento de uma conexão TCP/IP entre um servidor e um cliente ocorre da seguinte maneira (BANGA, 1997):

1 - O servidor fica "ouvindo" (*listening*) por possíveis conexões HTTP, por meio de um *socket* na sua porta lógica 80.

2 - O cliente envia uma requisição de conexão, iniciando o mecanismo de *three way handshake* por meio do envio de um pacote TCP SYN ao servidor.

3 - O servidor responde à requisição do cliente devolvendo um pacote TCP SYNACK, cria um novo *socket* para a conexão incompleta e o insere numa fila SYNRCVD (*syn´s* recebidos), onde fica no aguardo da confirmação.

4 – Ocorrendo a confirmação, o cliente envia a confirmação por meio de um pacote TCP ACK.

5 - Ao receber o pacote TCP ACK do cliente, o servidor remove o novo *socket* da fila SYN-RCVD e o insere na ACCEPT QUEUE (fila de *sockets* aceitos).

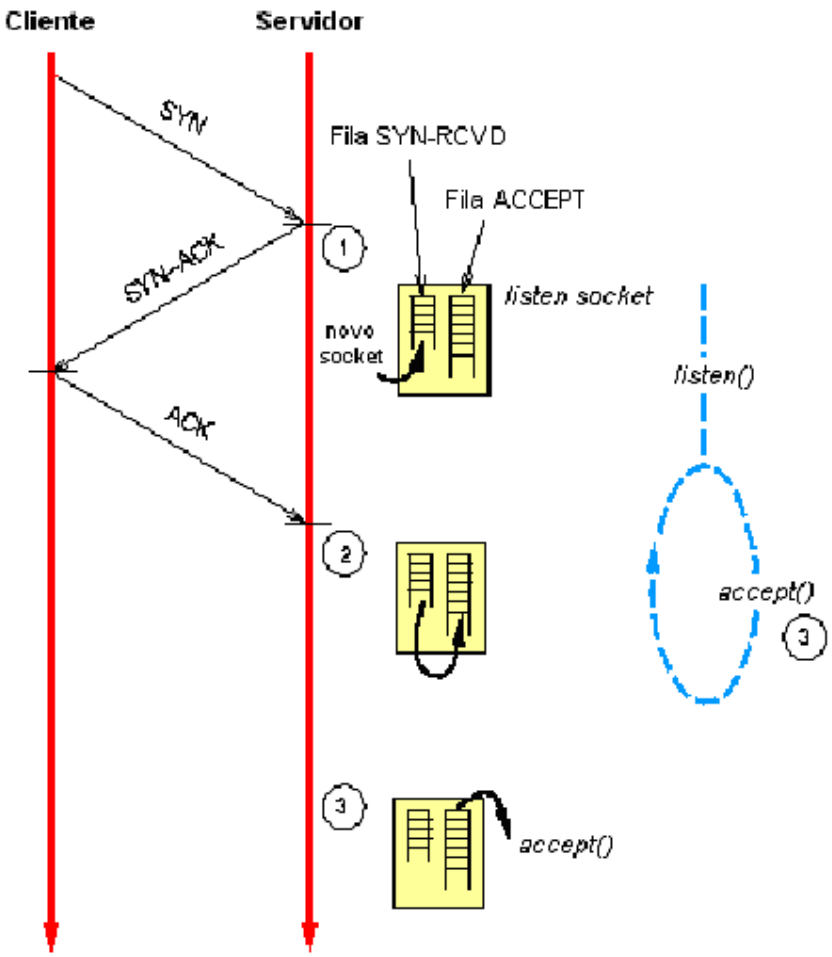

Figura 5: Atendimento a conexões em um servidor *web*

2.1.3.2 Processamento das requisições dos clientes

Na medida em que as conexões dos clientes vão sendo estabelecidas, outro processo no servidor, assíncrono, fica atendendo às requisições que chegam, da seguinte forma (BANGA, 1997):

1 - O servidor passa a conexão para um processo auxiliar.

2 - O processo auxiliar lê a requisição HTTP recebida do cliente na conexão.

3 - A requisição é atendida (envio da página inicial HTML da página, execução de uma *query* em um banco de dados ou algum outro procedimento).

4 - O processo encapsula a resposta em um pacote TCP e envia para o cliente.

5 - O processo finaliza (fecha) a conexão, liberando o *socket* (BANGA, 1997).

A Figura 6 (MOGUL, 1995) apresenta a seqüência de atendimento de requisições HTTP a partir do *browser* de um cliente:

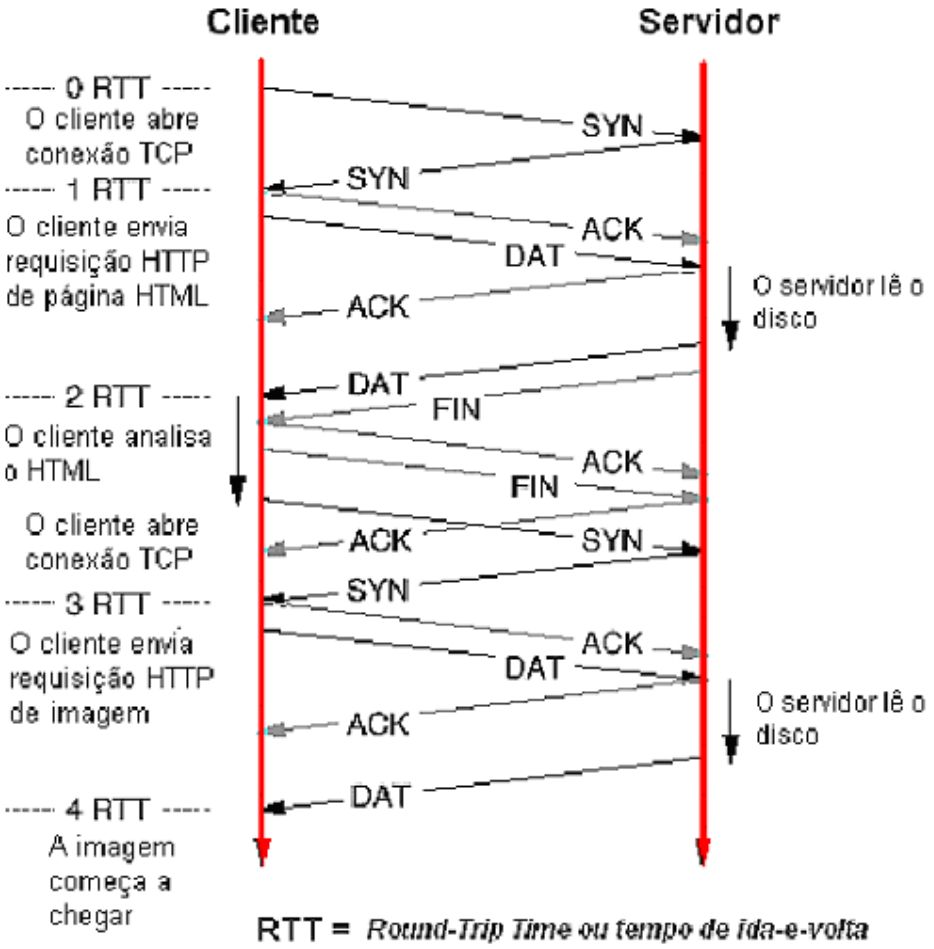

Figura 6: Seqüência completa de requisições HTTP

Se o servidor HTTP que recebe as conexões e processa as requisições dispor de uma lógica adequada, poderá redirecionar as requisições dos clientes já no processamento destas,

encapsulando-as em pacotes e transmitindo-os aos servidores que estiverem ociosos ou que contarem com as aplicações requeridas.

A inteligência para distribuição de processamento poderá ser agregada de diversas formas, desde simples *scripts* CGI até sistemas operacionais de rede, como é o caso do UNIX (e suas variações) e o Windows NT (MENASCÉ, 2003).

2.2. Benchmarks e Testes de Desempenho

O *benchmarking* é o principal método para medir o desempenho de uma arquitetura física real (KRISHNASWAMY, 2000). O *benchmarking* refere-se à execução de um conjunto de programas representativos em diferentes computadores e redes, medindo seus resultados. Os resultados do *benchmark* são usados para avaliar o desempenho de determinado sistema ao executar uma carga de trabalho bem definida. Grande parte da popularidade dos *benchmarks* padrão vem do fato de que eles possuem objetivos de desempenho e cargas de trabalho que podem ser medidos e repetidos. Normalmente, estudos de aquisição de sistema de computador e analises comparativas dos produtos se baseiam em *benchmarks*. Eles também são como ferramentas de monitoração e diagnóstico. Os fornecedores, desenvolvedores e usuários executam *benchmarks* para localizar problemas de desempenho em novos sistemas.

A técnica mais utilizada para obter resultados de desempenho de determinado serviço na Web é chamada de “teste de desempenho”, que significa executar testes para determinar o desempenho do serviço sob condições específicas da aplicação e da carga de trabalho. No ambiente Web, as empresas não podem se dar ao luxo de instalar serviços e aplicações antes de terem certeza de que eles funcionam realmente bem. É crucial prever como um serviço específico na Web responderá a uma carga de trabalho específica. Os benchmarks não são capazes de oferecer respostas exatas para perguntas sobre cenários específicos de carga de trabalho e aplicações. Portanto, é preciso procurar testes de desempenho específicos. Com base em determinada carga de trabalho e cenário de aplicação,

o teste de desempenho deve oferecer informações sobre como o serviço responderá a cargas realísticas antes que entre em atividade plena.

2.2.1 A Natureza dos Benchmarks

Tempo e velocidade são as medidas básicas de desempenho de um sistema de computação. Do ponto de vista do usuário, o tempo de execução do programa ou aplicação é o melhor indicador do desempenho do sistema. Os usuários não querem saber se o serviço é executado perto do seu microcomputador, na rede local ou se é processado a milhares de quilômetros de distância, conectado por meio de várias redes. Os usuários sempre desejam um tempo de resposta rápido. Do ponto de vista da gerência, o desempenho de um sistema é definido por uma métrica capaz de indicar quanto trabalho o sistema pode realizar. Por exemplo, os gerentes de sistema estão interessados em perguntas como: quantas transações o sistema pode executar por minuto ou quantas requisições o site web é capaz de atender por segundo? Além disso, tanto usuários quanto gerentes estão sempre preocupados com custo, refletido em perguntas como: qual é o custo do software operacional? Qual é o custo de compra do servidor? Como conseqüência de todos esses pontos de vista diferentes, o problema básico continua: qual é uma boa medição padrão do desempenho do sistema?

Os fabricantes de computador se referem à velocidade de um processador por seu tempo de ciclo, que pode ser expresso por sua extensão (por exemplo, um nanossegundo) ou por sua freqüência (por exemplo, 1 GHz). Contudo, um erro comum é o uso da velocidade do *clock* (seja em gigahertz ou em nanossegundos) para comparar o desempenho do processador. O desempenho geral de um processador é afetado diretamente por outras características da arquitetura, como caching, pipelining, unidades funcionais e tecnologia de compilador (HENNESSY, 1996). No passado, um modo popular de avaliar o desempenho do processador era usar a métrica MIPS (milhões de instruções por segundo). Embora o MIPS tenha sido

bastante empregado como base para comparações de processadores, é importante lembrar ao leitor sobre os problemas com seu uso. O MIPS depende do conjunto de instruções, o que torna difícil comparar computadores com diferentes conjuntos de instruções. Por exemplo, o uso do MIPS para comparar o desempenho de uma máquina RISC (*Reduced Instruction Set Computer*) com o de uma máquina CISC (*Complex Instruction Set Computer*) tem pouco significado. Como o MIPS não é definido em um domínio de qualquer aplicação específica, seu uso pode ser confuso. Diferentes programas, rodando no mesmo computador, podem chegar a diferentes níveis de MIPS (HENNESSY, 1996).

O desempenho do sistema é complexo; é o resultado da interação de muitos componentes de *hardware e software*. Complicando o problema está o fato de que cada arquitetura de serviço na Web é exclusiva em sua configuração, aplicações, sistemas operacionais e carga de trabalho. Além do mais, serviços na Web exibem uma grande variação no desempenho quando executam diferentes cargas de trabalho. Nenhum número isolado pode representar o desempenho de um sistema Web em todas as aplicações. Buscando encontrar uma boa medida de desempenho, programas padrão, conhecidos como *benchmarks*, têm sido utilizados para avaliar o desempenho dos sistemas.

2.2.2 Benchmarks de Servidores Web

Um conjunto de testes e cargas de trabalho é especificado para a medição de servidores Web. Os *benchmarks* de servidor Web mais usados são detalhados a seguir. Esta seção descreve três benchmarks de servidor Web: Webstone (MINDCRAFT, 2005), SPEC-web (SPEC, 2005) e o TPC-W (TPC, 2005). Esses programas simulam clientes Web. Eles geram requisições a um servidor, de acordo com algumas características de carga de trabalho especificadas, recebem as respostas retomadas pelo servidor e coletam as medições. É importante observar que os benchmarks de servidores Web normalmente são executados em

LANs pequenas e isoladas, quase sem erros de transmissão. Ao contrário, serviços na Web, oferecidos no mundo real, são acessados por meio da Internet ou de grandes intranets, que envolvem conexões WAN, gateways, roteadores, pontes e hubs que tornam o ambiente de rede ruidoso e passível de erros. Além do mais, as latências são muito mais altas nos ambientes do mundo real do que nas LANs usadas nos testes de *benchmarking*. Assim, os resultados da análise do *benchmark* da Web devem levar em consideração essa observação.

O *Transaction Processing Performance Council* (TPC) é uma organização sem fins lucrativos, criado com o objetivo de estabelecer critérios de performance de processamento de transações e de bancos de dados por meio de benchmarks, ou testes para estabelecer padrões de referência, (tais como o TPC-C, o TPC-W e o TPC-H), e divulgar os dados reais dessa performance a partir desses testes. Os *benchmarks* TPC são submetidos a exigências extremamente rigorosas, principalmente nos quesitos confiabilidade e durabilidade, e são sempre aplicados perante uma auditoria independente. Os membros desse conselho incluem as principais empresas de bancos de dados e fornecedores de sistemas de hardware do mercado.

As empresas participam dos *benchmarks* TCP para demonstrar, objetivamente, a performance de seus sistemas em um ambiente controlado, e para aplicar as tecnologias utilizadas durante o processo de testes à criação de produtos de hardware e de software ainda mais robustos e escaláveis.

O *benchmark* TPC-W simula as atividades de um servidor da Web dedicado a transações de negócios. A carga de trabalho é executada em um ambiente de comércio da Internet, devidamente controlado, e avalia uma série de componentes do sistema que estão associados a este ambiente.

A métrica de performance reportada pelo TPC-W é o número de interações da Web processadas por segundo. São utilizados vários tipos de interações da Web para simular a

atividade. Um exemplo é a atividade de uma loja de varejo na qual cada interação está sujeita a uma resposta dentro de um determinado limite de tempo. O porte da loja é selecionado a partir de uma escala de valores pré-definida, e inclui um certo número de itens no estoque. Esse número de itens pode variar de 1.000 a 10.000.000. A métrica de performance deste benchmark é expresso em Interações da Web por Segundo (WIPS - *Web interactions per second).*

Um dos pilares da respeitabilidade da TPC diz respeito a como ela é estruturada. As pessoas que formam a TPC são na verdade funcionários de várias empresas membros do TPC. Cada empresa pode ter um Representante Primário e vários Representantes Secundários junto à TPC. Muitas vezes a pessoa concilia suas atividades normais na empresa com as atividades da TPC, mas há casos de funcionários 100% dedicados às atividades relacionadas com a TPC.

2.3 Distribuição de processamento

A distribuição de processamento é o ponto crucial de *sites* de Internet para o atendimento às demandas geradas pelos clientes. Partindo do típico modelo cliente/servidor, é possível imaginar uma quantidade variável de clientes que em determinadas situações poderá chegar a números insuportáveis para o servidor ou servidores, passando a ser da maior importância o dimensionamento e a distribuição do poder de processamento.

Na realidade, o processamento distribuído torna-se vantajoso em função do compartilhamento de recursos, incremento da velocidade de processamento, além da própria escalabilidade (OLSTON, et al, 2005).

2.3.1 Onde começa a distribuição de processamento

A preocupação em dividir os esforços de processamento não se limita apenas aos processadores disponíveis para executar as tarefas. Ela começa antes, mais precisamente no

link de chegada a partir da Internet, de onde as requisições devem ser direcionadas para aqueles equipamentos que apresentem a maior disponibilidade de capacidade no momento.

O processamento poderá já de antemão ser direcionado para o servidor-alvo, a partir de um servidor destinado especificamente para a distribuição dos processos.

O procedimento de distribuir o processamento, conforme a capacidade disponível, é denominado de *balanceamento de carga.*

2.3.2 Balanceamento de carga

Para servidores multiprocessados o balanceamento de carga é inerente, já que tais equipamentos costumam ser dotados de um sistema operacional que proporciona transparência na distribuição de processamento. O enfoque e o cuidado na distribuição de processamento, e conseqüentemente, o balanceamento de carga, aparece quando se lida com servidores monoprocessados, cuja escalabilidade é predominantemente horizontal, mediante a agregação de novos servidores aos já existentes, ligados entre si por meio de conexões de rede.

2.3.3 Fatores críticos para a distribuição de processamento

Vários fatores podem determinar o bom ou mau funcionamento de um grupo de servidores cooperativos. Frente à necessidade *incremental* de um *site web* escalável, são analisados os fatores mais críticos que devem ser levados em conta para prover o crescimento desejado, buscar manter ou melhorar o desempenho. Assim, a análise está concentrada nos fatores que exercem influência sobre a busca de escalabilidade mediante a constituição de um *pool* de servidores, principalmente para evitar gargalos no processamento (ALLAIRE, 2000).

O primeiro problema descrito trata da programação ineficiente, ou *Lógica pobre de programação* é a razão mais comum de desempenho crítico de aplicações, e as aplicações *web* não fogem à regra. Se as aplicações forem implementadas com as melhores práticas da

indústria da engenharia de *software*, como padrões de codificação, e revisões de especificações, este problema poderá ser minimizado.

Mesmo em uma aplicação bem projetada e bem programada, a *capacidade do processador* poderá ter um desempenho crítico se a CPU do servidor não tiver potência suficiente. A maior carga de processamento e as aplicações de missão crítica devem residir em *hardware* compatível.

O aumento crescente da demanda de requisições pode ocasionar em determinadas situações o *Congestionamento de servidores*. O congestionamento refere-se não apenas aos servidores *web*, mas a todos os tipos de servidores. Os servidores de aplicação, *proxies*, de pesquisa e indexação, e servidores de *backoffice* poderão eventualmente ser submetidos a altos volumes que indiretamente afetam o desempenho do servidor *web*. Para afastar este problema, deve ser cuidadosamente planejada a topologia da rede, garantindo que os servidores sejam superiores à demanda prevista. Esta é uma das diretivas fundamentais da escalabilidade. Neste ponto é que deve ser avaliada a instalação de novos servidores destinados a repartir o processamento com os já existentes, afastando ou diminuindo a possibilidade de congestionamento.

Muitas aplicações dinâmicas que devem restringir acesso anônimo, em função de sediarem ou compartilharem informações confidenciais, devem estar resguardadas por um sistema de segurança *firewall* corporativo, podendo causar atrasos em requisições e respostas. Deve ser assegurado que as portas corretas estejam abertas no sistema de segurança *firewall*, para oferecer autenticação e validação seguras, e permitir que as comunicações apropriadas entre clientes e servidores sejam estabelecidas. Poderá ser necessário habilitar portas seguras adicionais para acomodar incrementos de tráfego.

Deve ser tratada as *Conexões de rede e largura de banda*, levando-se em consideração o tipo de rede no qual a aplicação irá funcionar (LAN, WAN ou Internet), e a

quantidade de tráfego que irá receber. Se o tráfego for demasiadamente pesado, devem ser instalados nós adicionais, novos roteadores, *switches* ou *hubs*.

Em Servidores Web Transacionais, um dos fatores críticos, tanto quanto a escalabilidade como a distribuição do processamento, refere-se ao gerenciamento das *Bases de dados***.** Se o acesso à base de dados for importante para a aplicação *web* e o seu conjunto de características, poderá ser também custoso em termos de desempenho ou escalabilidade se não houver um projeto eficiente (MENASCÉ, 2003)

A distribuição de processamento considerada dentro de um ambiente de rede, entre servidores que cooperem mutuamente, está intimamente ligada à utilização de algoritmos distribuídos. Estes algoritmos cuidam da manipulação de um conjunto de processos executando em diferentes máquinas e interagindo por meio da troca de mensagens, porém sem haver compartilhamento de memória física. São os algoritmos distribuídos que, bem implementados, trazem vantagens na relação entre custo e desempenho, além de prover escalabilidade e pôr decorrência, balanceamento de carga e tolerância a falhas.

2.3.4 Atendimento de conexões

Apesar do sucesso da Internet, segundo MOGUL, (1995), devido a sua simplicidade e facilidade de implementação de aplicações por meio do protocolo HTTP, este ainda faz uma utilização ineficiente dos recursos de rede e dos servidores, adicionando latência desnecessária pela criação de uma nova conexão TCP para cada requisição. Isto se deve ao fato de as conexões serem *não-persistentes*, ou seja, não ficam mantidas até o completo atendimento de uma requisição que deva retornar ao *browser* um número determinado de elementos de uma página *web*. O que contribui para a ineficiência são os chamados RTT ou *Round-Trip Times*: tempos de ida-e-volta das requisições do cliente até o recebimento das respostas. Uma das idéias de MOGUL (1995) é justamente trabalhar com conexões *persistentes*, visando reduzir a quantidade de períodos de RTT nas requisições dos clientes

aos servidores, o que consequentemente reduziria a carga da rede e a utilização de recursos dos servidores, bem como melhoraria a performance percebida pelos usuários.

O atendimento das conexões de rede e seu conseqüente encaminhamento para que as requisições sejam atendidas sempre terá como limite a capacidade dos servidores. Com efeito, um dos desafios mais críticos de servidores *web* é lidar com picos de carga. Sempre que for atingida a plena capacidade de atendimento de conexões, obviamente o servidor iniciará um processo de recusa de novas conexões até que seja restabelecida a capacidade de atendimento.

Em BANGA, (1997), são analisados alguns problemas que podem ocorrer no tratamento de conexões por parte de servidores *web* UNIX. Nestes sistemas, a variável de *kernel* denominada *somaxxcon* determina o tamanho máximo do *backlog* em um *socket* de conexão - aquele que fica "ouvindo" as conexões de chegada. O *backlog* é limitado pela soma dos tamanhos das filas SYN-RCVD e ACCEPT. Quando a soma de pacotes SYN exceder 1,5 vezes o tamanho do *backlog*, o servidor TCP passa a recusar estes pacotes.

Do lado do cliente TCP, quando este não recebe o pacote SYN-ACK do servidor, inicia um processo de retransmissão de pacotes SYN até que receba a resposta SYN-ACK ou até que o tempo de estabelecimento de conexão expire.

O tamanho médio da fila SYN-RCVD depende do tempo de ida-e-volta de uma mensagem (*RTT - round-trip time*) entre o servidor e seus clientes, e da taxa de requisição de conexões. Isto acontece devido ao fato de que um *socket* permanece nesta fila por um período de tempo igual ao RTT. Tempos de ida-e-volta muito longos e altas taxas de requisição contribuem para o aumento desta fila.

O tamanho da fila ACCEPT, por sua vez, depende da rapidez com que o servidor http processa as chamadas de *accept()* - ou taxa com que disponibiliza conexões, e da taxa de requisições. Se um servidor estiver operando no máximo da sua capacidade, este não poderá

executar chamadas *accept()* na velocidade suficiente para manter a taxa de requisição de conexões, fazendo com que a fila cresça.

O estado de cada *socket* é mantido em uma estrutura de dados denominada de *PCB - Protocol Control Block*, e o protocolo TCP mantém uma tabela com todas os PCB´s ativas o sistema. As PCB´s são criadas juntamente com os *sockets*, como resultado de alguma chamada de sistema ou do estabelecimento de alguma conexão. A existência de um PCB cessa no exato instante em que é executada uma chamada *close()* ou no momento em que chegar um pacote de controle FIN. Quando um pacote FIN é enviado, antes que retorne um pacote FIN-ACK e seja confirmado, o PCB é mantido por um intervalo igual ao tempo de *time-wait* da implementação. O propósito do *time-wait* é permitir retransmissões de processos ACK de finalização aos correspondentes FIN quando o ACK original é perdido, e também para permitir detecção de segmentos TCP duplicados ou atrasados.

Um dos problemas mais conhecidos das implementações tradicionais do protocolo TCP/IP que limita a vazão dos servidores *web*, são os valores máximos atribuídos à variável *somaxconn*. Este limite configurado com valores subestimados, pode ser atingido rapidamente, fazendo com que as filas ACCEPT e SYN-RCVD lotem, provocando a recusa de conexões sem necessidade. Se o limite de *somaxconn* for muito baixo, uma conexão poderá ser recusada mesmo se o servidor possuir suficientes recursos para atender à requisição. Mesmo no caso de uma longa fila de ACCEPT, é preferível aceitar a conexão, a menos que a fila já contenha carga suficiente para manter o servidor ocupado no mínimo pelo intervalo inicial de retransmissão (cerca de 6 segundos, considerando o Unix FreeBSD 4.4). Muitos fornecedores de sistemas Unix aumentaram o tamanho da variável *somaxxcon*. No Digital Unix foi fixado um valor de 32.767, enquanto que o Solaris o valor do *somaxxcon* apresenta valor de 1.000 (BANGA, 1997).

2.3.5 Distribuição de requisições

Se considerarmos um *pool* de servidores HTTP, o objetivo a ser alcançado será sempre a manutenção do desempenho no atendimento de requisições. Como decorrência surge a necessidade de se promover o *balanceamento de carga* entre os servidores, distribuindo-se as requisições no todo ou por partes, entre estes servidores.

As proposições existentes para distribuir o processamento estão apoiadas sempre no redirecionamento para os servidores que apresentem a menor carga no instante em que chega uma requisição.

Existem diversas formas de sincronismo entre servidores, como sincronismo de tempo, de processos e de dados. No caso da distribuição de processamento entre servidores *web* conectados em rede, a forma que adquire mais importância é o sincronismo de dados, já que o objetivo é a manutenção da integridade e consistência das páginas do *site web* entre os diversos servidores, e se possível, manter esta afinidade quando for necessário o acréscimo de mais equipamentos.

2.3.6 Uso de múltiplas *threads*

Sempre que um *site web* for construído, deve-se levar em conta ferramentas que permitam desenvolver aplicações *multi threaded*. O uso de *threads* indica como a aplicação responde a múltiplas requisições de usuários aos serviços que oferece. Se a aplicação for *single threaded*, poderá responder somente a uma requisição de usuário de cada vez, gerando um fila para atendimento das requisições, que são respondidas individualmente na ordem em que forem chegando. Um *site web multi threaded* está dentro do paradigma de escalabilidade, permitindo que um serviço de determinada aplicação possa atender múltiplas requisições simultaneamente, instanciando conexões ou *threads* separadas para cada requisição de

usuário. Aplicações que estejam preparadas para muita atividade de usuários concorrentes devem assegurar que sejam *multi threaded* (ALLAIRE, 2000).

2.3.7 Natureza das aplicações

Para que as aplicações *web* estejam preparadas para escalabilidade, devem estar adequadamente desenvolvidas e terem seus serviços corretamente particionados. Um objetivo-chave de desenvolvimento deve orientar o desenvolvimento no sentido de que cada parte ou módulo permita escalabilidade independente dos outros, de forma a evitar gargalos na própria aplicação.

Particionamento de aplicações refere-se ao desenvolvimento lógico e físico de três categorias básicas de serviços: *apresentação*, *negócios* e *acesso a dados*. Este tipo de projeto também está alinhado com o desenvolvimento de três camadas das aplicações cliente / servidor.

O serviço de *apresentação* determina como serão as interações do usuário com as facilidades da aplicação. Muitas tecnologias estão disponíveis, como Java *scripts*, *applets*, DHTML e ColdFusion, sendo que o uso adequado destas tecnologias pode minimizar o trabalho desnecessário do servidor, e evitar degradação de desempenho.

O serviço de *negócios* representa a lógica e as regras personalizadas do negócio. Dependendo do negócio e da freqüência com que suas regras são alteradas, deve-se implementar a lógica num servidor específico de fácil acesso, de tal forma que possibilite alterações rápidas. Outra solução é armazenar a lógica em *procedures* no servidor de banco de dados.

O serviço de *dados* refere-se à interação entre a aplicação e o banco de dados no qual são armazenados e manipulados os dados. A forma como os serviços de dados são manipulados afeta diretamente a capacidade de desempenho da aplicação. O tipo de *drivers* de bancos de dados usados para conexões (nativos ou ODBC), a construção costumeira de

queries SQL, a forma como as conexões ao banco de dados são estabelecidas e mantidas e a forma de implementar procedimentos fixos para acesso freqüente aos dados, são fatores que impactam diretamente o desempenho (ALLAIRE, 2000).

2.4. Aspectos desejáveis no processamento distribuído

2.4.1 Tolerância a falhas

A tolerância a falhas passou a ser um requisito altamente desejável, principalmente em sistemas críticos, dos quais dependam, por exemplo, a vida humana, os chamados *humanrated systems* (TANEMBAUM, 1997). Em sistemas distribuídos, existem interações entre processadores ou computadores por meio de um barramento ou mesmo por meio de conexões de rede, devendo haver preocupação especial para que as falhas que ocorram não determinem a perda da capacidade de atender às requisições dos usuários do sistema. Ou seja, tolerar falhas (obviamente até um determinado limite possível) não significa perder o controle que existe sobre o funcionamento do Servidor. Tolerância a falhas também é uma característica de *sobrevivência* de redes.

2.4.2 Confiabilidade e disponibilidade

Um aspecto determinante envolvido na construção de *site web* escalável é justamente proporcionar uma sensação de robustez ao usuário, proporcionando presença ininterrupta dos serviços. Muitos *sites* de comércio eletrônico, por exemplo, fazem da Internet o seu sustentáculo, e não podem correr o risco de ficarem indisponíveis. A confiabilidade e a disponibilidade aparecem como os ativos mais relevantes, e se ocorrerem falhas nestes quesitos podem comprometer seriamente as estratégias de negócio.

Em termos simples, confiabilidade e disponibilidade significam que um determinado *site web* pode ser acessado a qualquer momento em que se desejar, mediante a informação da URL no *browser* do cliente, e que todas as suas funcionalidades estarão disponíveis conforme

a expectativa do usuário. Estas características estão ainda relacionadas com o tempo em que o *site* estiver com as suas funcionalidades todas plenamente operacionais.

Esta disponibilidade também se estende a todos os servidores, como servidores de bancos de dados, de aplicações ou de arquivos. Para que o *site* seja considerado disponível, todos os servidores que provêm a sua funcionalidade devem estar funcionando.

A seguir estão descritas três tipos de falhas típicas que poderão colocar um servidor web transacional fora de serviço, falhas de hardware, falhas de software, ou falhas de servidores.

Primeiramente, falhas de *hardware, e*mbora sejam menos comuns do que falhas de *software*, estas podem eventualmente ocorrer, incluindo danos no disco rígido, pane do processador e danos nas placas de rede. A resolução destas falhas poderá demandar tempo razoável, devido à necessidade de busca de peças e tempo para reparação. Se a aplicação do *site web* for de *missão crítica*, deve ser agregada uma razoável redundância para evitar custos por paralisação de serviços. Uma boa estratégia inclui prever servidores adicionais, suportando assim possíveis falhas.

As falhas mais comuns de *software* estão situadas no sistema operacional, no *software* agregado ao servidor e na aplicação *web*. Se o sistema operacional tiver problemas, o servidor poderá não funcionar corretamente ou mesmo parar de funcionar, prejudicando a disponibilidade, confiabilidade e o desempenho do sistema. Se o *software* do servidor funcionar incorretamente, poderá fazer com que o servidor pare de trabalhar quando menos se espera. Para fazer frente a falhas de *software*, é importante que haja espelhamento adequado, que possa minimizar o tempo de indisponibilidade.

Em conjunto com o servidor *web*, outros servidores que compõem o cenário do *site web* podem falhar, trazendo também como conseqüência reflexos que podem diminuir ou eliminar as capacidades do sistema. Se por exemplo, um *proxy server* utilizado em aplicações

distribuídas ficar fora de serviço, as requisições destinadas a aplicações *web* não serão respondidas. Se o servidor de banco de dados ficar fora de serviço, as informações do banco não poderão ser recuperadas ou atualizadas. Ou, se o servidor de e-*mail* apresentar problemas, as comunicações da empresa com o mundo podem ficar seriamente comprometidas. É importante que a arquitetura do *site web* leve em consideração monitoramento de redes e aplicativos de notificação que possam rapidamente diagnosticar as falhas e produzir alertas (ALLAIRE, 2000).

2.4.3 Transparência

Os usuários de um *site web* devem ficar alheios aos redirecionamentos que ocorrem entre os servidores, enviando as requisições e recebendo as respostas sem vislumbrar qual o servidor encarregado do processamento. A distribuição *imperceptível* do processamento deve ser traduzida efetivamente em baixa latência ao usuário e ocultação de transferências de processos entre servidores.

2.5 Estudos de Processamento Distribuído

Neste ponto, passam a ser demonstradas algumas soluções adotadas para a distribuição de processamento entre servidores *web*. Entre os casos relacionados, serão apresentadas propostas acadêmicas, respaldada por experimentos, e também por aplicações concretas, decorrentes da necessidade de se obter o balanceamento de carga nos acessos concorrentes ao *site web*.

2.5.1 Servidores Web Cooperativos

Várias vantagens podem ser enumeradas pela adoção do DCWS (BACKER and MOON, 1998), visando à distribuição de documentos entre servidores. Estas técnicas foram batizadas com o nome de *DCWS - Distributed Cooperative Web Servers*, e o seu objetivo é

eliminar gargalos em recursos centralizados, pelo balanceamento de carga entre servidores distribuídos.

Qualquer sistema distribuído requer algumas técnicas para distribuir a carga de processamento entre os diversos servidores. Em muitos sistemas *web* escaláveis, a carga distribuída pelo nível de rede, usando alguma forma de roteamento específica. O roteador intercepta os pacotes que chegam, traduz o endereço IP para o endereço de um servidor e encaminha o pacote para uma determinada sub-rede. Desta forma, porém, o roteador cria um gargalo ao interceptar todos os pacotes.

Outra forma tradicional de distribuição de carga é mediante o uso de servidores DNS personalizados, distribuindo requisições que chegam a um mesmo *host* para vários endereços IP distintos. Esta solução, porém, obriga que os mapeamentos DNS sejam armazenados em muitos níveis dentro da hierarquia de serviços. Os mapeamentos DNS tem um parâmetro *TTL - time-to-live* que especifica quanto tempo uma informação será considerada válida. O parâmetro TTL é contraditório: um valor baixo produzirá maior controle sobre a distribuição a ponto de criar um gargalo no servidor DNS, ao passo que um valor alto produzirá menos controle sobre a distribuição, apesar de gerar menos carga no DNS.

Dentro dos conceitos do DCWS, os documentos *web* são visualizados como sendo um grafo, onde cada documento é um *nó* e cada *hyperlink* é uma conexão direta ou uma referência de imagem de um nó para o outro. A idéia é justamente distribuir este grafo entre vários servidores de uma forma tal que a carga de processamento também seja distribuída, ao serem alterados dinamicamente os padrões de acesso ao *site web*. Assim, estará resolvido o problema do balanceamento de carga, uma das questões mais cruciais na implementação de um servidor *web* distribuído. A solução do DCWS parte do princípio de que o acesso ao *site* dar-se-á sempre por meio de um *ponto de entrada conhecido*, que não é nada mais nada menos do que a URL digitada pelo usuário no *browser* da sua máquina cliente.

A proposta, então, é modificar dinamicamente os documentos *web*, alterando a sua conectividade, e distribuindo-os por meio dos vários servidores cooperativos, ou *co-op servers*.

Segundo os autores Scott Baker e Bongki Moon (BAKER, 1998), várias vantagens podem ser enumeradas pela adoção do DCWS, quando comparado a sistemas tradicionais baseados na manipulação de pacotes, ou baseados em servidores DNS, ou ainda em sistemas de arquivos distribuídos:

No DCWS, não são necessários tratamentos no nível de rede ou de transporte. Não existe qualquer roteador que necessite analisar pacotes que trafeguem entre um cliente e um servidor, eliminando os gargalos normalmente encontrados nos sistemas tradicionais.

Ao invés de fazer balanceamento de carga usando servidores DNS, os *servidores cooperativos* fazem uso da conectividade por meio de *hyperlinks* para controlar diretamente o balanceamento de carga em baixíssima granularidade nos documentos *web*.

Os *servidores cooperativos* não necessitam estar localizados dentro do mesmo domínio administrativo ou LAN. Podem estar geograficamente dispersos e podem distribuir tráfego por meio de múltiplas redes.

O acréscimo de novos servidores fica simplificado, flexível e com custo efetivamente baixo. Qualquer máquina disponível pode ser transformada em um *servidor cooperativo*, sem considerar a sua localização em relação aos outros servidores já existentes.

2.5.2 Princípios orientadores da solução

O DCWS baseia-se no fato de que a maioria dos *site web* tem único ponto de entrada conhecido, que é na realidade a *URL - Uniform Resource Locator* com que o *site* é conhecido na rede mundial. A partir desta realidade, a solução do DCWS registrou as seguintes observações sobre documentos *web*:

- Documentos *web* poderão ou não ter a sua URL publicada. Páginas que não tenham a sua URL publicada são normalmente acessadas na Internet por meio de um acesso inicial à URL conhecida (página *index.htm* ou *default.htm* ou qualquer outro nome), e depois o acesso é transferido por *hyperlinks* à página de destino.

- As URL´s das imagens embutidas dentro de um documento quase nunca são publicadas. Isto não é necessário, pois as imagens são automaticamente carregadas a partir dos acessos aos documentos HTML. Imagens também são consumidoras de largura de banda, quando trafegam pela Internet.

- O uso de *frames* em *site web* usualmente leva à publicação do modelo ou *template* do *frame* sem quase nunca publicar páginas de *frames* internos. O *template* é normalmente pequeno e facilmente hospedado no *home server*, enquanto que as páginas de *frames* internos poderão ser bem maiores, devendo ser migradas para outros servidores objetivando o balanceamento de carga.

- Os usuários não estão preocupados com as URL´s das páginas internas de um *site web*, visto que o acesso será sempre pela URL principal.

O DCWS define o servidor principal como *home server*, e os servidores agregados como *co-op servers*. Na instalação do *site web*, todos os documentos residem no *home server*, onde também são mantidas cópias dos documentos para conferir consistência e robustez à solução. Os pontos de entrada ou URL´s publicadas do *site* sempre estarão residentes no *home server*, ao passo que outros documentos vinculados aos documentos principais serão suscetíveis de migração para os *co-op servers* - a função destes servidores será sempre fazer o balanceamento de carga do servidor principal.

Na migração de documentos, os *hyperlinks* são modificados para redirecionar requisições de usuários de um servidor para outro, obtendo balanceamento de carga entre o *homeserver* e os *co-op servers*.

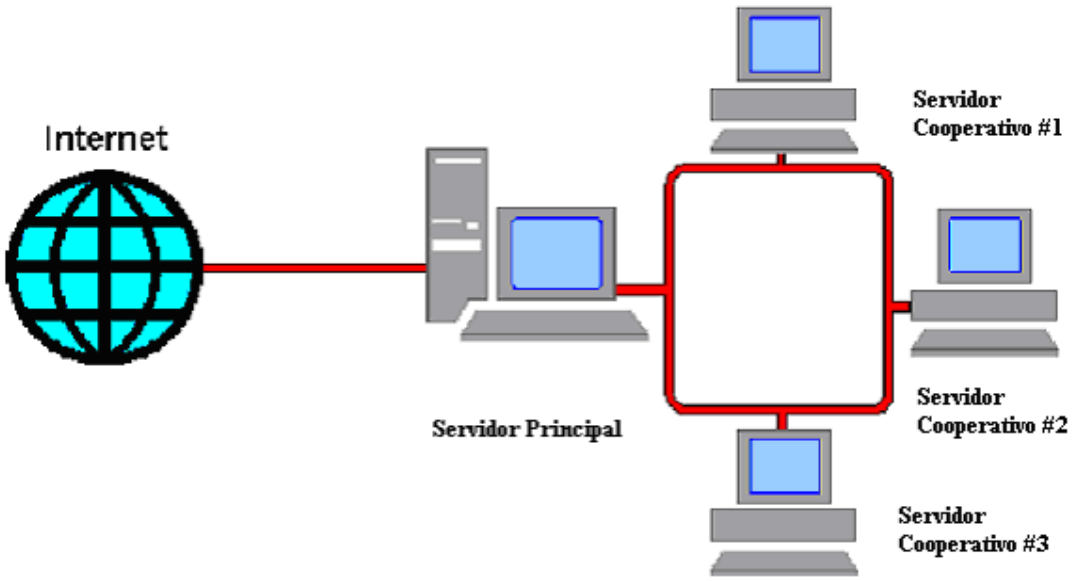

Figura 7:Esquema de servidores *web* cooperativos distribuídos.

2.5.3 Servidores Web Distribuidos com Balanceamento Geográfico

Objetivando a necessidade de ampliar a possibilidade de atendimento aos usuários que viessem em busca de informações sobre terremotos, trabalhos realizados (PRESCOTT, 1999) no Instituto de Pesquisas Geológicas dos Estados Unidos - U.S. Geological Survey (USGS), vieram a implementar uma solução para distribuir os acessos ao sistema de informações sobre terremotos dos EUA em 3 equipamentos geograficamente dispersos. Este sistema está baseado no conceito de DNS, e foi apropriadamente denominado de DNS para Riscos de Terremotos, ou simplesmente *EHZDNS - Earthquake Hazard Domain Name Service*.

O sistema se vale de um conjunto de *CGI scripts* em linguagem PERL, a cujo encargo está a distribuição do processamento de requisições oriundas dos usuários interessados em obter informações sobre tremores de terra em todo o mundo.

O EHZDNS era inicialmente baseado em único computador, o que provocava sérios problemas, desde a falta de opção quando houvesse a necessidade de parar o equipamento para manutenções, até a subjugação total do servidor e da rede nos casos de picos de acesso durante a ocorrência de terremotos. A falha maior foi em 1996, no 90º aniversário do terremoto ocorrido em 18 de Abril de 1906 em San Francisco: um dos sismologistas do USGS fez uma simulação pública dos novos mapas de terremotos, provocando uma avalanche de acessos que inundou o servidor e suas conexões.

A partir de então, houve a decisão de se fazer o *upgrade* do servidor, ao mesmo tempo em que se fez a integração de três computadores operando de forma cooperativa. Os computadores foram instalados propositalmente em áreas geográficas distintas, agregando um componente altamente desejável, além da escalabilidade: a sobrevivência da rede de servidores ante a ocorrência de um desastre de tal ordem que pudesse tirar algum dos servidores de operação. Assim, foram instalados dois servidores em Menlo Park e outro em Pasadena, todos no estado da Califórnia, Estados Unidos.

Cada um dos computadores utilizados roda o Sistema Operacional Linux e cinco daemons HTTP, e funcionam trocando dinamicamente dados como mapas de terremotos e listas de *hipocentros*, tudo com atualização em tempo real nos discos rígidos das três máquinas.

Páginas estáticas ou que não sofrem mudanças freqüentemente são mantidas em uma única máquina local, sendo distribuídas durante a noite para os outros servidores.

Não obstante ter sido resolvido o problema de disponibilidade das informações do *site web*, algumas falhas surgiram logo após a implementação dos três novos servidores. Uma das falhas apareceu quando o roteador que conectava as máquinas de Menlo Park à Internet ficou fora de serviço em Fevereiro de 1998. Em virtude de o servidor DNS em *wr.usgs.gov* ter sido configurado com um TTL (*Time do live*, período que as informações ficam no cache do

servidor DNS) de 3 dias, não era possível o redirecionamento para a máquina de Pasadena, que continha todas as páginas atualizadas.

Era preciso que o número IP em Menlo Park expire, para que *browsers* clientes com os endereços antigos no *cache* apontassem para o servidor alternativo.

2.6 Estratégias de Balanceamento de Carga em Servidores Web Transacionais

Um desafio enfrentado pelos servidores agrupados, no caso de servidores dinâmicos como os de comércio eletrônico, é que o ganho de desempenho (*speed-up*), isto é, a melhoria do desempenho resultante do aumento do número de processadores, não é linear. A distribuição de requisições entre um número de servidores impõe novas demandas de processamento para garantir a consistência das informações armazenadas por eles. Por exemplo, se dois clientes, sendo atendidos por dois servidores diferentes, tentam adquirir o mesmo produto, os dois servidores devem garantir que o produto não será vendido duas vezes. Esta computação adicional pode crescer a um ponto tal que o acréscimo de servidores pode até piorar o desempenho global.

Outra fonte de custos adicionais, além da manutenção da consistência, é o gerenciamento da distribuição de requisições aliado à manutenção de estado. O protocolo HTTP não prevê a manutenção de estado, de tal forma que cada requisição é tratada de forma independente de outras submetidas pelo mesmo usuário, podendo ser enviada a qualquer servidor que compõe o servidor agrupado. Entretanto, em serviços de comércio eletrônico, por exemplo, a interação do cliente com o servidor cria certa quantidade de informação de estado, como os produtos adicionados a uma cesta de compras e identificação dos usuários, entre outros. Nesse caso, a distribuição de requisições deve considerar a existência e localização dessa informação de estado, o que torna a tarefa mais complexa. Todas essas características

podem resultar em desbalanceamento de carga entre servidores que são responsáveis por atender as requisições (MEIRA Jr., 2000).

Estratégias de balanceamento de carga usualmente se baseiam no princípio de que uma redistribuição bem planejada de carga entre os servidores equaliza as cargas dos mesmos. O planejamento das operações de transferência de carga entre servidores por sua vez se baseia no princípio de que a carga migrada mantém a sua intensidade após a migração (é possível prever o comportamento da carga após sua transferência). O que se verifica na prática, entretanto, é que a carga de trabalho de servidores de comércio eletrônico normalmente é caracterizada por uma alta variabilidade em diversos dos seus componentes (TAVARES, 2003). Essa variabilidade afeta significativamente a eficácia de estratégias de balanceamento de carga, pois torna o trabalho de prever o comportamento futuro da carga a ser migrada um grande desafio.

2.6.1 Servidores Web Transacionais

Com o objetivo de entender os detalhes de um servidor Web transacional, é necessário conhecer as relações entre as várias entidades envolvidas e a arquitetura do servidor. Sem perda de generalidade, pode-se discutir os detalhes de servidores transacionais por meio da análise de uma arquitetura típica: servidores de comércio eletrônico, ou lojas virtuais (MEIRA Jr. 2000).

Pode-se classificar as cargas transacionais na web usando as formas básicas de: entidades, organização em níveis e a organização dos dados.

Para a organização em entidades, são três os serviços básicos em comércio eletrônico (MEIRA Jr., 2000): produtos, clientes e sessões.

A primeira entidade compreende os objetos das transações, havendo dois tipos de dados a respeito deles: estático e dinâmico. Dados estáticos modelam informações que não são afetadas pelos serviços providos pelo servidor, como a descrição de um livro, ou as

características de um equipamento eletrônico. Dados dinâmicos, por outro lado, modelam informações que são alteradas pelas operações executadas durante o provimento dos serviços, como o estoque dos produtos, ou a identidade do comprador.

A segunda entidade representa os clientes, que também demandam o armazenamento de dados estáticos e dinâmicos. Neste caso, informações como nome e endereço são estáticas, enquanto outros atributos, como o conteúdo da cesta de compras, são dinâmicos. A identificação de dados estáticos e dinâmicos é crucial para a distribuição dos serviços em servidores agrupados. Enquanto dados estáticos podem ser facilmente replicados nos servidores agrupados, acessos a dados dinâmicos devem ser controlados de forma a evitar inconsistências.

A terceira entidade representa a relação entre as duas primeiras entidades, isto é, a interação entre clientes e produtos. Essa interação é sintetizada pela sessão de usuário no contexto de servidores de comércio eletrônico, a qual é construída à medida que o servidor responde às requisições do usuário. Uma sessão de servidor de comércio eletrônico combina referências a dados tanto estáticos quanto dinâmicos de clientes e produtos que estejam associados à sua interação (MEIRA Jr, 2000).

Outra forma de classificar cargas transacionais na web é usando a organização em níveis e a organização dos dados.

Para armazenamento de dados e atendimento às requisições, os servidores transacionais são estruturados como uma arquitetura de três níveis (MENASCÉ, 2003): servidor WWW, servidor de aplicações e servidor de banco de dados. Esses níveis são ilustrados na Figura 8 e discutidos a seguir.

a) **Servidor WWW**: É o gerente das tarefas, sendo responsável pela interface com os usuários (clientes), interagindo diretamente com eles, recebendo as solicitações de consultas ao banco de dados, disparando-as, e repassando os resultados. Provê a interface

entre a ferramenta de acesso do cliente (normalmente um *browser*) e o servidor de comércio eletrônico.

b) **Servidor de Aplicação**: Realiza o processamento das requisições submetidas ao servidor de comércio eletrônico, como a adição/remoção de um novo produto à cesta de compras. Implementa a lógica específica do negócio.

c) **Banco de Dados**: Armazena todas as informações da loja virtual, como descrição do produto e nível de estoque. Mais que um simples repositório, agrega uma série de funcionalidades que permitem o acesso padronizado, seguro e eficiente aos dados, por meio, por exemplo, da criação de índices e controle de acesso utilizando autenticação por usuário.

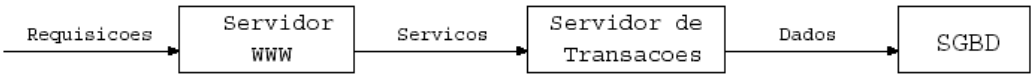

Figura 8: Arquitetura de servidor transacional de comércio eletrônico.

Em servidores de comércio eletrônico centralizados, os três níveis podem ser implementados como uma aplicação monolítica por razões de desempenho. Entretanto, na maioria dos casos, o serviço é implementado utilizando processos separados para cada um dos componentes. Uma primeira abordagem para melhorar o desempenho é simplesmente usar máquinas separadas para cada um dos componentes, o que tem um alcance limitado, pois para que ocorra escalabilidade efetiva é necessário que componentes de processamento sejam replicados e as requisições distribuídas adequadamente. No caso de serem usados servidores agrupados na implementação de serviços transacionais, esses são normalmente configurados como vários servidores WWW e de aplicação, com um só servidor de banco de dados.

É possível utilizar parte do agrupamento para se implementar um servidor de banco de dados distribuído, que é uma tecnologia mais complexa e dispendiosa que os bancos de dados tradicionais, porém essa ainda não é uma solução viável na maioria dos casos (MENASCÉ, 2003). Em geral, dados não são armazenados no servidor WWW, sendo divididos entre servidores de aplicação e banco de dados. Armazenar dados no servidor de banco de dados é uma solução mais robusta, mas associada a um maior custo, tanto em termos de recursos necessários, quanto em termos de latência de acesso.

Por último uma terceira forma de classificar cargas transacionais na web é usando a organização dos dados.

Distribuir dados estáticos pode ser visto como um problema de replicação. Cada servidor tem uma capacidade limitada de armazenamento e devem gerenciar esse espaço levando em consideração aspectos como custos de comunicação entre servidores, localidade de referência e custos de armazenamento no servidor. Considerando esses fatores, há várias estratégias de gerenciamento de replicação que podem ser aplicadas. Se o custo de armazenar é baixo, replicação total em todos os servidores de aplicação pode reduzir as latências de resposta, uma vez que todos os servidores vão conter todos os dados. Por outro lado, se o custo de armazenamento é alto, caches mutuamente exclusivos permitem um melhor aproveitamento do espaço, demandando, entretanto, significativa comunicação entre servidores. Algumas soluções intermediárias podem ser adotadas, replicando informações populares e reduzindo a quantidade de comunicação em alguns casos (MENASCÉ, et al, 2003).

Distribuir dados dinâmicos é bem mais complicado em razão das questões de coerência, ou seja, se um dado dinâmico é armazenado em dois servidores, deve haver uma forma de garantir que ambos os servidores serão notificados das mudanças. Este problema vem sendo pesquisado no contexto de banco de dados distribuídos. Entretanto, é interessante

notar que grande parte da carga de trabalho associada a clientes envolvem dados estáticos (MENASCÉ, et al, 2003) como descrição de produtos e imagens, o que representa uma oportunidade para replicação.

2.7 Servidores Web Transacionais Baseados em Clusters

Na sua forma mais básica um cluster é um sistema que compreende dois ou mais computadores ou sistemas (denominados nós) na qual trabalham em conjunto para executar aplicações ou realizar outras tarefas, de tal forma para que os usuários que os utilizam tenham a impressão que somente um único sistema responde para eles, criando assim uma ilusão de um recurso único (computador virtual). Este conceito é denominado transparência do sistema. Como características fundamentais para a construção destas plataformas inclui-se elevação da: confiança, distribuição de carga e performance.

2.7.1 Cluster de Alta Disponibilidade

A alta disponibilidade está ligada diretamente a crescente dependência aos computadores, pois agora eles possuem um papel crítico principalmente em empresas cuja maior funcionalidade é exatamente a oferta de algum serviço computacional, como e-business, notícias, sites web, banco de dados, dentre outros.

Um cluster de Alta Disponibilidade visa manter a disponibilidade dos serviços prestados por um sistema computacional replicando serviços e servidores, por meio da redundância de hardware e reconfiguração de software. Vários computadores juntos agindo como um só, cada um monitorando os outros e assumindo seus serviços caso alguns deles venham a falhar. A complexidade do sistema deve estar no software que deve se preocupar em monitorar outras máquinas de uma rede, saber que serviços estão sendo executados, quem os está executando, e como proceder em caso de uma falha. Perdas na performance ou na

capacidade de processamento são normalmente aceitáveis; o objetivo principal é não parar. Existem algumas exceções, como sistemas de tempo real e de missão crítica.

2.7.2 Cluster de Balanceamento de Carga

O balanceamento de carga entre servidores faz parte de uma solução abrangente em uma explosiva e crescente utilização da rede e da Internet. Provendo um aumento na capacidade da rede, melhorando a performance. Um consistente balanceamento de carga mostra-se hoje, como parte integrante de todo o projeto de Web Hosting e comércio eletrônico. Mas não se pode ficar com as idéias presas de que isso é só para provedores, deve-se aproveitar as suas características e trazer para dentro das empresas esse modo de usar a tecnologia para o atendimento dos clientes internos das empresas(VISWANATHAN, 2001).

Os sistemas de cluster baseado em balanceamento de carga integram seus nós para que todas as requisições provenientes dos clientes sejam distribuídas de maneira equilibrada entre os nós. Os sistemas não trabalham junto em um único processo, mas redirecionando as requisições de forma independente, assim que chegam baseados em um escalonador e um algoritmo próprio.

Este tipo de cluster é especialmente utilizado em serviços de comércio eletrônico e provedores de internet que necessitam resolver diferenças de carga provenientes de múltiplas requisições de entrada em tempo real. Adicionalmente, para que um cluster seja escalável, tem que assegurar que cada servidor seja utilizado completamente.

Quando não é feito o balanceamento de carga entre servidores que possuem a mesma capacidade de resposta a um cliente, começa a aparecer às dificuldades, pois um ou mais servidores podem responder a requisição feita e a comunicação fica prejudicada. Por isso é necessário colocar o elemento que fará o balanceamento entre os servidores e os usuários e configurá-lo para isso, entretanto pode-se colocar múltiplos servidores de um lado que, para os clientes, eles parecerão ser somente um endereço.

Balanceamento de carga é mais que um simples redirecionamento do tráfego dos clientes para outros servidores. Para implementação correta, o equipamento que fará o balanceamento precisa ter características como verificação permanente da comunicação, checagem dos servidores e redundância. Todos esses itens são necessários para que suporte a escalabilidade do volume de tráfego das redes sem vir a se tornar um gargalo ou um ponto único de falha. A Figura 9 apresenta uma distribuição e balanceamento de carga de atendimentos de requisições por tipo de processamento de página *web*.

Figura 9: Balanceamento de carga por tipo de processamento de página.

Os algoritmos para balanceamento são um dos fatores de maior importância neste contexto, há vários de algoritmos para seleção dos nós do cluster, que dependendo da situação, podem ter melhor ou pior performance.

a) Least Connections. Esta técnica redireciona as requisições para o servidor baseado no menor número de requisições/conexões. Por exemplo, se o servidor 1 está controlando atualmente 50 requisições/conexões, e o servidor 2 controla 25 requisições/conexões, a próxima requisição/conexão será automaticamente direcionado para o servidor 2, desde que atualmente o servidor tenha um número menor de requisições/conexões ativas.

b) Round Robin. Este método usa a técnica de sempre direcionar as requisições para o próximo servidor disponível de uma forma circular. Por exemplo, as conexões de entrada são dirigidas para o servidor 1, depois servidor 2 e finalmente servidor 3 e depois retorna ao servidor 1.

c) Weighted Fair. Esta técnica dirige os pedidos para os servidores baseados na carga de requisições de cada um e na capacidade de resposta dos mesmos (performance). Por exemplo, se o servidor 1 é quatro vezes mais rápido no atendimento aos pedidos do que o servidor 2, o administrador coloca um peso maior de trabalho para o servidor 1 do que o servidor 2

2.7.3 Clusters Beowulf e Alta Disponibilidade com Balanceamento de Carga

Esta solução combinada visa prover uma solução de alta performance aliada a possibilidade da não existência de paradas críticas. Este cluster combinado apresenta-se como uma solução adequada para ISP (*Internet Service Provider*, Servidor de Serviços na Internt) e aplicações de rede nas quais a continuidade de suas operações é muito crítica (VISWANATHAN, 2001).

Um dos mais notáveis avanços tecnológicos dos dias atuais, tem sido o crescimento da performance computacional dos PCs (Computadores Pessoais). A verdade é que o mercado de PCs é maior que o mercado de workstations, permitindo que o preço de um PC decresça, enquanto sua performance aumenta substancialmente, sobrepondo, em muitos casos, a performance de estações de trabalho dedicadas. A figura 10 apresenta uma configuração de cluster combinado.

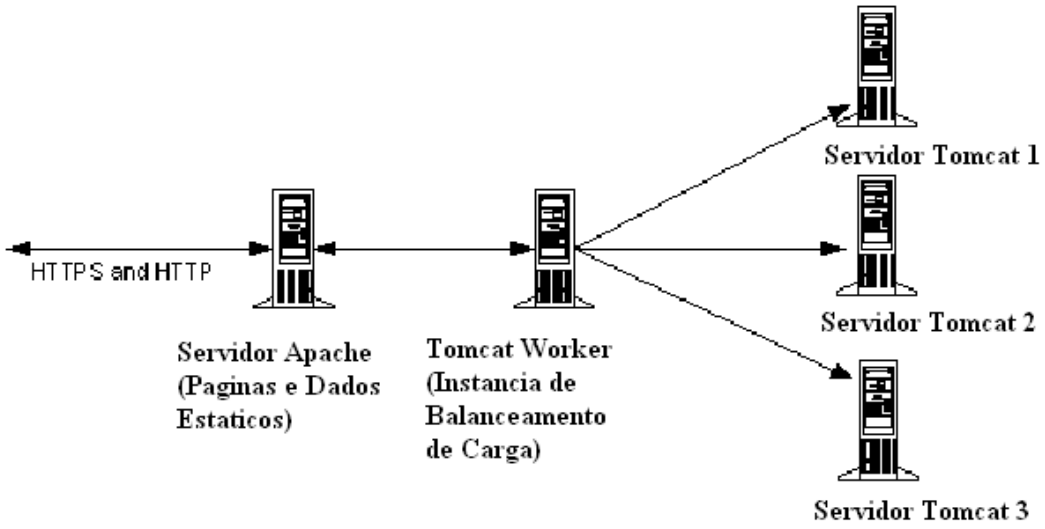

Figura 10: Balanceamento de carga em servidores web em cluster combinado.

Um dos mais notáveis avanços tecnológicos dos dias atuais, tem sido o crescimento da performance computacional dos PCs (Computadores Pessoais). A verdade é que o mercado de PCs é maior que o mercado de workstations, permitindo que o preço de um PC decresça, enquanto sua performance aumenta substancialmente, sobrepondo, em muitos casos, a performance de estações de trabalho dedicadas.

O cluster Beowulf foi idealizado com o objetivo de suprir a crescente e elevada capacidade de processamento em diversas áreas cientificas com o objetivo de construírem sistemas computacionais poderosos e economicamente viáveis. Claro que a evolução constante do desempenho dos processadores tem colaborado e muito na aproximação entre PCs e Workstations, a diminuição do custos das tecnologias de rede e dos próprios processadores e o sistema operacional aberto e gratuito, como o GNU/Linux em muito influenciam as pesquisas para melhoria desta nova filosofia de processamento de alto desempenho em clusters.

Uma característica chave de um cluster Beowulf, é o software utilizado, que é de elevada performance e gratuito na maioria de suas ferramentas, como exemplo pode-se citar os sistemas operacionais GNU/Linux e FreeBSD sobre os quais estão instaladas as diversas ferramentas que viabilizam o processamento paralelo, como é o caso das API's MPI e PVM.

O trabalho do Cluster Beowulf é dividido em um nó controlador denominado front-end (normalmente denominado de nó mestre), cuja função é controlar o cluster, monitorando e distribuindo as tarefas, atua como servidor de arquivos e executa o elo entre os usuários e o cluster. Grandes sistemas em cluster podem distribuir diversos servidores de arquivos, nó de gerencia pela rede para não sobrecarregar o sistema. Os demais nós são conhecidos como clientes ou backends (denominados nós escravos), e são exclusivamente dedicados para processamento das tarefas enviadas pelo nó controlador.

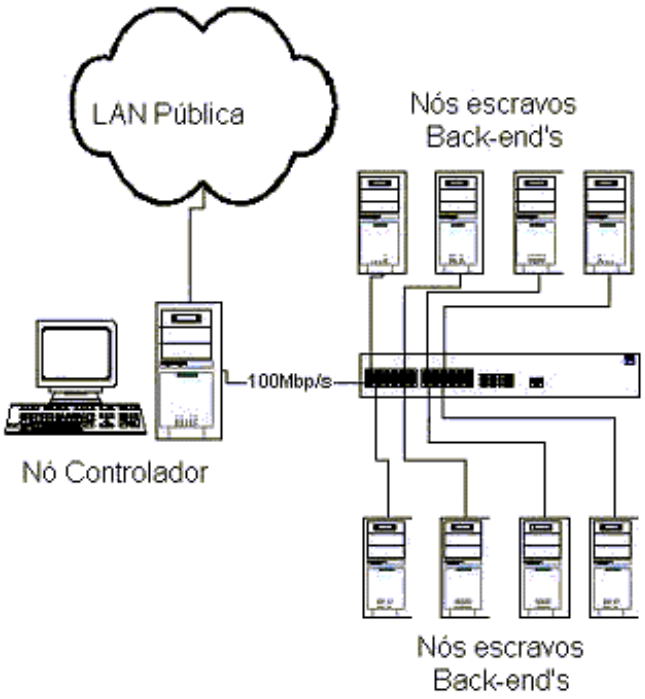

Figura 11: Cluster Beowulf para servidor web.

O Beowulf é um projeto bem sucedido. A opção feita por seus criadores de usar hardware popular e software aberto tornou-o fácil de se replicar e modificar, a prova disso é a grande quantidade de sistemas construídos à moda Beowulf em diversas universidades, empresas americanas e européias e até residenciais. Mais do que um experimento foi obtido um sistema de uso prático que continua sendo aperfeiçoado constantemente.

2.8 Conclusões e considerações sobre modelos de arquiteturas para servidores web

As situações apresentadas neste capítulo enfatizam a importância e complexidade do planejamento da capacidade dos sistemas de servidores web. Questões como escalabilidade, distribuição do processamento, balanceamento de carga, confiabilidade e disponibilidade, determinam o desempenho dos serviços web. Foram apresentados *benchmarks* que caracterizam o comportamento do tráfego na Intenet, como: carga de trabalho que será analisada; identificação dos componentes básicos da carga de trabalho; seleção de parâmetros que capturam as características mais relevantes para interpretação; particionamento da carga de trabalho em classes de componentes; calculo de valores que caracterizam cada classe. Os *benchmarks* padrão não são ferramentas adequadas para planejamento de capacidade de um sistema com uma carga de trabalho personalizada, contudo, os resultados de *benchmark* podem ser usados em conjunto com modelos de desempenho para fins de planejamento de capacidade.

Este capítulo também apresentou arquiteturas de cluster para serviços web, suas características a aplicações dependendo da camada de aplicação e as estratégias para distribuição e balanceamento de carga.

3. ASPECTOS DO DESEMPENHO DE SERVIÇOS NA INTERNET

Os servidores são os principais elementos da Internet e Intranets (MENASCÉ, 2002). Eles oferecem informações, na forma de texto, imagens, som, vídeo e diversas combinações multimídia destes. Os serviços na Web disponíveis na Internet e em intranets são fornecidos por vários tipos de servidores, usando uma grande variedade de computadores e software. Os servidores recebem, armazenam e encaminham informações na Internet e intranets. Vários tipos diferentes de servidores oferecem suporte a serviços na Web (MENASCÉ, 2002), como: servidores Web, servidores de transação, servidores de proxy, servidores de cache e servidores-espelho. Clientes da Internet e de intranets, conseguem comunicar um computador, utilizando software chamado *Browser* (Navegador), com outro cliente ou qualquer outro servidor na rede. A medida que as empresas utilizam cada vez mais sistemas baseados na Web, o desempenho dos serviços se torna cada vez mais importante. Quanto mais informações e serviços uma empresa disponibilizar em um Site web, mais pedidos ela receberá, e quanto mais pedidos um *site web* recebe, maior a probabilidade que os usuários ou clientes esperem mais tempo por uma resposta do sistema.

Os problemas de desempenho na Internet são aumentados pela natureza imprevisível dos pedidos de informação e serviços pela *Web*. As vezes, os *sites* são quase ociosos, sem visitantes ou requisições. De repente, sem aviso, o tráfego pode aumentar dez vezes (MENASCÉ, 2002). Esse tipo de pico de carga, também conhecido como "*flash crowds*", é enfrentado por muitos site web (MOGUL, 1995).

3.1 Cargas em rajadas

Existe um conjunto considerável de trabalhos sobre caracterização da carga de trabalho do trafego da Web. Algumas das características consideradas tratam das distribuições

de tamanho de arquivo, distribuição da popularidade do arquivo, auto-semelhança no trafego da Web, localidade de referência e padrões de requisição do usuário.

Em um estudo (CROVELLA, 1997), mostrou-se que o trafego da Web contém rajadas observáveis por diferentes escalas de intervalo de tempo, conforme Figura 12. As taxas de pico durante as rajadas excedem a taxa média por fatores de cinco a dez e podem superar a capacidade do servidor.

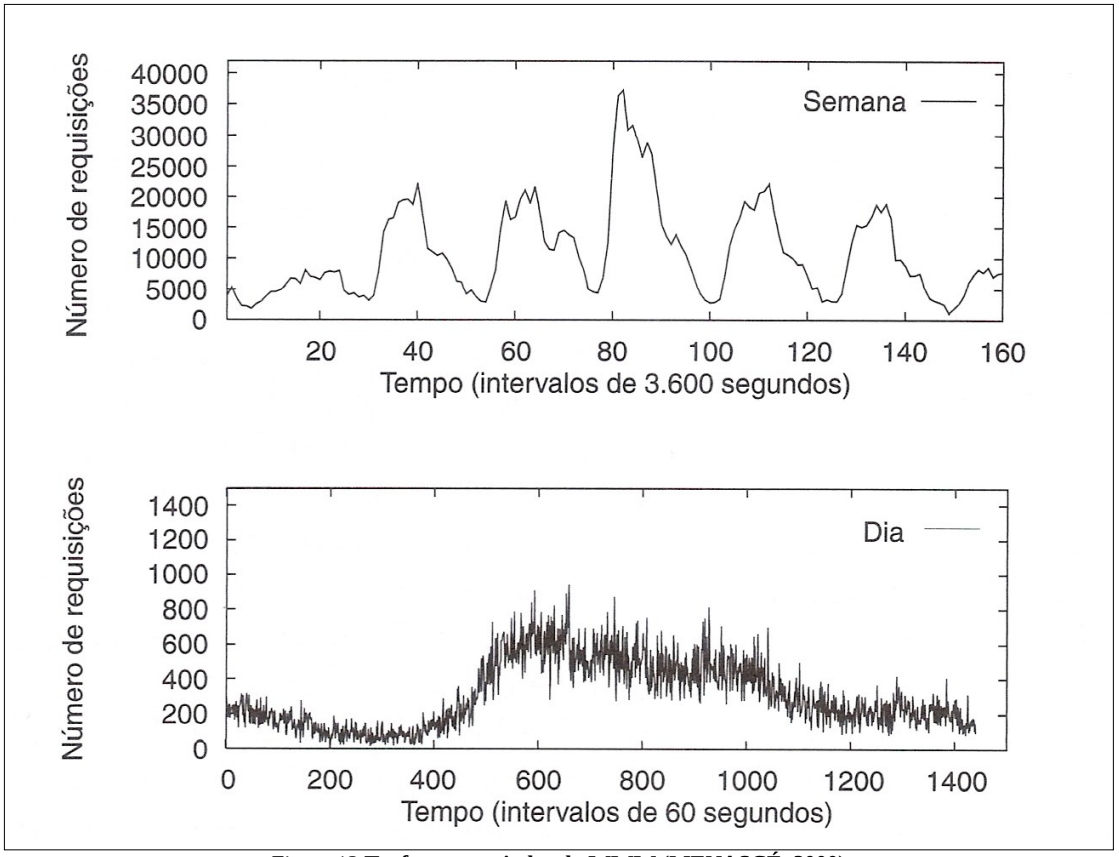

Figura 12 **Trafego em rajadas da WWW (MENASCÉ, 2000)**

3.2 Servidores e serviços na Web

Com a evolução da Web, os usuários não vêem servidores, nem subsistemas de armazenamento nem redes, mas vêem os serviços que esses dispositivos habilitam. Um serviço na Web é um serviço disponível por meio da Internet, que completa tarefas ou realiza transações (BANGA, 1997), os serviços na Web são aplicações autocontidas, modulares, que

podem ser descritas, publicadas e invocadas pela Internet. Um serviço na Web pode ser um processo comercial, uma aplicação ou ainda um recurso computacional. Os exemplos variam desde serviços de pagamento eletrônico até serviços de distribuição de conteúdo e serviços de armazenamento. Os serviços na Web podem ser invocados automaticamente pelos programas de aplicação, ou por um programa navegador (browser).

Um serviço na Web pode ser executado em um site ou pode ser um resultado da combinação de vários serviços executados em diferentes sites. De qualquer forma, o desempenho depende da infra-estrutura básica, composta de aplicações, servidores e redes. Muitos fatores podem contribuir para diminuir o desempenho de um serviço na Web.

Alguns dos problemas mais comuns estão relacionados a largura de banda insuficiente em horários de pico, servidores sobrecarregados, cargas desiguais no servidor, remessa de conteúdo dinâmico, conexões insuficientes entre servidores de aplicação e servidores de banco de dados, falha de serviços de terceiros e remessa de conteúdo de multimídia. Assim, é importante entender o comportamento de um provedor de serviços na Web, a fim de identificar os problemas exatos que prejudicam o desempenho.

3.3 Arquitetura de avaliação de desempenho

3.3.1 Sistema Operacional

O desempenho do servidor depende principalmente do comportamento de seus elementos básicos: a plataforma de *hardware* e o sistema operacional. Do ponto de vista do *hardware*, o desempenho de um servidor é função do número e velocidade dos seus processadores, da quantidade de memória principal, da largura de banda e da capacidade de armazenamento do subsistema de disco.

Os servidores Web podem rodar em cima de sistemas operacionais de tempo compartilhado e multiusuários, como Unix, Windows NT, Windows 2000, Linux e outros. A

confiabilidade, o desempenho, a escalabilidade e a resistência são alguns dos recursos que devem ser considerados quando se decide sobre um sistema operacional.

Neste trabalho a opção pelo Sistema Operacional foi pelo FreeBSD versão 6.01. O FreeBSD é um sistema operacional baseado no 4.4BSD-Lite, do Grupo de Pesquisa em Sistemas Computacionais (CSRG) da Universidade da Califórnia, Berkeley, para computadores Intel (x86), DEC Alpha™, e Sun UltraSPARC® (FreeBSD, 2006). O FreeBSD tem muitas características valiosas. Algumas destas são: *Multitarefa preemptiva* com ajustes dinâmicos de prioridade que garantem compartilhamento claro e racional do computador entre as aplicações e usuários, mesmo sob a mais intensa demanda. *Caracteristicas multi-usuário* que permite várias pessoas utilizarem um sistema FreeBSD de forma simultânea, para uma variedade de coisas. Isto implica, por exemplo, que os periféricos do sistema como impressoras e dispositivos de fita serão apropriadamente compartilhados entre todos usuários no sistema ou na rede, e que limites individuais possam ser definidos para usuários e grupos de usuários, protegendo recursos críticos do sistema de sobrecarga. *Proteção de memória* garante que aplicações (ou usuários) não interferirão entre si. A falha de uma aplicação não afetará outras de forma alguma. *Compatibilidade binária* com quaisquer programas compilados para Linux, SCO, SVR4, BSDI e NetBSD (FREEBSD, 2006).

A opção pelo Sistema Operacional FreeBSD também se deve ao fato da utilização de extensão de Kernel NAT (Network Address Translator), que implementa por meio das rotinas de IPFIREWALL o redirecionamento dos endereços IP (Internet Protocol) e ainda pela facilidade da implementação da Biblioteca Libpcab, que é uma biblioteca para captura de pacotes que permite o monitoramento dos pacotes e a utilização de filtros BPF(Berkeley Packet Filter). A Libpcap foi desenvolvida no Lawrence Berkeley National Laboratory, Universidade da Califórnia, Berkeley, CA. É um eficiente e simples mecanismo quando utilizado em uma rede, é ideal para monitorar o seu tráfego(FREEBSD,2006).

3.3.2 Servidor de Web (no protocolo http)

Neste trabalho, a opção foi pelo servidor de *Web* Apache, versão 2.0. O Apache tem como base o servidor *Web* NCSA 1.3 (*National Center of Supercomputing Applications*), da universidade de Illinois, nos Estados Unidos da América. A primeira versão oficial do Apache foi a 0.6.2, lançada em Abril de 1995, sendo melhorada gradativamente, em 1999 os membros do grupo de pesquisa Apache formaram a *Apache Software Foundation*, para prover apoio organizacional, legal e financeiro ao projeto e conforme publicação na página institucional da fundação, em pesquisa realizada em novembro de 2005, pela empresa *Netcraf Web Server Su*rvey, 70% dos site web da Internet estavam utilizando o servidor de HTTP Apache.

Os documentos na Web são escritos em uma "linguagem de marcação" simples, chamada Hipertext Markup Language – HTML (BERNERS-LEE, 1994), que é o sistema usado para criar documentos de hipertexto. A HTML permite descrever a estrutura dos documentos, indicando títulos, ênfase, links para outros documentos, bem como a inclusão de imagens e outros objetos multimídia, permitindo a integração de vídeo, aplicações de software e objetos de multimídia na Web.

A maioria dos documentos HTML consiste em texto e imagens em linha, apontadas por referências (links). As imagens em linha representam um grande impacto sobre o desempenho do servidor. Quando um navegador analisa os dados HTML recebidos de um servidor, ele reconhece os links associados às imagens em linha e automaticamente solicita os arquivos de imagem apontados pelos links. Na realidade, um documento HTML com imagens em linha é uma combinação de vários objetos, e que gera várias requisições separadas ao servidor, enquanto o usuário vê apenas um documento, o servidor vê uma série de requisições separadas para esse documento.

Em termos de desempenho do sistema, é importante entender que um único clique dado por um usuário pode gerar uma serie de requisições de arquivos a um servidor.

O http é um protocolo ao nível de aplicação, usado na comunicação entre clientes e servidores na Web e define uma única interação de requisição e resposta, que pode ser vista como uma “transação Web”. Cada interação consiste em uma requisição enviada do cliente ao servidor, seguida por uma resposta enviada de volta pelo servidor ao cliente.

Quando um servidor recebe uma requisição, ele analisa a requisição e toma a ação especificada pelo método, o servidor, então, responde ao cliente nas seguintes etapas: a) uma linha de status, indicando o sucesso ou falha da requisição; b) meta-informações descrevendo o tipo do objeto retornado e a informação solicitada; c) um arquivo ou uma saída gerada por uma aplicação no servidor. Cada uma dessas etapas possui um custo inerente, que depende do desempenho do servidor e da rede.

O protocolo HTTP é conhecido como um “protocolo sem estado”, pois não inclui o conceito de sessão ou interação além da entrega do documento solicitado. Uma conversação é restrita à transferência de um documento ou imagem, onde cada transferência é totalmente isolada da requisição anterior ou seguinte. Considerando que um site web é acessível a milhões de clientes, a ausência de estado do protocolo HTTP aumenta a eficiência do sistema, pois os servidores não precisam controlar o registro de quem são os clientes ou que requisições foram atendidas.

3.3.3 Aplicativo para balanceamento de carga

A maior dificuldade para a implementação de cluster *web* é o sistema de balanceamento de carga, onde além de dispor algoritmo de redirecionamento das requisições, precisa manter a integridade das sessões iniciadas. Para este trabalho a opção foi pelo aplicativo Pen. O Pen é um "*load balancer*", ou um balanceador de carga, para serviços baseados em protocolo tcp. Ele permite que um servidor principal distribua serviços de requisições entre vários servidores secundários (PEN, 2006).

Ao contrário de vários balanceadores de carga, o Pen além de implementar um algoritmo circular (round-robin) para a distribuição de conexões, o Pen mantém um registro sobre os clientes, em caso de uma re-conexão por parte de um cliente, a requisição será enviada para o mesmo servidor que foi utilizado anteriormente, isto permite que algumas aplicações que utilizam cache e variáveis trabalham normalmente por trás do balanceador.

O Pen monitora freqüentemente o estado dos servidores para os quais ele irá desviar a carga, caso o servidor se torne indisponível, ele será retirado da lista de servidores e caso ele torne-se novamente disponível, ele será adicionado a fila de servidores (OLIVEIRA, 2006).

3.3.4 Aplicativo utilitário de avaliação de desempenho

Na avaliação de desempenho das arquiteturas, a opção foi pelo software *Siege* versão 2.65, que além de ser um software livre, apresenta características importantes na implementação, principalmente nas rotinas de autoconfiguração, instalando os módulos compatíveis com a arquitetura a ser avaliada, outra característica é a configurabilidade dos parâmetros de teste, onde podem ser simulados a quantidade de clientes que estão tentando acessar o serviço simultaneamente em um determinado intervalo de tempo, tendo como resposta o número de transações efetivadas, a quantidade de dados transferidos e conseqüentemente a vasão (*throughput*) em Megabytes por segundo (SIEGE, 2006).

3.4 Conclusões e considerações sobre servidores e serviços na web

Este capítulo apresentou as características dos componentes selecionados para a implementação da avaliação, suas principais características e critérios de escolha. O sistema operacional FreeBSD por ser um sistema de código aberto (open source), estável, multi-usuário, multi-tarefa, com todas as características favoráveis para servidores web. O servidor Apache, também de código aberto, utilizado na maioria dos servidores *web*. Os aplicativos

Pen para balanceamento de carga e o software Siege como gerador de carga, mais utilitário de *benchmark*, por apresentarem características de compatibilidade e portabilidade com o sistema operacional e com os objetivos do estudo.

4. AMBIENTE DE AVALIAÇÃO PARA WEB TRANSACIONAL

A arquitetura inicial para avaliação de desempenho foi montada a partir de um cluster com 3 microcomputadores Pentium, 166 Mhz, 64 Mbytes de memória RAM, com Sistema Operacional FreeBSD, versão 6.1 e Servidor de HTTP Apache versão 2.0, a conexão feita por meio de Hub de 10 Mbytes por segundo e software de Benchmark SIEGE versão 2.65.

Na configuração do *Web cluster*, foi utilizado o NAT (network address translator). O NAT funciona como um roteador transparente, também conhecido como IP *Masquerading,* é uma extensão do *Kernel* do sistema operacional FreeBSD, ele ira funcionar como um roteador para uma pequena LAN, o que significa depois de configurar o servidor é possível redirecionar as conexões das portas lógicas.

O primeiro passo é alterar o arquivo de configuração do sistema Operacional, incluindo as opções descritas na figura 13.

Primeiramente é necessário máquina com *FreeBSD* instalado com os pacotes de Kern-Developer, para encontrar os fontes do kernel para recompilá-lo.

Com os fontes do kernel, login como root e caminho até /usr/src/sys/i386/conf:

$ cd /usr/src/sys/i386/conf

$ cp GENERIC SERVER

Agora o fonte GENERIC do FreeBSD foi copiado para Server e nessa copia do fonte é editada as alterações das opções:

```
options         INCLUDE_CONFIG_FILE
options         IPFIREWALL
options         IPFIREWALL_FORWARD
options         IPFIREWALL_DEFAULT_TO_ACCEPT
options         IPFIREWALL_VERBOSE
options         IPFIREWALL_VERBOSE_LIMIT=100
options         IPDIVERT
options         DUMMYNET
options         TCP_DROP_SYNFIN
```

Figura 13: Parâmetros de configuração do Kernel FreeBSD.

Com as alterações feitas na cópia do fonte do *Kernel*, basta compilar este arquivo, preservando o fonte anterior “GENERIC”, com os comandos descritos na figura 14.

$ config SERVER

O comando irá retornar uma mensagem parecida com esta:

Don't forget to do a ``make depend''

Kernel build directory is ../../compile/SERVER

Instruindo: não esqueça do make depend e a configuração do kernel está em ../compile/SERVER.

$ cd ../compile/SERVER

$ make depend

$ make

$ make install

Se em nenhum destes passos ocorreu algum erro, o kernel estará compilado e deverá funcionar, então reinicia-se o servidor.

$ reboot

Figura 14: Rotina de compilação do *Kernel* FreeBSD.

Após a reinicialização do sistema , os comandos de direcionamento “IPFIREWALL” estarão habilitados sendo possível utilizar vários algoritmos de balanceamento de carga.

A implementação do balanceador de carga Pen é realizada de forma extremamente simples, via “*Ports*” do FreeBSD, ou seja, vem como pacote na distribuição do sistema operacional.

Instalando o Pen

O Pen está disponível para o FreeBSD via o Ports. O caminho para o ports é /usr/ports/net/pen.

Para instalar:

```
# cd /usr/ports/net/pen
# make install clean
```

Figura 15: Rotina para instalação do balanceador de carga Pen.

Utilizando o Pen

A sintaxe básica do Pen é

```
# pen EndereçoLocal:PortaLocal
EndereçoServidor1:Porta:LimiteDeConexões
EndereçoServidorn:Porta:LimiteDeConexões
```

Figura 16: Rotina para configuração e utilização do Pen.

4.1 Benchmark Siege

Foram feitas duas avaliações nessa arquitetura, a primeira chamada de “carga local”, onde o *software* de *Benchmark Siege* foi instalado no servidor principal e gerou a carga de trabalho no próprio Servidor e a segunda, chamada “carga remota” o *software* de *Benchmark Siege* foi instalado em um outro computador, fora da rede, gerando a carga de trabalho por meio de uma conexão de internet, conforme figura 17, nas duas situações foram simuladas cargas de requisição de 50 a 250 clientes acessando simultaneamente o servidor em um intervalo de um minuto.

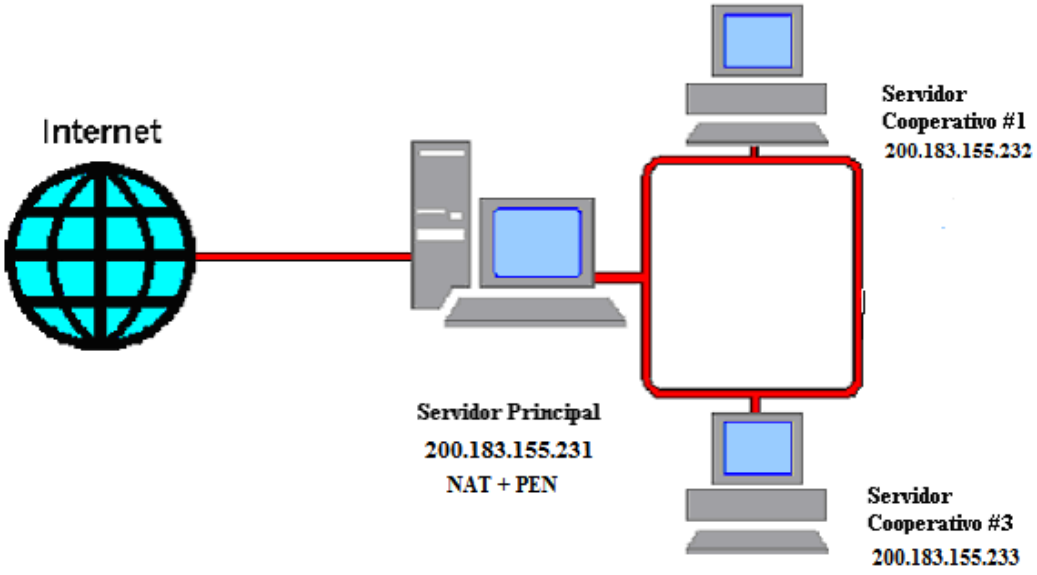

Figura 17: Arquitetura para avaliação do servidor cluster web

Na avaliação foram testadas quantas transações (“*hits*”) são executadas por minuto, taxa de transferência de dados em *Megabytes* por minuto, tempo de resposta por segundo das requisições, taxa de transações executadas por segundo, vazão (*throughput*) de dados em *Megabytes* por segundo, número de requisições simultâneas em concorrência , maior tempo de resposta de uma requisição e menor tempo de resposta de uma requisição.

4.2 Resultados gerados a partir da carga no servidor

Na tabela 1, estão descritos os resultados da avaliação do desempenho do servidor submetido a uma carga de requisições, onde na linha “clientes”, foram simuladas situações de 50, 100, 150, 200 e 250 clientes requisitando simultaneamente a página padrão do Servidor Apache, requisições estas geradas a partir do próprio servidor.

Tabela 1: Resultados gerados a partir de carga no Servidor

comando -> siege –c50 -t1M localhost					
clientes	50	100	150	200	250
transações (hits)	1455	1346	1327	1372	1033
disponibilidade (%)	100%	100%	100%	99%	100%
Tempo decorrido (seg)	64,35	62,56	60,48	60,52	64,66
Dados Transferidos (MB)	2,14	2,1	2,1	2	1,59
Tempo de Resposta (seg)	0,545	1,765	3,16	6,74	10,54
taxa de transação(trans/seg)	24,25	22,22	21,94	22,67	16,14
throughput (MB/seg)	0,04	0,03	0,03	0,03	0,025
Concorrência	22,83	40,97	69,04	152,75	158,73
Transações com sucesso	1480	1391	1346	1380	1037
Transações com falha	0	0	0	9	0
Maior tempo de transação	1,36	6,445	9,57	27,51	44,44
Menor tempo de transação	0,042	0,04	0,03	0,04	0,04

Na tabela 2 estão descritos os resultados da avaliação do desempenho do *Cluster Web* submetido a uma carga de requisições a página padrão do Servidor Apache, rigorosamente idênticas à primeira situação, de 50 a 250 clientes simultâneos, geradas agora, a partir de um computador remoto, por meio de uma conexão de rede, que envia as requisições para o Servidor principal, que distribui o processamento para os Servidores Cooperativos, por meio do software de balanceamento de carga *Pen.*

Tabela 2: Resultados gerados a partir de carga remota no cluster web

comando -> siege –c50 -t1M 200.183.155.231					
clientes	50	100	150	200	250
transações (hits)	2284	2201	2221	2207	2170
disponibilidade (%)	100%	100%	100%	100%	98%
Tempo decorrido (seg)	60,77	60,33	60,91	60,69	60,25
Dados Transferidos (MB)	3,3	3,18	3,21	3,19	3,15
Tempo de Resposta (seg)	0,59	1,9	3,27	4,31	4,63
taxa de transação(trans/seg)	37,58	36,48	36,47	36,37	36,02
throughput (MB/seg)	0,05	0,05	0,05	0,05	0,05
Concorrência	22,33	69,43	119,16	156,58	166,77
Transações com sucesso	2284	2201	2221	2207	2175
Transações com falha	0	0	0	5	35
Maior tempo de transação	1,45	2,73	5,95	25,92	29,5
Menor tempo de transação	0,14	0,23	0,34	0,24	0,08

Em uma analise inicial, já é possível observar uma alteração considerável na avaliação do desempenho, sendo que na avaliação descrita no quadro 1 o software de benchmark está compartilhando os recursos de processador e memória com o servidor de HTTP Apache, enquanto na avaliação descrita do quadro 2 o software de benchmark foi instalado em um computador remoto conectado por uma conexão de rede, chamado de “webmaster”, responsável pela geração da carga de trabalho, coleta e interpretação dos resultados. O computador “webmaster” envia as requisições para o Cluster Web, somente para o endereço 200.183.155.231, do Servidor Principal, estas requisições têm seus endereços reconfigurados pelo software de balanceamento de carga Pen e o redirecionamento das requisições é feito ao nível da camada de transporte pelo NAT (network address translator), o retorno das requisições será feito pelo endereço do Servidor Principal, de modo que o computador “webmaster” reconhece o Cluster Web como uma única máquina.

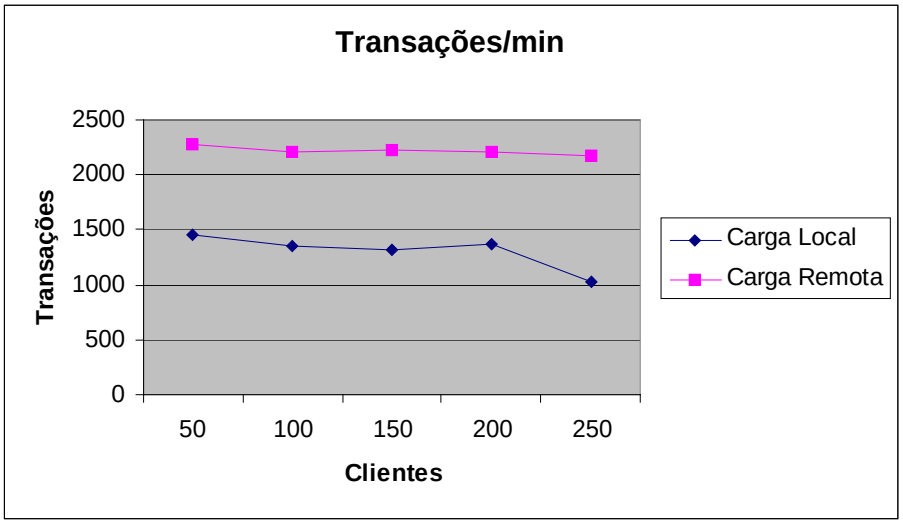

Figura 18: Gráfico de comparação transações realizadas por minuto

Na análise das transações realizadas por minuto, conforme Figura18, observando a linha de carga remota além do aumento considerável é possível observar uma estabilidade no atendimento das requisições mesmo acima de duzentos clientes simultâneos.

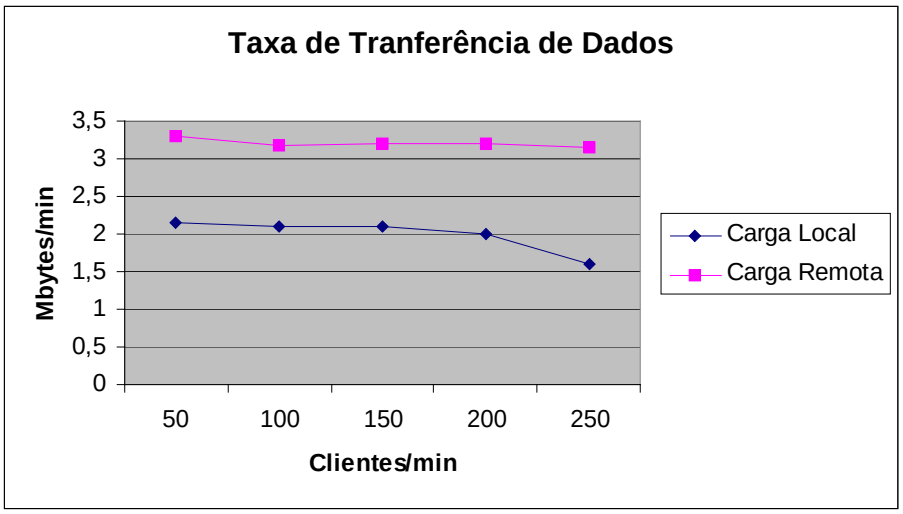

Figura 19: Gráfico comparativo da taxa de transferência de dados

A taxa de transferência de dados entre a máquina "Servidor" e as máquinas "Cliente", na carga remota além de um aumento considerável, manteve-se estável entre 3,15 Mbytes e 3,30 Mbytes por minuto.

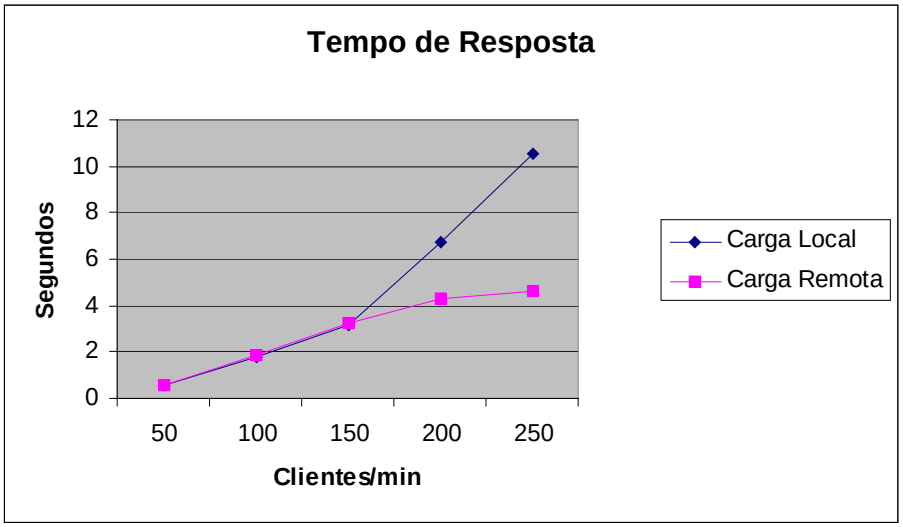

Figura 20: Gráfico de comparação do tempo médio de resposta das requisições

Um dos valores que melhor representam a performance de um Servidor Web é o tempo médio que atende e responde às requisições, onde na figura 20, pode-se observar que a partir de 150 clientes simultâneos, o modelo com a carga local começa a apresentar um

crescimento exponencial no tempo de resposta e o modelo com carga remota apresenta crescimento linear durante toda a simulação.

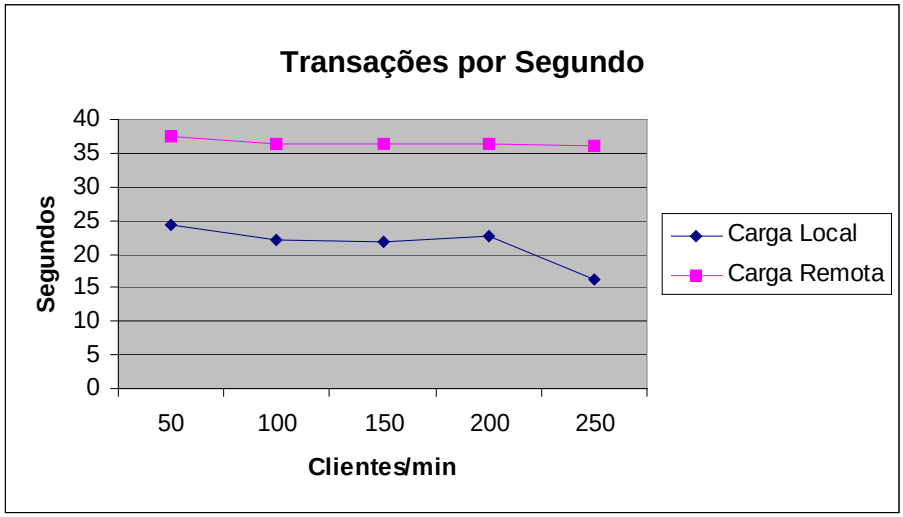

Figura 21: Gráfico comparativo das transações por segundo

A figura 21 demonstra, em média, quantas páginas podem ser enviadas por segundo, onde mais uma vez o modelo com carga remota apresenta um comportamento estável, enquanto o modelo com carga local começa a degradar a partir de 200 clientes simultâneos.

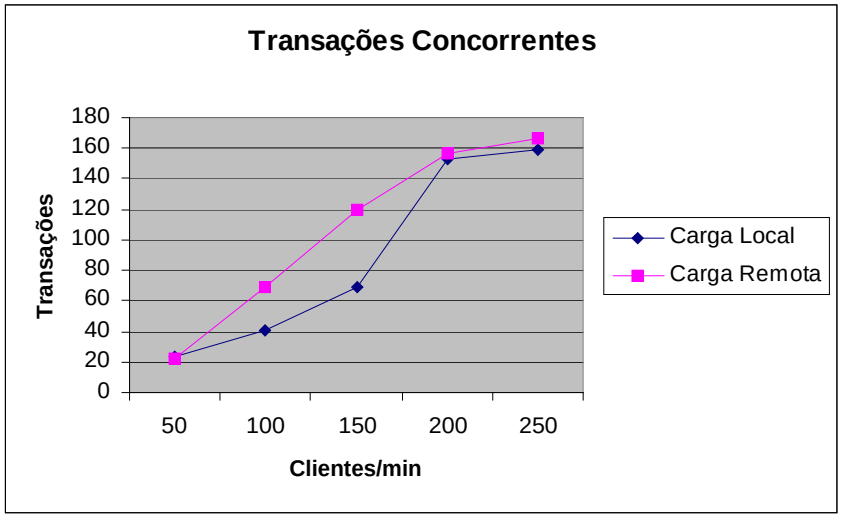

Figura 22: Gráfico comparativo das transações concorrentes

Na análise das transações concorrentes os dois modelos apresentam um aumento gradual, implicando no aumento do tempo de resposta das requisições.

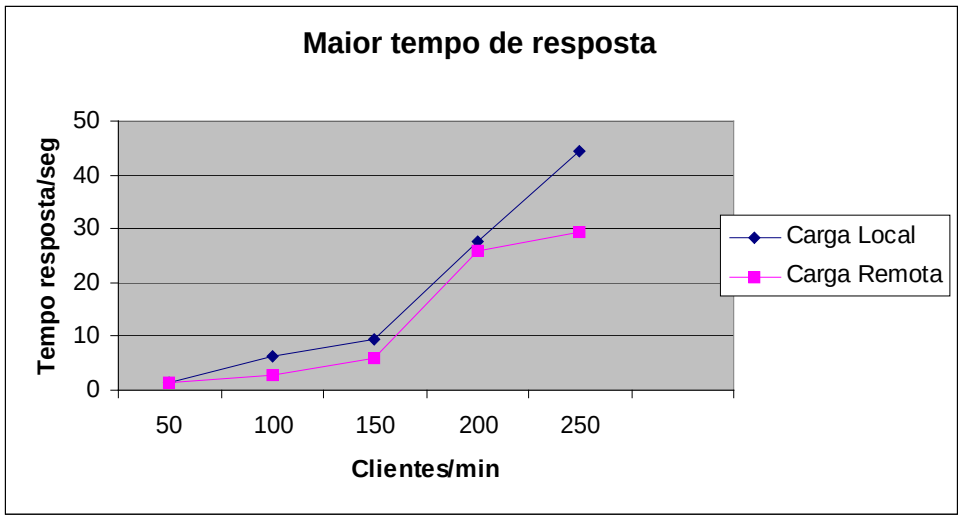

Figura 23: Gráfico comparativo do maior tempo de resposta de requisição

Na simulação, conforme aumenta o número de clientes simultâneos o servidor demora mais tempo para responder, chegando a demorar mais de 40 segundos para atender a uma requisição. O desempenho de Cluster Web apresentou um maior tempo de resposta de 29,5 segundos para 250 clientes simultâneos.

4.3 Conclusões e considerações sobre cluster com FreeBSD

Na confrontação dos dados gerados a partir das duas simulações o cluster FreeBSD para servidor de HTTP, mostrou um desempenho favorável, atendendo a um número maior de requisições por minuto, apresentou um ganho na taxa de transferência de dados e vazão (*throughput*), além de manter o sistema estável em situações criticas, quando o número de conexões simultâneas ultrapassa 200 clientes.

Na análise estatística apresentada pela interpretação da biblioteca Libpcap a ativação do software de balanceamento de carga *Pen*, ocorreu em tempo de execução, o que indica que qualquer diretiva de redirecionamento de Endereço IP, encaminhada ao Sistema Operacional, mais um algoritmo de balanceamento de carga permitira o redirecionamento das requisições para os Servidores Secundários e a distribuição do processamento.

5 CONCLUSÕES E TRABALHOS FUTUROS

A principal finalidade deste trabalho é o estudo das características de carga de trabalho em um servidor *web*. A motivação surgiu da constatação da complexidade de implementação de uma solução baseada em *cluster*. Apesar das limitações na arquitetura de *hardware* disponível, foi possível constatar resultados favoráveis nas simulações.

A partir do estudo de técnicas e modelos usados no planejamento de capacidade, conclui-se duas abordagens a serem analisadas: a primeira descreve que a capacidade do *hardware* influi diretamente na performance do modelo e a segunda que a distribuição do processamento cria alternativas para ampliar essa capacidade.

Os principais resultados encontrados descrevem um aumento na capacidade de atendimento de requisições, passando de mil e quatrocentos para algo em torno de duas mil transações por minuto.

Em outra análise, mesmo aumentando o número de clientes simultâneos o cluster *web*, além de uma melhor performance apresentou maior estabilidade, principalmente quando o número de clientes simultâneos ultrapassa duzentos. Na figura 18, que trata das transações por minuto, na figura 19 que trata a taxa de transferência de dados e na figura 21, que trata as transações por segundo, é possível notar que o servidor com carga local apresenta uma queda de performance a partir de 200 clientes simultâneos, como interpretação, o servidor utiliza um maior tempo de processamento para controlar as transações concorrentes, degradando o processo de atendimento das requisições. Enquanto o servidor cluster web, além de melhor performance apresentou uma estabilidade na execução dos processos, indicando que o modelo também apresenta distribuição nos processos de controle.

Outro resultado relevante foi o aumento na taxa de transmissão de dados, onde a distribuição das requisições provocou o aumento na vazão (*throughput*) dos dados.

Complementando os resultados anteriores, independente do tipo de serviço *web*, é possível, com o servidor cluster *web* obter uma melhor performance transacional.

Na figura 22 que trata as transações concorrentes, os dois modelos de servidores apresentaram um aumento gradativo de transações concorrentes, conseqüentemente um maior tempo médio de resposta às requisições, indicando que há um gargalo ou limite para a expansão do sistema, onde mesmo com um grande número de servidores o modelo poderá degradar ou mesmo entrar em colapso, demandando um grande volume de processamento de controle.

Completando a análise a figura 23 demonstra a demanda de controle sobre as transações concorrentes, indicando que quanto maior o número de transações concorrentes maior a demanda de tempo para atender as requisições.

Neste trabalho foram tratados apenas os componentes que afetam os serviços web em nível de atendimento de requisições do servidor de HTTP, para trabalhos futuros a sugestão é que sejam incorporados componentes ao nível de aplicação e no nível de banco de dados, fechando todo o modelo para atendimento de arquitetura de comercio eletrônico na Internet. No tratamento da capacidade de controle das transações concorrentes trabalhos correlatos sobre *Kernel* Distribuído e Máquina Virtual Distribuída, se apresentam como sugestão para a solução de gargalos na distribuição de processos.

A utilização dos recursos do Sistema Operacional por meio da reconfiguração do Kernel, para atender e configurar o Cluster Web permitiu a implementação de um sistema de processamento distribuído das requisições de modo dinâmico, configurável em tempo de execução, o que possibilita a ampliação da capacidade de processamento do Servidor Cluster Web em função da alteração da demanda por serviços.

6. REFERÊNCIAS.

ALLAIRE **ColdFusion 4.5 Web Application Server. ColdFusion Documentation - Administering Cold Fusion Server - Chapter 6, Creating Scalable and Highly Available Web Sites.** 2000. disponível em:
www.allaire.com/handlers/index.cfm?Method=Full&ID=14567 , Acesso em: janeiro de 2005.

BAKER, Scott M. and MOON, Bongki. **Scalable Web Server Design for Distributed Data Management**. 1998.University of Arizona. Disponivel em http://www.cs.arizona.edu/research/reports.html. Acesso em: janeiro de 2005

BANGA, G., DRUSCHEL, P. and MOGUL, J. C. **Measuring the Capacity of a Web Server. Proceedings of USENIX Symposium on Internet Technologies and Systems**, 1997. Disponivel em: http://www.cs.rice.edu/CS/Systems/Webmeasurement/paper/paper.html. Acesso em: janeiro de 2005

BERNERS-LE, T, et al, ; **The Word Wild Web**. Comm ACM, vol 37, nº 8, 1994.

CROVELLA M., BESTRAVOS, A.; **Self-Similarity in World Wild Web Traffic: Evidence and Possible Causes**. IEEE/ACM Transactions on Networking, p. ?, 1997.

FREEBSD**. Handbook**. Disponivel em: http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html. Acesso em janeiro de 2006.

HENNESSY, J., **The Future of System Researh**. 1999. Computer.

KRISHNASWAMY, U., SCHERSON, I.; **A Framework for Computer Performance Evaluation Using Benchmark Sets**. IEEE Trans. Computers, vol. 49, 2000.

MEIRA Jr., W., MENASCÉ, D., ALMEIDA, V., and FONSECA, R. . **E-representative: a scalability scheme for e-commerce**. In Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS'00), 2000.

MENASCÉ, Daniel A., ALMEIDA, Virgilio A.F.. **Planejamento de capacidade para serviços da web.** Rio de Janeiro. Editora Campus. 2002.

MENASCÉ, D., ALMEIDA, V., RIEDI, R., PELIGRINELLI, F., FONSECA, R., and MEIRA Jr., W. **A hierarchical and multiscale approach to analyze e-business workloads**, Performance Evaluation, 2003.

MILLER, Roger Leroy. **Microeconomia: teoria, questões e aplicações**. São Paulo. Editora McGraw-Hill do Brasil. 1981.

MINDICRAFT. **Webstone, The Benchmark for Web Server**. Disponivel em : http://www.mindcraft.com/webstone/. Acesso em: janeiro de 2006.

MOGUL, Jeffrey C. **The Case for Persistent-Connection HTTP**. WRL Research Report 95. Western Research Laboratory. 250 University Avenue. Palo Alto, California 94301 USA. 1995.

OLIVEIRA, Daniel Bristol. **Balanceamento de carga e redundância para Servidores Web.** Disponivel em: http://www.fug.com.br/index2.php?option=com_content&do_pdf=1&id=108. Acesso em agosto 2006.

OLSTON, Christopher, et.al. **A Scalability Service for Dynamic Web Appliccations**. CIDR 2005. Diponivel em: http:/www2.cs.cmu.edu/~olston/publications/scalability-service.pdf. Acesso em dezembro de 2005.

PEN. **Pen Load Balancer**. Disponivel em: http://siag.nu/pen/. Acesso em: março 2006.

PETERSON, Alec & FREEDMAN, **Avi. Dynamic Selection of Geographically Distributed Servers**. 1997. Disponivel em: http://www.academ.com/nanog/october1997/hilander/sld001.htm. Acesso em dezembro 2005.

PRESCOTT, William H. e SILVERMAN, Stanley**. EHZDNS: Web Server Load Sharing and Failure Protection**. 1999. U.S. Geological Survey, IEEE. Disponivel em : Internet Computing online. http://computer.org/internet/v3n6/ehzdns.htm. Acesso em: agosto 2005.

SIEGE**. HTTP Regression Testing and Benchmarking Utility**. Disponivel em : http://www.joedog.org/Siege/ReadMe. Acesso em: março 2006.

SPEC. **SPECweb2005 – Standard Performance Evaluation Corporation**. Disponível em: http://www.spec.org/web2005/. Acesso em agosto 2005.

TAVARES, T; et. al. **Load Balancing on Stateful Clustered Web Servers**. SBAC 2003. Disponivel em: www.dcc.ufmg.br/~dorgival/artigos/sbac03.pdf. Acesso em: dezembro de 2005

TPC. **Transaction Processing Performance Council – TPCW**. Disponível em: http://www.tpc.org/tpcw/default.asp. Acesso em: agosto 2005.

VISWANATHAN, Vivek. **Load Balancing Web Applications**. The O´Reilly Network. 2001. Disponível em: www.oreillynet.com/pub/a/a/onjava/2001/09/26/load.html. Acesso em: janeiro de 2006.

APENDICE A

Configuração do Kernel para redirecionamento dos endereços IP.

Abaixo comandos e opções que devem ser implementados no kernel FreeBSD para ativação do NAT (Network Adress Translactor).

Primeiramente é necessário máquina com *FreeBSD* instalado com os pacotes de Kern-Developer, para encontrar os fontes do kernel para recompilá-lo.

Com os fontes do kernel, login como root e caminho até /usr/src/sys/i386/conf:

$ cd /usr/src/sys/i386/conf

$ cp GENERIC SERVER

Agora o fonte GENERIC do FreeBSD foi copiado para Server e nessa copia do fonte é editada as alterações das opções:

Abaixo os comandos para compilação do kernel para a ativação do NAT.

$ config SERVER

O comando irá retornar uma mensagem parecida com esta:

Don't forget to do a ``make depend''

Kernel build directory is ../../compile/SERVER

Instruindo: não esqueça do make depend e a configuração do kernel está em ../compile/SERVER.

$ cd ../compile/SERVER

$ make depend

$ make

$ make install

APENDICE B

Comandos para instalação do balanceador Pen.

Instalando o Pen

O Pen está disponível para o FreeBSD via o Ports. O caminho para o ports é /usr/ports/net/pen.

Para instalar:

 # cd /usr/ports/net/pen

 # make install clean

Comando para utilização e configuração do balanceamento de carga do Pen.

Utilizando o Pen

A sintaxe básica do Pen é

 # pen EndereçoLocal:PortaLocal

EndereçoServidor1:Porta:LimiteDeConexões

EndereçoServidorn:Porta:LimiteDeConexões

pen 200.183.155.231:80 10.0.30.1 10.0.30.2 10.0.30.3

APENDICE C

Resultados secundários obtidos:

Resultados das cargas aplicadas variando o numero de clientes simultâneos:

Simulação de carga em servidor individual com cinqüenta clientes simultâneos:

comando -> siege –c50 -t1M localhost	
clientes	50
transactions (hits)	1455
availability (%)	100%
Elapsed time (seg)	64,35
Data transferred (MB)	2,14
Response time (seg)	0,545
transaction rate(trans/seg)	24,25
throughput (MB/seg)	0,04
concurrency	22,83
Successful transactions	1480
Failed transactions	0
longest transaction	1,36
Shortest transaction	0,042

Simulação de carga em servidor individual com cem clientes simultâneos:

comando -> siege –c100 -t1M localhost	
clientes	100
transactions (hits)	1346
availability (%)	100%
Elapsed time (seg)	62,56
Data transferred (MB)	2,1
Response time (seg)	1,765
transaction rate(trans/seg)	22,22
throughput (MB/seg)	0,03
concurrency	40,97
Successful transactions	1391
Failed transactions	0
longest transaction	6,445
Shortest transaction	0,04

Simulação de carga em servidor individual com cento e cinqüenta clientes simultâneos:

comando -> siege –c150 -t1M localhost	
clientes	150
transactions (hits)	1327
availability (%)	100%
Elapsed time (seg)	60,48
Data transferred (MB)	2,1
Response time (seg)	3,16
transaction rate(trans/seg)	21,94
throughput (MB/seg)	0,03
concurrency	69,04
Successful transactions	1346
Failed transactions	0
longest transaction	9,57
Shortest transaction	0,03

Simulação de carga em servidor individual com duzentos clientes simultâneos:

comando -> siege –c200 -t1M localhost	
clientes	200
transactions (hits)	1372
availability (%)	99%
Elapsed time (seg)	60,52
Data transferred (MB)	2
Response time (seg)	6,74
transaction rate(trans/seg)	22,67
throughput (MB/seg)	0,03
concurrency	152,75
Successful transactions	1380
Failed transactions	9
longest transaction	27,51
Shortest transaction	0,04

Simulação de carga em servidor individual com duzentos e cinqüenta clientes simultâneos:

comando -> siege –c250 -t1M localhost	
clientes	250
transactions (hits)	1033
availability (%)	100%
Elapsed time (seg)	64,66
Data transferred (MB)	1,59
Response time (seg)	10,54
transaction rate(trans/seg)	16,14
throughput (MB/seg)	0,025
concurrency	158,73
Successful transactions	1037
Failed transactions	0
longest transaction	44,44
Shortest transaction	0,04

Simulações com carga remota no servidor cluster web com cinqüenta clientes simultâneos

comando -> siege –c50 -t1M 200.183.155.231	
clientes	50
transactions (hits)	2284
availability (%)	100%
Elapsed time (seg)	60,77
Data transferred (MB)	3,3
Response time (seg)	0,59
transaction rate(trans/seg)	37,58
throughput (MB/seg)	0,05
concurrency	22,33
Successful transactions	2284
Failed transactions	0
longest transaction	1,45
Shortest transaction	0,14

Simulações com carga remota no servidor cluster web com cem clientes simultâneos

comando -> siege –c100 -t1M 200.183.155.231	
clientes	100
transactions (hits)	2201
availability (%)	100%
Elapsed time (seg)	60,33
Data transferred (MB)	3,18
Response time (seg)	1,9
transaction rate(trans/seg)	36,48
throughput (MB/seg)	0,05
concurrency	69,43
Successful transactions	2201
Failed transactions	0
longest transaction	2,73
Shortest transaction	0,23

Simulações com carga remota no servidor cluster web com cinqüenta clientes simultâneos

comando -> siege –c150 -t1M 200.183.155.231	
clientes	150
transactions (hits)	2221
availability (%)	100%
Elapsed time (seg)	60,91
Data transferred (MB)	3,21
Response time (seg)	3,27
transaction rate(trans/seg)	36,47
throughput (MB/seg)	0,05
concurrency	119,16
Successful transactions	2221
Failed transactions	0
longest transaction	5,95
Shortest transaction	0,34

Simulações com carga remota no servidor cluster web com duzentos clientes simultâneos

comando -> siege –c200 -t1M 200.183.155.231	
clientes	200
transactions (hits)	2207
availability (%)	100%
Elapsed time (seg)	60,69
Data transferred (MB)	3,19
Response time (seg)	4,31
transaction rate(trans/seg)	36,37
throughput (MB/seg)	0,05
concurrency	156,58
Successful transactions	2207
Failed transactions	5
longest transaction	25,92
Shortest transaction	0,24

Simulações com carga remota no servidor cluster web com duzentos e cinqüenta clientes simultâneos

comando -> siege –c250 -t1M 200.183.155.233	
clientes	250
transactions (hits)	2170
availability (%)	98%
Elapsed time (seg)	60,25
Data transferred (MB)	3,15
Response time (seg)	4,63
transaction rate(trans/seg)	36,02
throughput (MB/seg)	0,05
concurrency	166,77
Successful transactions	2175
Failed transactions	35
longest transaction	29,5
Shortest transaction	0,08