Uso de Beowulf’s como servidores Web

Roberto Ferreira Brandão, Ricardo de Oliveira Anido

Instituto de Computação, UNICAMP
Rua Albert Einstein, 1251, Campinas, São Paulo, Brasil
{brandao, ranido}@ic.unicamp.br

Resumo

O aumento do uso dos serviços oferecidos pela Internet faz com que a carga de trabalho de servidores populares seja muito aumentada, o que exige a instalação de servidores com alto poder de processamento.

Computadores com arquitetura Beowulf consistem em cpu’s ligadas por uma rede de comunicação. Essa arquitetura possui a vantagem de oferecer a possibilidade da construção de um computador com alto poder de processamento a baixo custo. Além disso, facilita a implementação de servidores escaláveis e tolerantes a falhas.

Esse artigo apresenta um estudo sobre a possibilidade da utilização de um Beowulf como servidor Web. São discutidos aspectos como desempenho, escalabilidade e tolerância a falhas. Para fins de análise de desempenho e estudo de parâmetros relativos ao fluxo das requisições, são apresentados resultados de simulações da utilização de Beowulf’s como servidores Web.

Abstract

Due to the increasing use of Internet services, popular web servers receive a high request for file rates, which demands the instalation of high processing power computers as web servers.

The Beowulf architecture consists of clusters of off the shelf cpu's connected by a network. They have the advantage of offering the possibility of building at low cost, a high processing power computer. The architecture makes it more easy to implement scalable and fault tolerant servers.

This paper presents a study about the possibility of using a beowulf computer as web server. Aspects like performance, scalability and fault tolerancy are discussed. Beowulf's simulations results are presented to show performance and scalability capacities and to analyze some requisition's flow characteristics.

Palavras-chave

Avaliação de desempenho, Servidor Web, Beowulf, Cache para Web.

Keywords

Performance evaluation, web server, Beowulf, web cache.

1. Introdução

Com o aumento da popularidade dos serviços oferecidos pela Internet, muitos estudos têm sido realizados com o objetivo de aumentar o desempenho dos sistemas de transmissão de documentos. Apesar de muitos trabalhos terem se concentrado principalmente no lado cliente desse sistema, através da elaboração de estratégias de posicionamento de caches, a capacidade do servidor de documentos é um elemento fundamental no desempenho do sistema de transmissão de documentos.

Apesar de um alto percentual de documentos requisitados a um servidor Web ainda consistir de documentos estáticos, tornam-se cada vez mais comuns as páginas geradas no momento do processamento das requisições, contendo informações muitas vezes tiradas de bancos de dados no momento da geração da página. Essa nova funcionalidade dos servidores Web exige dos servidores uma capacidade de processamento cada vez maior, fazendo-se necessária a distribuição de processamento de requisições entre servidores cooperativos.

Para o processamento paralelo de requisições feitas a endereços Internet muito populares, podem ser utilizados clusters de processadores. Os nós formadores do cluster dividem a carga de requisições entre si, aumentando a taxa de requisições respondidas pelo sistema. Uma arquitetura de computador baseada em cluster que vem destacando-se entre as demais por seu baixo preço e pela ausência do limite técnico para a quantidade máxima de PC's que podem ser conectados é a arquitetura Beowulf [1].

Entretanto, a utilização de um Beowulf como servidor Web apresenta problemas de distribuição e balanceamento da carga de requisições entre os nós, bem como a distribuição das tarefas a serem executadas por cada nó e dos documentos as serem armazenados.

Esse trabalho tem como objetivos analisar a utilização de computadores baseados na arquitetura Beowulf e propor técnicas de solução dos problemas mais comins encontrados quando da tentativa de usar computadores dessa arquitetura como servidores Web. Para isso, é proposto um modelo que descreve o servidor, sendo também apresentados resultados de simulações de desempenho, baseadas no modelo proposto.

Na seção 2, serão descritas brevemente as principais publicações correlatas a esse estudo, até o momento. A seção 3 descreve resumidamente as principais características da arquitetura Beowulf. Na sessão 4 são descritos os elementos formadores do sistema enquanto que a seção 5 descreve o seu funcionamento. A seção 6 apresenta e discute características do modelo proposto enquanto que a seção 7 apresenta resultados de simulações desse meodelo.

2. Trabalhos relacionados

As principais informações referentes à arquitetura Beowulf podem ser encontradas em [1].

Os principais trabalhos relativos a utilização de clusters e servidores distribuídos, principalmente no aspecto de escalabilidade e tolerância a falhas estão em [2], [3], [4], [5] e [6].

Técnicas de balanceamento de carga de trabalho em servidores distribuídos são descritas em [7], [8] e [9], enquanto que particularidades relativas ao fluxo de requisições são analisadas em [10], [11].e [12].

Características relativas a caches para documentos Web são estudadas em, [13], [14], [15], [16], [17], [18], [19], [20] e [21].

3. A arquitetura Beowulf

Computadores com a arquitetura Beowulf consistem em cpu's ligadas por uma rede de comunicação. Na classificação de compuatores paralelos, a arquitetura Beowulf se encaixa entre os Processadores Massivamente Paralelos (MPP) como o nCube e o Cray e os NOW's (Networks of Workstations).

Computadores com arquitetura MPP são geralmente maiores e bem mais caros. Porém, apresentam tempos menores de troca de informações entre processadores que os computadores baseados na arquitetura Beowulf. Utilizar uma NOW consiste apenas em aproveitar a capacidade de processamento ociosa de computadores potencialmente diferentes interligados por rede, apresentando o problema de necessitar de algoritmos eficientes em balanceamento de carga. Qualquer programa que rode em uma NOW irá rodar, no mínimo, tão bem em um Beowulf [1]

O primeiro computador com arquitetura Beowulf foi montado por Thomas Sterling e Don Becker em 1994. Desde então, com a redução dos preços dos componentes de computadores e o aumento da capacidade de processamento dos PC's, a arquitetura Beowulf tem se tornado uma boa opção para se conseguir alta capacidade de processamento a baixo custo. Apesar da proposta inicial da arquitetura Beowulf ser direcionada principalmente aos meios acadêmico e científico, a sua utilização em companhias que necessitam de solução envolvendo grande capacidade de processamento tem se tornado cada vez maior.

4. Modelo utilizado

A figura 4.1 apresenta o modelo utilizado na análise do desempenho do servidor Web. A direção das setas indica o fluxo de informações enviadas. Note que não estão representadas na figura 4.1 as interconexões entre os nós de processamento e os caches.

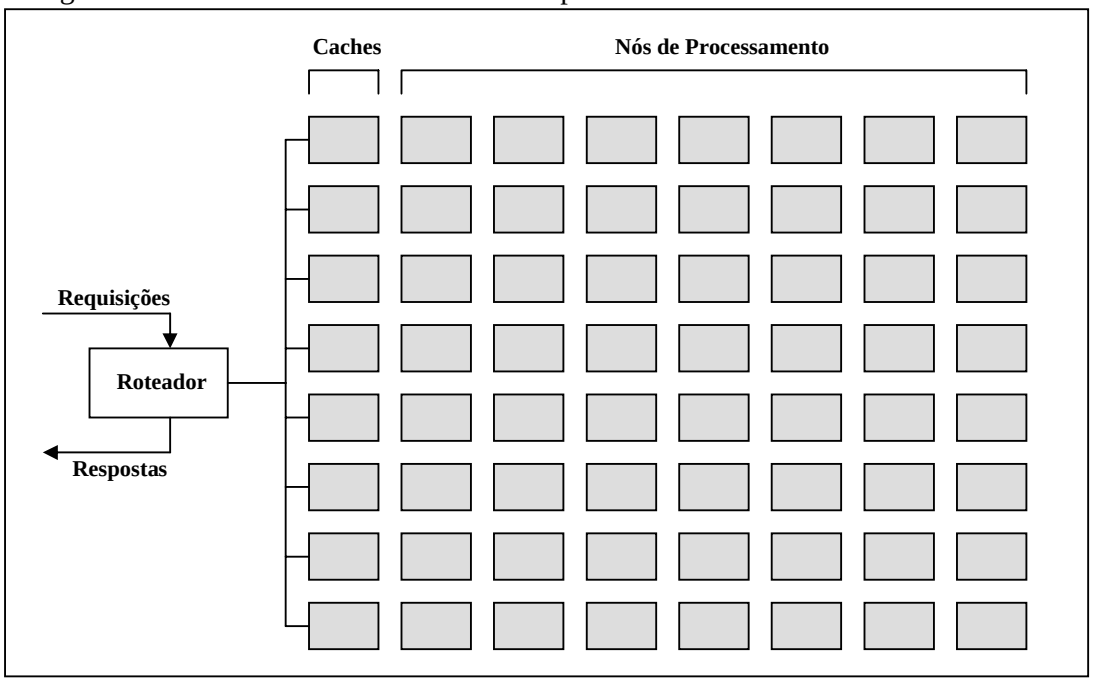

Fig. 4.1 - Modelo de servidor utilizado

Os principais elementos nesse modelo são o roteador, os caches e o conjunto de nós de processamento. Cada um desses elementos será discutido a seguir.

4.1. O roteador

O roteador tem a função de receber as requisições dos clientes e enviá-las para os elementos do sistema que vão tratá-las: os caches. Existem duas implementações possíveis para o elemento roteador: a utilização de um *gateway* ou um *switch*. Como os endereços IP dos nós de processamento e dos caches são endereços de rede privada, a única maneira de acessá-los é através do roteador. Assim, o roteador deve ser capaz de responder a requisições HTTP feitas por clientes. A não ser que seja instalado um *switch* capaz de trabalhar como um Proxy, respondendo a requisições HTTP, deve ser usado um computador como roteador para receber essas requisições e repassá-las aos caches.

O *switch* tem a vantagem de trabalhar em altas velocidades podendo, muitas vezes, gerenciar várias conexões de rede. O problema em se usar um *switch* é que ele deve ser capaz de trabalhar como Proxy, fazendo com que esse elemento exija um maior investimento.

Usar um computador dedicado como roteador tem a vantagem de permitir um melhor direcionamento das requisições, visto que podem ser implementadas políticas mais inteligentes de distribuição de requisições. Porém, a replicação de um roteador implementado em um computador dedicado é mais difícil que em um roteador que utiliza um *switch*.

Além dos problemas de tolerância a falhas, o roteador torna-se um gargalo quando o número de requisições se torna muito alto. Isso ocorre porque cada requisição tem que ser processada pelo *gateway*, fazendo com que seja prejudicada a escalabilidade do número de requisições e processadores do cluster.

O problema da escalabilidade é diminuído quando usado um *switch* como roteador, pois o *switch* pode ser projetado de forma a ser otimizado para aumentar a taxa de requisições processadas e a velocidade de processamento de requisições.

4.2. Caches

Os caches do sistema são implementados usando os nós do Beowulf. Nesse trabalho, considera-se que cada nó de processamento possui capacidades individuais de armazenamento em disco. Essa é uma das principais diferenças entre esse modelo e os modelos de servidores Web distribuídos, onde os caches são instalados fora da malha de servidores.

Os caches são colocados para economizar requisições feitas aos nós de processamento. Apesar de existirem técnicas para o *caching* de documentos dinâmicos, os caches são utilizados principalmente com documentos estáticos. Nesse modelo, o espaço em disco dos caches é usado para armazenar documentos estáticos requisitados recentemente aos nós de processamento.

No caso do servidor Web enviar muitos documentos estáticos ou raramente atualizados, a utilização de caches é recomendável. Por outro lado, se a maioria do tráfego corresponde a páginas geradas quando do processamento da requisição, a economia de requisições feita aos servidores não compensa o atraso causado pela inserção de caches no fluxo de dados.

4.3. Nós de processamento

Os nós de processamento são os processadores do Beowulf responsáveis pelo armazenamento de arquivos e geração de páginas dinâmicas.

Nesse trabalho, considera-se que cada nó de processamento é responsável por um subconjunto dos arquivos armazenados no servidor. Uma maneira de decidir quais processadores são responsáveis por quais arquivos é calcular, usando uma função de *hash*, um número natural relativo a cada arquivo no servidor. Pode-se então usar esse número para decidir o número do nó de processamento responsável por aquele documento.

Os nós de processamento são também responsáveis pela geração de páginas dinâmicas, que são geradas quando do processamento das requisições.

Para gerar a página, o nó de processamento poderá utilizar-se de estruturas de páginas armazenadas em disco, dados tais como a data e hora atuais, parâmetros passados na pelo cliente através da requisição e dados armazenados em bancos de dados, locais ou não. Caso sejam utilizados bancos de dados distribuídos, estudos com relação à arquitetura e desempenho dos mesmos devem ser realizados. Em [24], é possível encontrar um bom texto introdutório sobre o assunto.

Não existem restrições quanto ao software a ser usado para processar as requisições e enviar os documentos de resposta. Pode ser usado tanto um software de código aberto, tal como o Apache [25] quanto qualquer outro software capaz de processar requisições e, se for o caso, gerar páginas dinâmicas. Essa independência com relação ao software utilizado ocorre porque, do ponto de vista do servidor, as requisições serão recebidas como se ele estivesse se comunicando diretamente com os clientes, sendo transparente a presença do roteador e dos caches.

5. Funcionamento do Sistema

O servidor pode trabalhar sob duas condições: funcionamento normal e reconfiguração do Beowulf. Cada uma dessas condições define um estado de funcionamento do Beowulf.

5.1. Funcionamento sob condições normais

É considerado que o sistema está trabalhando em condições normais quando está no estado de funcionamento normal. Esse é o estado inicial do Beowulf e o estado no qual espera-se que ele trabalhe na maior parte do tempo.

Nesse estado, requisições de clientes são endereçadas ao roteador. O roteador passa as requisições para um dos caches. Caso o documento seja estático e esteja armazenado no cache, ele é enviado ao roteador que o repassa ao cliente.

Caso a requisição seja por um documento dinâmico, o cache repassa a requisição ao nó de processamento responsável pelo documento requisitado. Para saber qual é o nó de processamento responsável por cada arquivo requisitado, o cache deverá também utilizar a função de *hash* usada para distribuir os arquivos dentre os nós de processamento.

Caso a requisição seja por um documento estático não encontrado no cache, este fará uma requisição ao nó de processamento responsável pelo documento. O nó de processamento verifica se possui o documento ou não. Caso possua, envia-o para o cache que armazena uma cópia para si e envia uma cópia ao roteador, que a repassa ao cliente.

Quando do armazenamento do documento no cache, este verificará se existe espaço disponível. Em caso negativo, o cache terá que aplicar uma política de substituição para

remover os documentos considerados com menor utilidade. Essa política de substituição pode ser escolhida de acordo com o perfil dos documentos armazenados nos nós de processamento. Por exemplo, caso existam documentos estáticos muito acessados , pode-se usar a política LFU, que retira do cache os documentos menos acessados, aumentando a chance de que documentos populares sejam encontrados no cache.

A escolha do cache pode ser feita usando um algoritmo simples, por exemplo, *round-robin* ou aleatório. Isso pode ser feito porque as requisições seguem uma distribuição Zipf [10], o que indica que os documentos mais acessados são muito mais acessados que os documentos menos acessados. Por isso, depois de um certo tempo de funcionamento, cada cache terá uma cópia dos documentos mais acessados, o que aumentará a disponibilidade de documentos populares, diminuindo o tempo de resposta do servidor.

5.2. Reconfiguração do Beowulf

Quando existe uma alteração no número de nós de processamento do sistema, devido ao acréscimo de nós para proporcionar um melhor desempenho ou decréscimo de nós por falha ou outro motivo, o Beowulf passa a operar no estado de reconfiguração.

O processo de reconfiguração deve ser iniciado manualmente pelo administrador do sistema ou automaticamente, por exemplo, quando detectada uma falha em um nó de processamento, no caso de não existirem nós de reserva. Os nós de reserva são descritos na seção 6.2.3.

Quando é iniciada a reconfiguração do sistema, são informados aos caches a alteração de estados, o novo número de nós e os endereços IP dos nós envolvidos no processo. Nesse momento, é iniciado um processo responsável por mover todos os arquivos para as suas novas posições. Esse processo pode ser configurado para utilizar apenas a capacidade de processamento e a banda passante ociosas nos nós de processamento.

Para informar aos caches da mudança de estado, deve ser implementado um algoritmo de difusão confiável da mensagem de troca de estado. Além disso, o processo responsável por mover os arquivos durante a reconfiguração deve ser implementado considerando as possibilidades de falha nesse processo.

No estado de reconfiguração, os únicos elementos que passam a agir de forma diferente são os caches. Considerando que antes do início da reconfiguração existiam **N** nós de processamento e que o sistema está sendo remodelado para utilizar **M** nós, sendo

hash(A) mod N - o índice do nó de processamento responsável pelo arquivo **A** antes da reconfiguração e

hash(A) mod M - o índice do nó de processamento responsável pelo arquivo depois de encerrada a reconfiguração, temos o algoritmo.

```
Algoritmo procura_durante_reconfiguração
Início
   Se Copia ( hash(A) mod N, A) então
      Envia A para roteador
   Senão
      Se Copia ( hash(A) mod M, A) então
         Envia A para o roteador
      Senão
         Esse arquivo não existe no servidor
      Fim Se
   Fim Se
```

Fig. 5.1 – Algoritmo de procura durante processo de reconfiguração.

A função *Copia* procura um arquivo em um nó de processamento identificado por um número. Obviamente, é necessário que o cache mantenha uma tabela relacionando o índice do nó de processamento e seu endereço IP. Se o arquivo for encontrado, ele é copiado para o cache.

Durante a reconfiguração, os arquivos deverão ser movidos entre os nós de processamento para a posição correta dentro do Beowulf, considerando o novo número de nós. O algoritmo *Procura_durante_reconfiguração* procura primeiro o arquivo na sua posição antes da reconfiguração. Ele será encontrado caso ainda não tenha sido movido. Em caso contrário, o arquivo, se existir no servidor, já terá sido movido para a nova posição, onde será encontrado.

Depois que todos os arquivos já estiverem em suas respectivas posições finais, são enviadas mensagens aos caches avisando do fim da reconfiguração. Os caches, por sua vez, mudam o estado para o de operação normal.

6. Discussão sobre características do sistema

Os critérios de avaliação do sistema considerados nesse trabalho serão a escalabilidade, o nível de tolerância a falhas e o desempenho do sistema em operação normal e durante o processo de reconfiguração. Cada um desses aspectos é discutido nas seções a seguir.

6.1. Escalabilidade

A escalabilidade de um sistema formado por múltiplos processadores é a medida da capacidade desse sistema em aumentar a sua capacidade de processamento (diminuir o tempo de resposta, no caso de um servidor Web), proporcionalmente ao aumento do número de processadores [22]. A escalabilidade reflete a habilidade de um sistema em usar efetivamente um eventual ganho de recursos.

Um sistema é dito escalável se é possível manter o desempenho desse sistema, perante o aumento da carga de trabalho com um aumento proporcional do número de processadores.

No caso do sistema descrito nesse trabalho, o grande problema em relação a escalabilidade é o roteador. Isso acontece porque um único elemento deve ser capaz fazer toda a intercomunicação entre os clientes e o servidor. Porém, pode-se pensar no roteador como sendo formado de vários elementos que dividem a carga de requisições entre si. Essa divisão pode ser feita, por exemplo, através da atribuição de múltiplos endereços IP associados a um único nome. Nesse caso, a divisão da carga seria feita por servidores de DNS.

Considerando que os nós do Beowulf são interligados por *switchies*, o que favorece a comunicação ponto a ponto, tanto os caches quanto os nós de processamento podem ser considerados escaláveis. Isso porque tanto o número de caches, visando o aumento da disponibilidade dos documentos populares, quanto o número de nós de processamento podem ser aumentados indefinidamente, desde que a estrutura de comunicação entre os nós seja também escalável, o que pode ser conseguido usando-se *switchies.*

6.2. Tolerância a falhas

Falhas são desvios do comportamento do sistema para situações indesejadas. Em servidores Web, falhas muitas vezes causam grandes prejuízos devido à interrupção ou degradação da qualidade dos serviços oferecidos.

Falhas no sistema são resultado de faltas em componentes de software ou hardware que formam o sistema. As faltas são divididas em três categorias [23]:

1 – Faltas transientes: são as faltas que se iniciam em um tempo específico, permanecem durante algum tempo e desaparecem. Um exemplo é a interferência eletromagnética em um dispositivo de comunicação.

2 – Faltas permanentes: são faltas que aparecem e persistem até serem reparadas. Por exemplo, a queima de um processador é uma falta permanente.

3 – Faltas intermitentes: são faltas transientes que se repetem, em intervalos fixos ou não. Um exemplo é a interrupção do funcionamento de um dispositivo de hardware devido a sobre-aquecimento. O dispositivo pára de funcionar, resfria e volta a funcionar novamente, parando novamente depois de um período, pelo mesmo motivo.

O nível de tolerância a falhas apresentado por um sistema computacional representa o nível de garantia de manutenção da qualidade de serviço oferecida pelo sistema na presença de falhas.

Nessa seção, serão discutidos resumidamente, aspectos de tolerância a falhas do servidor Web implementado em um computador com arquitetura Beowulf. Para isso, são discutidos os aspectos de tolerância a falhas individuais dos componentes que formam o servidor: o roteador, os caches e os nós de processamento.

6.2.1. O roteador

Esse é o elemento mais sensível a falhas dentre os que formam o servidor. Isso ocorre porque o roteador de requisições funciona como uma interface entre o servidor e os clientes e, caso ele pare de funcionar, os clientes não mais terão acesso aos serviços.

A melhor maneira de introduzir capacidade de tolerância a falhas nesse elemento é a colocação de um ou mais elementos redundantes a ele. Essa estratégia consiste na instalação de equipamentos que não são usados durante as operações normais do roteador. Eles checam a todo instante o funcionamento correto do roteador principal e, caso detectem uma falha deste, assumem a operação, evitando a interrupção do oferecimento dos serviços.

O equipamento redundante ficará responsável pela interface entre os clientes e o servidor até que o roteador principal volte a funcionar. Isso pode levar pouco tempo, no caso de faltas transientes ou até vários dias, no caso de faltas permanentes.

Essa abordagem leva a sistemas mais confiáveis. Porém, note que o sistema não é completamente tolerante a falhas, pois os equipamentos redundantes também podem falhar. A capacidade de tolerância a falhas pode então ser aumentada adicionando-se tantos elementos redundantes quantos se achar necessário para atingir um nível considerado satisfatório de tolerância a falhas. A estratégia de colocação de vários elementos redundantes é chamada Redundância N-Modular.

6.2.2. Os caches

Para que os caches apresentem uma alta tolerância a falhas, basta que o elemento roteador seja programado para que, caso um cache pare de responder, elimine-o da lista de caches disponíveis. No caso em que, por motivo de simplicidade e desempenho do roteador não existirem operações de gerenciamento da lista de caches, o roteador deve ser configurado para marcar um tempo de *timeout* após enviar um pedido de arquivo a um cache, depois do qual será escolhido um novo cache para enviar a requisição.

Dessa forma, a ocorrência de falha em um cache fará com que sua carga de trabalho seja redirecionada para os demais caches, o que aumentará o tempo médio de resposta dos caches, que terão que processar um número maior de requisições. Essa característica de diminuição gradual da capacidade de serviço é chamada de degradação gradual ou *fail soft.*

6.2.3. Os nós de processamento

Se considerarmos que a carga de trabalho submetida a um nó de processamento em falha pode ser dividida entre os outros nós de processamento, podemos considerar que o conjunto dos nós de processamento apresenta características de degradação gradual de desempenho perante falhas. Porém, é importante lembrar que cada nó de processamento é responsável por uma parte dos arquivos no servidor. Nesse caso, a estratégia de usar redundância parece mais uma vez ser a mais apropriada.

À primeira vista, pode-se pensar em colocar um nó de processamento redundante a cada nó de processamento em operação no Beowulf. Porém, uma estratégia melhor seria a de criar uma classe de nós de processamento cujos elementos possam vir a substituir qualquer nó em falha. Para isso, os elementos do conjunto de nós redundantes devem verificar constantemente o funcionamento dos nós de processamento. Além disso, uma cópia de todos os arquivos do servidor deve ser distribuída pelos nós redundantes.

Quando um nó de processamento redundante verifica que um nó em operação falhou, ele copia a partir dos outro nós redundantes todos os arquivos que devem ser servidos pelo nó falho. Inicia então o processo que trata as requisições e envia mensagens aos caches da substituição do nó responsável por aquela faixa de arquivos.

Caso não seja possível armazenar todos os arquivos nos nós redundantes, pode-se usar um repositório de arquivos. Esse repositório pode ser implementado em um computador que não faça parte do Beowulf, mas que possua uma alta capacidade de armazenamento de arquivos. Esse repositório pode ser usado como *backup* dos arquivos do servidor. O computador que implementa o repositório pode também ter um computador redundante.

6.3. Desempenho

Durante o estado de funcionamento normal, o tempo de resposta, **TR**, a uma requisição é dado por:

TR = TPR + TTRRC + TPC + TTRCN + TPN + TTANC + TTACR, sendo

TR: Tempo de resposta do sistema a uma requisição;
TPR: Tempo de processamento da requisição pelo roteador;
TTRRC: Tempo de transmissão da requisição do roteador para o cache;
TPC: Tempo de processamento da requisição pelo cache;
TTRCN: Tempo de transmissão da requisição do cache para o nó de processamento;
TPN: Tempo de processamento do nó de processamento;
TTANC: Tempo de transmissão do arquivo do nó de processamento para o cache;
TTACR: Tempo de transmissão do arquivo do cache para o roteador.

Essa expressão mostra como pode ser calculado o valor do **TR** médio esperado. Esse valor representa o tempo desde o término do recebimento da requisição pelo roteador até o início da transmissão do arquivo ao cliente.

No caso dos documentos estáticos, o valor de **TR** passa a sofrer um impacto muito menor dos valores de **TTRCN**, **TPN** e **TTANC**. Isso porque existe uma grande probabilidade

de uma cópia do arquivo já estar armazenada no cache, economizando o acesso ao nó de processamento.

Por motivo de comparação, é apresentada a seguir a expressão do cálculo de **TR** para um servidor convencional, composto por apenas um computador, multiprocessado ou não:

TR = TPR + TAA, sendo

TPR: Tempo de processamento da requisição;
TAA: Tempo de acesso ao arquivo.

Apesar de **TR** em um servidor convencional ser menor, sob alguns aspectos, usar um Beowulf como servidor apresenta algumas vantagens.

A primeira vantagem é a escalabilidade, visto que a capacidade do Beowulf pode ser expandida indefinidamente, enquanto que o servidor convencional está limitado ao melhor computador possível de ser adquirida com a tecnologia atual. Além disso, o Beowulf pode ter a sua capacidade aumentada com a compra de novos nós enquanto que fica difícil realizar um *upgrade* no servidor convencional sem substituir grande parte dos componentes de hardware.

A capacidade de tolerância a falhas é outra grande vantagem apresentada pelo Beowulf. Enquanto o servidor convencional precisa de um servidor redundante com capacidade compatível e potencialmente caro, a tolerância a falhas no Beowulf pode ser conseguida com a redundância de elementos mais baratos, sendo que os gastos em equipamentos redundantes será muito menor que os gastos em equipamentos efetivamente ativos.

Outra vantagem importante seria vista no caso do tempo de processamento das requisições pelo nó de processamento necessitar de um tempo de computação significativo. Nesse caso, a divisão da carga de trabalho entre os nós de processamento gera uma capacidade potencialmente infinita, o que não é possível de conseguir com um servidor composto de um único computador.

É importante ainda a inclusão de alguns comentários sobre os tempos de processamento e envio de requisições e arquivos, que influem no **TR** de um Beowulf.

Considerando que a rede que interliga os elementos do Beowulf seja de alta capacidade (Fast Ethernet, por exemplo) e que utilize um ou mais *switchies*, o que permite o paralelismo ponto a ponto real nas transmissões, o atraso devido ao *overhead* da intercomunicação dos elementos é pequeno quando comparado ao tempo de envio de arquivos do roteador ao cliente, o que normalmente é feito através de uma rede de longa distância, que possui um taxa de transmissão menor que a rede local que interliga os elementos do Beowulf.

Durante a reconfiguração, o **TR** pode sofrer um aumento devido ao aumento do tempo para a procura do arquivo. O impacto da reconfiguração no tempo de resposta é diminuído devido à utilização dos caches para enviar cópias de documentos estáticos populares aos clientes, economizando a requisição aos nós de processamento que, por sua vez, contam com mais recursos para executar o processo de reconfiguração. Quanto maior o percentual de documentos estáticos e maior a eficiência dos caches, menos requisições forçarão a realização de duas consultas aos nós de processamento durante a reconfiguração.

7. Simulações

Para analisar aspectos de desempenho e escalabilidade do modelo de servidor Web proposto nesse trabalho foi implementado um simulador de servidores Web implementados em Beowulf's. Esse simulador, cuja interface é apresentada na figura 7.1, permite simular

Bewulf's com qualquer número de nós de processamento e caches. Permite configurar opções tais como os tempos de procura por arquivos e transmissão de requisições bem como a capacidade da rede e qualidade de serviço esperada do Beowulf, através do tempo de *timeout* das requisições.

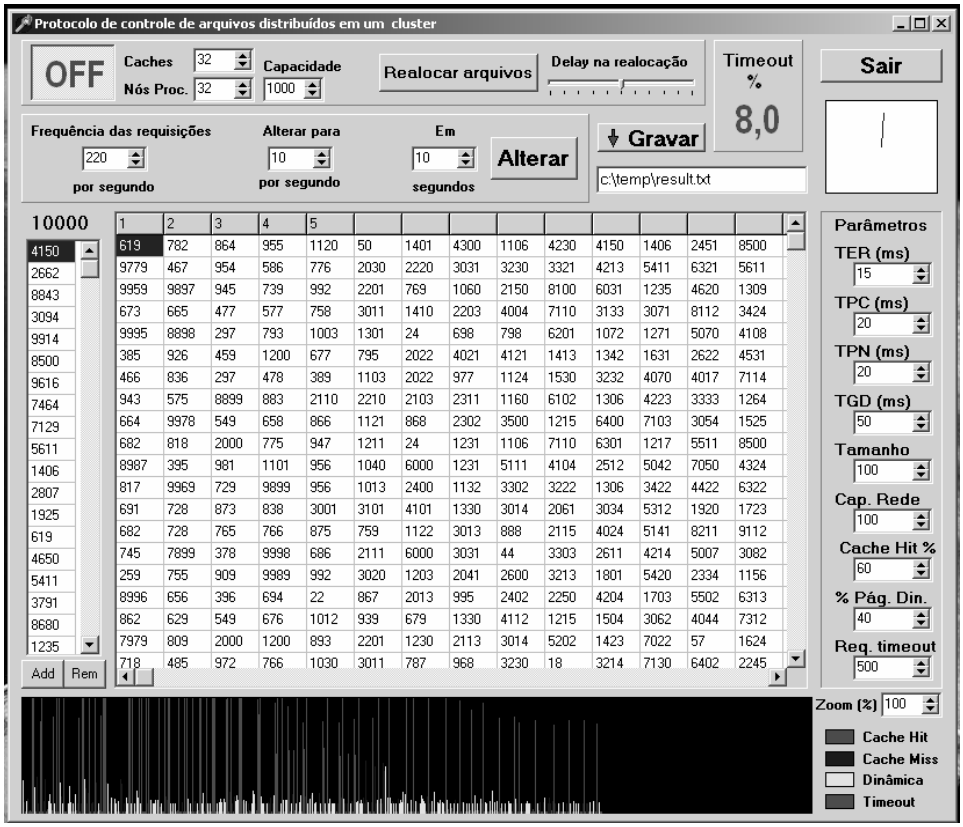

Fig. 7.1 – Interface do simulador

O simulador apresenta durante a simulação a possibilidade da variação da taxa de requisições, de forma a permitir verificar o impacto dessa variação nos tempos de resposta às requisições. Os tempos de resposta das requisições podem ser verificados na parte inferior do simulador. Além disso, dados relativos ao fluxo de requisições e tempo de resposta das mesmas podem ser gravados em um arquivo texto para posterior elaboração de gráficos e análise estatística dos dados, com uma ferramenta destinada a esse fim.

7.1. Premissas

A primeira premissa considerada nas simulações diz respeito à capacidade da rede que liga o roteador aos clientes. Nesse trabalho, consideramos essa rede de capacidade infinita. Isso significa que não existirá contenção na saída dos dados do servidor, o que garante que não existam barreiras para impedir que o fluxo de requisições leve o servidor a trabalhar no limite de sua capacidade.

Outra premissa diz respeito a características relativas ao fluxo de requisições. As características tomadas como *default* nas simulações são apresentadas na tabela da figura 7.2.

Portanto, a não ser que alterados numa simulação específica, os parâmetros que caracterizam o fluxo de execução são os apresentados na tabela.

Parâmetro	Valor
Tamanho dos arquivos (média)	100 Kbytes
Capacidade da rede de intercomunicação dentro do Beowulf	100 Mbps
Percentual de documentos dinâmicos	40 %
Tempo de procura médio dos servidores de cache	20 milissegundos
Desempenho dos servidores de cache (Hit Ratio)	60 %
Tempo de procura por arquivo dos cache e nós de processamento	20 milissegundos
Tempo de geração de página dinâmica	50 milissegundos

Fig. 7.2 – Valores usados na simulação da utilização do servidor

7.2. Simulações realizadas

As primeiras simulações realizadas destinaram-se a estudar a escalabilidade de um servidor Web implementado em um Beowulf. Para isso, definiu-se uma taxa de requisições alvo, nesse caso de 80 requisições por segundo, e iniciou-se a busca por uma configuração de Beowulf que conseguisse responder a essa taxa. O gráfico na figura 7.3 mostra os tempos de resposta para três configurações analisadas. Nessa simulação, foi considerada a existência de apenas 3 nós de processamento, além dos caches.

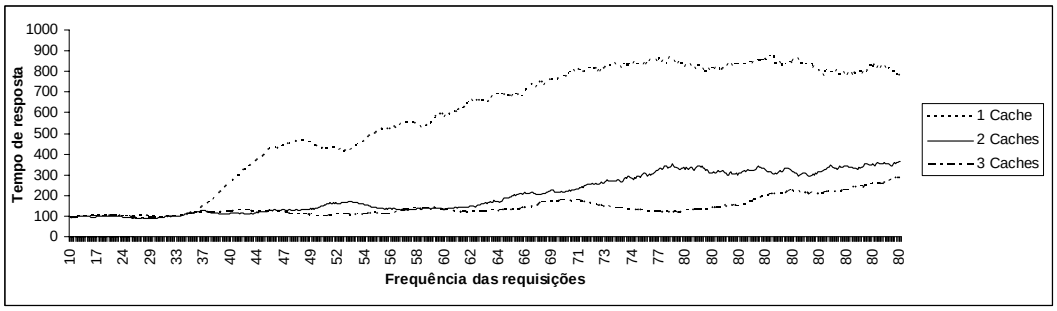

Fig. 7.3 – Tempos de resposta de um servidor composto de 3 nós de processamento

Observando o gráfico da figura 7.3, é possível notar que uma configuração possuindo apenas um cache não é suficiente para construir um servidor capaz de responder a uma taxa de 80 requisições por segundo. Apesar de a configuração com apenas 2 caches capacitar o Beowulf a responder a uma taxa de 80 requisições por segundo, a configuração usando 3 caches permite obter tempos médios de resposta menores, o que garante o desempenho do servidor mesmo se acontecer um pico de requisições um pouco maior que 80.

Os principais motivos das diferenças de tempos de resposta nas simulações da figura 7.3 encontram-se em características do fluxo das requisições e desempenho dos caches em *Hit Ratio*. Considerando que 40% das requisições são feitas a documentos dinâmicos, 60% são requisições a documentos estáticos. Como os caches apresentam um *Hit Ratio* de 60%, 36% de todas as requisições feitas ao servidor são respondidas diretamente pelos caches. Por esse motivo, os tempos de resposta da simulação com apenas um cache foram altos, pois o cache único estava sobrecarregado de trabalho.

Taxas de 60% de *Hit Ratio* são relativamente fáceis de serem alcançadas pelos caches, considerando a pouca quantidade de arquivos armazenados no Beowulf e a distribuição Zipf das requisições a esses arquivos.

Uma outra simulação interessante é a que permite verificar a taxa de requisições que pode ser respondida por um Beowulf com determinada configuração. A figura 7.4 apresenta os tempos de resposta de um Beowulf com 32 nós.

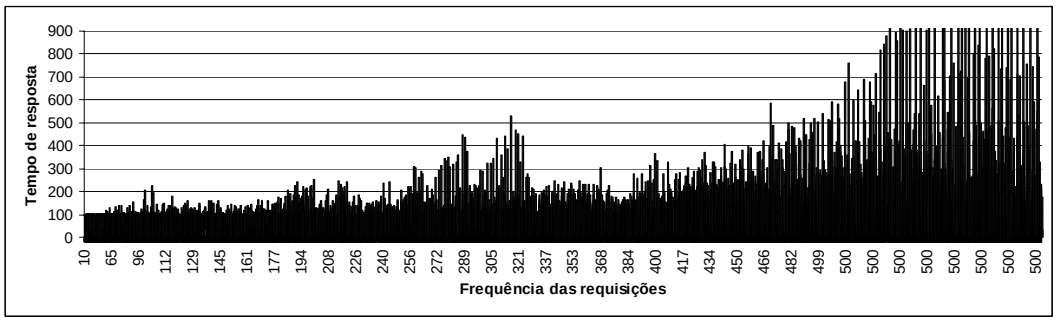

Fig. 7.4 – Simulação de Beowulf de 32 nós

A configuração utilizada na figura 7.4 foi a de 12 nós trabalhando como caches e 20 nós trabalhando como nós de processamento. Os demais parâmetros são os mesmos encontrados na tabela da figura 7.2. Essa foi a configuração capaz de responder à maior taxa de requisições usando 32 nós e, conforme pode ser visto na figura 7.4, foi por volta de 400 requisições por segundo.

O último resultado de simulação apresenta os tempos de resposta durante a reconfiguração do Beowulf. Os gráficos com esses tempos de resposta podem ser vistos nas figuras 7.5 a 7.8. Nas legendas das figuras, NP significa nó de processamento.

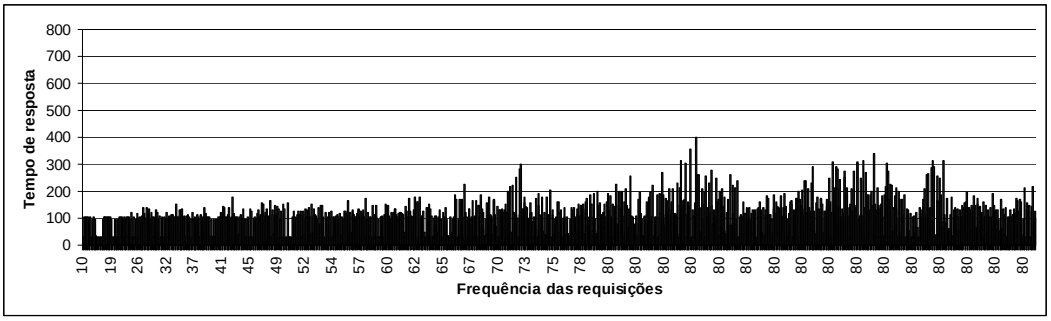

Fig. 7.5 - 4 caches; 4 NP; Cache Hit 60%; Pág. Dinâmica: 40%

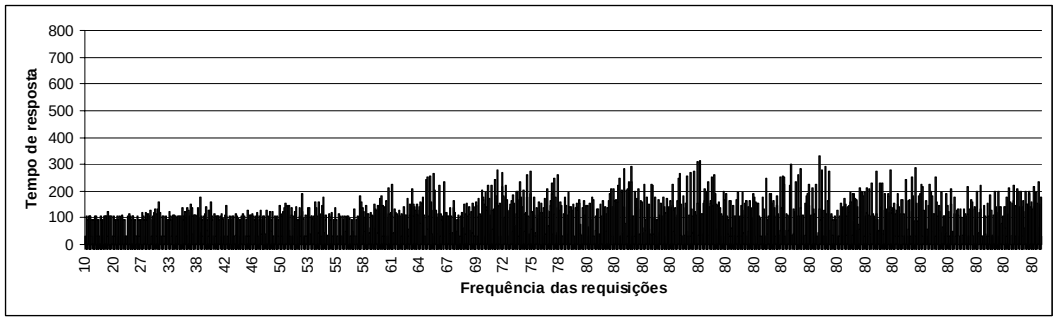

Fig. 7.6 - 4 caches; 5 NP; Cache Hit 60%; Pág. Dinâmica: 40%

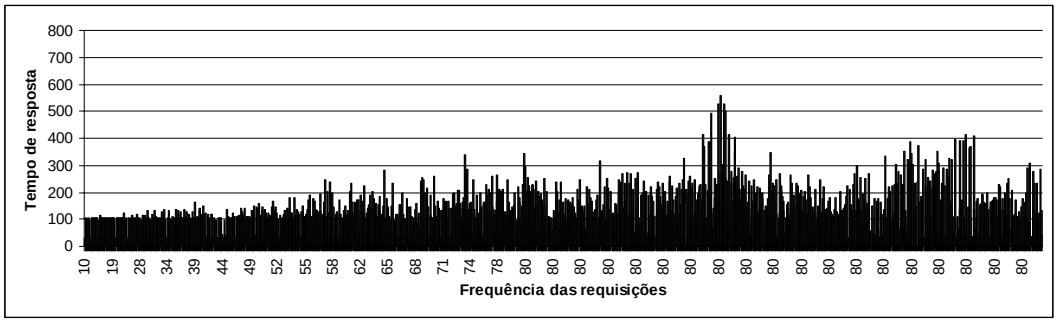

Fig. 7.7 - 4 caches; 5 → 4 NP; Cache Hit 60%; Pág. Dinâmica: 40%

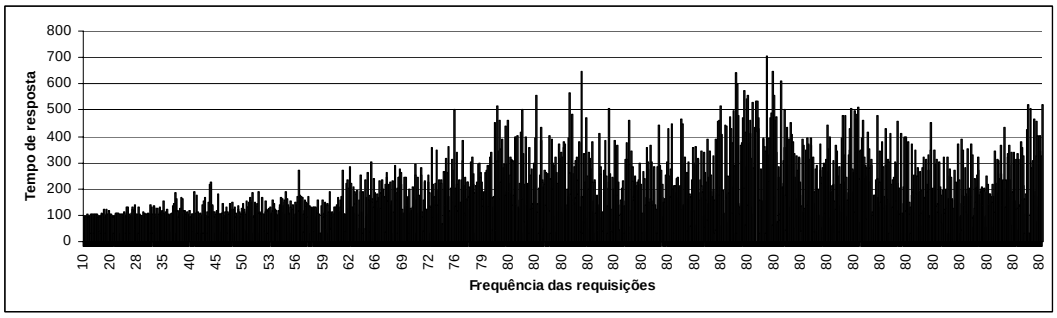

Fig. 7.8 - 4 caches; 5 → 4 NP; Cache Hit 30%; Pág. Dinâmica: 40%

A figura 7.5 apresenta os tempos de resposta de um Beowulf com 4 nós de processamento enquanto que o Beowulf da figura 7.6 possui 5 nós de processamento. Os tempos de resposta apresentados nos gráficos das figuras 7.7 e 7.8 são relativos a uma reconfiguração do Beowulf, com a alteração do número de nós de processamento, de 5 para 4. Pode-se notar que os tempos de resposta na figura 7.8 são maiores que na 7.7. Isso ocorre porque o Beowulf da figura 7.8 apresenta um desempenho de cache de apenas 30%. Isso mostra que o impacto da reconfiguração é menor em servidores cujos caches apresentam maior desempenho, o que pode ser conseguido, caso o percentual de requisições a documentos estáticos seja alto.

7. Conclusões e trabalhos futuros

O modelo apresentado nesse trabalho avalia a possibilidade de implementar um servidor Web utilizando um computador com arquitetura Beowulf.

Esse estudo mostrou, com o auxílio de simulações, que a utilização de características da arquitetura Beowulf pode ser feita para atingir objetivos importantes a um servidor Web tais como desempenho, escalabilidade e tolerância a falhas, sendo uma de suas principais características a capacidade de executar a reconfiguração dos nós do Beowulf sem interromper o fornecimento de arquivos requisitados.

Como trabalhos futuros, pretende-se avaliar o impacto no tempo de resposta do servidor da variação de parâmetros tais como o percentual de documentos dinâmicos, o desempenho em *Hit Ratio* dos caches e o tamanho médio dos arquivos. O modelo deve ainda ser implementado em um computador com arquitetura Beowulf pertencente ao Instituto de Computação da Unicamp. Esse Beowulf possui 66 nós com 256 MB de RAM cada, rodando o

sistema operacional Linux Red Hat. A implementação permitirá medir o desempenho do modelo em situações reais de trabalho.

Referências Bibliográficas

[1] The Beowulf Project web page. URL: http://www.beowulf.org, 2002.

[2] R. B. Bung, D. L. Eager, G. M. Oster, C. L. Williamson: "Achieving Load Balance and Effective Caching in Clustered Web Servers". *The 4th International Web Caching Workshop*, 1999.

[3] D. Andresen, T. Yang, V. Holmedahl, O. H. Ibarra: "SWEB: Towards a Scalable World Wide Web Server on Multicomputers". *Proceedings of IPPS*, 1996, Page(s): 850-856

[4] B. Narendran, S. Rangarajan, S. Yajnik: "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access". *Proceedings of the 16th Symposium on IEEE Reliable Distributed Systems*, 1997 , Page(s): 97 –106.

[5] "Extensible Web Server Project," The Systems Research Group, Department of Computer Science and Information Systems, The University of Hong Kong. URL: http://www.srg.csis.hku.hk/EWS/

[6] S. M. Baker, B. Moon: "Scalable Web Server Design for Distributed Data Management". *Proceedings of the 15th International Conference on Data Engineering*, 1999.

[7] A. Mourad and H. Liu:. "Scalable Web Server Architectures". *Second IEEE Symposium on Computers and Communications*, 1997, Page(s): 12 –16.

[8] V. Cardellini, M. Colajanni, P. S. Yu: "Dynamic Load Balancing on Web-Server Systems". *IEEE Internet Computing*, 1999

[9] V. Cardellini, M. Colajanni, P. S. Yu: "Redirection Algorithms for Load Sharing in Distributed Web-server Systems". *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999, Page(s): 528 –535.

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker: Web Caching and Zipf-like Distributions: Evidence and Implications. Proceedings of IEEE INFOCOM'99, New York, 03/1999.

[11] E. Cohen, B. Krishnamurthy, J. Rexford: "Efficient Algorithms for Predicting Requests to Web Servers". *IEEE*

[12] M. F. Arlitt, C. L. Williamson: "Web Server Workload Characterization: The Search for Invariants", *ACM Proceedings of the ACM SIGMETRICS conference on Measurement & Modeling of Computer Systems*, May 23-26, 1996

[13] P. Rodriguez, C. Spanner, E. W. Biersack: "Web Caching Architectures: Hierarchical and Distributed Caching". *The 4th International Web Caching Workshop*, 1999.

[14] S. Willians, M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox: Removal policies in network caches for World Wide Web Documents. *ACM SIGCOMM 96*, 293-305, 08/1996.

[15] P. Lorenzetti, L. Rizzo, L. Vicisano: Replacement policies for a proxy cache. Technical report, Universita di Pisa, Italy, 10/1996.

[16] H. Kim, K. Chon: Update policies for network caches. Technical Memo, Department of Computer Science, Korea Advanced Institute of Science and Technology, 07/1998.

[17] M. Kurcewicz, W. Sylwestrzak, A. Wierzbicki: A distributed WWW cache. Proceedings of the Third International WWW Caching Workshop, Manchester, England, 06/ 1998..

[18] R. Malpani, J. Lorch, D. Berger: Making World Wide Web caching servers cooperate. 4th International World-wide Web Conference,107-117, 12/1995.

[19] A. Dingle:. Web cache coherence. *Computer Networks and ISDN systems*, vol. 28, N° 7-11, 907-920, 1996.

[20] M. Abrams, C. R. Standridge, G. Abdulla, S. Willians, E. A. Fox: Caching proxies: Limitations and potentials. 4th International World-wide Web Conference, 119-133, 12/1995.

[21] I. Melve, L. Slettjord, H. Bekker, T. Verschuren: Building a Web caching system - architectural considerations. Desire Project, URL:http://www.uninett.no/prosjekt/desire/arneberg, 1997

[22] V. Kumar, A. Grama, A. Gupta, G. Karypis: "Introduction to Rarallel Computing – Design and Analisys of Algorithms", Chapter 4, The Benjamin Cummings Publishing Company, Inc. 1994.

[23] A. Burns, A, Wellings: "Real Time Systems and Programming Languages", Chapter 5, 2nd ed., Addison Wesley, 1996.

[24] C. J. Date: "*An Introduction to Database Systems"*, Addison Wesley, 1986

[25] The APACHE http Server projetc. URL: http://httpd.apache.org/, 2002.