PUC
RIO

Carlos Roberto Serra Pinto Cassino

Distribuição de Carga de Sistemas Web Controlada por Ferramentas de Construção de Páginas Dinâmicas

TESE DE DOUTORADO

Departamento de Informática

Rio de Janeiro, 19 de dezembro de 2000

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rua Marquês de São Vicente, 225 – Gávea
CEP 22453-900 Rio de Janeiro RJ Brasil
http://www.puc-rio.br

Carlos Roberto Serra Pinto Cassino

Distribuição de Carga de Sistemas Web Controlada por Ferramentas de Construção de Páginas Dinâmicas

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Doutor em Informática: Ciência da Computação.

Orientador: Roberto Ierusalimschy

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, 19 de dezembro de 2000

Aos meus pais, Fernando & Neide, e à Marcela.

Agradecimentos

Gostaria de agradecer ao meu professor e amigo, Roberto, pelos ensinamentos que moldaram minha formação, pelo apoio nas horas difíceis e pela amizade que marcou esses anos.

Agradeço também ao meu amigo Renato Cerqueira que, com seu companheirismo, me deu um grande incentivo durante todo esse tempo.

Agradeço muito à minha família e à Marcela que, sempre compreensivos, abdicaram de vários momentos para que eu pudesse me dedicar a este trabalho.

Agradeço aos laboratórios Tecgraf e LES pela infra-estrutura e pelo ambiente que me permitiram o desenvolvimento deste trabalho.

Agradeço à Rosane e demais funcionários do Departamento de Informática da PUC–Rio pelo apoio que me deram.

Finalmente, agracedeço à CAPES pelo auxílio financeiro ao longo do curso.

Resumo

Uma solução comumente adotada para resolver o problema de escalabilidade de servidores Web é o aumento do número de servidores através de algum mecanismo de distribuição de carga. Por outro lado, aplicações elaboradas, como, por exemplo, as de comércio eletrônico, utilizam ferramentas de construção de páginas dinâmicas como forma de prestar serviços cada vez mais sofisticados, o que exige uma capacidade de processamento ainda maior dos provedores de serviço. Com essa sofisticação das aplicações, a distribuição de carga pode sofrer restrições, como a necessidade de manter certos usuários fixos em certos servidores, o que diminui a eficiência do mecanismo de distribuição.

Nossa tese é que a integração do controle da distribuição de carga de um sistema Web na ferramenta de construção de páginas dinâmicas viabiliza o uso de informações às quais apenas a ferramenta de construção de páginas tem acesso, permitindo uma gerência mais efetiva dessa distribuição e uma maior eficiência do sistema. Neste trabalho nós propomos um modelo de distribuição no qual a ferramenta de construção de páginas controla, através da manipulação dos *links* internos das páginas geradas, a distribuição dos clientes nos servidores do *site*.

Uma vez que esse modelo de distribuição de carga difere de todos os demais modelos existentes, uma parte relevante de nosso trabalho é a implementação e análise de um protótipo que permita a validação das idéias propostas. Com o protótipo, podemos analisar o desempenho de diferentes algoritmos de distribuição, a sensibilidade a situações de sobrecarga, e outros. Através do protótipo, verificamos, por exemplo, que esse modelo de distribuição de carga apresenta um ótimo desempenho quando comparado a outros modelos comumente adotados. Além do desempenho, esse modelo integrador permite que as restrições impostas pelas aplicações sejam tratadas, como a fixação de usuários e a distribuição por classes de requisições.

Abstract

Load distribution among several servers is a scalability solution typically adopted in Web sites. E-commerce applications, among others, use dynamic pages to offer sophisticated services, demanding even more processing power from those servers. This sophistication imposes some constraints to the load distribution mechanism, such as fixing some clients on some servers, diminishing the performance of the distribution.

Our thesis is that integrating the load distribution control into the dynamic page construction tool allows the use of internal information of the tool, and therefore a more effective control of the distribution and more efficiency. In this work, we propose a load distribution model in which the page construction tool controls, by manipulating the embedded links of the generated pages, the distribution of the clients among the servers.

Since this load distribution model differs from other existing models, a relevant part of our work is the implementation and analysis of a prototype. With this prototype, we could analyze the performance of some load sharing algorithms, the behavior under overloads, and others. This analysis shows that this distribution model achieves a good performance when compared to others models commonly adopted. Moreover, this integrating model is able to handle the constraints imposed by the dynamic applications, such as client fixing and distribution guided by classes of requests.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Existem várias formas de se aumentar o desempenho de um site. Em termos de *hardware*, uma conexão de rede com maior taxa de transmissão, ou um servidor com maior capacidade de processamento podem ser alternativas viáveis. Entretanto, se a rede estiver corretamente dimensionada para comportar a demanda de acesso, a alternativa de aumento de capacidade do servidor não possui escalabilidade, uma vez que o aumento da demanda de serviços pode suplantar o aumento da capacidade do servidor, até o ponto onde não exista mais um servidor que tenha capacidade de suprir a demanda. Apesar do mesmo argumento poder ser usado para a rede, a capacidade desta geralmente é maior que a capacidade do servidor.

Uma solução normalmente adotada é o aumento no número de servidores do site, criando-se um *cluster* de servidores [Pfister, 1998]. Nessa abordagem, se a carga de trabalho for divida entre os servidores do cluster, um aumento na demanda de serviços poderá, teoricamente, ser acompanhada por um aumento no número de servidores. Naturalmente, a real escalabilidade dessa solução ainda pode esbarrar na capacidade de transmissão da rede. Todavia, em termos práticos, a multiplicação do número de servidores permite aumentos significativos de desempenho antes de alcançar os limites da rede, a custos mais baixos que os da troca de um servidor por outro de maior capacidade.

O aumento de desempenho de um site tem como objetivo direto a diminuição do tempo de espera do usuário. Com a distribuição da carga entre diferentes servidores, o processamento de requisições fica mais rápido e, conseqüentemente, o tempo médio de espera do usuário deve diminuir. Além do tempo de processamento da requisição pelo servidor, outro fator comumente responsável por uma fração significativa do tempo de espera do usuário é a transferência da resposta do servidor até o *browser*. Esse fato levou ao desenvolvimento de técnicas que permitem o *cache* de páginas em pontos intermediários da rede, como em *proxies* situados entre os browsers e os sites ou, até mesmo, nos próprios browsers dos clientes.

É importante notar que as técnicas de cache atuam em cooperação com as técnicas de distribuição de carga para um melhor desempenho de um site. Se a distribuição de carga equilibra o processamento de requisições entre os servidores, diminuindo o tempo médio de processamento, o uso de cache, por sua vez, colabora para a diminuição do número de requisições servidas diretamente pelo site, aumentando o desempenho dos servidores nas requisições restantes.

Simultaneamente aos ganhos de desempenho obtidos através da multiplicação de servidores e do uso de cache, novos desafios surgiram. Nos últimos anos, com o crescente aumento do número de usuários conectados à Internet, diversas empresas começaram a adotar a rede como veículo para a realização de novos negócios. Sites Web são usados hoje em dia como mídia de propaganda e venda de produtos, assim como para a prestação de serviços.

Esses usos trouxeram novas necessidades. Por exemplo, se uma empresa de aviação deseja disponibilizar um serviço de reserva de passagens através de seu site na rede, é necessário o uso de *páginas dinâmicas*. Páginas dinâmicas são assim chamadas por serem construídas no momento de sua requisição pelo usuário. Assim, é possível que uma mesma página, que em um dado momento tenha sido requisitada e indicasse determinado número de vagas em um vôo, em um outro momento posterior indicasse que o vôo não possui mais vagas disponíveis.

Outro exemplo do uso de páginas dinâmicas é na construção de sites que permitam uma dita *navegação contextual*. A navegação contextual é a capacidade de parametrizar uma página em função do caminho seguido pelo usuário até aquele ponto e/ou em função da própria identidade do usuário. Como um exemplo de navegação contextual, considere que um determinado conjunto de artigos possa ser pesquisado por autor ou por palavras-chave. Se um usuário faz uma pesquisa por autor, e essa pesquisa o leva à página de um determinado artigo, um *link* dessa página poderia indicar "próximo artigo do autor X". Já no caso da pesquisa ser feita por palavras-chave e levar inicialmente à mesma página da busca por autor citada anteriormente, o mesmo link, agora, poderia ser "próximo artigo contendo as palavras-chave ABC e XYZ".

O uso de páginas dinâmicas, como sugere o exemplo de navegação contextual exposto acima, pode ser ainda mais flexível em conjunto com o estabelecimento de sessões entre browsers e servidores. A idéia é que o servidor possa identificar os browsers que realizam as requisições para agir de forma diferenciada. Por exemplo, um recurso bastante utilizado comercialmente é o assim chamado *cesto de compras*. O funcionamento básico do cesto de compras é permitir que um usuário navegue pelo site, selecionando artigos, podendo comprá-los em uma só operação, ao fim da visita.

O aumento no uso de páginas dinâmicas tem sido acompanhado pelo aparecimento de diversas ferramentas de suporte à construção de sites baseados nesse tipo de páginas. Porém, como veremos a seguir, o uso de páginas dinâmicas e o estabelecimento de sessões com browsers, ao mesmo tempo em que permitem a criação de sites mais elaborados, exigem mais dos servidores e criam novos desafios para as técnicas que visam o aumento de desempenho de sites.

A construção de páginas dinâmicas foi inicialmente suportada por um padrão chamado Common Gateway Interface (CGI) [CGI, 1995]. O padrão CGI especifica uma interface através da qual um servidor Web pode executar um processo para construir a página que será retornada como resultado da requisição. Ou seja, para cada requisição, um novo processo é criado, esse processo recebe informações relativas à requisição e devolve ao servidor a página gerada, terminando sua execução em seguida.

O padrão CGI foi amplamente adotado pelos servidores Web e muito utilizado para a construção de páginas dinâmicas. Em particular, diversas linguagens de script como Perl [Wall et al., 1997], Tcl [Ousterhout, 1994], Python [Lutz, 1996] e Lua [Ierusalimschy et al., 1996], disponibilizam ferramentas ou bibliotecas para a construção de páginas dinâmicas. Essas ferramentas ou bibliotecas se baseiam no padrão CGI para oferecer ao programador a possibilidade de programar a página diretamente na linguagem de script correspondente.

O padrão CGI, entretanto, possui uma limitação em termos de desempenho, pois implica na execução de processo externo ao servidor para a construção de cada página solicitada. Por causa desse fator, diferentes servidores Web criaram APIs próprias para a codificação de programas de construção de páginas dinâmicas. Como exemplos podemos citar os módulos do servidor Apache [Laurie e Laurie, 1999], a interface NSAPI do servidor Netscape [Corporation, 1999b,a] e a interface ISAPI do servidor IIS da Microsoft [IIS, 1998]. Com uma interface própria do servidor exportada, o programa de construção de páginas pode ser carregado uma única vez, permanecendo em execução para tratar diversas requisições de serviço.

A introdução dessas interfaces proprietárias, por sua vez, estabeleceu um problema de portabilidade, pelo fato de elas serem diferentes entre si. O padrão Servlets [Davidson, 1999], para construção de páginas dinâmicas usando a linguagem Java [Arnold e Gosling, 1997], por exemplo, possui diversas implementações que exploram as interfaces próprias dos servidores suportados. O mesmo ocorre com aplicações como o CGILua [Hester et al., 1998a] que, além do padrão CGI, possui implementações utilizando ISAPI e módulo Apache [Hester, 1999].

Uma proposta de solução para esse problema de portabilidade, mantendo a melhora de desempenho, é o padrão FastCGI [Brown, 1996]. O FastCGI é uma extensão do padrão CGI, no qual a aplicação permanece em execução, mantendo uma conexão com o servidor, através da qual diversas requisições podem ser recebidas, processadas e retornadas.

Se, por um lado, a construção de páginas dinâmicas é diretamente suportada pelo padrão CGI e demais APIs, por outro lado o estabelecimento de sessões, técnica normalmente complementar ao uso de páginas dinâmicas, tem seu uso dificultado pelo fato do protocolo HTTP [Fielding et al., 1999] não possuir estado [Krishnamurthy et al., 1999]. Desse modo, só é possível identificar um usuário que esteja retornando ao site se ele próprio enviar essa informação. Dois mecanismos diferentes, baseados nesse princípio, são utilizados hoje em dia: reescrita de URLs e *cookies*. A reescrita de URLs se baseia na construção de páginas dinâmicas para incluir nas URLs de cada página um parâmetro que identifique o usuário. Com isso, quando o usuário seleciona um link dentro da página, ele inconscientemente está devolvendo ao servidor a informação sobre quem ele é, que havia sido inserida na página no momento de sua construção.

O outro método bastante adotado para manter uma sessão é guardar uma informação que identifique o usuário em um *cookie*, no seu browser. Um cookie, suscintamente descrito, é uma informação enviada pelo servidor para ser armazenada no programa que fez a requisição e ser devolvida ao servidor nas próximas requisições realizadas [Kristol e Montulli, 1997]. Assim, a mesma informação que, via reescrita, havia sido posta na URL, poderia ser armazenada em um cookie, sendo devolvida ao servidor automaticamente pelo browser nas requisições subseqüentes.

Páginas construídas dinamicamente, reescrita de URLs e cookies, embora eficazes na construção de sites mais elaborados e no estabelecimento de sessões, trazem novos desafios aos mecanismos de distribuição de carga e de cache de páginas. Por exemplo, com o uso de páginas dinâmicas, os mecanismos de cache ficam comprometidos, uma vez que diferentes requisições a um mesmo recurso podem receber diferentes respostas. Em particular, durante uma sessão entre um usuário e um site, as páginas passam a ser criadas para aquele usuário específico, diminuindo ainda mais a chance de que essas páginas, quando guardadas em proxies de cache, possam vir a ser reutilizadas. Com isso, há um aumento no número de requisições que chegam ao site, e o desempenho deste passa a depender mais fortemente do desempenho de seus servidores.

Mais ainda, trabalhos recentes [Cáceres et al., 1998; Feldmann et al., 1999] apontam que cookies estão presentes em parte significativa das requisições e que, uma vez que as respostas a essas requisições não podem ser armazenadas em caches, o desempenho destes cai significativamente. Em virtude disso, diversas técnicas já foram propostas para viabilizar o cache de páginas dinâmicas. Como exemplo de técnica de cache de páginas dinâmicas podemos citar o HPP [Douglis et al., 1997], que propõe a separação, através de novas *tags* HTML, entre as partes estáticas e as partes dinâmicas de uma página, para que as porções estáticas possam ser guardadas em caches e reaproveitadas. Outro exemplo, proposto como extensão do protocolo HTTP, é o chamado *delta-encoding* [Mogul et al., 1997], através do qual o servidor do recurso retornaria apenas as diferenças entre a nova resposta e uma outra, antiga, que estivesse armazenada em um cache. Todavia, é importante notar que, em ambos os casos, o servidor do recurso deve ser contactado a cada nova requisição, seja para fornecer o diferencial, seja para construir as parcelas dinâmicas da resposta. Novamente, o desempenho do site passa a ser mais dependente do desempenho de seus servidores.

Nesse cenário, os servidores passam a atender um número maior de requisições e, conseqüentemente, o mecanismo de distribuição de carga passa a ser mais exigido. Entretanto, essa distribuição da carga, responsável direta pelo desempenho do site, também é afetada pelo uso de páginas dinâmicas. Por exemplo, ferramentas de construção de páginas dinâmicas tipicamente oferecem meios de se manter um estado particular para cada sessão. Esse estado pode ser um conjunto de informações que reside na memória do servidor, persistindo, assim, entre consecutivas requisições de uma mesma sessão. Nesse caso, a distribuição de carga não pode mais ser igualmente aplicada a todas as requisições, pois acessos sucessivos provenientes de um mesmo browser, no caso do usuário deste browser estar com uma sessão em andamento, devem retornar sempre ao mesmo servidor, uma situação conhecida como "afinidade". Esse problema, como veremos, é de difícil solução, uma vez que o sistema que distribui a carga entre os servidores não possui a informação sobre quais requisições são provenientes de quais browsers, e nem se um determinado cliente está com uma sessão em andamento ou não.

Além da possibilidade do estabelecimento de sessões, outro fator desejável em aplicações de comércio eletrônico é a setorização do cluster. Essa setorização, ou seja, o particionamento do cluster em grupos de servidores, pode atender a diversos objetivos. Uma setorização por conteúdo, onde a separação é feita em função do conteúdo das páginas, pode garantir que processos mais caros computacionalmente sejam executados por servidores de maior capacidade. Já uma setorização baseada em características de usuários pode ser utilizada para separar clientes especiais dos demais clientes, de modo a garantir um melhor desempenho para os melhores clientes. O particionamento, assim como a manutenção de sessões, inviabiliza um tratamento homogêneo na distribuição das requisições por parte da ferramenta de distribuição de carga. Além disso, a questão da identificação do usuário que realiza uma requisição também pode ser crítica, como no caso do particionamento por grupos de clientes.

No nosso entender, os problemas acima descritos, associados ao uso de técnicas de distribuição de carga concomitantemente com o uso de ferramentas de construção de páginas dinâmicas, advém da falta de integração entre esses mecanismos. Nos parece que informações necessárias para um controle efetivo da distribuição de carga estão incorporadas na ferramenta de construção de páginas e nos próprios *scripts* que definem essas páginas. Assim, uma solução para os problemas citados acima poderia surgir da integração desses mecanismos.

De fato, nós analisamos diferentes técnicas de distribuição de carga e percebemos que diversas dessas técnicas simplesmente não abordam a questão de páginas dinâmicas e sessões, enquanto outras apresentam apenas soluções parciais para os problemas envolvidos. De forma análoga, fizemos também um estudo sobre técnicas de cache de páginas. Nesse estudo, observamos que, diferentemente do que acontece com a distribuição de carga, existem técnicas de cache que suportam o uso de páginas dinâmicas. Dentre essas técnicas, é possível notar que as mais flexíveis são justamente as que apresentam algum tipo de integração com os programas de construção de páginas. Esses estudos de técnicas de distribuição de carga e de cache de páginas colaboram para reforçar nossa idéia sobre a importância das informações detidas pela ferramenta de construção de páginas e pelos scripts.

Nossa tese é que a integração do controle da distribuição de carga de um sistema Web na ferramenta de construção de páginas dinâmicas viabiliza o uso de informações às quais apenas a ferramenta de construção de páginas tem acesso, permitindo uma gerência mais efetiva dessa distribuição e uma maior eficiência do sistema. Neste trabalho nós propomos um modelo de distribuição no qual a ferramenta de construção de páginas controla, através da manipulação dos links internos das páginas geradas, a distribuição dos clientes nos servidores do site.

Uma vez que esse modelo de distribuição de carga difere de todos os demais modelos existentes, uma parte relevante de nosso trabalho é a implementação e análise de um protótipo que permita a validação das idéias propostas. Com o protótipo, podemos analisar o desempenho de diferentes algoritmos de distribuição, a adequação de técnicas de manutenção de sessões, a sensibilidade a situações de sobrecarga, e outros. Através do protótipo, verificamos, por exemplo, que esse modelo de distribuição de carga apresenta um ótimo desempenho quando comparado a outros modelos comumente adotados.

Além do desempenho, o modelo que nós propomos possibilita um grande controle da distribuição das requisições, o que se traduz em flexibilidade. Como exemplo dessa flexibilidade, veremos que mecanismos de afinidade de clientes e de particionamento do cluster podem ser facilmente implementados. Adicionalmente, funcionalidades correlatas, como a utilização de métricas que visem a Qualidade de Serviço, também podem ser incorporadas.

Este trabalho está estruturado como se segue. No capítulo 2 nós apresentamos e analisamos um conjunto representativo de ferramentas de distribuição de carga. Para estruturar essa apresentação, nós propomos uma classificação das diferentes técnicas de distribuição de carga. Concluindo o capítulo 2, apontamos incompatibilidades entre o uso de páginas dinâmicas e as técnicas de distribuição de carga existentes.

Com base na análise das técnicas utilizadas para distribuição de carga, nós propomos, no capítulo 3, uma nova técnica que, integrada à ferramenta de construção de páginas dinâmicas, permite eliminar as incompatibilidades apontadas. Ainda no capítulo 3, nós apresentamos a arquitetura do protótipo que desenvolvemos para analisar a técnica de distribuição de carga que propomos.

No capítulo 4, nós expomos toda a estrutura de testes, isto é, os equipamentos utilizados, os programas e suas configurações. Além disso, explicamos os métodos de análise de desempenho que adotamos.

Em seguida, no capítulo 5, atestamos, embasados pelos testes realizados, a validade das idéias apresentadas. Ao longo do capítulo 5, exploramos diferentes algoritmos de distribuição, realizamos diversas análises de desempenho e, finalmente, fazemos uma comparação de desempenho com outras ferramentas.

Por fim, no capítulo 6, apresentamos as conclusões obtidas com esse trabalho e apontamos direções futuras.

Capítulo 2

Distribuição de Carga

Nesse capítulo nós analisamos várias alternativas para a distribuição de carga em clusters de servidores. Nós apresentamos tanto propostas acadêmicas quanto produtos comerciais. Nosso objetivo é mostrar um conjunto representativo das diferentes técnicas existentes para a distribuição de carga aplicáveis a servidores Web. Em particular, ao longo dessas apresentações, nós explicitamos, para cada caso, se há a possibilidade de estabelecimento de sessões e como é feita a manutenção dessas sessões. Outras características, como a possibilidade de particionamento do cluster, quando existentes, serão apontadas.

Ao término do capítulo, nós fazemos uma correlação entre as ferramentas apresentadas e o uso de páginas dinâmicas. Através dessa correlação, mostramos que pouco suporte é oferecido pelas ferramentas para viabilizar seu uso em sites dinâmicos. Em particular, mostramos que as técnicas de manutenção de sessões e de particionamento do cluster oferecidas não atendem integralmente a requisitos desejáveis em sites dinâmicos.

Apenas para estruturar a apresentação dos diferentes métodos de distribuição, nós propomos, na tabela 2.1, uma classificação dos agentes que, ativa ou passivamente, determinam como a distribuição é alcançada. Essa classificação divide, primeiramente, os agentes em duas categorias, em função de sua localização. Dentro dessas duas categorias, os agentes são, então, classificados em novas categorias, de acordo com o mecanismo utilizado para se obter efetivamente a distribuição. Desde já notamos que uma nova categoria será acrescentada a esse segundo nível da classificação para comportar o método de distribuição que iremos propor no capítulo 3.

A localização do controle da distribuição de carga, critério utilizado no primeiro nível da classificação, pode ser *externa* ou *interna* ao site. Por *site* nós entendemos a estrutura completa que, conectada

Classificação dos Agentes de Distribuição de Carga	
Externos	Usuários
	Browsers
	Proxies
Internos	Tradução de Endereços
	Manipulação de Pacotes
	Redirecionamento de Requisições
	Particionamento

Tabela 2.1: Classificação dos Agentes de Distribuição de Carga.

à rede, permite que o serviço seja prestado. Por exemplo, um site típico poderia ser composto por um cluster de computadores que executassem os servidores HTTP, um roteador que conectasse esses computadores à rede externa, e um servidor de nomes (*Domain Name System*, ou DNS) que fornecesse a tradução dos nomes divulgados em endereços IP a serem acessados pelos browsers. Nesse modelo, um site pode estar geograficamente distribuído, dispondo de mais de um conjunto como o exposto acima, todos prestando o mesmo serviço, ou parte dele.

É importante notar que estamos classificando os diferentes *agentes* responsáveis pela efetivação da distribuição de carga, e não os métodos em si. Ao longo das próximas seções, nós apresentamos cada uma das categorias, ilustrando-as com diferentes produtos ou propostas existentes. Em particular, segundo essa classificação, existem produtos e propostas híbridas que utilizam simultaneamente mais de um agente de distribuição de diferentes categorias.

2.1 Agentes Externos

Nessa seção nós apresentamos os agentes externos que efetivam a distribuição de carga em um cluster de servidores Web. Conforme apresentado, os agentes externos ao site, que são explorados na tarefa de distribuição, são os usuários, os browsers e os proxies.

2.1.1 Usuários

Do ponto de vista do administrador de um site, a forma mais simples de prover alguma distribuição das requisições dos usuários entre diferentes servidores é deixar essa tarefa a cargo dos próprios usuários. Ou seja, o agente responsável pela distribuição de carga é o usuário.

Essa política pode ser viabilizada através da criação de espelhos. Um espelho é uma cópia completa de um site (ou da parte do site que se deseja distribuir) em um servidor diferente do original. A distribuição das requisições entre os espelhos é viabilizada através da exibição, para os usuários, de uma lista de endereços nos quais um determinado site (ou recurso) pode ser acessado. Presumindo-se que os usuários farão diferentes escolhas, as requisições serão distribuídas entre todos os servidores. O diagrama da figura 2.1 exemplifica essa forma de distribuição.

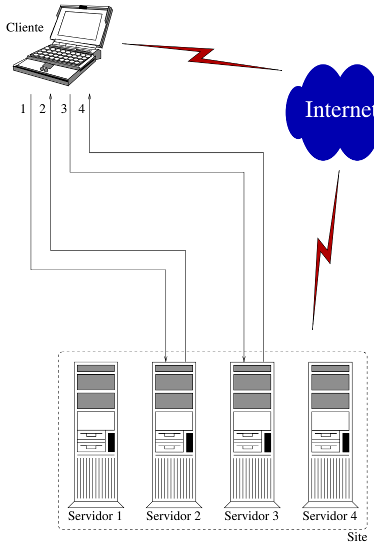
Através dessa forma de distribuição, a questão de manter uma sessão entre um usuário e o site deixa de ser um problema. Afinal, como o usuário é sempre atendido pelo mesmo servidor, o contexto sempre estará disponível.

Esse método, entretanto, possui pontos bem fracos. Um deles é o fato da distribuição ser dada ao acaso, dificilmente chegando até mesmo próximo a uma distribuição ideal. Outro ponto é o fato de que a granularidade do controle da distribuição é muito grande, isto é, uma vez que um usuário escolha um site, ele permanecerá lá, a menos que ele explicitamente mude para outro espelho. Por fim, essa tarefa adicional é muitas vezes inconveniente para o usuário. Por esses motivos, o uso de espelhos, como aqui exposto, não é muito adotado para a distribuição de carga.

Apesar das pesquisas indicarem que 10% dos sites existentes possuem espelhos [Bharat e Broder, 1999], é importante notar que espelhos possuem outras utilidades, como aumentar a disponibilidade e oferecer traduções, além da função de distribuição de carga aqui citada. Mais ainda, como veremos nas próximas seções, espelhos são utilizados por diversas outras técnicas de distribuição de carga que não requerem a intervenção, ou até mesmo o conhecimento, do usuário.

2.1.2 Browsers

É possível adotarmos uma forma de distribuição semelhante à anteriormente descrita mas deixando



1. Cliente realiza uma requisição.
2. A página retornada oferece diversos links para o mesmo recurso, em diferentes servidores.
3. Cliente escolhe uma das localizações e efetua a requisição.
4. O servidor contactado responde a requisição com a página correspondente.

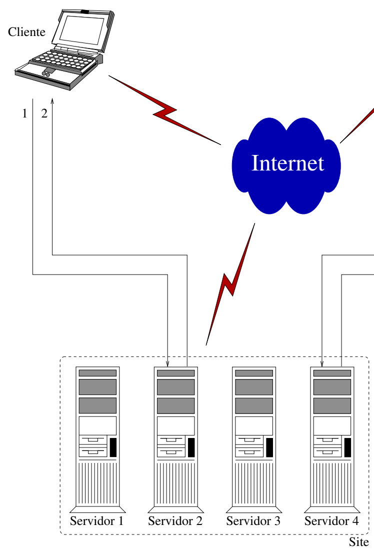
Figura 2.1: Usuário como agente de distribuição.

a seleção do espelho a cargo do browser. Dessa forma, o usuário não precisa conhecer o conjunto de espelhos e a distribuição pode vir a ser feita de forma mais adequada. A figura 2.2 mostra, esquematicamente, esse método de distribuição.

Existem duas maneiras distintas para que o browser faça a seleção de um servidor (espelho) a ser acessado. A primeira é a programação desse recurso na própria implementação do browser. Entretanto, essa codificação do controle da distribuição de carga no browser não pode ser considerada de uso geral, uma vez que se aplica apenas a empresas que constróem browsers. Além disso, os endereços dos servidores (ou pelo menos parte deles) devem ser conhecidos antes que os browsers sejam distribuídos. Se todo o conjunto de endereços for alterado, novas configurações dos browsers precisarão ser enviadas para os usuários. Essa técnica foi utilizada no browser da Netscape [Mosedale et al., 1997] para permitir a distribuição de carga em seu site.

Uma outra forma, mais flexível, que permite a escolha do servidor pelo browser é a execução de um aplicativo proveniente do próprio site onde a distribuição deve ser feita. Em [Yoshikawa et al., 1997], os autores propõem o uso de *applets* Java que, sendo executados nos browsers, escolheriam o melhor servidor a ser acessado para a obtenção do recurso sendo solicitado. Essa proposta procura eliminar os dois problemas citados acima: por um lado, a mudança de endereços não é problemática, uma vez que o applet sempre sabe de onde foi carregado e, se comunicando com seu servidor de origem, pode atualizar toda a lista de servidores disponíveis. Por outro lado, o applet também pode obter as informações de carga de cada servidor do cluster e, com base nelas, determinar qual a melhor escolha. Outro ponto positivo é que a manutenção de sessões e o particionamento do cluster podem ser controlados diretamente pelo applet.

Uma desvantagem do uso de applets é a comunicação adicional necessária entre o applet e o servidor para controlar a distribuição de carga. Em particular, essa comunicação estará acontecendo com a latência da rede que conecta o browser ao servidor. Por outro lado, esse modelo não possui boa



1. Browser traduz o nome do servidor no endereço IP de um dos espelhos que compõem o site e faz a requisição.
2. O servidor contactado responde a requisição com a página correspondente.

Figura 2.2: Browser como agente de distribuição.

portabilidade do ponto de vista do browser, uma vez que exige que este execute applets.

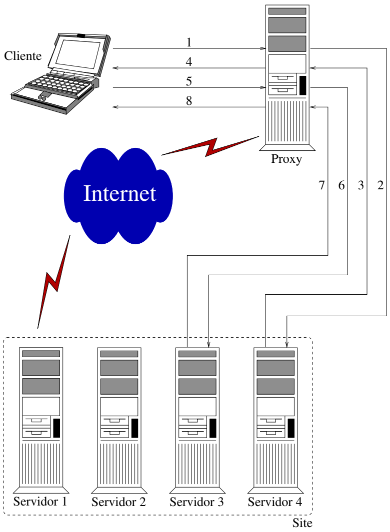
A distribuição de carga controlada por applets possui ainda outro problema: a impossibilidade do uso de uma estrutura HTML padrão, utilizando links normais nas partes HTML das páginas. Essa impossibilidade vem do fato de que o applet precisa ter controle sobre as requisições. Ou seja, se houver um link normal na página, o usuário pode segui-lo e o browser irá requisitar essa nova página sem que o applet seja informado. Para evitar isso, ou os links devem estar contidos no applet, ou uma programação JavaScript adicional passa a ser necessária, com o agravante de que a comunicação entre applets e JavaScript não é padronizada entre diferentes browsers.

Esse tipo de arquitetura, na qual um aplicativo proveniente do site realiza a distribuição, pode ser adotado para automatizar a distribuição de granularidade grande obtida através de espelhos. Para tal, o aplicativo pode atuar apenas na primeira página, direcionando o usuário para um servidor no qual ele permanecerá por todos os demais acessos, cumprindo dessa forma, o papel do usuário na escolha do espelho. Nesse caso os problemas da comunicação excessiva e a restrição ao uso de uma estrutura HTML padrão deixam de existir mas, em contrapartida, a distribuição alcançada não será a ideal.

2.1.3 Proxies

O terceiro e último agente externo ao site que pode ser responsável pela distribuição de requisições é o proxy. A idéia é fazer com que o proxy, ao qual o browser faz todas as requisições, conheça o conjunto de espelhos e, de alguma forma, distribua as requisições entre os servidores desse conjunto, como exemplificamos no diagrama da figura 2.3.

Um exemplo de uso desse tipo de agente é apresentado em [Baentsch et al., 1997], onde um serviço chamado *Web Location and Information Service*, implementado no proxy, guarda os múltiplos endereços de uma URL e encaminha as requisições para o servidor mais disponível. Outra arquitetura, mais flexível, poderia ser construída com os *cachelets* apresentados em [Cao et al., 1998]. Cachelets são pequenas aplicações, escritas em alguma linguagem portável, que, embutidas em páginas HTML,



1. Cliente requisita uma página ao proxy.
2. Proxy escolhe um dos servidores do site e repassa a requisição.
3. O servidor contactado responde a requisição com a página correspondente.
4. Proxy retorna a página recebida ao cliente que a solicitou.
5. Em um segundo acesso, o cliente requisita uma nova página.
6. Proxy escolhe um servidor possivelmente diferente e repassa a requisição.
7. O servidor contactado responde a requisição com a página correspondente.
8. Proxy retorna a página recebida ao cliente.

Figura 2.3: Proxy como agente de distribuição.

são detectadas e extraídas pelos proxies, onde são executadas. Seu objetivo primário é controlar o cache: o proxy, antes de retornar uma página do cache, executaria o cachelet que poderia autorizar o retorno, pedir para que uma nova página fosse buscada no servidor de origem ou, até mesmo, construir uma nova página para ser retornada ao browser que fez a requisição. Com essa funcionalidade, o cachelet poderia, sempre que fosse necessário obter uma nova página, fazer a escolha de um dos servidores e requisitar a página, distribuindo, assim, as requisições.

A distribuição localizada no proxy compartilha de uma vantagem também presente quando a distribuição está localizada no browser: apesar de haver um processamento efetivo, responsável pela distribuição de carga, esse processamento se encontra distribuído pela rede, permitindo que todo o processamento do cluster seja utilizado no tratamento das requisições. Entretanto, diferentemente dos mecanismos de distribuição baseados nos browsers, os localizados nos proxies não conseguem resolver de forma direta a questão da manutenção de sessões. Afinal, como as requisições de diversos clientes chegam a um mesmo proxy, seria necessário algum tipo de identificação que permitisse associar as requisições aos clientes. Essa questão não é abordada pelos trabalhos que analisamos.

Por fim, um mecanismo de distribuição implementado no proxy possui um sério problema de aplicabilidade, uma vez que os proxies existentes hoje em dia não possuem as funcionalidades exigidas para a implantação desse tipo de serviço. Ou seja, seria necessária uma mudança na estrutura atual da rede para que um mecanismo desse tipo pudesse ser adotado.

2.2 Agentes Internos

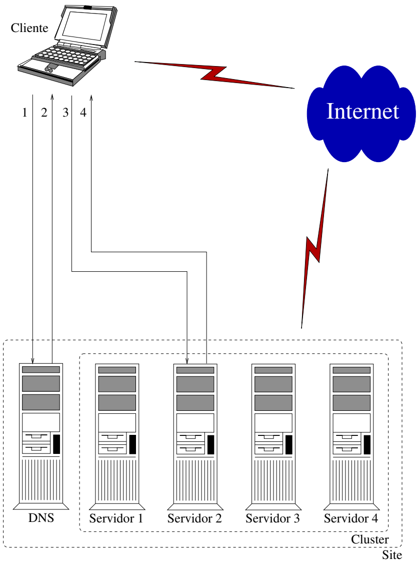
Nessa seção nós apresentamos os agentes internos ao site responsáveis pela efetivação da distribuição de carga no cluster de servidores Web. Dentro do site, os agentes que são utilizados pelas ferramentas de distribuição de carga existentes são: a tradução de endereços, a manipulação de pacotes de rede, o redirecionamento de requisições e o particionamento do site.

2.2.1 Tradução de Endereços

Internamente ao site, um agente que pode ser utilizado para distribuir as requisições por um conjunto de servidores é o servidor de DNS (*Domain Name System*) [Mockapetris, 1987]. De fato, o primeiro mecanismo adotado para fazer com que um cluster de servidores HTTP atuasse consistentemente como um único servidor utilizou uma rotação de endereços conhecida como *Round Robin* DNS (RR-DNS) [Brisco, 1995].

O princípio básico de funcionamento desse mecanismo é como se segue: um usuário escreve o endereço de um site em seu browser, requisitando que a página em questão seja exibida. Para tal, o browser, inicialmente, pede ao servidor de DNS local uma tradução do nome do site em um endereço IP correspondente. Esse servidor local de DNS, por sua vez, pode repassar o pedido a outro servidor de DNS e, em algum momento, essa cadeia de requisições termina por chegar ao servidor de DNS primário do site a ser acessado pelo browser. Esse último servidor de DNS, no momento da requisição da tradução, busca o próximo servidor em uma lista cíclica de todos os servidores disponíveis, retornando seu endereço IP como resultado. O resultado é, então, utilizado pelo browser para efetuar as requisições solicitadas pelo usuário. Na figura 2.4, nós apresentamos esquematicamente esse modelo de distribuição.

Esse mecanismo assume que todos os servidores tenham acesso a todas os recursos sendo solicitados, seja através de replicação (isto é, servidores espelhos), seja através de algum tipo de sistema de arquivos distribuído, ou outras formas. Ou seja, é assumido que qualquer servidor pode responder a



1. No primeiro acesso, o cliente requisita a tradução do nome do site em um endereço.
2. Servidor de DNS escolhe o endereço de algum dos servidores do cluster e o envia como tradução do nome.
3. Usando o endereço recebido, o cliente efetua a requisição.
4. O servidor contactado responde a requisição com a página correspondente.

Figura 2.4: DNS como agente de distribuição.

qualquer requisição.

A técnica de RR-DNS foi utilizada nos projetos do servidor Web da National Center for Supercomputing Applications (NCSA) [Katz et al., 1994; Kwan et al., 1995] e no Western Research Laboratory da DEC [Mogul, 1995b,a]. Entretanto, esses trabalhos apontam que o balanceamento alcançado pode ficar comprometido pois não há como limitar o número de vezes que uma tradução nome/IP, divulgada pelo DNS, será utilizada.

Esse comportamento fica claro se observarmos que a tradução divulgada pelo DNS possui um prazo de validade, conhecido como *Time To Live* (TTL). Após passado o tempo especificado no TTL, a tradução deve ser descartada e uma nova tradução deve ser solicitada. Porém, até que esse prazo tenha terminado, qualquer servidor de DNS da cadeia de requisições de traduções exemplificada acima pode guardar o endereço obtido e retorná-lo diretamente em resposta a qualquer outra requisição da mesma tradução. Com isso, o servidor de DNS de um grande provedor de acesso à rede, por exemplo, pode reutilizar a tradução inúmeras vezes, gerando um grande volume de requisições para um único servidor. Simultaneamente, um outro servidor de DNS de uma pequena rede local, por exemplo, pode utilizar uma outra tradução do mesmo nome apenas uma única vez, gerando um conjunto desprezível de requisições.

Teoricamente, é possível evitar essa situação com o uso de um valor pequeno de TTL, mas essa diminuição do tempo de validade das traduções é problemática em dois aspectos: por um lado, se o TTL for muito pequeno aumenta-se muito a comunicação entre o servidor que executa o RR-DNS e os demais servidores de DNS da rede. Justamente por causa dessa comunicação excessiva, diversos servidores de DNS impõem um valor mínimo de TTL, comprometendo a eficácia da diminuição do TTL. Por outro lado, quando um browser obtém a tradução de um nome em um endereço, ele tipicamente armazena essa informação para usos futuros. Esse comportamento é válido uma vez que o TTL se aplica apenas aos servidores DNS, e não aos browsers. Um estudo mostrado em [Dias et al., 1996] mostra que, ainda que um valor muito pequeno de TTL pudesse ser utilizado, o cache de IP feito nos próprios browsers seria suficiente para comprometer o balanceamento de carga no site.

O cache da tradução feito pelo browser, entretanto, faz com que a questão da manutenção de sessões seja diretamente resolvida: afinal, todas as requisições são dirigidas ao mesmo servidor, que pode armazenar o contexto referente à sessão. É importante notar, porém, que esse é um comportamento *típico*, não sendo especificado e podendo variar entre browsers. Nesse caso, não havendo o cache da tradução, o estabelecimento da afinidade entre um cliente e um servidor não teria como ser feito.

Outra questão a ser apontada é o particionamento do cluster. Uma divisão por serviços poderia ser feita através da criação de nomes distintos que se traduzissem em endereços do sub-conjunto de servidores responsável por cada serviço. Ou seja, não é possível criar partições utilizando o mesmo nome do site, afinal a tradução já teria sido feita e o browser acessaria diretamente o servidor cujo endereço IP tivesse sido retornado.

Com relação à questão da má distribuição de carga obtida através do Round Robin DNS, existem alguns recursos que podem ser utilizados. Primeiramente, é importante notar que a técnica de RR-DNS, como apresentamos acima, não leva em consideração o estado dos servidores. Por exemplo, se um servidor Web deixa de responder por falha, ou por sobrecarga, o servidor de DNS continua a utilizar seu endereço em novas traduções. Assim, um primeiro avanço é obter a informação sobre a disponibilidade dos servidores pois, com a retirada de um servidor parado, ou sobrecarregado, da lista de traduções, o desempenho geral do cluster pode ser bastante melhorado [Colajanni et al., 1998b]. Esse tipo de recurso é adotado, por exemplo, no sistema SunSCALR [Singhai et al., 1998].

Outro fator que pode ser considerado é a carga dos servidores. De posse da carga de cada servidor, um algoritmo de tradução de nomes pode, ao invés de uma simples rotação, efetuar uma distribuição baseada nas cargas. Esse tipo de algoritmo, baseado nas cargas dos servidores, é explorado tanto em produtos comerciais, como no Cisco DistributedDirector [Systems, 1996], quanto em propostas acadêmicas, como em [Cardellini et al., 1999a].

Diferentes informações provenientes do cliente que requisita a tradução também podem ser utilizadas para parametrizar as traduções. Um exemplo é a informação de proximidade entre o cliente e os diferentes servidores, ou clusters, disponíveis. O DistributedDirector utiliza essa informação, além da carga dos servidores, como mencionado acima, para tentar direcionar os clientes para o servidor mais próximo.

Cada servidor de DNS possui um "peso" inerente, isto é, um número potencial de clientes que poderão utilizar a tradução para efetuar requisições. Voltando ao exemplo citado no início dessa seção, o servidor de DNS do provedor de acesso à rede possui um peso maior que o servidor de DNS da pequena rede local. Esse peso, então, também pode ser utilizado na função de tradução. Em [Colajanni et al., 1997] o peso dos servidores de DNS é utilizado para evitar que dois servidores diferentes de grande peso recebam o mesmo mapeamento. Para tal, são criadas duas listas independentes de rotação de nomes, uma para os servidores cujo peso seja menor que um determinado valor, e outra para os servidores cujo peso seja maior.

Por fim, do ponto de vista da tradução, uma variável que pode ser explorada para se obter uma melhor distribuição é o valor de TTL. Por exemplo, servidores de DNS de grande peso podem receber traduções que expirem mais rapidamente, enquanto servidores de DNS de peso pequeno podem receber traduções mais duráveis. Além disso, traduções com tempos diferenciados de TTL podem ser utilizadas para lidar com clusters compostos por servidores de capacidades diferentes. Algoritmos de DNS que utilizam valores dinâmicos de TTL são apresentados em [Colajanni et al., 1998a].

Embora os diversos fatores apontados acima possam melhorar a distribuição da carga nos servidores, é importante notar que, ainda assim, o servidor de DNS não tem controle direto sobre as requisições que chegam ao site. Uma vez fornecida uma tradução, é possível estimar qual será a carga decorrente mas não é possível determinar com precisão qual será essa carga e nem por quanto tempo ela se manterá. A única forma de resolver esse problema, permitindo que o servidor de nomes alcance uma granularidade mais fina no controle das requisições, é através de uma mudança conceitual no serviço de nomes.

Um exemplo de proposta nessa linha é o chamado *Active Names* [Vahdat et al., 1999]. Nessa abstração, a tradução do nome do servidor e a obtenção do recurso aconteceriam em um único passo. A idéia é associar programas a nomes de serviços de forma a que esses programas sejam responsáveis por localizar e transportar o recurso solicitado. Esses programas poderiam, então, conhecer o conjunto de servidores e usar as diversas informações aqui apresentadas para distribuir a carga, a cada diferente requisição. Ou seja, todos os problemas decorrentes do cache das traduções deixariam de existir.

2.2.2 Manipulação de Pacotes

Como apresentado anteriormente, o servidor de DNS tem um controle parcial sobre as requisições que chegam ao site no sentido de que, uma vez que algum cliente (ou proxy) possua uma tradução válida de um nome em um endereço IP, as requisições são enviadas a esse endereço, sem que haja, ou possa haver, qualquer interferência por parte do servidor de DNS. Existem outros agentes de distribuição que podem obter um controle maior. A manipulação dos pacotes da comunicação entre o cliente e o servidor é um desses agentes.

Com a manipulação dos pacotes sendo transmitidos pela rede pode-se simular a conexão entre um cliente e um servidor qualquer. Simular porque, para o cliente, o servidor conectado pode ser um, enquanto o servidor que efetivamente recebe os pacotes referentes à requisição e retorna a resposta pode ser outro.

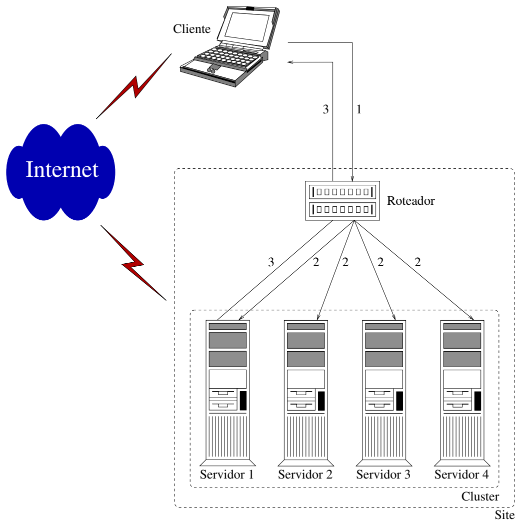
Existem diferentes formas de manipulação de pacotes:

1. Replicação
2. Reescrita dupla de endereços
3. Reescrita simples de endereços
4. Redirecionamento

Mecanismos de distribuição baseados na manipulação de pacotes tipicamente utilizam um elemento central responsável por essa manipulação. Esse elemento pode ser um servidor dedicado, um roteador, ou um switch, por exemplo. Em uma arquitetura típica, apenas o elemento central tem seu endereço IP divulgado para a rede externa através de um mapeamento estático de DNS. Ou seja, o nome do site fica associado ao endereço IP do elemento central. Todas as requisições provenientes dos clientes são, então, recebidas por esse elemento central, sendo então direcionadas para algum servidor do cluster. Dessa maneira, esse tipo de arquitetura permite um controle total sobre a distribuição das requisições nos servidores.

Replicação

Na distribuição das requisições através de replicação, o elemento central recebe todos os pacotes provenientes dos clientes e os repassa para a rede local utilizando o endereço de *broadcast*, o que faz com que todos os pacotes cheguem a todos os servidores. Com isso, é necessário que os próprios servidores decidam que pacotes devem ser tratados por quais servidores.



1. Cliente requisita uma página usando o endereço divulgado do site.
2. Roteador recebe os pacotes da requisição e os reescreve para envio a todos os servidores do cluster.
3. Apenas um servidor aceita os pacotes e responde ao cliente.

Figura 2.5: Replicação como agente de distribuição.

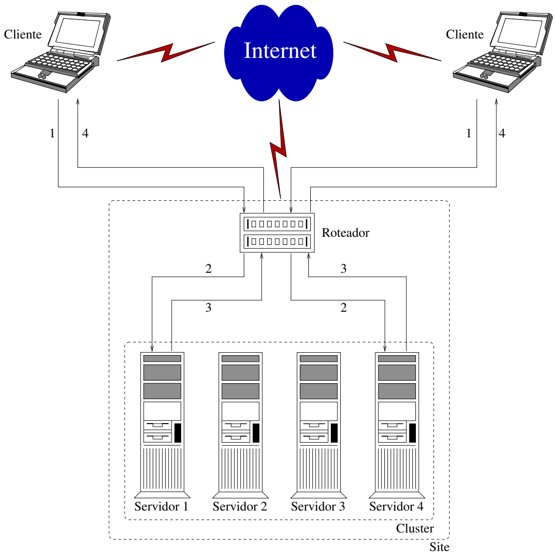
Uma técnica de distribuição de carga chamada ONE-IP [Damani et al., 1997] utiliza a replicação para obter distribuição. No ONE-IP, todos os servidores recebem todos os pacotes de todas as conexões. Assim, cada servidor decide quais requisições deve atender e processa apenas os pacotes relativos a essas requisições. Para que não haja dois servidores atendendo a uma mesma requisição, utiliza-se uma função de *hash*, sobre o endereço IP de origem do pacote, como forma de determinar qual servidor atenderá qual requisição. A figura 2.5 apresenta a estrutura de distribuição por replicação.

Essa política pode levar a uma má distribuição da carga entre os servidores. Por exemplo, um proxy de uma empresa, que realiza os acessos de inúmeros funcionários através do mesmo endereço IP, será sempre atendido pelo mesmo servidor. Além disso, como a distribuição das requisições não leva a carga dos servidores em consideração, não há nenhum procedimento que possa retirar um servidor de um estado de sobrecarga.

Apesar da questão do estabelecimento de sessões com os clientes não ser discutida, essa técnica apresenta, por construção, uma afinidade baseada no endereço IP, uma vez que todas as conexões provenientes de um mesmo endereço serão tratadas sempre pelo mesmo servidor. Entretanto, como veremos na seção 2.3, a afinidade baseada apenas em endereços IP não é suficiente para a manutenção de sessões. Por outro lado, esse tipo de afinidade torna o particionamento do cluster inviável, dado que cada cliente (IP) acessa apenas um único servidor.

Reescrita Dupla de Endereços

Na reescrita dupla de endereços, o elemento central, ao receber um pedido de conexão de um cliente, escolhe um servidor do cluster, reescreve o endereçamento do pacote vindo do cliente e o envia para esse servidor escolhido. Nesse mesmo momento, o elemento central atualiza uma tabela interna para lembrar que o cliente e o servidor escolhido estão com uma conexão aberta. O servidor, sem mesmo tomar conhecimento que houve uma tradução de endereços, processa a requisição e tenta enviar



1. Cliente requisita uma página usando o endereço divulgado do site.
2. Roteador recebe os pacotes da requisição e os reescreve para enviar ao servidor escolhido.
3. O servidor escolhido aceita e responde a requisição.
4. Roteador intercepta os pacotes da resposta e os reescreve para enviar ao cliente.

Figura 2.6: Reescrita dupla de endereços como agente de distribuição.

a resposta diretamente ao cliente. O elemento central, então, intercepta essa resposta e, novamente, reescreve os endereços do pacote para que o cliente tenha a impressão de que a conexão foi estabelecida com o IP originalmente acessado por ele. Novos pacotes que sejam trocados entre o cliente e o servidor passam pela mesma tradução, conforme especificado pela tabela de conexões que é mantida pelo elemento central. Ao detectar um pedido de término de conexão, o elemento central, após reencaminhar o pacote, retira a associação cliente/servidor de sua tabela de conexões. Nós apresentamos um esquema da distribuição por reescrita dupla de endereços na figura 2.6.

A reescrita dupla de endereços é executada pelo elemento central através de uma função conhecida como *Network Address Translator* (NAT) [Egevang e Francis, 1994]. NAT é uma função de roteador que atua como uma espécie de filtro, traduzindo endereços externos à sub-rede para endereços internos à sub-rede. Diversos trabalhos utilizam essa técnica para propor diferentes ferramentas de distribuição de carga em clusters. Dentre eles podemos citar o MagicRouter [Anderson et al., 1996], além de produtos comerciais, como o Cisco LocalDirector [Systems, 1997] e Alteon Web Switch [Systems, 1999a].

MagicRouter [Anderson et al., 1996] é uma proposta acadêmica que visa tanto a distribuição de carga quanto tolerância a falhas no cluster. O MagicRouter é, de fato, uma aplicação que deve ser instalada em um servidor que se situe entre a rede externa e o cluster. Assim, esse servidor faz o papel do elemento central, recebendo todos os pacotes provenientes dos clientes. Essa abordagem faz com que não seja necessária a aquisição de um roteador ou um switch específico para distribuir a carga no cluster mas sim um computador como outro qualquer do cluster.

MagicRouter faz um tratamento dos pacotes a nível de aplicação, mas utiliza memória compartilhada com o *kernel* do sistema operacional (no caso, Linux) para obter um desempenho essencialmente igual àquele conseguido quando o tratamento dos pacotes é feito no nível do kernel. Uma desvantagem dessa abordagem é que, apesar do MagicRouter ser uma aplicação, o kernel do sistema operacional teve que ser modificado para permitir sua implementação.

A função NAT é desempenhada pelo MagicRouter, que faz a reescrita de pacotes e os reencaminha, mantendo a tabela de conexões ativas. A escolha do servidor é feita através de um dentre três algoritmos de distribuição de carga: Round Robin, aleatório e 'carga incremental'. Esse terceiro algoritmo faz a seleção do servidor através de uma estimativa da carga de todos os servidores. Essa estimativa é feita mantendo-se uma tabela de carga dos servidores que é atualizada a cada conexão estabelecida ou terminada. Quando uma conexão se estabelece, a carga do servidor escolhido é adicionada de um fator que refletiria o aumento de carga relativo àquela conexão. Analogamente, ao término de uma conexão, o servidor em questão tem sua carga estimada decrescida desse fator. Com base nessa tabela de cargas, quando uma conexão se inicia, é escolhido o servidor de menor carga.

A detecção de falha é feita através de requisições que são enviadas para todos os servidores periodicamente. Se um servidor não responder em determinado tempo, considera-se que houve uma falha, e esse servidor passa a não receber mais conexões.

O Cisco LocalDirector [Systems, 1997] atua de forma semelhante mas é, de fato, um roteador, com software proprietário. Esse roteador mantém as tabelas de conexões e efetua a escolha do servidor que tratará cada conexão.

O Alteon Web Switch [Systems, 1999a] também é composto por hardware e software dedicados às funções de distribuição de carga e controle de falhas. Esse switch fornece três modos diferentes de seleção de servidores:

most-available pode utilizar o número de conexões de cada servidor para escolher o servidor mais disponível, ou então fazer a seleção através de Round Robin. Em ambos os casos, de maneira igualitária entre os servidores, ou utilizando pesos que permitam carregar mais os servidores de maior capacidade.

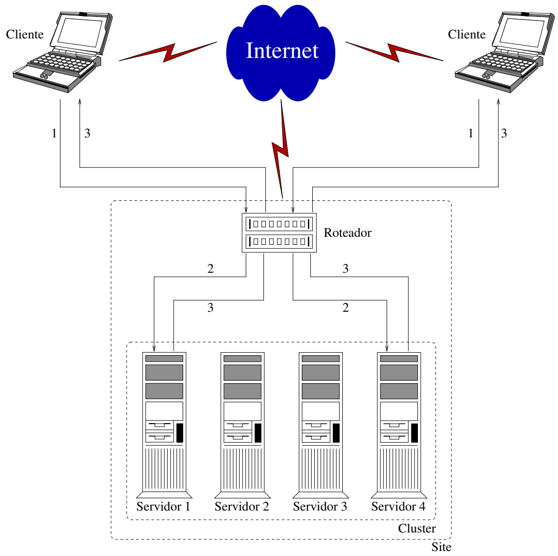
persistence-based permite manter sessões a nível de aplicação entre os clientes e os servidores. Existem três diferentes formas:

- **hash** usa o endereço IP do cliente como entrada de uma função *hash* que resulta em um índice na tabela de servidores.
- **SSL session tracking** usa o identificador da conexão segura estabelecida entre o cliente e o servidor para manter o cliente neste servidor específico.
- **cookie-based** o servidor inclui seu endereço IP em um cookie que é adicionado à resposta. Com base nesse cookie, novas conexões podem ser devolvidas ao servidor inicial.

URL-based permite a setorização do cluster através da criação de regras que associem URLs a servidores. É possível escolher um dado servidor para cada diferente tipo de URL (por exemplo, todas as URLs terminadas em '.jpg' vão para um servidor e todas que contenham '/cgi-bin/' vão para outro).

Esses métodos adotados para a manutenção de sessões e para o particionamento do cluster não são utilizados apenas pelo Alteon Web Switch e, como veremos na seção 2.3, apresentam alguns problemas.

Em todas as ferramentas que utilizam a técnica NAT, um problema se mantém: como todos os pacotes que fluem, entrando e saindo do site, passam por uma reescrita no elemento central, este pode ser um gargalo do sistema. Ou seja, o elemento central pode comprometer a real escalabilidade do sistema pois pode haver um ponto onde a capacidade da rede ainda não tenha sido esgotada, o número



1. Cliente requisita uma página usando o endereço divulgado do site.
2. Roteador recebe os pacotes da requisição e os reescreve para enviar ao servidor escolhido.
3. O servidor escolhido aceita e responde a requisição diretamente ao cliente.

Figura 2.7: Reescrita simples de endereços como agente de distribuição.

de servidores ainda possa ser aumentado, mas o elemento central não seja capaz de processar um número maior de requisições.

Além disso, o elemento central também se apresenta como ponto de falha do sistema pois uma interrupção no seu funcionamento implica na parada total do site. Tanto o MagicRouter quanto o Cisco LocalDirector resolvem esse último problema permitindo que um segundo elemento atue em modo 'hot-standby'. Esse segundo elemento não atua na distribuição das requisições mas fica apenas monitorando o elemento primário, assumindo suas funções em caso de falha.

Já no caso do Alteon Web Switch, dois switches podem atuar em modo ativo, onde ambos os switches atendem a requisições de clientes e as repassam para os servidores do cluster. Os endereços dos switches podem ser divulgados, por exemplo, através de RR-DNS. Se um dos switches falhar, metade das conexões existentes, em média, são afetadas.

De modo geral, as ferramentas que apresentamos podem ser economicamente inviáveis em determinadas situações. Por exemplo, para mudarmos de um servidor para dois, a curva de custo não é linear, mas a de capacidade sim. Essa situação é ainda agravada se desejarmos não ter o roteador como ponto de falha do sistema, pois implicaria na aquisição de dois aparelhos. No caso do MagicRouter e do LocalDirector o segundo elemento nem ao menos pode vir a colaborar para o aumento da capacidade do site.

Reescrita Simples de Endereços

Na reescrita simples de endereços, apenas os pacotes que chegam ao site têm seus endereços alterados. Nessa abordagem, o elemento central recebe os pacotes provenientes dos clientes e reescreve seus endereços IP para reenviá-los aos servidores escolhidos. Os servidores, entretanto, tratam as requisições e enviam as respostas diretamente para os clientes, usando o endereço IP do elemento central como endereço de origem em seus pacotes, como exemplificado na figura 2.7.

Um exemplo de uso de reescrita simples de endereços é o *TCP Router* [Dias et al., 1996]. Semelhante às ferramentas que utilizam reescrita dupla, essa ferramenta adota uma arquitetura na qual um roteador tem seu endereço associado ao nome do site, recebe as conexões dos clientes e reescreve os pacotes para distribuí-los aos servidores do cluster. Cada servidor recebe o pacote reescrito, processa a requisição e, diferentemente da reescrita dupla, responde diretamente ao cliente, usando o IP do roteador como origem do pacote enviado. Através desse mecanismo, os pacotes de resposta não precisam ser reescritos pelo roteador.

Um ganho relevante dessa técnica em relação ao NAT é que uma grande parte dos pacotes deixa de ser reescrita pelo elemento central, diminuindo o volume de processamento requerido. Em um servidor Web, em particular, o volume de saída é muito maior que o volume de entrada, pois as requisições tipicamente são pequenas quando comparadas aos resultados, que são páginas HTML, imagens, arquivos etc. Ou seja, é justamente o maior fluxo de tráfego que deixa de ser processado pelo elemento central. Em contra partida, é necessário que o sistema operacional de todos os servidores seja modificado para que eles possam responder diretamente aos clientes usando o endereço IP do elemento central como origem.

Essa ferramenta ainda compartilha de um problema comum às que utilizam o NAT: a centralização do controle. O elemento central é um ponto de falha do sistema e, embora menos que o do NAT, ainda um fator limitante de sua escalabilidade. Especificamente, o trabalho do TCP Router não aborda a questão de sessões a nível de aplicação, assim como a de particionamento do cluster.

Redirecionamento

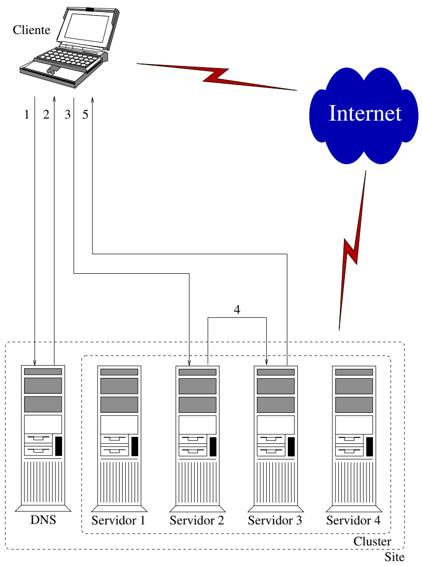
Nessa abordagem, o elemento central que conecta os clientes ao cluster apenas redireciona os pacotes de cada conexão para o servidor escolhido utilizando o seu endereço *Media Access Control* (MAC). Com esse redirecionamento baseado apenas no endereço MAC, o servidor que recebe o pacote o trata como se o tivesse recebido diretamente do cliente e, assim, pode responder também diretamente a esse cliente.

Para tal, é necessário que o endereço do elemento central seja registrado como um *alias*, ou *loopback*, em cada servidor do cluster. Além disso, os servidores não podem responder a pedidos de tradução do protocolo de resolução de endereços (ARP) para esse endereço. Porém, essas características são configurações que normalmente podem ser feitas nos sistemas operacionais. Com isso, essa abordagem torna-se totalmente transparente tanto para os clientes quanto para os servidores, o que a difere da reescrita simples de endereços, onde os servidores precisam ser modificados.

O redirecionamento se assemelha à reescrita simples de endereços, no sentido de que apenas os pacotes que chegam dos clientes precisam de algum tratamento por parte do elemento central. Assim, o tráfego de saída, justamente por não ter que ser processado, pode ser direcionado através de uma rede diferente da rede pela qual chegam as requisições. Uma vantagem dessa separação é que o dimensionamento das redes pode ser feito de forma independente.

Existem produtos comerciais que exploram o redirecionamento de pacotes para obter o balanceamento de carga. O IBM SecureWay Network Dispatcher [Gage, 1999; Goldszmidt e Hunt, 1997], e outro produto da Cisco, o MultiNode Load Balancing [Systems, 1999b], são exemplos que podemos citar.

A questão do estabelecimento de sessões a nível de aplicação foi abordada inicialmente pelo Network Dispatcher através de um mecanismo de afinidade, chamado *server affinity*. Quando ativo, esse mecanismo faz com que a primeira conexão de um cliente ao cluster seja distribuída do modo normal mas as demais conexões desse mesmo cliente voltem ao servidor que tratou a primeira. A afinidade é baseada no IP do cliente, ou seja, o Network Dispatcher usa o endereço IP do solicitante



1. No primeiro acesso, o cliente requisita a tradução do nome do site em um endereço.
2. Servidor de DNS escolhe o endereço de algum dos servidores do cluster e o envia como tradução do nome.
3. Usando o endereço recebido, o cliente efetua a requisição.
4. O servidor contactado responde a requisição diretamente ou a reencaminha para outro servidor do cluster.
5. O novo servidor responde a requisição com a página correspondente diretamente para o cliente.

Figura 2.8: DPR: distribuição através de RR-DNS e redirecionamento de pacotes.

para assumir que dois acessos provêem do mesmo cliente. Além disso, o período de tempo durante o qual cada cliente estará preso a um servidor particular é configurável.

Conforme veremos na seção 2.3, e segundo notam os próprios autores, esse tipo de controle de sessões pode ser ineficaz em diversas situações. Por isso, em uma versão posterior do sistema, foi disponibilizada uma API para controle do mecanismo de manutenção de sessões. Assim, se for necessário, é possível escrever o código que controle o mecanismo de sessões da forma adequada à aplicação específica.

Uma arquitetura híbrida, chamada *Distributed Packet Rewriting* (DPR), que utiliza a técnica de RR-DNS aliada ao redirecionamento de pacotes, foi proposta em [Bestavros et al., 1998] e utilizada em [Aversa e Bestavros, 1999] na construção de um cluster de servidores Web. Ao contrário de todos os produtos e as propostas aqui apresentadas, que utilizam diferentes tipos de manipulação de pacotes, essa abordagem não se baseia em um elemento central dedicado ao controle do trânsito dos pacotes. Ao invés, o DPR distribui essa tarefa entre os servidores do cluster.

Inicialmente, o DPR torna públicos os nomes de todos os servidores do cluster, através de RR-DNS. Essa divulgação através de RR-DNS leva a uma distribuição inicial (sabidamente não ideal) dos clientes nos servidores. Cada servidor, no entanto, ao receber um pedido de conexão, avalia se deve aceitar a conexão, ou repassá-la para outro servidor que esteja mais disponível no momento. Ou seja, cada servidor do cluster atua tanto como provedor do serviço, quanto como agente distribuidor de conexões. A figura 2.8 apresenta uma representação esquemática da distribuição de carga através do DPR.

Essa abordagem, sendo distribuída, soluciona o problema de escalabilidade presente em todas as demais ferramentas baseadas na manipulação de pacotes por meio de um elemento central. Uma vez que cada servidor atua tanto como provedor de serviço quanto como distribuidor de conexões, o aumento no número de máquinas se reflete não só em um aumento na capacidade de prestação do serviço, como também em um aumento na capacidade de distribuição das conexões.

Outro fator decorrente da distribuição do roteamento de conexões é que o DPR passa a ter uma curva de custo linear, pois não necessita de um servidor dedicado à essa tarefa. É importante notar que os produtos comerciais, baseados em hardware próprio, como MultiNode Load Balancing e o Network Dispatcher, representam um grande investimento, provavelmente injustificável para clusters pequenos—o que ainda se agrava caso a questão de tolerância a falha implique na aquisição de dois equipamentos para evitar a criação de um único ponto de falha.

Com relação à confiabilidade, a ferramenta distribuída não introduz um único ponto de falha do sistema. No caso de falha em um dos servidores do cluster, apenas as conexões servidas por este servidor (localmente ou através de redirecionamento) serão afetadas. Os trabalhos que apresentam o DPR não abordam a questão de manutenção de sessões com os clientes.

2.2.3 Redirecionamento de Requisições

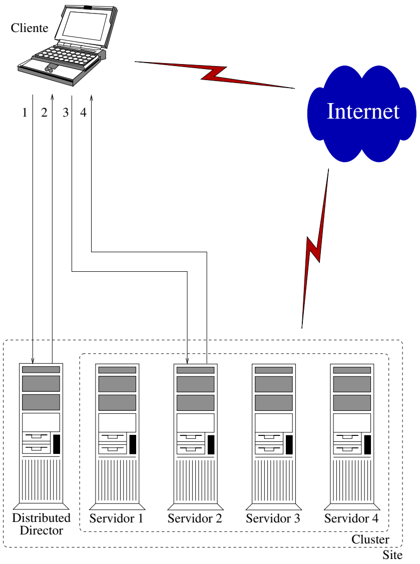
O protocolo HTTP possui diversos códigos que, retornados em resposta a uma requisição, indicam que o recurso desejado encontra-se em uma outra localização. Em particular, dentre esses códigos, existem alguns que permitem especificar que determinado recurso está temporariamente deslocado. Um browser, ao receber um desses códigos como resposta, realiza uma nova requisição ao endereço temporário, que também é fornecido na resposta, para obter o recurso originalmente requisitado pelo usuário. O fato do código significar um redirecionamento temporário implica que, caso o mesmo browser deseje obter novamente o mesmo recurso, ele o fará a partir do servidor originalmente contactado. Também é possível fazer um redirecionamento permanente, no qual o browser "esquece" o endereço original, passando a utilizar apenas o novo [Fielding et al., 1999].

Esse suporte a diferentes tipos de redirecionamento pode ser utilizado para se obter uma distribuição das conexões em um conjunto de servidores. Para tal, basta que o primeiro acesso de um cliente obtenha como resposta um redirecionamento que o leve ao servidor que efetivamente tratará sua requisição. Através de uma coordenação dos endereços retornados aos clientes, é possível obter uma boa distribuição das requisições. Entretanto, há certas ressalvas a serem observadas.

O protocolo HTTP permite que os browsers procedam com o redirecionamento (temporário ou permanente) de uma forma automática, isto é, sem pedir autorização ao usuário, apenas quando a segunda requisição for do tipo `GET` ou `HEAD`. Assim, se o browser faz uma requisição do tipo `GET`, e obtém um redirecionamento como resposta, uma nova requisição pode ser automaticamente enviada ao novo servidor para obter o recurso solicitado. Por outro lado, se a requisição original for do tipo `POST`, por exemplo, e um redirecionamento for obtido em resposta, duas diferentes situações podem ocorrer: se a resposta indicar que uma nova requisição do tipo `GET` deve ser feita ao endereço alternativo, então esta requisição pode ser feita automaticamente pelo browser, sem o conhecimento do usuário. Porém, se a nova requisição tiver que ser feita mantendo o método `POST` da requisição original, então o browser deve avisar ao usuário, permitindo que este decida se o redirecionamento deve ser feito, ou se a requisição deve ser cancelada [Fielding et al., 1999].

Com isso, requisições do tipo `POST`, comumente utilizadas em conjunto com formulários HTML, caso sejam redirecionadas, precisam ser transformadas em requisições do tipo `GET`. Caso contrário, o usuário seria forçado a cumprir um papel ativo na distribuição, sendo avisado pelo browser e tendo que autorizar os redirecionamentos recebidos. Esse comportamento, naturalmente, inviabilizaria o uso do redirecionamento HTTP para distribuição de carga.

O problema associado à troca do método de requisição é que ela pode acarretar uma mudança na resposta à requisição. Por exemplo, um site de páginas dinâmicas construído em Java, segundo o



1. Cliente requisita uma página usando o endereço divulgado do site.
2. DistributedDirector recebe a requisição, escolhe um servidor e retorna um redirecionamento para o cliente.
3. Cliente segue o redirecionamento e requisita novamente a página usando o novo endereço.
4. O servidor contactado responde a requisição com a página correspondente.

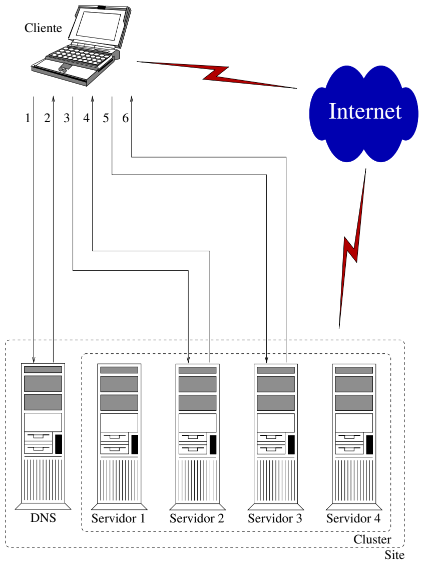
Figura 2.9: DistributedDirector: distribuição através de redirecionamento HTTP.

padrão Servlets, pode estar programado para aceitar apenas os métodos que o próprio site determina que sejam usados nas páginas (`POST`, por exemplo). Caso esse site seja instalado em um cluster cuja ferramenta de distribuição de carga utilize redirecionamentos HTTP, e realize a troca de métodos `POST` por `GET`, ele não funcionará [Davidson, 1999; Hunter e Crawford, 1998].

A solução para esse problema é simples: redirecionar apenas as requisições dos tipos `GET` e `HEAD`, cujo tratamento pode ser feito automaticamente pelo browser. A desvantagem dessa solução é que ela restringe a quantidade de requisições passíveis de redirecionamento, conseqüentemente diminuindo o controle que o método tem sobre a distribuição da carga.

O redirecionamento das requisições pode ser feito, por exemplo, por um elemento central, dedicado a essa tarefa. Um exemplo de um produto comercial que adota essa política é o Cisco DistributedDirector [Systems, 1996]. Esse produto pode atuar de duas formas distintas. Na primeira forma, que já apresentamos na seção 2.2.1, o DistributedDirector atua como um servidor de DNS, resolvendo os nomes para os clientes baseado em critérios como proximidade e carga dos servidores. Já o segundo modo de operação faz com que o DistributedDirector atue como um servidor HTTP. Nesse modo, as requisições provenientes dos clientes são aceitas, as mesmas métricas de proximidade e carga de servidores são aplicadas e, por fim, uma resposta HTTP, contendo um código de redirecionamento temporário e o endereço do servidor escolhido, é retornada para o cliente. Para esse funcionamento, o site deve ter seu nome mapeado, estaticamente, via DNS, para o endereço IP do DistributedDirector. Na figura 2.9, nós apresentamos um diagrama da distribuição realizada pelo DistributedDirector.

Outra forma de se utilizar o redirecionamento provido pelo protocolo é através de uma arquitetura distribuída, semelhante ao DPR, onde cada servidor pode realizar redirecionamentos de clientes para atingir um equilíbrio de cargas. Um sistema que utiliza essa abordagem é o SWEB [Andresen et al., 1997, 1995]. Nesse sistema, assim como no DPR, todos os servidores têm seus nomes divulgados através de um servidor de RR-DNS. Com isso, estabelece-se uma distribuição inicial dos clientes



1. No primeiro acesso, o cliente requisita a tradução do nome do site em um endereço.
2. Servidor de DNS escolhe o endereço de algum dos servidores do cluster e o envia como tradução do nome.
3. Usando o endereço recebido, o cliente efetua a requisição.
4. O servidor contactado responde a requisição diretamente ou redireciona o cliente para outro servidor do cluster.
5. Se for o caso, o cliente segue o redirecionamento e requisita novamente a página usando o novo endereço.
6. O servidor contactado responde a requisição com a página correspondente.

Figura 2.10: SWEB: distribuição através de RR-DNS e redirecionamento HTTP.

no servidores. Através de diversas métricas, porém, determinadas requisições podem ser eleitas para serem redirecionadas de um servidor para outro, por meio de respostas de redirecionamento temporário do HTTP. A figura 2.10 representa o funcionamento do SWEB.

Já em [Cardellini et al., 1999b], os autores utilizam as técnicas de DNS que descrevemos na seção 2.2.1 para obter a distribuição principal dos clientes nos servidores mas cada servidor, ao receber uma requisição de um cliente, pode decidir redirecioná-lo para algum outro servidor. Diferentemente do SWEB, é utilizado um redirecionamento permanente, ou seja, o usuário passa a enviar todas as demais requisições para o novo servidor.

Um problema geral das ferramentas que utilizam redirecionamento HTTP, além da impossibilidade de tratar requisições do tipo `POST` como já explicado, é o aumento da latência percebida pelo usuário. Isso ocorre porque o redirecionamento implica em um retorno ao browser para que este faça uma nova requisição ao servidor escolhido. Ou seja, duas requisições são feitas ao site para a obtenção de um recurso. Além disso, os trabalhos que apresentamos não abordam a questão de manutenção de sessões.

2.2.4 Particionamento

Uma forma bem simples de se dividir os acessos dos usuários em vários servidores é particionar o site por esses vários servidores, distribuindo os arquivos que o compõe. Se os links entre os arquivos são ajustados para refletir essa distribuição, a própria navegação dos clientes os levará a visitar diferentes servidores.

Essa abordagem singela possui seus problemas. Por um lado, a distribuição obtida dificilmente será boa, uma vez que, normalmente, a maior parte dos acessos a um site se concentram em poucas páginas, e cada página sempre será direcionada para um único servidor. Por outro lado, as tarefas de distribuição e manutenção das páginas ficam dificultadas em função da consistência dos links que ligam as páginas, e da própria localização dessas páginas.

Uma possível solução para esses problemas pode surgir da automação do processo de particionamento. Essa é a proposta de *Distributed Cooperative Web Servers* (DCWS) [Baker e Moon, 1999]. Esse trabalho se fundamenta no fato de que um site pode ser visto como um grafo direcionado, onde os recursos são os nós e os links são os arcos direcionados entre os nós. A idéia é, então, distribuir esse grafo entre os servidores de forma a que a carga seja igualmente distribuída entre eles. O método proposto se baseia na hipótese de que os sites tipicamente possuem poucos pontos de entrada conhecidos, através dos quais todos os clientes começam a navegação.

Em um cenário exemplo, todo o site está, inicialmente, em um único servidor. À medida que as requisições chegam, as páginas mais acessadas podem começar a migrar para os outros servidores que estejam disponíveis. Para que haja a migração de uma página, todos os links para essa página que existirem nas demais páginas do site são reescritos para seguir a página na sua nova localização, assim como os links da própria página são reescritos para manter as referências ao servidor original. Os pontos de entrada, contudo, são impedidos de migrar—apenas páginas internas do site são passíveis de migração.

Imediatamente após a migração de uma página, todo o site está em um estado consistente, onde os links apontam para as páginas nos servidores corretos. Contudo, novas requisições a uma página migrada podem chegar ao servidor que originalmente a mantinha. Nesse caso, o servidor deve responder com um redirecionamento que leve o cliente ao servidor correto.

O sistema DCWS é projetado para que cada servidor possa ser tanto um servidor primário para seus pontos de entrada e demais páginas, quanto um servidor em cooperação que receba páginas sendo migradas de outro servidor. As informações de carga são replicadas em todos os servidores para que as decisões de migração possam ser tomadas de modo a balancear da melhor forma possível a carga entre todos os servidores.

O DCWS, apesar de ser distribuído, não possui as mesmas vantagens das demais ferramentas distribuídas aqui apresentadas, como o ONE-IP ou o DPR: por exemplo, nesse caso, a falha de um servidor implica na perda das páginas que este mantinha, pois não há replicação de conteúdo entre os servidores. Já em uma ferramenta como a que utiliza o DPR, por exemplo, a falha de um servidor significava apenas a perda das conexões sendo tratadas (ou redirecionadas) por este servidor e, nesse caso, os clientes, após o erro, poderiam voltar a acessar a mesma página usando outro servidor.

Além disso, a migração de páginas entre servidores traz uma limitação adicional: o uso de *bookmarks*. Para que o modelo proposto no DCWS funcione, é necessário forçar o usuários a entrar no site por um dos pontos de entrada oficiais, para que o caminho até uma dada página possa ser encontrado. Esse modelo é comumente violado através de bookmarks nos clientes e do uso de mecanismos de busca, que tipicamente retornam referências diretas para páginas internas dos sites.

Os autores não abordam a questão de manutenção de sessões. Porém, como as páginas são distribuídas entre servidores, tendo seus links reescritos para manter as ligações originais entre páginas do site, nos parece não ser possível manter um usuário sendo atendido por um mesmo servidor, ao menos dentro da proposta de migração de páginas apresentada. Isso porque, para manter um usuário em um servidor, seria necessário que este servidor mantivesse todas as páginas do site (e não apenas um sub-conjunto, como proposto pelos autores), de modo a que esse cliente pudesse navegar por todo o site.

Mourad e Liu [1997] apresentam uma proposta híbrida, que combina as técnicas de particionamento, redirecionamento e mapeamento de nomes, na construção de um servidor Web escalável. Os autores propõem o particionamento do site por conjunto de servidores HTTP, sem replicação de conteúdo. Os clientes, distribuídos em um segundo conjunto de servidores através de RR-DNS, seriam, então, redirecionados aos servidores HTTP por meio do redirecionamento provido pelo protocolo HTTP.

Os autores defendem essa arquitetura argumentando que uma arquitetura na qual todos os servidores possuam cópia de todas as informações não é boa por dois motivos: o espaço gasto em disco com esse armazenamento é muito grande; e o fato de que o mecanismo de cache do sistema operacional não atinge um bom desempenho, dado que virtualmente todas as páginas estarão sendo servidas por todos os servidores.

Nessa arquitetura, os servidores HTTP informam periodicamente sua carga a um gerente de carga. Assim, através de um mecanismo automatizado de migração de páginas, esse gerente pode modificar o particionamento para obter uma melhor distribuição das requisições pelos servidores.

Os autores citam que uma grande vantagem dessa abordagem é que ela pode ser implementada sem nenhuma modificação nos softwares envolvidos (HTTPD, DNS, browsers). Porém, ela compartilha das mesmas desvantagens do DCWS: a falha de um servidor, na ausência de replicação, significa a perda de parte do site e o uso de bookmarks não pode ser bem resolvido.

2.3 Correlação com Páginas Dinâmicas

Nas seções anteriores, apresentamos diversas ferramentas de distribuição de carga, aplicáveis a sistemas Web. Porém, como apontamos ao longo dessas análises, as ferramentas de distribuição de carga tipicamente não provêem um suporte específico para o uso de páginas dinâmicas e outros recursos correlatos. Nessa seção nós procuramos detalhar um pouco mais as restrições existentes.

Como já foi dito, sob o enfoque da ferramenta de construção de páginas dinâmicas, um recurso tipicamente oferecido é o estabelecimento de sessões, com possibilidade de manutenção de contextos associados. Esse contexto poderia representar, por exemplo, os ítens do cesto de compras de um usuário. Já a implementação desse contexto, em diversas ferramentas, pode ser feita por um estado armazenado em memória [Forta et al., 1999; Davidson, 1999; Hester, 1999]. Nesse caso, um usuário que esteja como uma sessão estabelecida com o site não pode ser atendido por diferentes servidores. Caso o usuário seja desviado para outro servidor, seu contexto não estará acessível, invalidando a sessão. Ou seja, se uma sessão está em andamento, a ferramenta de distribuição de carga precisa atuar em cooperação com a ferramenta de construção de páginas, mantendo o usuário em um único servidor.

Outro aspecto importante a se considerar é o particionamento do cluster. Partições podem ser criadas por diferentes motivos: dedicar servidores de maior capacidade de processamento para determinado grupo de clientes, restringir certos serviços a um determinado sub-conjunto de servidores, e outros. Nesses casos, a ferramenta de distribuição precisa fazer os direcionamentos corretos, de forma a respeitar os particionamentos.

As restrições apresentadas, referentes à manutenção de sessões e ao particionamento de clusters, além de outras, representam sérios empecilhos às ferramentas de distribuição de carga expostas nesse capítulo. Algumas dessas ferramentas suportam, de diferentes maneiras, a manutenção de sessões a nível de aplicação. Existem, também, formas de se particionar um site. Entretanto, como expomos nas próximas seções, tanto os métodos de manutenção de sessões, quanto os de particionamento dos sites, são, de modo geral, insatisfatórios.

2.3.1 Manutenção de Sessões

Como apresentamos na seção 2.2.2, produtos comerciais, como o IBM Network Dispatcher e o Alteon Web Switch, possuem mecanismos de afinidade que fazem com que a primeira conexão de um cliente ao cluster seja distribuída de modo normal, mas que as demais conexões voltem sempre ao servidor que tratou a primeira. Uma vez que o protocolo HTTP não possui estado, essa afinidade é mantida com base no endereço IP dos clientes, ou seja, duas conexões provenientes de um mesmo IP são consideradas como oriundas de um mesmo cliente. Porém, esse tipo de abordagem pode trazer três problemas:

1. A degradação do desempenho do mecanismo de distribuição: o mecanismo de afinidade baseado no endereço IP é binário, isto é, ou todos os usuários do site estarão em afinidade, ou nenhum estará. Com isso, não é possível estabelecer a afinidade apenas para os clientes que realmente estiverem com uma sessão em andamento. Dessa forma, se alguma afinidade for necessária, então todos os usuários estarão em afinidade o tempo todo, diminuindo desnecessariamente o número de requisições passíveis de distribuição.

2. A geração de *hot-spots*: dado que clientes passam a ser direcionados para um servidor em particular, o algoritmo de distribuição de carga acaba se restringindo às novas conexões, não podendo distribuir as já existentes. Em decorrência do uso de proxies e firewalls, diversos clientes podem chegar a um site sob o mesmo endereço IP. Nessa situação, todos serão aglutinados em um único servidor, gerando um hot-spot.

3. Um proxy que utilize um *pool* de endereços IP para efetuar as conexões dos clientes inviabiliza esse algoritmo, pois um mesmo cliente pode chegar ao site, em conexões consecutivas, através de endereços IP distintos e, com isso, poderá ser direcionado a servidores diferentes.

O terceiro ponto, em particular, pode ser inadmissível pois pode fazer com que o controle de sessões simplesmente não funcione para determinados grupos de clientes.

Ambos os produtos permitem contornar esse problema com o uso de cookies. No caso do Alteon Web Switch, o gerente monitora as conexões e insere cookies nas respostas, de modo a que futuras conexões do mesmo cliente possam ser identificadas e redirecionadas para o mesmo servidor. Já no caso do Network Dispatcher, que não processa o fluxo de saída, o cookie, com igual finalidade, deve ser introduzido na resposta pelo servidor.

O uso de cookies acaba com o segundo problema exposto acima mas cria outros. Em ambos os casos, o gerente, ao invés de uma simples reescrita de endereços, passa a ter que fazer uma análise do conteúdo da mensagem HTTP para obter o valor do cookie e, só então, decidir a distribuição. Essa manipulação do conteúdo da mensagem é computacionalmente mais cara que a reescrita de endereços, o que pode comprometer a escalabilidade das ferramentas. Por outro lado, esse mecanismo exige que os browsers dos clientes sejam capazes de processar cookies. Por fim, o primeiro problema, de geração de hot-spots, se mantém.

Para resolver esses problemas, seria necessário: 1) que a reescrita de URLs pudesse ser utilizada como alternativa para o uso de cookies quando preciso; e 2) que essa reescrita, ou cookie, fosse utilizada apenas quando o autor das páginas desejasse estabelecer uma sessão com o cliente. Na verdade, esses problemas ocorrem porque a distribuição de carga é feita de maneira totalmente independente da ferramenta que constrói as páginas. Se a distribuição pudesse ser uma função dessa ferramenta de construção de páginas, então seria possível estabelecer uma sessão apenas no momento desejado e através do método desejado.

2.3.2 Particionamento

O Alteon Web Switch, assim como outras ferramentas, permite que o cluster seja dividido em partições lógicas para diferentes tarefas. Essas "tarefas" são, na verdade, classes de URLs. Por exemplo, é possível especificar que determinado grupo de servidores será responsável por requisições terminadas em '.jpg' e '.gif', e outro grupo será responsável por requisições do tipo '/cgi-bin/*'.

Esse recurso, quando ativo na ferramenta de distribuição, sofre o mesmo problema de desempenho que a alternativa de afinidade controlada por cookies: o processamento do conteúdo da mensagem HTTP, onde está a informação da URL, requer um esforço computacional muito maior que a tradução de endereços. A própria documentação do Alteon Web Switch e do IBM Network Dispatcher notam que o desempenho pode ser muito reduzido em função desse processamento.

É importante notar que existe uma ambigüidade entre o particionamento por tarefas e a manutenção de sessões. Com o mecanismo de afinidade ativo, os usuários sempre retornam ao mesmo servidor e, com o particionamento por tarefas, determinados conjuntos de URLs devem ser providos por conjuntos específicos de servidores. Com isso, um usuário que esteja em sessão e siga um link pertencente a uma partição que não seja provida pelo servidor que o esteja atendendo, ou terá sua sessão interrompida, ou não conseguirá acessar o recurso desejado. A documentação das ferramentas não indica como essa ambigüidade é tratada.

Já particionamentos mais elaborados, como, por exemplo, por categorias de clientes, não são oferecidos por nenhuma das ferramentas. Esse tipo de particionamento seria de difícil implementação por parte das ferramentas de distribuição, uma vez que seria necessário identificar os clientes e usar essa informação na decisão de distribuição. Ou seja, nesse caso, não bastaria que a ferramenta, por exemplo, incluísse um cookie para detectar que um mesmo usuário está retornando ao site, mas seria preciso saber que cliente é esse e qual particionamento ele deve utilizar.

2.3.3 Semântica de Cliente

Os problemas associados à manutenção de sessões e ao particionamento realizados pela ferramenta de distribuição de carga tem uma causa comum: a necessidade de se identificar o cliente que acessa o site. É necessário identificar o cliente para permitir que a afinidade e o particionamento por categorias de clientes possam ser realizados corretamente. Também pode ser necessário identificar o cliente para resolver a ambigüidade entre a manutenção de uma sessão e o particionamento por tarefas. Porém, como vimos, mesmo quando as ferramentas de distribuição incluem uma identificação nas respostas, o resultado ainda não é satisfatório. O motivo é semântico: ao incluir essa identificação nas respostas, só é possível saber que um mesmo *usuário* está retornando ao site, isto é, não é possível saber que *cliente* é esse. Afinal, o conceito de *cliente* é determinado pela ferramenta de construção de páginas dinâmicas—é ela que dá semântica aos *usuários*, transformando-os em *clientes*. E, em momento algum, essa semântica chega à ferramenta de distribuição de carga.

Seria possível, então, solucionar os problemas citados se a ferramenta de distribuição de carga possuísse a mesma programação da ferramenta de construção de páginas, permitindo identificar os clientes que chegam ao site. De fato, o IBM Network Dispatcher possibilita essa codificação, pois disponibiliza uma API de programação do controle da distribuição.

No nosso entender, um aspecto ruim do uso de uma API desse tipo é a falta de encapsulamento. Ou seja, conceitos definidos e manipulados pelo programador do site, dentro da ferramenta de construção de páginas dinâmicas, precisam ser reprogramados na ferramenta de distribuição de carga. Nesse sentido, a programação completa do site passa a ser feita em duas ferramentas distintas, criando uma dependência entre elas. Com isso também perde-se portabilidade, uma vez que um site, caso mude de gerente de distribuição, por exemplo em função de uma mudança de provedor, precisará de recodificação.

Por outro lado, se o controle da distribuição fosse uma função da ferramenta de construção de páginas, seria possível manter o encapsulamento. Com isso, a portabilidade seria maior, pois uma mudança de provedor acarretaria, a princípio, apenas uma mudança nos nomes dos servidores envolvidos, e não na programação da distribuição.

Capítulo 3

Proposta de Distribuição de Carga em Presença de Páginas Dinâmicas

Nesse capítulo nós propomos um novo modelo de distribuição de carga, específico para sites que utilizem páginas dinâmicas. Como vimos na seção 2.3, o fato das ferramentas de distribuição de carga atuarem independentemente das ferramentas de construção de páginas dificulta, ou até mesmo impossibilita, o uso de recursos como, por exemplo, o estabelecimento de sessões, o particionamento do cluster e o uso de métricas de Qualidade de Serviço. Assim, nossa idéia é incorporar o controle da distribuição na ferramenta de construção de páginas, eliminando a ferramenta dedicada à distribuição.

A próxima seção, 3.1, é dedicada à apresentação do modelo de distribuição de carga através da ferramenta de construção de páginas que propomos neste trabalho. Ao longo dessa apresentação, explicamos como a distribuição pode ser realizada, mostramos uma arquitetura que pode ser adotada para operacionalizar o método de distribuição proposto e apontamos as vantagens associadas a esse método. Já na seção 3.2, nós apresentamos a estrutura do protótipo que implementamos para validar as idéias propostas.

3.1 Distribuição por Construção Dinâmica de Ligações (CDL)

Uma ferramenta de construção de páginas dinâmicas atua a cada requisição, montando a página a ser devolvida ao usuário que realizou o acesso. Durante a montagem da página, a ferramenta é responsável pela geração de todo o conteúdo, inclusive os links que ligam a página em questão às demais páginas do site. Os links tipicamente são gerados através de funções de reescrita de URLs da ferramenta e são esses links que, sendo seguidos pelos usuários, permitem a navegação pelo site.

Em um site de páginas estáticas, os links podem ser manipulados para criar particionamentos, ou seja, para distribuir o conteúdo do site entre vários servidores. Para tal, basta que os links utilizados sejam absolutos e contenham o endereço do servidor onde o recurso se encontra. Essa técnica permite a distribuição da carga entre os servidores envolvidos, como apresentamos na seção 2.2.4.

Por outro lado, em um site dinâmico, onde as páginas são geradas a cada acesso, os links também poderiam ser utilizados para criar particionamentos e, da mesma forma, distribuir a carga entre um conjunto de servidores. Porém, nesse caso, a flexibilidade é muito maior pois, como os links também são criados dinamicamente junto com as páginas, é possível gerar links diferentes a cada vez que uma mesma página for construída. Dessa maneira, o particionamento do site passa a ser dinâmico, o que significa que cada usuário pode ser distribuído pelos servidores de um cluster de forma independente dos demais.

Com base nessa observação, nós propomos, então, utilizar a função de reescrita de URLs disponibilizada pela ferramenta de construção de páginas dinâmicas para, através dela, inserir nas páginas links que remetam o usuário a algum servidor do cluster que seja selecionado por um algoritmo de distribuição de carga integrado a essa ferramenta de construção de páginas. Em outras palavras, nós propomos a construção dinâmica de ligações (CDL) entre as páginas do site, de forma a distribuir os usuários em um cluster de servidores HTTP.

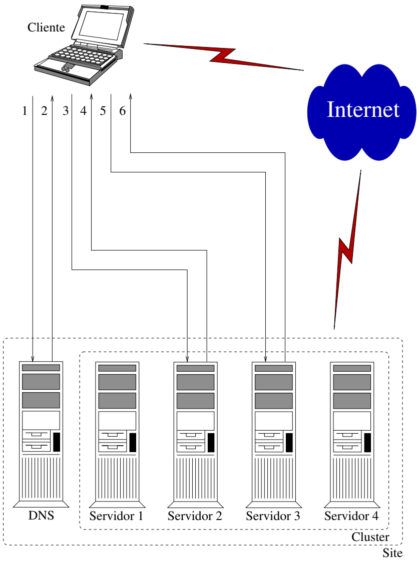
É importante notar que, dessa forma, estamos fazendo uma distribuição *a posteriori*: o usuário primeiro acessa algum servidor e, em resposta a esse acesso, obtém uma página que, caso algum de seus links seja seguido, o leva a um servidor possivelmente diferente do originalmente contactado. Ou seja, a cada acesso de um usuário ao site, é escolhido o servidor que esse usuário utilizará para efetuar o acesso subseqüente, mas a chegada do usuário a esse servidor, caso ocorra, só virá a acontecer em algum outro momento no futuro.

Neste trabalho, nós exploramos o método de distribuição de carga CDL como uma técnica de *load sharing*, uma vez que os algoritmos de distribuição utilizados não tentam equilibrar as cargas entre os servidores, mas buscam, como veremos no capítulo 5, apenas evitar que as diferenças entre essas cargas sejam muito grandes. Como bem sintetizado por Goldszmidt e Hunt, existem duas técnicas de distribuição de carga entre servidores: *load balancing* e *load sharing*. Enquanto a técnica de load balancing tenta equilibrar a carga entre os servidores, a técnica de load sharing busca apenas atenuar os eventuais desbalanceamentos entre os servidores. As estratégias de load balancing tipicamente consomem muito mais recursos que as de load sharing, e o custo desses recursos muitas vezes superam seus potenciais benefícios [Goldszmidt e Hunt, 1997, página 14].

A primeira observação a ser feita sobre o método de distribuição CDL é sobre como iniciá-lo. Para que a ferramenta de construção de páginas dinâmicas possa montar uma página que venha a distribuir um usuário em seu próximo acesso, essa página precisa ter sido pedida por esse usuário em um primeiro momento. Dessa forma, a ferramenta de construção de páginas só tem controle sobre a distribuição da segunda requisição em diante de cada usuário. A questão, então, é: como distribuir o primeiro acesso de cada usuário que chega ao site?

Nós propomos uma arquitetura para operacionalizar o método CDL que se baseia na divulgação dos endereços de todos os servidores do cluster através de RR-DNS. Com isso, o primeiro acesso de cada usuário ao site é governado pelo sistema de de resolução de nomes, levando a uma distribuição inicial dos clientes pelos servidores do cluster. Apesar dessa distribuição ser sabidamente não ideal, a idéia é que ela possa ser alterada pelo algoritmo de distribuição de carga incorporado na ferramenta de construção de páginas. Isto é, cada usuário, inicialmente, é designado a um servidor qualquer, através do sistema de resolução de nomes. Conectando-se a esse servidor, o usuário obtém, em resposta à sua requisição, a página desejada, com os links manipulados para que a próxima requisição seja distribuída para um outro servidor qualquer do cluster, dessa vez determinado pelo algoritmo de distribuição incorporado na ferramenta de construção de páginas. Nessa arquitetura, todos os servidores do cluster atuam como espelhos, possuindo a mesma tarefa de responder às requisições dos usuários. Uma diagrama esquemático dessa arquitetura pode ser visto na figura 3.1.

É importante notar que nessa arquitetura dois algoritmos diferentes distribuem requisições simultaneamente. O primeiro algoritmo é o RR-DNS, que age antes que o usuário chegue ao site, determinando que servidor irá recebê-lo e retornar a resposta à requisição. O segundo algoritmo é o que está incorporado na ferramenta de construção de páginas, que só entra em atividade uma vez que o usuário tenha chegado ao site; e, dessa forma, só tem controle sobre a distribuição da segunda requisição em diante. Com isso, o processo de distribuição possui, sob o ponto de vista dos acessos de um único usuário, duas etapas distintas: a primeira, governada pelo RR-DNS, atua sobre o primeiro acesso do usuário ao site, enquanto a segunda, governada pelo algoritmo de distribuição da ferramenta de



1. Para realizar o primeiro acesso, o cliente requisita a tradução do nome em um endereço.
2. O cliente obtém o endereço de algum servidor do cluster em resposta.
3. Usando o endereço recebido, o cliente efetua o primeiro acesso ao site.
4. O servidor contactado responde a requisição com a página correspondente.
5. No segundo acesso, seguindo um link da página, o cliente chega a outro servidor.
6. O novo servidor responde a requisição com a página correspondente.

Figura 3.1: Diagrama esquemático da arquitetura de distribuição por CDL.

Classificação Estendida dos Agentes de Distribuição de Carga	
Externos	Usuários
	Browsers
	Proxies
Internos	Tradução de Endereços
	Manipulação de Pacotes
	Redirecionamento de Requisições
	Particionamento
	Construção Dinâmica de Ligações

Tabela 3.1: Classificação Estendida dos Agentes de Distribuição de Carga.

construção de páginas, atua sobre os demais acessos desse usuário.

A arquitetura que propomos é híbrida pois utiliza dois diferentes agentes de distribuição simultaneamente. O RR-DNS já foi classificado e analisado no capítulo 2. O segundo agente responsável pela distribuição é a construção dinâmica de ligações (links) entre as páginas do site. Uma vez que esses links são criados como parte do processo de construção das páginas dinâmicas, cada usuário pode receber um conjunto diferente de links, levando à distribuição das requisições no cluster. Assim, a classificação proposta no capítulo 2 pode ser estendida, como apresentamos na tabela 3.1, para comportar esse novo agente responsável pela efetivação da distribuição.

O método CDL compartilha das vantagens de ferramentas de distribuição de carga distribuídas:

- Não introduz um único ponto de falha. Qualquer servidor que apresente uma falha pode deixar o cluster sem interromper as conexões sendo tratadas pelos demais servidores.

- A adição de um novo servidor aumenta não só a capacidade do site, como também a própria capacidade de distribuição, o que se reflete em uma verdadeira escalabilidade, que não é alcançada nas ferramentas onde a distribuição é centralizada.

- Pode apresentar melhor relação custo-benefício. No caso de clusters pequenos, não é necessário adquirir um equipamento (ou software) dedicado para a distribuição de carga.

Além dessas características, a distribuição feita pela ferramenta de construção de páginas nos permite estabelecer a afinidade entre um cliente e um servidor de forma relativamente simples, bastando, para tal, que os links internos das páginas que forem construídas para esse cliente se referenciem sempre ao servidor que esteja mantendo o contexto da sessão. Como o controle da afinidade é feito pela própria ferramenta que dá semântica aos clientes, e esse controle ainda é feito de forma individual para cada um desses clientes, nós conseguimos criar um mecanismo de afinidade adequado, que funciona para todos os clientes e é estabelecido apenas quando necessário.

Em particular, esse método de controle da afinidade entre um cliente e um servidor não compartilha de nenhum dos problemas presente nas demais ferramentas de distribuição de carga: em primeiro lugar, como o estabelecimento de uma sessão com o usuário é uma função da ferramenta de construção de páginas, esta só precisa criar a afinidade entre um usuário e um servidor quando a programação da página requisitar a sessão. Não havendo mais a decisão binária, que ou mantinha todos os usuários em sessão ou não mantinha nenhum, o desempenho da distribuição pode melhorar pois não haverá mais uma diminuição desnecessária do número de requisições passíveis de distribuição.

Em segundo lugar, o problema de geração de hot-spots pode ser totalmente eliminado, bastando, para isso, que a ferramenta procure criar cada sessão no no servidor mais adequado. Através dessa política, mesmo em situações extremas, como um site que estabeleça sessões com todos os clientes e desde o primeiro acesso, as sessões podem ficar equilibradamente distribuídas entre todos os servidores. Nesse caso extremo, os inúmeros clientes provenientes de um proxy, durante um determinado TTL, são inicialmente direcionados ao mesmo servidor. Entretanto, a sessão pode ser criada em algum outro servidor que possua uma carga menor, bastando que a página gerada tenha seus links escritos de forma a direcionar o próximo acesso do usuário para esse novo servidor que mantém o contexto da sessão. Além dessa distribuição realizada no momento da criação de cada sessão, seria possível, no caso de ainda haver desbalanceamento de carga, inclusive, migrar a sessão do usuário, transferindo seu contexto para outro servidor.

Em terceiro lugar, como a afinidade é mantida através dos links das próprias páginas que chegam ao cliente, o fato desse cliente estar acessando a rede diretamente, ou através de um proxy, usando sempre o mesmo endereço IP, ou endereços alternados, não influencia em nada o controle da afinidade. Dessa forma, a afinidade funciona para todos os clientes, independentemente da forma de acesso à rede. Em particular, todos esses recursos associados ao controle da afinidade independem de um gerente central que decida que requisições devem ser tratadas por quais servidores, o que poderia ser um gargalo do sistema.

Com a decisão distribuída, o próprio controle da manutenção de sessões possui escalabilidade, pois aumenta de capacidade com o aumento do número de servidores. Essa característica torna-se mais relevante quanto maior for o processamento necessário no controle de sessões. Cabe lembrar que a documentação das ferramentas de distribuição de carga comerciais apontam que a simples identificação de um usuário através do conteúdo de um cookie é uma operação dispendiosa, podendo, inclusive, comprometer a escalabilidade do mecanismo de distribuição. Nessa linha, um processamento ainda maior, que viabilizasse a setorização do cluster, ou a migração de sessões, por exemplo, só agravaria esse problema.

A questão do particionamento do cluster não é bem resolvida por nenhuma das ferramentas analisadas no capítulo 2. Utilizando CDL, entretanto, o particionamento pode ser livremente arbitrado, assim como acontece com a manutenção de sessões. Por exemplo, é possível definir sub-conjuntos de servidores do cluster e associá-los a diferentes conjuntos de URLs. Assim, no momento de escrever um link em uma página, é possível escolher um servidor entre os servidores do sub-conjunto responsável pela URL em questão. Ainda de maneira ilustrativa, um cliente, uma vez identificado, pode passar a ter os links internos de suas páginas criados de modo a mantê-lo em um determinado setor do cluster, reservado para clientes de sua categoria.

As diversas características que expomos e analisamos parecem indicar que a distribuição de carga controlada pela ferramenta de construção de páginas dinâmicas pode atender aos requisitos de aplicações sofisticadas, como as de comércio eletrônico e as voltadas para Qualidade de Serviço. Um fator determinante, entretanto, para essa viabilização, é que o modelo CDL apresente um bom desempenho na distribuição da carga.

Uma parte importante do nosso trabalho é o desenvolvimento de um protótipo que nos permita validar as propostas que apresentamos. O protótipo nos permite não apenas analisar o desempenho, mas também explorar diversas características, como, por exemplo, a escalabilidade e a adequação de diferentes algoritmos de distribuição de carga. Nós apresentamos a arquitetura desse protótipo na próxima seção.

3.2 Arquitetura do Protótipo Implementado

O protótipo que construímos utiliza a ferramenta de construção de páginas dinâmicas CGILua [Hester et al., 1998b,a, 1997]. Essa ferramenta disponibiliza a linguagem de *scripting* Lua [Ierusalimschy et al., 1998, 1996; de Figueiredo et al., 1996] para a construção das páginas. A ferramenta CGILua pode ser utilizada como um programa CGI, sendo executada a cada nova requisição, ou como uma extensão do servidor HTTP. Como apresentado em [Hester, 1999], o CGILua pode atuar como um módulo do servidor Apache [Corporation, 1999a] e como uma extensão do servidor IIS [IIS, 1998].

O CGILua, como outras ferramentas, permite que uma página seja totalmente construída através de programação. Ou seja, um script Lua pode ser executado e a saída produzida por esse script é a página a ser enviada em resposta à requisição. Também como outras ferramentas, o CGILua permite a construção de páginas mescladas, isto é, páginas que contenham código HTML misturado com código Lua. Essas páginas mescladas, chamadas *templates*, são processadas pela ferramenta de uma forma diferente. Esse processamento, basicamente, consiste em manter as partes HTML e executar as partes Lua, que são substituídas pelo resultado da execução.

Em ambos os casos, scripts ou templates, o programador dispõe de uma função de construção de URLs, chamada `mkurl`. Essa função recebe dois parâmetros, o caminho para o script que deve ser chamado através do link e uma tabela contendo informações a serem passadas para esse script, e atende a dois objetivos distintos. O primeiro objetivo é permitir a criação de URLs que independam da instalação física do CGILua. Isso é conseguido com a inserção do prefixo necessário na URL para que o CGILua volte a ser chamado para tratar o script em questão. O segundo objetivo é simplificar a passagem de informações entre os scripts. Essa passagem de parâmetros é feita através de informações que são inseridos pela função na URL gerada. Tais informações normalmente são utilizadas quando se deseja estabelecer uma sessão sem usar cookies.

A função de construção de URLs é o ponto onde propomos vincular o controle da distribuição de carga. Nós reimplementamos a função `mkurl` para que ela se conecte a um gerente local responsável pela distribuição da carga e obtenha deste gerente o servidor a ser utilizado no próximo acesso do cliente, transformando o que antes era uma URL relativa em uma URL absoluta, direcionada a esse servidor determinado pelo gerente de carga. Nós optamos pela criação de um gerente local externo à ferramenta uma vez que adotamos, para a construção do protótipo, a versão CGI do CGILua. Dessa maneira, se o gerente fosse parte da ferramenta, ele terminaria sua execução junto desta, ao término da construção de cada página. É importante notar que, no caso de utilizarmos uma versão persistente, isto é, acoplada ao servidor, esse gerente poderia ser parte da implementação da ferramenta.

No nosso protótipo, cada servidor possui, então, um gerente de carga local, responsável pela alocação de servidores nos links internos das páginas geradas. A informação de quais servidores compõem o cluster é obtida pelo gerente no início de sua execução através de um arquivo de configuração. O gerente de carga é, então, contactado por todos os processos CGILua que, em execução no servidor, precisem construir links internos para as páginas sendo criadas.

Essa comunicação entre os processos CGILua e o gerente de carga é feita através das primitivas de comunicação da biblioteca ALua [Ururahy e Rodriguez, 1999]. Resumidamente, a biblioteca ALua permite que diferentes agentes programados na linguagem Lua troquem mensagens. Como Lua é uma linguagem interpretada, tais mensagens podem ser programas Lua que serão executados pelo destinatário da mensagem. Esse mecanismo facilita e flexibiliza muito a comunicação, uma vez que um agente pode enviar para outro um trecho de código que, quando executado, obtém no ambiente de sua execução todas as informações desejadas e as envia de volta ao servidor de origem.

Diversos algoritmos de distribuição de carga podem, em princípio, ser implementados no gerente local. Dependendo do algoritmo implementado, pode ser necessário haver uma comunicação entre

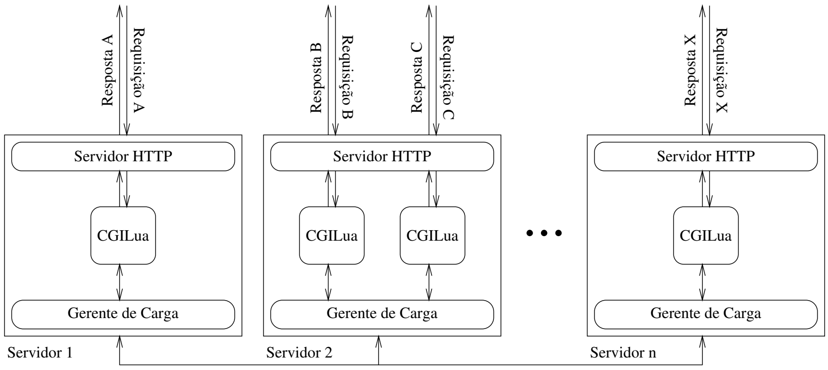


Figura 3.2: Diagrama esquemático da arquitetura do protótipo.

os gerentes. Por exemplo, um algoritmo que procure distribuir as requisições de forma diretamente proporcional à capacidade livre corrente de cada servidor necessita que os servidores troquem as informações de carga regularmente. Além disso, a comunicação entre gerentes também é necessária se usarmos algoritmos do tipo *heartbeat* para permitir a identificação de servidores parados e impedir que novos clientes sejam direcionados a esses servidores. Essa comunicação, em nosso protótipo, quando existente, também é feita através das primitivas de comunicação da biblioteca ALua. Na figura 3.2, nós apresentamos um diagrama esquemático da arquitetura do protótipo.

Capítulo 4

Estrutura de Teste

Nosso objetivo, nesse capítulo, é apresentar a estrutura de teste e de análise que montamos para explorar e avaliar o método de distribuição CDL. Na seção 4.1, nós apresentamos os equipamentos e os programas que utilizamos nos testes realizados. Suscintamente, os equipamentos se resumem a um conjunto de computadores, conectados por um switch, enquanto os programas envolvidos são, basicamente, o sistema operacional desses computadores, o servidor HTTP, o protótipo, composto pela versão modificada do CGILua e o gerente de carga local, e a ferramenta responsável por gerar a carga de acessos sobre os servidores.

A discussão de maior relevância na seção 4.1 é sobre a ferramenta Surge (*Scalable URL Reference Generator*) [Barford e Crovella, 1997], que utilizamos para gerar a carga. Essa relevância se deve à complexidade da ferramenta, que simula uma comunidade de usuários acessando um site, e ao fato de termos que adaptar essa ferramenta, incorporando novos conceitos, para que ela se adequasse ao CDL.

Em seguida, na seção 4.2, nós apresentamos os métodos que utilizamos para analisar os resultados dos testes. Diversas instrumentações foram inseridas na ferramenta de construção de páginas e na ferramenta que gera a carga de acessos para permitir a construção de gráficos sobre diferentes aspectos do desempenho da distribuição. Nessa seção nós explicamos o significado e a aplicabilidade dos gráficos mais utilizados nas análises apresentadas no capítulo 5. Nós apresentaremos os demais gráficos utilizados no capítulo 5 no momento de sua utilização.

4.1 Ambiente de Teste

Todos os testes de distribuição de carga foram realizados no cluster do laboratório do Departamento de Informática da PUC–Rio. Esse cluster é composto por 32 computadores PC IBM, conectados por um switch de 10 Mbit/s, formando uma rede local que não recebe tráfego externo. Dos 32 servidores, apenas um se encontra conectado à rede externa. Todos executam o sistema operacional Linux e sincronizam seus relógios através de NTP, garantindo precisão de milisegundo [Lills, 1994]. Uma especificação mais completa do cluster encontra-se no apêndice A.1. Durante os testes realizados, o cluster se encontrava em modo exclusivo, garantindo que todo o processamento dos servidores e o tráfego de rede eram decorrentes somente do teste em andamento. Por fim, o servidor HTTP utilizado nos testes foi o Apache, versão 1.3.12 [Laurie e Laurie, 1999].

Para testar o método CDL, nós simulamos a arquitetura apresentada na seção 3.1, na qual os endereços de todos os servidores do cluster são divulgados através de RR-DNS. Com isso, temos duas etapas de distribuição: RR-DNS governando o primeiro acesso de cada usuário e o nosso algoritmo de distribuição de carga, incorporado na ferramenta de construção de páginas, controlando a distribuição dos demais acessos.

Para simular a existência de proxies e firewalls que aglutinem conjuntos de usuários, conectando-os à rede, nós assumimos que os usuários que acessam o site encontram-se distribuídos por um conjunto de "pontos de conexão". Um ponto de conexão representa um conjunto de computadores que utilizam um mesmo servidor de DNS para efetuar as traduções de nomes em endereços IP. Ou seja, uma tradução feita pelo servidor de DNS do ponto de conexão junto ao RR-DNS do site pode atender a diferentes usuários espalhados por vários computadores.

A simulação dos usuários foi feita através da ferramenta Surge (*Scalable URL Reference Generator*) [Barford e Crovella, 1997]. Essa ferramenta simula uma comunidade de usuários acessando um único servidor Web. Para que a carga gerada pela ferramenta sobre o servidor se assemelhe a uma carga real, essa ferramenta incorpora diferentes modelos estatísticos que combinam com as observações empíricas de tamanho de arquivos, tamanho das requisições, popularidade das páginas, número de arquivos embutidos, localidade temporal, tempo gasto pelo usuário entre duas requisições, e, finalmente, número de requisições consecutivas de um usuário feitas a um site.

Os autores do Surge chamam esse conjunto de todos os acesso de um usuário a um site de "sessão". Nós não adotaremos essa nomenclatura pois, para nós, o termo sessão significa a manutenção de um contexto entre acessos consecutivos de um mesmo usuário. Assim, chamaremos de "caminho" seqüência completa de todos os acessos feito por um usuário a um site. Em particular, uma sessão entre um usuário e o site pode ser estabelecida durante todo o caminho desse usuário, em apenas parte dele, ou até mesmo não ser estabelecida em momento algum.

O conceito de caminhos é de grande importância para o método de distribuição de carga CDL. Afinal, o CDL distribui os acessos de cada cliente através das páginas que compõem seu caminho, fazendo com que cada novo acesso possa levar o usuário a um servidor diferente. Desse modo, em uma situação hipotética, onde todo usuário que acesse o site faça uma única requisição, ou seja, uma situação na qual todos os caminhos tenham comprimento um, o método CDL não conseguirá efetuar distribuição alguma da carga.

Trabalhos que estudam as distribuições dos comprimentos dos caminhos indicam que uma grande quantidade de usuários, de fato, realiza um único acesso ao site sendo visitado [Huberman et al., 1998; Arlitt, 2000]. Essa observação poderia ser crítica para a nossa proposta. Porém, como o Surge incorpora o modelo de comprimento de caminhos apresentado em [Huberman et al., 1998], nós pudemos constatar, através dos testes realizados, que, apesar do grande número de caminhos curtos, o método CDL efetivamente consegue distribuir uma parcela significativa das requisições, alcançando um melhor desempenho quando comparado com outras ferramentas de distribuição de carga.

O Surge é preparado para gerar a carga simulada sobre um único servidor. Para gerar essa carga, inicialmente é preciso construir o site sobre o qual o Surge efetuará as requisições. Esse site, e um conjunto de arquivos que definem diversas propriedades das requisições a serem efetuadas, são construídos através da execução de uma série de programas que compõem a ferramenta e que podem ser parametrizados para configurar diversas características do teste, como o tamanho do site e ajustes das distribuições utilizadas. A partir desse ponto, instala-se o site em um servidor e os programas que executarão as requisições, juntamente com seus arquivos de configuração, em um ou mais clientes. Os programas nos clientes, uma vez executados, entram em um processo de geração de requisições ao servidor, processo esse que é guiado pelos arquivos de configuração.

O volume da carga gerada é dado indiretamente, através do número de equivalentes de usuários (*User Equivalents*, ou UE). Cada UE é implementado por uma linha de execução (*thread*) que se alterna entre fazer requisições e ficar parada. As sucessivas requisições realizadas por cada UE são sempre parte do caminho de um usuário. Uma vez terminado um caminho, o UE inicia outro, representando um novo usuário que chega ao site. Nesse sentido, um único UE, ao longo de todo o período de teste, pode fazer o papel de inúmeros clientes reais.

O Surge adota um modelo no qual cada requisição feita pelo usuário pode ser composta por arquivos *base*, *embedded* ou *loner*. Esse modelo leva em consideração o fato de que, ao requisitarmos uma página, nós estamos, muitas vezes, requisitando um objeto, que é composto por uma página principal e um conjunto de páginas embutidas. A página principal, representada por arquivos do tipo *base*, é a página diretamente referenciada pela URL, enquanto as páginas embutidas, representadas por arquivos do tipo *embedded*, são recursos necessários para desenhar a página principal, como *frames*, imagens, sons, e outros. Naturalmente, às vezes uma requisição realmente se refere a uma única página, nesse caso representada por arquivos do tipo *loner*, como por exemplo, uma página HTML bem simples, ou um arquivo de dados qualquer.

Muito embora o modelo adotado simule uma estrutura real de dependência entre arquivos, todos os arquivos que compõem o site criado pelo Surge contém apenas um caracter, repetido um determinado número de vezes para dar o tamanho correto a cada arquivo. Ou seja, a única característica relevante dos arquivos gerados é o tamanho. Os demais fatores, como o tipo assumido por cada arquivo, as relações de interdependência, a freqüência de acesso e outros, são especificados tão somente através dos arquivos de configuração utilizados pelos clientes.

Porém, em nosso caso, uma simulação condizente com uma situação de uso real precisa considerar a distribuição dos clientes entre o conjunto de servidores obtida através do RR-DNS, respeitando suas características, como aglutinação de clientes nos pontos de conexão e tempo de TTL e, principalmente, atender à segunda etapa da distribuição, especificada através das URLs internas das páginas, ao longo de todos os acessos subseqüentes de cada cliente. Essas necessidades não são atendidas diretamente pelo Surge pois, por um lado, o acesso direto a um único servidor impede o uso da distribuição via RR-DNS, e, por outro, o conteúdo insignificativo dos arquivos do site não permite a obtenção do próximo servidor a ser acessado. Para a efetivação dos testes, então, nós procedemos com as modificações necessárias na ferramenta Surge.

Seguindo a ordem temporal na qual cada modificação atua durante a execução do Surge, o primeiro passo é a adoção de um modelo de distribuição dos clientes pelos pontos de conexão. Nós adotamos uma distribuição *Zipf* [Zipf, 1949], na qual a probabilidade P de um cliente pertencer ao i-ésimo ponto de conexão é dada pela equação abaixo:

$$P(i) = k/i^r$$

Nessa equação, k é um fator de normalização, enquanto r permite moldar a distribuição. No caso da distribuição dos clientes, seria possível variar desde uma distribuição homogênea, até o limite onde todos os clientes se aglutinam em um único ponto de conexão. Assim, através desse modelo, podemos estipular uma grande variação dos clientes pelos pontos de conexão, nos aproximando de observações empíricas. De fato, Arlitt e Williamson [1996] citam, como um invariante na caracterização da carga real imposta a sites Web, que os servidores são acessados por clientes provenientes de milhares de domínios (pontos de conexão) diferentes e que apenas 10% desses domínios são responsáveis por, no mínimo, 75% das requisições realizadas.

Atendendo a essa distribuição, o cliente Surge inicialmente sorteia um ponto de conexão e se conecta ao gerente de DNS, ao qual fornece a identificação desse ponto escolhido e recebe, em resposta, o endereço do servidor a ser acessado. Esse gerente de DNS representa toda a rede de servidores de DNS que existe entre os clientes e o site. Para tal, esse gerente armazena a tradução corrente para cada diferente ponto de conexão, fazendo o controle do TTL, e elegendo novas traduções, na medida do necessário, por meio de um Round Robin nos servidores do cluster.

De posse do servidor a ser utilizado, o Surge estabelece a conexão e obtém um objeto que, como já exposto, pode ser composto por diversos arquivos. Esse passo é realizado sem qualquer alteração. Porém, como a base da URL utilizada na obtenção do arquivo principal do objeto referencia o CGILua, este pode vincular na página o endereço do servidor a ser utilizado no próximo acesso do caminho corrente. Assim, uma segunda alteração no Surge foi feita para obter esse endereço e armazená-lo para o próximo acesso.

Esse ciclo de requisição e obtenção de novo servidor segue até que o caminho seja terminado. Nesse momento, o Surge volta ao ponto inicial, elegendo um novo ponto de conexão, se conectando ao gerente de DNS, e iniciando um novo caminho de um novo usuário que chega ao site. Com esse ciclo, nós simulamos o comportamento típico dos browsers, que realizam a tradução do nome do servidor a ser acessado em um endereço IP uma única vez. O que faz com que o browser mude de servidor a cada acesso é o fato dos links internos utilizarem o endereço IP, e não o nome, do servidor para o qual o cliente deve ser direcionado.

4.2 Métodos de Análise

Para analisar os testes de desempenho, nós utilizamos as informações constantes dos *logs* do servidor HTTP e do Surge, além de outras, fornecidas por instrumentações inseridas no próprio Surge, no gerente de DNS e nos gerentes de carga locais. Os logs do servidor HTTP e do Surge disponibilizam informações detalhadas sobre cada requisição, fornecendo o tempo de início, URL requisitada, tamanho da resposta e tempo de término. O gerente de DNS nos fornece o número de traduções pedidas pelos clientes, e a distribuição dos clientes pelos pontos de conexão. Já os gerentes de carga registram informações detalhadas sobre o estado do servidor, coletadas de `/proc`, como, por exemplo, a taxa de utilização da CPU, o espaço de memória utilizado e número de conexões TCP ativas.

Em primeiro lugar, é importante estabelecermos uma referência para comparar os desempenhos alcançados pelos diferentes métodos/algoritmos de distribuição. Para isso, nós realizamos uma série de testes para determinar o desempenho máximo alcançado por um único servidor. Dessa maneira, é possível analisar a escalabilidade dos métodos/algoritmos de distribuição que iremos apresentar. O gráfico da figura 4.1 mostra a evolução do desempenho de um único servidor frente ao aumento da carga. Cada teste tem a duração de 15 minutos, sendo que os clientes que geram a carga sobre o servidor se encontram distribuídos em 25 servidores do cluster.

Os valores de desempenho apresentados são medidos após os 10 minutos iniciais de cada teste. Todas as medidas de desempenho que iremos mostrar ao longo deste trabalho não consideram os 10 minutos iniciais. Esse tempo é considerado um transiente, após o qual assumimos que o Surge está atuando em estabilidade. Além disso, como iremos ver, eventuais sobrecargas, quando ocorrem, iniciam dentro deste período transiente.

Como mostra o gráfico, o melhor desempenho é atingido aos 76 UE. Nesse teste em particular, a taxa de transferência do servidor foi de 64.9 Kb/s.

É importante observar que este é um valor aproximado do desempenho máximo do servidor. Afinal, a carga de teste gerada pelo Surge é, como dissemos, variável em diferentes aspectos para simular a carga gerada por uma comunidade real de usuários. Ou seja, em duas execuções consecutivas de um mesmo teste, as cargas geradas pelo Surge serão diferentes, podendo exigir mais ou menos do servidor. Apesar dessa diferença, a comparação de desempenho utilizando esse tipo de carga é importante pois mostra mais fielmente os resultados que seriam obtidos em um site real.

Com as informações coletadas podemos analisar com detalhes o comportamento dos servidores ao longo de todo o teste. Na figura 4.2, nós apresentamos o acompanhamento de um teste no qual

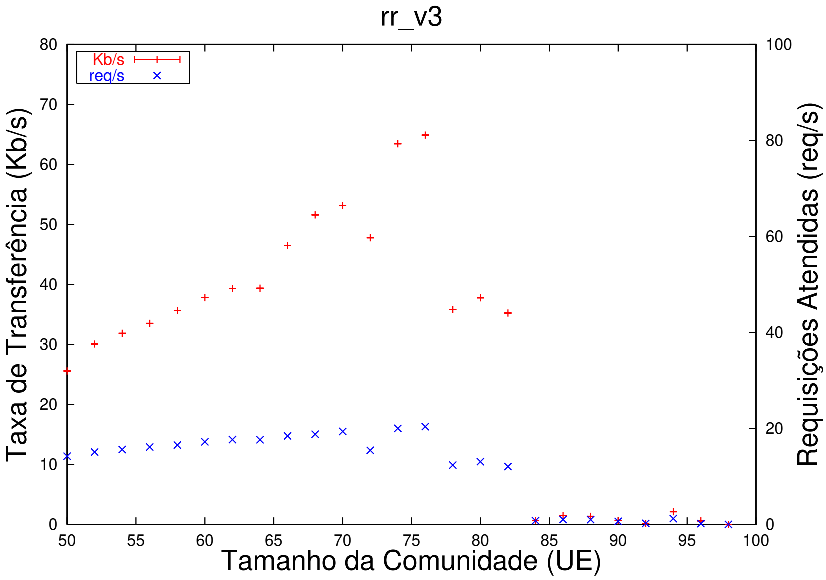


Figura 4.1: Evolução do desempenho de 1 servidor.

6 servidores foram sujeitos a uma carga de 200 UE, gerada por 25 outros nós do cluster, durante 15 minutos. Para completar os parâmetros de teste, foram utilizados 1000 pontos de conexão, o fator r, da distribuição dos usuários pelos pontos de conexão, utilizado foi igual a 1.07508, mantendo similaridade com distribuições reais [Arlitt, 2000] e, por fim, o TTL adotado foi de 240 segundos.

Como nosso objetivo, nesse ponto, é explicar os gráficos que utilizamos nas análises, nenhum algoritmo de distribuição foi utilizado na ferramenta de construção de páginas. O único responsável pela distribuição existente é o Round Robin DNS, ou seja, nesse teste só há a primeira etapa de distribuição, a segunda etapa mantém os usuários nos servidores a que chegaram no primeiro acesso. Isso significa que os gráficos refletem o desempenho do RR-DNS, como apresentado na seção 2.2.1, no nosso ambiente de teste. O desempenho médio do site após os 10 minutos iniciais, nesse teste, foi de 62.3 requisições atendidas por segundo (req/s) e 248.7 Kbytes transferidos por segundo (Kb/s).

Na figura 4.2, o gráfico do 'número de requisições em andamento' mostra o número de requisições que estão sendo servidas por cada servidor a cada instante. O gráfico do 'tamanho total das requisições' apresenta o volume total dos dados que devem ser transferidos para os clientes em um dado momento, em resposta às requisições sendo servidas, para todos os servidores. O gráfico do 'tempo de resposta das requisições' é um gráfico discreto que mostra o tempo gasto para responder cada requisição feita a cada servidor. Por fim, o gráfico do 'número de requisições recebidas por segundo' mostra o número de novas requisições estabelecidas com cada servidor durante cada segundo do período de teste.

O bom desempenho do algoritmo de distribuição, nesse caso apenas o RR-DNS, indicado pelas taxas de requisições atendidas e Kbytes transferidos por segundo, pode ser visualizado nos gráficos. Através do gráfico que mostra o número de requisições sendo atendidas por cada servidor em um dado instante de tempo podemos ver a distribuição dos usuários nos servidores, nesse caso bem homogênea. Já o volume de informações que deve ser transmitido pelo servidor em resposta às requisições possui uma distribuição bem diferente, como mostrado no gráfico do tamanho total das requisições em andamento, refletindo uma medida empírica de que os tamanhos dos arquivos variam por uma grande faixa. Essa variação dos tamanhos dos arquivos poderia desequilibrar a distribuição homogênea das requisições, dado que algumas dessas requisições levam mais tempo para serem respondidas do que outras. No teste em questão, entretanto, esse desequilíbrio não chega a ocorrer pois, como pode ser visto no gráfico dos tempos de resposta das requisições, mesmo as maiores requisições levam poucos segundos para serem respondidas. O último gráfico, do número de requisições recebidas por segundo, também mostra uma boa distribuição das requisições que chegam aos servidores.

Uma situação diferente pode ser vista nos gráficos da figura 4.3. Esses gráficos retratam um segundo teste de distribuição via RR-DNS, dessa vez aumentando-se a carga de 200 para 220 UE e mantendo todos os demais parâmetros de teste. O desempenho médio do site, nesse caso, caiu para 24.1 req/s e 83.9 Kb/s. Como podemos ver no gráfico do número de requisições em andamento, nesse teste ocorre um grande desbalanceamento de carga entre os servidores, com um único servidor acumulando um grande número de requisições. Especificamente, o servidor `s4`, imediatamente após o primeiro $1/3$ do período de teste, começa a acumular requisições, chegando, quase 2 minutos depois, próximo a 200 requisições simultâneas.

Com a sobrecarga, todas as requisições, até mesmo as mais simples, levam um tempo significativo para serem respondidas, como pode ser observado no gráfico que apresenta o tempo de resposta das requisições feitas ao site. Essa demora nas respostas colabora para que seja mantida a longa fila de requisições no servidor sobrecarregado, retratada no gráfico que apresenta o número de requisições em andamento. Já no gráfico do tamanho total das requisições, vemos que o volume de dados a serem transferidos se mantém elevado durante todo o período de sobrecarga.

O problema é que o servidor de DNS desconhece o estado de cada servidor HTTP e continua

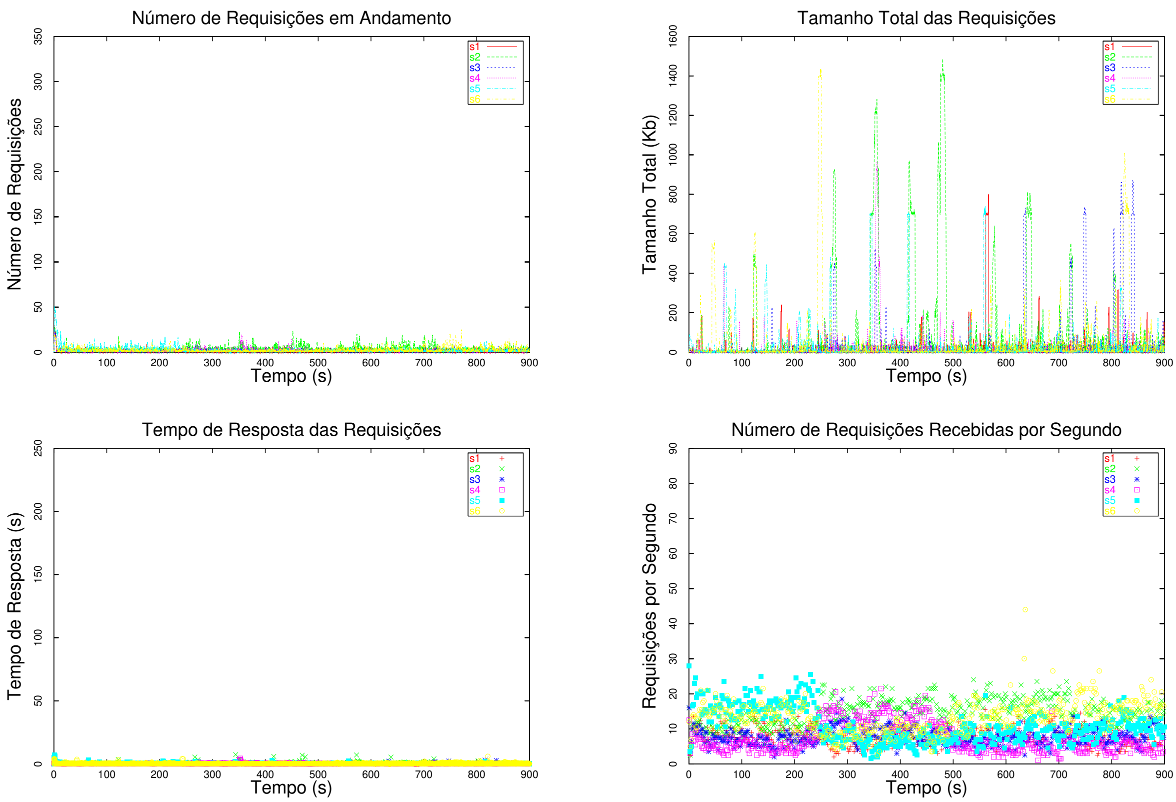


Figura 4.2: Teste de carga de 200 UE sobre 6 servidores HTTP, utilizando RR-DNS.

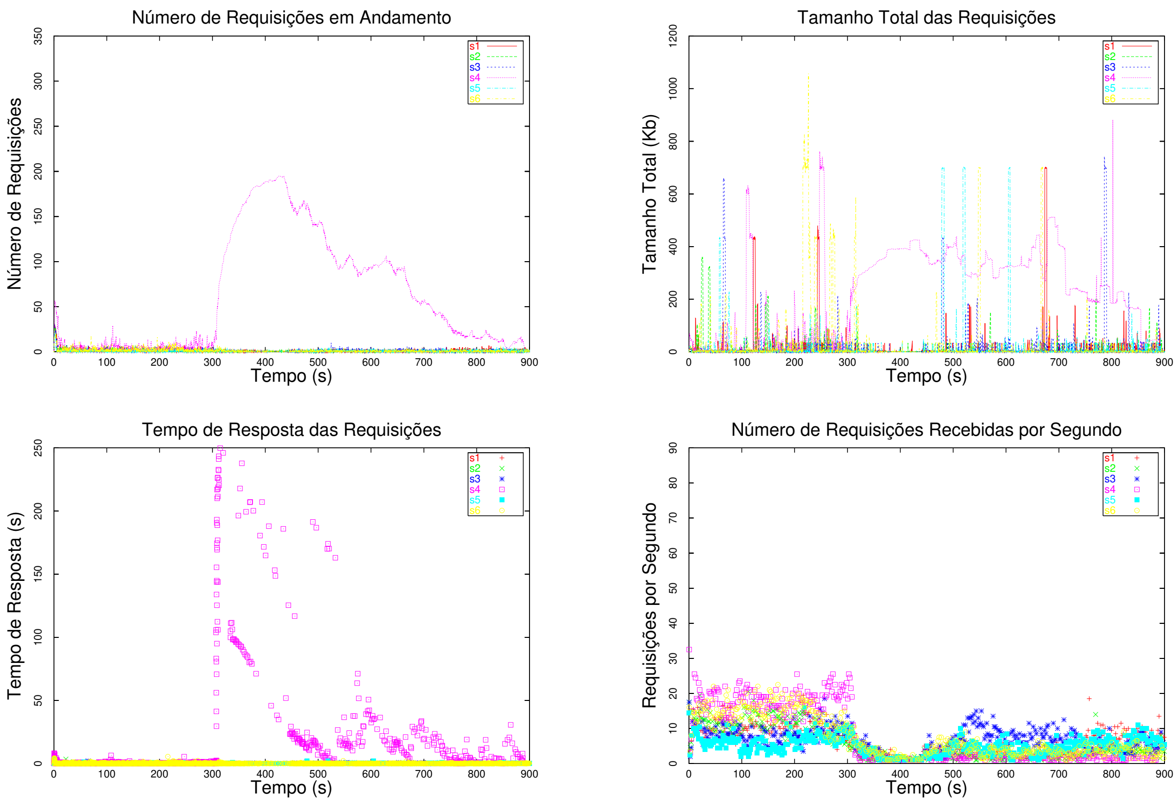


Figura 4.3: Teste de carga de 220 UE sobre 6 servidores HTTP, utilizando RR-DNS.

a fornecer os endereços de todos esses servidores, em Round Robin. Com isso, todos os clientes, eventualmente, entram na fila de espera por um recurso localizado no servidor sobrecarregado. A formação dessa fila, ao mesmo tempo em que coloca o servidor sobrecarregado em uma situação, mantida a carga, irreversível, diminui a taxa de requisições aos demais servidores, garantindo um bom desempenho por parte destes. A diminuição da taxa de requisições recebidas por segundo é sensível e pode ser vista no gráfico correspondente, enquanto o bom desempenho dos demais servidores permeia os gráficos de número, tamanho e tempo de resposta de requisições, apresentando ótimas taxas em todos.

A causa de um único servidor se manter nesse estado é o fato de que, ao entrar em sobrecarga, o servidor, em nosso ambiente de teste, consome toda a memória RAM disponível, passa a utilizar o espaço de swap e, então, começa a arrastar. Nós podemos atestar esse fato através das informações colhidas pelas instrumentações. Na figura 4.4, nós apresentamos quatro novos gráficos que retratam bem a situação de arrasto no teste: a utilização de memória e de CPU tanto do servidor sobrecarregado quanto de um dos servidores não sobrecarregados.

O gráfico de utilização de memória mostra a quantidade de memória RAM e de swap utilizados, em Mbytes. No caso do servidor `s4`, o consumo de toda a memória RAM leva à utilização do espaço de swap, como pode ser observado. Já o servidor `s1`, que não esgota sua memória RAM, não chega a utilizar o swap.

O gráfico de utilização de CPU mostra a carga, como obtida de `/proc`. É importante notar que o valor absoluto dos números não é relevante, mas sim as variações. Dessa maneira, podemos notar que o servidor `s4`, ao começar a arrastar, passa a apresentar uma carga muito maior que a apresentada pelo servidor `s1`.

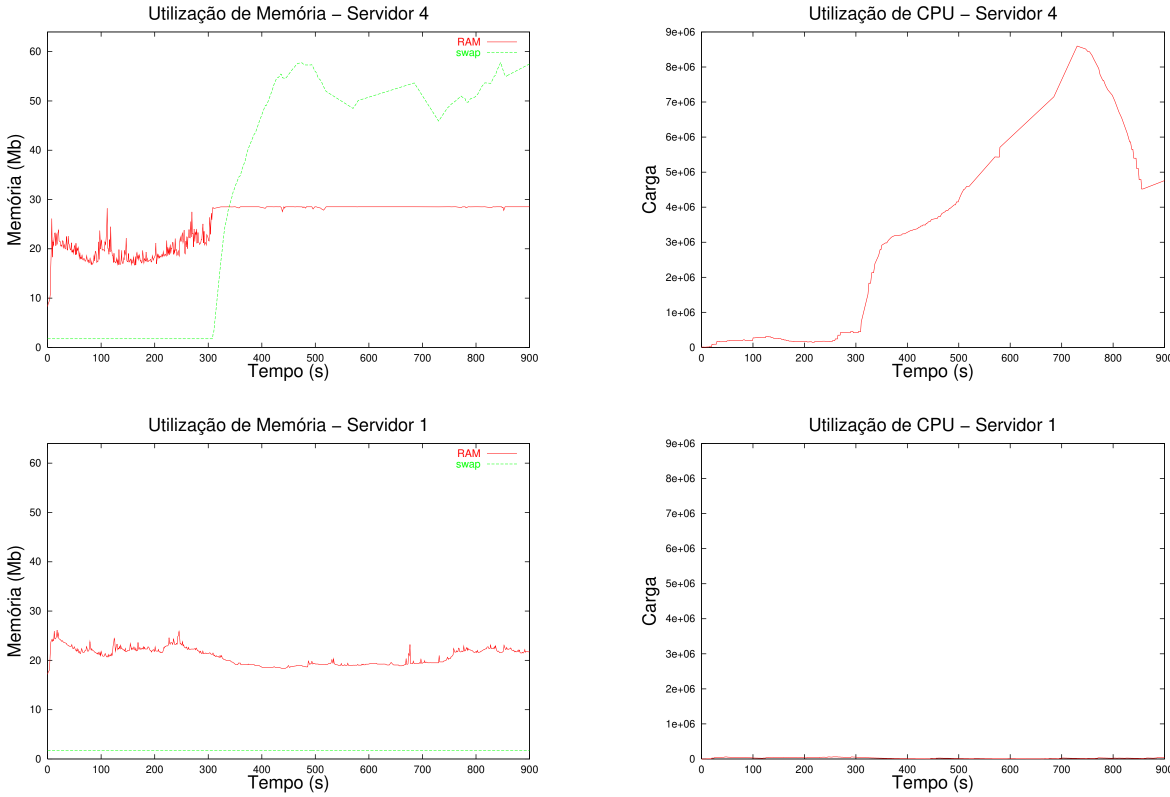


Figura 4.4: Teste de carga de 220 UE sobre 6 servidores HTTP, utilizando RR-DNS.

Capítulo 5

Distribuição por Construção Dinâmica de Ligações

Nesse capítulo, utilizando o protótipo desenvolvido e a estrutura de testes descrita no capítulo anterior, nós simulamos situações reais, onde o método de distribuição de carga por construção dinâmica de URLs realiza a distribuição. Através dessas simulações, exploramos diferentes algoritmos de distribuição, como apresentamos na seção 5.1.

As situações de sobrecarga, que podem surgir com todos os algoritmos analisados, são estudadas na seção 5.2. Nessa seção nós identificamos características dos estados de sobrecarga e propomos novos algoritmos que procuram eliminar essas situações. Novamente através de simulações, nós mostramos a eficácia desses métodos de eliminação de sobrecarga.

A questão da manutenção de sessões através de mecanismos de afinidade é apresentada na seção 5.3. Nessa seção nós testamos o mecanismo de afinidade e analisamos o impacto causado no desempenho dos algoritmos de distribuição.

Na seção 5.4, discutimos a viabilidade de uma outra arquitetura, baseada no uso de "pontos de entrada". Os pontos de entrada seriam servidores responsáveis por receber as primeiras requisições dos clientes, redirencionando-os para os servidores responsáveis por responder efetivamente as requisições. Essa arquitetura, como veremos, pode ser interessante pois tira do servidor de DNS a responsabilidade da distribuição do primeiro acesso de cada usuário.

Terminando o capítulo, nós apresentamos, na seção 5.5, uma comparação de desempenho entre o método de distribuição proposto e outros trabalhos correlatos.

5.1 Algoritmos de Distribuição

O primeiro algoritmo de distribuição de carga que nós implementamos foi o Round Robin. Através desse algoritmo, cada gerente de carga local, de maneira totalmente independente dos demais gerentes, escolhe o servidor a ser utilizado em uma URL de uma lista circular que contém todos os servidores do cluster. Em nossa implementação, a cada requisição de um cliente, a página solicitada conterá todos os seus links internos direcionados para o próximo servidor da lista circular. Para essa implementação do algoritmo, chamada `rr_v1`, o gerente de carga não utiliza nenhuma informação dos demais gerentes de carga do cluster.

A tabela 5.1 enumera todos os parâmetros utilizados nos testes de desempenho realizados e apresentados neste trabalho. É importante notar que o número de clientes apresentado na tabela representa

Parâmetros de Teste	
Duração	15 min
Número de servidores	6
Número de clientes	25
Pontos de conexão	1000
Fator r (Zipf)	1.07508
TTL	240 s

Tabela 5.1: Parâmetros padrão utilizados nos testes de carga.

o número de computadores do cluster que realizam o papel de clientes, e não o número de usuários que se conectam ao site.

Relembrando, os testes do RR-DNS apresentados na seção 4.2, sujeitos aos mesmos parâmetros da tabela 5.1, alcançaram as taxas de 62.3 req/s e 248.7 Kb/s, para uma comunidade simulada de 200 UE, e 24.1 req/s e 83.9 Kb/s, para uma comunidade de 220 UE. Já o algoritmo `rr_v1`, sujeito aos mesmos parâmetros de teste, e a uma carga de 300 UE, levou o site ao desempenho médio de 90.0 req/s e 406.5 Kb/s. Na figura 5.1, nós apresentamos o acompanhamento desse teste. Como podemos ver nos gráficos, o bom desempenho do site é mantido por todos os servidores e ao longo de todo o período de teste.

Conforme discutido na seção 4.2, os gráficos indicam o bom desempenho do teste, resultado de uma distribuição razoavelmente uniforme das requisições entre os servidores do cluster. Nesse caso, porém, as distribuições vistas nos gráficos refletem o efeito combinado das duas etapas de distribuição: RR-DNS na primeira requisição de cada usuário e `rr_v1` nas demais.

Apesar do bom desempenho do algoritmo `rr_v1` quando submetido a uma carga de 300 UE, bem maior que os 220 UE que não foram suportados pelo RR-DNS, ao elevarmos ainda mais a carga, para 340 UE, o desempenho médio do cluster cai significativamente. Nesse teste, as taxas médias obtidas através do `rr_v1` foram de apenas 8.4 req/s e 18.2 Kb/s.

Como o `rr_v1` não é um algoritmo adaptativo, isto é, não considera a carga dos servidores no processo de distribuição, mesmo que ocorra um desbalanceamento de carga, as requisições continuarão a ser igualmente distribuídas entre todos os servidores, sem qualquer alteração. Com isso, a tendência é que o desbalanceamento aumente, uma vez que o servidor sobrecarregado continuará recebendo novas requisições na mesma taxa que antes.

No caso do `rr_v1`, o desbalanceamento de carga entre os servidores pode ocorrer por diversos motivos. Por exemplo, no teste de 300 UE, a diferença entre os custos das requisições não gerou um desbalanceamento. Isto pode ser visto porque as poucas requisições mais demoradas levaram um tempo curto para serem atendidas. Porém, com o aumento do volume de requisições, podem surgir sobreposições entre várias requisições demoradas, o que pode levar a um desbalanceamento mais significativo da carga. Além disso, a própria distribuição inicial dos usuários nos servidores, obtida através de RR-DNS, pode gerar uma carga muito grande em um dos servidores ao mesmo tempo em que outros servidores podem estar sub-utilizados.

Na figura 5.2, nós apresentamos os gráficos referentes ao teste de carga de 340 UE. Como pode ser visto nos gráficos, com o aumento da carga, o desbalanceamento entre os servidores se acentua e leva um dos servidores a um estado de sobrecarga. Como citado, uma vez que a função de distribuição do algoritmo `rr_v1` não leva a carga dos servidores em consideração, o balanceamento diverge rapidamente e mantém o servidor sobrecarregado nesse estado de forma irreversível.

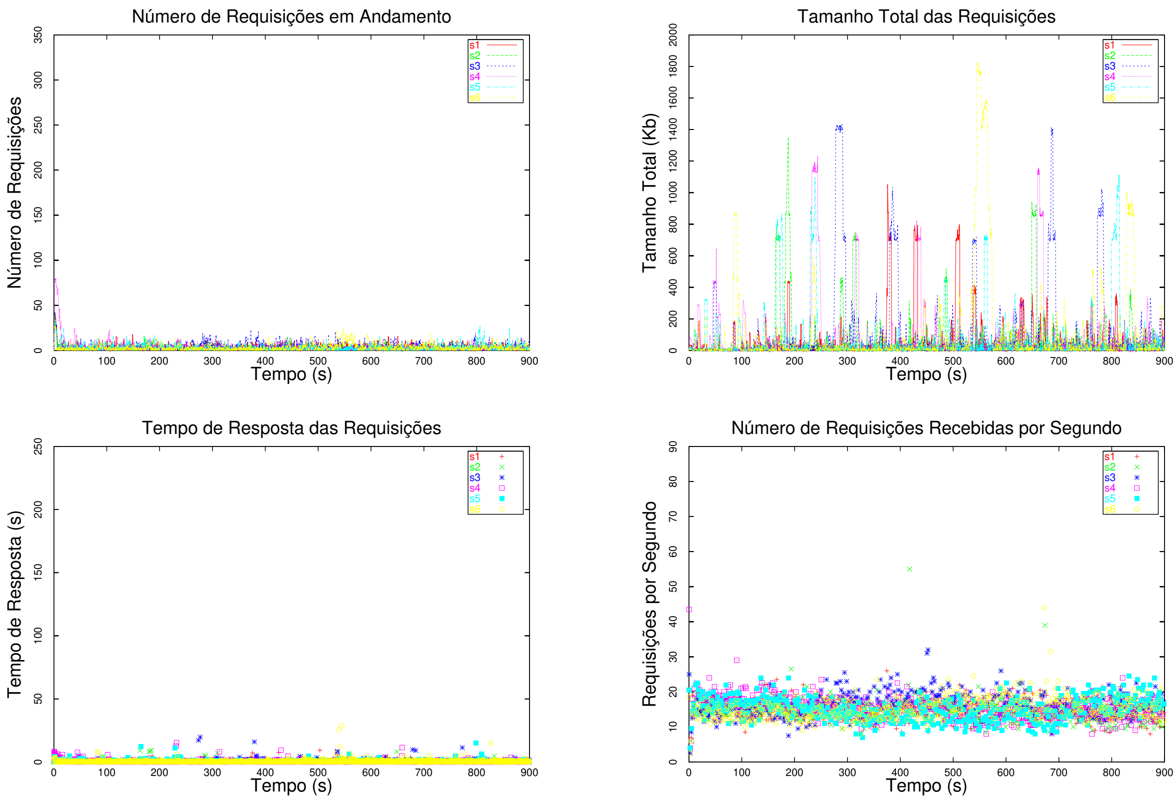


Figura 5.1: Teste de carga de 300 UE sobre 6 servidores HTTP, utilizando o algoritmo `rr_v1`.

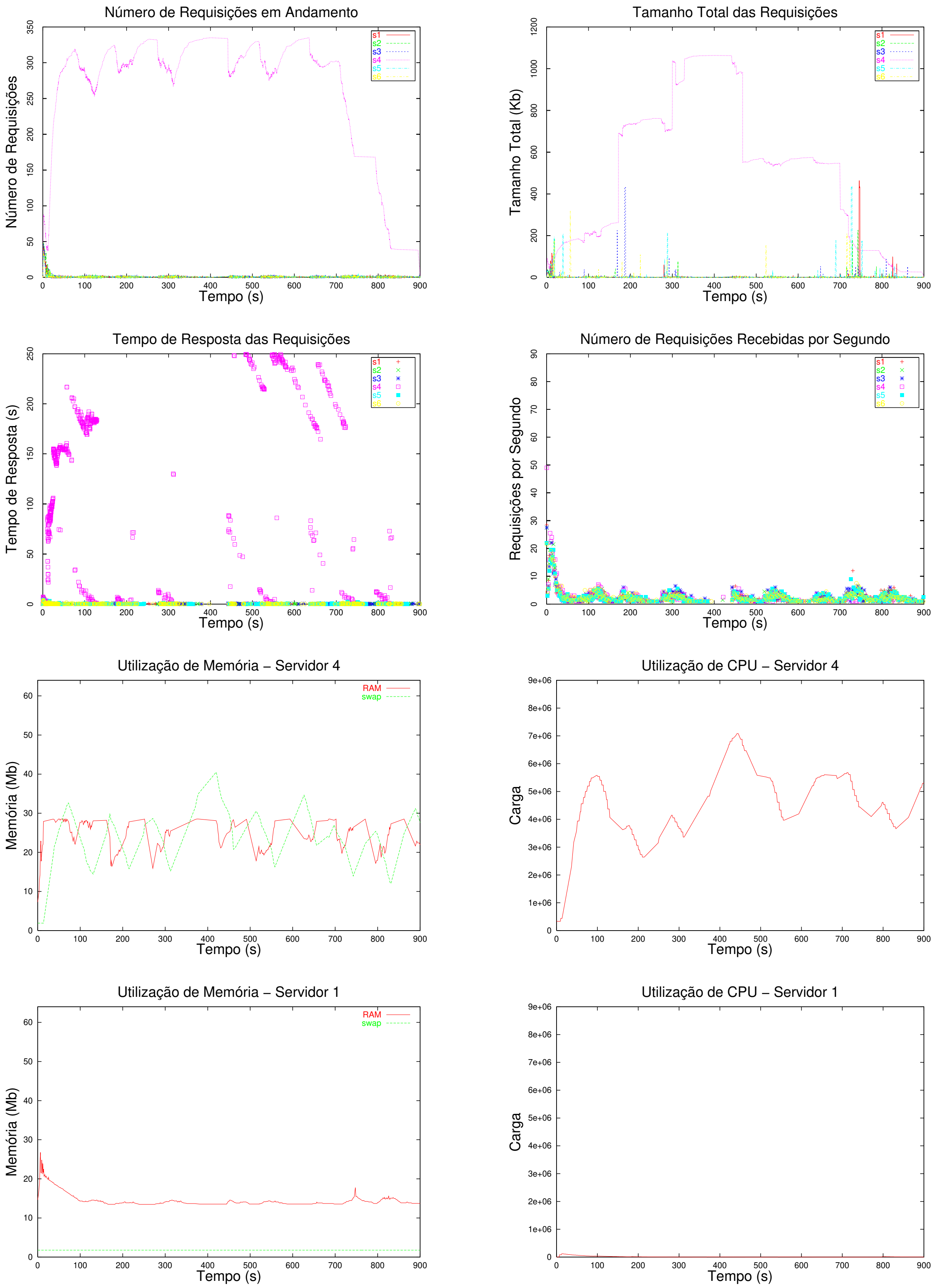


Figura 5.2: Teste de carga de 340 UE sobre 6 servidores HTTP, utilizando o algoritmo `rr_v1`.

Algoritmos adaptativos, que levam carga dos servidores em consideração no processo de distribuição, são alternativas que podemos buscar para o `rr_v1`. Assim, o segundo algoritmo que nós implementamos foi o *Least Loaded*, que chamamos `ll_v1`. Nesse algoritmo, cada gerente mantém uma tabela na qual são registradas as cargas de todos os servidores HTTP do cluster. Uma vez por segundo, cada gerente envia uma mensagem para cada um dos demais gerentes, pedindo as cargas atuais, e atualiza a tabela. No momento dessa atualização, o gerente identifica o servidor do cluster que apresenta a menor carga e o elege para receber as próximas requisições. No `ll_v1`, então, durante cada período de um segundo, compreendido entre duas atualizações das tabelas de cargas, todas as páginas criadas por cada servidor levarão os clientes ao servidor de menor carga.

O fato do `ll_v1` direcionar todas as requisições para o servidor mais disponível durante um período de um segundo faz com que haja um grande desbalanceamento de carga a cada instante. Afinal, todos os servidores, durante um segundo, estarão enviando todas as requisições para um único servidor do cluster. Naturalmente, esse servidor, na próxima atualização, não será mais o de menor carga, e, com isso, o direcionamento passará a outro servidor do cluster. O problema com esse algoritmo é que a carga gerada sobre um servidor durante o tempo entre atualizações pode ser suficiente para sobrecarregá-lo. Esse comportamento pode ser visto nos gráficos apresentados na figura 5.3, que mostram o desempenho dos servidores no teste do algoritmo `ll_v1`, sujeito à carga de 280 UE— ainda menor que a carga de 300 UE com a qual o algoritmo `rr_v1` alcançou um bom desempenho. O baixo desempenho do `ll_v1`, nesse teste, se reflete nos gráficos e nas médias alcançadas pelo cluster: 45.1 req/s e 155.8 Kb/s.

Duas alternativas podem ser adotadas para tentar aumentar o desempenho desse algoritmo. A primeira é a diminuição do tempo entre as atualizações de carga. Dessa maneira, diminuímos o volume de requisições que são enviadas ao servidor de menor carga, o que diminui a chance de que ele se sobrecarregue. A desvantagem, nesse caso, é o aumento da comunicação entre os servidores, comunicação esta que disputará a rede com as requisições e as respostas. Isso acontece porque, por um lado, quanto maior for o número de servidores, menor deverá ser o tempo de atualização, pois maior será a carga enviada ao servidor mais disponível. Por outro lado, quanto maior o número de servidores, maior é o número de mensagens para trocar informações de carga entre todos. Ou seja, um aumento no número de servidores, para comportar um aumento na carga sofrida pelo site, implica não só em um aumento do volume da comunicação interna, como em um aumento da freqüência dessa comunicação.

Outra alternativa é contabilizar o número de vezes que o servidor mais disponível é fornecido aos clientes para que, a partir de um certo ponto, os gerentes possam mudar de servidor, passando, por exemplo, para o segundo servidor de carga mais baixa. Entretanto, esse "ponto de corte", no qual há a troca de servidores, é difícil de se determinar, uma vez que usar o endereço de um servidor em uma resposta não significa que esse servidor virá a ser utilizado pelo cliente no futuro. Além disso, mesmo que o cliente venha a seguir o link, não há como determinar em que momento isso irá ocorrer.

Por esse motivo, nós implementamos outro algoritmo de distribuição de carga que procura manter um maior equilíbrio nos percentuais de carga dos servidores. Esse segundo algoritmo adaptativo, chamado `vl_v1`, procura fazer uma distribuição das requisições diretamente proporcional à capacidade livre de cada servidor. O `vl_v1`, assim como o `ll_v1`, mantém uma tabela contendo as cargas de todos os servidores. Além disso, o gerente também registra o número de requisições recebidas entre duas atualizações da tabela. Assim, sempre que a tabela de cargas é atualizada, o gerente calcula, em função desse número de requisições, uma estimativa do número de requisições que devem ser enviadas a cada servidor, de forma diretamente proporcional à capacidade livre atual de cada servidor. Com base nesse cálculo, o gerente retorna os servidores em resposta às requisições tentando garantir a proporção. Se, durante o próximo intervalo entre atualizações da tabela de cargas, o número de

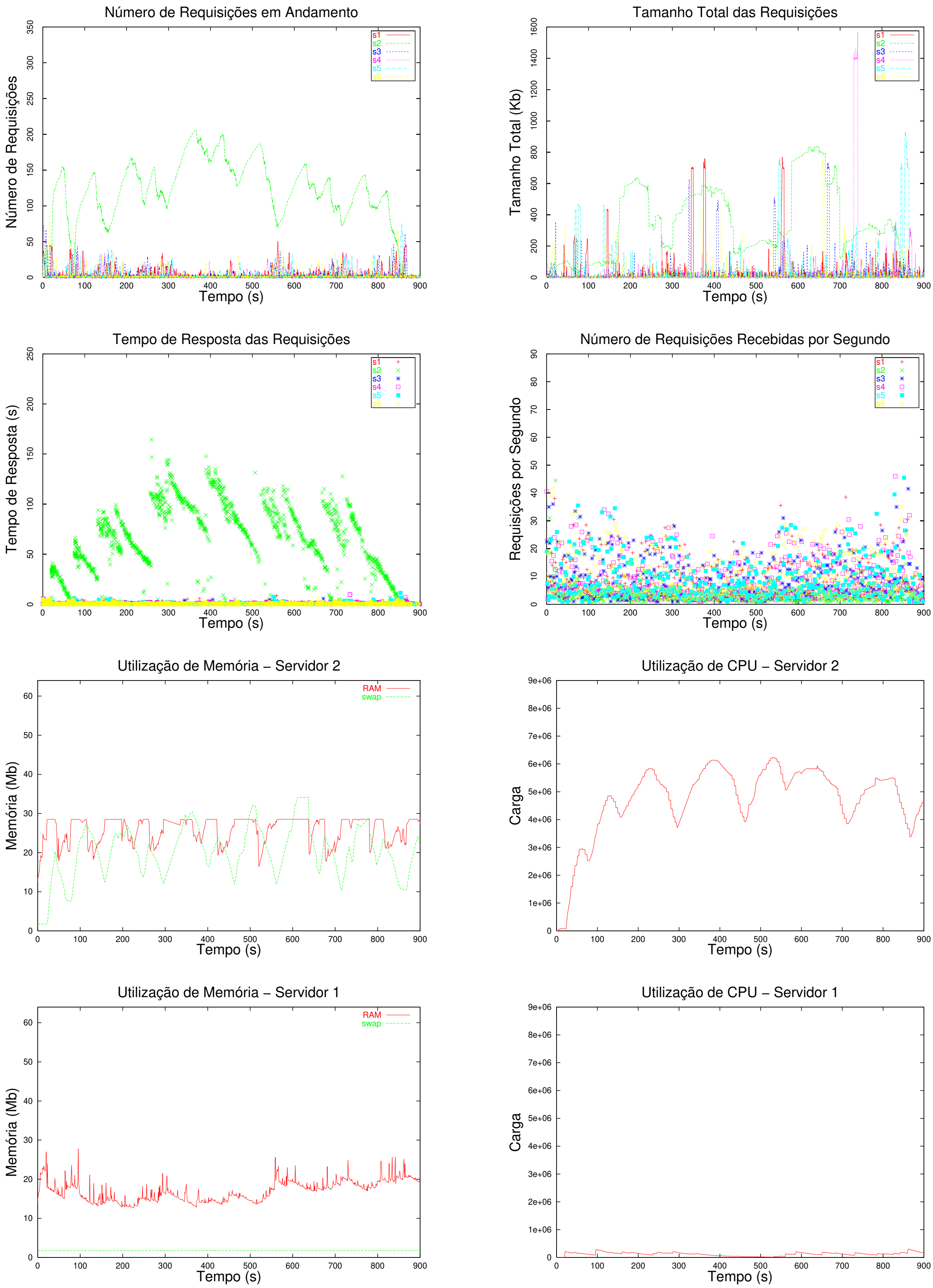


Figura 5.3: Teste de carga de 280 UE sobre 6 servidores HTTP, utilizando o algoritmo `ll_v1`.

requisições se mantiver estável, a proporção será ótima. Porém, caso o número de requisições mude, a proporção não será ótima, mas o gerente garantirá que os servidores que forem proporcionalmente mais utilizados nas respostas aos clientes serão os de menor carga.

É importante notar que uma "proporção ótima" significa o equilíbrio ideal em função do número de vezes que cada servidor é utilizado como ponto de retorno de um cliente. Apesar desse equilíbrio ideal, o algoritmo `vl_v1` continua podendo ser classificado como um algoritmo de *load sharing*. Afinal, mesmo com uma proporção ótima, não há nenhuma garantia de que a carga que retornará aos servidores estará balanceada, por diferentes motivos: em primeiro lugar, apenas uma parcela dos clientes retornará ao site. Em segundo lugar, requisições diferentes geram cargas diferentes. E, por fim, a distribuição dos novos clientes, feita através do RR-DNS, pode mudar bastante a carga nos servidores.

Um teste do algoritmo `vl_v1` pode ser analisado através dos gráficos da figura 5.4. Nesse teste, nós utilizamos a configuração padrão, com uma comunidade simulada de 300 UE. Sob essa carga, o algoritmo apresenta um ótimo desempenho, assim como o algoritmo `rr_v1` com a mesma carga, mantendo uma distribuição uniforme das requisições pelos servidores e atingindo uma média de 87.4 req/s e 388.3 Kb/s.

Pelo fato de ser um algoritmo adaptativo e procurar manter as cargas dos servidores balanceadas, nós esperávamos um melhor desempenho do `vl_v1` nas situações de maior carga, quando comparado ao `rr_v1`. Afinal, em virtude dessa distribuição adaptativa, os desbalanceamentos, advindos, por exemplo, da primeira etapa de distribuição (RR-DNS) ou da variedade de tamanhos de arquivos, podem ser compensados, anulando, ou reduzindo, a evolução que, no `rr_v1`, leva o servidor rapidamente ao estado de sobrecarga. Essa conjectura foi confirmada nos testes de carga que realizamos, como veremos a seguir.

Em um teste do algoritmo `vl_v1` sujeito a uma comunidade simulada de 340 UE e retratado pelos gráficos da figura 5.5, um servidor é levado à sobrecarga, exatamente como ocorreu no teste de mesma carga do algoritmo `rr_v1`. Porém, o desempenho médio do cluster apresenta uma melhora significativa: enquanto a taxa média de transferência no teste do algoritmo `rr_v1` foi de apenas 18.1 Kb/s, a taxa média no `vl_v1` chegou a 105.9 Kb/s, o que representa um aumento de 485.1%. Já a média de requisições atendidas, que no teste do algoritmo `rr_v1` ficou em 8.4 req/s, atingiu 29.5 req/s no `vl_v1`, um aumento de 251.2%.

É importante notar que o ganho de desempenho alcançado pelo `vl_v1` não foi causado por nenhum fator externo ao algoritmo, como, por exemplo, uma diferença de tempos de chegada à sobrecarga entre os dois testes. De fato, esse ganho de desempenho foi incremental e obtido ao longo de todo o teste, mostrando que o desempenho na sobrecarga é mais alto no algoritmo adaptativo. Esse comportamento pode ser visto no gráfico da figura 5.6, que apresenta o número acumulado de requisições atendidas ao longo dos dois testes.

Como qualquer algoritmo de distribuição de carga, o `vl_v1` possui limitações, como a própria existência de uma situação de sobrecarga nos mostra. Uma limitação de desempenho que nós vemos nesse algoritmo é uma conseqüência direta da distribuição probabilística dos comprimentos de caminhos. É importante lembrar que a distribuição *a posteriori* se baseia unicamente na distribuição da segunda à última requisição de cada caminho dos usuários. Ou seja, a primeira requisição de um usuário é distribuída através de RR-DNS (ou qualquer outro método que se venha a adotar) e apenas as demais serão distribuídas pelos gerentes de carga. Mas o ponto a ser notado é que o gerente de carga sempre *erra* na última requisição de cada cliente, no sentido de que ele fornece um servidor para uma próxima requisição que nunca ocorrerá, pois o cliente não retornará ao site. Esse comportamento pode ser crítico no `vl_v1`, pois a escolha dos servidores é função das escolhas anteriores que, supostamente, agregaram alguma carga a esses servidores.

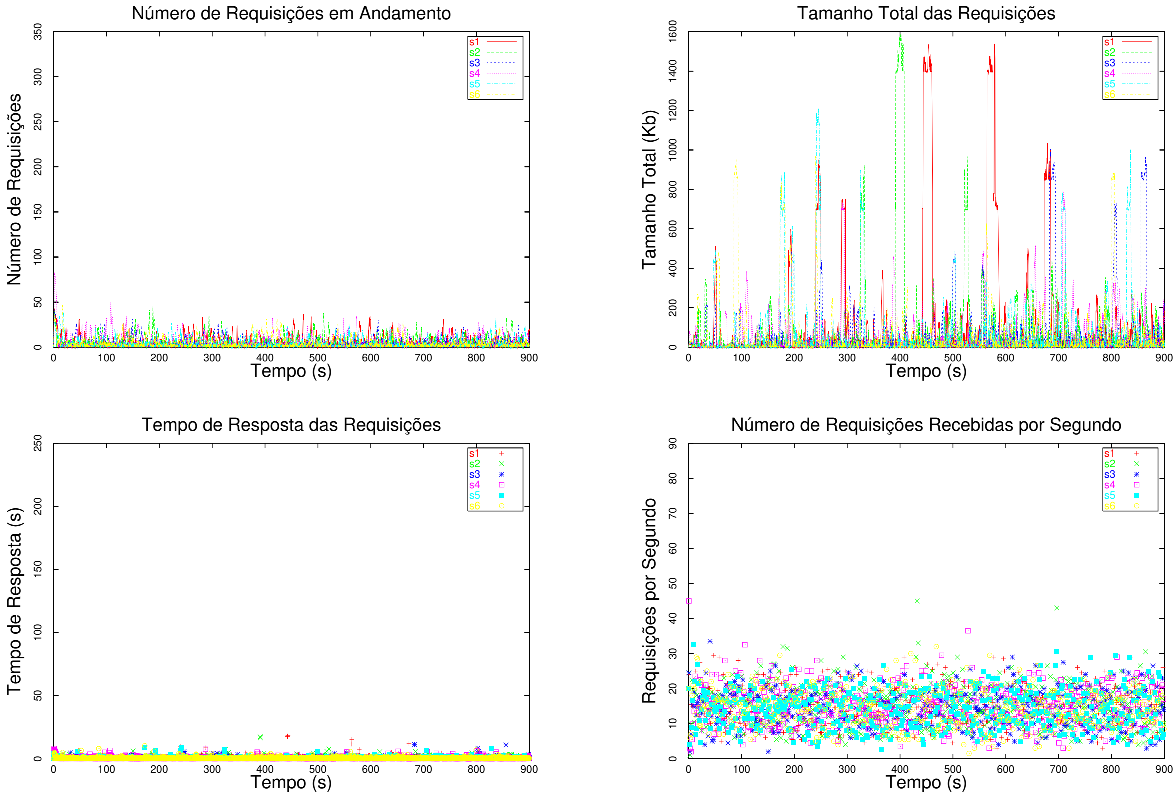


Figura 5.4: Teste de carga de 300 UE sobre 6 servidores HTTP, utilizando o algoritmo `v1_v1`.

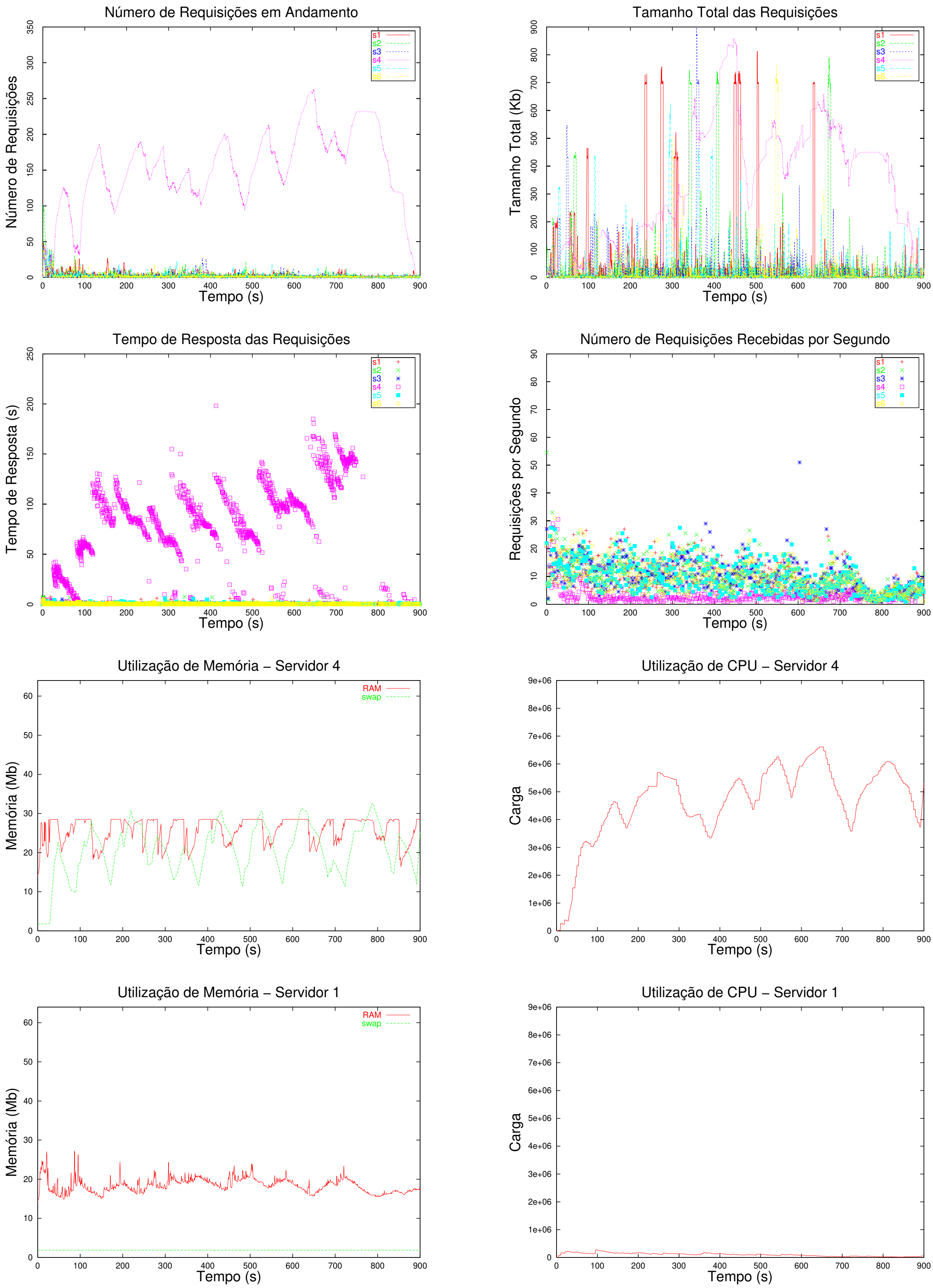


Figura 5.5: Teste de carga de 340 UE sobre 6 servidores HTTP, utilizando o algoritmo `v1_v1`.

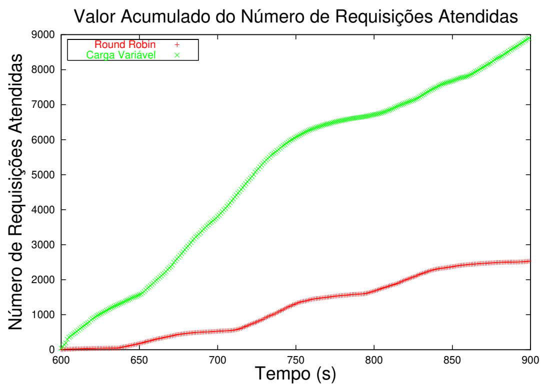


Figura 5.6: Desempenho total dos clusters `rr_v1` e `vl_v1`, com 340 UE.

O erro do gerente, decorrente da última requisição de cada caminho, então, é tão maior quanto maior for o número de "últimas requisições". A distribuição dos comprimentos de caminhos medidos empiricamente é modelada por Huberman et al. [1998], segundo a equação:

$$P(L) = \sqrt{\frac{\lambda}{2\pi L^3}} \exp\left[\frac{-\lambda(L-\mu)^2}{2\mu^2 L}\right]$$

Como notam os autores, dada a assimetria da curva, o comportamento típico dos usuários será percorrer um caminho de comprimento menor que o comprimento médio. Tanto nas observações realizadas por Huberman et al. [1998], quanto nas de Arlitt [2000], o número de caminhos curtos, atendendo ao modelo, é muito grande (com valores empíricos para os parâmetros da equação de, por exemplo, $\mu = 2.98$ e $\lambda = 6.24$), o que faz com que o erro obtido pelos gerentes de carga possa ser significativo em determinadas circunstâncias.

Esse grande número de caminhos curtos tem um impacto direto sobre algoritmos que façam estimativas baseadas no número de vezes que um servidor é usado em resposta a uma requisição. Naturalmente, pelo fato do algoritmo de distribuição ser adaptativo, ele próprio detecta o desbalanceamento e procura compensá-lo. Porém, com o aumento da carga imposta ao cluster, o desbalanceamento decorrente da composição de vários fatores, como a primeira etapa de distribuição (RR-DNS, não adaptativa), a variação nos custos das requisições e o erro das últimas requisições de cada caminho, pode ser muito grande para ser compensado, levando, assim, algum servidor a um estado de sobrecarga.

5.2 Situações de Sobrecarga

Os testes que realizamos comprovam que, no nosso método de distribuição de carga em servidores HTTP, uma técnica de *load sharing* é mais adequada que uma técnica de *load balancing*. Isso acontece porque, caso não haja sobrecarga em nenhum servidor, o desempenho alcançado pelo cluster é bom, independentemente da distribuição da carga entre os servidores. Por outro lado, caso haja sobrecarga em algum servidor, o desempenho geral do cluster cai significativamente. Assim, evitar que as situações de sobrecarga ocorram, ou até mesmo retirar um servidor de um estado de sobrecarga, devem ser objetivos básicos dos mecanismos de distribuição.

Conforme discutimos na seção anterior, dadas as características das cargas reais impostas aos sites, situações de sobrecarga podem ocorrer com todos os algoritmos de distribuição. Uma maneira de lidarmos com a questão da sobrecarga é evitar que um servidor sobrecarregado receba um número ainda maior de requisições, o que agravaria sua situação. Sob o ponto de vista dos clientes, essa política colabora com a diminuição do tempo de espera, uma vez que as novas requisições serão enviadas para os servidores fora do estado de sobrecarga. Assim, apenas as requisições que já estivessem sendo tratadas pelo servidor no momento em que ele entra no estado de sobrecarga apresentariam uma maior latência.

Embora esse procedimento seja importante para os clientes que já se encontram acessando o site, ele não é suficiente para retirar o servidor do estado de sobrecarga. Afinal, uma vez que a primeira etapa da distribuição é realizada através de RR-DNS, que não usa a carga dos servidores no processo de distribuição, novos clientes continuarão a chegar ao servidor sobrecarregado. Uma solução mais completa para o problema da sobrecarga precisaria tirar o servidor desse estado, ao invés de apenas evitar que a sobrecarga se agravasse.

Para diminuir a carga de um servidor sobrecarregado nós podemos utilizar o suporte ao redirecionamento provido pelo protocolo HTTP, analisado na seção 2.2.3. Esse recurso foi explorado em

[Cardellini et al., 1999b], onde a distribuição de carga era feita por algoritmos de DNS e o redirecionamento era usado para retirar os servidores do estado de sobrecarga.

Em nossa proposta, entretanto, a primeira etapa de distribuição, obtida através de RR-DNS, já é seguida por uma segunda etapa, controlada pelos gerentes de carga e, com isso, o redirecionamento atuaria como uma terceira etapa do processo de distribuição. Na nossa técnica de *load-sharing*, cada etapa atua sobre a distribuição obtida pela etapa anterior, de forma a torná-la mais homogênea. Como as distribuições se combinam, cada etapa pode atuar em uma parcela reduzida de requisições. Por causa dessa característica, os problemas associados ao redirecionamento, apontados na seção 2.2.3, podem ser muito diminuídos: se apenas as requisições do tipo `GET` e `HEAD` forem redirecionadas, o impacto não será tão grande quanto nos demais métodos que se baseiam fortemente no redirecionamento. Por outro lado, o aumento de latência só será percebido pelos usuários que chegarem a um servidor sobrecarregado—sendo que essa latência provavelmente ainda será menor do que a latência que ocorreria caso o processamento da requisição fosse feito pelo servidor sobrecarregado, se for razoável supor que o tempo de comunicação de rede entre o cliente e o servidor seja menor que o tempo de processamento de uma requisição por um servidor sobrecarregado.

Uma questão crítica no tratamento das situações de sobrecarga é o tempo gasto para caracterizarmos que um servidor está em sobrecarga. Quando mais rápido detectarmos uma situação de sobrecarga, melhor será o desempenho dos mecanismos de tratamento. Afinal, quanto mais cedo, menos carregado estará o servidor e menor será a fila de requisições a espera de uma resposta, o que permitirá uma execução mais rápida dos redirecionamentos, e necessitará menor esforço para devolver o servidor ao estado normal.

Nos testes realizados, a métrica mais adequada que encontramos para determinar que um servidor se encontra sobrecarregado foi o número de conexões TCP ativas. Naturalmente, essa métrica deve ser parametrizada em uma implementação real, pois, dependendo do hardware do servidor, o número de conexões pode variar, por exemplo, em função do gargalo se encontrar na CPU ou no disco [Barford e Crovella, 1999]. Por fim, um servidor pode ser responsável por outras tarefas, como, por exemplo, a gerência de uma base de dados e, nesse caso, outras métricas, como a taxa de utilização de disco, ou CPU, podem ser necessárias.

Para testarmos a técnica de redirecionamento no tratamento da sobrecarga, nós implementamos novas versões dos algoritmos de distribuição, chamadas `rr_v2`, `ll_v2` e `vl_v2`. Nessas versões, os algoritmos, além de retirar os servidores sobrecarregados das listas dos servidores ativos, ainda realizam o redirecionamento das novas requisições que chegam quando o servidor está sobrecarregado, transferindo-as para algum outro servidor que não esteja em sobrecarga.

Concomitantemente com essas novas versões dos algoritmos, outra modificação foi feita na ferramenta Surge, para que esta pudesse reconhecer uma resposta indicativa de redirecionamento. Em nossa versão modificada, o Surge, ao receber um redirecionamento, registra as informações relevantes para posterior análise, e procede com a nova requisição.

Um teste do algoritmo `rr_v2`, sob a mesma carga de 340 UE, com a qual o algoritmo `rr_v1` não obteve um bom desempenho, pode ser analisado através dos gráficos da figura 5.7. Nesse teste, em virtude dos redirecionamentos, não chega a ocorrer um acúmulo de requisições no servidor sobrecarregado, ao contrário de todos os testes anteriores que apresentaram sobrecarga. Outra diferença significativa desse algoritmo é o baixo tempo de resposta às requisições, como mostrado pelo gráfico correspondente. Entretanto, o desempenho médio do cluster alcançado nesse teste, ao contrário do que possa parecer, não foi bom, alcançando apenas as médias de 15.5 req/s e 63.8 Kb/s. Nós explicamos a seguir as razões desse baixo desempenho, bem como as interpretações dos gráficos correspondentes.

A detecção rápida do estado de sobrecarga, como expusemos acima, é necessária mas não suficiente para viabilizar um procedimento de diminuição da carga de um servidor nessa condição. É

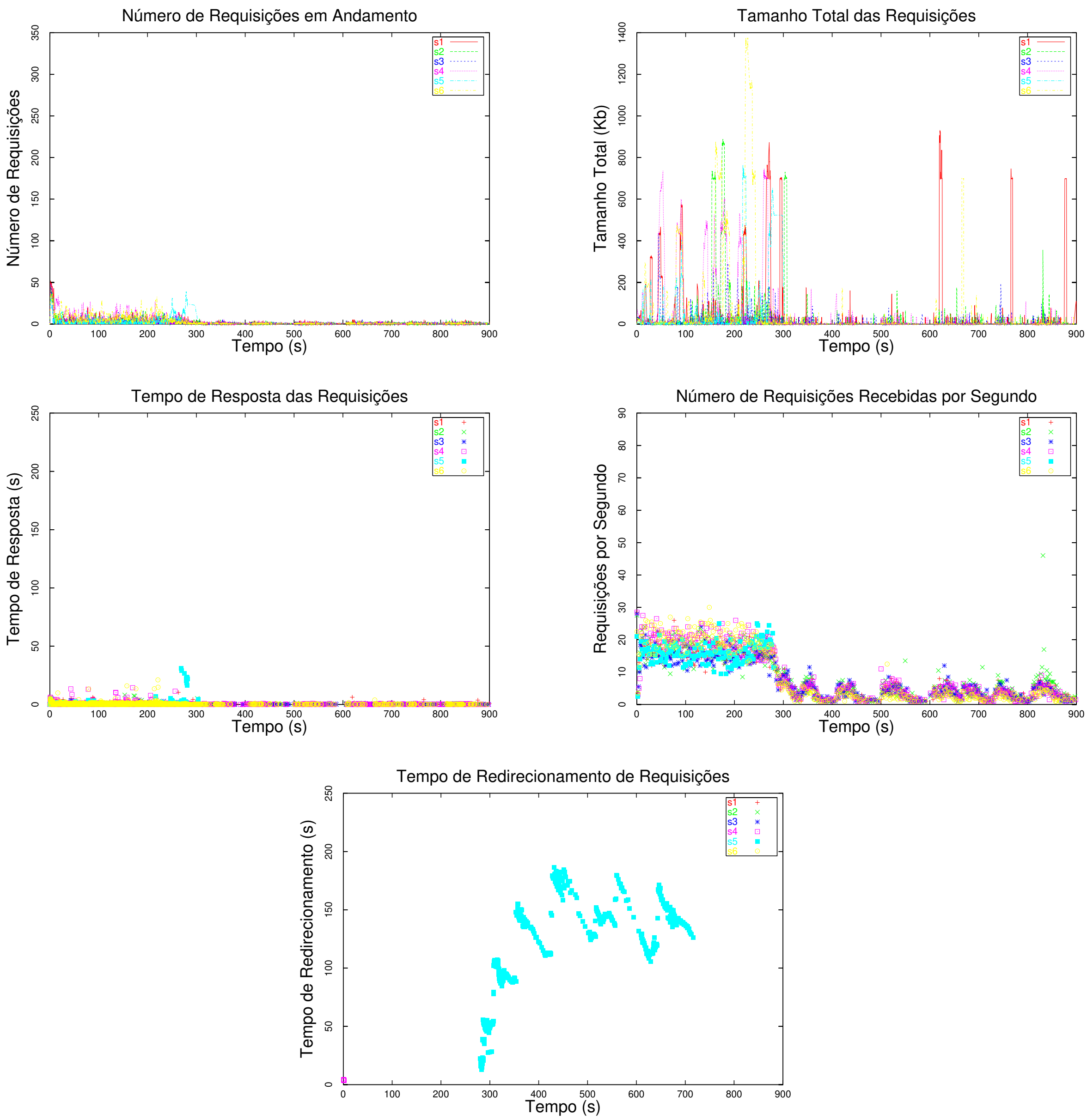


Figura 5.7: Teste de carga de 340 UE sobre 6 servidores HTTP, utilizando o algoritmo `rr_v2`.

importante notar que o redirecionamento, usado para passar o cliente para outro servidor, também é uma resposta a uma requisição HTTP, que será recebida pelo servidor, processada e, finalmente, respondida ao cliente. Ou seja, é fundamental que o custo associado ao redirecionamento seja menor que o custo da resposta à requisição original.

Em virtude de termos adotado a versão CGI do CGILua para prototiparmos nossas propostas, o custo associado ao redirecionamento realizado pelo algoritmo `rr_v2` é muito alto. Afinal, para cada redirecionamento, um novo processo CGILua será iniciado, se conectará ao gerente de carga e, só então, será informado da situação de sobrecarga e poderá retornar o código de redirecionamento. Como no nosso ambiente de teste a situação de sobrecarga faz com que o servidor arraste, todo esse procedimento torna-se muito caro computacionalmente.

Essa situação pode ser bem avaliada através do último gráfico exposto na figura 5.7, que apresenta o tempo gasto nos redirecionamentos. Nesse gráfico, cada ponto representa o tempo de espera de um cliente por uma resposta de redirecionamento vinda do site. É importante notar que os tempos de resposta, que pareciam ter diminuído muito nesse teste, na verdade apenas mudaram de lugar. Isto é, os clientes, ao invés de esperarem muito por uma resposta contendo o recurso desejado, esperam muito por um redirecionamento que os leva a um servidor de onde eles podem obter esse recurso mais rapidamente.

Nos parece razoável, então, que, caso o protótipo utilizasse uma versão residente do CGILua, o custo associado ao redirecionamento seria menor, o que viabilizaria o uso do redirecionamento. Para validar esse raciocínio, nós implementamos um módulo do servidor Apache que, através de um segmento de memória compartilhada com o gerente de carga, obtém as informações de carga dos servidores, e realiza o redirecionamento do cliente quando necessário. Dessa maneira, um novo processo não é criado para tratar o redirecionamento. Além disso, a comunicação por memória compartilhada é muito mais rápida que a comunicação por *sockets* utilizada pela biblioteca ALua.

As versões dos algoritmos que permitem o redirecionamento via servidor HTTP foram chamadas de `rr_v3`, `ll_v3` e `vl_v3`. A melhor eficiência do redirecionamento pode, então, ser avaliada através dos gráficos apresentados na figura 5.8. Esses gráficos refletem o resultado de um teste do algoritmo `rr_v3`, nas mesmas condições do teste anterior, e mesma carga de 340 UE. Nesse caso, o desempenho médio do site subiu para 71.1 req/s e 323.7 Kb/s, o que representa aumentos de 358.7% e 407.4%, respectivamente.

Como podemos ver, nesse teste não há grande acúmulo de requisições em nenhum dos servidores. No pior caso, o servidor `s4` chega a acumular pouco mais de 60 requisições. Além disso, o tempo de resposta às requisições é relativamente baixo, assim como o tempo gasto nos redirecionamentos. Como o tempo médio de resposta diminui sensivelmente em relação ao teste com redirecionamento via CGI, é possível observar um aumento na taxa de requisições recebidas por segundo.

Nós podemos analisar melhor a diferença de desempenho entre as versões CGI e módulo Apache sob dois enfoques: a capacidade de redirecionamento do servidor sobrecarregado e o desempenho do site em termos de requisições atendidas. Os gráficos referentes a essas duas comparações, realizadas com base nos testes dos algoritmos `rr_v2` e `rr_v3`, se encontram na figura 5.9.

No primeiro gráfico da figura, nós mostramos a comparação direta entre a capacidade de redirecionamento do processo CGI e a do servidor HTTP. Nesse gráfico, cada ponto representa um redirecionamento e indica, também, o tempo que esse redirecionamento levou para chegar ao cliente. A diferença entre os dois métodos, no teste em questão, é muito grande. Enquanto através de um processo CGI os tempos gastos nos redirecionamentos se distribuem, em sua maioria, entre 100 e 200 segundos, os redirecionamentos feitos através do servidor HTTP se distribuem, em sua maioria, entre 0 e 40 segundos, raramente ultrapassando 60 segundos. Além disso, o número de redirecionamentos realizados através do servidor HTTP é, aproximadamente, duas vezes maior que o total alcançado

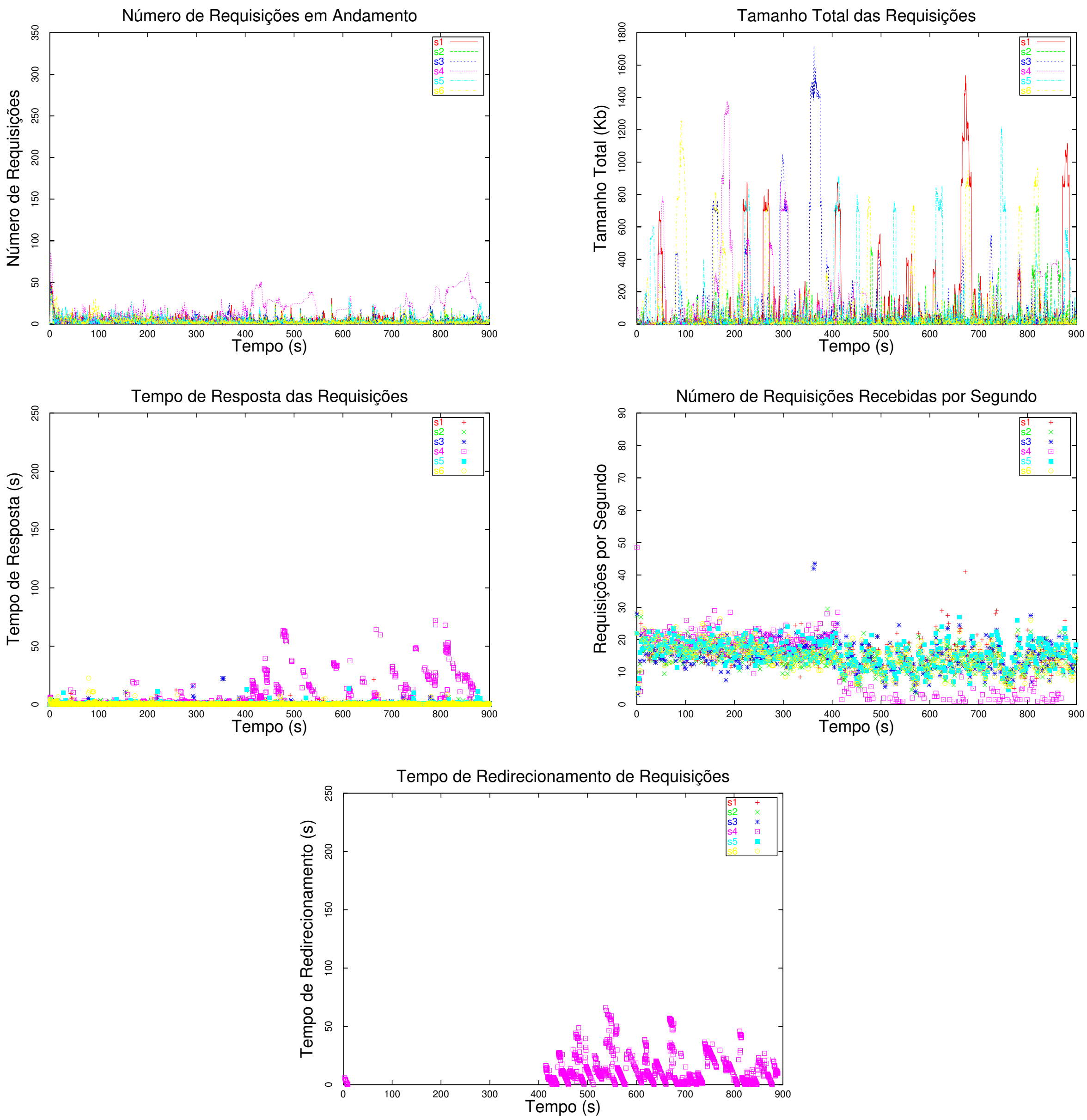


Figura 5.8: Teste de carga de 340 UE sobre 6 servidores HTTP, utilizando o algoritmo `rr_v3`.

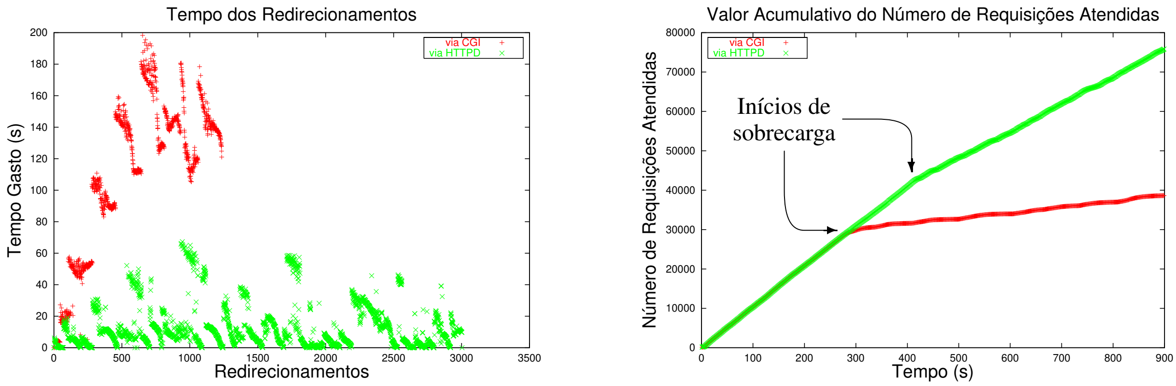


Figura 5.9: Eficácia dos métodos de tratamento de sobrecarga via CGI e servidor HTTP.

através de processos CGI.

No segundo gráfico, nós fazemos a comparação de desempenho através da evolução do número de requisições atendidas pelo site ao longo do tempo. Graças à taxa de redirecionamentos alcançada, o número total de requisições processadas no teste que realizou os redirecionamentos através do servidor HTTP é, aproximadamente, igual ao dobro do total obtido no teste baseado em CGI. Em relação ao teste do algoritmo `rr_v1`, a distribuição obtida através do algoritmo `rr_v3` permitiu um aumento de 746.4% no total de requisições atendidas e de 1688.4% na taxa média de transferência de informações.

No caso do algoritmo `ll_v3`, a retirada dos servidores sobrecarregados da lista de servidores ativos não tem significado, uma vez que o algoritmo utiliza apenas o servidor de menor carga. Já o redirecionamento atua significativamente, retirando as novas requisições dos servidores sobrecarregados, evitando que essas situações de sobrecarga se agravem. Apesar disso, como veremos, esse algoritmo leva a uma degradação do desempenho.

Em um teste do algoritmo `ll_v3`, submetido à mesma carga de 280 UE que manteve um servidor sobrecarregado durante todo o período de teste do algoritmo `ll_v1` (veja figura 5.3), nenhum servidor chega a ficar tão sobrecarregado, como pode ser visto nos gráficos da figura 5.10. Porém, enquanto a distribuição obtida através do `ll_v1` alcança as taxas médias de 45.1 req/s e 155.8 Kb/s, através do `ll_v3` chega-se às médias de 35.1 req/s e 123.6 Kb/s.

A diminuição de desempenho pode ser explicada pela própria distribuição da carga. Como explicamos na seção 5.1, o Least Loaded não evita o desbalanceamento da carga, causando rajadas de requisições que podem levar um servidor à sobrecarga. Entretanto, esses acúmulos periódicos de requisições pelos servidores não foram retratados na análise do algoritmo `ll_v1`, uma vez que um servidor se manteve sobrecarregado durante todo o período de teste, acumulando quase a totalidade das requisições. Já no teste do algoritmo `ll_v3`, essa característica pode ser observada.

No gráfico do número de requisições em andamento, apresentado na figura 5.10, a ampliação de um pequeno período de tempo do teste mostra como os acúmulos de requisições se seguem entre os servidores. Esses acúmulos de requisições inicialmente levam um servidor à sobrecarga, assim como no teste do algoritmo `ll_v1`. Como o algoritmo `ll_v3` redireciona as novas requisições que chegam a esse servidor sobrecarregado, a longa fila de requisições não se forma, pois as requisições são distribuídas pelos demais servidores. Com isso há um volume suficiente de requisições para formar novos acúmulos nesses demais servidores do cluster, levando um deles à sobrecarga. Essa situação é bem retratada na figura 5.10, onde o gráfico de redirecionamentos mostra que diversos servidores efetuaram redirecionamentos durante o período de teste. Ou seja, o mecanismo utilizado para retirar um servidor da sobrecarga, no algoritmo `ll_v3`, praticamente troca a sobrecarga de servidor, ao invés de eliminá-la. Na verdade, o que obtemos com o `ll_v3` foi uma multiplicação do número de servidores sobrecarregados.

No algoritmo de carga variável, as técnicas de tratamento de sobrecarga representam um ganho intermediário quando comparado aos ganhos atingidos pelo algoritmo Round Robin. Em função da distribuição proporcional de cargas, os problemas presentes no `ll_v3` não se repetem no `vl_v3`, garantindo um desempenho melhor: em um teste de carga de 340 UE, o desempenho médio do cluster foi de 75.2 req/s e 332.3 Kb/s. Por outro lado, o próprio algoritmo `vl_v1` já apresentava, na sobrecarga, um desempenho muito superior ao desempenho do `rr_v1`. É essa diferença nas bases de comparação que faz com que o ganho obtido pelo `vl_v3` seja menor que o ganho do `rr_v3`. De fato, em termos absolutos, `rr_v3` e `vl_v3` apresentam, basicamente, o mesmo desempenho final. Em relação ao algoritmo `vl_v1`, o algoritmo `vl_v3` apresenta um aumento de 154.9% no total de requisições atendidas e de 213.8% na taxa média de transferência. Os gráficos referentes ao teste do algoritmo `vl_v3` são apresentados na figura 5.11.

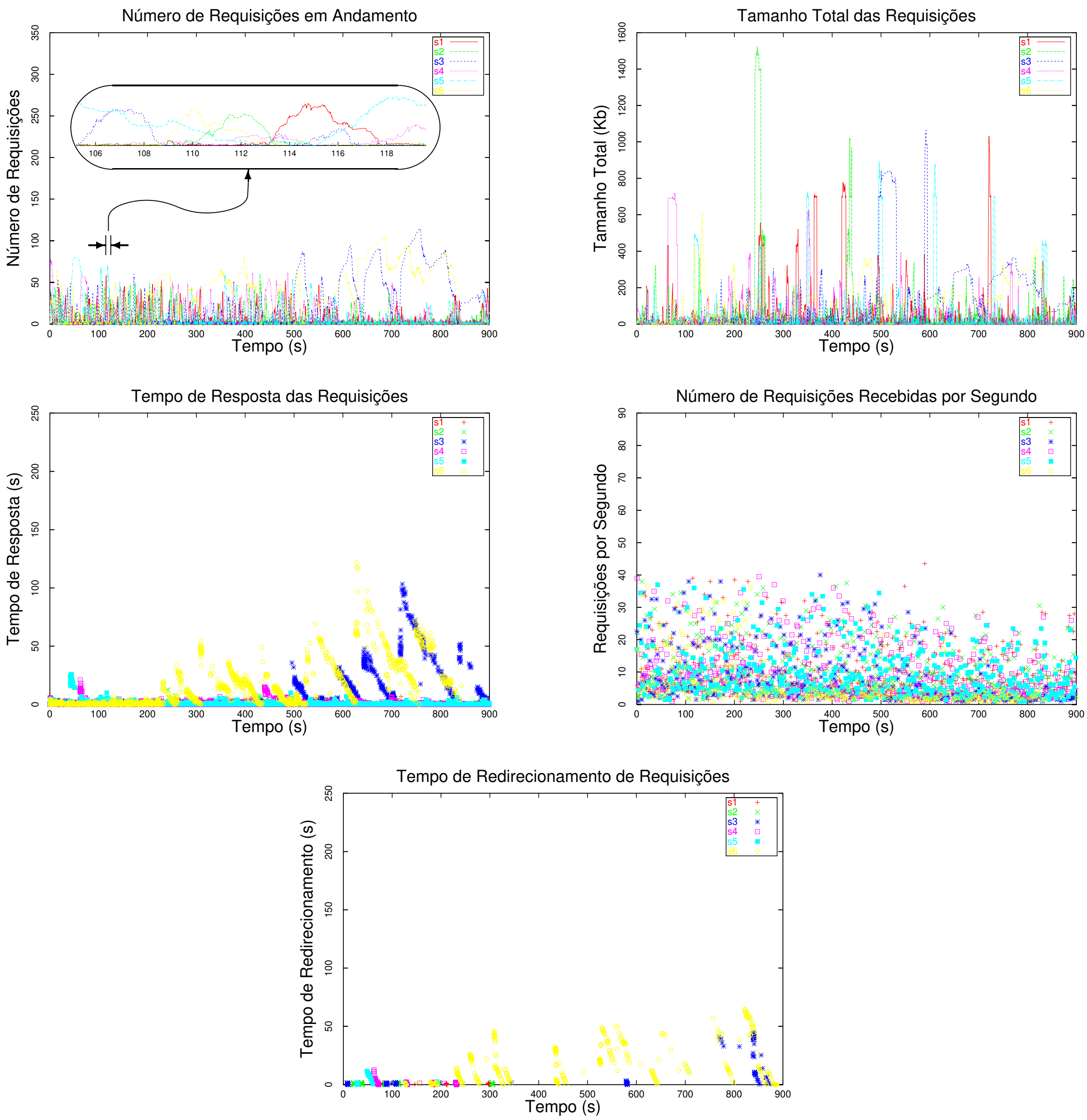


Figura 5.10: Teste de carga de 280 UE sobre 6 servidores HTTP, utilizando o algoritmo `ll_v3`.

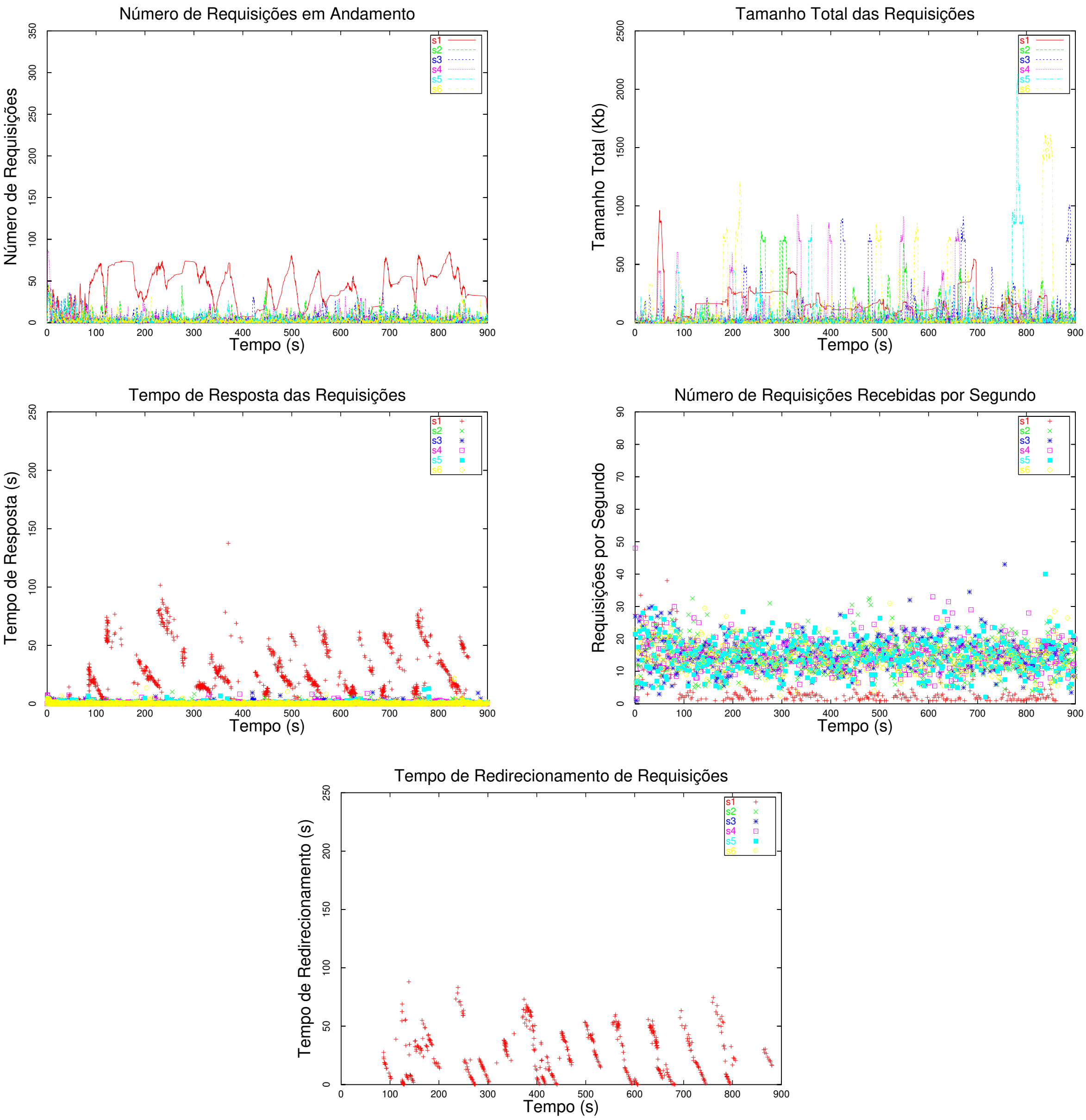


Figura 5.11: Teste de carga de 340 UE sobre 6 servidores HTTP, utilizando o algoritmo `v1_v3`.

5.3 Sensibilidade à Manutenção de Sessões

Um ponto positivo do método de distribuição de carga que propomos é o fato da gerência da manutenção de sessões entre os clientes e o site poder ser feita de forma eficiente. No método CDL, a afinidade pode ser implementada trivialmente, bastando, para isso, manter os links internos das páginas construídas para o cliente se referenciando sempre ao mesmo servidor. Mais do que isso, com a integração da distribuição na ferramenta de construção de páginas, o estabelecimento das sessões sempre funciona e pode ser feito apenas quando necessário, características ausentes de todas as ferramentas de distribuição de carga analisadas no capítulo 2.

Independente da forma de implementação, a manutenção de sessões através de afinidade estabelece um ponto crítico para qualquer ferramenta de distribuição de carga: o fato de todas as requisições provenientes de um usuário precisarem ser tratadas sempre pelo mesmo servidor. Esse é um objetivo que, claramente, é contrário ao objetivo primordial de um mecanismo de distribuição de carga. Em um caso extremo, onde todos os usuários estabeleçam sessões com o site, o mecanismo de distribuição fica restrito aos novos usuários que cheguem ao site. Ou seja, quanto maior for o percentual do número de sessões estabelecidas em relação ao número total de usuários acessando o site, menor será a capacidade de distribuição.

Em nossa proposta, em particular, se nenhuma requisição posterior à primeira puder ser distribuída, então não há distribuição alguma além da primeira etapa. Na implementação analisada até aqui, essa distribuição se resumiria àquela obtida através do RR-DNS.

Para verificar o impacto que o estabelecimento de sessões através da afinidade tem sobre a nossa proposta, nós modificamos, novamente, o Surge para que os clientes pudessem ser parametrizados com um *percentual de afinidade*. Esse percentual de afinidade indica, em nossa implementação, um percentual do número de requisições de cada caminho que deve ser realizado com uma sessão estabelecida.

Para isso, nós implementamos um mecanismo no CGILua que permite o estabelecimento de uma sessão entre um cliente e o site, permitindo a manutenção de informações relativas a essa sessão. A afinidade, em nossa implementação, é mantida através de reescrita de URLs. Assim, sempre que o cliente solicita uma sessão, ela é criada no servidor que atende a requisição, e o identificador dessa sessão é retornado nas URLs da página gerada, que não distribuem o cliente, mantendo-o sempre preso ao servidor em questão.

Embora em uma implementação real as sessões sejam requisitadas pelas próprias páginas sendo construídas, em nosso caso elas são requisitadas pelos clientes. Apesar de irreal, essa implementação foi feita dessa maneira em função das páginas geradas pelo Surge não possuírem um conteúdo significativo. Porém, esse fato não compromete a análise que iremos fazer: a sensibilidade do método de distribuição CDL à manutenção de sessões.

É importante entender um detalhe da implementação para podermos tirar conclusões sobre os resultados obtidos nas simulações. Esse detalhe é o fato de que o Surge, antes de efetuar cada requisição ao site, determina qual o valor percentual do total de requisições do caminho sendo simulado que ainda falta ser realizado para que esse caminho termine. E é esse percentual que, sendo igual ou menor que o percentual de afinidade, faz com que uma sessão seja requisitada ao site. Com isso, a requisição de uma sessão será feita no primeiro acesso de um caminho apenas quando o percentual de afinidade for 100%. Em outras palavras, mesmo para um percentual de afinidade de 99.9%, a primeira requisição de todos os clientes sempre será distribuída pelo cluster e a sessão poderá, se for o caso, ser requisitada a partir do segundo acesso.

Essa característica da implementação, associada à distribuição de comprimentos dos caminhos dos usuários, faz com que o percentual de requisições fixas (isto é, as requisições "presas" em um servidor por conta da afinidade) seja menor que o percentual de afinidade. A evolução dessa diferença entre os percentuais pode ser vista no primeiro gráfico da figura 5.12, montados a partir de testes do algoritmo `v1_v3`, conforme a configuração padrão da tabela 5.1 e uma carga de 300 UE. Estão representados no gráfico os resultados referentes aos testes utilizando os valores 0, 20, 40, 60, 80, 90, 99 e 100% de afinidade. Como mostra o gráfico, a partir de 90% de afinidade o percentual de requisições fixas praticamente se mantém constante—uma característica decorrente do fato de que existem poucos caminhos longos—e menor que o percentual de afinidade—conseqüência direta da primeira requisição estar fixa somente aos 100% de afinidade.

Através dessa implementação, podemos verificar que a sensibilidade da distribuição de carga à manutenção de sessões é praticamente nula. No segundo gráfico da figura 5.12, nós mostramos a evolução do desempenho geral do site, como o volume de informações transferidas em média por cada servidor (valor médio e variações) e a taxa média de requisições atendidas por segundo, em função do aumento do percentual de afinidade. O gráfico mostra que o bom desempenho é mantido ao longo de, basicamente, toda a faixa de percentuais de sessão, isto é, de 0% a 99%, caindo apenas aos 100%.

Conforme observamos na seção 3.1, o mecanismo de afinidade poderia ser levado ao extremo, fixando os usuários desde o primeiro acesso ao site, impedindo a distribuição feita na segunda e terceira etapas. Essa situação é retratada pelo ponto referente à afinidade de 100%, no segundo gráfico da figura 5.12. O baixo desempenho aos 100% de afinidade é resultante da distribuição de carga obtida apenas através do RR-DNS.

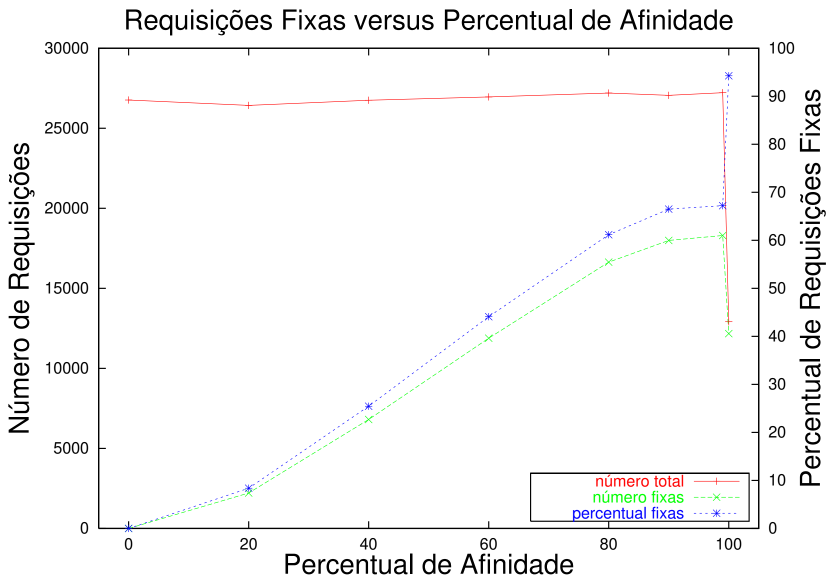
Entretanto, o valor máximo de requisições fixas, atingido aos 99% de afinidade, nos mostra que basta que uma única requisição (a primeira) de cada cliente possa ser distribuída pela segunda etapa de distribuição para que o desempenho melhore significativamente. Desse fato, podemos concluir que, através de um mecanismo que crie a sessão no servidor mais conveniente, como sugerimos na seção 3.1, podemos tornar o desempenho do CDL independente da manutenção de sessões.

5.4 Pontos de Entrada

Na seção 3.1, a arquitetura que apresentamos para operacionalizar o método CDL foi a divulgação, através do uso de RR-DNS, dos endereços de todos os servidores do cluster. Uma outra alternativa é direcionar o primeiro acesso dos usuários a um servidor cuja tarefa seja redirecionar cada usuário para algum outro servidor do cluster que responderá a requisição. Nós chamamos esse servidor, dedicado à função de redirecionamento dos usuários, de “ponto de entrada”, uma vez que ele atua como o ponto de entrada dos usuários no site. Em particular, diversos pontos de entrada podem ser utilizados através da distribuição dos usuários nesses pontos via RR-DNS.

As implementações dos gerentes de carga analisadas até aqui podem ser utilizadas para testar essa arquitetura baseada em pontos de entrada. Para isso, podemos configurar o gerente do ponto de entrada para que ele sempre julgue que o servidor está sobrecarregado, o que faz com que todas as requisições sejam redirecionadas para os demais servidores do cluster. Como a métrica utilizada para determinar que um servidor está sobrecarregado é o número de conexões, é suficiente fazer esse limite igual a zero. Além disso, ao retirarmos o ponto de entrada da lista de servidores dos demais gerentes, evitamos que os clientes voltem a ele após a primeira requisição.

Por exemplo, usando seis servidores, como fizemos nos testes anteriores, podemos deixar apenas um para realizar os redirecionamentos e os cinco restantes para responder aos clientes. Nessa configuração, o desempenho máximo alcançado nos testes que realizamos foi obtido aos 280 UE, ponto no qual o cluster apresentou uma taxa média de transferência de 291 Kb/s e 84 req/s. Acima



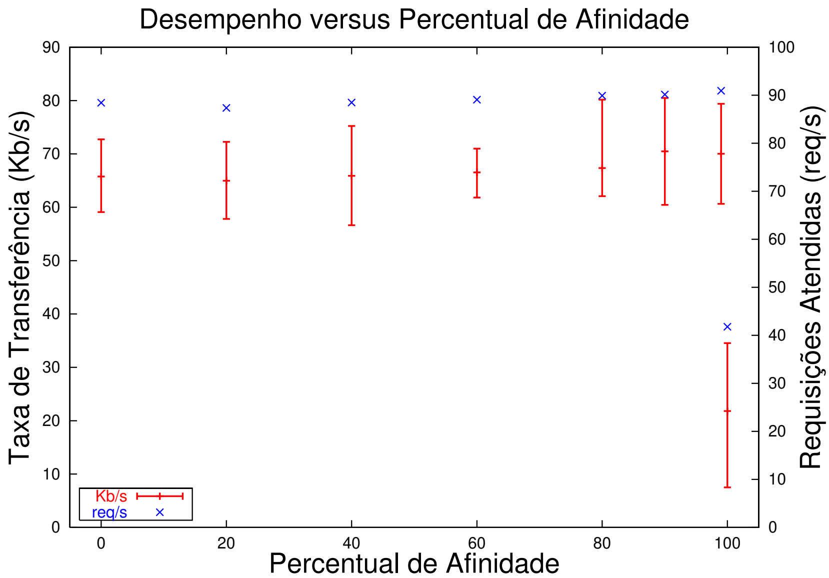


Figura 5.12: Sensibilidade da distribuição de carga à manutenção de sessões.

de 280 UE, o ponto de entrada passa a se sobrecarregar com o volume de requisições, causando uma piora no desempenho do cluster.

Por outro lado, utilizando RR-DNS na primeira requisição dos clientes, sem utilizar pontos de entrada, é possível aumentar a carga para 300 UE e alcançar um desempenho de 406.5 Kb/s e 90.0 req/s, sem que haja sobrecarga nos servidores, como vimos nas seções anteriores. Ou seja, o uso de pontos de entrada é pior, em termos de desempenho, do que a distribuição dos clientes por todos os servidores via RR-DNS. Afinal, como um ou mais servidores do cluster passam a realizar apenas os redirecionamentos, nós diminuímos o número de servidores que efetivamente respondem às requisições dos clientes.

Não obstante, a distribuição dos clientes nos servidores feita através dos pontos de entrada pode ser melhor que a distribuição obtida através do RR-DNS. Afinal, todos os problemas associados ao cache e ao TTL das traduções que fazem com que o RR-DNS apresente uma distribuição ruim, não existem no caso do ponto de entrada. Ou seja, o mesmo algoritmo de Round Robin, quando aplicado através de um ponto de entrada, terá um desempenho melhor do que quando aplicado através do servidor de DNS. Outro ponto importante a ser notado é que os algoritmos baseados em carga, quando aplicados nos pontos de entrada, não sofrem mais com o erro associado às últimas requisições de cada cliente pela simples razão de que não existem “últimas requisições” nos pontos de entrada. Todo cliente chega a um ponto de entrada apenas na sua primeira requisição e, obrigatoriamente, segue para o servidor indicado no redirecionamento. Com isso, a estimativa de carga realizada pelo algoritmo passa a ser mais precisa.

Com uma melhor distribuição na primeira etapa, os servidores alcançam maiores taxas de resposta a requisições, com uma menor probabilidade de que haja desbalanceamento de carga. Esse fator, entretanto, não foi suficiente, nos testes que realizamos, para compensar a perda de desempenho decorrente da diminuição do número de servidores dedicados ao tratamento das requisições. É importante notar que o número de redirecionamentos efetuados no ponto de entrada, necessários para distribuir uma determinada comunidade no cluster, tipicamente será muito maior que o número de traduções que o servidor de DNS faria, para a mesma comunidade. Dessa forma, diversos servidores podem ser necessários como pontos de entrada para distribuir os clientes quando, em uma mesma situação, um único servidor de DNS poderia ser utilizado.

Apesar do menor desempenho, o uso de pontos de entrada pode ser adotado por permitir um maior controle da distribuição dos clientes nos servidores. Por exemplo, o uso de pontos de entrada poderia ser explorado para implementar diferentes tipos de setorização do cluster.

5.5 Comparação com Outros Métodos

Nessa seção, nós apresentamos uma comparação de desempenho direta entre o CDL e dois outros métodos existentes. Um desses métodos é o próprio RR-DNS. Apesar de haver consenso de que o RR-DNS não alcança uma distribuição de carga ideal, é interessante analisar seu desempenho por dois motivos: o primeiro é que outros trabalhos utilizam o RR-DNS como base de comparação de desempenho, o que nos permite comparar, indiretamente, o desempenho do nosso método com esses outros trabalhos. O segundo motivo é o fato de que o RR-DNS ainda é utilizado na prática como método de distribuição de carga em sites Web.

O outro método de distribuição de carga que nós implementamos para efetuar a comparação de desempenho é o RR2-Thr, proposto por Colajanni et al. [1997]. O RR2-Thr é um algoritmo que substitui o Round Robin no servidor de DNS. Os autores realizaram diversos testes para analisar se o uso de algoritmos mais elaborados no servidor de DNS poderia melhorar o desempenho. A conclusão desse trabalho é que essa melhora é possível e a alternativa mais promissora apresentada é o algoritmo RR2-Thr.

O algoritmo RR2-Thr estabelece uma divisão dos pontos de conexão em dois (ou mais) grupos, baseado na "carga oculta" de cada ponto. A carga oculta é, essencialmente, uma medida do número de requisições que podem chegar ao site em conseqüência da tradução fornecida. Assim, um grande provedor terá uma carga oculta maior que uma pequena rede local, por exemplo. Os autores sugerem a criação de um grupo que contenha 40% dos clientes e outro com os 60% restantes, sendo que os clientes de maior carga oculta devem ficar no menor grupo. Com base nesses dois grupos, o RR2-Thr efetua as traduções de endereços usando um algoritmo de Round Robin, utilizando todos os servidores do cluster, de forma independente para cada grupo. O objetivo desse procedimento é evitar que dois pontos de conexão de grande carga oculta sejam associados a um mesmo servidor. Completando o algoritmo, cada servidor envia, periodicamente, sua informação de carga para o servidor de DNS que, ao detectar um servidor sobrecarregado, o retira da lista de servidores usados nas traduções.

Na figura 5.13 nós apresentamos a comparação entre o desempenho obtido através de RR-DNS, RR2-Thr e o método CDL, utilizando os algoritmos `rr_v1`, `rr_v3` e `vl_v3`. Os gráficos indicam a taxa média de transferência dos servidores, junto com os valores mínimo e máximo, e a média total do número de requisições atendidas por segundo para cada diferente método, sujeitos a cargas que variam na faixa de 100 UE a 350 UE. Todos os testes realizados utilizaram os parâmetros padrão da tabela 5.1.

Diversas características podem ser observadas nos gráficos dessa figura. Em primeiro lugar, para cargas baixas, dentro da faixa de 100 UE a 200 UE, todos os métodos evitam as situações de sobrecarga. Porém, a variação da taxa de transferência entre os servidores do cluster, em um mesmo teste, indica que o CDL atinge um equilíbrio significativamente maior da carga entre esses servidores. A melhor distribuição faz com que as situações de sobrecarga só comecem a surgir no CDL, independentemente do algoritmo utilizado nos gerentes, para cargas próximas de 300 UE, enquanto nos outros métodos as sobrecargas podem ser observadas já a partir de 210 UE ou 220 UE.

Outra característica que fica clara nos gráficos é o ganho de desempenho nas situações de sobrecarga decorrentes da aplicação dos mecanismos analisados na seção 5.2. Na ausência desses mecanismos que atuam para a retirada do servidor do estado de sobrecarga, a queda do desempenho geral do cluster é significativa. Já na presença desses mecanismos, embora a queda ainda possa ser notada, o desempenho médio apresenta quedas consideravelmente menores.

De modo geral, há variações mais bruscas nas medidas dentro de uma faixa de cargas na qual o site começa a apresentar baixo desempenho. Isso ocorre em função de vários fatores, como diferenças no tempo de chegada a situações de sobrecarga e características do tráfego gerado pela ferramenta Surge. Assim, podemos entender os gráficos como indicativos de uma faixa inicial de cargas na qual um determinado método tipicamente alcança um bom desempenho, uma faixa de transição dentro da qual o desempenho começa a cair, ficando muito sensível às características do tráfego, e um limite a partir do qual o desempenho tipicamente se mantém baixo.

Outro trabalho ao qual nós podemos, de certa maneira, comparar o CDL é o DCWS [Baker e Moon, 1999], que apresentamos na seção 2.2.4. Conforme explicado, o DCWS obtém a distribuição dos clientes em um conjunto de servidores através da distribuição das páginas do site entre esses servidores. Para isso, o DCWS estima, dinamicamente, qual o melhor particionamento e o efetua através da migração das páginas necessárias. Em particular, para migrar uma página, o DCWS reescreve os links que ligam essa página às demais páginas do site para refletir a nova localização.

Mesmo considerando o custo de reescrita de páginas, os autores afirmam que o ganho de desempenho é praticamente linear com o aumento do número de servidores. Porém, esse sistema tem dois problemas: a falha de um servidor significa perda de parte do site, pois não há replicação, e o uso de

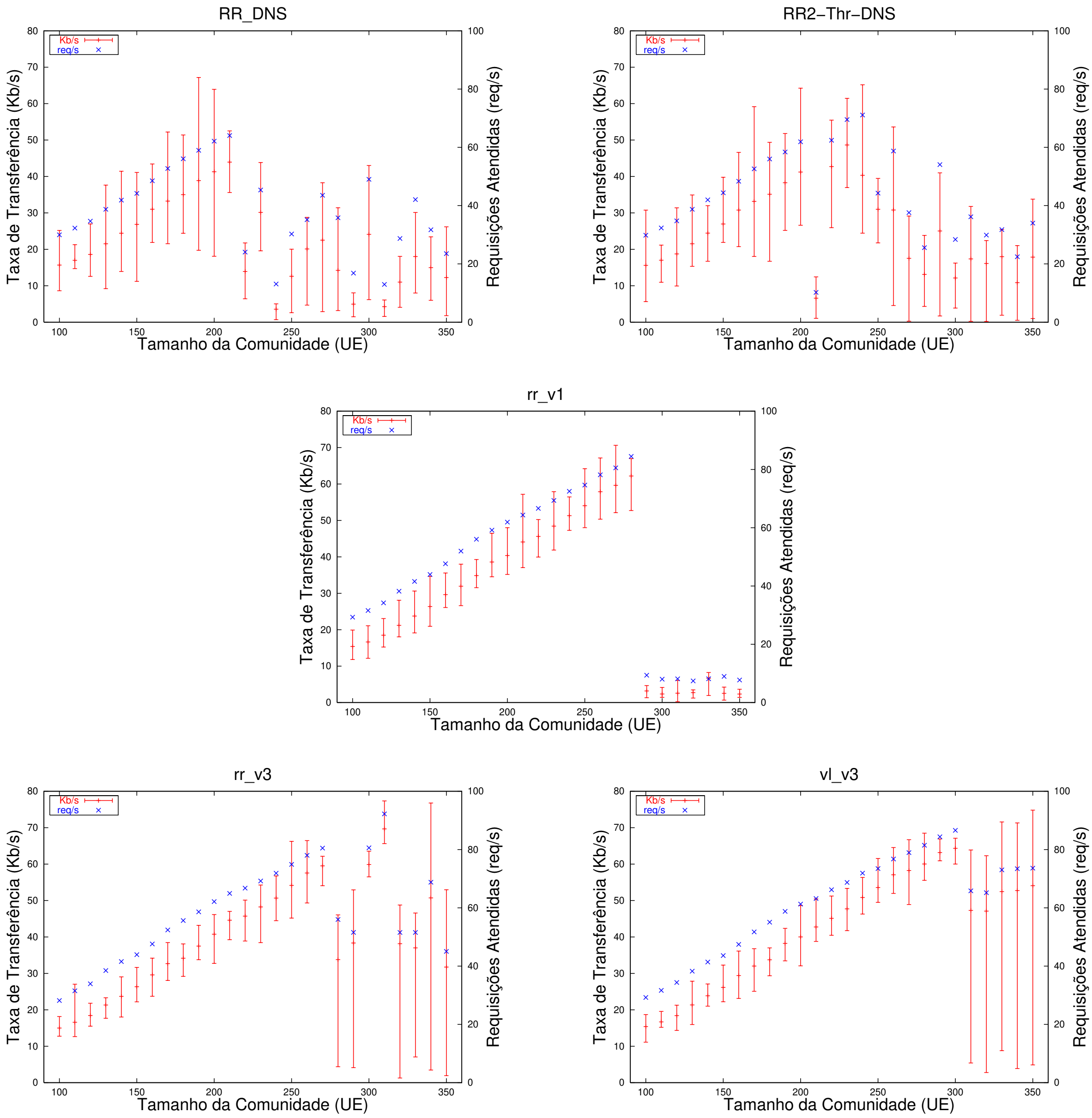


Figura 5.13: Evolução do desempenho para diferentes métodos de distribuição.

Comparação de Escalabilidade				
Método	Algoritmo	Taxa de transferência máxima alcançada (Kb/s)	Número de servidores	Aproveitamento de cada servidor
RR-DNS	–	264.5	6	0.68
RR2-Thr	–	292.9	6	0.75
CDL	`rr_v1`	374.5	6	0.96
	`rr_v3`	419.4	6	1.08
	`vl_v3`	387.3	6	0.99

Tabela 5.2: Comparação entre a escalabilidade de diferentes métodos.

bookmarks não pode ser permitido, uma vez que as páginas podem mudar de localização.

Ao invés de apresentar uma comparação direta de desempenho entre o CDL e o DCWS, nós podemos notar que o DCWS poderia ser implementado por meio do CDL, com um provável ganho de desempenho. Para isso, bastaria que o algoritmo de particionamento do DCWS fosse implementado nos gerentes de carga locais. Esse gerentes, então, ao invés de migrarem fisicamente as páginas, simplesmente escreveriam os links de forma a respeitar o particionamento. Dessa maneira, nós estabeleceríamos o mesmo particionamento do DCWS, sem incorrer nos custos de migração e reescrita de links. Além disso, como em nosso modelo há replicação, no caso de falha de um servidor, os gerentes poderiam mudar o particionamento, desconsiderando esse servidor, e todo o site continuaria acessível. Por fim, também em função de haver replicação, o uso de bookmarks não traria nenhum problema na implementação do DCWS através do CDL, uma vez que qualquer página poderia ser construída por qualquer servidor.

5.5.1 Escalabilidade

Voltando aos gráficos da figura 5.13, podemos ver os pontos de maior desempenho do cluster para cada método/algoritmo, utilizando 6 servidores. Com base na taxa de transferência média de cada um desses testes, e no resultado do teste de desempenho de um único servidor, como apresentado na figura 4.1 (página 40), podemos montar a tabela 5.2, comparativa de desempenho e indicativa da escalabilidade de cada método/algoritmo.

Como nos mostra a tabela, o método CDL apresenta uma escalabilidade bem maior que a dos outros métodos. Em particular, com o algoritmo `vl_v3`, a escalabilidade é praticamente linear: ao multiplicarmos por 6 o número de servidores, o desempenho, medido como a taxa de transferência do cluster, foi também multiplicado por 6. Já no caso do algoritmo `rr_v3`, o desempenho foi multiplicado por 6.5. Dois fatores explicam esse aumento, acima do desempenho máximo medido de um servidor: a imprecisão do valor máximo, como discutimos acima, e o fato de que esse algoritmo não envia novas requisições para um servidor que começe a se sobrecarregar, distribuindo as requisições entre os servidores não sobrecarregados, o que pode evitar completamente a sobrecarga. O mesmo não ocorre na medição de carga de um único servidor.

É importante observar que nos algoritmos `rr_v3` e `vl_v3` há troca de informações de carga entre todos os servidores do cluster. Como cada servidor se comunica com todos os demais, o custo dessa comunicação cresce com o quadrado do número de servidores. No caso dos testes realizados, não há um comprometimento do desempenho do sistema uma vez que o número de servidores é pequeno.

Esse é um problema bem conhecido na área de sistemas distribuídos [Crowcroft, 1996]. Existem diferentes alternativas para evitar a troca de informações entre todos os servidores, como, por exemplo, uma estratégia híbrida e hierárquica, com coleta de informações de carga centralizada e distribuída, ou o uso de estatísticas que permitam diminuir a freqüência de coleta [Authié et al., 1994]. Assumindo a existência de tais algoritmos, nós não nos preocupamos em evitar o algoritmo quadrático neste trabalho.

Capítulo 6

Conclusões

Vários métodos de distribuição de carga podem ser adotados para criarmos um cluster de servidores que atuem cooperativamente como um único servidor HTTP. No capítulo 2, nós propomos uma classificação desses diferentes métodos, juntamente com uma análise de diversas ferramentas que os empregam.

Através dessa análise, mostramos, na seção 2.3, que nenhuma das ferramentas de distribuição de carga analisadas se adequa a aplicações Web elaboradas, isto é, que utilizem páginas dinâmicas, mantenham sessões com os usuários ou criem particionamentos no cluster. Nas poucas ferramentas que permitem o uso de sessões no nível da aplicação, o desempenho pode cair sensivelmente. As alternativas de particionamento existem apenas baseadas em conteúdo, e sujeitas às mesmas restrições de desempenho associadas ao controle de sessões. Em particular, mostramos que esses problemas advem da falta de integração entre a ferramenta de distribuição de carga e a ferramenta de construção de páginas dinâmicas.

Nós apresentamos, no capítulo 3, uma nova forma de distribuição de carga, chamada CDL, que, estando integrada à ferramenta de construção de páginas dinâmicas, tem acesso às informações necessárias para realizar um controle mais efetivo da distribuição. O mecanismo básico de distribuição se baseia na manipulação dos links internos das páginas geradas pela ferramenta, de forma a distribuir, *a posteriori*, a próxima requisição de cada cliente que acessa o site.

Para provar as idéias que propomos, implementamos um protótipo, cuja estrutura descrevemos na seção 4.1. Utilizando esse protótipo, criamos um ambiente de teste que nos permitiu fazer análises de desempenho de todos os algoritmos de distribuição propostos, segundo as métricas que mostramos na seção 4.2.

Como mostramos ao longo das seções 5.1 e 5.2, o método CDL atende à principal finalidade de um método de distribuição de carga: ser capaz de distribuir eficientemente as requisições dos clientes entre os servidores do cluster. Adicionalmente, o CDL também resolve os problemas associados à manutenção de sessões, sem perda de desempenho, como exposto na seção 5.3. Conforme análise apresentada na seção 5.5, o CDL ainda pode alcançar um ótimo desempenho quando comparado com outros métodos de distribuição de carga.

Além dessas características, o CDL tem uma implementação distribuída. Graças a essa característica, a própria ferramenta de distribuição de carga tem sua capacidade de processamento aumentada em função de um aumento no número de servidores do cluster, refletindo uma real escalabilidade do método. Também em função da distribuição do processamento, o CDL não cria um único ponto de falha no sistema. No caso de falha de um servidor, apenas as requisições sendo tratadas por ele serão prejudicadas.

Estando integrado à ferramenta utilizada para construir o site, esse método de distribuição não representa um custo adicional para a função de distribuição. Como não é necessário um hardware dedicado à função de distribuição, as curvas de custo e desempenho passam a ser lineares com o aumento do número de servidores. Essa característica pode viabilizar o uso em clusters pequenos, talvez economicamente inviáveis através de uma ferramenta baseada em um hardware dedicado.

6.1 Trabalhos Futuros

Em nosso trabalho, nós mantivemos o foco em aplicações que utilizam páginas dinâmicas, o que inclui, em particular, sites de comércio eletrônico. Uma característica que pode ser importante para determinados sites de comércio eletrônico é a possibilidade de se estipular métricas que visem garantir a Qualidade de Serviço (QoS) oferecida.

Pandey et al. [1998] argumentam que as métricas de QoS podem ser baseadas no servidor (*server-centric*) ou no cliente (*client-centric*). As métricas baseadas no servidor não fazem qualquer distinção entre diferentes requisições a uma mesma página, se aplicando apenas à determinação de prioridades entre as páginas que compõem o site. Já as métricas baseadas no cliente permitem que as prioridades de acesso às páginas sejam estipuladas em função das características do cliente que requisita a página.

Em particular, os autores se limitam a explorar métricas baseadas no servidor e propõem uma notação, chamada WebQoSL, através da qual é possível especificar, entre outras, o volume de recursos alocados para determinadas páginas, a disponibilidade de grupos de páginas a cada instante e a especificação de garantias sobre o desempenho. Em seu trabalho, os autores ainda apresentam um servidor HTTP que implementa essas métricas de QoS.

Nessa linha, um trabalho que poderia ser explorado, baseado no CDL, é o uso de métricas de QoS. É importante notar que, ao invés de utilizar um único servidor em todos os links internos de cada página gerada, como fizemos nesse trabalho, é possível utilizar um servidor diferente para cada link; e a escolha desse servidor pode levar os critérios de QoS em consideração. Por exemplo, agrupando diferentes conjuntos de páginas em diferentes conjuntos de servidores, seria possível obter desempenhos diferenciados. Além disso, caso fosse necessário, pontos de entrada poderiam ser utilizados para garantir a distribuição desejada desde o primeiro acesso de cada cliente.

Voltando a um argumento que nós já apresentamos, é razoável que a ferramenta proposta por Pandey et al. [1998] não inclua métricas de QoS baseadas no cliente, uma vez que essa ferramenta é parte do servidor HTTP e, estando dissociada da ferramenta de construção de páginas, não possui informações que configurem os "clientes" como entendidos pelos sites. Em nossa proposta, com a integração das métricas de QoS na ferramenta de construção de páginas, seria possível estabelecer grupos de clientes e utilizar suas características na determinação das prioridades.

Outra característica associada à QoS, também relacionada a aplicações de comércio eletrônico, é a importância das sessões. Como apresentado por Cherkasova e Phaal [1999], em um site de comércio eletrônico, o número de sessões completadas por segundo pode ser mais importante do que o número de requisições completadas por segundo. De fato, como tipicamente a efetivação de uma transação comercial é feita nos últimos acessos de cada usuário, caso uma sessão seja interrompida no meio, todas as requisições anteriores desse usuário podem perder seu valor.

Em seu trabalho, os autores introduzem o conceito de *Session Based Admission Control* (SBAC), como uma forma de garantir "uma chance justa de ser completada para qualquer sessão aceita, independente de seu comprimento" [Cherkasova e Phaal, 1999, página 1] (apesar dos autores não dizerem *como* é feita a identificação de um cliente que retorna ao site).

No nosso entender, um mecanismo do tipo do SBAC poderia, também, ser explorado em um trabalho que visasse a incorporação de QoS na nossa proposta. Afinal, esse critério pode ser uma das métricas de QoS baseadas no cliente. Assim, como em nossa proposta a identificação de um cliente que retorna ao site é simples, a rejeição de novas sessões poderia ser feita, por exemplo, no mesmo nível do mecanismo de redirecionamento que atua nas situações de sobrecarga.

Referências Bibliográficas

Eric Anderson, Dave Patterson, e Eric Brewer. The MagicRouter, an application of fast packet interposing, Maio 1996. Submitted for publication in the 2nd Symposium on Operating System Design and Implementation. http://www.cs.berkeley.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps.

Daniel Andresen, Tao Yang, Vegard Holmedahl, e Oscar H. Ibarra. SWEB: towards a scalable World Wide Web server on multicomputers. Technical Report TRCS95-17, Department of Computer Science, University of California, Santa Barbara, CA, 1995. http://cs-tr.cs.cornell.edu:80/Dienst/UI/1.0/Display/ncstrl.ucsb_cs/TRCS95-17.

Daniel Andresen, Tao Yang, e Oscar H. Ibarra. Toward a scalable distributed WWW server on workstation clusters. *Journal of Parallel and Distributed Computing*, 42:91–100, 1997.

Martin F. Arlitt. Characterizing Web user sessions. Em *Proceedings of the Performance and Architecture of Web Servers Workshop*, Junho 2000.

Martin F. Arlitt e Carey L. Williamson. Web server workload characterization: The search for invariants. Em *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Maio 1996. ftp://ftp.cs.usask.ca/pub/discus/paper.96-3.ps.Z.

Ken Arnold e James Gosling. *The Java Programming Language, 2nd Edition*. Addison-Wesley, 1997.

Gérard Authié, Afonso Ferreira, Jean-Louis Roch, Gilles Villard, Jean Roman, Catherine Roucairol, e Bernard Virot. *Algorithmes parallèles—analyse et conception*. Hermès, 1994.

Luis Aversa e Azer Bestavros. Load balancing a cluster of Web servers using distributed packet rewriting. Technical Report BUCS-TR-99-001, Boston University, Computer Science Department, Janeiro 1999. http://www.cs.bu.edu/techreports/99-001-dpr-cluster-load-balancing.

M. Baentsch, L. Baum, e G. Molter. Enhancing the Web's infrastructure: from caching to replication. *IEEE Internet Computing*, 1(2):18–27, March/April 1997.

Scott M. Baker e Bongki Moon. Distributed cooperative Web servers. Em *Proceedings of the 8th International World Wide Web Conference*, Maio 1999. http://www8.org/w8-papers/2a-webserver/distributed/distributed.html.

Paul Barford e Mark Crovella. Generating representative Web workloads for network and server performance evaluation. Em *Proceedings of ACM SIGMETRICS*, páginas 151–160, Novembro 1997. http://cs-www.bu.edu/techreports/97-006-surge.ps.Z.

Paul Barford e Mark E. Crovella. A performance evaluation of hyper text transfer protocols. Em *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, páginas 188–197, Maio 1999. http://www.cs.bu.edu/techreports/1998-016-http-protocols-evaluation.ps.Z.

Azer Bestavros, Mark Crovella, Jun Liu, e David Martin. Distributed packet rewriting and its application to scalable server architectures. Em *Proceedings of ICNP'98: The 6th IEEE International Conference on Network Protocols*, Outubro 1998. http://www.cs.bu.edu/fac/best/res/papers/icnp98.ps.

Krishna Bharat e Andrei Broder. Mirror, mirror on the web: a study of host pairs with replicated content. Em *Proceedings of the 8th International World Wide Web Conference*, Maio 1999. http://www8.org/w8-papers/4c-server/mirror/mirror.html.

Thomas P. Brisco. DNS support for load balancing, Abril 1995. RFC 1794. http://www.ietf.org/rfc/rfc1794.txt.

Mark R. Brown. FastCGI specification, Abril 1996. http://www.fastcgi.com.

Pei Cao, Jin Zhang, e Kevin Beach. Active cache: caching dynamic contents (objects) on the Web. Em *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Setembro 1998. http://www.cs.wisc.edu/~cao/papers/active-cache.html.

Valeria Cardellini, Michele Colajanni, e Philip S. Yu. DNS dispatching algorithms with state estimators for scalable Web server clusters. *World Wide Web*, 2(3):101–113, Julho 1999a. http://traianus.ce.uniroma2.it/dws/WWW99.ps.

Valeria Cardellini, Michele Colajanni, e Philip S. Yu. Redirection algorithms for load sharing in distributed Web server systems. Em *19th International Conference on Distributed Computing Systems*. IEEE, Junho 1999b. http://traianus.ce.uniroma2.it/dws/icdcs99.ps.

Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, e Michael Rabinovich. Web proxy caching: the devil is in the details. Em *Workshop on Internet Server Performance*, Junho 1998. http://www.cs.wisc.edu/~cao/WISP98/final-versions/anja.ps.

Common gateway interface, 1995. W3C—World Wide Web Consortium. http://www.w3 .org/CGI.

Ludmila Cherkasova e Peter Phaal. Session based admission control: a mechanism for improving performance of commercial Web sites. Em *Proceedings of Seventh International Workshop on Quality of Service*, 31 de Maio–4 de Junho 1999. IEEE/IFIP event. http://www.hpl.hp.com/personal/Lucy_Cherkasova/papers/qos99-paper.pdf.

Michele Colajanni, Philip S. Yu, e Valeria Cardellini. Dynamic load balancing in geographically distributed heterogeneous Web servers. Em *18th International Conference on Distributed Computing Systems*, páginas 295–302. IEEE, Maio 1998a. http://traianus.ce.uniroma2.it/dws/icdcs98.ps.

Michele Colajanni, Philip S. Yu, e Daniel M. Dias. Scheduling algorithms for distributed Web servers. Em *Proceedings of the 17th International Conference on Distributed Computing Systems*, páginas 169–176, Maio 1997.

Michele Colajanni, Philip S. Yu, e Daniel M. Dias. Analysis of task assignment policies in scalable distributed Web server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600, Junho 1998b. http://traianus.ce.uniroma2.it/dws/TPDS98.ps.

Netscape Communications Corporation. NSAPI programmer's guide for enterprise server 4.0, Setembro 1999a. http://developer.netscape.com/docs/manuals/enterprise/40/nsapi/nsapipdf.zip.

Netscape Communications Corporation. Programmer's guide to enterprise server 4.0, Setembro 1999b. http://developer.netscape.com/docs/manuals/ enterprise/40/pg/pgpdf.zip.

Jon Crowcroft. *Open Distributed Systems*. UCL, 1996.

Om P. Damani, P. Emerald Chung, Yennun Huang, Chandra Kintala, e Yi-Min Wang. ONE-IP: Techniques for hosting a service on a cluster of machines. Em *Proceedings of the 6th International World Wide Web Conference*, Abril 1997. http://www.scope.gmd.de/info/www6/technical/paper196/paper196.html.

James D. Davidson. Java Servlet specification, version 2.2, Outubro 1999. http://java.sun.com/products/servlet.

Luiz H. de Figueiredo, Roberto Ierusalimschy, e Waldemar Celes. Lua—an extensible embedded language. *Dr. Dobb's Journal*, 21(12):26–33, 1996.

Daniel M. Dias, William Kish, Rajat Mukherjee, e Renu Tewari. A scalable and highly available Web server. Em *Proceedings of IEEE COMPCON'96*, páginas 85–92, 1996. http://www.research.ibm.com/webvideo/compcon96.ps.

Fred Douglis, Antonio Haro, e Michael Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. Em *USENIX Symposium on Internet Technology and Systems*, Dezembro 1997. http://www.research.att.com/~douglis/papers/hpp.ps.

Kjeld B. Egevang e Paul Francis. The IP network address translator (NAT), Maio 1994. RFC 1631. http://www.ietf.org/rfc/rfc1631.txt.

Anja Feldmann, Ramón Cáceres, Fred Douglis, Gideon Glass, e Michael Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. Em *IEEE Infocom '99 Conference*, Março 1999. http://www.research.att.com/~anja/feldmann/papers/infocomm99_proxim.ps.gz.

Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, e Tim Berners-Lee. Hypertext transfer protocol – HTTP/1.1, Junho 1999. RFC 2616. http://www.ietf.org/rfc/rfc2616.txt.

Ben Forta, Nates Weiss, Ashley King, Mike Dinowitz, Gerry Libertelli, e Jonathan Bellock. *Advanced ColdFusion 4.0 Application Development*. QUE, 1999.

Chris Gage. IBM SecureWay Network Dispatcher version 2.1, Abril 1999. White Paper, IBM Corporation. ftp://ftp.software.ibm.com/software/network/dispatcher/whitepapers/nd21wp.pdf.

Germán Goldszmidt e Guerney Hunt. NetDispatcher: A TCP connection router. Research Report RC 20853, IBM T.J. Watson Research Center, Maio 1997. ftp://ftp.software.ibm.com/software/network/dispatcher/whitepapers/research_tr.pdf.

Anna Hester, Renato Borges, e Roberto Ierusalimschy. CGILua: A multi-paradigmatic tool for creating dynamic WWW pages. Em *XI Simpósio Brasileiro de Engenharia de Software*, páginas 347–360, http://www.tecgraf.puc-rio.br/~rborges, 1997.

Anna Hester, Renato Borges, e Roberto Ierusalimschy. Building flexible and extensible Web applications with Lua. Em *Webnet'98 World Conference of the WWW, Internet & Intranet*, páginas 427–432, Novembro 1998a.

Anna Hester, Renato Borges, e Roberto Ierusalimschy. Building flexible and extensible Web applications with Lua. *Journal of Universal Computer Science*, 4(9):748–762, 1998b. http://www.iicm.edu/jucs_4_9/building_flexible_and_extensible.

Anna M. Hester. A ferramenta CGILua em múltiplas APIs com o servidor HTTP. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Julho 1999.

Bernardo Huberman, Peter Pirolli, James Pitkow, e Rajan Lukose. Strong regularities in World Wide Web surfing. *Science*, 280:95–97, 1998.

Jason Hunter e William Crawford. *Java Servlet Programming*. O'Reilly & Associates, Outubro 1998.

Roberto Ierusalimschy, Luiz H. de Figueiredo, e Waldemar Celes. Lua—an extensible extension language. *Software: Practice & Experience*, 26(6):635–652, 1996.

Roberto Ierusalimschy, Luiz H. de Figueiredo, e Waldemar Celes. *Reference Manual of the Programming Language Lua 3.1*. TeCGraf, Computer Science Department, PUC-Rio, 1998. `http://www.tecgraf.puc-rio.br/lua/manual`.

Microsoft internet information server training, Março 1998.

Eric D. Katz, Michelle Butler, e Robert E. McGrath. A scalable HTTP server: the NCSA prototype. *Computer Networks and ISDN Systems*, 27:155–163, 1994.

Balachander Krishnamurthy, Jeffrey C. Mogul, e David M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. Em *Proceedings of the 8th International World Wide Web Conference*, Maio 1999. `http://www8.org/w8-papers/5c-protocols/key/key.html`.

David M. Kristol e Lou Montulli. HTTP state management mechanism, Fevereiro 1997. RFC 2109. `http://www .ietf.org/rfc/rfc2109.txt`.

Thomas T. Kwan, Robert E. McGrath, e Daniel A. Reed. NCSA's World Wide Web server: design and performance. *IEEE Computer*, 28:68–74, 1995. `http://www-pablo.cs.uiuc.edu/Publications /Papers/ComputerWWW.ps.gz`.

Ben Laurie e Peter Laurie. *Apache: The Definitive Guide, Second Edition*. O'Reilly & Associates, Fevereiro 1999.

David L. Lills. Improved algorithms for synchronizing computer network clocks. Em *Proceedings of the SIGCOMM'94 Symposium on Communications Architectures and Protocols*, páginas 317–327, Agosto 1994.

Mark Lutz. *Programming Python*. O'Reilly & Associates, 1996.

P. Mockapetris. Domain names - implementation and specification, Novembro 1987. RFC 1035. `http://www.ietf.org/rfc/rfc1035.txt`.

Jeffrey C. Mogul. Network behavior of a busy Web server and its clients. Research report 95/5, DEC Western Research Laboratory, Outubro 1995a. `http://research.compaq.com/wrl/techreports /abstracts/95.5.html`.

Jeffrey C. Mogul. Operating systems support for busy Internet servers. Technical note TN/49, DEC Western Research Laboratory, Maio 1995b. `http://research.compaq.com/wrl/techreports /abstracts/TN-49.html`.

Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, e Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. Em *ACM SIGCOMM '97 Conference*, Setembro 1997. `http://www.acm.org/sigcomm/sigcomm97/papers/p156.html`.

D. Mosedale, W. Foss, e R. McCool. Lessons learned administering Netscape's Internet site. *IEEE Internet Computing*, 1(2):28–35, March/April 1997.

Antoine Mourad e Huiqun Liu. Scalable Web server architectures. Em *Proceedings of the 2nd IEEE Symposium on Computer and Communications*, páginas 12–16, Julho 1997.

John Ousterhout. *Tcl and TK Toolkit*. Addison-Wesley, 1994.

Raju Pandey, J. Fritz Barnes, e Ronald Olsson. Supporting Quality of Service in HTTP servers. Em *Proceedings of the SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'98)*, Junho 1998. `http://www.acm.org/pubs/articles/proceedings/podc/277697/p247-pandey/p247-pandey.pdf`.

Gregory F. Pfister. *In Search of Clusters, 2nd Edition*. Prentice Hall PTR, 1998.

A. Singhai, S.-B. Lim, e S. R. Radia. The SunSCALR framework for Internet servers. Em *IEEE Fault Tolerant Computing Systems (FTCS'98)*, Junho 1998.

Alteon Web Systems. Next steps in server load balancing, 1999a. White Paper. `http://www.alteonwebsystems.com/products/white_papers/SLB/index.shtml`.

Cisco Systems. Scaling the World Wide Web, 1996. White Paper. `http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/swww_wp.pdf`.

Cisco Systems. Scaling the Internet Web servers, 1997. White Paper. `http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/scale_wp.htm`.

Cisco Systems. MultiNode load balancing, 1999b. White Paper. `http://www.cisco.com/warp/public/cc/cisco/mkt/iworks/data/mnlb/tech/mnlb_wp.htm`.

Cristina Ururahy e Noemi L. Rodriguez. ALua: An event-driven communication mechanism for parallel and distributed programming. Em *Proceedings of ISCA 12th International Conference on Parallel and Distributed Computing Systems (PDCS'99)*, páginas 108–113, Agosto 1999.

Amin Vahdat, Michael Dahlin, Thomas Anderson, e Amit Aggarwal. Active Names: Flexible location and transport of wide-area resources. Em *USENIX Symposium on Internet Technologies and Systems*, Outubro 1999. `http://www.cs.utexas.edu/users/dahlin/papers/usits99-aname.ps`.

Larry Wall, Nate Patwardhan, Ellen Siever, David Futato e Brian Jepson. *Perl Resource Kit – UNIX Edition*. O'Reilly & Associates, 1997. `http://www.oreilly.com/catalog/prkunix/excerpt/UGtoc.html`.

Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, e David Culler. Using Smart Clients to build scalable services. Em *Proceedings of the 1997 USENIX Conference*, Janeiro 1997. `http://now.CS.Berkeley.EDU/SmartClients/usenix97.ps`.

G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, 1949.

Apêndice A

Dados Adicionais

A.1 Especificação do Cluster do DI/PUC–Rio

- 1 switch IBM, 10 Mbit/s
- 32 micro-computadores IBM-PC
  - processador Pentium II 300 MHz
  - 32 Mb de memória RAM
  - placa Ethernet de 10 Mbit/s
  - configurado com 64 Mb de swap
  - conteúdo do arquivo `/proc/cpuinfo`:

```
processor     : 0
cpu           : 686
model         : Pentium II (Deschutes)
vendor_id     : GenuineIntel
stepping      : 2
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
fpu           : yes
fpu_exception : yes
cpuid         : yes
wp            : yes
flags         : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca
                cmov 16 17 mmx 24
bogomips      : 397.31
```