DOI: 10.4025/actascitechnol.v33i4.3882

Análise do balanceamento de requisições em clusters *Web* não-dedicados

Jô Sato[1*], João Angelo Martini[2] e Ronaldo Augusto Lara Gonçalves[2]

*[1]Prefeitura do Município de Londrina, Av. Duque de Caxias, 635, 86015-901, Londrina, Paraná, Brasil. [2]Programa de Pós-graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, Paraná, Brasil. *Autor para correspondência. E-mail: josato@gmail.com*

RESUMO. A quantidade de serviços e informações disponibilizados na Internet, tais como movimentações bancárias, transações comerciais, redes sociais, blogs e até jogos interativos tem aumentado continuamente. Os provedores de web precisam estar preparados para suportar esta demanda crescente de informação e comunicação. Neste sentido, o uso de cluster de servidores web em conjunto com o balanceamento de requisições tem sido uma estratégia importante pelo fato de distribuir os serviços entre vários processadores de forma balanceada. 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 denominado *byalltraffic* 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.

Palavras-chave: serviço *web* distribuído, apache.

ABSTRACT. Analysis of request balancing in non-dedicated web clusters. The amount of services and information available on the Internet, such as banking and business transactions, social networks, blogs and even interactive games, has steadily increased. Web providers must be prepared to support this growing demand for information and communication. In this sense, the use of web server clusters together with the balancing of requests has been an important strategy for distributing the services among several processors in a balanced way. This paper describes and discusses architectural and operational issues of request balancing, especially dynamic balancing, covering concepts, techniques and solutions. Real experiments were performed in different configurations on a cluster of Apache servers non dedicated exclusively to web service. The *mod_proxy_balancer* load balancing module was used with intensive synthetic overloads and a new balancing method called *byalltraffic* was proposed and evaluated. The results show that the balancing in this type of system will be more efficient if the external network workload to the web service is detected and used in the balancing factor.

Keywords: distributed web service, apache.

Introdução

A popularização da Internet e o consequente 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 et al. (2002), 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 (estudos de Aversa e Bestavros apud TEO; AYANI, 2001; CARDELLINI et al., 1999, 2002; IYENGAR et al., 2000; SCHROEDER et al., 2000).

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, no 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 descreve a organização de um cluster *web*, apresenta o servidor *web*, discorre sobre alguns mecanismos de 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 denominado *byalltraffic* foi proposto e avaliado. Os resultados de experimentos, apresentados na seção Resultados e discussão, mostram a sua eficiência.

Material e métodos

Este trabalho caracterizou-se pelo estudo do funcionamento de um sistema *web*, desde os seus componentes até os mecanismos de redirecionamentos. O primeiro componente, o 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 et al. (1999, 2002), 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 onde cada nó possui um endereço IP publicamente visível aos clientes, conforme a Figura 1. 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.

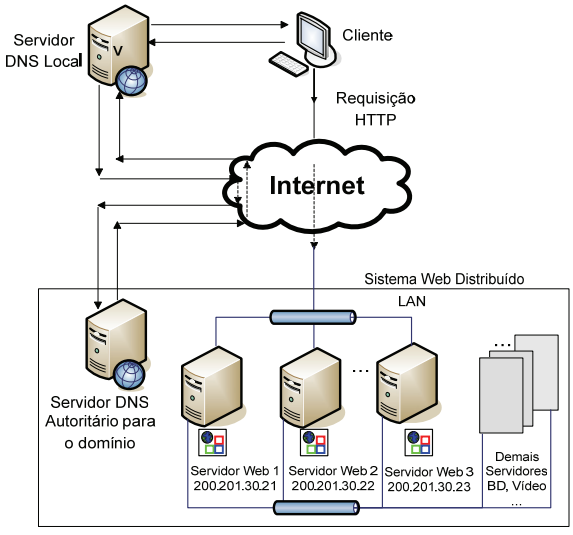

Figura 1. Sistema *web* distribuído.

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. Existem algoritmos baseados no estado

dos servidores *web* e no estado dos clientes, sendo o mais famoso e utilizado, por ser de fácil implementação, o DNS Round Robin, no qual não se utiliza qualquer informação do sistema *web* distribuído e nenhum balanceamento é realizado.

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 a Figura 2.

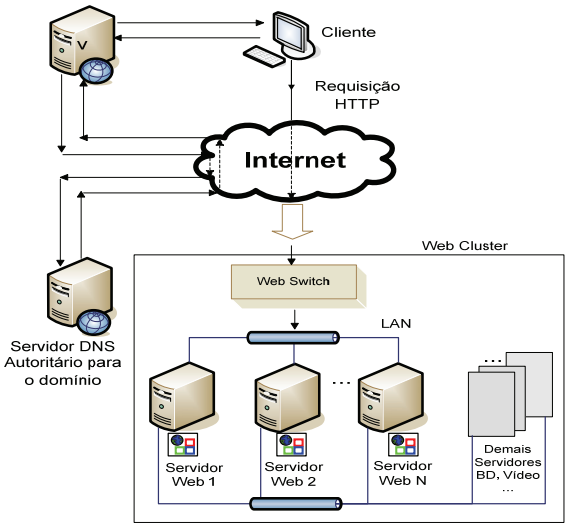

Figura 2. Arquitetura de um cluster *web*.

Somente o seu endereço IP é visível publicamente aos clientes. Os demais nós têm seus endereços "mascarados" pelo redirecionador. Cada nó servidor do cluster, normalmente, possui seu próprio disco e sistema operacional. O endereçamento das requisições dos clientes pode ser controlado em *hardware* ou em *software*. Na Figura 2, o servidor DNS autoritário para o domínio não faz parte do cluster, pois não possui função no sistema de roteamento de requisições dos clientes.

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.

O redirecionamento de requisições *web* é classificado de acordo com as diferentes camadas do modelo OSI em que ocorrem, sendo os principais:

- transporte/enlace (camadas 4/2): o nó escravo é identificado pelo seu endereço físico (MAC). Sua implementação é mais complexa;

- transporte/rede (camadas 4/3): cada nó é o nó escravo é identificado pelo seu endereço IP, exigindo a reescrita do pacote;

- aplicação (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 nesta camada.

As técnicas mais utilizadas para distribuição de requisições, de acordo com Andreolini, conforme citado por Zhang et al. (2005), são:

- estáticas (*State-blind*): não verifica o estado do servidor que irá receber a requisição;

- baseadas em informações do servidor (*server state-aware*): são consideradas informações como utilização de cpu, memória, I/O de rede;

- baseadas em informações do cliente (*client state-aware*): de acordo com a informação solicitada são definidos servidores especializados para o atendimento das requisições, tais como servidores de multimídia, conteúdo dinâmico;

- baseadas em informações do cliente e servidor (*client* e *server state-aware*).

Algumas técnicas para balanceamento de requisições *web* são apresentadas, a seguir, e constituem a fundamentação para o desenvolvimento das implementações:

- políticas estáticas e adaptativas para transferência de carga de trabalho entre os vários nós de um sistema distribuído são descritas por Eager et al. (1985). A política estática utiliza-se somente da informação do comportamento padrão do sistema. As decisões de transferência de carga são independentes do estado atual do sistema;

- o Distributed Cooperative Apache Web Server (DC-Apache) é uma proposta de Li e Moon descrita em Cardellini et al. (2003) para encontrar escalabilidade de desempenho para servidores Web. Esta solução se baseia na hipótese que a maioria de sites Web tem somente alguns pontos de entrada bem conhecidos que os usuários utilizam para iniciar a navegação. O sistema de DC-Apache manipula dinamicamente os hyperlinks dos documentos *HTML*, a fim de distribuir pedidos do acesso de clientes entre os múltiplos servidores Web cooperados;

- o projeto Mod_Backhand, desenvolvido por Theo Schlossnagle descrito por Cao et al. (2003), é um módulo para o Apache versão 1.x, desenvolvido na Universidade Johns Hopkins, em Baltimore/Maryland/Estados Unidos, destinado a efetuar a distribuição da carga entre vários servidores HTTP, utilizando informações referentes a recursos computacionais utilizados;

- o algoritmo de balanceamento, denominado de MOLL (Migration-Optimized Least Loaded), é proposto por Nilson e outros, publicado no evento "Networks and Communication Systems", realizado na Tailândia em 2005. MOLL combina a utilização de duas variáveis muito importantes na tomada de decisão para o redirecionamento: a carga do servidor e o tempo de atraso durante a migração de sessão.

Os estudos e implementações utilizadas neste trabalho utilizaram o módulo *mod_proxy_balancer*, derivado do *mod_proxy*, do servidor *web* Apache. O Apache é o servidor *web* mais utilizado no mundo. 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). Ele é configurável, extremamente robusto e objetivo alto de desempenho e de disponibilidade. Suas funcionalidades são mantidas por meio de uma estrutura em módulos, permitindo inclusive que o usuário escreva seus próprios módulos utilizando a API do *software* (MOCKUS et al., 2002).

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 dois métodos de balanceamento: por quantidade de requisições (*byrequests - request counting*) e por quantidade de sobrecarga de tráfego (*bytraffic - weighted traffic counting*). A Figura 3 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. O método de balanceamento é definido 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. A Figura 4 ilustra um exemplo de configuração de balanceamento para quatro servidores *web* com o método *byrequest* e o fator de balanceamento igual a 1 que foram utilizados nos testes descritos na próxima seção.

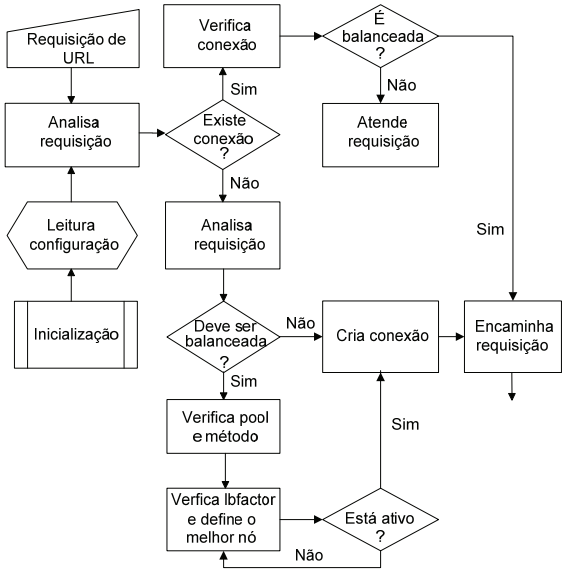

Figura 3. Tratamento da requisição no *mod_proxy_balancer*.

```
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 4. Configuração no httpd.conf com o *mod_proxy_balancer*.

Resultados e discussão

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*, *byrequest*s 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, Http_load e o *Httperf* (ANDREOLINI et al., 2002). 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). Para utilização do *Httperf* é necessário especificação de alguns parâmetros, como endereço do servidor *web*, a página a ser acessada e o número conexões a serem efetuadas. A partir destas informações são geradas as conexões na qual cada uma executa uma requisição.

Os testes foram realizados em um ambiente experimental com o Apache versão 2.2.3, instalado em todas as máquinas de um cluster composto de um nó mestre e oito nós escravos (escravo1 a escravo8), interligados por meio de um switch fast ethernet. Com relação aos equipamentos, os nós escravos de 1 a 4 são processadores Pentium IV 2,5 GHz com 256 MB de RAM e os escravos de 5 a 8, Pentium IV Hyperthreading de 3.0 GHz com 512 MB de RAM. A Figura 5 apresenta a sua organização.

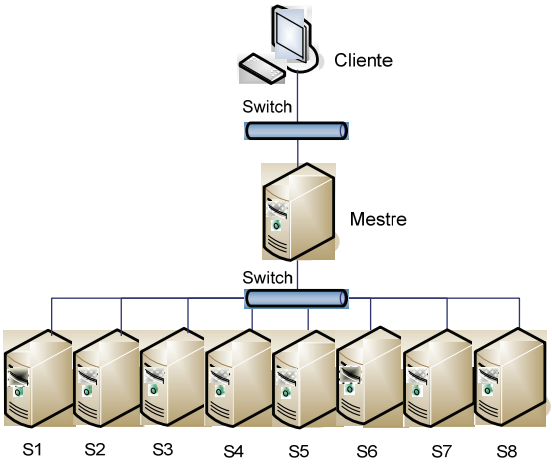

Figura 5. Organização do cluster utilizado.

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.

Três experimentos foram realizados para avaliação do *mod_proxy_balancer*: dedicado ao serviço *web*, não dedicado ao serviço *web* e com isolamento de nodos sobrecarregados durante o compartilhamento do cluster, conforme explicado nas próximas seções. A partir da análise destes experimentos, um novo método de balanceamento é proposto para o *mod_proxy_balancer* para ser usado em situações de compartilhamento do cluster.

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. Os métodos *bytraffic* e *byrequest* foram experimentados nas mesmas condições e os tempos obtidos também não apresentaram diferenças significativas. Por isso, os tempos médios são mostrados como resultados.

Os primeiros testes visam avaliar o desempenho do *mod_proxy_balancer* em condições comuns de uso do cluster, quando este é usado exclusivamente para atendimento a requisições *web*. As requisições recebidas pelo mestre eram redirecionadas a 1, 2, 3, 4, 6, 7 e 8 nós escravos, gradativamente em experimentos consecutivos.

A Figura 6 apresenta o tempo médio de atendimento, em que o eixo x representa o número de nós utilizados no redirecionamento, e x igual a 0 a situação em que o mestre atendeu sozinho todas as requisições. Pode-se observar que o uso do balanceamento (redirecionamento das requisições para os escravos) aumenta o tempo de resposta. Este fato mostra que o uso do cluster *web* em questão provê maior disponibilidade, mas não maior desempenho. Isto ocorre porque todas as requisições são centralizadas no mestre que aguarda a finalização dos serviços executados pelos escravos para então repassar aos clientes. Os escravos não repassam diretamente aos clientes. As páginas recuperadas pelos escravos são devolvidas ao mestre para que este as repasse aos clientes.

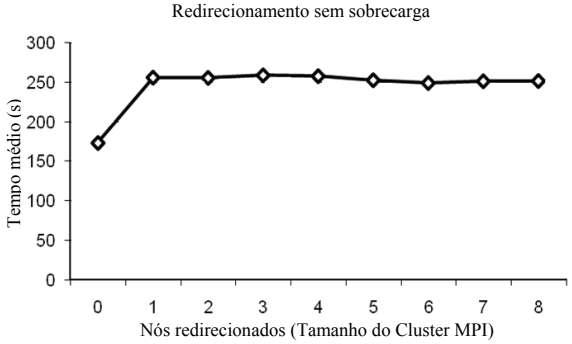

Figura 6. Redirecionamento sem sobrecarga de tráfego.

Assim, como se pode observar na Figura 6, não importa se o balanceamento possui 1, 8 ou qualquer número de máquinas envolvidas, pois o tempo é determinado pelo "afunilamento" no mestre que trabalha sobre todas as requisições. O benefício causado pela distribuição e paralelização do acesso ao disco (pela recuperação simultânea das páginas pelos nós escravos) não é maior do que o prejuízo do tráfego de rede (dos escravos para o mestre) e da serialização do tratamento das páginas no mestre. Este fato comprova que o cluster *web* é voltado à disponibilidade e não ao desempenho.

Nos experimentos anteriores, o cluster foi utilizado de forma dedicada ao serviço *web*. Entretanto, o presente trabalho visualiza outros cenários onde o cluster pode ser compartilhado com outros tipos de aplicações, as quais podem saturar um ou mais nós do cluster de forma não detectada pelo *mod_proxy_balancer*. Esta carga de processamento e comunicação não relacionada ao serviço *web* não é considerada no fator de balanceamento e por isso o mesmo é prejudicado.

Em instituições de pesquisa e desenvolvimento, principalmente aquelas com restrições orçamentárias, usar os recursos do cluster para a realização de testes e execuções com aplicações científicas, é uma solução economicamente atrativa, desde que não prejudique os serviços *web*.

Para analisar o *mod_proxy_balancer* em situações de compartilhamento do cluster, experimentos de sobrecarga sintética de rede (forçada) foram realizados para simular o compartilhamento do cluster por aplicações que não são identificadas pelo mestre, mas que demandam alto processamento e comunicação. Aplicações científicas, normalmente, apresentam estas características, mas como o objetivo aqui não é investigar este tipo de aplicações, pelo menos neste momento da pesquisa, o *wget*, um programa para efetuar *download* de arquivos, foi utilizado para simular uso intensivo de processamento e comunicação, simultaneamente com o serviço *web*.

Os *downloads* foram realizados entre os próprios nós escravos, causando sobrecarga sintética entre pares de nós (nó solicitante e nó atendente). O número de pares envolvidos foi variado até a saturação completa do cluster. Em todos os testes, o fator de balanceamento foi o mesmo para todos os nós. Os tempos obtidos são apresentados na Figura 7.

Observa-se que todos os tempos obtidos nos atendimentos com sobrecarga foram superiores aos apresentados na Figura 6. Obviamente, o valor obtido quando nenhum nó era sobrecarregado é o mesmo daquele obtido somente com o redirecionamento para oito escravos, apresentado na Figura 6. Quanto maior o número de nós com sobrecarga, maior o tempo necessário para o atendimento às requisições.

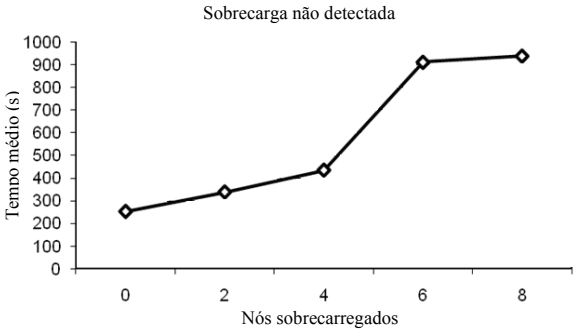

Figura 7. Redirecionamento com sobrecarga de tráfego externo.

Os valores exatos obtidos com zero, dois, quatro, seis e oito nós sobrecarregados correspondem a 252, 338, 435, 912 e 939 s, respectivamente. Isto corresponde a acréscimos de aproximadamente 0, 34, 72, 262 e 272 pontos percentuais, no tempo de execução. Este fato ocorre porque as solicitações de *downloads* não são detectadas pelo mestre, pelo fato de serem solicitações externas aos serviços *web* (de aplicações concorrentes a este serviço). O *mod_proxy_balancer* não está preparado para este tipo de situação de compartilhamento de processamento.

Antes de investir tempo no desenvolvimento de um novo método de balanceamento que possa considerar a presença de carga externa no fator de balanceamento, realizamos novos testes isolando manualmente os nodos sobrecarregados dos repasses das requisições. O objetivo foi o de analisar o impacto no desempenho de um possível método de balanceamento que assim o fizesse. A Figura 8 ilustra estes novos resultados.

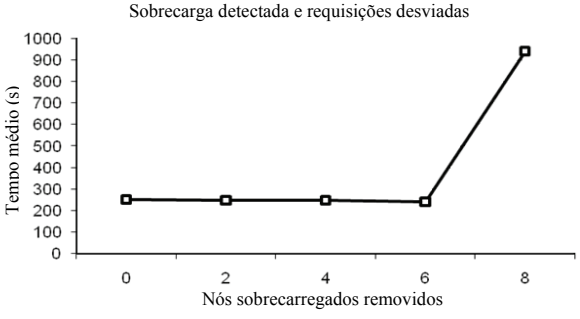

Figura 8. Redirecionamento com sobrecarga de tráfego detectado.

O ganho no desempenho é visivelmente perceptivo. Uma análise rápida permite perceber que a própria característica de disponibilidade do cluster é a responsável pelo ganho, pois, como o

desempenho do cluster independe do número de nós utilizados no redirecionamento (Figura 6), a remoção dos nós sobrecarregados causa ganho significativo na redução do tempo de execução.

O maior benefício ocorre nas situações, onde seis nós estão sobrecarregados artificialmente, pois o tempo reduz de 912 para 242 s, representando um ganho de 73%. No caso em que todos os escravos estão sobrecarregados, não existe a possibilidade de isolamento, pois não existe escravo disponível para assumir o atendimento das requisições, e o tempo de atendimento permanece elevado da mesma forma quando a carga não é detectada.

Convém lembrar que o isolamento aplicado aos nós sobrecarregados foi feito de forma monitorada e manual, antes dos experimentos que avaliam seu efeito, sendo, portanto uma solução teórica. Em um sistema mais realista, esta sobrecarga deveria ser detectada pelo próprio sistema e com isso, a decisão de isolamento sofreria certo atraso no tempo e o ganho seria diferente e dependente do tamanho deste atraso. Na próxima seção, apresentamos uma solução real de balanceamento, para situações específicas, na qual os nós com muito tráfego recebem menos ou não recebem requisições.

A partir da análise dos resultados dos testes, verificou-se que o *mod_proxy_balancer* pode ser melhorado se considerar o tráfego de rede gerado por direcionamentos do *mod_proxy*. No contexto do presente trabalho, os *downloads* que causam as sobrecargas no sistema são realizados via mestre e são assim detectados pelo *mod_proxy*, muito embora não sejam detectadas pelo *mod_proxy_balancer*. Entretanto, uma vez que o módulo *mod_proxy_balancer* é uma especialização do módulo *mod_proxy*, as informações deste podem se tornar acessíveis pelo módulo mais especializado com pequeno ajuste no código. Criou-se então o método *byalltraffic* por meio de uma modificação do método *bytraffic*.

Neste novo método, durante o processo de seleção de um nó para o encaminhamento de requisições, o tráfego do *host* contabilizado pelo *mod_proxy_balancer* é somado ao tráfego de todos os demais direcionamentos do *mod_proxy* que tenham este mesmo *host* como destino. Após esta contabilização, o nó com o menor volume de tráfego (definido pela variável curmin) é o escolhido. O algoritmo simplificado para este método é apresentado na Figura 9.

```
curmin = 0
  para cada servidor que participa do balanceamento
    transf = servidor bytes_transferidos;
    read = servidor bytes recebidos;
    worker = primeiro servidor do conjunto de servidores (mod_proxy)
    para (j=0;j<qtde de servidores;j++)
      se (worker.hostname=servidor.hostname)
        transf = transf + worker1 bytes_transferidos;
        read = read +worker1 bytes recebidos;
        worker1++;
        trafego = (transf/servidor lbfactor)+(read/servidor lbfactor);
        se (nenhum candidato ou trafego < curmin)
          candidato = servidor
          curmin = trafego
```

Figura 9. Algoritmo simplificado do método *byalltraffic*.

A Figura 10 ilustra os resultados obtidos nos experimentos onde o método *bytraffic* foi comparado com o *byalltraffic*. Duas configurações principais de clusters foram utilizadas: com quatro e com oito servidores *web*, indicadas pelos pontos 4.x e 8.x no eixo horizontal da mesma figura, onde x representa o número de nodos sobrecarregados com os *downloads* e atinge o máximo de 4.

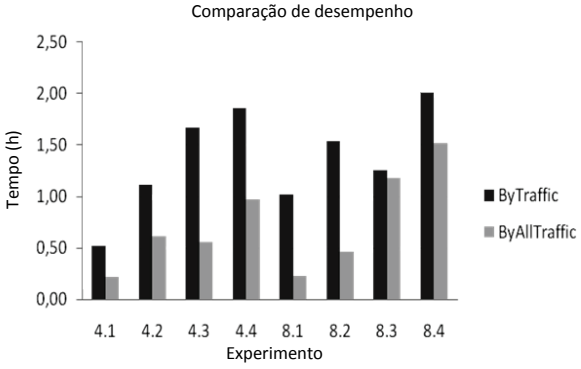

Figura 10. Comparação tempos método *bytraffic* e *byalltraffic*.

Por exemplo, na configuração 4.2, um cluster com quatro servidores *web* sofreu uma sobrecarga com *downloads* não-detectados pelo *mod_proxy_balancer* em dois nodos.

Os *downloads* foram forçados de slave a slave passando pelo nó mestre, e os *escravos* de 5 a 8 e de 1 a 4 executaram *downloads* de arquivos nos *escravos* 1 a 4 e 5 a 8, respectivamente. Para cada teste, os downloads eram iniciados antes das requisições *web*, continuando também durante a execução da ferramenta *httperf* na máquina cliente.

Pode-se observar que a redução no tempo de atendimento às requisições é substancialmente menor usando o *byalltraffic*, atingindo valores acima de 69% no experimento 8.2, o qual é o melhor resultado. O pior resultado foi obtido no experimento 8.3 (menos de 6%). Na média, o ganho está acima de 49% de redução no tempo de execução.

Conclusão

A avaliação do cluster de servidores web não-dedicados mostrou 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. Quando cargas são processadas e trafegam na rede sem o conhecimento do redirecionador (mestre), o desempenho é fortemente prejudicado, podendo elevar o tempo de atendimento em mais de 270%.

O isolamento dos nós sobrecarregados pode causar ganhos teóricos superiores a 70% na redução do tempo de execução. Usando um método simples de detecção de tráfego de downloads que passa pelo nó mestre, o ganho pode atingir a escala de até 69% no melhor caso. Entretanto, a necessidade do isolamento total dos nós sobrecarregados deve ser avaliada, bem como o impacto no ganho causado por atrasos na detecção da sobrecarga.

Agradecimentos

Agradecemos a Capes, pelo suporte financeiro concedido ao projeto "Laboratório Multidisciplinar de Computação de Alto Desempenho, Serviços *Web* e Banco de Dados" no contexto do Edital 01/2007 (Pró-Equipamentos).

Referências

ANDREOLINI, M.; CARDELLINI, V.; COLAJANNI, M. Benchmarking models and tools for distributed web-server systems. In: CALZAROSSA, M. C.; TUCCI, S. (Ed.). **Performance evaluation of complex systems: techniques and tools**. London: Springer-Verlag, 2002. p. 208-235.

CAO, J. N.; SUN, Y. D.; WANG, X. B.; DAS, S. K. Scalable load balancing on distributed web servers using mobile agents. **Journal of Parallel and Distributed Computing**, v. 63, n. 10, p. 996-1005, 2003.

CARDELLINI, V.; CASALICCHIO, E.; COLAJANNI, M.; YU, P. S. The state of the art in locally distributed web-server systems. **ACM Computing Surveys**, v. 34, n. 2, p. 263-311, 2002.

CARDELLINI, V.; COLAJANNI, M.; YU, P. S. Dynamic load balancing on web-server systems. **IEEE Internet Computing**, v. 3, n. 3, p. 28-39, 1999.

CARDELLINI, V.; COLAJANNI, M.; YU, P. S. Request redirection algorithms for distributed web systems. **IEEE Transactions on Parallel and Distributed Systems**, v. 14, n. 4, p. 355-368, 2003.

EAGER, D. L.; LAZOWSKA, E. D.; ZAHORJAN, J. A comparison of receiver-initiated and sender-initiated adaptive load sharing. **ACM Sigmetrics Performance Evaluation Review**, v. 13, n. 2, p. 1-3, 1985.

IYENGAR, A.; CHALLENGER, J.; DIAS, D.; DANTZIG, P. High-performance web site design techniques. **IEEE Internet Computing**, v. 4, n. 2, p. 17-26, 2000.

MOCKUS, A.; FIELDING, R. T.; HERBSLEB, J. D. Two case studies of open source software development: Apache and Mozilla. **ACM Transactions on Software Engineering and Methodology**, v. 11, n. 3, p. 309-346, 2002.

SCHROEDER, T.; GODDARD, S.; RAMAMURTHY, B. Scalable web server clustering technologies. **IEEE Network**, v. 14, n. 3, p. 38-45, 2000.

TEO, Y. M.; AYANI, R. Comparison of load balancing strategies on cluster-based web servers. **Simulation**, v. 77, n. 5-6, p. 185-195, 2001.

ZHANG, Q.; RISKA, A.; SUN, W.; SMIRNI, E.; CIARDO, G. Workload-aware load balancing for clustered web servers. **IEEE Transactions on Parallel and Distributed Systems**, v. 16, n. 3, p. 219-233, 2005.

Received on June 12, 2008.
Accepted on March 15, 2010.