Balanceamento Dinâmico de Requisições em Cluster de Servidores Web

Jô Sato, João A. Martini, Ronaldo A. L. Gonçalves
Universidade Estadual de Maringá - Departamento de Informática
Programa de Pós-Graduação em Ciência da Computação
Av. Colombo, 5790 – Bloco 19 – 87.020-900 – Maringá – PR – Brasil
jo@londrina.pr.gov.br,{jangelo,ronaldo}@din.uem.br

Resumo

Informações provenientes de movimentações bancárias, transações comerciais, pesquisas educacionais, bate-papo e jogos interativos, entre outros, têm crescido intensamente na Internet. Os provedores de serviços precisam estar preparados para sustentar esta demanda crescente de informação e comunicação. Neste sentido, a distribuição de requisições entre vários servidores web de forma balanceada tem se tornado uma estratégia fundamental para garantir melhor qualidade no serviço. O presente trabalho descreve e discute questões arquiteturais e operacionais sobre o balanceamento de requisições, principalmente o dinâmico, abordando conceitos, técnicas e soluções. Experimentos reais foram realizados em diferentes configurações sobre um cluster de servidores Apache não dedicado exclusivamente ao serviço web. O módulo de balanceamento de carga mod_proxy_balancer foi usado com sobrecargas sintéticas intensivas e um novo método de balanceamento chamado byquery foi proposto e avaliado. Os resultados mostram que o balanceamento neste tipo de sistema será mais eficiente se a carga de rede externa ao serviço web for detectada e usada no fator de balanceamento.

1. Introdução

A popularização da Internet e o conseqüente aumento no número de usuários fizeram com que a web se tornasse a interface padrão para acesso a aplicações e serviços remotos de sistemas de informação. Atualmente, a disponibilização de aplicações e serviços web permite movimentações bancárias, declaração de imposto de renda, solicitação de certidões e compras, entre outras atividades. Isto produz um tráfego intenso de dados na Internet, exigindo que os provedores de conteúdo e serviços estejam preparados para atendê-lo. Arquiteturas escaláveis, que crescem de acordo com a demanda e reduzem o tempo de resposta dos servidores web, são uma tendência contemporânea para o atendimento de requisições.

Segundo Cardellini [1], a velocidade das redes de computadores cresce de maneira mais rápida que a capacidade de processamento dos servidores, tornando o lado do servidor web um possível gargalo para todo o sistema. Contribui para isto, a substituição dos mecanismos básicos de busca de conteúdo estático por complexos mecanismos para disponibilização de conteúdo dinâmico, utilizados principalmente por aplicações e serviços web que requerem maiores recursos computacionais e maior segurança na comunicação de dados. Um site popular pode receber milhares de requisições de acesso por segundo, exigindo alta disponibilidade e capacidade de atendimento às requisições. Tais características podem ser obtidas com o balanceamento de requisições entre vários servidores de um cluster [1, 2, 3, 4, 5, 6].

Com o balanceamento de requisições, todas as requisições dos usuários são repassadas de forma balanceada entre as máquinas do cluster, de acordo com fatores tais como capacidade de processamento, quantidade de requisições, carga de trabalho e tráfego de rede. Um cluster web é comumente dedicado ao atendimento exclusivo de requisições web. Entretanto, em instituições de pesquisa e desenvolvimento, principalmente aquelas com restrições, o uso de cluster de servidores web não dedicado, na qual as máquinas também podem ser utilizadas individualmente para atender a serviços específicos e independentes do serviço web, é uma solução economicamente atrativa.

O presente trabalho contextualiza o estado da arte no balanceamento de requisições e avalia o uso do módulo de balanceamento mod_proxy_balancer do servidor Apache em um cluster de servidores não dedicados exclusivamente ao serviço web. Com base em experimentos reais, um novo método de balanceamento chamado byquery foi proposto e avaliado. Os resultados mostram a sua eficiência.

Nas próximas seções o presente artigo descreve a organização de um cluster web, discorre sobre o balanceamento de requisições, apresenta o servidor web Apache, analisa o módulo de balanceamento de

requisições do mod_proxy_balancer em experimentos práticos, propõe um novo método chamado byquery, discute os resultados e mostra as conclusões. As referências bibliográficas são listadas na última seção.

2. Arquitetura de um *cluster web*

Um servidor web é um aplicativo responsável por fornecer os dados solicitados pelo aplicativo navegador do cliente (browser), disponibilizando páginas de textos, gráficos, fotos ou qualquer outro tipo de objeto em tempo real. Pode também receber dados, processá-los e enviar os resultados para que o cliente possa tomar a ação desejada.

Quando o usuário solicita um acesso a um site web na Internet, o cliente Web solicita o endereço IP deste site para o servidor DNS da rede local. O servidor DNS verifica se este site se encontra na cache e, em caso afirmativo, retorna a informação para o cliente. Caso contrário, ele repassa esta solicitação aos servidores da hierarquia DNS até atingir o servidor DNS autoritário do domínio solicitado, que confirma ou não a existência de tal site em sua base de dados. Uma vez confirmada a existência do site e recebido o endereço IP do mesmo, o cliente efetua requisições diretamente ao servidor web do site desejado. Assim que recebe a requisição, o servidor web analisa-a e, de acordo com restrições de segurança, e transmite os resultados para o navegador do cliente.

Conforme Cardelline [1,3] e Aversa [2], para sistemas com mais de um servidor web, há duas classes definidas de acordo com a entidade que distribui as requisições: o sistema web distribuído e o cluster web.

O sistema web distribuído é o mais antigo dos modelos de balanceamento de requisições, consistindo de nós localmente distribuídos no qual cada nó possui um endereço IP publicamente visível aos clientes. O roteamento de requisições para um nó servidor é feito da forma convencional (por um servidor DNS qualquer), durante a fase de resolução de endereço IP. O servidor DNS tem um controle limitado sobre o direcionamento das requisições, pois existem muitos servidores de nomes intermediários que podem fazer cache do mapeamento "nome↔endereço IP" para reduzir tráfego de rede. Os clientes também podem fazer cache local dessas informações.

Já no cluster web, um nó do cluster (ou um dispositivo de encaminhamento específico para esta finalidade) é configurado como redirecionador, sendo o responsável pela tarefa de distribuir as requisições dos clientes aos servidores do cluster, conforme Figura 1. Somente o endereço IP do redirecionador é visível publicamente aos clientes. Os demais nós têm seus endereços "mascarados" pelo redirecionador.

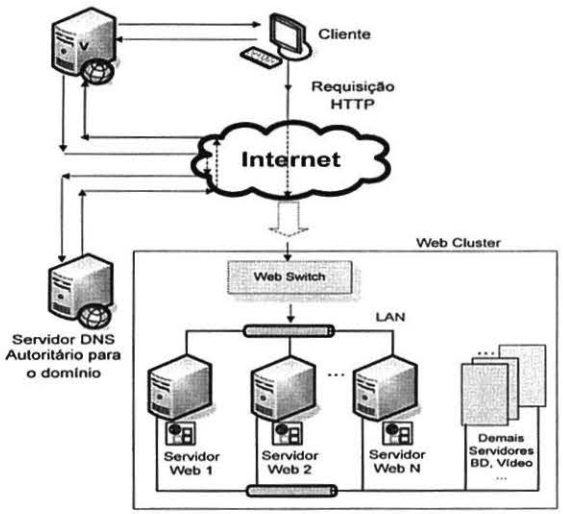

Figura 1. Arquitetura de um cluster web

Cada nó servidor do cluster normalmente possui seu próprio disco e sistema operacional. O controle do endereçamento das requisições dos clientes pode ser feito em hardware ou em software. No exemplo da Figura 1, o servidor DNS autoritário para o domínio não faz parte do cluster, pois não possui função no sistema de roteamento de requisições dos clientes.
Em um cluster web, cada nó deve possuir um endereço privado (interno) de forma que possa ser identificado de maneira única pelo redirecionador e possa receber as requisições de clientes. Esta identificação pode ocorrer em diferentes camadas do modelo OSI, principalmente em nível das camadas de transporte/enlace (camadas 4/2), transporte/rede (camadas 4/3) e aplicação (camada 7).

Redirecionamento na camada 4/2

Nesta abordagem, existe um servidor de despacho na rede que seleciona um servidor do cluster usando seu endereço físico (MAC) sem necessidade de modificação do cabeçalho do pacote IP, tanto do lado cliente como do servidor Web. Esta solução é transparente para ambos os lados, uma vez que não necessita de reescrita de pacotes. O servidor de despacho pode definir políticas de despacho dinamicamente de acordo com a carga e a disponibilidade do servidor. No entanto, existe uma maior complexidade para sua implementação.

Redirecionamento na camada 4/3

O redirecionamento na camada 4/3 pode utilizar as seguintes técnicas [3]:

- Reescrita Única de Pacotes: existe um único equipamento (redirecionador) visível na Internet que recebe a solicitação e encaminha para um dos servidores do cluster através de um algoritmo round-robin. Ao definir o servidor que irá atender as requisições do cliente, o redirecionador passa a reescrever o destino dos pacotes IP com o endereço IP do servidor escolhido, e este passa a responder diretamente ao cliente. É importante observar que a comunicação cliente → nó servidor Web escolhido sempre passa pelo redirecionador. Já a comunicação servidor Web → cliente é direta, sem passar pelo redirecionador.
- Reescrita Dupla de Pacotes: Na reescrita dupla de pacotes, também existe um único equipamento (redirecionador) visível na Internet que recebe a solicitação e encaminha para um dos servidores do cluster através de um algoritmo round-robin. Definido o servidor que irá atender as requisições do cliente, o redirecionador passa a reescrever o destino dos pacotes IP com o endereço IP do servidor escolhido, da mesma forma que reescreve a origem dos pacotes com o endereço do cliente. A comunicação cliente → nó servidor Web e nó servidor Web → cliente é feita pelo redirecionador.

O Linuxvirtualserver [7] é um exemplo de redirecionador que trabalha na camada 4. Consiste em implementações que quando aplicadas ao kernel do Linux adicionam esta funcionalidade a este sistema operacional. O KTCPVS (Kernel TCP Virtual Server) (Linux Virtual Server Project) é um projeto baseado no Linuxvirtualserver que implementa funcionalidades para operar também na camada 7.

Redirecionamento na camada 7

O redirecionador tem controle do conteúdo da requisição HTTP, podendo usar algoritmos de balanceamento mais eficazes e prover requisitos de acordo com os diferentes tipos de conteúdo. No entanto, ele é mais lento se comparado ao redirecionamento na camada 4/3, pois existe um gasto excedente de tempo de processamento para análise das requisições. O Apache com mod_proxy_balancer é um exemplo de redirecionador que atua na camada 7. Entre as possíveis configurações para esta técnica, estão as seguintes.

- Gateway TCP: Após o redirecionador definir o nó que irá atender a requisição, uma conexão é estabelecida entre eles. O redirecionador passará a agir como uma porta de ligação entre o servidor e o cliente.
- Junção TCP (TCP Splice): Existe uma conexão entre o cliente e o redirecionador, também chamado de proxy, e uma conexão diferente entre o proxy e o servidor. Os pacotes são encaminhados a nível de camada de rede.

3. Breve descrição do Apache

O Apache é atualmente o servidor web mais utilizado no mundo, segundo a Netcraft [8]. É um software livre e de código aberto, possuindo versões para sistemas operacionais como Windows, Novell, Linux e diversos outros do padrão POSIX (Unix, FreeBSD e outros). Foi criado em 1995 por Rob McCool, então funcionário do NCSA (National Center for Supercomputing Applications), na Universidade de Illinois. Ele é configurável, extremamente robusto e objetivo alto desempenho e disponibilidade, sendo continuamente atualizado por uma equipe de voluntários (Apache Group) que busca incluir neste servidor web as inovações pesquisadas no meio acadêmico e pelo próprio grupo.

Este servidor é compatível com o protocolo HTTP versão 1.1 e suas funcionalidades são mantidas através de uma estrutura em módulos, permitindo inclusive que o usuário escreva seus próprios módulos utilizando a API do software [9]. O módulo mod_proxy permite ao Apache operar como proxy e proxy reverso. O proxy reverso é geralmente usado para distribuir as requisições entre vários servidores web, ou prover cache de páginas html para servidores web mais lentos. O uso de proxy reverso permite que vários servidores compartilhem uma mesma URL (Uniform Resource Locator). Para o cliente web, o servidor proxy reverso representa o servidor web do domínio.

O módulo mod_proxy_balancer, disponível a partir da versão 2.1 do Apache, adiciona a funcionalidade de balanceamento de requisições ao mod_proxy, provendo suporte para os protocolos HTTP, FTP e AJP13. Atualmente, o mod_proxy_balancer possui 2 métodos de balanceamento: por quantidade de requisições (byrequests - request counting) e por quantidade de sobrecarga de tráfego (bytraffic - weighted traffic counting). A Figura 2 apresenta o processo de análise de uma requisição pelo mod_proxy_balancer.

O balanceamento de requisições do Apache exclui os nós que falham do rol de processadores ativos, em tempo de execução. A definição do método de balanceamento é feita pela diretiva lbmethod, previamente definida no arquivo de configurações do Apache. Uma questão importante é a funcionalidade do fator de balanceamento, loadfactor, existente no mod_proxy_balancer. Este parâmetro determina qual será o peso de cada servidor durante a distribuição das requisições, variando de 1 a 100. Quanto maior seu

valor, maior será o número de requisições que um servidor membro do balanceamento irá atender.

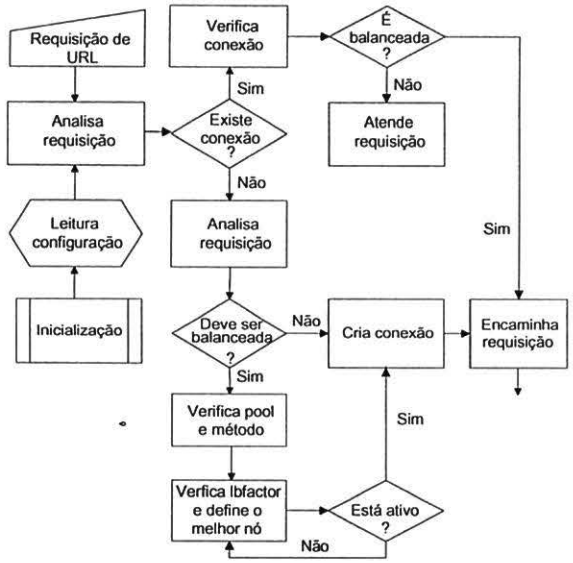

Figura 2. Requisição no mod_proxy_balancer

A Figura 3 mostra um exemplo de configuração para balanceamento para quatro servidores web, método byrequest e loadfactor igual a 1, o qual foi utilizado nos primeiros testes descritos na próxima seção.

```
ProxyPass /din22 balancer://mestre lbmethod=byrequests
ProxyPassReverse /din22/ http://192.168.1.1:8080/din22/
ProxyPassReverse /din22/ http://192.168.1.2:8080/din22/
ProxyPassReverse /din22/ http://192.168.1.3:8080/din22/
ProxyPassReverse /din22/ http://192.168.1.4:8080/din22/
<Proxy balancer://mestre/din22>
  BalancerMember http://192.168.1.1:8080/din22 loadfactor=1
  BalancerMember http://192.168.1.2:8080/din22 loadfactor=1
  BalancerMember http://192.168.1.3:8080/din22 loadfactor=1
  BalancerMember http://192.168.1.4:8080/din22 loadfactor=1
</Proxy>
```

Figura 3. Configuração do mod_proxy_balancer

3. Avaliando o *mod_proxy_balancer*

O presente trabalho analisa o balanceamento de requisições provido pelo módulo mod_proxy_balancer do Apache com o objetivo de identificar as ineficiências de seu uso em clusters de servidores web não dedicados e assim propor melhorias. Os dois métodos de balanceamento providos pelo módulo mod_proxy_balancer, byrequests e bytraffic, foram avaliados e os tempos para atendimento das requisições foram analisados. Para facilitar a investigação, os experimentos foram realizados com uma ferramenta que simula requisições a URLs e avalia o desempenho em termos de tempo de resposta, de forma sistemática e monitorada. Entre as ferramentas deste tipo, três foram identificadas: ApacheBench [10], Http_load [11] e o Httperf [12].

As três ferramentas foram pré-avaliadas neste trabalho e, apesar de apresentarem funcionalidades e características comportamentais semelhantes, a ferramenta Httperf foi escolhida pelo fato de não apresentar limitações no tamanho das páginas (tal como ocorre com a ferramenta ApacheBench) e no número máximo de requisições (tal como ocorre com a ferramenta Http_load).

Ambiente Experimental

Os testes foram realizados com o Apache versão 2.2.4, instalado em todas as máquinas de um cluster composto de um nó servidor mestre e oito nós servidores escravos (S1 a S8). O mestre tem processador Amd 1.0GB e 512MB de RAM. Os escravos de S1 a S4 são processadores Pentium IV 1.8GHz com 512MB de RAM e os escravos de S5 a S8, Pentium IV Hyperthreading de 3.0GHz com 512MB de RAM, compondo o que denominamos de escravos mais lentos e escravos mais rápidos, respectivamente. A Figura 4 mostra a sua organização.

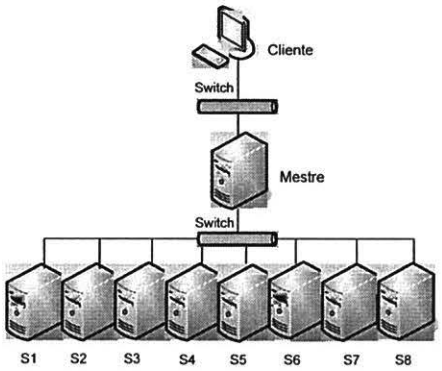

Figura 4. Organização do cluster utilizado

Apenas o Apache instalado no nó mestre foi estendido com o módulo mod_proxy_balancer, desempenhando a função de redirecionador de requisições. Cada uma das URLs foi configurada com páginas html distintas, mas com o mesmo tamanho de 13 Kb (para melhor cercar os experimentos e facilitar a análise dos resultados). O caminho da requisição foi previamente definido no arquivo httpd.conf do mestre. Neste trabalho, o Httperf foi utilizado para simular blocos de 100 mil requisições de clientes ao servidor web. Os resultados são apresentados em termos do

tempo necessário para atender tais blocos de requisições.

As URLs foram hospedadas sobre os nós escravos e os acessos foram realizados de uma máquina que efetua solicitações ao nó mestre. A cada teste, os serviços web do mestre e dos escravos eram reinicializados, sendo finalizados após o término dos testes. Os arquivos de log eram renomeados para serem posteriormente analisados. A partir da análise destes experimentos um novo método de balanceamento foi proposto para o mod_proxy_balancer, chamado byquery, para ser usado em situações de compartilhamento do cluster, o qual mostrou maior desempenho neste caso.

Desempenho em condições normais

Primeiramente, com o intuito de observar o comportamento do mod_proxy_balancer em situações normais, sem sobrecarga de utilização do cluster web, experimentamos apenas redirecionar as requisições de 12 diferentes maneiras, conforme mostra a Tabela 1.

Teste	Redirecionamento
0	Mestre atende sozinho
Usando os escravos mais lentos	
1	S1
2	S1; S2
3	S1; S2; S3
4	S1; S2; S3; S4
Usando os escravos mais rápidos	
5	S5
6	S5; S6
7	S5; S6; S7
8	S5; S6; S7; S8
Usando todos os escravos	
9	S1; S2; S3; S4; S5
10	S1; S2; S3; S4; S5; S6
11	S1; S2; S3; S4; S5; S6; S7
12	S1; S2; S3; S4; S5; S6; S7; S8

Tabela 1: Redirecionamentos das requisições

As requisições recebidas pelo mestre eram redirecionadas a um subconjunto de escravos em cada teste realizado, divididos em 3 grupos. Por exemplo, no teste 3, as requisições recebidas pelo mestre eram redirecionadas aos servidores S1, S2 e S3. Neste caso, o mestre apenas utilizava seu processamento na atividade de redirecionamento. No teste 0, o mestre executou sozinho todas as requisições, ou seja, não houve redirecionamento. A Figura 5 mostra os tempos para atendimento de solicitações a páginas estáticas usando o método byrequests. Pode-se observar que o uso do balanceamento aumenta o tempo de resposta, contrariando nossa expectativa inicial que seria a de aumento do desempenho. Assim, fica claro que em situações normais, este tipo de cluster web é capaz de prover maior disponibilidade, mas não de prover maior desempenho.

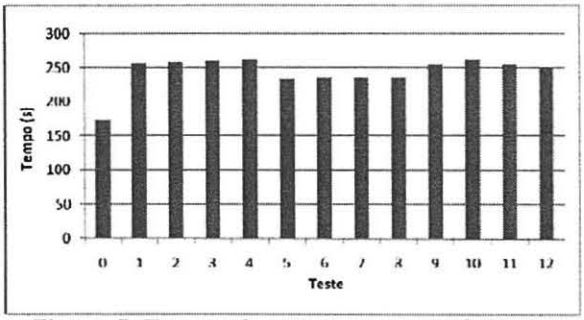

Figura 5. Tempo de resposta com páginas estáticas usando o byrequests

Nos testes de 1 a 4, usando as máquinas mais lentas, os tempos de atendimento são maiores. Nos testes de 5 a 8, utilizando máquinas mais rápidas, os tempos são menores. Nos testes de 9 a 12, na qual máquinas mais rápidas foram utilizadas em conjunto com as máquinas mais lentas, os tempos praticamente representavam uma média dos tempos maiores e menores. Em cada grupo de simulação, os tempos são praticamente os mesmos entre si, independentemente do número de nós envolvidos no redirecionamento.

Analisando o arquivo de log do Apache, verificou-se que as requisições são distribuídas igualmente entre os nós participantes do balanceamento, característica do algoritmo round-robin, e que o fator de balanceamento é o mesmo para todos os servidores. Assim, não importa se o balanceamento possui 1, 2, 3 ou 4 máquinas envolvidas, pois o tempo é determinado pela velocidade de atendimento do mais lento.

Experimentamos também redirecionar requisições a páginas dinâmicas. Neste caso, testamos apenas 10 requisições a páginas PHP que efetuam o cálculo do PI. O resultado é mostrado na Figura 6. Neste caso, observa-se certa melhora no desempenho com a utilização do balanceamento de requisições. Percebe-se que à medida que máquinas mais rápidas passam a participar do balanceamento, o tempo diminui.

O método bytraffic também foi experimentado, nas mesmas condições das simulações até aqui relatadas e os tempos obtidos se apresentaram semelhantes aos tempos mostrados na Figura 5. Uma possível justificativa para a inércia do balanceamento na

redução do tempo de atendimento apóia-se no fato de não ter havido qualquer interferência de carga de rede.

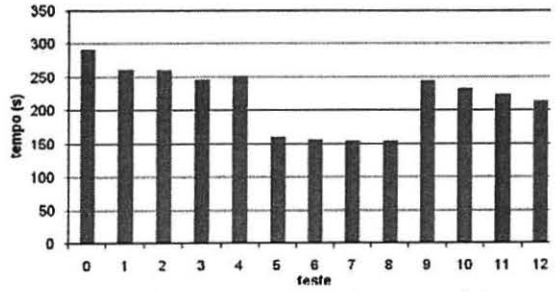

Figura 6: Tempo de resposta com páginas dinâmicas usando o byrequests

Desempenho com tráfego de rede não detectado

Vislumbra-se neste trabalho a possibilidade de usar o cluster de forma não dedicada ao serviço web, como no caso de centros com poucos recursos computacionais ou como forma de maximizar o uso do cluster sem prejudicar o serviço web. Neste contexto, para simular a existência de aplicações compartilhando o cluster web, aplicamos sobrecarga monitorada com a ajuda do wget, um programa para efetuar download de arquivos, instalado nos servidores escravos. As sobrecargas foram aplicadas externamente ao serviço web e assim não eram detectadas pelo mod_proxy_balancer. Os testes 4 e 8 foram experimentados novamente neste novo cenário, usando somente o método bytraffic, uma vez que o gargalo no sistema seria a comunicação.

A sobrecarga foi aplicada gradativamente em até 4 nós, no intuito de simular oscilações no tráfego de rede e então analisar o mod_proxy_balancer. Os testes utilizando os escravos S1 a S4 foram realizados ao mesmo tempo em que o programa wget instalado nos escravos S5 a S8 efetuava downloads de vários arquivos de 1 e 1,5GB localizados nos escravos S1 a S4. O nó mestre, a partir de diretivas do módulo mod_proxy, informava a localização destes arquivos. Os testes utilizando os escravos S5 a S8 foram realizados de forma análoga.

Note que as máquinas que simulam clientes efetuando downloads se encontram no mesmo barramento de rede que terá, teoricamente, o dobro de carga, influindo diretamente do tempo de atendimento às requisições dos testes. No entanto, este ambiente será o mesmo para todos os testes realizados com sobrecarga de rede.

No primeiro teste, baseado no teste 4, identificado por "4.1", somente o escravo S5 solicita os downloads de arquivos. No segundo teste, o "4.2", os escravos S5 e S6 solicitam os downloads e assim sucessivamente até que no quarto teste, o "4.4", os escravos S5, S6, S7 e S8 realizam os downloads. Para cada teste primeiramente eram iniciados os downloads e em seguida era executada a ferramenta httperf na máquina cliente. O mesmo procedimento foi adotado para os testes utilizando a configuração do teste 8. Os tempos obtidos estão nas Tabelas 2 e 3.

Observa-se que todos os tempos obtidos nos atendimentos foram muito superiores aos apresentados na Figura 5 (aproximadamente 260 e 238 segundos, para os testes 4 e 8, respectivamente). Quanto maior o número de nós com sobrecarga, maior o tempo necessário para o atendimento às requisições, atingindo um aumento máximo acima de 3.000%.

Teste	Nó Sobrecarregado	Tempo (s)	Aumento sobre o teste 4 inicial
4.1	S1	1862,56	716,08%
4.2	S1; S2	3992,21	1534,85%
4.3	S1; S2; S3	5989,90	2302,89%
4.4	S1; S2; S3; S4	6660,27	2560,62%

Tabela 2: Teste 4 com sobrecarga não detectada

Teste	Nó Sobrecarregado	Tempo (s)	Aumento sobre o teste 8 inicial
8.1	S5	3653,28	1532,09%
8.2	S5; S6	5500,62	2306,82%
8.3	S5; S6; S7	4492,50	1884,04%
8.4	S5; S6; S7; S8	7201,43	3020,09%

Tabela 3: Teste 8 com sobrecarga não detectada

Em uma experimentação alternativa, utilizando as mesmas configurações para o método byrequests, os tempos de atendimento foram bem piores do que quando usando o bytraffic. Uma análise do arquivo de log mostrou que neste método as requisições não eram balanceadas, portanto todos os nós recebiam o mesmo número de requisições. Já no método bytraffic, as requisições e os downloads são distribuídos entre todos os nós que participam do balanceamento, muito embora os tempos sejam bem maiores do que em situações normais vista na seção anterior.

As requisições deveriam ser direcionadas aos escravos de acordo com o volume de tráfego, em bytes, de cada escravo. No entanto, o tráfego a mais decorrente dos downloads nos escravos não foi computado no balanceamento, pois no log de acesso de cada um dos 4 escravos constava 1/4 do total de requisições, ficando o tempo de atendimento das requisições condicionado ao tempo do nó mais lento. Os tempos obtidos nestes testes ocorreram pelo fato

dos nós com sobrecarga continuarem recebendo o mesmo número de requisições dos nós sem sobrecarga.

Desta maneira, pode-se concluir que o Apache com o mod_proxy_balancer executa um balanceamento efetivo de requisições quando os direcionamentos se encontram declarados em suas diretivas, não percebendo o tráfego de dados que até "passa" pelo Apache, mas não é controlado pelo mod_proxy_balancer. Isto ocorre porque a contabilização de tráfego, em bytes, é feita somente dentro do módulo e direcionamentos feitos por outros módulos, como o mod_proxy, não são considerados para o balanceamento. Os tempos obtidos são piores do que os testes anteriores pelo fato dos downloads não serem balanceados. É essa situação que queremos amenizar e assim propomos o método byquery mostrado na próxima seção.

Desempenho com tráfego de rede não detectado usando o novo método byquery

O método consiste da verificação de quanto um nó está sem processamento de CPU (ocioso), para tomada de decisão de balanceamento. Os nós mais ociosos, ou seja, com menos carga de CPU, atenderão mais requisições. A informação sobre quanto um nó está ocioso é fornecida por um daemon em conjunto com o programa sysidle. O sysidle foi desenvolvido a partir da ferramenta de monitoramento de desempenho chamada sysstat [13]. Um de seus módulos, o mpstat, fornece informações de utilização de um processador específico ou de todos da máquina em tempo real.

O daemon é um programa que aceita conexões TCP na porta 3054, executa o programa sysidle e retorna a informação de ociosidade do sistema para o cliente que efetuou a conexão. Depois de testados, o daemon e o sysidle foram instalados nos nós escravos. O próximo passo foi implementar o método byquery no mod_proxy_balancer. O novo método utiliza o mesmo conceito do método byrequests, no entanto, em vez de contabilizar as requisições foi utilizada a porcentagem de ociosidade de cada nó.

A cada intervalo de tempo, o nó mestre se conecta aos nós escravos solicitando a informação de ociosidade. Este valor é multiplicado pelo fator de balanceamento para a quota de trabalho de cada nó naquele determinado momento. Essa multiplicação se faz necessária para manter a proporcionalidade na distribuição de requisições especificada pelo fator de balanceamento. Como já visto o tempo de atendimento total a um conjunto de requisições fica condicionado ao tempo de atendimento do nó mais lento.

Visando evitar o envio de requisições para nós com sobrecarga, a partir de certa porcentagem de utilização de CPU o nó é considerado indisponível. No entanto, é necessário definir qual deve ser a melhor porcentagem. Além disto, também é necessário definir o intervalo de tempo para realização das consultas aos nós do cluster. Um intervalo muito grande pode não conseguir identificar situações de sobrecarga de CPU e intervalos muito pequenos podem gerar overhead.

Para definir estes valores, analisamos experimentalmente a influência do intervalo de consulta aos nós escravos sobre o tempo total de atendimento. Foi utilizada a configuração do teste 8, no qual os direcionamentos são para os escravos 5 a 8, sem sobrecarga, com 100.000 requisições solicitadas. Os intervalos de consulta testas foram de 5, 10, 15, 20, 30 e 60 segundos. Os tempos obtidos com a ferramenta httperf são mostrados na Figura 7.

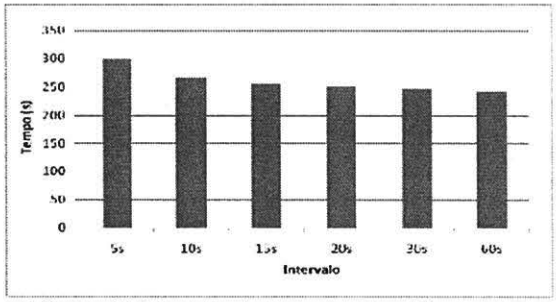

Figura 7. Tempo de resposta utilizando o método byquery em intervalos de tempo fixo.

Observa-se que intervalos muito pequenos (5 s) geram maior atraso no atendimento às requisições, sendo descartados dos próximos testes. E apesar do intervalo de tempo de 60s possuir o melhor resultado, este também foi descartado por ser considerado muito grande para as proporções dos experimentos realizados neste trabalho. Assim, os intervalos de tempo 10s, 15s, 20s e 30s foram combinados com 75%, 80%, 85%, 90% e 95% de ociosidade de CPU. Conforme pode se observar na Figura 8 o melhor intervalo de tempo para consulta aos escravos foi de 15 segundos e as melhores porcentagens foram de 90% e 95%, que representa 10% e 5% de utilização de CPU. Sendo 10% de utilização de CPU um valor que melhor represente um cenário de sobrecarga, decidiu-se que com menos de 90% de ociosidade o nó é considerado com sobrecarga.

Desta forma, o método byquery foi testado nos experimentos apresentados na seção anterior, já com os valores ajustados do intervalo de tempo e do limite de sobrecarga. Os resultados obtidos também são comparados com os dos métodos byrequests e bytraffic, conforme mostra a Figura 9.

Nesta figura observamos o melhor desempenho do método byquery em todo o conjunto de testes 4.2 e 8.2. Os tempos obtidos são em média 30% menores, para o

melhor caso, e os tempos são similares no pior caso. A descoberta de nós com sobrecarga de CPU foi benéfica ao mod_proxy_balancer como pode ser constatado através da análise do log de acesso do Apache (escravos) nos testes 4.1 a 4.4.

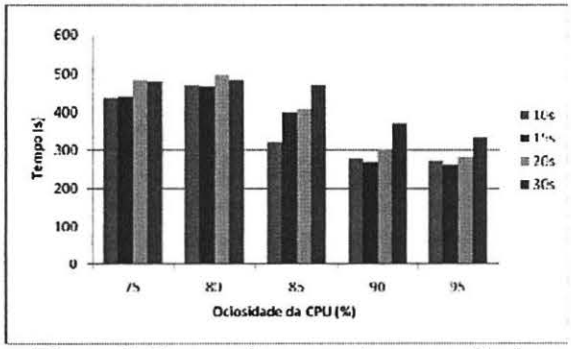

Figura 8. Tempo de resposta com método byquery. Intervalo x ociosidade da CPU.

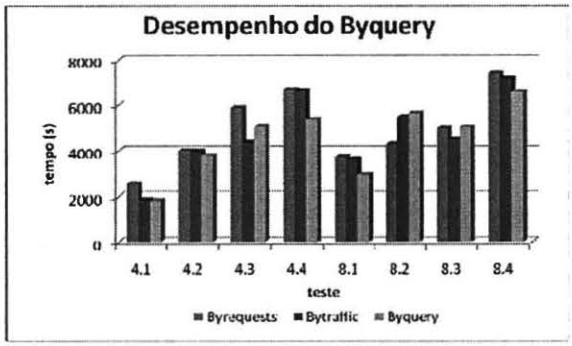

Figura 9. Tempos pelos métodos byrequests, bytraffic e byquery com sobrecarga.

4. Conclusões e trabalhos futuros

O presente trabalho concentra-se na avaliação de *cluster* de servidores *web* não dedicados que utilizam o Apache com *mod_proxy_balancer* como redirecionador de requisições. Os testes mostram que o redirecionamento representa um acréscimo de tempo no atendimento às requisições, comprovando que este modelo de cluster é voltado para alta disponibilidade e não para alto desempenho. Além disso, quando cargas trafegam na rede sem o conhecimento do *mod_proxy_balancer*, o desempenho é fortemente prejudicado, podendo elevar o tempo de atendimento em mais de 3000% nos casos aqui examinados.

A detecção da carga externa ao *mod_proxy_balancer* e a aplicação do método byquery, pode gerar ganhos superiores a 30% na redução do tempo de atendimento as requisições web. Como trabalhos futuros, maiores esforços devem ser feitos no sentido de avaliar a sobrecarga de rede usando aplicações distribuídas reais, pois o uso de downloads parece estressar o barramento de uma forma muito contínua e pode não expressar a dinâmica das aplicações distribuídas.

5. Agradecimentos

Os autores agradecem o suporte financeiro concedido pela CAPES.

6. Referências bibliográficas

[1] Cardellini, V. et al. "The State of the Art in Locally Distributed *Web*-Server Systems", ACM Computing Surveys, 2002.

[2] Aversa, L. and Bestavros, A. "Load balancing a *cluster* of *Web* servers using Distributed Packet Rewriting". In Proceedings of the 19th IEEE International Performance, Computing, and Communication Conference. IEEE Computer, 2000

[3] Cardellini, V., Colajanni, M., AND YU, P. S. "Dynamic load balancing on *Web*-server systems". IEEE Internet Computing, 3, p. 28–39, 1999.

[4] Iyengar, A. et al, P. "High performance *Web* site design techniques" IBM Thomas J. Watson Res. Center, Yorktown Heights, NY, Mar/Apr 2000.

[5] Marwah, M., Mishra, S. and Fetzer, C. "Fault-Tolerant and Scalable TCP Splice and *Web* Server Architecture". Technical Report. Department of Computer Science, University of Colorado, Boulder,

[6] Schroeder, T., Goddard, S. and Ramamurthy, B. "Scalable *Web* Server *Cluster*ing Technologies". IEEE Network, p. 38-45, may 2000.

[7] Linux Virtual Server Project "Linux Virtual Server Documentation" Disponível em: http://www.linuxvirtualserver.org. Acessado em 02 de maio de 2006. CO. Number: CU-CS-1003-06, jan 2006.

[8] Netcraft. "*Web* Server Survey" Disponível em http://news.netcraft.com em 20 de março 2006.

[9] Mockus, A., R. Fielding, and J. Herbsleb. "A Case Study of Open Source Software Development: The Apache Server" In Proceedings of International Conference on Software Engineering, 2000.

[10] ApacheBench. "AB-Apache HTTP server benchmarking tool". http://httpd.apache.org/docs/2.2/programs/ab.html, 2006.

[11] Poskanzer, Jef. "Http load - multiprocessing http test client", http://www.acme.com/software/http_load/.

[12] Mosberger, D. and Jin, T. "httperf: A tool for measuring *Web* server performance", Workshop on Internet Server Performance. ACM, p. 59-67, 1998.

[13] Sysstat. "Sysstat". Disponível em http://perso.orange.fr/sebastien.godard/, Acessado em 01 de agosto de 2007.